Online digital archive of aerial photographs (1935–1941) of Ethiopia

Jan Nyssen1 | Martijn Debever1 | Gezahegne Gebremeskel2 | Bart De Wit1 | Kiros Meles Hadgu3,4 | Steven De Vriese1 | Jeffrey Verbeurgt1 | Amaury Frankl1,5 | Tulu Besha2 | Jan Kropáček6 | Astrid Forceville1 | Biadgilgn Demissie3,7

1Department of Geography, Ghent University, Gent, Belgium
2Ethiopian Geospatial Information Institute, Addis Ababa, Ethiopia
3Institute of Geoinformation and Earth Observation Sciences, Mekelle University, Mekelle, Ethiopia
4World Agroforestry (ICRAF) - Ethiopia, Addis Ababa, Ethiopia
5INRAE, AMAP, IRD, CIRAD, CNRS, University Montpellier, Montpellier, France
6Department of Physical Geography and Geocology, Charles University, Prague, Czech Republic
7Department of Geography and Environmental Studies, Mekelle University, Mekelle, Ethiopia

Correspondence
Jan Nyssen, Department of Geography, Ghent University, Krijgslaan 281 (S8), B-9000 Gent, Belgium.
Email: jan.nyssen@ugent.be

Abstract
The archive of aerial photographs, dating 1935–1941 and covering parts of north and central of Ethiopia, is one of the few archives with pre-1960 remotely sensed data in Africa. It allows adding 30 years of time depth for geographical studies, in contrast to the commonly known oldest imagery dating to 1964, sometimes 1958. These photographs were originally acquired by the Italian military forces, to obtain raw material for warfare purposes and the production of topographic maps. To make the archive accessible, it has been scanned and georeferenced. Procedures used in georeferencing and digitally archiving are detailed in this paper, as well as our experiences with orthorectification and use of the photosets in scientific research. This data set of aerial photographs of Ethiopia in the 1930s has been availed to the wider scientific community through the Pangaea website http://doi.org/10.1594/PANGAEA.920077 (Nyssen et al., 2020, Aerial photographs of Ethiopia 1935–1941). Additionally, a web interface (http://www.ethiopia1935.ugent.be/) allows scientists visualizing the location of relocated photographs, and to select and freely order scenes of interest.

KEYWORDS
Ethiopia, historical aerial photographs, Italy, orthorectification, remote sensing

Dataset details
Dataset Identifier: https://doi.org/10.1594/PANGAEA.920077; http://www.ethiopia1935.ugent.be/
Creator: Jan Nyssen, Martijn Debever, Gezahegne Gebremeskel, Bart De Wit, Kiros Meles Hadgu, Steven De Vriese, Jeffrey Verbeurgt, Sultan Mohammed, Amaury Frankl, Tulu Besha, Jan Kropáček, Astrid Forceville, Biadgilgn Demissie
Dataset correspondence: jan.nyssen@ugent.be
Title: Aerial photographs of Ethiopia 1935-1941
Publisher: Pangaea; UGent
Publication year: 2020
Resource type: Data set
Version: 20210310

This is an open access article under the terms of the Creative Commons Attribution License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited.
© 2021 The Authors. Geoscience Data Journal published by Royal Meteorological Society and John Wiley & Sons Ltd.
There are few publications concerning digital archives of historical aerial photographs, mostly regarding the USA, particularly California in 1938 (Ma, 2013), Illinois in the 1930s (Luman et al., 1997), the Everglades in the 1920s (Smith et al., 2010) or the overall USA (McAuliffe et al., 2017), and rarely from Europe such as Wales (Wilson et al., 2016). Very few stress the need for user-friendly digital archives or online interface (Caswell, 2012; Long et al., 2005; Mathews, 2005). In addition to the earlier listed historical imagery (Nyssen et al., 2016), millions of analogue historical aerial photographs covering parts of Africa (sometimes dating back to the 1920s) are dispersed in archives, among others in Zimbabwe (Whitlow, 1988), France (Michel, 2018), U.K., Uganda, Tanzania and Kenya (Kuns, 2010), and Zambia (Pullan, 1976). Yet, few digital archives of aerial photographs are available for the African continent (Table 1).

For Ethiopia, an archive of aerial photographs (APs), dating 1935–1941 and covering large parts of the northern and central parts of the country, is available. It allows adding 30 years of time depth for geographical studies, in contrast to the commonly known oldest APs dating mainly to 1964 and 1986, sometimes to 1958. The APs of the 1930s were originally acquired by the Italian military forces, mostly by the 7th Topocartographic Section of the Istituto Geografico Militare (IGM), to obtain raw material for warfare purposes and the production of topographic maps during Italian occupation. The archive consists of approximately of 34 000 APs, taken along 94 different flight lines. These APs are of great value for environmental and historical studies about the area they are covering, such as changes in land cover, hydrology or geomorphology. In the next step, these photographs have been digitally availed to Ethiopian universities, research institutes and to the broader scientific community.

In this paper, we discuss the data set and its processing as well as procedures followed to semi-automatically avail it to end users. State-of-the-art information is also presented on ways to process and analyse subsets of the AP collection.

2 | METHODS: REALIZATION OF THE AERIAL PHOTOGRAPHS

2.1 | Equipment

For the purpose of mapping in Libya and East Africa, where a wide angular coverage of large areas of terrain in a single set of exposures was required for small-scale mapping, a four-coupled version of Santoni’s Model II camera (Gentilli, 1992; Traversi, 1964) was developed. The original Model II twin camera unit was equipped with two lenses, with each lens having a focal length of 175 mm. The twin cameras used photographic glass plates to record the resulting images in two separate focal planes, with each negative image being 10 × 15 cm in size. Each of the two rotating cylindrical drum magazines held 200 of such glass plates (IGM, 1939). The four-coupled version comprised two of the twin camera units coupled together in such a way that each angle between successive lenses was 30°, which led to coverage with a consistent geometry (IGM, 1939).

The four-camera unit could be oscillated around its horizontal axis between successive sets of exposures, thus allowing the exposure of the high-oblique photographs alternately to the left and right of the flight line. The high-oblique photographs nearly do not overlap, due to their alternating left and right position. The wide image coverage allowed flying at lower altitudes (IGM, 1939). Obviously, this has led to large geometric distortions, a problem that was considered as secondary to the high precision of the photographs (Bergaglio, 2001).

2.2 | Flights and photography

The data acquisition was executed by the 7th Topocartographic Section of IGM. A three-engined, high-wing Caproni Ca-101 D2 aircraft and a crew were provided by the Italian Royal Air Force (Regia Aeronautica) to undertake the flying of the aerial photographic coverage (IGM, 1939). Flights were operated systematically, combining both direct military needs and the preparation of topographic map sheets (Bergaglio, 2001). However, there were neither available flight coordinates nor regular flight patterns (Nyssen et al., 2016).

