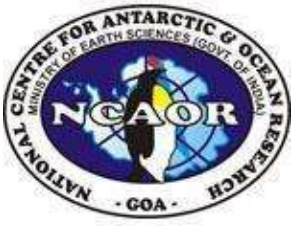
A REPORT ON

# Monitoring the change in volume of the Gangotri Glacier through satellite imagery
# and
# Estimation of Subglacial topography and Glacier Volume using Artificial Neural Networks (ANNs)




BY

Hitesh Vinod Bhagchandani 2015A7PS023H

At

**National Centre for Antarctic and Ocean Research** (**NCAOR**), **Goa.**

A Practice School-I Station of

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

**(JULY 2017)**

A REPORT

ON

# Monitoring the change in volume of the Gangotri Glacier through satellite imagery and Estimation of Subglacial topography and Glacier Volume using Artificial Neural Networks (ANNs)


BY

Hitesh Vinod Bhagchadani 2015A7PS023H B.E. (Hons.) Computer Science


Prepared in partial fulfillment of the

Practice School –I Course No.

BITS F221

At

**National Centre for Antarctic and Ocean Research** (**NCAOR**), **Goa.**

A Practice School-I Station of

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**
**(JULY 2017)**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE**
**PILANI (RAJASTHAN)**
**Practice School Division**

**Station:** National Centre for Antarctic and Ocean Research (NCAOR) **Centre:** Goa

**Duration:** 55 days **Date of Start**: 22nd MAY, 2017
**Date of Submission:** 12th JULY, 2017

**Title of the Project:** Monitoring the change in volume of the Gangotri Glacier through satellite imagery and estimation of subglacial topography using Artificial Neural Networks(ANNs).

**ID No./Name/ :** Hitesh Vinod Bhagchandani (2015A7PS023H)

**Discipline of The student :** B.E. (Hons.) Computer Science

**Name and Designation Of the expert :** Mr. Sakthivel Samy V, Head of Information Communication Technology Division (ICTD)

**Name of the PS Faculty :** Dr. Amit Setia

**Key Words :** Artificial Neural Networks (ANNs), DEM-images, machine learning, image processing, Python 3.5, glaciers, remote-sensing

**Project Area :** Remote Sensing, Image Processing , Machine Learning

**Abstract:** Monitor glaciers using satellite images and infer changes in their properties by using relevant theories. A major part of the first project deals with image processing and its subsequent implementation in OpenCV and MATLAB. An empirical cubic polynomial is the result of fitting the data. The second project aims to estimate the subglacial topography of glaciated mountains by employing Artificial Neural Networks (ANNs) on geographical data provided by DEM images and shapefiles of the glacier under consideration. Though ANNs are not the best technique for doing the same, this venture would surely clarify some of the intricacies involved in implementation of neural networks in Python 3.5.

Signature of Student
Date:

Signature of PS faculty
Date:

# Contents

# Acknowledgements

This venture into image-processing and subsequent analysis of satellite images to get valuable information about glaciers wouldn't be possible without the mentorship of Mr. Sakthivel Samy.

We express our wholehearted thanks to Dr. M. Ravichandran, Director, National Centre for Antarctic and Ocean Research (NCAOR), Goa, for allowing us to undertake this project in this esteemed organization. We are also thankful to Dr. K.R. Anupama, Associate Dean Practice School for making our stay in Goa campus comfortable.

Needless to say, our efforts wouldn't have borne this fruit if it weren't for the comfortable and stimulating environment provided by Dr. Swati Alok, Dr. Rahul Mohan and Dr. Amit Setia.

# PROJECT-1

## Introduction

### Motivation

Monitoring of glaciers is primarily done through satellite images collected over regular intervals of time. Due to advancements in satellite imagery and image processing, today, more information can be inferred from images than was hitherto possible. The current project was motivated, primarily, by the alarming need to monitor the changes taking place in the cryosphere and correlate the same to global climate change.

### Problem Statement:

Calculating the change in volume of the Gangotri Glacier over a period of approximately 1 year ie- from 15$^{th}$ Oct, 1999 to 2$^{nd}$ Nov, 2000[7].

### Brief Overview:

After gaining access to the USGS(United States Geological Survey) repository of LANDSAT earth images and ASTER-DEM images, those pertaining to the Gangotri Glacier (30°N,79°E) are acquired. An appropriate level of pre-processing is done to make them suitable to carry out further work.

A great part of the project deals with assessing the various image segmentation algorithms and choosing the one with the best performance.An efficient segmentation algorithm is subsequently employed to retrieve the regions representing glaciers. The DEM images are read to get the elevation of every pixel and the change in elevation across the images is calculated. This change coupled with the change in area gives us the desired change in volume.
The change in the area is obtained by comparing ice cover across 2 images and recording the difference.

## Acquiring Of Data

### Acquisition of Satellite Images:

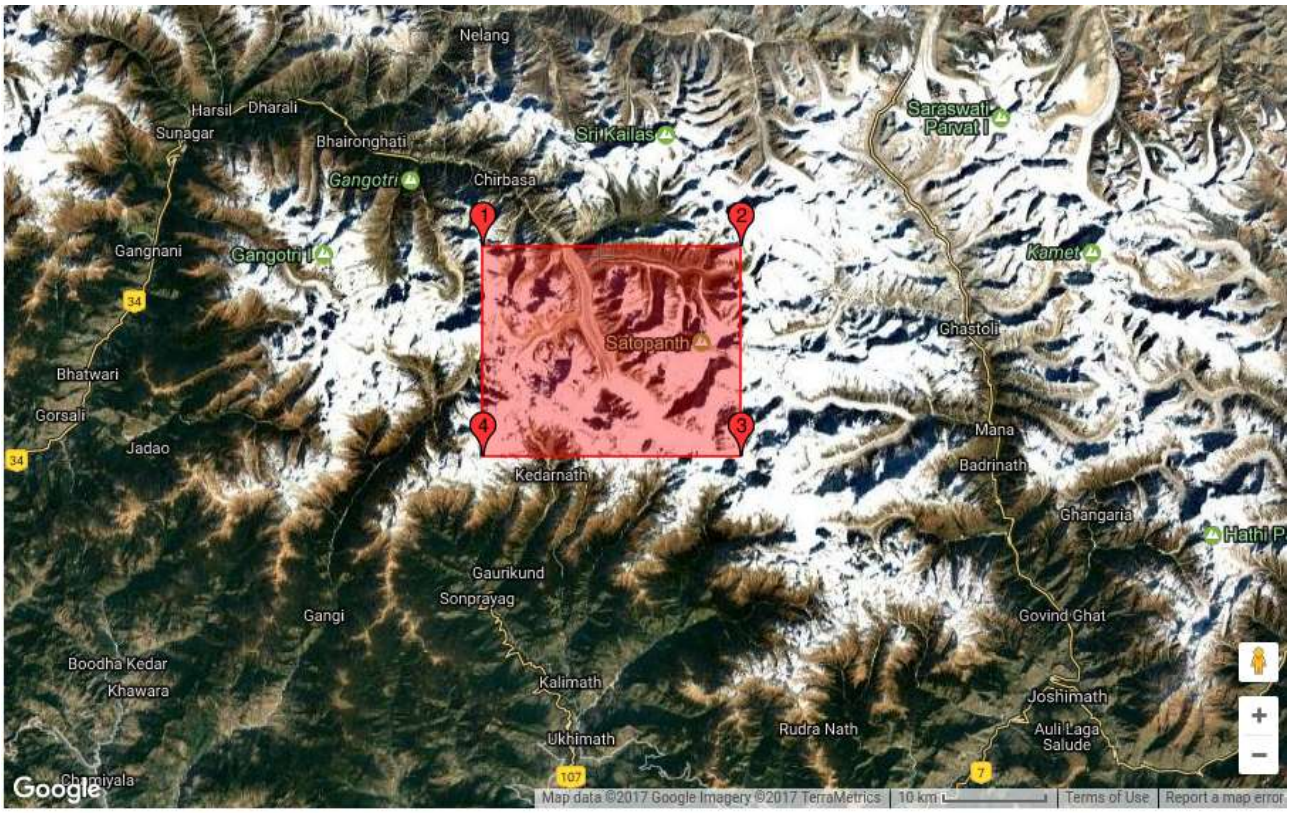
The USGS website provides authorised users access to a vast repository of images collected by a variety of satellites like commercial satellites, ISRO Resourcesat, Landsat, ASTER (Advanced Spaceborne Thermal Emission and Reflection Radiometer) etc.

