Prediction of Cloud Movement from Satellite Images using Neural Networks

Marius E. Penteliuc and Marc Frincu
West University of Timisoara
Faculty of Mathematics and Computer Science
Department of Computer Science
Romania
{marius.penteliuc, marc.frincu}@e-uvt.ro

***Abstract*—Predicting cloud movement and dynamics is an important aspect in several areas, including prediction of solar energy generation. Knowing where a cloud will be or how it evolves over a given geographical area can help energy providers to better estimate their production levels. In this paper we propose a novel approach to predicting cloud movement based on satellite imagery. It combines techniques of generating motion vectors from sequential images with neural networks. First, the images are masked to isolate cloud pixels, then Farneback's version of the Optical Flow algorithm is used to detect motion from one image to the next and generate motion vector flow for each pair of images. After that, a feed forward back propagation neural network is trained with the vector data derived from the dataset imagery. Different parameters for the duration of the training, size of the input, and the neighborhood radius of one point in the scene are used. Promising results are presented and discussed to weight the potential of the proposed algorithm for forecasting cloud cover and cloud position in a scene.**

***Index Terms*—cloud motion; forecasting; neural networks; satellite imagery;**

I. Introduction

Many satellite programs are funded and running with the purpose of capturing Earth data and provide valuable historical products that can be used in research. These products include spectral images of the surface reflectance in visible and near-infrared bands. Studies are making use of the satellite imagery to evaluate land cover, vegetation growth, human influence over land surface, cities expansion, weather and climate change, monitor the environment and natural resources. Despite the great aid these image provide in monitoring change, about two thirds of the captured surfaces are covered by clouds [1] which negatively impact the work of monitoring applications and evaluating land surface change. Solar energy generation is one application which requires accurate estimates of cloud coverage over specific areas [11]. Algorithms that work on such images need to have a clear view of the surface to output usable results. If clouds are present and obstruct the view, these algorithms can misinterpret data from cloudy regions and output unusable data. Clouds are a contaminant and their shadows is also impeding remote sensing applications.

Due to this restrictive factor, the necessity of assessing cloud cover became more pressing and techniques were developed to enable classification of images based on cloud coverage percentage. Then cloud and cloud shadow screening methods took it a step further by detecting and masking out cloud and cloud shadow pixels from the image in order to make use of the clear pixels instead of discarding the entire image. More details will be given in Section II.

Harnessing the power of solar irradiation is a task entirely dependent on weather conditions, specifically on the absence of cloud cover. As clouds passing over a solar power plant cannot be eradicated, a solution is to identify clouds and predict their near future position by analyzing their movement. This way, plant operators can benefit from the cover of shadow by performing maintenance on the installed equipment, and not restrict it from running when the sky is clear.

The contents of this paper are structured as follows: In Section II we will present work that has been done regarding cloud detection, movement, and simulation; a new algorithm is proposed in Section III; in Section IV we describe the setup for making experiments and the results obtained; finally conclusions and future directions are presented in Section V.

II. Related Work

A. Cloud Movement Detection

Hamill et al. [2] developed a cross-correlation-based technique to take cloud features and find them in separate frames and generate motion vectors. They applied multiple cross-correlation analysis over an area in a frame with the surrounding areas in the next one. The results of the correlation were defined as displacement vectors and, by using a backward trajectory technique, they forecast cloud motion up to a few hours, but is best in the first hour. However, limitations such as smoothing image too much over time, problems near the image boundaries, and terrain advection left room for improvement and raised a necessity for cloud masking to do a better job of determining motion by having cloudy pixels identified first, then motion vectors computed for each pixel.

B. Cloud Detection Algorithms

One of the earliest methods for identifying cloud-contaminated scenes retrieved from satellite imagery is the Automated Cloud-Cover Assessment (ACCA) algorithm [6]. It was developed for the Landsat 7 ground system and to process Enhanced Thematic Mapper Plus (ETM+) data. An older version of the algorithm was incorporated into the Thematic

Mapper Image Processing System (TIPS) for the Thematic Mapper (TM) sensor on board the satellites Landsat 4 and Landsat 5 in the early 1980s [3]. The computational load was reduced due to the technical limitations at that time and only three bands were used and the images were sub-sampled down to almost 6% of the original data.