Flights were reportedly planned to have an approximate overlap of 60% between subsequent sets of APs to ensure stereo-coverage of the terrain to use them for stereographic restitution (IGM, 1939). Overlapping areas were calculated in ArcGIS on five subsets of subsequent APs taken over Tembien (Debever, 2019) using Honeycutt’s ‘spaghetti and meatball’ Count Overlapping Polygons method (Taylor, 2019).

The lack of overlapping areas for subsequent APs within one flight line varies between 5% and 28% for the vertical APs (also referred to as ‘frames’) and between 8% and 22% for the low-oblique frames. In some flights, overlap decreased as the flight progressed, probably flight velocity was increased as clouds were building through the day (Debever, 2019). Flights over Tembien were dense in relation to Italian military needs for the Battles of Tembien (Barker, 1968). Yet, the area has not been fully covered by vertical and low-oblique frames (Figure 1). When including the high-oblique frames, the coverage is much better (Debever, 2019). Such bird’s eye views can however only be used for qualitative analysis (Figure 2).

3 | DATA RECORDS

3.1 | Original photographs

The original photographs were organized as assemblages consisting of a label with metadata and four photographs – one nadir-pointing, two low-oblique (28°30’ to the nadir)
<table>
<thead>
<tr>
<th>Country</th>
<th>Period</th>
<th>Type of online database</th>
<th>Online ordering and delivery</th>
<th>Web address</th>
<th>Number of APs</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sudan, Egypt</td>
<td>1920–1959</td>
<td>Not georeferenced</td>
<td>N</td>
<td><a href="https://www.dur.ac.uk/library/asc/sudan/">https://www.dur.ac.uk/library/asc/sudan/</a></td>
<td>Approx. 20</td>
<td>Durham University (2020)</td>
</tr>
<tr>
<td>Africa, Mainly East Africa</td>
<td>1920–1959</td>
<td>Not georeferenced</td>
<td>Y</td>
<td><a href="http://www.iwm.ac.uk">www.iwm.ac.uk</a></td>
<td>Approx. 50</td>
<td>IWM (2020)</td>
</tr>
<tr>
<td>Sudan, White and Blue Nile Regions</td>
<td>1952–3</td>
<td>Georeferenced</td>
<td>Y</td>
<td><a href="http://www.wossac.com">www.wossac.com</a></td>
<td>N/A</td>
<td>WOSSAC (2020)</td>
</tr>
</tbody>
</table>
and one high-oblique photograph (57° to the nadir) – glued on 50 × 20 cm hardboard tiles (Figure 3). The frames were cropped manually before gluing, to facilitate the contemporary interpretation of the photosets. The label, present on each assemblage, mentioned at least (1) the flight date of the photoset and (2) a few locations or landmarks covered by the flight (Figure 4). The scale of the APs varies for the vertical frames around 1:12,000 and 1:14,000, and for the oblique frames (valid at their central point) between 1:15,000 and 1:26,000 (Frankl et al., 2015a).

**FIGURE 1** Example of ground coverage and overlapping areas for the vertical and low-oblique frames of 1935–36 taken above Dogu’a Tembien. Red rectangle shows area represented in Figure 8. Base map: Jacob and Nyssen (2019)

**FIGURE 2** High-oblique aerial photograph of the mouth of Gelda River in Lake Tana (11.731126°N, 37.436163°E) in 1938. The view is towards the west. Part of Photoset No. 19381001-86-BaharDar-86-3-100 (Nyssen et al., 2020), this bird’s eye view is suited for qualitative interpretation: as compared to the 2020 conditions, the delta was much less developed in 1938; the ‘cloud’ in the lake is a sediment plume.
3.2 | Digitization and data format

As a result of a contract for scientific collaboration between Ghent University (Belgium), the Ethiopian Mapping Agency (EMA) (now the Ethiopian Geospatial Information Institute) and Mekelle University (Ethiopia) (21/2/2012), all the photographs in the archive have been transformed into digital form at the EMA offices in Addis Ababa in 2012 using a
Plustek A3 scanner (Optic Pro A320) with a resolution of 600 spi (samples per inch, a measurement of the image scanner's resolution).

### 3.3 Relocation of the aerial photographs

Data of parallel recording of location, bearing and tilt of the APs (IGM, 1939; Schermerhorn, 1970) could not (yet) be recovered (Nyssen et al., 2016). Hence, we endeavoured to locate photosets through visualization of current imagery on screen, using rough locations provided on the labels, typical landmarks on some photographs and sequences along flight lines. Though the work was repetitive and time-taking, it allowed a thorough understanding of the data set and landscapes featured. So far, 3,132 photosets of four frames could be relocated this way (Figure 5).

< Place name mentioned on label > − < serial No. > − < Place name mentioned on label > − < serial No. > − < Place name mentioned on label > − < start No. of series > − < end No. of series >.

(3)

### 3.4 Metadata structure

A major disadvantage for film-based aerial photography is the limited availability of metadata and its inconsistency (Morgan et al., 2010). The conditions in which the APs of Ethiopia in the 1930s were realized involve additional disadvantages: glass plate technology, irregular flight lines, no flight plans, recorded coordinates – if any – not transferred to the photographs. Possible identifiers for the photosets were flight date, locations mentioned on the label (Figure 4) and sequential number, between 1 and 200 which was the maximum number of glass plates the rotational magazines could hold (Nyssen et al., 2016). As the recovered original APs were not organized by flight line nor date, and made up a volume of approx. 1 m², we have chosen to scan them without preliminary manual sorting by flight lines. During the scanning operations, we had not yet a comprehension of the flight schemes and the photographs, thus, received a preliminary identifier consisting of.

(1)
Later, the date mentioned on the labels allowed organizing the photosets into folders by date. Fortunately, on most dates, there was only one flight. Only on two dates, there were two flights, one by IGM and the other by the Compagnia Nazionale Imprese Elettriche (CONIEL) in the framework of hydropower planning (Massi, 1940; Podestà, 2013), what resulted in corresponding ‘bis’ folders. For sake of identification, a simple number consisting of $\text{YYYYMMDD} - \text{< serial No.>}$ was used to allow unequivocally naming the photosets and linking them to coordinates. We have chosen to further maintain the naming given during the scanning stage in order to be able to trace back the photosets in case something would go wrong in renaming. This resulted in:

Label numbers were constructed in the same way, adding ‘-label’ at the end.

Python scripts were developed in order to prepare the scanned photosets for publication on the website. In a first script (Appendix A), all filenames were checked on consistency using regular expressions. Log files were created by the script, indicating the files complying with the file naming proposed in (1) and the files needing manual adaptations to comply with (1). After preparing all files, a second script (Appendix B) was used to automatically list all .TIF files in

![Online interface to select and order historical APs. Selected APs in this screen-print cover areas in Amhara, Tigray, Benishangul-Gumuz, Oromo, Dire Dawa, Harari, Addis Ababa and Southern regions of Ethiopia and date back to 1935-1940. Base map: ©OSM contributors](https://example.com/image.png)
a directory and its subdirectories. All filenames consistent with (1) were renamed to the structure as given in (3) and converted to .JPG files using settings that reduce the file size to about 10% of the original with nearly no quality loss (~99%). Again, log files were created to indicate possible problematic files or failed conversions. Finally, a third script (Appendix D) was used to read in a .CSV file containing a table of dates, AP numbers and coordinates. Each converted .JPG file was cross-checked with the table and linked to coordinates in case of a matching date and photo number.