This project used images from ASTER and Landsat[7]. ASTER and Landsat images are used, because their footprint means that a regional view of the ice is provided, but their relatively

fine resolution means that even small glacier structures, such as crevasses and melt ponds, can be imaged and mapped. The long time series of images from LANDSAT and ASTER satellites is useful for mapping changes over time.

Aster DEM images in the Geotiff format is grid spanning an area of 111 km X 111 km with cell size of 30 m X 30 m. Each cell contains average elevation value of all points within cell.

ASTER DEM images of Gangotri glaciers were obtained after selecting the polygon formed by the coordinates-(30° 55' 30" N,79° 0' 0" E),( 30° 55' 30" N,79° 15' 0" E), (30° 45' 0" N,79° 15' 0" E) and (30° 45' 0" N,79° 0' 0" E).



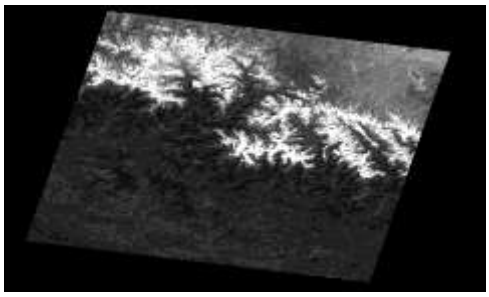



Fig 1.1 - a: 15-Oct-1999

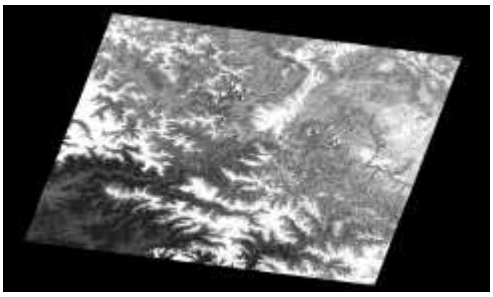


Fig 1.1 - b: 8-Oct-2000

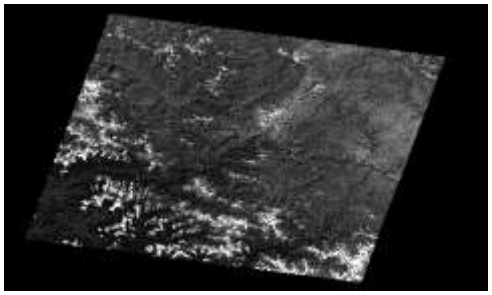


Fig 1.1 - c: 2-Jun-2000

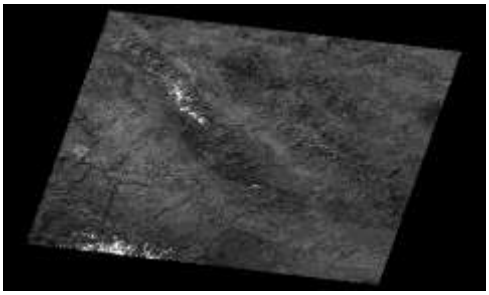


Fig 1.1 - d: 02-Nov-2000

Fig 1.1 : Landsat Images with dates of procurement.

**Pre-Processing**

A few of the blurry images were subjected to Gaussian Blur which is a low-pass filter, for attenuation of high frequency signals.

The images were resized to 500 X 500 pixels to maintain uniformity of image segmentation results and ensure the proper working of the algorithms.

## Image Segmentation

Image Segmentation is the process of partitioning a digital image into multiple segments (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyse. Given below are the various techniques that were explored in the process.

**Simple Thresholding (Fig 2.1):**

Here, the matter is straight forward. If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black). The results obtained were:

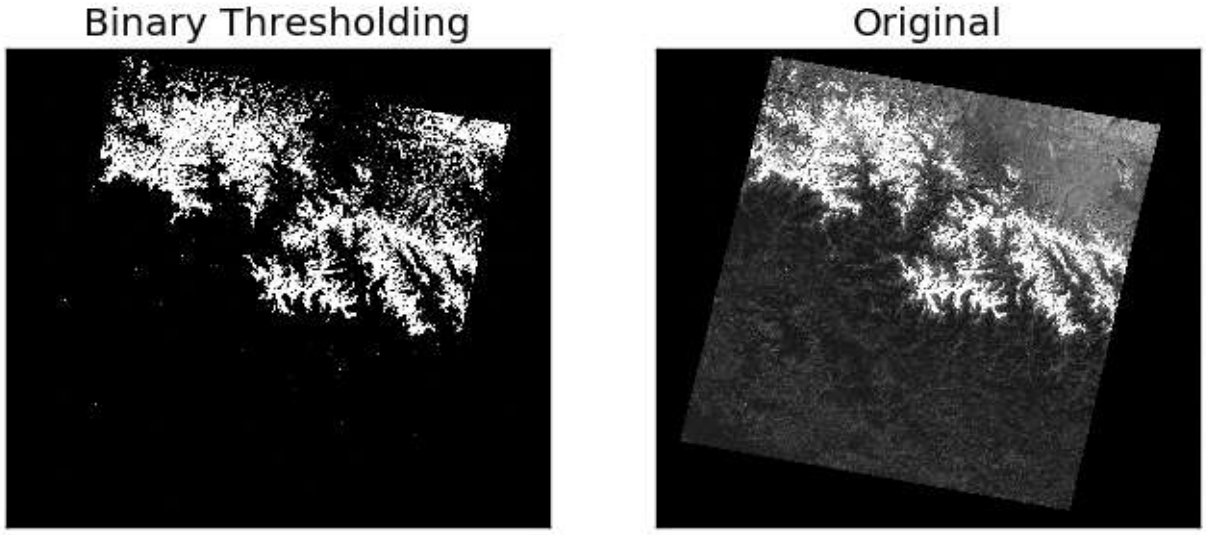



Fig 2.1: Binary Thresholding (Normal)

The above result was obtained by setting pixels that were higher than a threshold value to 255 and the rest to 0.