The new ACCA algorithm took advantage of Landsat 7's more precise instruments and successfully assessed cloud cover with an error of $\pm 5\%$. It takes as input top of atmosphere (TOA) reflectance and at sensor temperature converted from the raw numbers provided by the sensor [4] and process the images in two passes. The first pass uses eight filters to determine a precise cloud signature and categorize regions into three classes: clouds, non-clouds, and ambiguous data. The second pass then begins a thermal analysis using a cloud signature derived from either the cloud or non-cloud classes, or both. The presence of snow or desert regions in the scene are taken into account when determining the cloud signature. The ambiguous data is revised by threshold comparison and non-cloud pixels that have 5 or more cloud pixel neighbors are labelled as cloud to boost the cover content and reflect how much of a scene is unusable data. A grid overlay will determine the cloud cover percentage score for the scene.

While ACCA did successfully identified the presence of clouds in a scene, it could not draw a clear boundary to mask out the clouds in a scene, which lead to imprecise isolation. It also had trouble detecting warm cirrus clouds and often miss-classified high altitude snow and ice as clouds [7]. All that considered, it is important to remember that the intended purpose of the ACCA system was to assess cloud coverage percentage, not to screen clouds precisely.

Different methods were developed over time to screen clouds and cloud shadows precisely including image fusion techniques to remove clouds and their shadows [8], or using a Normalized Difference Snow Index (NDSI) to isolate clouds over ice sheets. A two-pass, similar to ACCA, algorithm that generates an internal cloud mask [12]. Zhang et al. (2002) developed a haze correction method [14], modified by Hgarat-Mascle and Andr in 2009 to detect cloud pixels by a distance from them to a clear-sky line [15]. Cloud shadow detection used to be done by spectral test, but geometry-based methods, like object matching and scattering differencing were producing better results.

Zhu & Woodcock developed in 2012 a new method called Fmask (Function of mask) to precisely identify, isolate, and mask clouds [16]. Similar to ACCA, this method takes as input the TOA reflectances for Bands 1 through 5 and 7, but also the brightness temperature for Band 6 of Landsat's TM and ETM+ (values converted using the LEDAPS atmosphere correction tool [5], [12]). It identifies a cloud mask by passing twice over an image: the first pass runs a series of tests to remove land, desert, rock, and water. This pass will exclude pixels that are certainly not cloud, but might include pixels representing bright rock, water, snow, and ice, leaving a layer of absolute clear pixels. If the layer of absolute clear pixels is at least 0.1% of the scene, the second pass will compute the cloud probability over water and the cloud probability over land separately using water temperature probability and brightness probability, and land temperature probability and variability probability respectively.

The potential cloud shadow layer is computed by performing a flood-fill transformation on Band 4 data [13], and observing the difference on the transformed Band 4 data and the original Band 4 data. The potential snow layer is identified using a modified MODIS snow mapping algorithm [9] with a different Normalized Difference Snow Index (NDSI) threshold of 0.15 that has been used by Wildt et al. [10]. The cloud shadow mask is refined by using an object-based cloud and cloud shadow match, which basically make use of the know sattelite sensor angle, solar zenith angle, and solar azimuth angle to project the direction of a cloud's shadow. Fmask will prioritize labelling of pixels in the following order: cloud pixels with highest priority, followed by shadow pixels, and finally snow pixels with lowest priority.

Fmask is widely known in the field and is an important and reliable tool to mask clouds in a scene. Although it has some limitations in failing to isolate thin warm clouds, misidentifying as cloud cold, very bright land features (salt pans, cold snow), and having difficulties in very complex surface reflectances, Fmask has seen improvements in its performance when Zhu, Wang & Woodcock expanded the algorithm in 2014 to work on Landsat 8 and Sentinel 2 images, and modifying parts of the original workflow to have better results [17]. In 2017, Qiu, He, Zhu et al.[18] further improved on the algorithm by introducing Mountainous Fmask (MFmask) which could better detect cloud and cloud shadow in mountainous regions.

Frantz et al. [20] made an improvement in 2018 to Fmask's functionality on Sentinel 2 images by better separating clouds from very bright land surfaces using the parallax effect of clouds when viewed from the satellite's three near infrared bands to create a Cloud Displacement Index (CDI). Later, Qiu et al., 2019 integrated the CDI into the FMask algorithm version 4.0 (in beta as of this writing)[22], along with integrating Global Surface Water Occurence (GSWO), Digital Elevation Models (DEM)[18], and making other adjustments. Many more are integrating FMask into existing workflows to achieve more off of the cloud mask that is generated by it.

