ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

MSc THESIS

Bole Wilfried TIENIN

CLOUD COVERAGE PREDICTION WITH DEEP LEARNING METHODS

DEPARTMENT OF COMPUTER ENGINEERING

ADANA-2018

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

CLOUD COVERAGE PREDICTION WITH DEEP LEARNING METHODS

Bole Wilfried TIENIN

MSc THESIS

DEPARTMENT OF COMPUTER ENGINEERING

We certify that the thesis titled above was received and approved the award of degree of the Master of Science by the board of jury on

……..	…….	……..
Assist. Prof. Dr. B. Melis ÖZYILDIRIM	Prof. Dr. S. Ayşe ÖZEL	Assist. Prof. Dr. A. Kamil DEMIR
SUPERVISOR	MEMBER	MEMBER

This MSc Thesis is written at the Department of Institute of Natural And Applied Sciences of Çukurova University.
Registration Number:

Prof. Dr. Mustafa GÖK
Director
Institude of Natural and Applied Science

ABSTRACT

MASTER THESIS

CLOUD COVERAGE PREDICTION WITH DEEP LEARNING METHODS

Bole Wilfried TIENIN

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Assist. Prof. Dr. Buse Melis ÖZYILDIRIM
Year: 2018, Pages: 81
Jury : Prof. Dr. Selma Ayşe ÖZEL
: Assist. Prof. Dr. Alper Kamil DEMIR
: Assist. Prof. Dr. Buse Melis ÖZYILDIRIM

In this work, cloud images classification and segmentation have been implemented by using Deep Learning techniques. All images utilized in this thesis were obtained from TUBITAK National Observatory Bakirliktepe Campus, Cukurova University Space Science and Solar Energy Research and Application Center, and Singapore Whole-sky Imaging CATegories database. The main goal of our study is to compare the traditional image processing results with the Deep learning techniques. Firstly, two classes' classification solutions have been used. Our goal was to separate the cloud images from the others. We highlighted on Convolutional Neural Network (CNN) as classification and on SoftmaxWithLoss for the prediction. During our training phase, 92% accuracy has been scored as after fine-tuning our model. Secondly, some image processing techniques have been used to detect and segment the cloud images. Some edge detectors and watershed techniques have been implemented. Thirdly, some segmentation solutions have been proposed. Fully Convolutional Network (FCN) and U-NET have been used in this thesis. Two different methods of Deep Learning have been proposed to segment the cloud images and then make the prediction. Using U-NET model for the segmentation, 87% as dice coefficient and 45% as loss have been scored during the testing step. Different learning techniques were implemented on FCN such as stochastic gradient descent, Adam's momentum technique and Nesterov's momentum technique. The highest segmentation accuracy on FCN has been 63.12% with Adam's momentum technique.

Keywords: Deep Learning, Cloud segmentation, Convolutional Neural Networks, FCN, U-NET.

ÖZ

YÜKSEK LİSANS TEZİ

DERİN ÖĞRENME İLE BULUTLULUK TAHMİNİ

Bole Wilfried TIENIN

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Dr. Öğr. Üyesi Buse Melis ÖZYILDIRIM
Yıl: 2018, Sayfa: 81
Jüri : Prof. Dr. Selma Ayşe ÖZEL
: Dr. Öğr. Üyesi Alper Kamil DEMIR
: Dr. Öğr. Üyesi Buse Melis ÖZYILDIRIM

Bu çalışmada, derin öğrenme yöntemleriyle bulut görüntülerinin sınıflanması ve segmentasyonu uygulanmıştır. Tezde kullanılan tüm görüntüler TUBİTAK Ulusal Gözlemevi Bakırlıktepe Kampüsünden, Çukurova Üniversitesi Uzay Bilimleri ve Güneş Enerjisi Araştırma ve Uygulama Merkezinden, ve Singapur Tüm Gökyüzü Görüntüleme Kategorileri veritabanından elde edilmiştir. Bu çalışmanın temel amacı, geleneksel görüntü işleme yöntemlerini derin öğrenme teknikleri ile kıyaslamaktır. İlk olarak, iki sınıflı sınıflandırma çalışması yapılmıştır. Bu çalışma ile bulut resimleri ile bulut içermeyen resimlerin ayrıştırılması amaçlanmıştır. Tahmin ve sınıflama işlemlerinde Konvolusyonel Sinir Ağlar ve SoftmaxWithLoss fonksiyonundan yararlanılmıştır. Eğitim aşamasında, modelin ince ayar ile eğitiminden sonra %92 oranında doğruluk elde edilmiştir. Daha sonraki aşamada, bulut görüntüleri üzerinde görüntü işleme teknikleri kullanılarak tespit ve segmentasyon işlemi yapılmıştır. Bu aşamada, kenar tespit yöntemleri ve watershed algoritması uygulanmıştır. Üçüncü aşamada ise segmentasyon için Tam Konvolüsyonel Ağlar (TKA, FCN) ve U-NET derin öğrenme yöntemleri uygulanmıştır. İki farklı derin öğrenme yöntemi ile bulut görüntülerin segmentasyonu sağlanmıştır. U-NET ile eğitim aşamasında Dice katsayısı olarak %87, kayıp olarak ise %45 bulunmuştur. FCN ağı ise stokastik eğim düşüm, Adam's momentum ve Nesterov momentum teknikleri olmak üzere farklı eğitim yöntemleri uygulanmış ve en yüksek başarı Adam's momentum tekniği ile %63.12 olarak elde edilmiştir.

Anahtar Kelimeler: Derin Öğrenme, Bulut Segmentasyonu, Konvolusyon Sinir Ağları, FCN, U-NET.

EXTENDED ABSTRACT

Over the past few years, many machine learning topics have been studied, and many methods and solutions have been released to solve some specific issues. Our goal in this work is to show the best way to conduct the classification and segmentation of cloud images. Achieving this goal required the usage of several images. Most of them have been provided by TUBITAK National Observatory. In addition, cloud images from Cukurova University Space Science and Solar Energy Research and Application Center (Assoc. Prof. Dr. Nazım Aksaker Ç.U Uzaymer) and Singapore Whole-sky Imaging CATegories database have been used for our final solution. From these data, the tasks of cloud images classification and segmentation using image processing techniques and Deep Learning techniques have been implemented. Firstly, two classes' classification solution has been implemented. Our goal is to separate the cloud images from the others. To reach this goal CAFFE framework has been used. With CAFFE, we have built our own Convolutional Neural Network (CNN). Then this CNN (for the classification) and the SoftmaxWithLoss (for the prediction) have been used. Two different CNN models, training from scratch and fine-tuned. For this task, 995 images have been used (680 for training and 315 for testing). During our training phase, 67% and 92% (respectively scratch training and fine-tuning training) have been scored as accuracy. During our testing phase, we have tried to classify correctly into two different classes. The class (0) belongs to cloud images and the class (1) belongs to other images different than cloud images (non-cloud). After using the model from scratch training for the testing phase, out of 315 images, 14 images have been misclassified by our model and 301 images have been classified correctly. After using the fine-tuning, 314 images have been correctly classified and our model has misclassified one image. With our model from scratch, 95% have been scored as

testing accuracy and with our model from the fine-tuning 99% have been scored as testing accuracy.

Secondly, some image processing techniques have been used to detect and segment the cloud images. Among the entire image processing techniques, Otsu technique and some edge detectors such as Canny, Roberts, Prewitt, Sobel, Gradient magnitude and watershed techniques have been implemented in this study.

Thirdly, some segmentation solutions have been proposed. FCN and U-NET have been used in this thesis. We proposed two different methods of Deep Learning to make the segmentation and then make the prediction of the cloud images. For this task 913 images have been used (613 for the training and 300 for the testing).

Using U-NET 4 different models of U-NET have been implemented. Each of them has different input shape (U-NET 128×128, U-NET 256×256, U-NET 512×512, U-NET 1024×1024). These models have been trained twice with 20 epochs for the training from scratch, and 10 epochs for the fine-tuning training. Among all of the training, our fine-tuning models have shown better results than these one from scratch. Indeed, 87% have been scored as the highest training accuracy and 45% as the lowest loss. We also observed that the longest training model was U-NET 1024×1024 with 2 days 23 hours 22 minutes for the training from scratch and 1 day 11 hours 47 minutes for the fine-tuning training.

FCN has been utilized for segmentation. Different learning methods have been implemented for segmentation problem such as Stochastic gradient descent, Adam's optimizer, and Nesterov's optimizer. According to the test results, the best segmentation accuracy, 63.12% has been achieved by using Adam's optimization.

ACKNOWLEDGMENTS

I would like to give special thanks, warmth, and appreciation to some people and institutions below who have made my research successful and assisted me at every single point to reach my goal:

To my Supervisor, Assist. Prof. Dr. Buse Melis ÖZYILDIRIM for her vital support, patience, and assistance. Her encouragements and enthusiasm made it possible for me to achieve my goal. Accomplishing this final product without Assist. Prof. Dr. Buse Melis ÖZYILDIRIM was impossible.

To the members of jury Assist. Prof. Dr. Buse Melis ÖZYILDIRIM, Prof. Dr. Selma Ayşe ÖZEL and Assist. Prof. Dr. Alper Kamil DEMIR for their suggestions and corrections.

To the Turkish Government through Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı (YTB) which granted me a scholarship to study my master degree.

To TÜBİTAK National Observatory, Cukurova University Space Science and Solar Energy Research and Application Center (Assoc. Prof. Dr. Nazım Aksaker Ç.U Uzaymer) and Singapore Whole-sky Imaging for providing me the data.

To Cukurova University staff members and the Computer Engineering Department, whose services turned my researches into a success.

To mom, dad, my sisters, and friends, without whom I wouldn't have reached this level; they didn't only assist me financially, they also extended their support morally and emotionally.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF ABBREVIATIONS

AdaGrad	: Adaptive Gradient
AI	: Artificial Intelligence
BAIR	: Berkeley AI Research
CAFFE	: Convolutional Architecture for Fast Feature Embedding
CBBT	: cloud-based brightness temperatures
CCM	: cross-correlation method
CNN	: Convolutional Neural Networks
Conv	: Convolution
CPU	: Central Processing Unit
DIGITS	: Deep Learning GPU Training System
DNN	: deep neural networks
Feat	: Feature
FCN	: Fully Convolutional Networks
GPU	: Graphics Processing Unit
LMDB	: Lightning Memory-Mapped Database
RCNN	: Regions with Convolutional Neural Networks
RLE	: Run length encoding
RMSProp	: Root Mean Square Propagation
SGD	: Stochastic Gradient Descent
SWIM-CAT	: Singapore Whole-sky IMaging CATegories database
VOF	: Variational Optical Flow

1. INTRODUCTION

As human beings, we usually highly care about our happiness and time. But in the context of make our life easy, we also care about another precious resource, named energy. Scientists have set more than one method and techniques in order to produce this energy. Through the studies of the great scientists, we can to collect energy from atoms, wind, solar, etc. These energies are respectively called, nuclear energy, wind power, and solar energy. Today scientists are still developing techniques to optimally collect the solar energy. In order to optimize the production of solar energy, some of them decided to study the clouds present in the sky.

Clouds are essential for the world. They can have a positive or negative effect on solar radiation and precipitation. To understand well our topic, we have to define what cloud coverage is. From the definition, we can say that, Cloud coverage or Cloudiness or Cloud motion is related to the portion of the sky obscured by clouds when observed from a specific position. Through this definition, we have been able to notice that there is more than one term to describe the position of cloud in the sky. The presence of clouds plays a crucial role in Satellite images analysis. Through the clouds and land Elevation, NASA's Ice Satellite (ICESat) demonstrated that 70% of the world's atmosphere is covered with clouds (Dev, et al., 2016). In our future studies, we are planning to use machine learning techniques to measure and predict the quantity of clouds in the sky. According to the previous studies, Cloudless is an open source project of computer vision. It uses a pipeline system for orbital satellite data, and it is also powered by data from Planet Labs and can use deep learning solutions. In (Jedlovec, 2010) and (Neuberg, 2016), the goal of Cloudless was to focus on applying deep learning techniques to detect clouds in Planet Labs' satellite

imagery. In other words, cloudless focuses on cloud detection and localization, while pipeline just focuses on the visual orbital detection and localization task.

Moreover, IBM is also working in the field of cloud cover. In 2013, IBM's research division, using Department of Energy funding, decided to work on how to benefit from artificial intelligent engine Watson for clean power. Today IBM Research had almost 200 partners that use its solutions (solar and wind forecasting technology). These solutions can predict solar and wind conditions from between 15 minutes up to 30 days. By the way, IBM and the University of Michigan have a joint project related to cloud coverage. They designed a solar car which predicts cloud coverage in the real-time (IBM research, 2015).

Furthermore, in (Chow, et al., 2015), short-moment variability in power obtained from solar energy creates challenges for power system engineers and operators. The highly predictable diurnal and annual irradiance pattern aside, clouds have the most substantial impact on solar energy production. In fact, the problem that we are facing here is to find a good accuracy of cloud coverage that will allow us to get more energy through the solar panel. Sometimes the obtained weather prediction information or satellite cloud images are not the expected. For more accuracy of irradiance prediction, research that analyzes images obtained from devices capturing skies were developed. Ground-based sky camera systems have been used to capture the images of the sky, which allows some researchers to study the relationship between the sun and clouds and the effect of clouds (Chen and Lin, 2016). In (Baslar, 2012), automatic cloudiness analysis system was proposed. Image processing approaches have been applied to images obtained from an all-sky camera located at TUBİTAK National Observatory. Several approaches have been compared to calculate sky coverage from day and night images. There are also studies that use fish eye camera to detect clouds (Pfister, et al., 2003) and (Heinle, et al., 2010).

In the past few years, cloud coverage calculation approaches were based on temperature differences between the sky and the earth. In (Clay, et al., 1999), cloud detector was developed utilizing infrared sensors known as thermopile. A thermopile is a set of thermocouples used to measure small quantities of radiant heat. Thermocouple is a thermoelectric device that is used to measure temperature, it consists of two wires of different metals connected at two points, and then a voltage is generated between the two junctions in proportion to the temperature difference (Kartal, 2006). There are also commercial cloud detectors using thermocouples. This kind of systems cannot locate clouds and provide exact information about its amount.

In this study, deep learning-based cloud segmentation methods have been implemented and tested on TUBITAK National Observatory's all-sky images, images from Cukurova University Space Science and Solar Energy Research and Application Center (Assoc. Prof. Dr. Nazım Aksaker Ç.U Uzaymer) and from Singapore Whole-sky Imaging CATegories database compared with traditional image processing methods.

2. LITERATURE REVIEW

In this section of the study, some previous related works to our topic have been described. As we know, few studies were proposed to describe how to use deep learning methods to predict the cloud coverage. Indeed, some studies are using other algorithms and methods belonging to machine learning and other fields, which can be useful for our study.

2.1. Variational optical flow (VOF) technique

The variational optical flow is a technique used to estimate cloud motion and stability, through ground-based sky images. In this study, some images have been captured by a sky imager at UC San Diego. They were studied and then after the VOF algorithm was applied. Some other methods like cross-correlation method (CCM) and image persistence method have been used to compare the different obtained results.