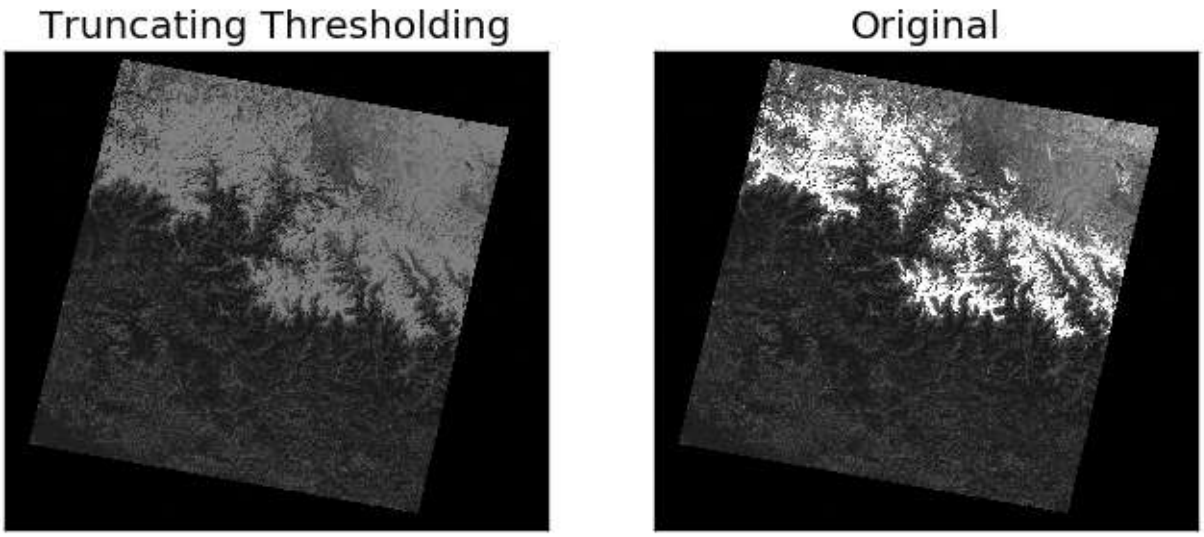



Fig 2.1: Binary Thresholding (Truncating)

The above image was obtained by leaving the pixel intensities as they are if the source pixel is not greater than the supplied threshold.

**Adaptive Thresholding:**

In the previous section, we used a global value as threshold value. But it may not be good in all the conditions where image has different lighting conditions in different areas. In that case, we go for adaptive thresholding. In this, the algorithm calculates the threshold for a small region of the image. So we get different thresholds for different regions of the same image and it gives us better results for images with varying illumination.

**Otsu's Thresholding (Fig 2.2):**

Otsu's method is used to automatically perform clustering-based image thresholding or the reduction of a graylevel image to a binary image.

The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal.

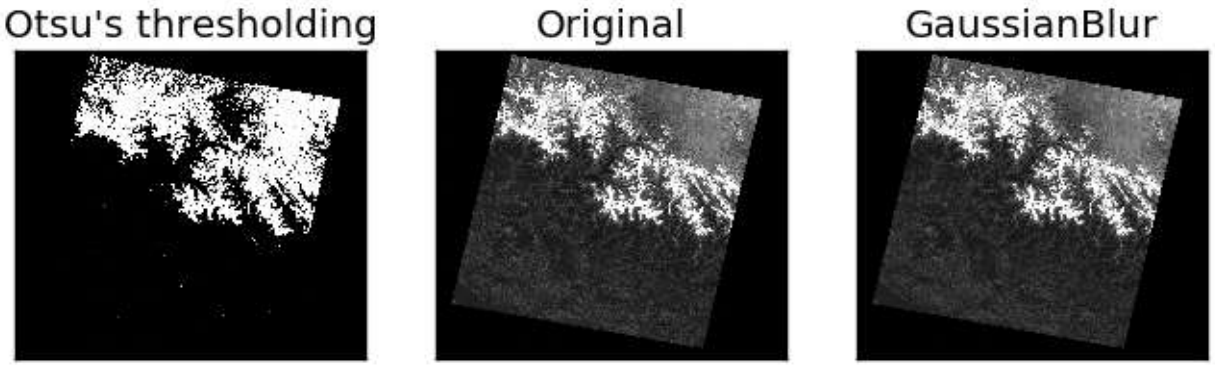


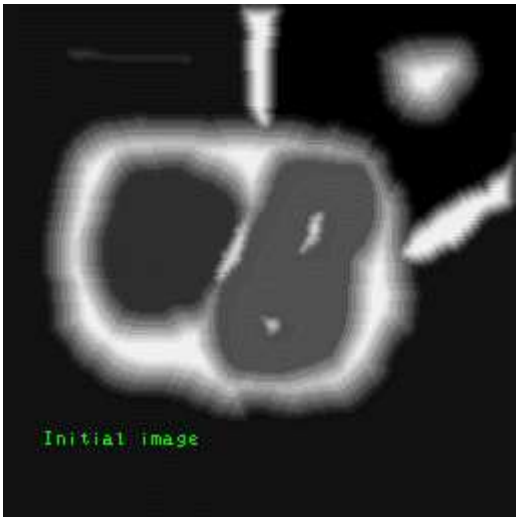

Fig 2.2 : Otsu's Thresholding algorithm

**Marker-based Watershed Algorithm (Fig 2.3):**

Any grayscale image can be viewed as a topographic surface where high intensity denotes peaks and hills while low intensity denotes valleys. You start filling every isolated valley (local minima) with different coloured water (labels). As the water rises, depending on the peaks (gradients) nearby, water from different valleys, obviously with different colours will start to merge. To avoid that, you build barriers in the locations where water merges.

You continue the work of filling water and building barriers until all the peaks are under water. Then the barriers you created give you the segmentation result. This is the "philosophy" behind the watershed.

Fig 2.3: Marker-based Watershed algorithm




**Gaussian Mixure Model followed by Expextation-Maximisation Algorithm (Fig 2.4):**

The pixels of the image are seen as clusters of independent Gaussian distributions. So, for each point we will have a probability that it belongs to each of these distributions/clusters. This technique is different from the usual k-means algorithm in the following ways (non-exhaustive):

1) The Gaussian Mixture Models approach will take cluster covariance into account when forming the clusters.

K-means only considers the Euclidean Distance and assigns a point to a cluster only if the point is closer to that cluster's representative than any other cluster's representative.

2) Standard k-means performs a hard assignment of data points to clusters–each point is assigned to the closest cluster.

With Gaussian Mixture Models, what we will end up is a collection of independent Gaussian distributions, and so for each data point, we will have a probability that it belongs to each of these distributions / clusters. Refer to A3 for the underlying mathematics.


Fig 2.4: Results of applying the Gaussian Mixture of Means model (GMM)

**Hidden Markov Field Model:**

The gist of this model is based on the seminal paper written by L.R. Rabiner[5].

We look at the picture as a set of nodes, where each pixel is node and is connected to its neighbours by edges and has a label - this can be called a Markov Random Field[5]. MRFs can be solved, i.e. give an optimal labelling for each node and thus an optimal labelling, in a number of ways, one of which being graph cuts based on maximal flow.

Refer to [6,8] for a detailed analysis of the theory and its applications to medical sciences.

Refer to A6 for a possible implementation in MATLAB.

**Conclusion:**

We cannot say that one method is better for all the images, because the results depend on the type, the resolution and even the content of the images. However we can say that some algorithms are generally better than the others in remote sensing. A study has been carried out to evaluate different segmentation algorithms. Some of them used clustering (K-means) and some others region merging (watershed) algorithms, and the best software of this comparison was using watershed.

It turns out that out of the 3 contending algorithms: watershed, Markov-model based segmentation and Gaussian Mixture Models, the Hidden Markov Random Fields gives the best results as far as remote sensing is concerned. This is primarily because of the contextual information provided by such models.