C. Cloud Movement Simulations

In 2018, Alisson R. Silva et al. [23] presented and evaluated a parallel implementation of a complex system for cloud simulation with a cellular automaton by using a parallel stencil programming tool called PSkel. To simulate cloud dynamics in two dimensions as a function of the environment temperature they used five partial differential equations, for: horizontal and vertical air speed, temperature, water vapor, and cloud water. The programming languages used for developing and testing the algorithms were OpenMP (Open Multi Processing), TBB (Threading Building Blocks), and CUDA (Compute Unified Device Architecture). The parallel model was shown to be stable with respect to the thermal equilibrium, and the

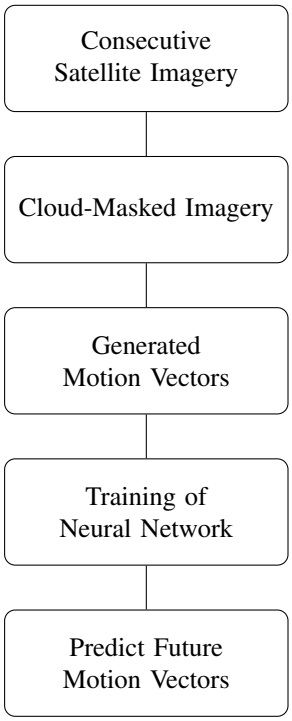


Fig. 1: Proposed algorithm, From top to bottom: the consecutive satellite images are cloud-masked using a threshold technique; then motion vectors are generated by using OpenCV's Optical Flow method; the neural network is trained with motion vector data; and the predicted output is a new set of motion vectors.

runtime decreased as the number of threads increased (until the computing limit). There is a 6.5 fold performance gain using the parallel model over the sequential version. They observed that shorter simulation times were stable, while longer ones were unstable.

To summarize, demand and interest for cloud detection and cloud evolution prediction is high and advances in this area are needed. Our aim in this paper is to present a novel approach to predicting cloud coverage using neural networks trained on motion vectors generated by movement detection. The results are satisfactory and show that forecasting of clouds in a scene can be achieved from motion vector data.

In the following chapter the setup for running experiments is described and the results are discussed.

III. Proposed Prediction Algorithm

The strategy for the algorithm is illustrated in Fig. 1 and consists of three main steps:

I Use one masking algorithm (that is fit for the dataset) and isolate cloudy pixels in the scene. This is not relevant for the prediction itself, but for creating a cloud mask when using predicted data.

II Generate motion vectors that describe the movement of each image pixel between frames. The vectors are assembled into a structure that is used as training data for a neural network.

III A neural network is trained on motion vector data with the scope of predicting future data from series of motion vectors. The cloud mask can be obtained by distorting pixels of the last frame according to the predicted motion vectors.

154	200	243	255
93	167	211	242
64	112	178	215
17	85	104	149

$\xrightarrow{199}$

0	0	243	255
0	0	211	242
0	0	0	215
0	0	0	0

Fig. 2: Example of applying thresholding to zero on a matrix for a threshold value of 199.

A. The masking algorithm

Given the focus of the paper is prediction, the algorithm uses a rather straightforward thresholding technique to determine cloudy pixels in a scene. The training images are series of snapshots taken from satellites in a geosationary orbit, which captures the whole visible disk of the Earth every half hour, rather than satellites in a polar or low-earth orbit, which capture the same location much less frequent. The size of the captured scene also means snow and water bodies are not a concern in cloud identification and the thresholding technique is suited for the dataset.

First, the images are converted to grayscale using a basic method of converting pixel values to gray. Then the gray images are processed after a threshold value. When working with thresholds, we make use of the fact that an image has 255 levels of brightness starting from 0 (being the darkest) up to 255 (which is the brightest), and an image can be changed in many ways by having different thresholding operations with them. One operation could be applying a binary thresholding to an image, meaning all values over the threshold will be set to the maximum brightness level and all values equal or less than it will be set to 0, leaving an image with just the two brightness levels: 0 and 255. Another example of thresholding is applying a truncate thresholding to an image, where all values greater than the threshold are set to the threshold value and all values smaller than the threshold remain unchanged, essentially cutting the brightest parts of the image and reducing them to a lower level. The thresholding operation used here is commonly referred to as thresholding to zero. When thresholding to zero, all values below the specified threshold are set to 0, the darkest possible value, and the rest are left untouched, as shown in Fig. 2.

