A Computer Vision Application for Predicting Satellite Imagery Focusing on Glacier Evolution

Maria-Minerva VONICA
West University of Timişoara
Timişoara, Romania.
maria.vonica98@e-uvt.ro

Marc Eduard FRINCU
West University of Timişoara
Timişoara, Romania.
mfrincu@info.uvt.ro

***Abstract*—Over the last decade, climate change has impacted Earths' atmosphere and environment more than anytime before, glaciers being the most sensitive indicators. In this work we are going to analyse the retreat of glaciers over time by using advanced computer vision algorithms on aerial imagery collected from the Landsat 8 satellite. Our aim is to detect movement based on dense optical flow generation over a time series of images and use the results for generating new ones in the series. Our results show that we were able to get relevant information by extracting motion tendencies across time and we have successfully generated new snapshots based on these.**

***Index Terms*—Glacier Retreat, Computer Vision, Landsat 8, Image alignment, Dense Optical Flow**

I. Introduction

A. State of the Art

In [Winsvold et al., 2016], the authors use both Sentinel and Landsat satellites for image collection in order to produce a higher quantity of qualitative images for glacier mapping, while also covering wider areas. They propose four approaches for studying automatic glacier mapping. The first is by analysing different band ratios for multispectral image creation while the second proposes robust methods used for improving glacier mapping by exploitation of seasonal variation yielded by the spectral properties of snow. The third one highlights spatio-temporal variation of glacier surface types and the fourth one analyses how the chosen band ratio images generated from the first application can be used for automatically detecting changes in glaciers. Their results show that the alpine and arctic advances and retreats provide the most visible signs of climate change and they propose to revisit current methods for glacier mapping in order to reduce the practice of manual glacier mapping.

Another paper which studies glacier sensitivity to climate change is [Tak and Keshari, 2020], in which the authors propose a remote sensing based framework used on multispectral satellite imagery, used for studying the ablation and accumulation processes of glaciers in order to quantify their mass balance. They have studied the mass loss during of the Parvati glacier, located in the western Himalaya, between 1998 and 2016. Their results show that the glacier responds to climate change factors each year through a high mass loss, resulting in a strong effect on water availability and river flows.

In [Racoviteanu et al., 2019], the authors propose a decision-based image classification algorithm for separating ice, debris and snow surfaces on glaciers and extracting snow lines both monthly and annually. They achieved this by automatically partitioning glacier surfaces through band ratios combined with topographic criteria which were extracted for each pixel. They have conducted the study on Karakoram and Trishuli regions located in eastern Himalaya, with images taken from 2000 to 2016 and they have concluded that snow lines were the most sensitive to manual corrections, elevation dataset, topographic slope and the calculated thresholds for the band ratios for the spring and winter months.

In [Winsvold et al., 2018], the authors propose descriptive methods for glacier mapping by using aerial time series produced by the Synthetic Aperture Radar sensor from Radarsat-2 and Sentinel-1A. Out of their five scenarios, the first tracks transient snow lines and it proved to correlate with both Landsat 8 and Sentinel-2A produced aerial images. As a conclusion, they state that automatically derived satellite imagery products prove to be important in analysing glacier change.

Another interesting approach to determine glacier melt was analysed in [Wang et al., 2017], where the authors have compared the band ratio method with the Normalized Difference Snow Index (NDSI) one for extracting parameters which highlight glaciers. They have conducted their studies on the Karakorum area with satellite imagery collected from Landsat 8, starting from 2014. Their results show that for boundary extraction, the band ratio technique yielded better results. They also state that visual interpretation is still an major factor in analysing the obtained results.

As for motion estimation, Farnebäck, Gunnar proposes an interesting approach in [Farnebäck, 2003]. The author presents an algorithm for estimating motion between two frames by approximating each neighbourhood of both by quadratic polynomials. Their transformation under translation is then analysed in order to extract displacement fields.

B. Motivation

Over the last years, climate changes has affected carbon dioxide levels in the atmosphere, which are now growing more than they can be naturally eliminated, causing an increase in phenomena such as floods, sea level rise and a decrease of water availability. Glaciers are the most sensitive indicators to these changes and through the means of Earth observation satellites, enough satellite imagery has been collected over the

years such that confident analysis techniques can be carried based on the archived datasets. As an addition to this, over the last years we have developed more powerful tools which, due to increased storage and computational power, are able to take on such investigations.

The goal of this thesis is to make use of such data by analysing changes of glaciers in the last decade with the purpose of creating predicted images which could highlight future glacier retreat and different patterns of accumulation. Creating support for easy asset acquisition of satellite imagery is also important, since current methods do not scale for larger datasets.

II. Proposed Approach

One of the most popular services for satellite imagery downloading is USGS Earth Explorer, which can query and order aerial data collected from various satellites. The tool is particularly useful when the main focus is to analyse a specific area rather than trying to acquire a large dataset of scenes. However, downloading a large set of assets proves to be rather difficult by using this tool alone, since the parameters for each scene need to be manually set. On top of this, the query results have to be picked by hand and then passed for downloading through another application which handles their bulk download.