Variational optical flow forecast: The main theory of optical flow is that an image pixel value does not change over consecutive frames, but just the position is shifting. We can mathematically explain it through this equation (Chow, et al., 2015):

$$I_t(x_t, y_t) = I_{t+1}\big(x_t + u_t(x_t, y_t), y_t + v_t(x_t, y_t)\big) \quad (2.1)$$

$\boldsymbol{I}$, is pixel values, usually (R, G and B) or grayscale intensity.

$\boldsymbol{x_t}$ and $\boldsymbol{y_t}$ represent the Cartesian pixel indices.

$\boldsymbol{u_t(x_t, y_t)}$ and $\boldsymbol{v_t(x_t, y_t)}$ represent the motion vector components for pixel $\boldsymbol{(x_t, y_t)}$ in frame $\boldsymbol{t}$.

The goal is to calculate the optical flow field $\{\boldsymbol{u_t(x_t, y_t)}, \boldsymbol{v_t(x_t, y_t)}\}$ between two successive frames of an image sequence. The brightness constancy equation is a nonlinear equation in $\boldsymbol{u}$ and $\boldsymbol{v}$. To simplify the nonlinear equation and solve for the optical flow field, the equation is linearized using the first order Taylor expansion and leading us to the well-known optical flow constraint (OFC) equation:

$$0 = \frac{\partial I}{\partial t} + \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v. \quad (2.2)$$

The OFC is one of the underdetermined systems because of the number of unknowns. In the above equation, we have two unknown terms, u and v**.** To solve this equation, we have to add some constraints. Some methods such as the local method of Lucas and Kanade (1981) etc., were proposed to solve these kinds of equations. After the computing of those mathematical formulae, in order to shift the cloud map using the motion vector, the optical flow method has to get the pixel positions with a heterogeneous flow field $\boldsymbol{u_t(x_t, y_t)}$ and $\boldsymbol{v_t(x_t, y_t)}$**.**

Cloud forecast metric: to evaluate the VOF forecast we convert our captured images in binary, then we transform them to Cartesian coordinates (cloud map). We have also applied the cross-correlation method (CCM) to the cloud map that is why we have been able to obtain the nowcast (i.e., 0min forecast). In fact, the nowcast is calculated by shifting the cloud map at the time:

$$t_0 - dt(dt = 30s). \quad (2.3)$$

Point trajectories and forecast confidence: Based on (Sundaram et al., 2010) we can develop an optical flow tracker in order to obtain point trajectories. To start the tracking process, we initialize some tracking points (sub-sampled) from

every twenty pixels of the first frame. Each of the points is tracked using the optical flow field:

$$(x_{t+1}, y_{t+1}) = (x_t, y_t) + \big(u_t(x_t, y_t), v_t(x_t, y_t)\big). \quad (2.4)$$

The following rules are used to stop the tracking of a point (C.W.Chow et al., 2015):

1. Point is advected out of the forecast domain.
2. Forward and backward optical flow yield inconsistent results (Sundaram et al., 2010). We stopped the tracking if the inconsistency is larger than a threshold, which varies as a linear function of motion magnitude.
3. The image structure around the trajectory point decreases. The local image structure can capture the dynamics of the cloud evolution as, for example, cloud evaporation decreases the local RBR gradient. However, image structure can also decrease due to measurement errors or optical effects; for example, clouds moving into the solar region appear to have less structure due to pixel saturation.

In this literature, motion estimation and point trajectories were described to determine the accuracy of variational optical flow (VOF) forecasts and cloud stability. For 0,5,10, and 15 minutes the VOF forecast offer an average error reduction of 39%, 21%, 19%, and 19% respectively.

2.2.Measuring Cloud Cover and Brightness Temperature with a Ground-Based Thermal Infrared Camera

In this literature, a thermal infrared camera has been used to observe clouds and the sky at high spatial and temporal resolutions. This study proves that cloud cover can be detected in the day and the night over a field of view extending to zenith angles. By using a new parameterization for the variation of sky brightness temperature with zenith angle the cloud-based brightness temperatures (CBBTs) can be calculated. In (Smith and Toumi, 2008) with the special ground-based camera, the blackbody has been used for the calibration and to avoid the standard deviation of the pixel reading from the captured images of the clouds. The goal of this algorithm is to establish a threshold value for cloudy/noncloudy pixels at any angle in the following way: the radiance (I) coming on a pixel at a given angle when a thin cloud passes over it can be written as:

$$I = (1 - \tau_S)I_{IS} + \tau_{IS}(1 - \tau_C)I_C + \tau_{IC}\tau_C(1 - \tau_{US})I_{US}. \qquad (2.5)$$

With$\boldsymbol{I_{IS}}$,$\boldsymbol{I_{US}}$, and $\boldsymbol{I_C}$ representing the blackbody radiances from the sky below the cloud, the sky above the cloud, and the cloud itself, and $\boldsymbol{\tau_{IS}}$, $\boldsymbol{\tau_{US}}$, and $\boldsymbol{\tau_C}$ representing their transmittances. The cloud cover in the image is computed as the fraction of pixels above the threshold line.

2.3 Cloudless (Open Source Deep Learning Pipeline for Orbital Satellite Data)

Planet Labs is a startup that built nanosats. They have launched eighty of their nanosats, named Doves, into Low Earth Orbit. Doves are built from commercially available hardware and can be launched into orbit quickly. The cost of that satellite is much lower and can allow very fast iteration in what is known as “agile aerospace.” Through the project Cloudless, Planet Labs team focused on automated cloud detection using pipeline as a method. However, even if the

Cloudless' pipeline is currently working on cloud detection and localization, the entire pipeline works on what would be useful for any visual orbital detection and localization task (Neuberg, 2016). So, in this section some simple principles belong to the pipeline have been described:

1. A pipeline is made of nodes which have an expected input and output type.
2. We can only fit together nodes if the output type of the first node matches the input type of the second node.
3. Pipelines themselves can be thought of as "nodes" and can thus be composed to form new pipelines.
4. Wherever possible, nodes should take RDDs as input and produce them as output. This encourages node developers to think in parallel data terms.

The approach of Cloudless consisted of three pieces (Neuberg, 2016):

- An annotation tool that takes data from the Planet Labs API and allows users to draw bounding boxes around clouds to bootstrap training data.
- A training pipeline that takes annotated data runs it on EC2 on GPU boxes to fine tune a neural network model using CAFFE and then generates validation statistics to relate how well the trained model performs.
- A bounding box system that takes the trained cloud classifier and attempts to draw bounding boxes on orbital satellite data, in this case, clouds.

For the first training with cloudless, 700 images from San Francisco Bay Area were used. The final accuracy was 62.5% for cloud detection. Then the second training with a new parameter (RapidEye, which can detect in the near infrared) has been done. During this training, 4000 images from the same location were used. The final accuracy was 89.69% of cloud detection.

3. MATERIALS AND METHODS

In this section, the frameworks, data, and methods used are explained.

3.1 Materials

3.1.1 Data Collection

The Data used in this thesis were provided by all-sky cameras located at TUBITAK National Observatory. Digital camera system providing all-sky images during day and night was established at TÜBİTAK National Observatory Bakırlıktepe Campus. Images are obtained with Canon EOS5D DSLR camera with fisheye lens (Baslar, 2012). From these data images, we were able to design our database for the implementation of our solution on CAFFE.

We also got some data from Cukurova University Space Science and Solar Energy Research and Application Center. Indeed, theSky Camera at UZAYMER displays in real time and saves images that are analyzed with special software written in IDL to investigate sky events such as cloudiness and Meteor.

We also used SWIM-CAT (Singapore Whole-sky IMaging CATegories database), SWIMSEG and SWINGSEG. These databases provided us images captured by WAHRSIS, a calibrated ground-based whole sky imager (Dev, et al., 2014). We used some patches of cloud categories from images that were captured in Singapore during January 2013 up to December 2016. The SWIMCAT, SWIMSEG, and SWINSEG database are available for downloading from http://vintage.winklerbros.net/resources.html.

From these data, 680 images have been used in the classification training step. The images were resized (227×227) and saved under lmdb database format. Then 315 images have been used in the classification testing step. For the segmentation part, 613 images have been used during the training. The images

were firstly labeled, using a web tool named LABELME. Secondly the images were preprocessed (RLE as compressing system, uint8 has been used, resized to 500×500, and converted from RGB to BGR). Then 300 images have been used for the testing step. In total 1908 images have been used in this study.

3.1.2. Frameworks

3.1.2.1. CAFFE framework

CAFFE is a deep learning framework. CAFFE stands for Convolutional Architecture for Fast Feature Embedding. It was developed by Berkeley AI Research (BAIR) and by a community of some contributors. Yangqing Jiaduring his PhD at UC Berkeley created the project. CAFFE is also under the BSD 2-Clause license. To perform image analysis like Convolutional Neural Networks (CNN) and regional analysis of images like Regions with Convolutional Neural Networks (RCNN) CAFFE is one of the best frameworks. The main advantage of the CAFFE framework is the fact of having a wide repository of pre-trained neural network models and a lot of image classification tasks, called the Model Zoo. In this study, CAFFE is one of our chosen deep learning frameworks. This choice was guided by the following reasons mentioned below in the table that we got after the comparative study between the five best framework.

3.1.2.2. KERAS

KERAS is another popular framework frequently used in deep learning studies. François Chollet, who is one of Google engineer, developed and maintained KERAS. Using KERAS rather than the other deep learning framework give the following advantages:

- Modularity: Models can be seen as sequences or graphs.

- Minimalism: Possible outcome, no frills and maximizing readability
- Extensibility: Easy to add new components such us run it on top of other frameworks (THEANO and TENSORFLOW)
- Python: Native of Python (same model files and formats)

In this study, KERAS was used to backend TENSORFLOW and also because of CPU and GPU computing. Indeed, KERAS has been used to define and create the training models.

3.1.2.3. Nvidia Digits

Over the past decade, deep neural networks (DNNs) have become much more popular in the machine learning (image classification (Krizhevsky et al. 2012), object detection (Szegedy et al. 2013), image segmentation (Long et al. 2014) etc). Diverse open-source frameworks exist for the implementation of CNNs. CAFFE, TENSORFLOW, TORCH and THEANO are the most popular ones. They performed on the newest and fastest computational techniques and are used by the engineers who are comfortable with command line tools. Several projects have released visualization tools for these frameworks, but none of them haven't released a full training solution yet. DIGITS provides a web interface for deep learning, avoiding direct interaction with the underlying frameworks. DIGITS still exposes the details of optimization and network architecture even when we need to design our own model. System resources are shared to several concurrent training jobs (which enables rapid prototyping), and the status of each job can be shared with collaborators through the web interface. The database for training and validation can be created easily on DIGITS. A model can be trained on the dataset and tested in several ways. A RESTful web application can be created using the Flask python web framework. That can permit to the users to create and delete both datasets and models through a web page. To create a model, some standard

networks are already available: LeNet-5 (Lecun et al. 1998), AlexNet (Krizhevsky et al. 2012) and GoogLeNet (Szegedy et al. 2014). Among all of these networks which already exist we can also decide to create a custom network. We can start from a template, and then with a visualization tool we can have an overview of the network. DIGITS provides also different options for data transformation such as mean subtraction, random cropping, etc, and gradient descent optimization such as batch size, learning rate, etc. During the training, the validation metrics of accuracy and loss can be seen graphically through JavaScript charts which update them in real-time. The model can also be tested during the training. After the training, the necessary files to deploy the model can be downloaded.

Table 3.1. Comparative table of Frameworks

Software	**Code and Models**	**Performances**	**Model Deployment**	**Creator**
TENSORFLOW	A lot of codes and models provided by Google	TENSORFLOW supports C++, and Python. Also CPU, GPU computing and even horizontal scaling using gRPC.	Firstly, export the model Secondly, create a Docker container with the model And finally, deploy the model with Kubernets into a Google cloud or Amazon AWS	Google Brain team
THEANO	A lot of codes to begin Loss of momentum and	Good in numerical computations optimization.	Need Python for model deployment	University of Montreal

	less recent models Less pretrained models New codes are sometime unstable	It doesn't have multi-GPU support nor horizontal capabilities		
KERAS	Can use Models from TENSORFLOW, THEANO, KERAS and CAFFE	KERAS is a high-level library that can work on top of TENSORFLOW or THEANO	Docker &Google Container Registry and Flask Front-end Web server	François Chollet (now at Google)
TORCH	Classification models available in ModelZoo Can convert CAFFE models to TORCH	No need to compile the models. Some layers aren't efficient because of inner buffer	Require Lua	Ronan Collobert, Koray Kavukcuoglu, Clement Farabet
CAFFE	A lot of code online A lot of Pretrained models Able to fine tune networks	Fast training, Testing and Validation Multi GPU Python or Matlab Can be used as interfaces	Based on C++ Can be compiled on different devices	Berkeley Vision and Learning Center

3.2. Methods

3.2.1. Deep Learning

First of all, what is “Learning”? Learning can be defined as a kind of skill acquired by studying. It can also take the meaning of knowing or having the experience. Having the knowledge or the expertise literally means that we have learnt from someone or something. Got experience or knowledge is two qualities of human being. Most of the time just humans have these abilities, machines do not have them. Since we have been able to define the term learning, we can say that machine learning refers to the field of computer science where machines are designed or programmed to act like humans. And then, machine learning is about learning to do better in the present or the future based on what was experienced in the past.

Concretely, Deep Learning is a technique of training allowing a machine, for example, to recognize the contents of an image or to understand the spoken language. Complex challenges that the community of researchers in artificial intelligence faced a long time ago. The technology of Deep Learning learns how to represent the world. In other words, it describes how the machine will represent the world or the image for example. Yann LeCun is one of the pioneers of Deep Learning. He is also one of the most influential researchers in the field. In the past, it was necessary to do it with the hand, to explain to the tool how to transform an image in order to classify it. Today, with Deep Learning, the machine learns how to do it by itself. And it does it much better than the engineers. To understand the Deep Learning, it is necessary to reconsider the supervised training, which is a standard technique in Artificial Intelligence (AI) and which allows the machines to learn. Concretely, for a program to learn how to recognize a cloud, for example, we simply need to train the program with a lot of images of clouds that we have labeled according to the different classes. The training, can require hours, even

days and in some cases weeks. Once involved, it can recognize clouds on new images.

3.2.2 Convolutional Neuron Network (CNN)

A CNN or, Convolutional Neural Network, is a deep learning technique and a powerful type of neural network used for image classification. One of the latest architecture of CNN was designed by Yann LeCun, one of the pioneers of Machine Learning (LeCun et al., 1998). CNN has several layers that process and transform an input to produce an output. We can train a CNN to do image analysis tasks like, scene classification, object detection and segmentation. That is why Google, Facebook, Snapchat and other companies which deal with images use convolutional neural networks. To understand well CNNs work, the main layers of this network have been clearly explained in the below section.