After applying the thresholding operation to each image, they are saved together as frames in a video file to be easily available when processing them for the identification of motion vectors.

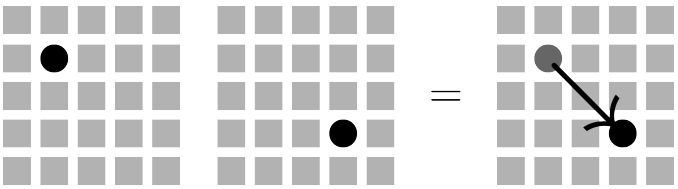


Fig. 3: Motion vector: one pixel from the first image is located into the second image and a vector depicting direction and length is generated (arrow).

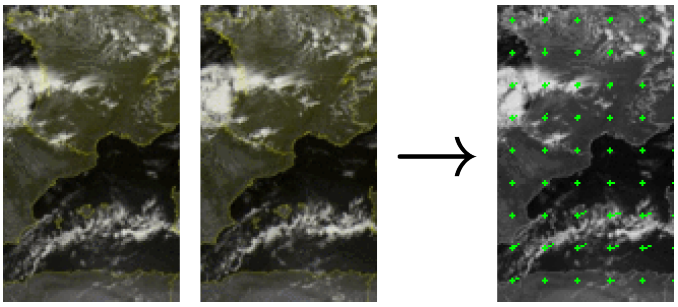


Fig. 4: From one pair of images a flow of motion vectors is generated.

B. Motion vector algorithm

The output of the masking algorithm applied to the series of images is an ordered collection masked frames. To identify movement and generate motion vectors, a technique called Optical Flow is used to process the sequence of frames. This technique has several variants in use, but upon analyzing the needs of the workflow, we have decided to use the variant based on Farneback's algorithm to detect cloud movement [19]. The frames are fed to the algorithm as input two by two such that each frame (except for the first and last) is fed two times, once with the frame before, and once with the frame after. The algorithm will try and compute were each pixel from the first frame has gone in the next frame, essentially detecting the movement of the pixels that changed locations, see Fig. 3.

Each pixel is associated a touple as follows: If a pixel has moved, it is given two values: a length value and a direction value such that when applying these values to its location in the first frame, its result is the new position in the second frame. If a pixel has not changed location, the values will be both zero. All pixels are given two such values that constitute a vector, and the output of the algorithm is a motion vector flow of the two frames. The process is repeated for each consecutive pair of images and all flows are saved in a collection as indicated by Fig. 4.

The output of this step is a series of motion vector flows equivalent to the number of consecutive pairs of frames, that is one less than the number of processed images. One might find it easier to understand these vectors as a representation of the direction and strength of wind at a point location.

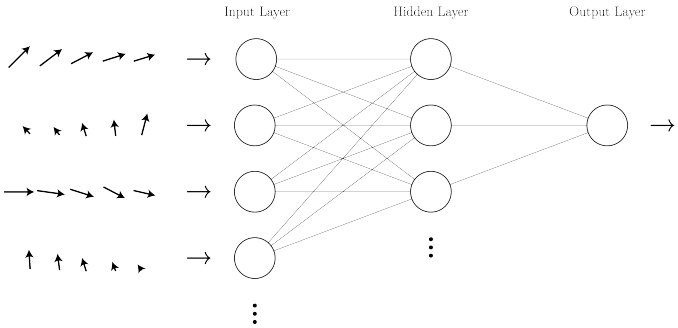


Fig. 5: Representation of a neural network containing the input layer, one hidden layer, and the output layer. The arrows on the left represent the motion vectors that are fed into the network.

C. Neural network

The action of predicting a value based on a number of inputs is a task that requires special performance that only specific techniques can achieve. Neural networks are such techniques that when given a dataset and a target output, can learn how to predict numbers, strings, or other data types. A network is able to achieve this because of the architecture of nodes and arches it is built on. The nodes are commonly called neurons – hence the name of the network – and the connections between them can be associated with synapses – the biological name given to the communication paths that exist between the neurons in the human brain.

The specific neural network used in this algorithm is a simple multilayer perceptron running in a feed forward back propagation configuration [24] consisting of three neuron layers: an input layer, a hidden layer and an output layer, as shown in Fig. 5.