In order to fix the problem of excessive manual labour which appeared by using the USGS Earth Explorer, we rather implemented an endpoint of the SpatioTemporal Asset Catalog API (http://nsidc.org/data/glacier_inventory/index.html). Searching data through manual input for each glacier is transferred to using a script to bulk search and download based on an already existing database which has over 130.000 entries representing glaciers and their afferent parameters (such as location, elevation and name).

By using this method we can choose which glaciers we want to download directly from the database and the dataset will be automatically structured as they are fetched. Since there might be clouds which could obfuscate the area of interest in the image, we also added the option to set a maximum allowed cloud coverage, specified when running the script. This tool works as a plug-in and it is independently ran from the main application.

The approach for solving this problem starts with extracting ice and snow pixels from the scenes collected by Landsat 8 sensors. In order to do so, we have decided to use the **Normalized Snow Difference Index** measure with a threshold of 0.3, which is obtained through Equation 1.

$$NDSI = \frac{green - swir1}{green + swir1} \tag{1}$$

Snow and ice usually have a very low reflectance in the shortwave infrared spectrum and very high in the visible one, which is useful for mapping out most types of clouds from the scene [Yan et al., 2015]. We can then apply the thresholding function from Equation 2 in order to separate snow free land from covered one.

$$thresholding(pixel) = \begin{Bmatrix} \text{snow/ice}, & NDSI \geq 0.3 \\ \text{snow-free}, & NDSI < 0.3 \end{Bmatrix} \tag{2}$$

For easier visual analysis, we have coloured the result such that pixels which are considered to represent snow are white, while snow free land is represented with green. The resulting NDSI image can be viewed in Figure 1.

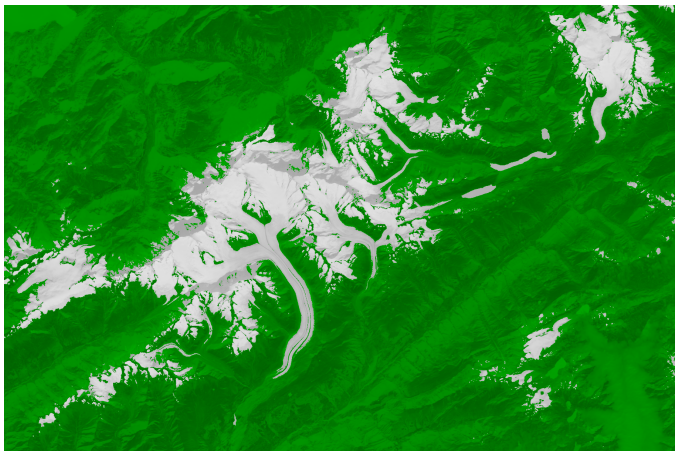

Fig. 1. NDSI image of scene LC81950282014271, Jungfrau-Aletsch-Bietschhorn glacier.

A. Alignment

Landsat's trajectory orbiting Earth is not precise, therefore not all images will be aligned pixel-to-pixel. The green and swir1 bands have a spacial resolution of 30 meters, therefore a misalignment of just 50 pixels we would end up with a 1.5 km difference between two scenes. Tracking pixel motions through a series of images without aligning them first would yield in erroneous results, due to inconsistent geographical coordinates. Figure 2 highlights the misalignment between scenes LC81950282013316 and LC81950282013364.

We have solved this problem by using a strong keypoint detection algorithm which collects features from all bands of a scene and matches them with a given reference. In our case, we have found that the most reliable keypoints were found in mountain edges and other geographical features present in the image. The computer vision algorithms used for this are **ORB (Oriented FAST and Rotated BRIEF)** together with **Harris Corner Detector** and **RANSAC (Random Sample Consensus)**, applied on the raw 16bit grayscale bands of each scene.

ORB represents a fusion between the features extracted by using the fast accelerated segment test (FAST) keypoint detector combined with the binary robust independent elementary features (BRIEF) descriptor. The FAST detector finds keypoints in the image and uses Harris corner measure to select a number of top points from the generated list. We have found that keeping the top 25% matches extracted from 5000 keypoints yielded the best results. Splitting the

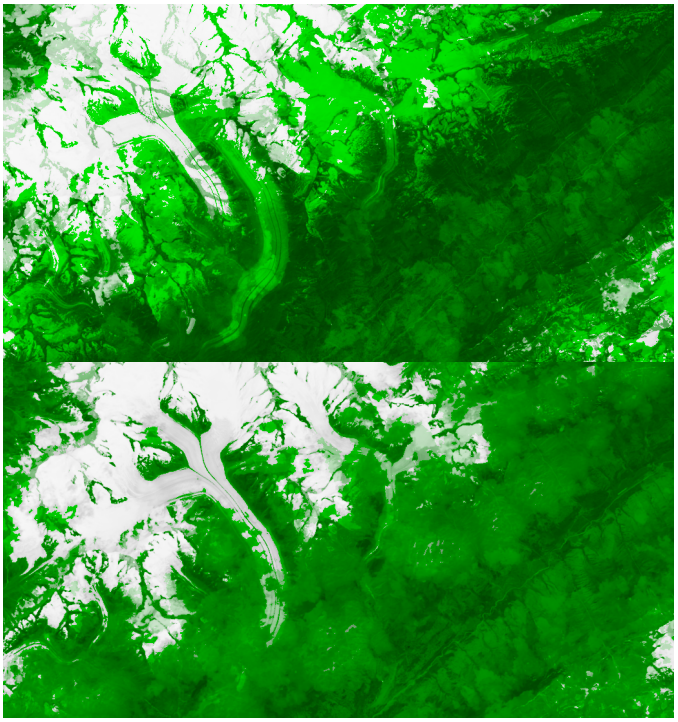

Fig. 2. Overlapped unaligned (top) and aligned (bottom) NDSI scenes LC81940282014216LGN01 and LC81940282013213LGN01.