## Results of Segmentation

The results of applying the Gaussian Random Field model are given in the figure below:

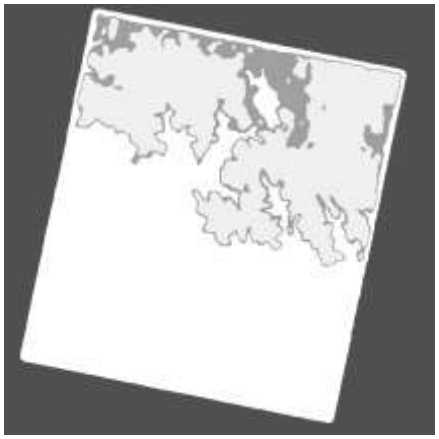


Fig 3.1 a. 15th Oct, 1999

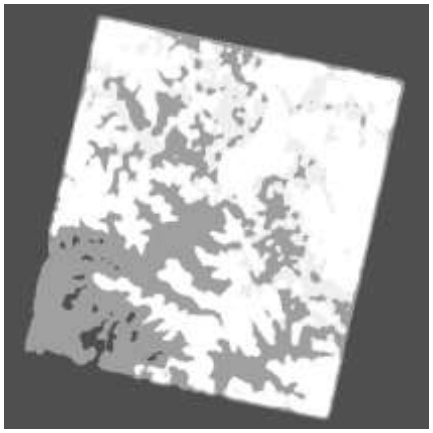


Fig 3.1 b. 8th Oct, 2000

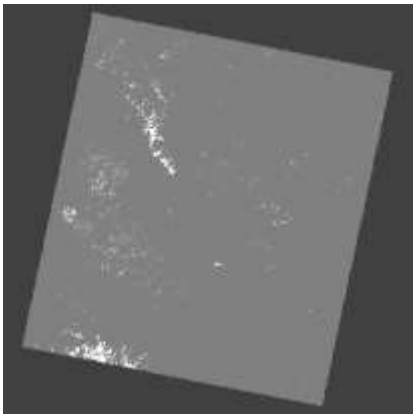


Fig 3.1 c. 2nd Nov, 2000

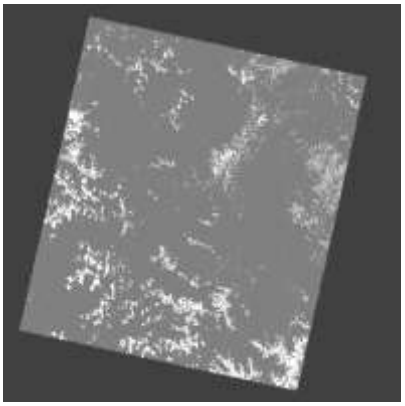
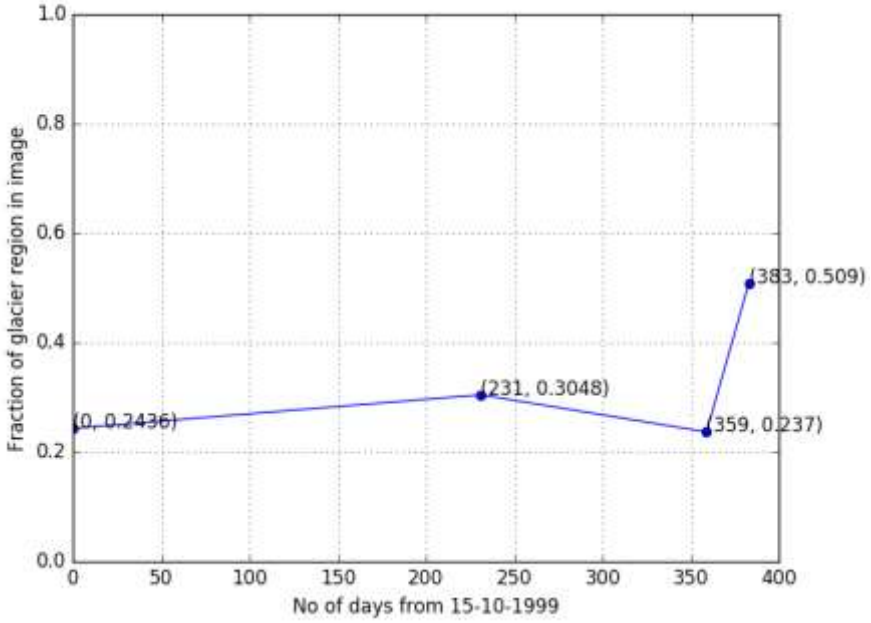


Fig 3.1 d. 2nd June, 2000

Fig 3.1: Results of segmentation

Plot of the fraction of ice in the images obtained-



# Calculation of change in volume of the glacier

The segmented images are subjected to thresholding to set all values to either 0 or to 255. The results are (only one is shown) shown in fig. 3.3.

Fig 3.3 : Result of performing binary thresholding on one off the segmented images to get fraction of ice.

Considering 500 X 500 sub grid , we calculate the change in elevation of the glacier and this is multiplied with the cell area to give the final change in volume of the glacier.

Refer to A4 for the implementation of the above in MATLAB.

## Preliminary Results

The plot of the fraction of ice in the images against the number of days elapsed can be satisfactorily fit with a cubic polynomial of the form (refer to A5 for the Python implementation):

$$y = ax^3 + bx^2 + cx + d$$

a = 0.000000209555151 /day^3
b= -0.000125850972709/day^2
c= 0.018154437321913/day
d=0.243600000000000

This can be used to predict the fraction of ice in the images obtained at some other point in the future.

## Estimation of approximate geographical area and change in the volume of glacier from 15-Oct-1999 to 8-Oct-2000.

*Approximate geographical area = (Fraction of glacier region in the image) * (499*30)*(499*30) meters^2 [spatial resolution of the 500x500 image considered is 30 meters].

*Geographical Area Calculation:

15-Oct-1999 : 0.2436*14970*14970=54590979.24 m2=54.590 km2.

02-Jun-2000 :0.3048*14970*14970=68305954.32 m2=68.305 km2.

08-Oct-2000 :0.2370*14970*14970=53111913.30 m2=53.111 km2.

02-Nov-2000 :0.5090*14970*14970=114067358.1 m2=114.06 km2.

**To find the change in the volume of glacier from 15-Oct-1999 to 8-Oct-2000:**

*Change in volume= 2.433 km^3

*Change in mass of glacier from 15-Oct-1999 to 08-Oct-2000
= density of ice * ΔV
= 916.7 x 10^9*2.433
= 2230.33 x 10^9 kg.

## An Alternate Approach:

Glaciers flow because permanent deformation occurs as a result of strain in response to stress. Creep is the deformation of ice crystals. Movement can occur between or within ice crystals (Cuffey & Paterson 2010)[10]. The relationship between creep and stress can be given by Glen's Flow Law:

$\varepsilon = A\tau^n$.....(1)

Where ε = strain rate, A and n are constants, and τ is the basal shear stress. The constant n = 3, and the value of A is dependent on ice temperature, crystal orientation, debris content and other factors. Glacier ice may form beautiful folds or structures in response to creep.

In (1),
$\tau = \rho g h \sin\alpha$.....(2)

(where τ = driving stress, and
α = ice surface slope in radians)

The disadvantage of this method is that Mass-balance distribution data over large glaciers in the Himalaya are inaccurate and not easily available in some cases.

# PROJECT-2

# Introduction

## Motivation

There has been a substantial increase in the contribution of glaciers and ice sheets to the global sea level[1]. Despite such compelling reasons to be interested in the volumes of glaciers and ice caps, only a few hundred of the more than $10^5$ glaciers and ice caps that exist today have been geophysically mapped.

To predict the rate and consequences of shrinkage of the earth's mountain glaciers and ice caps, it is necessary to have improved regional-scale models of mountain glaciation and better knowledge of the subglacial topography upon which these models must operate. Several physics-based approaches exist, which allow ice thickness to be estimated but are subject to error if their underlying assumptions are not fulfilled.

This was the impetus behind venturing into Machine Learning to find efficient methods to estimate the subglacial topography of mountains.

## Problem Statement

For a DEM cell centred at some map position (i, j) with surface elevation $S_{ij}$, estimate the ice thickness $H_{ij}$ or, equivalently, the bed surface elevation $B_{ij} = S_{ij} - H_{ij}$.