The Excel file was exported as CSV, and, using QGIS, transformed into GeoJSON, a format for encoding geographic data structures (Köbben, 2014), that allowed incorporating the AP locations in a Leaflet interface.

3.5 Online interface

All aerial photographs that could be relocated have been made available to the wider scientific community through a web-based interface. The website is a fully responsive website (Sarabadani Tafreshi et al., 2017; Schade, 2014), so that it is also convenient to use on smaller screens like tablets or smartphones. Although possibly less of an issue for an AP serving tool, fully responsive properties are important given that online search engines like Google or Bing have a mobile-first approach in search rankings (Google Search Guides, 2020).

The technology used for creating the website is a mainly Javascript-focused stack, running on a VMWare virtual machine hosted by Ghent University. The backend is a Node.js (OpenJS Foundation, 2020) server protected by an NGINX (F2020 Inc., 2020) webserver (reverse proxy). The Leaflet.js map script (LeafletJS, 2020) in combination with some additional custom functionality (URL encoding, item listing, ...) and responsible for the photo selection is written in client side Javascript. In order to stress the project’s Ghent University origins, the UGent corporate identity was used to style all pages (Ghent University, 2020).

Because of the email focused nature of this project, a database or database server was not necessary to implement, dramatically improving online security. For automated email communication between the user and the project owners, the third party email service MailJet is used (MailJet, 2020). At present, the free tier is more than sufficient to satisfy all needs. Upgrade to a higher plan is relatively cheap should the need occur in the future.

The web interface (http://www.ethiopia1935.ugent.be/) uses three pages only, with short texts. A first page presents the archive, an overview map with the available photographs, and the project partners (Ghent University, Department of Geography; Mekelle University, Institute of Geoinformation and Earth Observation Sciences (IGEOS); Ethiopian Geospatial Information Institute). One sample photograph is included at high resolution and downloadable so that the users can see whether this type of APs fits their purpose.

A second page concerns the use of these aerial photographs for research and development. Given the early instrumentation used, the photographs are quite different from classic imagery. Flight lines are not straight, overlap is not always present and the images may be deformed due to their long preservation in poor conditions (Nyssen et al., 2016). There is also no systematic coverage of the area. Nevertheless, the photographs have already been used in a dozen of studies. These publications are hyperlinked on the webpage so that potential users can understand possibilities (and drawbacks) before ordering aerial photographs. For sake of copyright, the publications are not directly downloadable, but can be accessed either through the publisher’s website, or requested as offprint from the authors.

The third page allows ordering aerial photographs (Figure 6); this is limited to a maximum of 20 photographs, typically the magnitude of locality-based research in Ethiopia. The coverage of the APs is insufficient to think about synoptic use over wide areas. All photosets have been placed as .JPG files on an internal server and can be accessed through a shapefile holding the coordinates of the AP principal points. Using these coordinates, the location of the photosets is visualized through Leaflet, an open-source JavaScript library for interactive maps (https://leafletjs.com). On the map, the area covered by the APs is schematically represented by resizable circular areas, four kilometre across, so that the photoset can be selected by simple clicking. Corresponding script is available as Appendix D.

Visualization of the area of interest is possible as OpenStreetMap® imagery, OSM cycling map (renamed ‘OSM topography’, which is particularly of use in this mountainous country) and ‘Satellite’ (which is the ArcGIS online imagery). For the best visualization of the background map, it was chosen not to have names of photos on the map or while hovering over the map. The selection of APs takes place based on their location. Upon pre-selection, the name of the AP appears at the bottom of the map.

While ordering photosets, information is requested on the user and the purpose of the download request. Confirmation of non-commercial use and appropriate literature quotation is mandatory. After selection and approval by the manager, links are transmitted by email allowing a download of the selected photographs from the server.

4 TECHNICAL VALIDATION: RELIABLE USE OF DATA IN ANALYSIS

4.1 Map preparation

In the 1930s, in warfare conditions, topographic maps were produced very rapidly from the aerial photographs
by IGM’s 7th Topocartographic Section, a unit that was equipped with extensive photographic, photo-mechanical and printing facilities, located in Asmara (Nyssen et al., 2016). These initial maps were mainly planimetric: topographic characteristics were added by rough contour lines to show the relative heights, the shape and the character of the landforms (Nyssen et al., 2019). Later on, accurate maps were produced for the same areas at 1:50,000 and 1:100,000 scales (IGM, 1939). The scanned versions are available at the Istituto Geografico Militare Italiano (http://www.igmi.org/anciente/).

4.2 | Qualitative and quantitative studies

A few representative examples of the use of these historical APs include qualitative geomorphic analyses. In the 1930s, the APs were used to study the structural geomorphology of Tigray (Merla & Minucci, 1938) and more recently the long-term dynamics of mountain streams (Ghebreyohannes et al., 2015). The APs were also successfully used in a study on historical road engineering geology in the Blue Nile gorge (Hearn, 2019).

Qualitative studies of land cover change (Frankl, 2012; Hishe et al., 2020) and land tenure in feudal times (Lanckriet et al., 2015; Nyssen & Denaeyer, 2019) have also been carried out using these APs, as well as interpretations of changes to the level of Lake Hayq and urban expansion of Mekelle since the 1930s (Nyssen et al., 2016).

Qualitative change studies were carried out using the point-count method (Bellhouse, 1981; Zeimetz et al., 1976), hence without orthorectification concerned, particularly in relation to changes in land use (Guyassa et al., 2018b) in the Giba basin in Tigray. In the same area, densities of gully and SWC networks were measured by counting the number of features on transversal transect lines (Guyassa et al., 2018a).

4.3 | Georeferencing and orthorectification methods and spatially explicit studies

First to third order polynomial transformation using tie points (Hughes et al., 2006), also called rubbersheeting or spline, was used to orthorectify the APs and carry out diachronic analysis of land-use changes on the western Rift Valley escarpment (Ghebreyohannes et al., 2018), north of Dessie (Kassa et al., 2011), around church forests (Scull et al., 2017) and at Mt. Guna’s treeline (Jacob, 2015). Landsliding was investigated in Dessie, using APs that were processed in the same way (Kropáček et al., 2019).

A further step has been the reconstruction of ortho-mosaics and preparation of 3D models of the historical landscapes. We built upon the combination of Structure-from-Motion (SfM) and MultiView Stereo (MVS) (Frankl et al., 2015b; James & Robson, 2012) to construct ortho-mosaics and 3D models from aerial photographs. The SfM part makes it possible to reconstruct an area based on an unordered collection.
of images. By detecting and matching textures in the photographs, they can be matched to each other. The algorithms calculate the camera parameters and the orientation of every picture. Particularly, SfM allows reconstructing the three-dimensional scene geometry and the position of the cameras during the image acquisition period of the images captured around a scene or a landscape, even when imagery is already degraded or when the imagery lacks calibration information (Sevara et al., 2018). SfM allows this without using prior topographic data (Peterson, 2017).