image into multiple boxes and applying the ORB algorithm on each separate part of the image such that features are homogeneously distributed also improved valid match making. As a final optimisation, we set the maximum allowed shifting euclidean distance between any two pixels to be at most 200, so 6km on the map, therefore getting rid of outlier matches based on the distance.

The obtained matches can be then used to create an affine transformation matrix, generated through the **Random Sample Consensus (RANSAC)** algorithm, proposed by Fischler and Bolles in [Fischler and Bolles, 1981]. The affine transformation matrix describes the rotation and translation of the image to be aligned in comparison to its reference and it is used for warping it such that no two pixels are misaligned.

B. Movement extraction through optical flow

Since our dataset can be viewed as a time series of satellite images, we propose that in order to generate a new image of this series, we should **track the motion** of each pixel from one frame to another.

Extracting the motion vectors between two consecutive frames can be achieved by calculating their optical flow. Optical flow is defined as the motion of objects between consecutive frames of a series, produced by the relative movement between the object and camera. By using computer vision algorithms which calculate the optical flow of two scenes, we could track the changes in glacier retreat across a time series dataset such that we can estimate their current velocity and predict their position in the next frames.

Figure 3 emphasizes the problem visually, where we can express an image as a function of space, with the coordinates (x, y), and time t. If we take the first image to be I(x, y, t) and we move its pixels by a distance of dx, dy over a timestamp dt, we obtain the new image as follows: I(x + dx, y + dy, t + dt).

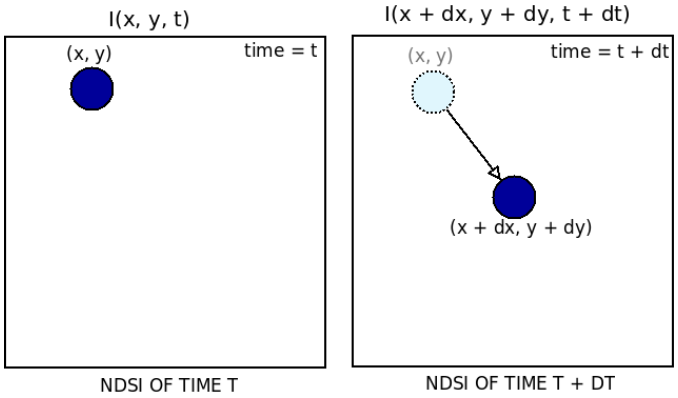

Fig. 3. Optical flow between NDSI($time = t$) and NDSI($time = t + dt$).

There are multiple types of optical flow generation algorithms, but for the purpose of our thesis, we have chosen a **dense optical flow** algorithm proposed by Farnebäck, Gunnar in [Farnebäck, 2003]. Even if dense implementations have a higher cost, we chose to make this trade mainly because it calculates the motion for each pixel of the frame while also having a higher accuracy compared to sparse methods such as in the one proposed in [Lucas and Kanade, 1981].

The best results have been generated by using this method of generating the optical flow with a pyramid scale of 0.5 and 6 pyramid levels, having in mind that the images that we work with are large. By choosing this combination of parameters, we state that at each level, the image is going to be reshaped at half the size of the previous one; therefore the area of search for motion is small enough such that the optical flow is able to track movement, with 3 iterations over each layer. The rest of the parameters have been been set at their typical values.

The generated results will be the computed motions for each pixel, treated as a pair representing the distance and direction that its coordinates moved from one frame to another. As referring to Figure 3, each item from the result represents the motion distance vector (dx, dy) as calculated between NDSI($time = t$) and NDSI($time = t + dt$). As an example, the generated optical flow vectors for scene LC81940282015363LGN02 can be seen in Figure 4. In order not to visually flood the image with vectors for each pixel, we are only drawing them from 30 to 30 pixels (900 to 900 meters).