## Prerequisites

### i) Artificial Neural Networks (ANN)

These are black boxes which take multidimensional inputs and give the desired output after performing several matrix operations like multiplication and transpose. They can be used in both classification and regression settings where the class of the input must be predicted or where the output is a continuous variable must be estimated. The basic architecture of Neural Networks (NN) consists of 'layers' which are made of 'nodes'. NNs are designed to emulate the working of neurons in the human brain (fig 1.1).

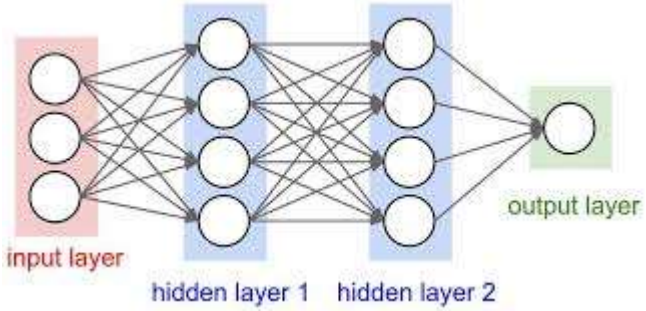



Fig 1.1 : Basic structure of ANNs.

Each layer(see A1 for implementation) consists of the following entities:

1) W : the weight matrix

2) Z : the output of the layer
3) S : the input to the layer
4) $F_p$ : the derivative of the input to the layer
5) D : the error matrix for the layer.

The crux of the architecture lies in its ability to transform input by using nonlinear functions like f(x) = tanh(x), f(x) = sigmoid(x), f(x) = max(x,0). A1 further elucidates the Python implementation of these functions.

In an intriguing series of recent papers, ANNs have been trained to emulate the fluctuations in glacier length that are associated with changes in air temperature and precipitation. This approach has allowed historical variations in glacier mass balance to be constructed (Steiner et al. 2005)[2], past length variations to be comprehended (Steiner et al. 2008), and future changes to be predicted (Zumbuhl et al. 2008)—all without reference to a physical ice dynamics model.

In the present setting, ANNs are used to statistically estimate the thickness of ice by extracting geometric information from DEM (Digital Elevation Model) images.

**ii) Geometric information about glaciers.**

Two sources of information about glaciers were used:
1) USGS (United States Geological Survey) repository available for download at [https://earthexplorer.usgs.gov/].
2) GLIMS-RGI (Global Land Ice Measurements from Space – Randolph Glacier Inventory) available for download at [https://www.glims.org/RGI/].

ASTER (Advanced Space-borne Thermal Emission and Reflection Radiometer) DEM images were obtained from source 1. Aster DEM images in the Geotiff format is a grid spanning an area of 111 km X 111 km with cell size of 30 m X 30 m. Each cell contains average elevation value of all points within cell.

The RGI repository provides us with the shapefiles of the glacier chosen in the form of a zip file containing the .shp file required to render the glacier.

In this project, the glaciers in the Western USA-Canada region were targeted. In particular, the coordinates of the glacier were: 51.3333°N, 125.33°W. The shapefile of the glacier rendered in PNG format is shown in Fig 1.2.

Fig 1.2: PNG image of the glacier outline.

The geometric premise upon which the model was built includes the following key points:

1) Within a particular geographical region there is a sameness to the landscape that is a consequence of the sameness of the bedrock geology, geological and environmental history, and present conditions for that region.

2) The deglaciated portions of landscapes that today are partially ice covered have geometrical similarities to portions that are glaciated; for the most part, the areas that are now ice denuded were formerly ice covered and therefore subject to similar landscape-shaping processes that currently operate on the ice-covered landscape.

3) Because the geological and environmental settings are spatially varying, it follows that a neural network that has been trained to estimate ice thickness in a particular geographical region may not perform well if applied to another region.

## Architecture of the ANN

### Input to the model

The various inputs fed to the network include:

1) The surface elevation of the ice-covered cell.
2) The orientation of the surface slope at (i,j)
3) The distance between the point (i,j) and one or more points on the surrounding valley walls.
4) The slope at points on the surrounding valley walls.

The real creativity of the model lies in finding the distance between the point (i,j) and the surrounding valley walls. Garry Clarke and et al. clearly illustrate the process in their paper[3].

They suggest applying a ‘sectorial stencil’ at the point (i,j) and divide the compass circle into M circles and measure the minimum horizontal distance $R_m$ between the point (i, j) and the valley walls within that sector. Fig 2.1 clearly illustrates this process.

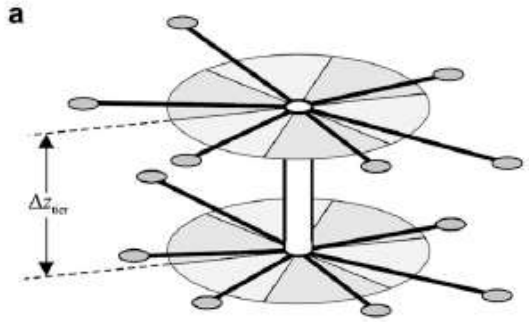



Fig 2.1 : sectorial stencil suggested in the paper to find the distance from valley walls.

The current project only deals with input values of the type 4) and the magnitude of the slope at (i,j).

In addition to the DEM of surface elevation S, it is necessary to define an ice mask **I** that has the properties $I_{ij} = 0$ for an ice-free DEM cell and $I_{ij} = 1$ for an ice covered one. A second mask that can prove useful is a surface slope mask that gives the magnitude of slope at each point.

For better convergence to ‘ground-truth’ and adherence to real-world ice-thickness estimation, it is advised to also consider the slope mask **G** where, for steeply sloping topography $G_{ij} = 1$ and for gently sloping topography, $G_{ij} = 0$. In our work, the threshold between the two is taken as $|\theta| = 25°$ but is not a sensitive parameter and could be adjusted.

G` stands for

**Size of training data**

Due to computational constraints, only a 100 DEM rasters and ice masks were used, whereas, the effect of taking $N_{set}$ = 1805, 3610, 9025, 18050, and 36100 has been established. It has been found that the training performance improved with increasing values of $N_{set}$ but that the estimation error ceased to decrease significantly when $N_{set} > 18050$.

**Hidden Layers and their composition**

Though several combinations of hidden layers and the functions they apply to the layer’s input are possible, the combination chosen here is:

**4X – 6S – 6T-Output**

Only 2 hidden layers have been chosen here due to reasons pertaining to computation time and data acquisition. Based on comparisons of network architecture we conclude that, for the present application, three-layer ANNs do not outperform two-layer ANNs, and, furthermore, they require substantial additional computer time for network training.

# Procedure

## Generation of training data

In essence, training reduces to the problem of optimizing the values of the weights $W_{ij}$ and biases for every active node of the neural network, with the aim of minimizing the error $\hat{H}_{ij} - H_{ij}$ of the output estimate $Y = \hat{H}$. (Here, $\hat{H}$ stands for the final estimate of ice thickness and is the variable to be estimated by the ANN. )Here we are faced with the dilemma that the ice-covered portions of a DEM are unsuitable for training purposes because, in most cases, we have no prior information about the ice thickness.

Thus the network must be trained using DEM cells that are not ice covered. This is supported by the fact that the ice-covered parts of the mountain, if heavily denuded, would resemble nearby areas that are currently deglaciated.