To execute the process of SfM-MVS, the most common software PhotoScan, distributed by Agisoft, was used, which extends SfM with MVS. The method was successfully applied in small areas in Tigray (Frankl et al., 2015a) and in the Lake Tana basin (Frankl et al., 2019), using vertical and low-oblique frames, all manually cropped to single images.

The method was further developed for wider areas. The scanned photosets do not only contain the scene of interest (SOI), but are assemblages of four frames. The photographs each contain black borders; strips of the hardboard on which the photographs are glued are also visible on the scans. The scanned photosets were automatically cropped, divided into the individual frames (Appendix E) and contrast stretched using python scripts allowing batch processing. As the frames had been cropped before gluing on hardboard, there is no overlap between the vertical and low-oblique APs within the photosets, hence a lack of tie points between the frames which can hinder the correct alignment of the APs in a single geometrical model in the SfM processing. To overcome this problem, the subsets are slightly larger than the AP frames to include also a narrow stripe of the adjacent APs which served as a ‘false overlap’. This results in a number of model points in the overlaps between the AP from the same photoset (Figure 7). This approach improved the alignment of the APs but as a drawback it may lead to a lower geometrical accuracy of the model resulting in the occurrence of irregular shapes and gaps in the resulting ortho-mosaic (Forceville, 2018).

A dense point cloud was then built, using the calculated camera positions from the alignment step, from which it calculated depth information for each camera location. Through combination, a single dense point cloud was obtained (Figure 8), which was used as a more detailed and accurate input for the generation of Meshes, Digital Elevation Model (DEM) and the Tiled Model (Agisoft, 2018).

For the construction of a dense point cloud, we set the ‘quality’ to Ultra High, meaning that the processing was carried out with the original resolution of the photographs. Lower quality parameters result in processing results based on downscaled image sizes. The ‘Depth filtering mode’ was set to Aggressive for each time period (Debever, 2019), as recommended by Agisoft PhotoScan.

Next, a polygonal model – a so-called mesh – was generated, which served as final input for the generation of the ortho-mosaics. As we processed the frames, the mesh processing parameter ‘surface type’ was set to ‘Height field’. The necessary camera parameters (Seitz et al., 2006) were computed through SfM, and a spatial resolution of 0.50 m was achieved.

Although Agisoft Photoscan offers the possibility to georeference the ortho-mosaics during the process of orthomosaicking by adding ground control points (GCPs) after the alignment step, the option of georeferencing the ortho-mosaics in ArcMAP was preferred. An investigation of the optimal number of GCPs to be added to the mosaicking process in Agisoft Photoscan showed that with an increasing amount of GCPs, the RMSE in X and Y stagnated to an error of approximately 30 m.

As the bearing and tilt information could, unfortunately, not yet be recovered and besides this, the scanners that were used for the scanning operations of the APs could not be calibrated in order to minimize distortions caused by the scanner (Nyssen et al., 2016), a spline transformation was used to georeference the ortho-mosaics using numerous GCPs (www.pro.arcgis.com).

On a sample data set, the accuracy was measured by calculating the difference between the X-coordinates (dX) and the Y-coordinates (dY) from a checkpoint on the georeferenced data set – this is a point on the data set that was not used as GCP – and its equivalent in the current landscape. The checkpoints were chosen by means of a regular grid with 1,500 m between nodes. An estimation of the error in X and Y was then interpolated by means of the kernel density tool in ArcMAP. Errors in the data sets 1935–36 were less than 30 m in Y and less than 15 m in X (Figure 9) (Debever, 2019).

Errors could be related to the non-uniform scale and geometry of the photographs (particularly for the low-oblique frames in mountain areas) or to random errors during the localization of ground control points. None of the georeferenced data sets show high systematic errors, but some areas have a clearly higher error than the others. The extent of the ortho-mosaics was determined by the quality of the scanned aerial photographs and the number of overlapping APs (Figure 10). Poor quality of some frames is for instance related to local differences in contrast, moisture-induced damage of photographs or presence of clouds. An alternative approach could be to work with less GCPs, generate an error map and then add GCPs in the areas with greatest errors (Persia et al., 2020).

Manually cropped subsets, without creating false overlaps, have generally good orthophotographs with only occasionally an artefact or a gap. However, because of a lack of overlap between the vertical photographs and their adjacent low-oblique photographs, in most cases only one camera line is matched.
When the automatically cropped aerial photographs were processed in PhotoScan, they had a clearly different output. First of all, there were significantly more photographs matched. This is due to the false overlap between low-oblique and vertical photographs. Many of the resulting orthophotographs have however irregular shapes and gaps, and brown lines, parts of the hardboard tiles on which the photographs were glued, appearing between photographs along the flight line. Rather than processing all photographs of one flight line with the same parameters, they would need to be visually inspected and grouped, so that cropping occurs with adjusted parameters (Forceville, 2018). Recent expertise indicates that the implementation of GCPs and terrain heights (Pinto et al., 2019) could help in the ortho-rectification, while taking into account incomplete metadata (James et al., 2019).
4.4 Further relocation of photosets

The APs do not at all cover the full country, and the number of photosets is far too small. Besides, within flights some photographs could not be recovered. A major issue is that so far, we failed relocating 5,149 photosets based on merely a few place names in a wider area, particularly those without readily recognizable features. Hence, a semi-automated method to localize and scale these aerial photographs has been tested (Walter & Fritsch, 1999). At first, within an unlocated series of successive photosets, the photographs were aligned to create an ortho-mosaic, and DEM extracted (when sufficient overlap between adjacent photos). The result was used to extract the drainage network of that unknown area (Ariza-Villaverde et al., 2015). Next, similarities were searched with parts of the recent
drainage network (extracted from the SRTM30). This comparison was performed by a script to detect matches (Hussnain et al., 2016; Mustière & Devogele, 2008). The algorithm was based on Strahler’s stream orders of the rivers (Luo et al., 2014; Molloy & Stepinski, 2007), the angle difference between the historical and recent segments (Borgefors, 1988) and the scale between both networks (Forceville, 2018).

Though the method was successful on regular imagery taken in 1964, the Italian imagery from the 1930s over Ethiopia could not be relocated through it. Indeed, the algorithm performs best with high stream orders and a long array of consecutive streams, but these were difficult to obtain from the chunks of flight lines available for the studied photosets. Another problem was that incorrect drainage networks were generated because of gaps in the generated DEM and artefacts induced by the borders of the frames (Forceville, 2018).

The best option that remains is manual tracing using Google Earth or similar, possibly as geo-crowdsourcing (Porto De Albuquerque et al., 2016; Produit & Ingensand, 2018), in which interested people try to locate photosets from flights in selected approximate coverage areas (Figure 11).