C. Generating predicted images based on motion

Optical flow allows us to detect movement of the ice front, therefore we propose to generate NDSI images whose pixels will be predicted based on the past extracted motions for a frame. We can obtain the motion predicted NDSI image by

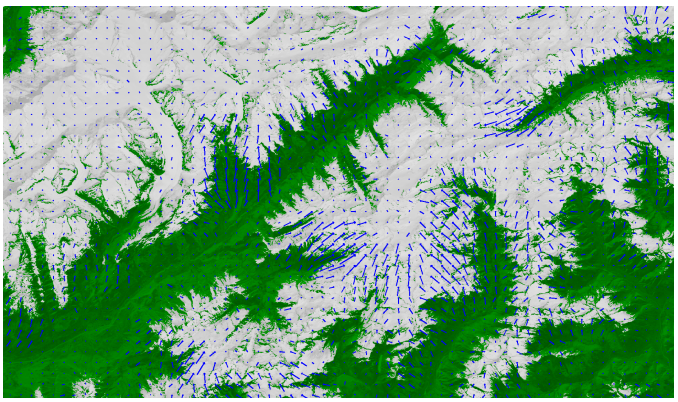

Fig. 4. Overlayed motion vectors generated by optical flow for scene LC81940282015363LGN02.

relocating each pixel value from the NDSI($time = t + dt$) to a location $(x + 2 * dx, y + 2 * dy)$, thus generating the new image NDSI($time = t + dt$) as described in Figure 5, $I(x + 2 * dx, y + 2 * dy, t + 2 * dt)$.

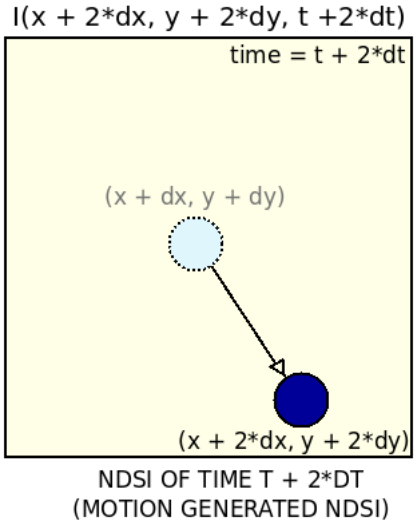

Fig. 5. Overlayed motion vectors generated by optical flow for scene LC81940282015363LGN02.

As a first approach of populating the new image, we have simply iterated over all the pixels in the NDSI($time = t + dt$) image and calculated their new coordinates based on their respective motion vector, as it can be seen in Algorithm 1. Since this is an iterative approach, it does not scale for images as large as the ones we are using. On top of this, since we are using the Python language, which is an interpreted one, each operation has to go through an interpreter before being processed. For very granular data processing this proves to be inefficient. For a typical image of 8543x8039 resolution, it took as much as 10 minutes for generating the image on the machine that we have used for processing (see SectionIII-E).

Since the data used for this thesis is large in size, relying on Python alone for data structures for processing does not scale. Python is an interpreted language, not a compiled one, which

Algorithm 1: Algorithm used for motion predicted image generation based on the optical flow vectors and NDSI($time = t + dt$)

1 function generate $(NDSI, motion\_vectors)$;
  Input : The NDSI($time = t + dt$ and the motion vectors generated by optical flow between NDSI($time = t$) and NDSI($time = t + dt$)
  Output: The motion predicted NDSI($time = t + 2 * dt$))
2 $motion\_predicted\_image \leftarrow NDSI.shape$;
3 $width \leftarrow NDSI.width()$;
4 $height \leftarrow NDSI.height()$;
5 **for** y *in* $[height, width]$ **do**
6 **for** x *in* $[height, width]$ **do**
7 $dx \leftarrow motion\_vectors[y][x][0]$;
8 $dy \leftarrow motion\_vectors[y][x][1]$;
9 $new\_x \leftarrow x + dx$;
10 $new\_y \leftarrow y + dy$;
11 $motion\_predicted\_image[new\_y][new\_x] \leftarrow NDSI[y][x]$;
12 **end**
13 **end**
14 **return** $motion\_predicted\_image$;

means that for each operation its interpreter has to do extra work such that it translated the bytecode instruction into a form that is machine executable. Since we have high resolution images, this does not scale. Instead, we resorted to using NumPy, which is an open-source library which brings support of large data computation, such as multi-dimensional arrays and matrices, as well as a large collection of mathematical functions which can be easily used for their operation [Harris et al., 2020]. The library has a **well optimised C code core** which can handle large processing by bypassing the python interpreter, which results in the speed of a compiled language, not an interpreted one. Therefore, based on the needs of our dataset, we have used NumPys' main data structure, the ndarray (n-dimensional array) as a data structure to hold our images and various optimisations applied throughout processing focused on using only this type of data structure in order to make use of NumPys' optimisations and speed. By doing this instead of simply iterating, we were able to improve the processing time by around 1700%, bringing it down approximately from 10 minutes to 35 seconds.

In the iterative approach, for each pixel we add its motion to its position such that we get its new location. These numbers can be computed ahead of time and transformed into arrays such that we only use optimised operations. By creating an **array of the initial coordinates** (x, y) and **adding the motion vectors** (dx, dy) array to it, we **generated the absolute coordinates** where each pixel from the NDSI image should be translated. By adding this modification we got rid of lines 5, 6, 7, 8, 9, 10 and 11 from the algorithm and replaced them with the code as it can be seen in Algorithm 2. The

$absolute\_coordinates$ and $NDSI$ can be treated as a sparse array which is then densified using numpy capabilities.