3.2.2.1. Convolutional Layer

In the convolutional layers, a kernel matrix has been passed over the input matrix to create a feature map. The dimensions of the kernel can also be defined to produce a different feature map, or to expand the data along one dimension while reducing its size along the other axes. Usually, the matrix values of the feature map are computed by applying the sum of the result from the multiplication of the kernel and an appropriately sized section of the input matrix. Another technique to improve CNNs is to use multiple kernels in a given convolutional layer and concatenate the results to create the feature map. The fact that one kernel is used for the entire image makes convolutional neural networks very location-invariant and prevents them from over fitting. Here is an example of a convolution:

We have an image of size 6*6 and we define a weight matrix of size 3*3, which will extract the feature map.

Input Image

18	54	51	239	244	188
55	121	75	78	95	88
35	24	204	113	109	221
3	154	104	235	25	130
15	253	225	159	78	233
68	85	180	214	245	0

Weight

1	0	1
0	1	0
1	0	1

Figure 3.1. Initialization of the Convolutional

Firstly, we have to initialize the weight matrix as we did in the above table (3*3). Secondly, the weight matrix will run across the image and make sure that all pixels are covered once in the time, and then give the convolved output.

429	505	686	856
261	792	412	640
633	653	851	751
608	913	713	657

Figure 3.2. Results from the first convolution

The 6*6 is now converted into a 4*4 image after the following convolution operation:

- **429=(18*1)+(54*0)+(51*1)+(55*0)+(121*1)+(75*0)+(35*1)+(24*0) +(204*1)**

- **505=(54*1)+(51*0)+(239*1)+(121*0)+(75*1)+(78*0)+(24*1)+(204*0)+(113*1)**
- **657=(235*1)+(25*0)+(130*1)+(159*0)+(78*1)+(233*0)+(214*1)+(245*0)+(0*1)**

As it is seen, the main target of the convolutional layer is to detect local conjunctions of features from the input matrix and mapping their appearance to a feature map (Le Cun et al., 2015). The image is split into perceptrons, as a result of convolution in neural networks, then creating local receptive fields and finally compressing the perceptrons in feature maps of size $A \times B$

3.2.2.2. Pooling Layer

Pooling is a kind of non-linear down sampling. The main aim of the pooling layer is to reduce the spatial size of the input image progressively. In addition, it reduces the number of parameters and computation in the network and it controls over fitting. There are many ways to implement pooling. Among all of these methods, we have the max pooling which is the most common one. Usually, we apply the pooling using filters of size 2x2 and using a stride of two. A pooling layer of size 2x2 with stride of two will shrink the given input image. Here is an example of two different pooling methods:

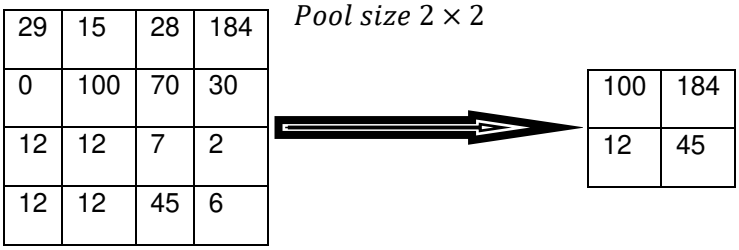

Figure 3.3. Max pooling

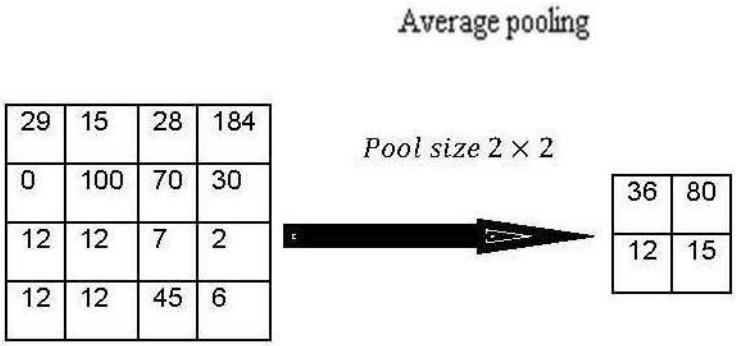

Figure 3.4. Average pooling step

For max pooling, we have chosen the highest pixel value. This intuitively corresponds to choosing the most prominent features.

For average pooling, we take the average of the pixel values. The average pooling produces smoother results than max pooling.

3.2.2.3. Fully Connected layer

The Fully Connected layer works like Multi-Layer Perceptron that uses a SoftMax activation function in the output layer. “Fully Connected” means that the neuron from the previous layer is connected to the other neuron from the next layer. The fully connected layers are trained to take the output from convolution and predict the correct output.

The below image illustrated perfectly a resume of the different steps that are applied to an image during the convolutional neural network (CNN).

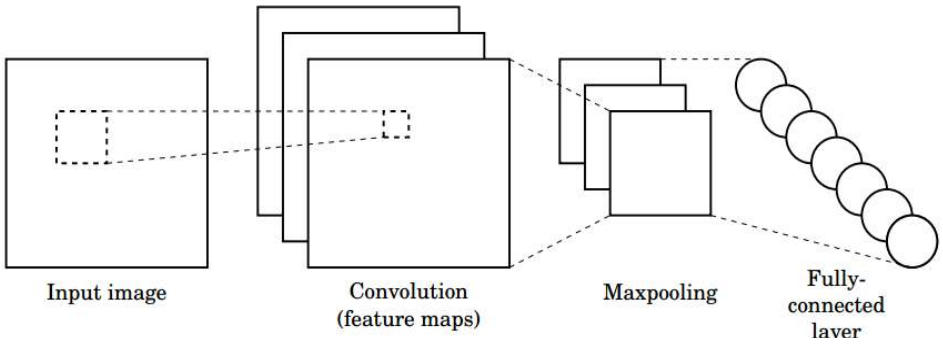

Figure 3.5. Fully Connected layer. (Viblo, 2017)

Here in this example, an input image with three kernels has been convolved. Then three feature maps have been created and maxpooled. At the end of this architecture, we can see that the output has been flattened into a fully connected layer, which will be used for training and prediction.

3.2.3. Classification using CNN

As we explained in the above section, the Convolutional layers + Pooling layers act as Feature Extractors from the input image, and the Fully Connected layer acts as a classifier. In this section, the different steps of the classification using CNN have been explained.

The traditional classification using CNN has two (02) phases: the training phase and the prediction phase. During the training phase, a convolutional neural network has been trained, using a dataset composed of cloud images and the corresponding labels of these cloud images. During the prediction phase, a trained model has been used to predict the labels of some test images.

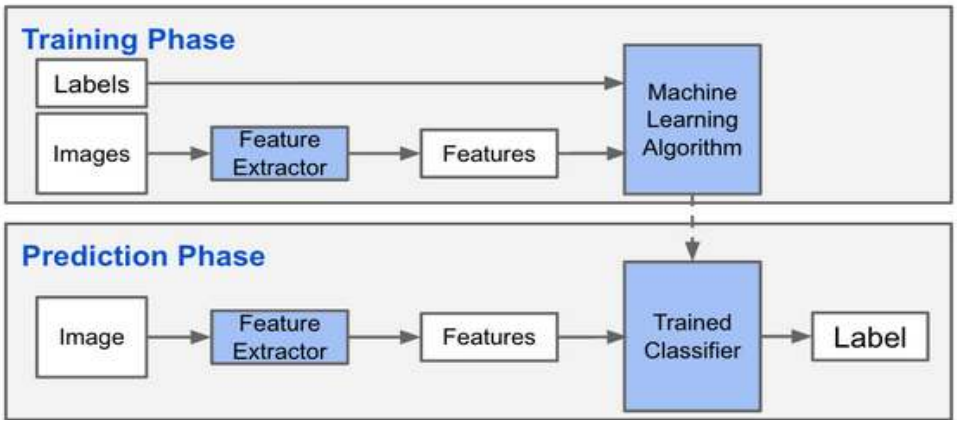

Figure 3.6. Training and Prediction Phase. (Moujahid, 2016)

The above image describes the different steps during a classification using machine learning algorithm. We can clearly see the two main phases that we have mentioned at the beginning of this section. In order to start the classification, we have to define our input images and their labels. From the images data, features will be extract and then while using the features and the labels, a machine-learning algorithm will be applied in order to generate a trained model. When we got the trained model, the next step of the classification can begin. While using our trained model, our convolutional neural network will be able to predict the correct label (giving the accuracy of each classes) from the test images that we have defined.

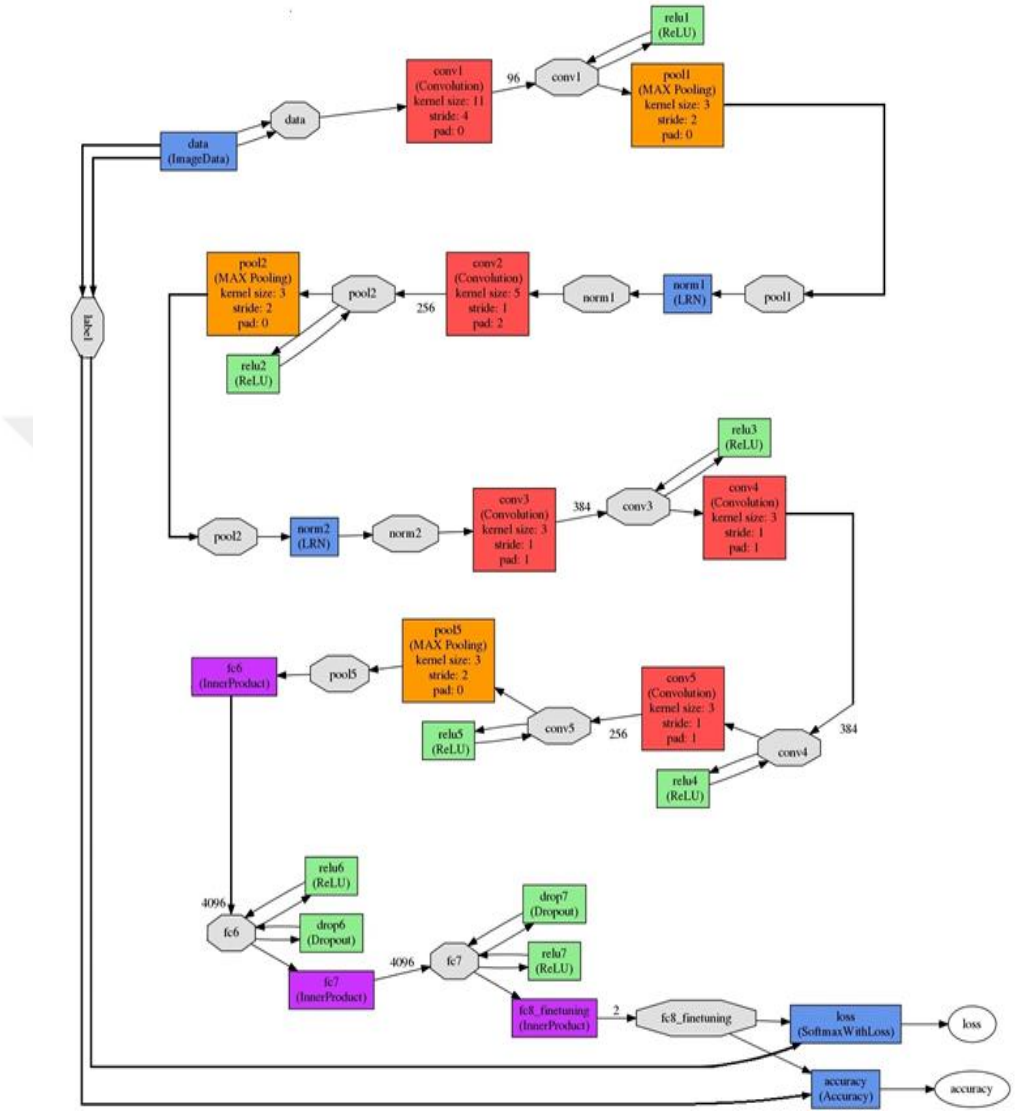

Figure 3.7. CNN architecture

The convolutional neural network in this architecture allowed us to classify input images into two classes: cloud and non-cloud. Initially, our model will store the “ImageData” defined as input and give us the class score as output. The architecture, illustrated above, is very deep. It has five convolutional layers, with

pooling, and three fully connected layers. It has also (ReLUs) and a new technique to reduce the over-fitting problems (Dropout).

3.2.4. Image Segmentation Techniques

Image segmentation is an essential and challenging process of Deep learning. Image segmentation is the technique of dividing an image into small parts called segments. The goal of segmentation is simplification. It means representing an image into meaningful and easily analyzable way. Image segmentation aims at dividing an image into several parts/segments in order to have similar features or attributes. The main applications of image segmentation are Medical image processing, Object detection, and Recognition Tasks, etc. There are diverse techniques for dividing the images into segments. All these techniques usually follow two basic approaches for the segmentation. The two approaches are region-based approach and edge-based approach. In this study, we used two deep learning frameworks to implement the segmentation of the cloud images.

- U-NET

U-net is encoder-decoder neural network architecture for image segmentation. The name of this architecture comes from its unique shape. In fact in this architecture, the feature maps from convolution part (in down-sampling step) are fed to the up-convolution part (in up-sampling step). U-net has been used extensively for biomedical applications to detect cancer, kidney pathologies and tracking cells etc. U-net is one of the most powerful image segmentation tools. Another advantage of using U-net is that it does not have any fully connected layers; therefore, we do not need to care about the size of the input image. U-net allows us to extract features from images. U-net allows us to work with small data and does not require a specific size for our input images. These abilities make U-net a powerful tool for image segmentation tasks.

During the training, input images and their corresponding annotated images (image masks) are used to train the network. Due to the unpadded convolutions, the output image is usually smaller than the input. Also, during the training, the dice coefficient is calculated using the pixel-wise SoftMax over the final feature map combined with the cross-entropy loss function. The dice coefficient is defined by:

$$\frac{2\times|X\cap Y|}{|X|+|Y|} \tag{3.1}$$

Where, X is the predicted image and Y is the target image (mask). $|X|$ is the cardinality of the set X (the number of elements in X) and respectively for $|Y|$. Finally, $\cap$ is the intersection between $X$and Y.

In our U-net model, the dice coefficient is a metric that measures the similarity between our input image and the output image mask. When our program does 20 epochs, we get the trained model and we see the dice coefficient and the loss.