Types of neural network, known as convolutional neural networks, had been trained to recognize patterns even if images are distorted to a degree [25], [26], [27], but taking the same pixel location of a scene over several snapshots and looking to see if it was cloudy or not is little information to use for making predictions. But vectors that show the motion of clouds have rich data about their direction and length (which show the direction and distance a cloud pixel travels). They are more useful for making predictions of what properties future vectors will have using multilayer perceptrons[1], and use that information to reveal where each cloudy pixel will move. The input layer is the entry point of the network and the data that is to be fed into the network is copied here. The hidden layer is called hidden because it is managed entirely by the algorithm

[1]The current implementation does not use a convolutional neural network, however, there is potential of such a network to produce good results under the circumstances of working with motion vector data. Currently, I am implementing and testing this type of network on a larger dataset and will update the paper to include the findings.

and we can have no direct user influence or outside interaction. The output layer is where the results of the network processing is made available to be retrieved.

One neuron in the input layer is holding the two motion vector values (length and direction) of one point in space for all values they have during the sequence, meaning that for the five flows of motion vectors, one neuron in the input layer will have as input an array of 10 values (length and direction * five motion vectors). This array can increase if we have more than five flows available. Also, if we send motion the vector data of the neighboring pixels, the array will further increase. I do this to keep one neuron responsible for one motion vector and its neighbors throughout the entire sequence. When talking about input size, I am referring to how many points of the sequence of frames are sent through the network.

This network adapts itself to the training input by the means mentioned before and also by increasing or decreasing the number of neurons in the input layer according to how large is the training input. This way the network can work on different sized inputs while automatically adjusting the number of neurons it has. The number of neurons in the hidden layer is equal to the number of values in one pixel sequence.

At runtime the network will split the data into training input and training target (a target value is the last value had by a point in the sequence) according to the arguments, and use a common activation function to map vector values in the range 0 to 1. The is done to make the algorithm work on the dataset without having values too small or too large. Then the connections between each layer neurons synapses are created and initialized with random weights, in the range -1 to 1. If a previous training was run with the same arguments, the network will load and reuse those trained synapses.

A loop over the desired number of epochs is set and the training process begins. The input target is copied to the input layer. Then the dot product of the input layer and the synapses between it and the hidden layer is computed and saved into the neurons of the hidden layer. Same is done next where these neurons and the second set of synapses are computed in a dot product and the result is saved into the output layer. This result is said to be the network's prediction and it is unlikely, at first run, to be anything close to reality because of the random weighted synapses. The steps up to this point in the loop are part of the feed forward reasoning behind the networks naming. Data so far has traveled from left to right through the input layer, then the hidden layer and finally the output layer. Back propagation means that some data has to travel backwards through the network and when doing so, will actually make the network adjust itself to make better predictions, essentially learning its way around the data.

The output that the network has produced is a result of processing the training input. This result is compared to the training target of the corresponding input and an absolute difference is saved as the error, representing how far off is the guessed output from the real value. The error is distributed to the synapses between the hidden layer and the output layer and then over to the synapses between the input layer and the hidden layer. This distribution will nudge the weights proportionally up or down, depending on how much has the weight contributed to the error. The synapses now are not that random, but are actually a bit tuned to the training input. To this point, a single training epoch has passed. The steps are taken again and repeated until the desired number of epochs for training is met and the error should get smaller and smaller, depending on the configuration and the dataset used. The synapses are saved locally so that the information contained by them can be later used for prediction or further trained.

With enough training epochs, the algorithm should be able to converge and be reliable in predicting a value with a sufficiently small error. Again, the success rate of the training is strongly dependent on the training data. Input can be given in smaller or larger amounts, containing few or more related values. The number of epochs can be of a reasonable duration for training or can be so time consuming that it is not satisfiable to continue training until a small enough error is reached.

After the first prediction, the algorithm can be applied on the motion vectors used for making the prediction and the prediction itself, and so on. However, the longer the predicted is in the future, the more uncertain is the prediction, as is with any forecasting because we are we will no longer predict on real data, but on other predictions.