Algorithm 2: Improved algorithm used for motion predicted image generation based on the optical flow vectors and NDSI($time = t + dt$)

function generate $(NDSI, motion\_vectors)$;
Input : The NDSI($time = t + dt$ and the motion vectors generated by optical flow between NDSI($time = t$) and NDSI($time = t + dt$)
Output: The motion predicted NDSI($time = t + 2 * dt$))
$motion\_predicted\_image \leftarrow NDSI.shape$;
$width \leftarrow NDSI.width()$;
$height \leftarrow NDSI.height()$;
$index\_array \leftarrow [[[0,0],[1,0]...[width,0]$
$[0,1],[1,1]...[width,1]$ ...
$[0,height],[1,height]...[width,height]]]$;
$absolute\_coordinates \leftarrow$
$motion\_vectors + index\_array$;
$motion\_predicted\_image[absolute\_coordinates] \leftarrow$
$NDSI$;
return $motion\_predicted\_image$;

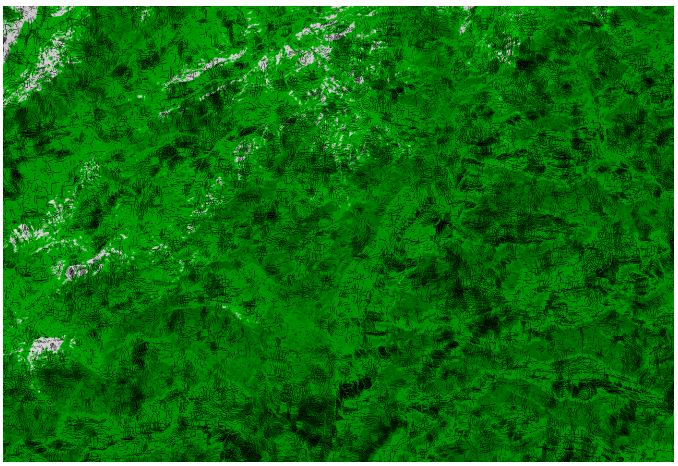

Fig. 6. Raw generated NDSI image, unfiltered, zoomed in for a clear view.

The function used to generate the predicted NDSI does not have a **one-to-one domain**, thus making it undefined for certain inputs. This results in a very **noisy** generated image, as it can be seen in Figure 6. We have implemented a **filter** which uses the weighted average of the neighbouring pixels values to fill the missing ones.

D. Generated image filtering

A first step into creating the filter for the motion predicted NDSI image is to create a **mask** which will be applied on the black border of the scene such that we are looking for **undefined values** only inside it. Using numpy, the **coordinates** of these pixels are extracted into an array which then can be processed in parallel by Pythons' **multiprocessing** library. We have used multiprocessing instead of threading because Python does not have true parallelization in multithreading, only concurrency, which would not be a real improvement on the processing time. Since all the processes have to write different chunks of the same image and they do not share the same memory space, we chose to solve this problem by creating a shared memory buffer to hold the image.

The value of each found undefined pixel has to be calculated as an **weighted average** composed of its **neighbouring pixels**, as shown in Figure 7 top. We created the **kernel** which holds the weight of each pixel in the neighbourhood such that pixels closer to the centre have a higher weights than those near the edge. However, we have the case when we have multiple undefined pixels in the same neighbourhood. Since we cannot take their values in consideration, we do not want to include them in the weighted average, thus we set their weight to be 0, as shown in Figure 7 left, bottom. The result of the weighted average will be then stored as the value of the currently focused undefined pixel, as highlighted in Figure 7 right, bottom.

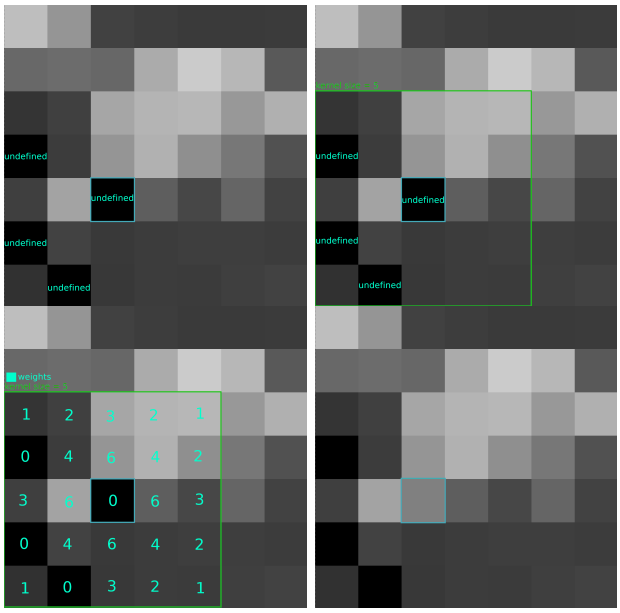

Fig. 7. Zoomed in part of the predicted NDSI (left, top). Neighbourhood extracted (right, top). Kernel of weights (left, bottom). Generated value (right, bottom).

III. Experiments

We have picked two glaciers and downloaded all their public available satellite imagery with at most 20% cloud coverage. We have split our dataset based on $path, row, month$ pairs, as suggested in [Racoviteanu et al., 2019], due to different seasonal snow coverage fluctuations. In order to better visualise the differences between the two images, they have been overlapped and colourized such that orange areas of snow that are predicted to melt, while cyan represents areas which are predicted to develop snow build up.