4.5 | Automated aerial photograph interpretation

These historical aerial photographs offer a great opportunity to prolong the timespan over which land cover data, and studies have been undertaken to understand the complexity of land cover changes in a more accurate way (see above). Yet, analogous interpretation of black and white orthophotographs is very time-consuming. The possibilities of such historical aerial imagery for the (semi-) automated extraction

![FIGURE 11 Flight lines and approximate areas for which photosets are available, but georeferencing needed](image-url)
of the land cover classes ‘cropland’ and ‘woody vegetation’ were investigated. A subset of the APs was analysed within the study area covering 323 km² in the eastern part of the Dogu’a Tembien district in the Tigray region (Figure 1). For the classification of cropland, we applied the combination (Vogels et al., 2017) of object-based classification technique and random forest machine learning technique (Breiman, 2001; Liaw & Wiener, 2002), of which the classification results achieved a Kappa coefficient (Lillesand et al., 2008) between 0.40 and 0.70. From the 22 different object variables that were taken into account during the classification process, textural variables based on the grey-level co-occurrence matrix seemed to play a more important role during the classification process than geometrical variables and the brightness of the objects (Debever, 2019).

In view of the large number of possible objects in areas with woody vegetation, it was preferred to use a pixel-based classification technique for the ‘woody vegetation’ that was anticipated to appear with a lower brightness (Kadmon & Harari-Kremer, 1999; Sharp & Bowman, 2004). At first, this led to an overestimation of the amount of ‘woody vegetation’. Whereas such errors would be corrected through manual editing in case of smaller areas (Frankl et al., 2019), for Tembien, the results were optimized by means of a process that integrated the pixel- and object-based approaches (Debever, 2019). These historical APs offer many perspectives for analysing and modelling land cover changes in Ethiopia, but there are challenges related to a sometimes lesser quality of the APs.

ACKNOWLEDGEMENTS
Availing the collection of historical aerial photographs would not have been possible without the strong support by Sultan Mohammed, director of EMA in 2012. The late Gordon Petrie (University of Glasgow) contributed a lot to sharpening our insights in the instrumentation of the 1930s. Further, Koen Hufkens (Harvard University), Neil Munro (Mekelle University) and Maarten Wynants (University of Plymouth) hinted available archives of historical aerial photographs covering parts of Africa. Last but not least, communities in many of the portrayed scenes (Lake Tana basin, Dogu’a Tembien, Kemise graben) are gratefully acknowledged for guiding team members to specific locations of ground control points and other geographical features in the APs.

CONFLICT OF INTERESTS
The authors declare that they have no conflict of interest.

AUTHOR CONTRIBUTIONS
Jan Nyssen traced the collection of aerial photographs, unearthed it, set up the institutional framework, relocated photosets and prepared the manuscript. Martijn Debever evaluated the validity of the data set for change studies, relocated the larger part of photosets and contributed to manuscript writing. Gezahegne Gebremeskel contributed to unearthing the data set and to manuscript writing. Bart De Wit prepared the data set for online availing and contributed to manuscript writing. Kiros Meles Hadgu set up the institutional framework for data set acquisition and contributed to manuscript writing. Steven De Vriese prepared the data set for online availing and contributed to manuscript writing. Jeffrey Verbeurgt prepared the data set for online availing and contributed to manuscript writing. Amaury Frankl evaluated the validity of the data set for preparation of orthophotographs and for change studies, and contributed to manuscript writing. Tulu Besha set up the institutional framework for data set acquisition and critically read the manuscript. Jan Kropáček evaluated the validity of the data set for preparation of orthophotographs and for change studies, and contributed to manuscript writing. Astrid Forceville evaluated the validity of the data set for preparation of orthophotographs and contributed to manuscript writing. Biadgilgn Demissie set up the institutional framework for data set acquisition and contributed to manuscript writing.

DATA AVAILABILITY STATEMENT
The data set of aerial photographs of Ethiopia in the 1930s has been availed to the wider scientific community through the Pangaea website http://doi.org/10.1594/PANGAEA.920077 (Nyssen et al., 2020). Selection and downloading of relocated photographs are best done using web interface http://www.ethiopiap1935.ugent.be/.

OPEN PRACTICES
This article has earned an Open Data badge for making publicly available the digitally-shareable data necessary to reproduce the reported results. The data is available at https://doi.org/10.1594/PANGAEA.920077 and http://www.ethiopiap1935.ugent.be/ Learn more about the Open Practices badges from the Center for OpenScience: https://osf.io/tvyyz/wiki.

ORCID
Jan Nyssen https://orcid.org/0000-0002-2666-3860
Jeffrey Verbeurgt https://orcid.org/0000-0002-9045-6049

REFERENCES


Project. https://tanganyikabase.africamuseum.be/aerial_photos_survey.pdf. (last accessed on 19/2/2021)


APPENDIX A

Python script for filename consistency

```python
# -*- coding: utf-8 -*-

Created on Mon May 11 13:13:24 2020

@author: Jeffrey Verbeurgt

This script is used to check all filenames of the .TIF-files. The .TIF-files are located in the original directories, which are named after date of acquisition. The .TIF-files are either aerial pictures or the label describing details of an aerial picture. The general structure of the filenames is:

[filename]-[photono]-[startnumber serie]-[endnumber serie].TIF

OR

[filename]-[photono]-[startnumber serie]-[endnumber serie]-label.TIF

The script generates three logfiles:

1. Summary containing the number of accepted and rejected filenames
2. File containing all accepted filenames and their location
3. File containing all rejected filenames and their location

Based on the logfiles, the filenames can be manually adapted to conform to the filenames structure.

```
# Use timestamp to separate logfiles from different runs
timestamp = datetime.datetime.now().strftime("%Y %m %d %H:%M:%S")

# create log files: summary, correct filenames, wrong filenames
# create a string which will contain full log for each file
log_summary = "log filename summary for run
{\n".format(timestamp)
log_summary_filename = str("01_" +
    log_summary.rstrip('\n').replace(" ", ") +
    ".log") # name of logfile

log_wrong = "log filename wrong filenames for run
{\n".format(timestamp)
log_wrong_filename = str("01_" +
    log_wrong.rstrip('\n').replace(" ", ") +
    ".log") # name of logfile

log_correct = "log filename correct filenames for run
{\n".format(timestamp)
log_correct_filename = str("01_" +
    log_correct.rstrip('\n').replace(" ", ") +
    ".log") # name of logfile

# Some counters to use in the logfiles
counter_wrong_filenames = 0
counter_correct_filenames = 0
counter_total_filenames = 0

# Actual filenaming analysis
# get a list of all sub-directories in parent directory
my_dirs = [d for d in os.listdir(directory) if os.path.isdir(os.path.join(directory, d))]

# define patterns for the regular expressions
pattern_label = '^.*-[0-9]+-[0-9]+.*\-label\..tif$' # pattern for label files
pattern_aerial = '^.*-[0-9]+-[0-9]+.*\..tif$' # pattern for aerial files