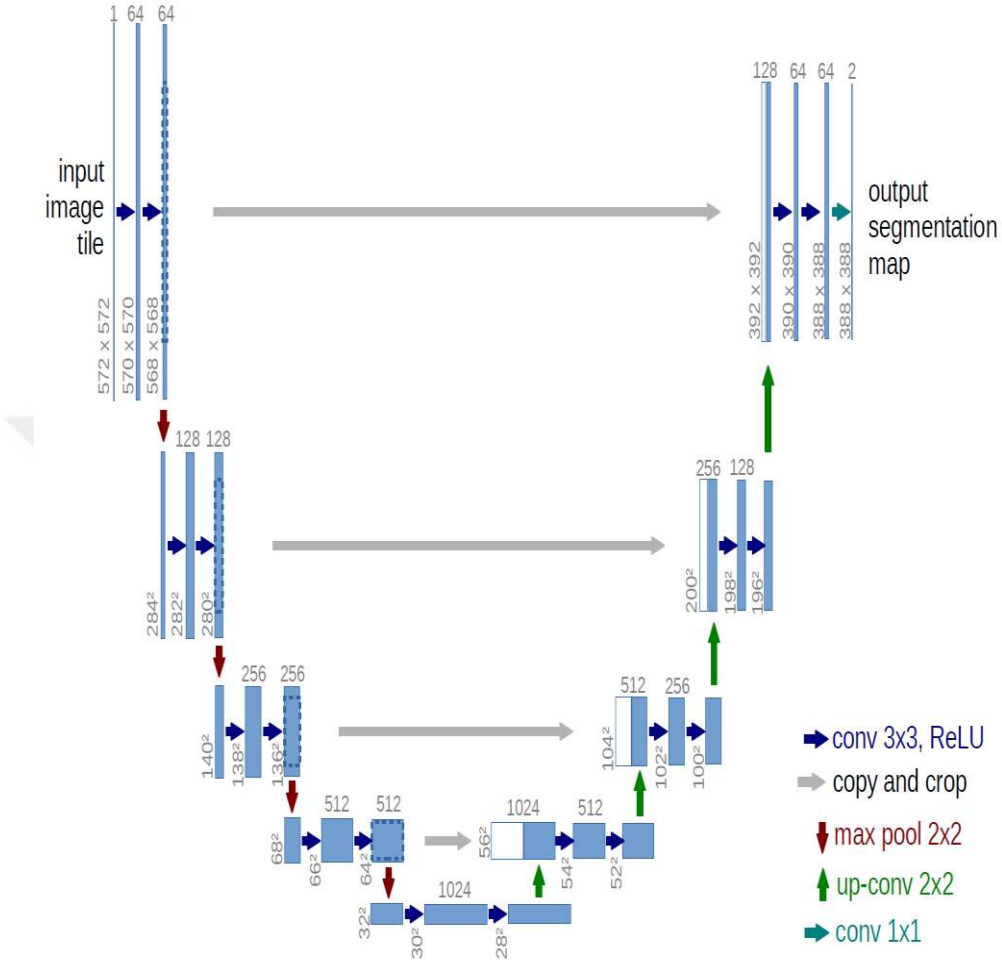

Figure 3.8. U-NET architecture. (Ronneberger, et al., 2015)

Here the blue boxes represent the multi-channel feature map. On top of each blue box, we have the number of the channels. Similarly, at the bottom of each blue box, we have their sizes (x-y). The white boxes correspond to the feature maps that have been copied during the expansion step. The arrows represent the different operations.

- ❖ FCN: FCN is a deep architecture providing information at pixel level, which is called semantic segmentation. It was proposed by (Long et al.,

2014) in 2014. Unlike classification architectures, in FCN there is not any fully connected layer. It takes arbitrary size inputs and generates output of corresponding size. This allows generating segmentation map (Long et al., 2014).

Input images are three-dimensional arrays with size (height, width and channel), layer calculations are the same as that of classification networks. Fully connected layers are transformed into convolutional layers. For this aim, 1x1 convolutional layers were utilized and for up sampling deconvolutional layer were used. Loss function in FCN is defined as sum over the spatial dimensions of the last layer. There exist three types of networks named as FCN-32, FCN-16 and FCN-8. These values denote amount of stride in deconvolutional structure of the FCNs. In FCN-32, 7^{th} convolutional layer has 1x1 kernel size, number of outputs is 4096 and it is followed by score layer which is another convolutional layer with 1x1 kernel size and number of outputs equals to number of objects in the image. Deconvolutional layer is applied. In this layer, kernel size is 64, stride is 32 and number of output equals to number of objects in the image. Unlike FCN-32, in FCN-16 there exist two deconvolutional layers. While in the first deconvolutional layer kernel size is four, stride is two; in the second deconvolutional layer, kernel size is 32 and stride is 16. In FCN-8, there exist three deconvolutional layers. Their kernel sizes are 4, 4, and 16; their strides are two, two, and eight, respectively. In FCN-16 and FCN-8 there is also additional fuse layers between deconvolution operations (Long, et al., 2014). In the image below, segmentation performances of FCN-32, FCN-16, and FCN-8 are shown (Long, et al., 2014).

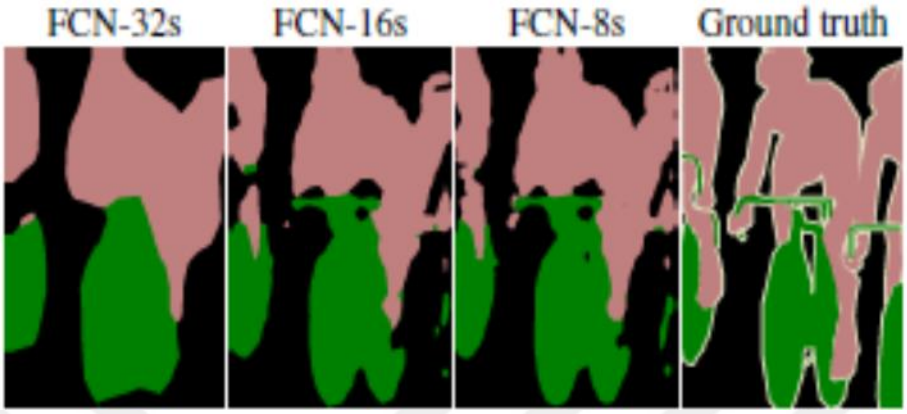

Figure 3.9. Segmentation performances of FCN-32, FCN-16, and FCN-8. (Tyantov, 2017)

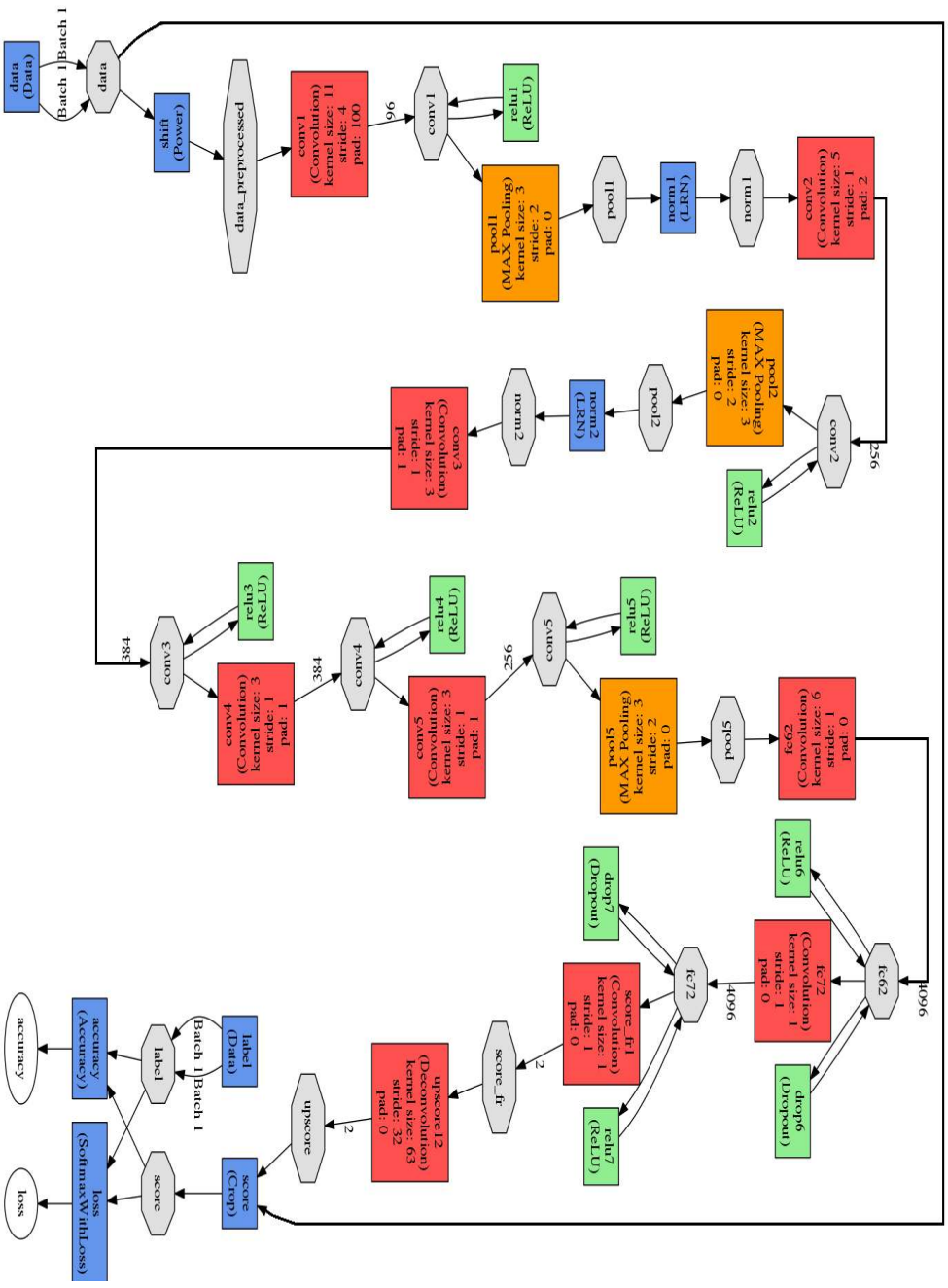

Figure 3.10. FCN-8s architecture

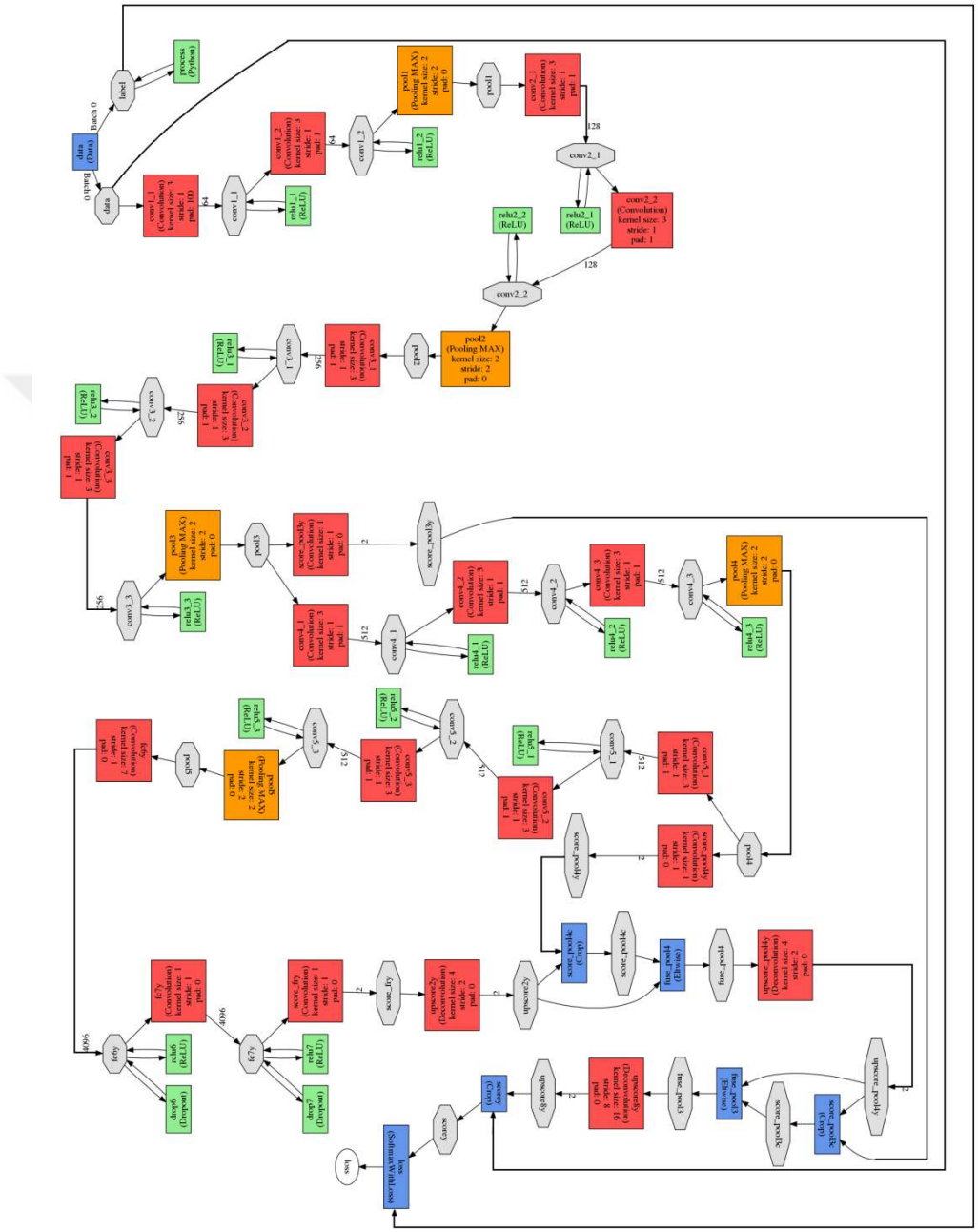

Figure 3.11. FCN-32s architecture

4. RESULTS AND DISCUSSIONS

4.1. Overview of Our Implemented Solution

In this section, the results were described and discussed. In fact, our solution consisted of using deep learning techniques to implement a neural network system, which allow us to firstly classify and predict cloud images, secondly to do the segmentation and the prediction of these cloud images.

4.1.1. Classification

For this classification, a neural network using convolutional neural network has been trained. We trained our neural network from scratch with CAFFE framework (using random initialization) and then we saved the weights of CNN as snapshots (. cafffemodel).

The steps of this classification are listed as follows:

4.1.1.1 Data Preparation

CAFFE framework which is one of the best frameworks for deep learning classification has been used. In fact, CAFFE provides GPU mode to get an amazing accelerated training. Data are stored in a database to reduce the I/O operations. Usually LMDB or LevelDB are recommended. In our case, our classification problem includes two classes (0 and 1). We grouped our data into a training folder and a validation folder. CAFFE trains our model on one set of images (training images) and tests the accuracy on the other set of images (validation images). From each set (training and validation) we created a text file specifying the classes that the images belong to.

/home/tienin/code/clouds/training/train(1).jpg 0

/home/tienin/code/clouds/training/train(301).jpg 1

Where the image “train(1).jpg 0” belong to the “class 0” and the image “train(301).jpg 1” belong to the “class 1”.

And then train_leveldb and val_leveldb folders have been generated by a python script. During the execution of this script the images contained in the training and validation folders have been resized to 227×227 format (Width=227 and Height=227) and saved in the lmdb files. Since our model has required the subtraction of the mean image from each image, the mean image (clouds_mean.binaryproto) has been computed from the leveldb folders.