The first glacier region which will be analysed is named **Jungfrau-Aletsch-Bietschhorn**, located in the **Swiss Alps**, specifically at latitude 46.47735081308319 and longitude 8.056887228860798, with an elevation which varies between 809 and 4,274 m. This location is at the intersection of multiple WRS-2 coordinates, therefore we will have different sets of $path, row, month$ which we will be focusing on.

A. Jungfrau-Aletsch-Bietschhorn (194, 28, 4)

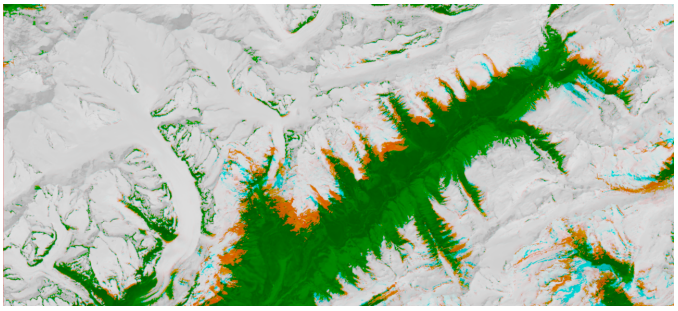

Fig. 8. Overlapped motion generated NDSI for scene LC81940282017112LGN00.

The motion generated image for this path and row can be seen in Figure 8, while and Figure 9 holds the snow coverage values extracted from each image in the dataset. This scene was captured in April, when the temperatures are still low and the snow fall fluctuates a lot, resulting in a low signal to noise ratio. This pattern proved to yield inconsistent results for the generated images.

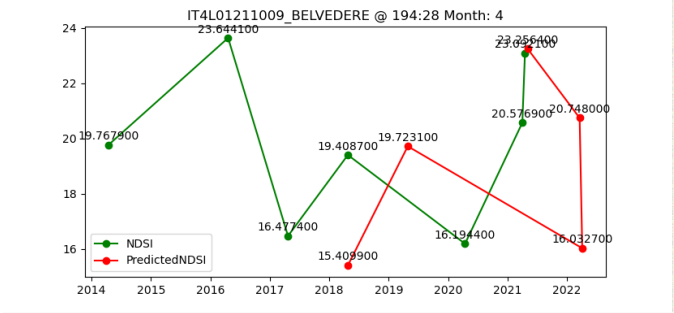

Fig. 9. Comparison between the NDSI values of the actual NDSI images for Jungfrau-Aletsch-Bietschhorn(194, 28, 4), and the NDSI of each predicted motion image.

B. Jungfrau-Aletsch-Bietschhorn (194, 28, 7)

Figure 10 represents the motion predicted NDSI image afferent to this path, row and month pair. As it can be seen in Figure 11, the scene was captured in July, hence the more consistent snow fall. These results yield a better estimation of movement than the one obtained in early spring.

C. Jungfrau-Aletsch-Bietschhorn (194, 28, 8)

The result of the experiment is captured in Figure 12, while its graph of snow coverage can be located at Figure 13. The

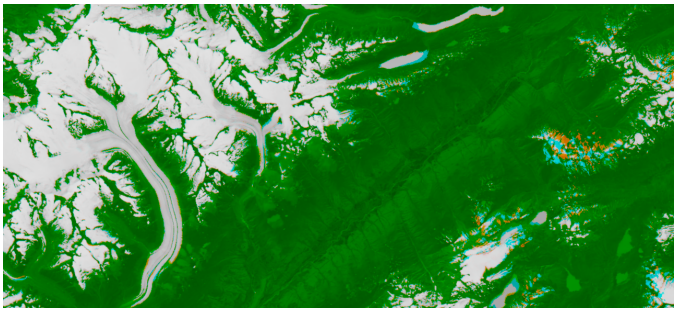

Fig. 10. Overlapped motion generated NDSI for scene LC81940282020201.

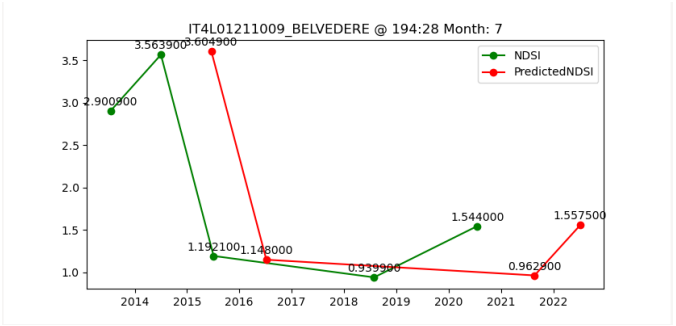

Fig. 11. Comparison between the NDSI values of the actual NDSI images for Jungfrau-Aletsch-Bietschhorn(194, 28, 7), and the NDSI of each predicted motion image.