# loop over all files in all directories
for directory in my_dirs:
    # loop over each dir
    log_wrong += ("DIR: " + str(directory) + "\n") # append current dir to log
    log_correct += ("DIR: " + str(directory) + "\n") # append current dir to log

    for filename in os.listdir(directory):
        # loop over each file in dir
        counter_total_filenames += 1 # add 1 to count of total files
        result_label = re.match(pattern_label, filename.lower())
        result_aerial = re.match(pattern_aerial, filename.lower())

        if result_label:
            log_correct += "Label found: " + filename + "\n"
            counter_correct_filenames += 1

        if result_aerial:
            log_correct += "Aerial found: " + filename + "\n"
            counter_correct_filenames += 1
if result_label and not result_aerial: #if match with
label regex
counter_correct_filenames += 1 #add 1 to counter of
accepted files
log_correct += "{}\n".format(filename) #write filename
to log

elif result_aerial and not result_label: #if match with
aerial regex
counter_correct_filenames += 1 #add 1 to counter of
accepted files
log_correct += "{}\n".format(filename) #write filename
to log

else: #if filename not matches
counter_wrong_filenames += 1 #add 1 to counter of
rejected files
log_wrong += "{}\n".format(filename) #write filename
to log

#do this at end of looping over all files of a dir
#this is purely for esthetical reasons in the logfiles
log_wrong += "{}\n".format(counter_wrong_filenames)
log_correct += "{}\n".format(counter_correct_filenames)

# Write results to logfiles
with open(log_summary_filename, "w") as log_summary_file:
    log_summary_file.write("Number rejected files:\n{}\n".format(counter_wrong_filenames))
    log_summary_file.write("Number accepted files:\n{}\n".format(counter_correct_filenames))
    log_summary_file.write("Number total files:\n{}\n".format(counter_total_filenames))

with open(log_correct_filename, "w") as log_correct_file:
    log_correct_file.write(log_correct)

with open(log_wrong_filename, "w") as log_wrong_file:
    log_wrong_file.write(log_wrong)
APPENDIX B

Python script for consolidation of filenames and conversion to .JPG

```python
# -*- coding: utf-8 -*-

Created on Mon May 11 20:24:34 2020

@author: Jeffrey Verbeurgt

This is the second script to be used in the process of the file renaming.

Make sure all filenames are conform the following structure:
   [filename]-[photonumber]-[startnumber serie]-[endnumber serie].TIF
   OR
   [filename]-[photonumber]-[startnumber serie]-[endnumber serie]-label.TIF

The script renames and converts the files, output is stored in one folder.
Renaming according to following structure:
   [DIRNAME]-[photonumber]-[filename]-[photonumber]-[startnumber serie]-[endnumber serie].TIF
   OR
   [DIRNAME]-[photonumber]-[filename]-[photonumber]-[startnumber serie]-[endnumber serie]-label.TIF

The original .TIF-files are converted to JPEG-files. Settings are used which reduce filesize with 90% without giving up on much quality.

#---------------------------import modules
#---------------------------import os  #to loop over files in directories
#---------------------------import re  #to use regular expressions on the filenames
#---------------------------import datetime  #to timestamp the log-files
#---------------------------from PIL import Image  #to convert TIF to JPEG

#---------------------------user defined settings for location of input and output
#---------------------------mypath = "."
#---------------------------outpath_accepted = "accepted_files_output"  #for TIF>>JPEG with correct filename
#---------------------------outpath_rejected = "rejected_files_output"  #for TIF>>JPEG with wrong filename

#---------------------------create the logfile
#---------------------------timestamp = datetime.datetime.now().strftime("%Y %m %d %H_%M_%S")
#---------------------------create log files: summary, succesful filenames, failed filenames
#---------------------------create a string which will contain full log for each file
```
log_summary = "log rename and convert summary for run 
{}\n".format(timestamp)
log_summary_filename = mypath + "/" + str("02_" +
    log_summary.rstrip('\n').replace(" ", ") + 
    ".log")

log_failed = "log failed rename and convert for run 
{}\n".format(timestamp)
log_failed_filename = mypath + "/" + str("02_" +
    log_failed.rstrip('\n').replace(" ", ") + 
    ".log")

log_sucess = "log sucessful rename and convert for run 
{}\n".format(timestamp)
log_sucess_filename = mypath + "/" + str("02_" +
    log_sucess.rstrip('\n').replace(" ", ") + 
    ".log")

#Some counters to use in the logfiles
counter_failed_files = 0
counter_sucess_files = 0
counter_total_files = 0

#----------------------------------------------------------------------------------------
# Perform actual analysis to rename and convert each file
#----------------------------------------------------------------------------------------
#define patterns for the regular expressions
pattern_label = '^.\-{0-9}\-{0-9}\-{0-9}\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-*\-
location, number, series1, series2, label =
filename.rsplit(',',4)
new_filename = str(os.path.basename(root)) + '-' +
number + '-' + filename
g2g = 1 #File is according to filenaming structure
else:
with aerial regex
location, number, series1, series2 =
filename.rsplit(',',3)
new_filename = str(os.path.basename(root)) + '-' +
number + '-' + filename
g2g = 1 #File is according to filenaming structure
log_failed += ('Something wrong with filename
{}

'.format(os.path.join(root, name)))
#Print to stdout to inform researcher on progress
print(filename + 'n' + new_filename + 'n
')
#test if already converted in previous run + if
correct filename
if os.path.isfile(os.path.join(mypath,
outpath_accepted, new_filename) + '.jpg') and g2g == 1:
counter_successes_files += 1

# If a jpeg is NOT present + correct filename:
create JPEG
elif not os.path.isfile(os.path.join(mypath,
outpath_accepted, new_filename) + '.jpg') and g2g == 1:
outfile = os.path.join(mypath, outpath_accepted,
new_filename) + '.jpg'
try:
im = Image.open(os.path.join(root, name))
im.thumbnail(im.size)
https://jdhaio.github.io/2019/07/20/pil_jpeg_image_quality/
im.save(outfile, "JPEG", quality=95,
subsampling=0)
log_success += ('Successful jpeg for {} to
{}

'.format(name, new_filename))
log_success += ('Old file:
{}

'.format(os.path.join(root, name)))
log_success += ('New file:
{}

'.format(outfile))
counter_successes_files += 1
except:
log_failed += ('Exception: jpeg for {} to
{}

'.format(name, new_filename))
log_success = log_success.rsplit('n',4)[0]
#Remove lines written to succes log


counter_successes_files -= 1
counter_failed_files += 1

test if already converted in previous run + wrong
filenameformat
elif os.path.isfile(os.path.join(mypath,
outpath_rejected, new_filename) + '.jpg') and g2g == 0:
log_failed += ("Already converted file
{}\n".format(os.path.join(root, name)))
counter_sucses_files += 1