The following procedure describes the **'Bathtub-filling process'** (see fig 3.1) employed to generate close-to-reality datasets:

1) Choose a DEM image of the glacier giving the elevation of points in the image.
2) Choose a random ice-denuded cell (i,j).
3) Cover this cell by ice of a randomly generated thickness value, $\underline{\underline{H}}^*_{ij}$ to yield an ice elevation $\tilde{Z}^*_{ij}$ at that site.
4) The entire DEM is filled to this ice level to generate a new landscape realization $S^*_{ij} = \max(S_{ij}, Z^*_{ij})$ .
5) An ice mask is generated for this filling level such that $I^*_{ij} = 1$ when $(I_{ij} = 1) \vee Z^*_{ij} > S_{ij}$ and $I^*_{ij} = 0$ otherwise.
6) **The training target, hence, becomes $\underline{\underline{H}}^*_{ij}$ .**

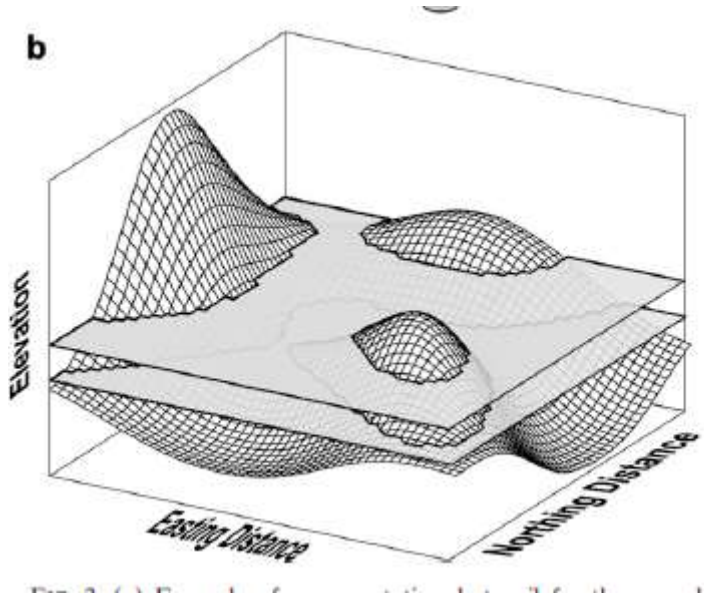



Fig 3.1: Illustration of Bathtub-filling process.

## Training of the Neural Network

The most commonly used ANNs have a multi-layered architecture with a unidirectional feed-forward flow of information and are termed multilayer perceptrons [4].

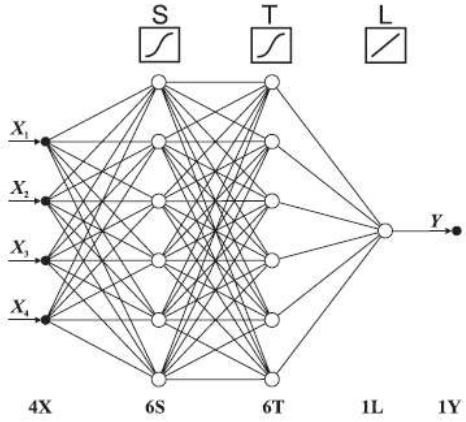



Fig 3.2: Architecture of the Neural Network used in the current project.

The steps involved in the process of training are summarised in Fig 3.3 below.

1. **Input** $x$**:** Set the corresponding activation $a^1$ for the input layer.

2. **Feedforward:** For each $l = 2, 3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$.

3. **Output error** $\delta^L$**:** Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. **Backpropagate the error:** For each $l = L-1, L-2, \ldots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. **Output:** The gradient of the cost function is given by $\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j$ and $\frac{\partial C}{\partial b^l_j} = \delta^l_j$.

Fig 3.3: Process of training the Neural Network.

An explanation of the notation used and the mathematics behind the back-propagation is given in A2.

**Generation of test datasets**

Geophysical mapping of the subglacial topography of mountain glaciers has focused on those few glaciers that have attracted scientific interest and this creates an observational bias that favors safe, accessible glaciers of modest size.

Thus we rely on glacier modelling to generate test datasets. In this respect, Prof. Garry Clarke, himself, was approached via email and his code for the 'IceFlow' module written in Python as a numerical ice dynamics model was obtained (GitHub link to the code).

**Conclusion**

Though the final DEMs of the estimated subglacial topography were not generated, the ice thickness was estimated and it is found that ANN depth estimates can yield plausible subglacial topography with a representative RMS elevation error of (+/-) 70 m and remarkably good estimates of ice volume.

Volumes were calculated using the formula given in :

$$\tilde{V}_{I\wedge G} = \int_{A_I\wedge G} \tilde{H}_{I\wedge G} dx\, dy,$$

$$\tilde{V}_{I\wedge G'} = \int_{A_I\wedge G'} \tilde{H}_{I\wedge G'} dx\, dy,$$

(here, G and G` are as described in the previous sections. ^ stands for logical AND. For example: I ^ G implies ice-covered valley walls.).

# References


1. Ohmura, A., 2004: Cryosphere during the twentieth century. The State of the Planet: Frontiers and Challenges in Geophysics, Geophys. Monogr., Vol. 150, Amer. Geophys. Union, 239–257.
2. Steiner, D., A. Walter, and H. J. Zumbuhl, 2005: The application of a non-linear back-propagation neural network to study the mass balance of Grosse Aletschgletscher, Switzerland. J. Glaciol., 51, 313–323.
3. Clarke, G. K. C., Berthier, E., Schoof, C. G., and Jarosch, A. H.: Neural networks applied to estimating subglacial topography and glacier volume, J. Climate, 22, 2146–2160, 25 doi:10.1175/2008JCLI2572.1, 2009. 4815.
4. Reed, R. D., and R. J. Marks II, 1999: Neural Smithing: Supervised Learning in FeedforwardArtificial Neural Networks.MITPress, 358 pp.
5. Lawrence Rabiner: "A tutorial on Hidden Markov Models and selected applications in speech recognition", PROCEEDINGS OF THE IEEE, vol. 77, no. 2, February 1989 (1989-02-01), pages 257 – 286.
6. Zhang, Yongyue, Michael Brady, and Stephen Smith. "Segmentation of brain MR images through a hidden Markov random field model and the expectation-maximization algorithm." IEEE transactions on medical imaging 20.1 (2001): 45-57.
7. M Anul Haq (2011). “Change Montoring of Gangotri Glacier Using Satellite Imagery” . 12th Esri India User Conference 2011. Retreived from: http://www.esri.in/~/media/esri-india/files/pdfs/events/uc2011/papers/NRM_UCP0035.pdf
8. Quan Wang. GMM-Based Hidden Markov Random Field for Color Image and 3D Volume Segmentation. arXiv:1212.4527 [cs.CV], 2012.a.
9. Andrew Blake, Pushmeet Kohli, Carsten Rother (2011). “Markov Random Fields for Vision and Image Processing”. Retreived from: http://www.cs.toronto.edu/~kyros/courses/2503/Handouts/Blake2011.pdf
10. Cuffey, K. M., & Paterson, W. S. B. (2010). The physics of glaciers (4th ed.). Burlington, MA: Butterworth-Heinemann/Elsevier.

# Appendix

## A1. Python implementation of the various abstractions like Layer, the Neural Network and corresponding functions to aid training.