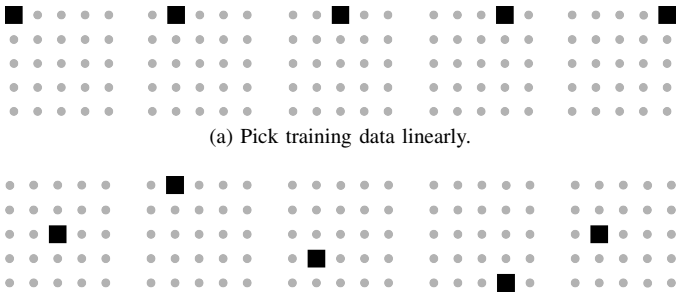
IV. Experiments

In this section we focus on the setup designed to run experiments and the results of our approach.

A. Setup

Data used for the purposes of training the network was made available by Sat24.com, a web service that specializes in weather products such as rain forecasting, and pictures of locations on Earth. The pictures are taken at time intervals of about 15 to 20 minutes by geostationary satellites and are downloaded to the server as soon as the communication links allows. The product we used for the training of the network can be accessed automatically by a system command to save the resource locally. It consists of six images sized at 400 pixels width by 290 pixels height, adding it to a total of $696\,000$ pixels per product. The images have text overlays in the margins and translucent thin country borders, but this does not affect the algorithm because of the magnitude with which clouds are moving in the scene. After processing the six images, five collections of motion vectors (flows) are formed, one for each consecutive pair of images. Each vector has the two attributes, length and direction, so all five collections have $1\,160\,000$ values that can be used for network training.

The language used for the processing and training of the network is Python 3. The Numpy library was used to perform matrix calculations on the data, and OpenCV was used to perform image adjustment and vector generation through the Optical Flow API.



(a) Pick training data linearly.

(b) Pick training data randomly.

Fig. 6: Selection methods for training. Top: the training input is selected sequentially (the points are neighbors); Bottom: the training input is selected arbitrary (the points are not necessary neighbors).

Fig. 7: One pixel with a neighborhood radius of 0 (left), 1 (center), and 2 (right).

The scripts were run on an 2.6 GHz Intel Core i5 x86-64 machine running macOS with Python 3.7, the Numpy library, OpenCV 3.4.2.

The training process is run on various scenarios. They differ by the amount of the input size given each run, the amount of neighboring motion vectors taken around a point, the duration of the training, and the way input is selected from the dataset – either starting with the top left point of the sequence and taking following points by row, either picking randomly from somewhere in the image every 10% intervals of the training duration, see Fig. 6. Using random points for training decreases the chance that all points are outside cloudy regions which would not help training. Training with points that are next to each other will help in generalizing vector data of several points into a single value because neighboring points are in a tendency to move together.

We varied all tests by the factors mentioned above using the values below:

- Size of the input: 1, 4, 8, 16, 32, and so on;
- Neighborhood radius: 0, 1, and 2 (see Fig. 7);
- Duration of training cycles: 1000, 10 000, and 100 000 epochs;
- The selection method: random and sequential;
- The five images used are 400 by 290 pixels.

To asses the overall accuracy of the prediction, we computed the mean error between the target output and the predicted one.

B. Results

The correctness was tested using the Mean Absolute Percentage Error (MAPE):

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|,$$

where A_t is the actual target value, and F_t is the forecast value. And the formula for MSE is:

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}\left(A_t - F_t\right)^2.$$

For an input size of four in all three neighborhood radii cases, the error was computed at 0.34, and for input size of eight the error was computed at 0.44. Their respective Mean Squared Errors (MSE, used in numerical predictions) are 4.50e-6, and 9.76e-6.

Increasing the neighborhood radius will mostly decrease the error, except for rare cases. This is most probably due to the fact that neighboring points will generally move together as a cluster of points. One particular outcome to note is that, usually, a random selection type over a linear one does not improve the error for this particular dataset. Also, during the same number of epochs, randomization of training data will decrease the error slower than not randomizing it – even when taking into account neighborhood radius.

Fig. 8 shows evolution of the mean error for relevant configurations of the network.

Configurations with random selection type and one neighborhood radius have a higher mean error then other configurations with the same training epochs. This means that while at a linear selection type, results get better with training time (number of epochs) and also with an increased neighborhood radius, using the random selection method results will get better only with more epochs increasing the neighborhood radius does not guarantee better results in this case.

More tests are required to further verify the correctness of the algorithm and the suitableness for predicting cloud cover with cloud location.