4.1.1.2. Define A Model and the Solver

In this section, our training model was defined. We created and saved a file as “/home/tienin/code/clouds/my_train_val.prototxt”.

This file (my_train_val.prototxt)contains multiples include sections specifying either phase: TRAIN and TEST phase. These sections allow us to define two closely related networks in one file: the training network and the testing network. These two networks are almost identical. There are two main parts in this model definition: The input layers and the output layer.

Input Layers: The training network’s input layer draws its data from "/home/tienin/code/clouds/trainfile.txt" and the testing network’s input layer takes data from "/home/tienin/code/clouds/valfile.txt"

Output Layer: The loss layer is used during the training to compute the loss function and to initialize the back-propagation. During the validation this loss is just reported. The accuracy layer is also used to report the accuracy on the test set.

The solver is a file (my_solver.prototxt) that defines all information about how gradient descent will be conducted. In addition, the solver is responsible of the model optimization and guides the network’s forward inference and backward gradients. In our final solution, the solver computed the accuracy of our model

using the validation set every 100 iterations. The optimization process ran for a maximum of 1000 iterations and saves a snapshot of the trained model every 100 iterations. base_lr, lr_policy, gamma, momentum and weight_decay are hyper parameters that we have to tune up if we want to get a good convergence of our model. We chose lr_policy: "step" with stepsize: 100, base_lr: 0.0001 and gamma: 0.1. During our training, we started with a learning rate of 0.0001, and we dropped the learning rate by a factor of ten every 100 iterations.

net: "/home/tienin/code/clouds/my_train_val.prototxt"
test_iter: 100
test_interval: 100
base_lr: 0.0001
lr_policy: "step"
gamma: 0.1
stepsize: 100
display: 60
max_iter: 1000
momentum: 0.9
weight_decay: 0.0005
snapshot: 100
snapshot_prefix:"/home/tienin/code/clouds/finetune/fine3/finetune_clouds"
type: "sgd"
solver_mode: CPU

4.1.1.3. Fine-Tuning

Our network is retrained with the snapshots that CNN generated. The goal was to fine-tune the neural network. In order to fine-tune the network in this step, CNN model was retrained while using the previous snapshots and while changing

the parameters of the last Fully Connected Layer. This strategy, allowed us to avoid the over fitting issues.

The following paragraph will describe the different steps that are necessary if we want to fine tune a model:

- Usually in the case of fine tune, the strategy is to ***truncate the last layer*** (softmax layer) of the pre-trained network and replace it with our new softmax layer. For example, pre-trained network on ImageNet comes with a softmax layer with 1000 categories. In our neural network, the trained model has to come with a softmax layer with two categories (2 classes). Then our task will be a classification on two categories (2 classes), the new softmax layer of the network will be of two categories instead of 1000 categories. We also ran back propagation on the network to fine-tune the pre-trained weights. We have to make sure that holdout is performed well. That will allow the network to generalize well.
- ***A smaller learning rate*** should be used for the training. The learning rate is a parameter which tells the optimizer how far to move the weights in the direction of the gradient for a mini-batch. If the learning rate is too low, then training is more reliable, but the optimization will request more time because steps towards the minimum of the loss function are tiny. If the learning rate is too high, then training may not converge or even diverge. This can lead us to the rise of the weight values. And the optimizer will overshoot the minimum and will make the loss worse.

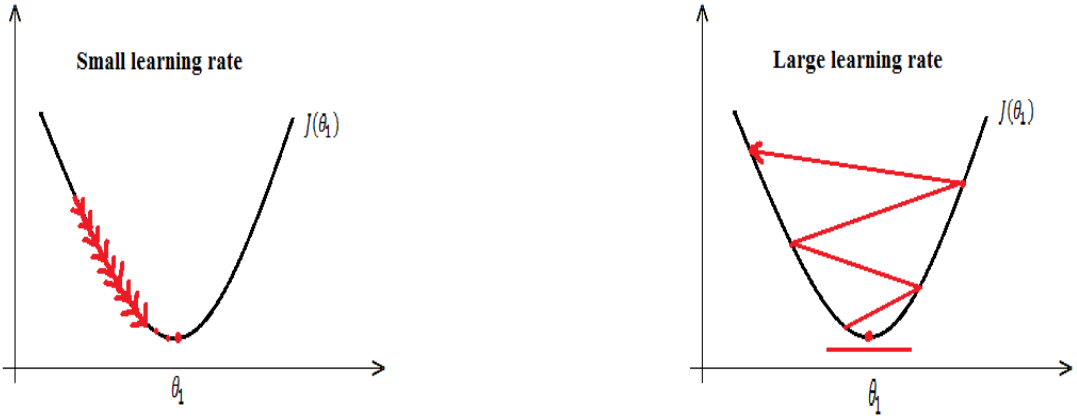

Figure 4.12. Learning rate

If η is too small, gradient descent can be slow.

If η is too large, gradient descent can overshoot the minimum. It may fail to converge or even diverge.

$$\theta_1 := \theta_1 - \eta \frac{\partial}{\partial \theta_1} J(\theta_1) \tag{4.1}$$

Where θ_1 is the local minima, η is the learning rate and $J(\theta_1)$ is the loss function

- Another usual practice is to ***freeze the weights of the first few layers*** of the pre-trained network. This method is usually utilized because the first few layers capture universal features (curves and edges). Finally, by minimizing the cross-entropy loss function using stochastic gradient descent (SGD) algorithm, model can be fine-tuned.

4.1.1.4 Deploy the Model and Prediction

After training the model, the weights snapshots are used to make predictions on a test data. We used a Python script for the predictions of the different classes. In order to run this script, we have to provide the following parameters:

- Test images: we gave the path of our test images.
- Mean image: we gave also the path of the mean image that we computed.
- Model architecture file: we deleted the data layers, added an input layer and changed the last layer type to SoftMax. From the previous my_train_val.prototxt file, and then we saved the new file as my_deploy
- Trained model weights: We used the snapshot that we got after the training (finetune_clouds_iter_10000.CAFFEmodel).

After the script has completed the running step, the predicted classes will be stored in the file "/home/tienin/code/clouds/predictions/submission_model.csv".

4.1.2. Segmentation

For the segmentation section, we trained a neural network using two different methods. These methods are respectively: FCN and U-NET.

4.1.2.1.FCN Segmentation

4.1.2.1.(1).Data Preparation

In FCN, images label must be uint8 type. Each object is represented with a color. Each label has four dimensions. Python layer is utilized to prepare labeled images. While 255 denote background, 0 denotes clouds in the image. Images and labels were resized to 500x500. Images are in BGR format and their dimensions are (channel X width X height).

Table 4.1. FCN data preparation

layer {
name: "data"
type: "Data"
top: "data"
top: "label"}
layer {
name: "process"
type: "Python"
bottom: "label"
top: "label"
python_param {
module: "digits_python_layers"
layer: "preprocess"}}
	import CAFFE
	import random
	import numpy as np
	class preprocess(CAFFE.Layer):
	
	def setup(self, bottom, top):
	
	pass
	
	def reshape(self, bottom, top):
	
	top[0].reshape(1,1,500,500)
	
	def forward(self, bottom, top):
	
	label=np.array(bottom[0].data[...],dtype=np.uint8)
	label = label[np.newaxis, ...]
	top[0].data[...]= bottom[0].data[...]
	
	def backward(self, top, propagate_down, bottom):
	pass

4.1.2.1.(2). Model Preparation and Training

In this study, two FCN models: FCN32 and FCN8 were utilized. Both models were fine-tuned with Berkeley VOC-FCN-32 and VOC-FCN-8 models for 20 epochs. These models had been trained with PASCAL VOC 2011 Challenge Dataset. In this dataset, there are 28952 images with 20 objects and 1 class is used to denote background (Everingham, et al, 2011).

In this thesis, three different learning algorithms: Stochastic Gradient Descent, Adam's Momentum and Nesterov's techniques were used during fine-tuning. For these models, learning rates were chosen as 0.0001, 0.01, and 0.001, respectively.

4.1.2.1.(2).a. Stochastic Gradient Algorithm

Gradient descent is one of the most used algorithms to optimize neural networks. Indeed, the goal of the gradient algorithm in Machine Learning is to minimize the lost function ($Q(w)$). The gradient descent has three variants, but in this work the stochastic gradient descent (SGD) has been used. SGD tried to find the best w, by minimizing Q.

$$w \leftarrow w - \eta \nabla Q(w) \tag{4.2}$$

$$Q(w) = \frac{1}{n}\sum_i Q_i(w) \Rightarrow \nabla Q(w) = \frac{1}{n}\sum_i \nabla Q_i(w) \tag{4.3}$$

Where, w is the machine learning model (the parameters), Q_i is i^{th} observation in the training data, n.is total number of data and η is the learning rate
Here are the different steps of the stochastic gradient algorithm:

- Initialize the parameters w and the learning rate η
- Repeat until an approximate minimum is obtained:
  - Shuffle randomly examples in the training set
  - For $i = 1,2, \ldots, n$, do:

$$w \leftarrow w - \eta \nabla Q_i(w) \tag{4.4}$$

4.1.2.1.(2).b Nesterov Algorithm

With Nesterov momentum, basically the gradient but not the current one for the weights is calculated. It is indeed the current weights plus the momentum constant times the previous delta. So, there's an alternative form of Nesterov momentum where the delta looks a bit different but is exactly the same mathematically. The alternative form uses just the gradient calculated for the current weights, which is much easier to compute.

$$\Delta w = -1 \times \left(\eta \times \frac{\partial Q}{w}\right) + (\alpha \times \Delta w^{(t-1)} \quad (4.5)$$

$$w = w + \Delta w \quad (4.6)$$

Where α is the momentum (constant value which is chosen as 0.9) and t the number of steps.

4.1.2.1.(2).c Adam Algorithm

The Adam optimization algorithm is an extension to stochastic gradient descent. It is a new method in deep learning applications, computer vision and natural language processing. Adam means adaptive moment estimation. Adam allows the computing of individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Adam is also the combination of two methods: Adaptive Gradient (AdaGrad) and Root Mean Square Propagation (RMSProp).

Adam update rule consists of the following steps:

Here m_t is the first moment (mean) of the gradient and v_t is the second moment (uncentered variance) of the gradient. β_1 and β_2 are the decay rates. ε is a given constant. $\widehat{m_t}$ and $\widehat{v_t}$ are respectively the bias-corrected first moment and the bias-corrected second moment.

- Compute gradient g_t at current time t
- Update biased first moment estimate:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t. \quad (4.7)$$

- Update biased second raw moment estimate:

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (4.8)$$

- Compute bias-corrected first moment estimate:

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t} \quad (4.9)$$

- Compute bias-corrected second raw moment estimate:

$$\widehat{v_t} = \frac{v_t}{1-\beta_2^t} \quad (4.10)$$

- Update parameters:

$$w = w_{t-1} - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}}+\varepsilon} \quad (4.11)$$

4.1.2.2. U-NET Segmentation

U-NET is implemented with KERAS and TENSORFLOW as framework for this task. The model is trained from scratch (using random initialization) and then the weights have been saved as snapshots (. hdf5). In addition, the model has been retrained to improve the results.

4.1.2.2.(1). Data Preparation

In this section, 613 images have been used for the segmentation. In order to achieve the segmentation of our cloud images we needed to annotate the input images for the segmentation training. This annotation step is also called labeling. During this study two methods for the annotation have been used:

- Labelme: is a WEB-based image annotation tool that allows us to label images and generate the annotated masks. In order to annotate our cloud images, the Labelme tool has been used to map the cloud

regions and then generate the image mask from the original image. During this step, our goal was to set as pixel value of the background zero (0) and the foreground one (1). Here the foreground represented the cloud regions. So, the annotated images had two different classes.

- Otsu thresholding technique: Since we have used two classes for the segmentation, the Otsu thresholding can be used on our dataset in order to subdivide the image into two classes. Indeed, the Otsu thresholding is an image processing technique that calculates the optimum threshold separating the two classes (foreground and background).

After we have finished the labeling part, we encoded our mask image using the Run Length Encoding/Decoding (RLE) compressing algorithm. Indeed, the Run length encoding (RLE) is a technique, which compresses digital data by representing successive runs of the same value in the data as the value followed by the count, rather than the original run of values. The goal was to reduce the amount of data needed to be stored.

Example of RLE:

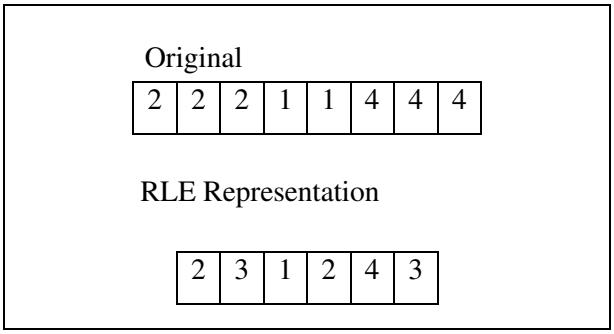

Figure 4.13. RLE encode example

As we can see from the above example, RLE performs well when it is applied to data where there are successive runs of the same values. Also, one of the advantages of the RLE is that we can decode it and display the image.

4.1.2.2.(2). Define Model and Training

In this model definition, we built an Encoder-Decoder network. Our goal here is to set up a neural network that learns a representation enough to distinguish pixels of the cloud structure from the rest of the images. Our network has 4 max pooling layers and each max pooling layer has 2 convolutions operations. There is a simple contracting path and then an expanding path, which embed the information present in the entire image into lower dimensional vectors. The size of this vector is halved at every max-pooling layer. Auto encoders perform well when it learnt the representation of the features but does not perform well when tried to learn the spatial localization present in the image, during the learning step. In other words, this network captures the *'what'* in the image but not the *'where'*. Indeed, in this architecture the feature maps from convolutional layer part in down sampling step are fed to the up-convolution part in up-sampling step.

For the training, we were looking for a way to measure model performance quantitatively. We have decided to use the dice coefficient. The model will output a mask *X*, and the dice coefficient compared it to the mask *Y* that has been produced during the annotation step:

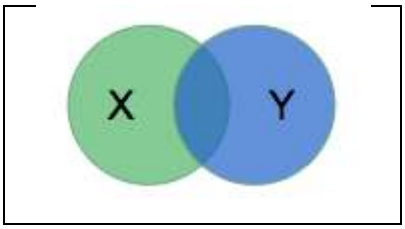

Figure 14.3. Dice computing

$$Dice(X,Y) = \frac{2 \times |X \cap Y|}{|X| + |Y|} \quad (4.12)$$

Here the dice metric is twice the ratio of the intersection between X and Y over the sum of areas (X and Y). Our model is trained for 20 epochs. After 20 epochs, the Dice coefficient and the Loss function have calculated. The weights are updated by RMSprop optimizer and 1e-5 has been used as learning rate. During the training, our model's weights are saved in HDF5 format (. hdf5).