### 1. Layer

```
class Layer(object):
    def __init__(self,rng,size,minibatch_size,is_input=False,
                 is_output = False,activation = tanh):
        self.is_input = is_input
        self.is_output = is_output

        self.Z = np.zeros((minibatch_size, size[0]))
        self.activation = activation
        #note that we need to store derivatives of the activations, the
        #activations themselves, weights and and the inputs.
        #all this for backprop.
        # W is the outgoing weight matrix for this layer
        self.W = None
        # S is the matrix that holds the inputs to this layer
        self.S = None
        # D is the matrix that holds the deltas for this layer
        self.D = None
        # Fp is the matrix that holds the derivatives of the activation function
        self.Fp = None

        if not is_output:
            self.W = np.asarray(
            rng.uniform(
                low=-np.sqrt(6. / (size[0]+size[1])),
                high=np.sqrt(6. / (size[0]+size[1])),
                size=size
                ),
                dtype = np.float32
            )
            if activation == sgm:
                self.W *= 4
        # if not is_output:
            # self.W = np.random.normal(size=size, scale=1E-4)
            print("shape of w: ",self.W.shape)

        if not is_input:
            self.S = np.zeros((minibatch_size, size[0]))
            self.D = np.zeros((minibatch_size, size[0]))
            print("shape of s: ",self.S.shape)
            print("shape of d: ",self.D.shape)

        #the hidden layer acts as f(x)=x when no non-linearity is specified.

        if not is_input and not is_output:
            self.Fp = np.zeros((size[0], minibatch_size))
            print("shape of z: ",self.Z.shape)
            print("shape of w: ",self.W.shape)
            print("shape of fp: ",self.Fp.shape)
```

## 2. MLP (Multi-Layer Perceptron)

```
class MLP(object):
    def __init__(self,rng,layer_config,minibatch_size=2):
        self.layers = []
        self.num_layers = len(layer_config)
        self.minibatch_size = minibatch_size

        for i in range(self.num_layers-1):
            if i==0:
                print("Initializing input layer with size {0}.".format(
                    layer_config[i]
                ))
                self.layers.append(Layer(rng=rng,size = [layer_config[i]+1, layer_config[i+1]],
                                         minibatch_size=minibatch_size,
                                         is_input=True))
            else:
                print("Initializing hidden layer with size {0}.".format(
                    layer_config[i]
                ))

                self.layers.append(Layer(rng=rng,size= [layer_config[i]+1, layer_config[i+1]],
                                         minibatch_size=minibatch_size,
                                         activation=sgm))

        print("Initializing output layer with size {0}.".format(
            layer_config[-1]
        ))
        self.layers.append(Layer(rng=rng,size=[layer_config[-1], None],
                                 minibatch_size=minibatch_size,
                                 is_output=True))
        print("Completed!")

    def forward_propagate(self, data):
        #add bias to input before trying to forwrd propagate.
        #node that if the input layer has 'n' nodes and the next layer has
        #'n_next' nodes, the weight matrix will be of size : (n_next X (n+1))

        self.layers[0].Z = np.append(data, np.ones((data.shape[0], 1)), axis=1)
        for i in range(self.num_layers-1):
            self.layers[i+1].S = self.layers[i].forward_propagate()
        return self.layers[-1].forward_propagate()

    def backpropagate(self, yhat, labels):
        self.layers[-1].D = (yhat - labels).T
        for i in range(self.num_layers-2, 0, -1):
            # do not calculate deltas for the bias values
            W_nobias = self.layers[i].W[0:-1,:]
            self.layers[i].D = np.multiply(W_nobias.dot(self.layers[i+1].D),self.layers[i].Fp)

    def update_weights(self, eta): #eta is the learning rate
        for i in range(0, self.num_layers-1):
            W_grad = -eta*(self.layers[i+1].D.dot(self.layers[i].Z)).T
            self.layers[i].W += W_grad

        #note that this is w = w - eta*(a_inXdelta_out). Refer to nnanddl.net
```

## 3. Generation of training data by 'bathtub-filling' process.

```
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from osgeo import gdal
import pickle

ds = gdal.Open("glacier.tif")
original_dem = np.array(ds.GetRasterBand(1).ReadAsArray())

im = plt.imread("im_new.png")
#here, 0:ice and 1:no ice.
target = []
#generate 100 cases by bathtub filling.
for i in range(1,51):
    print("Generating " + str(i) + "...")
    rand_x = np.random.randint(500)
    rand_y = np.random.randint(500)
    rand_height = np.random.randint(50)
    new_ice_height = rand_height + original_dem[rand_x][rand_y]
    new_dem = np.array(np.ones((500,500)))
    new_im = np.array(np.ones((500,500)))
    target.append(rand_height)
    for j in range(500):
        for k in range(500):
            new_dem[j][k] = max(new_ice_height,original_dem[j][k])
            if (im[j][k] == 0) or new_ice_height > original_dem[j][k]:
                new_im[j][k] = 0
            else:
                new_im[j][k] = 1
    pickleFileName1 = str(i) + "_dem.pickle"
    pickleFile = open(pickleFileName1, 'wb')
    pickle.dump(new_dem, pickleFile, pickle.HIGHEST_PROTOCOL)
    pickleFileName1 = str(i) + "_im.pickle"
    pickleFile = open(pickleFileName1, 'wb')
    pickle.dump(new_im, pickleFile, pickle.HIGHEST_PROTOCOL)
    pickleFile.close()

pickleFileName1 = "target.pickle"
pickleFile = open(pickleFileName1, 'wb')
pickle.dump(target, pickleFile, pickle.HIGHEST_PROTOCOL)
```

4. **Non-linearities.**

```
def sgm(X, der=False):
    if not der:
        return 1 / (1 + np.exp(-X))
    else:
        return np.multiply(sgm(X),1-sgm(X))

def relu(x, der=False):
    """Rectifier activation function.
    Use der=True for the derivative."""
    if not der:
        return np.maximum(0, x)
    else:
        if x <= 0:
            return 0
        else:
            return 1

def tanh(x,der = False):
    if not der:
        return np.tanh(x)
    else:
        return 1-x**2
```

## A2. Mathematics behind Backpropagation.

**Summary: the equations of backpropagation**

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{BP1}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{BP2}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{BP3}$$

$$\frac{\partial C}{\partial w^l_{j\,k}} = a^{l-1}_k \delta^l_j \tag{BP4}$$

Another term, 'error matrix' is introduced to track the changes in the cost function :

$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

when the input to each layer changes ie

$$\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}.$$

.

**(All the above formulae are easily derived by repeated application of chain rule of partial derivatives.)**

**BP1.** δ of the output layer is just the difference between the estimate and the 'ground-truth' multiplied by the activation (tanh, simoid, relu etc.) of the derivative of the input to the output layer.

**BP2.** δ of all the other layers is just the weighted sum of the errors with respect to the weights of the layer under consideration multiplied by the the activation (tanh, simoid, relu etc.) of the derivative of the input.

$$= a_{\text{in}} \delta_{\text{out}},$$

**BP3.** The rate of change of the cost function with respect to the weights is:

**BP4.** The rate of change of the cost function with respect to the bias terms is simply, δ.

A3. **Mathematical Formulation of the Gaussian Mix**ture Model and the EM algorithm**:**

For GMMs, we will find the clusters using a technique called “Expectation-Maximization”.

1. **INITIALISATION:**

To kickstart the EM algorithm, we’ll randomly select data points to use as the initial means, and we’ll set the covariance matrix for each cluster to be equal to the covariance of the full training set. Also, we’ll give each cluster equal “prior probability”. A cluster’s “prior probability” is just the fraction of the dataset that belongs to each cluster. We’ll start by assuming the dataset is equally divided between the clusters.