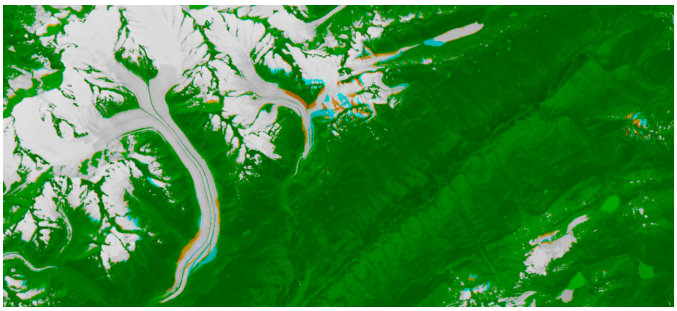

Fig. 12. Overlapped motion generated NDSI for scene LC81940282016238LGN01.

image was captured during August, which in this case has the least amount of snow fall. The result for this data set was the most reliable, as one can observe since the moraines are clearly exposed.

The second glacier which we have taken into consideration is named **Parvati** and it is located in **India**, at latitude 31.754 and longitude 77.675, with a maximum elevation of 5599 meters. We have chosen this glacier such that we can compare our results with the ones provided in [Tak and Keshari, 2020]. They have chosen a dataset of images located at WRS-2 path 147, row 38. However, we could not fetch the same images for that specific path and row in order to be able to make this comparison. The result that were achieved are however listed below for the available path and row.

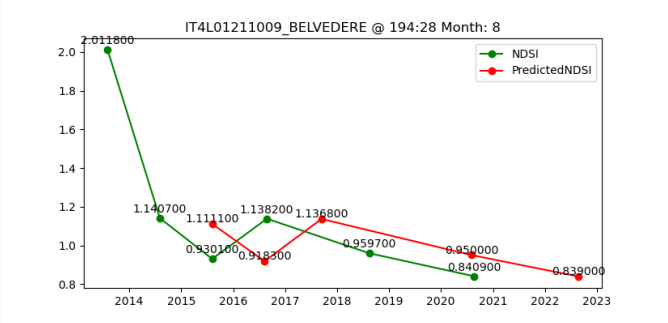

Fig. 13. Comparison between the NDSI values of the actual NDSI images for Jungfrau-Aletsch-Bietschhorn(194, 28, 7), and the NDSI of each predicted motion image.

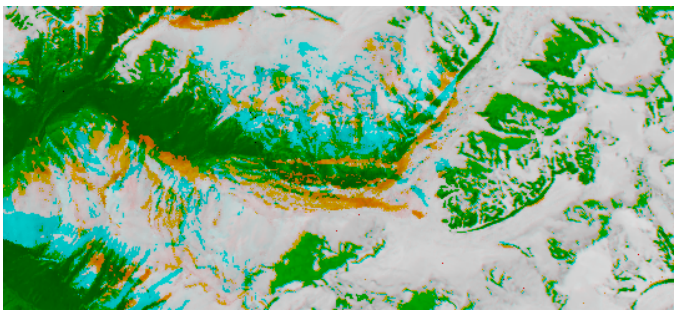

Fig. 14. Overlapped motion generated NDSI for scene LC81460382021091.

D. PARVATI (146, 38, 4)

Here we can see again that the high snow fluctuation yields in big differences between the predicted image and the actual one, as it can be observed in Figure 14. Given that the Parvati region is located at a higher altitude than the Jungfrau-Aletsch-Bietschhorn one, its snowfall is higher. This can also be seed while analysing the snow coverage values for the values from Figure 9 and Figure 15, where the maximum snow fall for Parvati is around 57% (2014) and the one for Jungfrau-Aletsch-Bietschhorn is around 23% (2016).

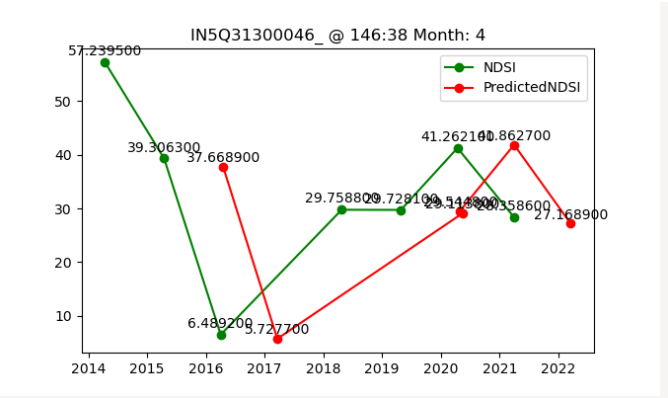

Fig. 15. Comparison between the NDSI values of the actual NDSI images for Parvati(146, 38, 4), and the NDSI of each predicted motion image.

E. Performance

For the performance evaluation we are using a machine which has an Intel i7-7700HQ CPU, with 24 GB of RAM. The images were stored on my personal server and they were locally mounted through NFS.

Downloading the dataset of two glaciers (IN5Q31300046, IT4L01211009) which have been used for the experiment took around two hours on my machine, while their querying took around half an hour. For IN5Q31300046 (Parvati), there were 185 entries, while for IT4L01211009 (Jungfrau-Aletsch-Bietschhorn), there were 109. The processing and caching of all the datasets took around 8 hours, which outputs around 1 minute and 30 seconds per scene.