4.1.2.2.(3). Testing and Prediction

For this section, we have used 315 images to test our model. The goal here was to give as input new cloud images different than the training images, and then our model will generate the binary output images (segmented images). To achieve this goal, we used the training weights, which our different models saved.

4.2. Results

In this section, the results of the classification and of the segmentation have been displayed.

4.2.1.Classification results

We used convolutional neural network to achieve our goal.

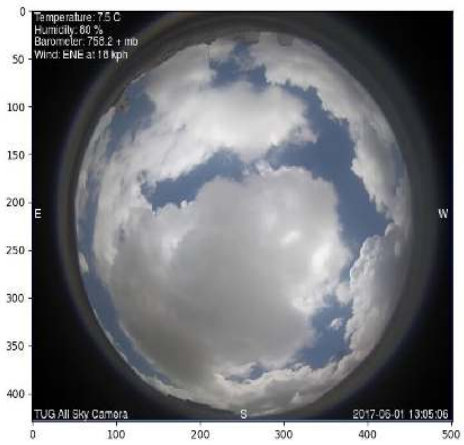

Figure 15. Predicted image (class 0)

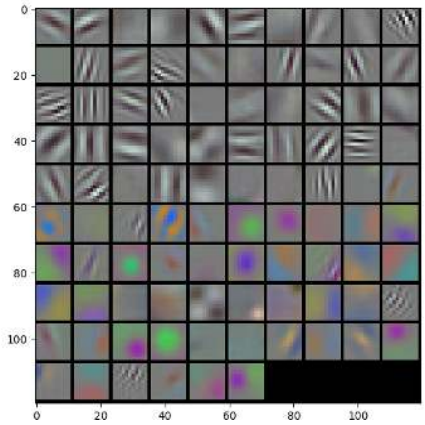

Figure 16. First convolution Layer (conv1)

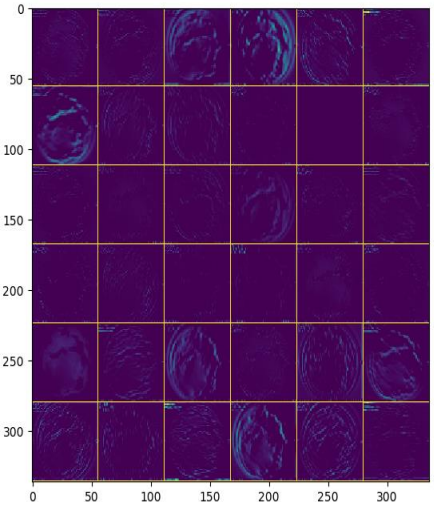

Figure 17. Feature Mapping (feat1)

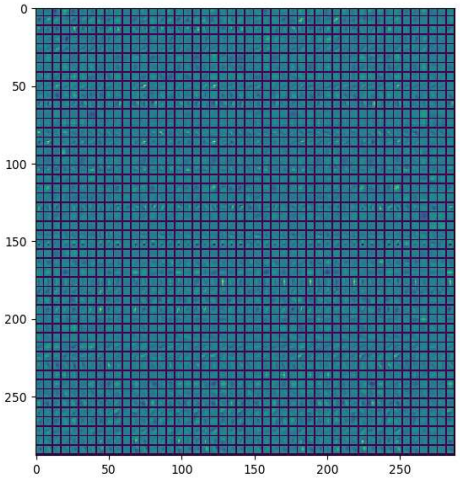

Figure 18. Second convolutinal layer (conv2)

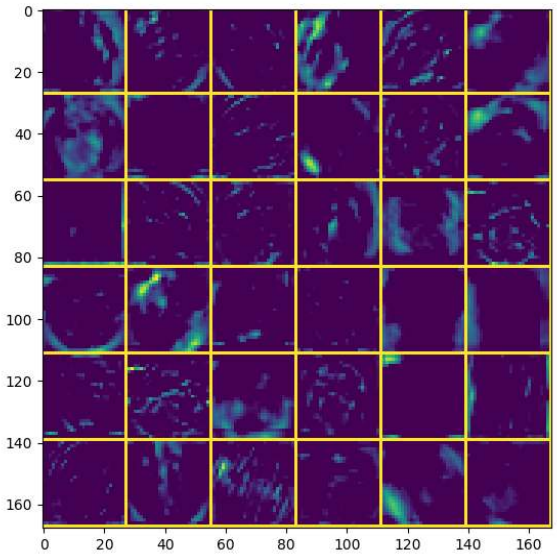

Figure 19. Feature Mapping (feat2)

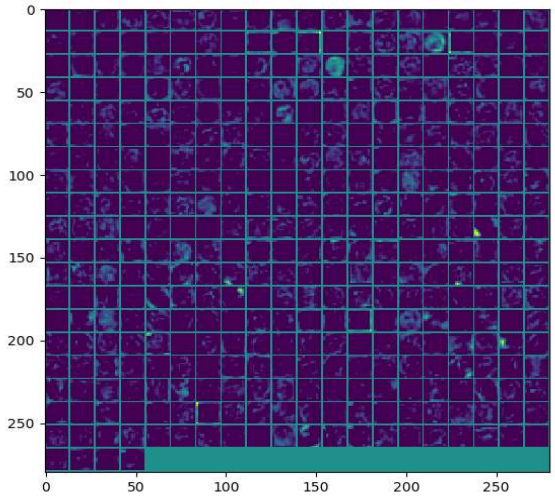

Figure 20. Third Convolutional layer (conv3)

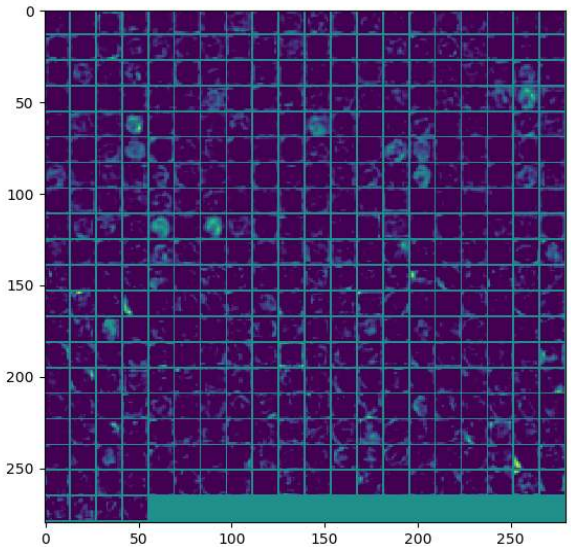

Figure 210. Fourth Convolutional layer (Conv4)

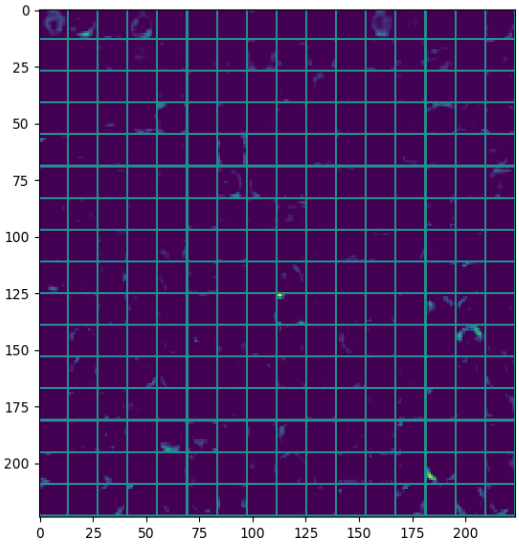

Figure 22. Fifth Convolutional layer (Conv5)

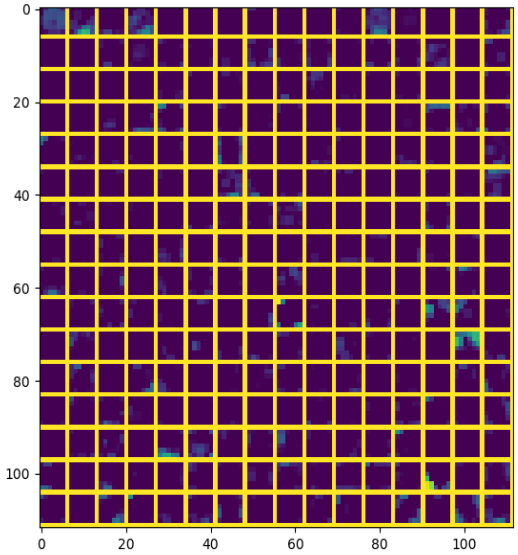

Figure 4.123. Pooling (Pool5)

4.2.2. Segmentation

4.2.2.1. Image Processing Segmentation

Image processing segmentation techniques: We have used Otsu technique for the segmentation and some edge detectors. We also used morphological watersheds techniques. Here are the different results that we have got:

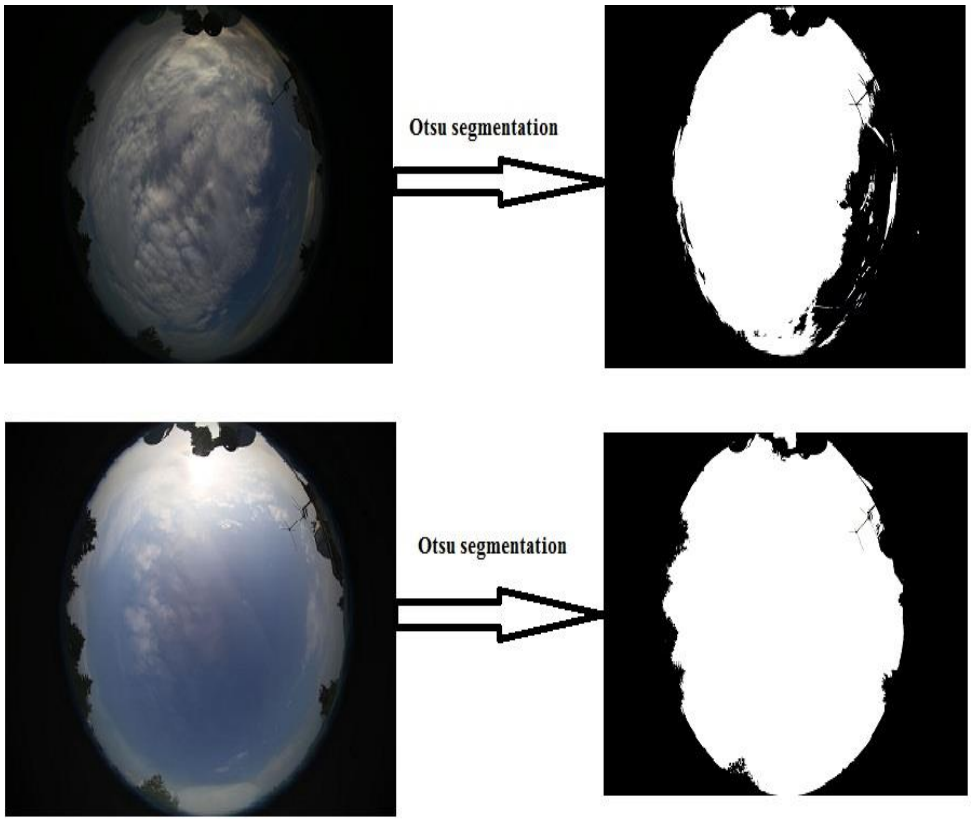

Figure 24.13. Otsu segmentation

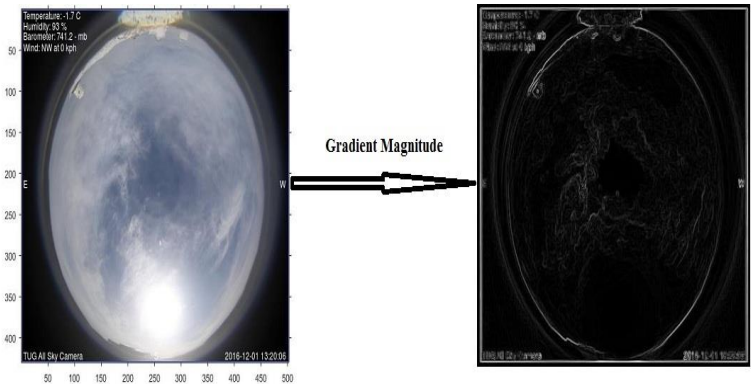

Figure 25. Gradient Magnitude as edge detector

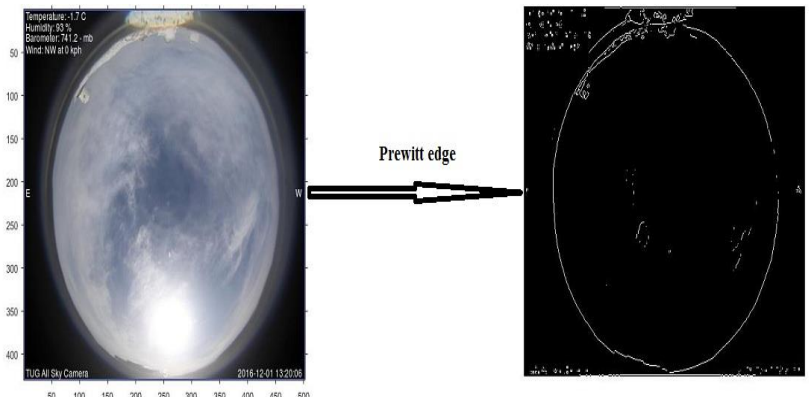

Figure 26. Prewitt as edge detector

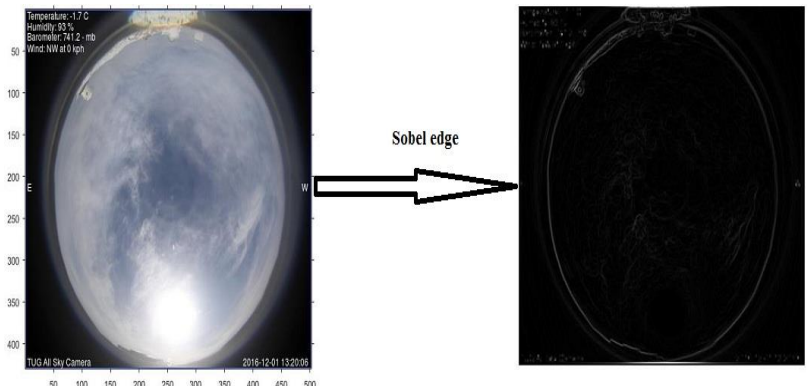

Figure 27. Sobel as edge detector

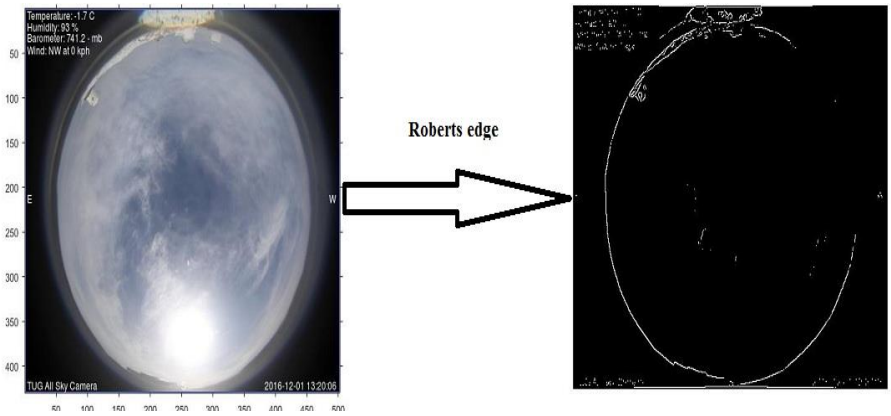

Figure 28. Roberts as edge detector

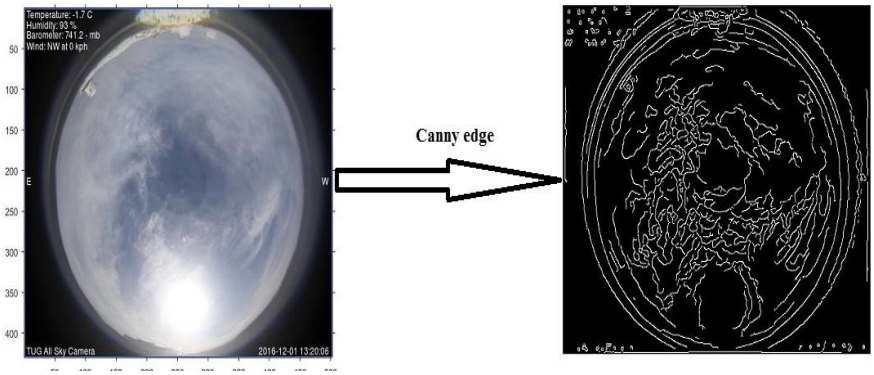

Figure 29. Canny as edge detector

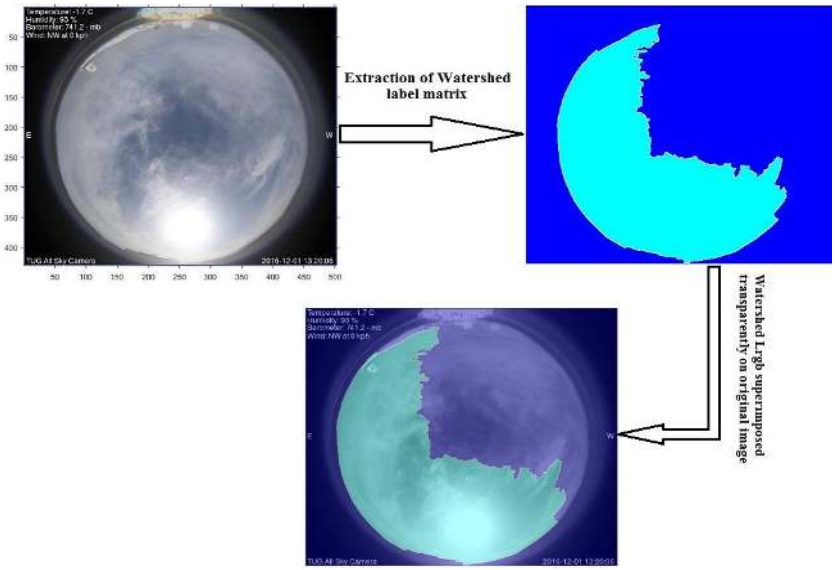

Figure 30. Watershed for segmentation

4.2.2.2. FCN Segmentation

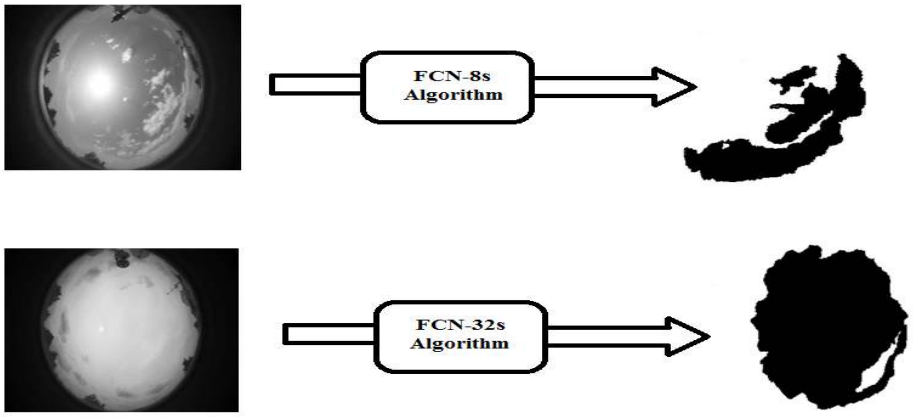

Figure 31. Testing step on FCN

Table 4. 2. FCN accuracy table

FCN-8	Testing Accuracy	Testing Loss
Adam	63.12%	66.23%
Stochastic Gradient Descent	52.42%	71.45%
Nesterov	53.56%	70.67%
FCN -32	Testing Accuracy	Testing Loss
Adam	58.24%	68.34%
Stochastic Gradient Descent	45.42%	78.1%
Nesterov	50.04%	73.02%

Here FCN calculates loss and accuracy by finding similarities between pixel values of the labels and the output.

4.2.2.3. U-NET Segmentation

U-NET: U-net with different input sizes: $128 \times 128,\ 256 \times 256,\ 512 \times 512\ and\ 1024 \times 1024$, has been implemented. These are the different results of our U-net:

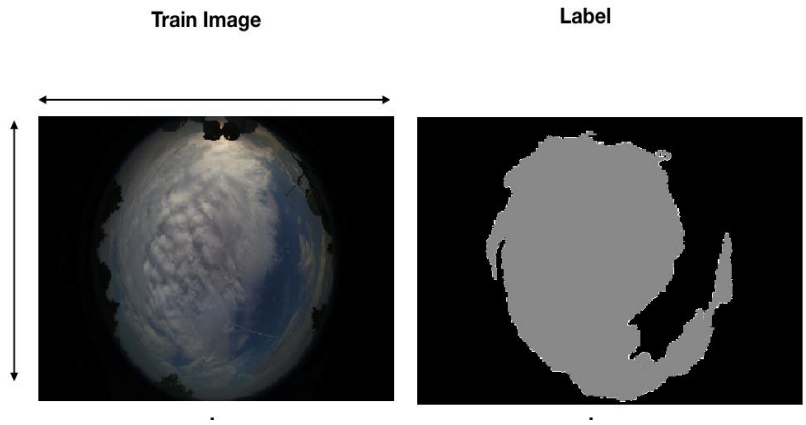

Figure 32. Labeling Step

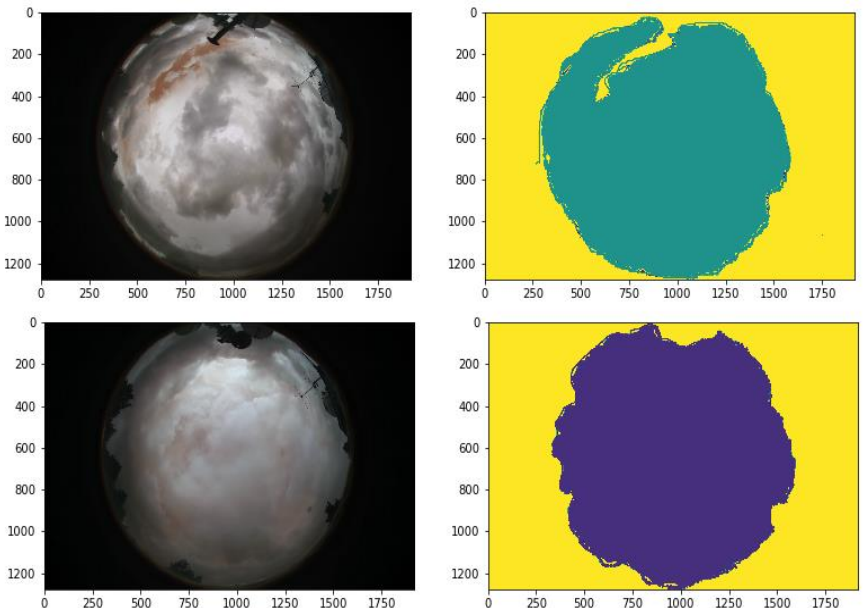

Figure 33. Labeling visualization from KERAS

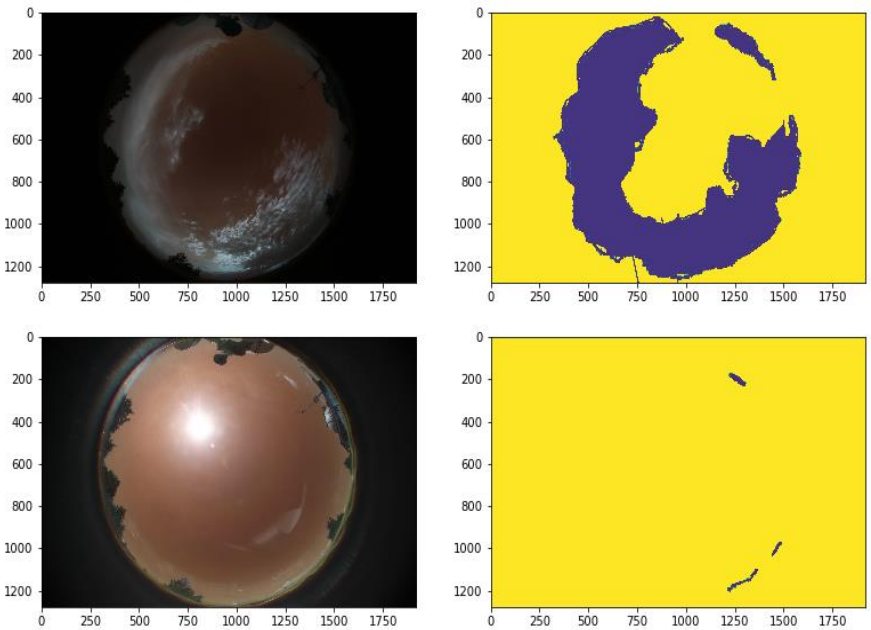

Figure 34. Labeling visualisation from KERAS

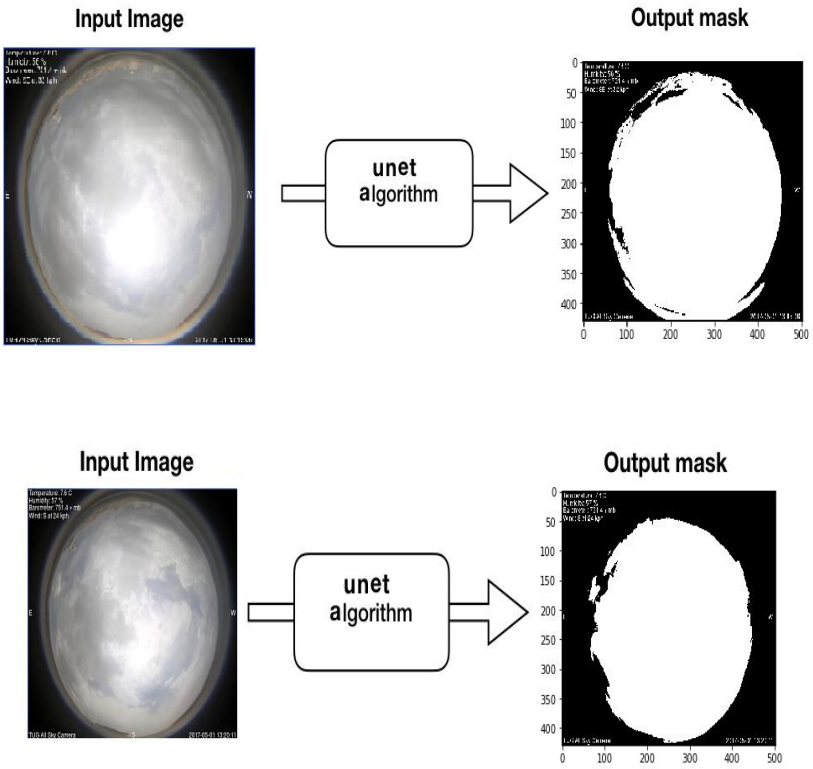

Figure 35. Testing step generated binary images

Results during the training from scratch

```
Epoch 1/50
 - 4251s - loss: 0.6948 - dice_loss: 0.7737 - val_loss: 1.0379 - val_dice_loss: 0.6344
Epoch 2/50
 - 4222s - loss: 0.5847 - dice_loss: 0.8189 - val_loss: 0.9874 - val_dice_loss: 0.6530
Epoch 3/50
 - 4240s - loss: 0.5462 - dice_loss: 0.8325 - val_loss: 0.9715 - val_dice_loss: 0.6585
Epoch 4/50
 - 4226s - loss: 0.5110 - dice_loss: 0.8454 - val_loss: 0.8718 - val_dice_loss: 0.6961
Epoch 5/50
 - 4214s - loss: 0.4993 - dice_loss: 0.8504 - val_loss: 0.8627 - val_dice_loss: 0.6999
Epoch 6/50
 - 4210s - loss: 0.4883 - dice_loss: 0.8546 - val_loss: 0.9419 - val_dice_loss: 0.6719
Epoch 7/50
 - 4212s - loss: 0.4793 - dice_loss: 0.8579 - val_loss: 0.8336 - val_dice_loss: 0.7138
Epoch 8/50
 - 4213s - loss: 0.4757 - dice_loss: 0.8598 - val_loss: 0.8222 - val_dice_loss: 0.7188
Epoch 9/50
 - 4208s - loss: 0.4632 - dice_loss: 0.8628 - val_loss: 0.7673 - val_dice_loss: 0.7413
Epoch 10/50
 - 4201s - loss: 0.4630 - dice_loss: 0.8643 - val_loss: 0.7624 - val_dice_loss: 0.7437
Epoch 11/50
 - 4203s - loss: 0.4553 - dice_loss: 0.8660 - val_loss: 0.7154 - val_dice_loss: 0.7611
Epoch 12/50
 - 4199s - loss: 0.4572 - dice_loss: 0.8664 - val_loss: 0.6567 - val_dice_loss: 0.7871
Epoch 13/50
 - 4202s - loss: 0.4470 - dice_loss: 0.8692 - val_loss: 0.6086 - val_dice_loss: 0.8053
Epoch 14/50
 - 4204s - loss: 0.4509 - dice_loss: 0.8680 - val_loss: 0.5945 - val_dice_loss: 0.8102
Epoch 15/50
 - 4201s - loss: 0.4464 - dice loss: 0.8698 - val loss: 0.5841 - val dice loss: 0.8146
Epoch 16/50
 - 4210s - loss: 0.4424 - dice_loss: 0.8704 - val_loss: 0.5941 - val_dice_loss: 0.8100
Epoch 17/50
 - 4202s - loss: 0.4412 - dice_loss: 0.8711 - val_loss: 0.5200 - val_dice_loss: 0.8399
Epoch 18/50
 - 4198s - loss: 0.4452 - dice_loss: 0.8707 - val_loss: 0.5235 - val_dice_loss: 0.8412
Epoch 19/50
 - 4197s - loss: 0.4378 - dice_loss: 0.8723 - val_loss: 0.5415 - val_dice_loss: 0.8337
Epoch 20/50
 - 4197s - loss: 0.4414 - dice_loss: 0.8709 - val_loss: 0.4893 - val_dice_loss: 0.8552
```