    # If a jpeg is *NOT* present + wrong filename
create JPEG
    else:
        outfile = os.path.join(mypath, outpath_rejected,
new_filename) + ".jpg"
        try:
            im = Image.open(os.path.join(root, name))
im.thumbnail(im.size)
            #see:
https://jdhao.github.io/2019/07/20/pil_jpeg_image_quality/
            im.save(outfile, "JPEG", quality=95,
subsamples=0)
            log_sucses += ("Succesful jpeg for {} to
{}\n".format(name, new_filename))
            log_sucses += ("Old file:
{}\n".format(os.path.join(root, name)))
            log_sucses += ("New file:
{}\n\n".format(outfile))
counter_sucses_files += 1
            except:
                log_failed += ("exception: jpeg for {} to
{}\n".format(name, new_filename))
            log_sucses = log_sucses.rsplit("\\n",4)[0]
#Remove lines written to success log
counter_sucses_files -= 1
counter_failed_files += 1

# Write results to logfiles

#Write results to logfiles

#Write results to logfiles

with open(log_summary_filename, "w") as log_summary_file:
    log_summary_file.write(log_summary)
    log_summary_file.write("Number
success:\t{}\n".format(counter_failed_files))
    log_summary_file.write("Number
failure:\t{}\n".format(counter_sucses_files))
    log_summary_file.write("Number
total:\t{}\n".format(counter_total_files))

with open(log_sucses_filename, "w") as log_sucses_file:
    log_sucses_file.write(log_sucses)

with open(log_failed_filename, "w") as log_failed_file:
    log_failed_file.write(log_failed)
APPENDIX C

Python script for reading all .JPG in a .CSV file

```python
# -*- coding: utf-8 -*-

Created on Tue May 12 21:21:22 2020

#author: Jeffrey Verbeurgt

This script reads in a file containing in each row a key-value, X and Y. The key-value has following structure:
[YEAR]-[MONTH]-[DAY], [photonumber]

In the next step, the script reads in all JPEG-files in a directory, extracts the key-value from the filename and looks up whether the key exist in the input. When a match is found, the filename is combined with the key and the coordinates in an output file. If no match is found, the filenames and constructed key are written to a second file.

```
```python
import os
import pandas as pd

# make a dictionary of the known elements, this speeds up the comparison later on
# Key is date + fotonumber, value is a list with X, Y, and filenames
dict_known = {}
for index, row in df_known.iterrows():
    dict_known[row[2]] = [row[0], row[1]]

# make a dictionary which we will fill with the unknown elements
dict_unknown = {}

# Loop over all JPEG in the directory as defined above
for filename in os.listdir(mypath):
    if filename.endswith(".jpg"):
        # Extract year, month, day
        # Construct the key from the filename
        if "bis" in filename:
            key_bis = datum + ", " + filename[12:].split("-")[0] + "bis" # add date and foto number together
            key = datum + ", " + filename[9:].split("-")[0] # add date and foto number together
        else:
            key = datum

        # Check whether constructed key exists in actual key
        if key in dict_known:
            dict_known[key].append(filename) # if match, add filename to row
        else:
            dict_known[key].append(filename) # if match, add filename to row

        # if match with key in unknown, add filename
        if "bis" in filename:
            dict_known[key_bis].append(filename)
        else:
            dict_known[key].append(filename)

    else: # if not a jpg, skip this file
        continue

# Write dictionaries to txt file
```

with open(filename_matching,'w') as file1:
    #First write header
    file1.write("X;Y;key;pictures;labels;nr pictures;nr labels\n")
    for key in dict_known:
        #define an empty line in output string
        line = ""
        #add to output string the X, Y and key (semicolon-separated)
        line += str(dict_known[key][0]) + ";" + str(dict_known[key][1]) + ";" + str(key) + ";"
        #Since there are picture-files and label-files, we need to check each
        #file to which group it belongs
        pictures = [] #list for picture filenames
        labels = [] #list for label filenames
        for element in dict_known[key][2:]:
            if "label" in element.lower():
                labels.append(element)
            else:
                pictures.append(element)
        #add list of picture filenames-semicolon-label filename
        line += str(len(pictures)) + ";" + str(labels) + ";"
        line += str(len(pictures)) + ";" + str(len(labels)) + ";n"
        file1.write(line)

with open(filename_nonmatching,'w') as file2:
    #First write header
    file2.write("key;pictures;labels;nr pictures;nr labels\n")
    for key in dict_unknown:
        #define an empty line in output string
        line = ""
        #add to output string the X, Y and key (semicolon-separated)
        line += str(key) + ";"
        #Since there are picture-files and label-files, we need to check each
        #file to which group it belongs
        pictures = [] #list for picture filenames
        labels = [] #list for label filenames
        for element in dict_unknown[key]:
            if "label" in element.lower():
                labels.append(element)
            else:
                pictures.append(element)
        #add list of picture filenames-semicolon-label filename
        line += str(len(pictures)) + ";" + str(labels) + ";"
        line += str(len(pictures)) + ";" + str(len(labels)) + ";n"
        file2.write(line)
APPENDIX D

Script for interactive map

// References to DOM elements
var btnOrder = document.getElementById("btn-order");
var itemsInList = document.getElementById("items-in-list");
var requestedItems = document.getElementById("requested-items");
var txtRequests = document.getElementById("requests");
var txtRequestsFeedback = document.getElementById("requests-feedback");
var mapRef = document.getElementById("osm-map");
var selectionsList = document.getElementById("selections-list");
var orderForm = document.getElementById("order-form");
var overlay = document.getElementById("overlay-wzmap");
var txtName = document.querySelector("input[name='name']");
var txtWork = document.querySelector("input[name='work']");
var txtResearch = document.querySelector("input[name='research']");
var txtContributors =
document.querySelector("input[name='contributors']");
var txtEmail = document.querySelector("input[name='email']");
var agree1 = document.querySelector("input[name='agree_1']");
var agree2 = document.querySelector("input[name='agree_2']");
var btnSubmitOrder = document.getElementById("btn-submit-order");
var btnCancelOrder = document.getElementById("btn-cancel-order");

// Array that will hold users selection
var arrOrders = [];

// Supported tile layers
var topoLayer = L.tileLayer("https://a.tile.openstreetmap.org/{z}/{x}/{y}.png", {
  maxZoom: 17
});
var cycleLayer = L.tileLayer("https://a.tile-cyclosm.openstreetmap.fr/cyclosm/{z}/{x}/{y}.png", {
  maxZoom: 17
});
var aerialLayer = L.tileLayer("https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}",{ maxZoom: 17 });

// For use in map legend
var baseLayers = [
  "OpenStreetMap": topoLayer,
  "Satellite": aerialLayer,
  "OSM topography": cycleLayer
];

// Add map (view settings replaced by map.fitBounds {from geojson data})
var map = L.map('osm-map', {
  center: [44.096621, 06589108, 39.115447990846875],
  zoom: 10,
  layers: [topoLayer]
});
L.control.layers(baseLayers, null, {collapsed: false}).addTo(map);

// Array for all markers
var arrMarkers = [];