V. Conclusion

In this paper I have summarized past and current work on the topic of cloud cover screening. Observations of the cloud cover had been ongoing for a long time and substantial research has been done to correctly assess cloud coverage and isolate it from the Earth surface. The Optical Flow technique has proved to be very reliable in the detection of movement if used with the right parameters for the data. The motion vectors generated by the algorithm are useful in determining the angle (direction) and magnitude (length) of the movement from frame to frame. The use of neural networks for the prediction of cloud movement based on motion vectors determined from cloud masks is promising and should be studied in more depth as the potential of this method producing accurate results is of great value.

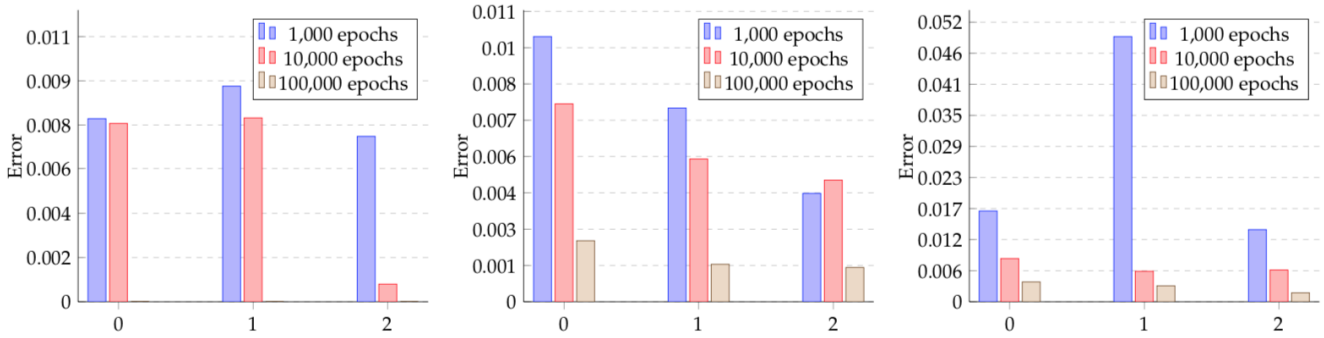


Fig. 8: Mean Error evolution in percentage for different neighborhood radii after training with random selection type – input size of 1 (left), 4 (center), and 8 (right).

A bigger dataset for training the network should yield better results and different configurations of input data should be tested to verify how effective the algorithm is in various scenarios and how should the data be fed in order to output the best results. There are many more configurations to experiment with in predicting motion vectors.

Future work will focus on more detailed experiments and comparisons with existing state of the art algorithms.

REFERENCES

[1] A.S. Belward and C.R. Valenzuela, *Remote sensing and geographical information system for resource management in developing countries*, Kluwer Academic Publishers, Dordrecht/Boston/London, 1991.

[2] Hamill, T.M., Nehrkorn, T., 1993. *Short-term Cloud Forecast Scheme using Cross Cor- relations*. Weather Forecast. 84, 401411.

[3] Su, Jih-Jui, *Enhanced ACCA Algorithm*, Space Imaging Corporation Technical Memo IT81-LSD-SA&E Memo 274, 1984

[4] Markham, B.L. & Barker, J.L.. (1986). *Landsat MSS and TM Post Calibration Dynamic Ranges, Exoatmospheric Reflectance and at-satellite Temperatures. Landsat MSS and TM Post-Calibration Dynamic Ranges*, Exoatmospheric Reflectances and At-Satellite Temperatures. 3-8.

[5] Masek, Jeffrey & Vermote, E & Saleous, Nazmi & Wolfe, Robert & Hall, Forrest & F Huemmrich, Karl & Gao, Feng & Kutler, Jonathan & Lim, Teng-Kui. (2006). *A Landsat Surface Reflectance Data Set for North America*, 19902000. Geoscience and Remote Sensing Letters, IEEE. 3. 68 - 72. 10.1109/LGRS.2005.857030.

[6] Irish, R. (2000). *Landsat-7 automatic cloud cover assessment algorithms for multispectral, hyperspectral, and ultraspectral imagery.* The International Society for Optical Engineering, 4049, 348355.

[7] Irish, R., Barker, J. L., Goward, S. N., & Arvidson, T. (2006). *Characterization of the Landsat-7 ETM+ Automated Cloud-Cover Assessment (ACCA) algorithm.* Photogrammetric Engineering and Remote Sensing, 72(10), 11791188.