IV. Conclusions and Future Work

The best and consistent results were obtained for scenes which were acquired between April and September, due to lower snow fall and better glacier exposure. We have found that scenes which are taken during the winter months are prone to erroneous results due to higher snow coverage which creates high fluctuations and outliers. In [Tak and Keshari, 2020], only September was taken into consideration when conducting the experiments, mainly because it is the month which has the lowest recorded snow coverage of the area. Also, in In [Racoviteanu et al., 2019], the experiments were performed on seasonal organized data and the scenes with higher snow coverage proved problematic as well. Another factor which influenced the results is cloud coverage, since even if the shortwave infrared band filters out most of the clouds found in a scene, still there are some cases when the coverage is very dense and the land underneath cannot be analysed. Since the terrain is not visible, the glacier pixels cannot be properly extracted, resulting in erroneous snow coverage and distorted motion generated images at the locations where the clouds have appeared or disappeared.

During the process of creating the application, multiple problems were met. One of them was that even though the number of available aerial images collected in the Landsat 8 archive are high, most of them have very high cloud coverages. Filtering out results which have lower percentages create datasets with low number of entities, which means that we cannot test our results on longer time series. Since clouds interfere both with extracting glacier pixels from a scene and analysing motion, two approaches on solving this problem would be:

- harvesting images provided by **multiple satellites**, such as Sentinel and older versions of Landsat, as proposed in [Winsvold et al., 2018]. Even if these satellites use other types of sensors for Earth observation, by matching the wavelengths of their bands one could organize and pair their datasets, resulting in time series with more entities and more scenes which have lower cloud coverage. By only using data provided by the Landsat 8 satellite, even with an allowed cloud coverage of 20% as we have used for our experiments, the average length of a $path, row, month$) dataset was around 15;

- implementing **cloud mapping algorithms** such that clouds could be detected and trimmed out before the calculation of the normalized snow difference index and optical flow extraction. For now, the optical flow algorithm ran between scenes which have a high cloud coverage finds large vectors of motion at the locations where clouds existed from one frame to another. The result does not take into consideration the difference between snow pixels and cloud ones.

Another improvement could be changing the way that the coordinates of the pixels of the motion generated image are calculated. An interesting approach would be using **time series forecasting models** applied on the distance vectors generated by optical flow for each pixel over time in order to generate future entries. This could be done with statistical methods such as the autoregressive integrated moving average (ARIMA). However, doing this forecast over each pixel of a high resolution image would take a lot of processing power; also, a larger $path, row, month$ dataset would be needed for such an analysis.

One of the main slow downs in developing and using the application was the slow time of processing due to the large data files. A machine which has a more powerful processing unit would yield much faster results. This could be done by **migrating** the processing unit to **cloud** and hiring a machine with a CPU which has a high number of cores and better performance. One of the possible downsides would then be the amount of time needed for passing the large scene files through he network to the machine on cloud, which is easily dependable on the bandwidth. A solution to this downside would be passing the dataset before starting the processing and make sure that they are on the same machine. However, cloud storage can get very expensive and the trade off between processing time and expenses could prove too unbalanced.

As for a final future development idea, migrating the codebase from Python to a language which supports multithreading as well as compiling to machine language, such as C++, would yield in much faster results.

REFERENCES

[Farnebäck, 2003] Farnebäck, G. (2003). Two-frame motion estimation based on polynomial expansion. In Bigun, J. and Gustavsson, T., editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

[Harris et al., 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with numpy. *Nature*, 585(7825):357–362.

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Racoviteanu et al., 2019] Racoviteanu, A. E., Rittger, K., and Armstrong, R. (2019). An automated approach for estimating snowline altitudes in the Karakoram and eastern Himalaya from remote sensing. *Frontiers in Earth Science*, 7:220.

[Tak and Keshari, 2020] Tak, S. and Keshari, A. K. (2020). Investigating mass balance of parvati glacier in himalaya using satellite imagery based model. *Scientific Reports*, 10(1):12211.

[Wang et al., 2017] Wang, H., Yang, R., LI, X., and CAO, S. (2017). Glacier parameter extraction using landsat 8 images in the eastern karakorum. *IOP Conference Series: Earth and Environmental Science*, 57:012004.

[Winsvold et al., 2018] Winsvold, S. H., Kääb, A., Nuth, C., Andreassen, L. M., van Pelt, W. J. J., and Schellenberger, T. (2018). Using sar satellite data time series for regional glacier mapping. *The Cryosphere*, 12(3):867–890.

[Winsvold et al., 2016] Winsvold, S. H., Kääb, A., and Nuth, C. (2016). Regional glacier mapping using optical satellite data time series. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 9(8):3698–3711.

[Yan et al., 2015] Yan, H., Zhou, G., and Lu, X. (2015). Comparative analysis of surface soil moisture retrieval using VSWI and TVDI in karst areas. In Zhou, G. and Kang, C., editors, *International Conference on Intelligent Earth Observing and Applications 2015*, volume 9808, pages 37 – 49. International Society for Optics and Photonics, SPIE.