// Add layer with all markers
var markersLayer = L.geoJSON({data}, {
  onEachFeature: onEachFeature,
pointToLayer: function (feature, latlng) {
    return arrMarkers[feature.properties.Name] = 
    L.circle(latlng, {
        radius: 1000,
        fillColor: "#e9f0fa",
        color: "#1E64C8",
        weight: 1,
        opacity: 1,
        fillOpacity: 0.9
    });
}).addTo(map);

// Add click event to marker; show popup with photo name
function onEachFeature(feature, layer) {
    layer.on('click', function (e) {
        onMarkerClick(feature.properties.Name);
    });
    //layer.bindPopup(feature.properties.Name.replace('.TIF',''))
    , {closeButton: false});
    //layer.on('mouseover', function() { layer.openPopup(); });
    //layer.on('mouseout', function() { layer.closePopup(); });
}

// Zoom to all markers
map.fitBounds(markersLayer.getBounds());

// Event handler for marker clicks
function onMarkerClick(markerName) {
    if (arrOrders.indexOf(markerName) === -1) {
        // Add to order list
        if (arrOrders.length < 20) {
            // Change color to indicate selection
            setMarkerColor(markerName, '#1E64C8');
            arrOrders.push(markerName);
        } else {
            alert("Maximum of 20 photographs reached."");
        }
    } else {
        // Change color back to default
        setMarkerColor(markerName, '#e9f0fa')
        // Remove from order list
        arrOrders = arrOrders.filter(function(item) {
            return item !== markerName
        })
        updateListCount();
        processSelectionsList();
    }
}

function processSelectionsList() {
    var selectionsHtml = '';
    arrOrders.forEach(function(item) {
        selectionsHtml += '<li>' + item.replace('.jpg','') + '<li>';
    })
    if (arrOrders.length === 0) selectionsHtml += '<li>-</li>';
    selectionsList.innerHTML = selectionsHtml;
}
function updateListCount() {
    itemsInList.innerHTML = arrOrders.length == 1 ? '1 item' : arrOrders.length + ' items';
    requestedItems.innerHTML = arrOrders.length;
    btnOrder.disabled = arrOrders.length === 0 ? true : false;
}

function setMarkerColor(markerName, color){
    arrMarkers[markerName].setStyle({fillColor : color})
}

// Event handler for order button clicks
btnOrder.onclick = function() {
    var requestsStr = '';
    var requestsFeedbackStr = '';
    arrOrders.forEach(function (item) {
        var lng = data.features.filter(e => e.properties.Name === item[0].properties.X);
        var lat = data.features.filter(e => e.properties.Name === item[0].properties.Y);
        var point = new GeoPoint(lng, lat);
        requestsStr += item.replace('.jpg', '') + '<br>';
        requestsStr += 'Photo: http://we12s018.ugent.be/download/' + window.btoa(item) + '<br>';
        requestsStr += 'Label: http://we12s018.ugent.be/download/' + window.btoa(item.replace('.jpg', '-label.jpg')) + '<br>';
        requestsStr += 'Location of principal point: lat ' + point.getLatitude() + ', long ' + point.getLongitude() + '<br>';
        requestsFeedbackStr += item.replace('.jpg', '') + '<br>';
    });
    txtRequests.value = requestsStr;
    requestsFeedback.value = requestsFeedbackStr;
    overlay.style.display = "block";
    return false;
}

// Event handler for order button clicks
// Checks user input and submits form
btnSubmitOrder.onclick = function() {
    if (txtName.value.length === 0 ||
        txtWork.value.length === 0 ||
        txtResearch.value.length === 0 ||
        txtContributors.value.length === 0 ||
        txtEmail.value.length === 0 ) {
        alert("Please fill out all form fields.");
        return false;
    }
    if (!isValid(txtEmail.value)) {
        alert("Please enter a valid email address.");
        return false;
    }
    if (!agree1.checked || !agree2.checked) {
        alert("Please confirm you agree with our terms.");
    }
return false;
}
if (txtEmail.value.toUpperCase().indexOf('GMAIL') > -1 ||
    txtEmail.value.toUpperCase().indexOf('YAHOO') > -1 ||
    txtEmail.value.toUpperCase().indexOf('HOTMAIL') > -1 ||
    txtEmail.value.toUpperCase().indexOf('AOL') > -1 ||
    txtEmail.value.toUpperCase().indexOf('OUTLOOK') > -1 ||
    txtEmail.value.toUpperCase().indexOf('ZOHO') > -1 ||
    txtEmail.value.toUpperCase().indexOf('YANDEX') > -1 ||
    txtEmail.value.toUpperCase().indexOf('ICLOUD') > -1 ||
    txtEmail.value.toUpperCase().indexOf('GMX') > -1 ||
    txtEmail.value.toUpperCase().indexOf('PROTONMAIL') > -1 ||
    txtEmail.value.toUpperCase().indexOf('TUTANOTA') > -1)
{
    alert("Please use an institutional email address.");
    return false;
}
orderForm.submit();
}
btnCancelOrder.onclick = function() {
    overlay.style.display = "none";
}
APPENDIX E

Script for cutting of four individual frames from the photoset, named *_A, B, C and D

```python
# subsets four individual photos from each photoset of Italian 1936 APs
# in the input directory
from PIL import Image
import os

mydir = r'D:\AP_Ethiopia_1930s\19381112_BlueNile3'

os.chdir(mydir)

files = os.listdir(mydir)

outdir = mydir+'\subsets'

print 'outdir: ', outdir

if not os.path.exists(outdir):
    os.makedirs(outdir)

for name in files:
    if not 'label' in name:
        print 'current file: ', name
        img = Image.open(name)

        p, l = img.size
        print 'image size :', p, l

        # rotate image if landscape
        if p > l:
            temp = p
            p = l
            l = temp

        # initial crop of the photoset
        cru = int(0.02 * l)
        crl = int(0.01 * l)

        imgCR = img.crop((0, cru, p, l-crl))

        # define four subsets
```

```
es = int(0.01 * l)  # number of overlap pixels between
the subsets
# print number of overlap pixels between the subsets: ', es
p, l = imgCR.size
# select one of the standard sizes of the subset frame
(3800, 3700 and 3600)
if p > 3800:
    PP = 3800
elif p > 3700 and p <= 3800:
    PP = 3700
elif p > 3600 and p <= 3700:
    PP = 3600
else:
    PP = p
LL = 2230
os.chdir(outdir)

# subset A
area = (0, 0, PP, LL)
sub = imgCR.crop(area)
sub.save(name[:4] + '_A.tif')

# subset B
area = (0, int(l/4 - es), PP, int(l/4 - es) + LL)
sub = imgCR.crop(area)
sub.save(name[:4] + '_B.tif')

# subset C
area = (0, int(l/2 - es), PP, int(l/2 - es) + LL)
sub = imgCR.crop(area)
sub.save(name[:4] + '_C.tif')

# subset D
area = (0, l-LL, PP, l)
sub = imgCR.crop(area)
sub.save(name[:4] + '_D.tif')

os.chdir(mydir)
```