[8] Wang, B., Ono, A. Muramatsu, K., and Fujiwarattt, N., 1999. *Automated detection and removal of clouds and their shadows from landsat TM images*, IEICE Transactions on Information and Systems E82-D, no. 2: 453-460

[9] K. Hall, Dorothy & A. Riggs, George. & Salomonson (2001). *Algorithm Theoretical Basis Document (ATBD) for the MODIS Snow and Sea Ice-Mapping Algorithms*.

[10] Wildt, M. D. R. D., Seiz, G., & Gruen, A. (2007). *Operational snow mapping using multitemporal Meteosat SEVIRI imagery*. Remote Sensing of Environment, 109, 2941.

[11] T. Hashimoto, and Y. Nagakura, 2011. *Prediction of Output Power Variation of Solar Power Plant by Image Measurement of Cloud Movement*, Journal of Advanced Research in Physics 2(2).

[12] Vermote, E., & Saleous, N. (2007). *LEDAPS surface reflectance product description-Version 2.0.*, Technical Document, Departement of Geography, University of Maryland. USA.

[13] Soille, P. (1999). Morphological image analysis: Principles and applications (pp. 173174). Springer-Verlag.

[14] Zhang, Y., Rossow, W. B., Lacis, A. A., Oinas, V., & Mishchenko, M. I. (2004). *Calculation of radiative fluxes from the surface to top of atmosphere based on ISCCP and other global data sets: Refinements of the radiative transfer model and the input data*, Journal of Geophysical Research: Atmospheres, 109(D19). ISO 690

[15] Le Hgarat-Mascle, S., & Andr, C. (2009). *Use of Markov random fields for automatic cloud/shadow detection on high resolution optical images*, ISPRS Journal of Photogrammetry and Remote Sensing, 64(4), 351-366.

[16] Zhu, Z. and Woodcock, C. E., *Object-based cloud and cloud shadow detection in Landsat imagery*, Remote Sensing of Environment (2012), doi:10.1016/j.rse.2011.10.028

[17] Zhu, Z. and Woodcock, C. E., *Improvement and Expansion of the Fmask Algorithm: Cloud, Cloud Shadow, and Snow Detection for Landsats 4-7, 8, and Sentinel 2 images*, Remote Sensing of Environment (2014) doi:10.1016/j.rse.2014.12.014

[18] Qiu S., He B., Zhu Z., et al. *Improving Fmask cloud and cloud shadow detection in mountainous area for Landsats 48 images*, Remote Sensing of Environment (2017), doi.org/10.1016/j.rse.2017.07.002

[19] Farnebck, Gunnar. (2003). *Two-Frame Motion Estimation Based on Polynomial Expansion*. Image analysis. 2749. 363-370. 10.1007/3-540-45103-X_50.

[20] D. Frantz, E. Ha, A. Uhl, J. Stoffels, and J. Hill, *Improvement of the Fmask algorithm for Sentinel-2 images: Separating clouds from bright surfaces based on parallax effects*, Remote Sensing of Environment, vol. 215, pp. 471481, 2018.

[21] Qiu, S., Lin Y., Shang R., Zhang J., Ma L., and Zhu Z., *Making Landsat Time Series Consistent: Evaluating and Improving Landsat Analysis Ready Data*, Remote Sensing (2019), doi.org/10.3390/rs11010051

[22] Fmask 4.0: Improved cloud and cloud shadow detection in Landsats 4-8 and Sentinel-2 imagery, Remote Sensing of Environment. accepted - pending publication

[23] Alisson R. Silva, Maury M. Gouvła, Lus F.W. Ges, Carlos A.P.S. Martins, *A parallel implementation of a cloud dynamics model with cellular automaton*, Mathematics and Computers in Simulation, Volume 154, 2018, Pages 65-93, ISSN 0378-4754, https://doi.org/10.1016/j.matcom.2018.05.020.

[24] Rojas, Raul. (1996). *The Backpropagation Algorithm*. 10.1007/978-3-642-61068-4_7.

[25] Fukushima, K. *Neural network model for a mechanism of pattern recognition unaffected by shift in position - Neocognitron*, Trans. IECE, J62-A(10):658665, 1979.

[26] Fukushima, K. *Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position*, Biological Cybernetics, 36(4):193202, 1980.

[27] Fukushima, K.*Increasing robustness against background noise: visual pattern recognition by a Neocognitron*, Neural Networks, 24(7):767778, 2011.