2. **EXPECTATION:**

In the “Expectation” step, we calculate the probability that each data point belongs to each cluster.

$$g_j(x) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_j|}} e^{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)}$$

| Symbol | Meaning |
|---|---|
| $g_j(x)$ | The PDF of the multivariate Gaussian for cluster j; the probability of this Gaussian producing the input x |
| $j$ | Cluster number |
| $x$ | The input vector (a column vector) |
| $n$ | The input vector length |
| $\Sigma_j$ | The n x n covariance matrix for cluster j |
| $\lvert\Sigma_j\rvert$ | The determinant of the covariance matrix |
| $\Sigma_j^{-1}$ | The inverse of the covariance matrix |

A multivariate Gaussian (“multivariate” just means multiple input variables) is more complex because there is the possibility for the different variables to have different variances, and even for there to be correlation between the variables.

$$w_j^{(i)} = \frac{g_j(x)\phi_j}{\sum_{l=1}^{k} g_l(x)\phi_l}$$

| Symbol | Meaning |
|---|---|
| $w_j^{(i)}$ | The probability that example i belongs to cluster j |
| $g_j(x)$ | The multivariate Gaussian for cluster j |
| $\phi_j$ | The "prior probability" of cluster j (the fraction of the dataset belonging to cluster j) |
| $k$ | The number of clusters |

**3. MAXIMISATION:**

$$\phi_j := \frac{1}{m}\sum_{i=1}^{m} w_j^{(i)},$$

$$\mu_j := \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}},$$

$$\Sigma_j := \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}$$

The parameters are revised and the "Expectation" step is carried out again. This process is repeated till the parameters do not change by a great amount or some other indicator is used.

## A4. MATLAB scripts

### 1. MATLAB code for segmentation and thresholding to get fraction of ice in the image:

```
I=imread(originalImage);
I=imresize(I,[500,500]);
I=double(I);
class_number=6;
potential=0.005;
maxIter=30;
seg=ICM(I,class_number,potential,maxIter); %function for segmentation
                                            using GHMRF_EM Algorithm.
area=0;
mini=900*ones(1,class_number);
max=-1*ones(1,class_number);
c=zeros(1,class_number);
for i=1:500
        for j=1:500
                for k=1:class_number
                        if seg(i,j)==k
                                c(k)=c(k)+1;
                        end
                end
        end
end
for i=1:500
        for j=1:500
                for k=1:class_number
                        if seg(i,j)==k && I(i,j)<mini(k)
                                mini(k)=I(i,j);
                        end
                        if seg(i,j)==k && I(i,j)>max(k)
                                max(k)=I(i,j);
                        end
                end
        end
end
X=abs(255*ones(1,class_number)-mini);
[l y]=min(X);
for i=1:500
        for j=1:500
                if seg(i,j)==y
                        R(i,j)=255;
                        area=area+1;
                else
                        R(i,j)=0;
                end
        end
end
for k=1:class_number
        if mini(k)==0
                s=c(k);
        end
end
```

```
area=area/(500*500-s);
imshow(R);
```

**2. MATLAB code for calculating change in elevation using DEM-Geotiff images:**

```
[A, R] = geotiffread(originalImage);
%I=imread(originalImage);
%I=rgb2gray(O);
min=10000000;
max=-1;
for i=1:3601
        for j=1:3601
                if(A(i,j)<min)
                        min=A(i,j);
                end
                if(A(i,j)>max)
                        max=A(i,j);
                end
        end
end
B=int16(2000.*ones(3601,3601));
A=A-B;
```

## A5. Python Codes

**1. Python code to implement various segmentation algorithms and compare them.**

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

img = cv2.imread('15Oct-1999.jpg',cv2.IMREAD_UNCHANGED)

ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_BINARY)
#ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_TOZERO)
#ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH_TRUNC)
#th = cv2.adaptiveThreshold(imgb,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
        cv2.THRESH_BINARY,11,2)

#blur = cv2.GaussianBlur(img,(5,5),0)
#ret,thresh1 = #cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)

fig = plt.figure()

plt.subplot(1,2,1),plt.imshow(thresh1,cmap = 'gray', interpolation = 'bicubic')
plt.title('*Type of thresholding*')
plt.xticks([]),plt.yticks([])
```

```
plt.subplot(1,2,2),plt.imshow(img,cmap = 'gray', interpolation = 'bicubic')
plt.title('Original')
plt.xticks([]),plt.yticks([])
plt.show()
```

**2. Python code for plotting the polynomial model to fit the observations:**

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.polynomial import polynomial as P


fig = plt.figure()

ax = fig.add_subplot(111)
ax.set_xlabel('No of days from 30-9-2000')
ax.set_ylabel('Fraction of glacier region in image')

c = P.polyfit(x,y,3)
p = np.poly1d(c)

xp = np.linspace(0, 800, 100)

plt.ylim(0,1)
plt.plot(x, y,'-o')
for xy in zip(x, y):
        ax.annotate('(%s, %s)' % xy, xy=xy, textcoords='data')
plt.grid()
plt.show()

fig.savefig('plot3.png')
```

## <u>A6.</u> Hidden Markov Random Field implementation in MATLAB:

**1. HMRF_EM.m**

```
function [X GMM]=HMRF_EM(X,Y,GMM,k,g,EM_iter,MAP_iter,beta)

sum_U=zeros(1,EM_iter);

for it=1:EM_iter
   fprintf('Iteration: %d\n',it);
   %% update X
   [X sum_U(it)]=MRF_MAP(X,Y,GMM,k,g,MAP_iter,beta,0);

   %% update GMM
   GMM=get_GMM(X,Y,g);
```

```
    if it>=3 && std(sum_U(it-2:it))<0.01
        break;
    end
end

figure;
plot(1:it,sum_U(1:it),'LineWidth',2);
hold on;
plot(1:it,sum_U(1:it),'.','MarkerSize',20);
title('sum of U in each EM iteration');
xlabel('EM iteration');
ylabel('sum of U');

figure;
imagesc(X);
```

**2. MRF_MAP.m**

```
function [X sum_U]=MRF_MAP(X,Y,GMM,k,g,MAP_iter,beta,show_plot)

[m n temp]=size(Y);
x=X(:);
y=reshape(Y,[m*n 3]);

sum_U_MAP=zeros(1,MAP_iter);
for it=1:MAP_iter % iterations
    fprintf('  Inner iteration: %d\n',it);
    U=zeros(m*n,k);
    U1=U;
    U2=U;

    for l=1:k % all labels
        for c=1:g
            mu=GMM{l}.mu(c,:);
            Sigma=GMM{l}.Sigma(:,:,c);
            p=GMM{l}.PComponents(c);

            yi=bsxfun(@minus,y,mu);
            temp1=(yi*inv(Sigma)).*yi;
            temp1=sum(temp1,2)/2;
            temp1=temp1+log(sqrt(det(Sigma)));
            U1(:,l)=U1(:,l)+temp1*p;
        end

        for ind=1:m*n % all pixels
            [i j]=ind2ij(ind,m);
            u2=0;
            if i-1>=1
                u2=u2+(l ~= X(i-1,j))/2;
```

```
            end
            if i+1<=m
               u2=u2+(l ~= X(i+1,j))/2;
            end
            if j-1>=1
               u2=u2+(l ~= X(i,j-1))/2;
            end
            if j+1<=n
               u2=u2+(l ~= X(i,j+1))/2;
            end
            U2(ind,l)=u2;
         end
      end
      U=U1+U2*beta;
      [temp x]=min(U,[],2);
      sum_U_MAP(it)=sum(temp(:));
      X=reshape(x,m,n);

      if it>=3 && std(sum_U_MAP(it-2:it))<0.01
         break;
      end
   end

   sum_U=0;
   for ind=1:m*n % all pixels
      sum_U=sum_U+U(ind,x(ind));
   end
   if show_plot==1
      figure;
      plot(1:it,sum_U_MAP(1:it),'r');
      title('sum U MAP');
      xlabel('MAP iteration');
      ylabel('sum U MAP');
      drawnow;
   end
```