Figure 36. U-NET 128x128 training from scratch

```
Epoch 1/20
 - 6420s - loss: 0.8429 - dice_loss: 0.7189 - val_loss: 1.0894 - val_dice_loss: 0.6151
Epoch 2/20
 - 6199s - loss: 0.6564 - dice_loss: 0.7885 - val_loss: 1.0751 - val_dice_loss: 0.6199
Epoch 3/20
 - 6182s - loss: 0.6296 - dice_loss: 0.7997 - val_loss: 0.9939 - val_dice_loss: 0.6492
Epoch 4/20
 - 6650s - loss: 0.5920 - dice_loss: 0.8151 - val_loss: 0.8844 - val_dice_loss: 0.6907
Epoch 5/20
 - 5934s - loss: 0.5756 - dice_loss: 0.8215 - val_loss: 0.7351 - val_dice_loss: 0.7520
Epoch 6/20
 - 5933s - loss: 0.5518 - dice_loss: 0.8290 - val_loss: 0.7135 - val_dice_loss: 0.7588
Epoch 7/20
 - 6027s - loss: 0.5430 - dice_loss: 0.8329 - val_loss: 0.6387 - val_dice_loss: 0.7895
Epoch 8/20
 - 6030s - loss: 0.5326 - dice_loss: 0.8364 - val_loss: 0.9148 - val_dice_loss: 0.6857
Epoch 9/20
 - 5839s - loss: 0.5223 - dice_loss: 0.8397 - val_loss: 0.9220 - val_dice_loss: 0.6943
Epoch 10/20
 - 5914s - loss: 0.5210 - dice_loss: 0.8414 - val_loss: 0.6229 - val_dice_loss: 0.7973
Epoch 11/20
 - 5948s - loss: 0.5091 - dice_loss: 0.8450 - val_loss: 0.6179 - val_dice_loss: 0.8026
Epoch 12/20
 - 5847s - loss: 0.5057 - dice_loss: 0.8459 - val_loss: 0.6374 - val_dice_loss: 0.7966
Epoch 13/20
 - 5903s - loss: 0.4957 - dice_loss: 0.8495 - val_loss: 0.5673 - val_dice_loss: 0.8233
Epoch 14/20
 - 5964s - loss: 0.4943 - dice_loss: 0.8500 - val_loss: 0.6148 - val_dice_loss: 0.8147
Epoch 15/20
 - 5850s - loss: 0.4839 - dice_loss: 0.8534 - val_loss: 0.5960 - val_dice_loss: 0.8118
Epoch 16/20
 - 5933s - loss: 0.4822 - dice_loss: 0.8546 - val_loss: 0.5237 - val_dice_loss: 0.8382
Epoch 17/20
 - 6352s - loss: 0.4755 - dice_loss: 0.8556 - val_loss: 0.5003 - val_dice_loss: 0.8524
Epoch 18/20
 - 5893s - loss: 0.4718 - dice_loss: 0.8577 - val_loss: 0.5099 - val_dice_loss: 0.8445
Epoch 19/20
 - 5855s - loss: 0.4740 - dice_loss: 0.8572 - val_loss: 0.5163 - val_dice_loss: 0.8536
Epoch 20/20
 - 5934s - loss: 0.4679 - dice_loss: 0.8593 - val_loss: 0.5169 - val_dice_loss: 0.8479
```

Figure 37. U-NET 256x256 training from scratch

```
Epoch 1/20
 - 8125s - loss: 0.9912 - dice_loss: 0.6559 - val_loss: 1.0423 - val_dice_loss: 0.6316
Epoch 2/20
 - 8121s - loss: 0.8751 - dice_loss: 0.6942 - val_loss: 0.9203 - val_dice_loss: 0.6773
Epoch 3/20
 - 8123s - loss: 0.8373 - dice_loss: 0.7093 - val_loss: 0.7827 - val_dice_loss: 0.7297
Epoch 4/20
 - 8140s - loss: 0.7822 - dice_loss: 0.7293 - val_loss: 0.7224 - val_dice_loss: 0.7594
Epoch 5/20
 - 8118s - loss: 0.7510 - dice_loss: 0.7407 - val_loss: 0.6925 - val_dice_loss: 0.7661
Epoch 6/20
 - 8130s - loss: 0.7172 - dice_loss: 0.7533 - val_loss: 0.9235 - val_dice_loss: 0.7162
Epoch 7/20
 - 8130s - loss: 0.6912 - dice_loss: 0.7646 - val_loss: 0.6633 - val_dice_loss: 0.7803
Epoch 8/20
 - 8131s - loss: 0.6624 - dice_loss: 0.7748 - val_loss: 0.6434 - val_dice_loss: 0.7862
Epoch 9/20
 - 8132s - loss: 0.6360 - dice_loss: 0.7846 - val_loss: 0.6043 - val_dice_loss: 0.8007
Epoch 10/20
 - 8114s - loss: 0.6169 - dice_loss: 0.7926 - val_loss: 0.6611 - val_dice_loss: 0.7961
Epoch 11/20
 - 8121s - loss: 0.5944 - dice_loss: 0.8022 - val_loss: 0.5722 - val_dice_loss: 0.8146
Epoch 12/20
 - 8123s - loss: 0.5739 - dice_loss: 0.8104 - val_loss: 0.5605 - val_dice_loss: 0.8189
Epoch 13/20
 - 8123s - loss: 0.5576 - dice_loss: 0.8176 - val_loss: 0.5410 - val_dice_loss: 0.8280
Epoch 14/20
 - 8134s - loss: 0.5427 - dice_loss: 0.8238 - val_loss: 0.5356 - val_dice_loss: 0.8318
Epoch 15/20
 - 25968s - loss: 0.5280 - dice_loss: 0.8300 - val_loss: 0.5144 - val_dice_loss: 0.8391
Epoch 16/20
 - 7999s - loss: 0.5118 - dice_loss: 0.8363 - val_loss: 0.5212 - val_dice_loss: 0.8479
Epoch 17/20
 - 7990s - loss: 0.5032 - dice_loss: 0.8407 - val_loss: 0.5343 - val_dice_loss: 0.8373
Epoch 18/20
 - 7994s - loss: 0.4948 - dice_loss: 0.8447 - val_loss: 0.4948 - val_dice_loss: 0.8483
Epoch 19/20
 - 7993s - loss: 0.4802 - dice_loss: 0.8499 - val_loss: 0.4939 - val_dice_loss: 0.8619
Epoch 20/20
 - 7990s - loss: 0.4757 - dice_loss: 0.8529 - val_loss: 0.4935 - val_dice_loss: 0.8501
```

Figure 38. U-NET 512x512 training from scratch

```
Epoch 1/20
 - 12951s - loss: 0.8835 - dice_loss: 0.6938 - val_loss: 0.8039 - val_dice_loss: 0.7216
Epoch 2/20
 - 13171s - loss: 0.7568 - dice_loss: 0.7418 - val_loss: 0.6827 - val_dice_loss: 0.7707
Epoch 3/20
 - 12911s - loss: 0.7185 - dice_loss: 0.7564 - val_loss: 0.6296 - val_dice_loss: 0.7919
Epoch 4/20
 - 12913s - loss: 0.6865 - dice_loss: 0.7699 - val_loss: 0.6115 - val_dice_loss: 0.8000
Epoch 5/20
 - 12852s - loss: 0.6576 - dice_loss: 0.7802 - val_loss: 0.5956 - val_dice_loss: 0.8056
Epoch 6/20
 - 12880s - loss: 0.6357 - dice_loss: 0.7877 - val_loss: 0.6037 - val_dice_loss: 0.8023
Epoch 7/20
 - 12779s - loss: 0.6078 - dice_loss: 0.7986 - val_loss: 0.5675 - val_dice_loss: 0.8177
Epoch 8/20
 - 12743s - loss: 0.5812 - dice_loss: 0.8089 - val_loss: 0.5694 - val_dice_loss: 0.8230
Epoch 9/20
 - 12793s - loss: 0.5688 - dice_loss: 0.8143 - val_loss: 0.5417 - val_dice_loss: 0.8283
Epoch 10/20
 - 12758s - loss: 0.5445 - dice_loss: 0.8233 - val_loss: 0.5281 - val_dice_loss: 0.8342
Epoch 11/20
 - 12829s - loss: 0.5302 - dice_loss: 0.8299 - val_loss: 0.5271 - val_dice_loss: 0.8333
Epoch 12/20
 - 12744s - loss: 0.5218 - dice_loss: 0.8341 - val_loss: 0.5152 - val_dice_loss: 0.8415
Epoch 13/20
 - 12822s - loss: 0.5082 - dice_loss: 0.8393 - val_loss: 0.4974 - val_dice_loss: 0.8479
Epoch 14/20
 - 12890s - loss: 0.4942 - dice_loss: 0.8448 - val_loss: 0.4949 - val_dice_loss: 0.8508
Epoch 15/20
 - 12829s - loss: 0.4933 - dice_loss: 0.8470 - val_loss: 0.4851 - val_dice_loss: 0.8561
Epoch 16/20
 - 12801s - loss: 0.4885 - dice_loss: 0.8490 - val_loss: 0.4750 - val_dice_loss: 0.8614
Epoch 17/20
 - 12859s - loss: 0.4761 - dice_loss: 0.8537 - val_loss: 0.4706 - val_dice_loss: 0.8631
Epoch 18/20
 - 12753s - loss: 0.4721 - dice_loss: 0.8565 - val_loss: 0.4678 - val_dice_loss: 0.8641
Epoch 19/20
 - 12800s - loss: 0.4655 - dice_loss: 0.8589 - val_loss: 0.5091 - val_dice_loss: 0.8762
Epoch 20/20
 - 12892s - loss: 0.4618 - dice_loss: 0.8611 - val_loss: 0.4859 - val_dice_loss: 0.8552
```

Figure 39. U-NET 1024x1024 training from scratch

Table 4.3. The best scores during the training from scratch

U-net description	Training Dice coeff	Training Loss	Validation Dice coeff	Validation Loss
128×128; Contracting path filter sizes:64, 128, 256, 512, 1024.	0.8723	0.4378	0.8552	0.4893
256×256; Contracting filter sizes: 32, 64, 128, 256, 512, 1024.	0.8593	0.4679	0.8524	0.5003
512×512; Contracting filter sizes:16, 32, 64, 128, 256, 512, 1024.	0.8529	0.4757	0.8619	0.4935
1024×1024; Contracting filter sizes: 8, 16, 32, 64, 128, 256, 512, 1024.	0.8611	0.4618	0.8762	0.4678

Table 4.4. U-net 128×128 fine tuning training

Epoch Number	Running time(s)	Training Loss	Training dice coeff	Validation Loss	Validation Dice coeff
1	4234	0.4363	0.8730	0.4920	0.8529
2	4272	0.4335	0.8732	0.4856	0.8566
3	4199	0.4359	0.8726	0.4616	0.8673
4	4195	0.4317	0.8737	0.4544	0.8713
5	4200	0.4310	0.8741	0.4616	0.8687
6	4210	0.4316	0.8743	0.4607	0.8693
7	4229	0.4279	0.8754	0.4510	0.8738
8	4229	0.4304	0.8747	0.4557	0.8732
9	4228	0.4313	0.8742	0.4737	0.8795
10	4189	0.4251	0.8762	0.4613	0.8797

Table 4.5. U-net 256×256 fine tuning training

Epoch Number	Running time(s)	Training Loss	Training dice coeff	Validation Loss	Validation Dice coeff
1	6246	0.4611	0.8599	0.4949	0.8605
2	6002	0.4595	0.8605	0.4838	0.8627
3	6064	0.4557	0.8616	0.4724	0.8624
4	5872	0.4553	0.8617	0.4714	0.8595
5	5906	0.4555	0.8616	0.4699	0.8591
6	5881	0.4494	0.8631	0.4699	0.8584
7	6032	0. 4537	0.8620	0.4659	0.8607
8	6255	0.4530	0.8622	0.4646	0.8612
9	5996	0.4527	0.8625	0.4630	0.8619
10	5903	0.4523	0.8622	0.4620	0.8625

Table 4.6. U-net 512×512 fine tuning training

Epoch Number	Running time(s)	Training Loss	Training dice coeff	Validation Loss	Validation Dice coeff
1	7980	0.4535	0.8600	0.4621	0.8621
2	7978	0.4525	0.8599	0.4621	0.8622
3	7978	0.4537	0.8595	0.4618	0.8623
4	7979	0.4539	0.8599	0.4618	0.8623
5	7971	0.4546	0.8596	0.4632	0.8615
6	7977	0.4543	0.8599	0.4624	0.8621
7	7980	0.4534	0.8599	0.4625	0.8620
8	7976	0.4513	0.8600	0.4619	0.8623
9	7980	0.4547	0.8598	0.4619	0.8623
10	7995	0.4574	0.8584	0.4612	0.8626

Table 4.7. U-net 1024×1024 fine tuning training

Epoch Number	Running time(s)	Training Loss	Training dice coeff	Validation Loss	Validation Dice coeff
1	12990	0.4564	0.8617	0.4675	0.8625
2	12973	0.4573	0.8609	0.4704	0.8608
3	12894	0.4514	0.8623	0.4660	0.8625
4	12882	0.4549	0.8615	0.4650	0.8627
5	12898	0.4540	0.8617	0.4625	0.8645
6	12925	0.4503	0.8628	0.4620	0.8640
7	12904	0.4469	0.8641	0.4618	0.8655
8	12794	0.4527	0.8625	0.4626	0.8643
9	12808	0.4485	0.8640	0.4619	0.8642
10	12804	0.4524	0.8626	0.4590	0.8654

Table 4.8. The best scores during the fine-tuning training

U-net description	Training Loss	Training Dice coeff	Validation Loss	Validation Dice coeff
128×128; Contracting path filter sizes:64, 128, 256, 512, 1024.	0.4222	0.8770	0.4544	0.8797
256×256; Contracting filter sizes: 32, 64, 128, 256, 512, 1024.	0.4494	0.8631	0.4620	0.8625
512×512; Contracting filter sizes:16, 32, 64, 128, 256,512,1024.	0.4513	0.8600	0.4612	0.8626
1024×1024; Contracting filter sizes: 8, 16, 32, 64, 128, 256, 512, 1024.	0.4469	0.8641	0.4590	0.8654

Table 4.9. Running time of the training from scratch

128x128	256x256	512x512	1024x1024
4251	6420	8125	12951
4222	6199	8121	13171
4240	6182	8123	12911
4226	6650	8140	12913
4214	5934	8118	12852
4210	5933	8130	12880
4212	6027	8130	12779
4213	6030	8131	12743
4208	5839	8132	12793
4201	5914	8114	12758
4203	5948	8121	12829
4199	5847	8123	12744
4202	5903	8123	12822
4204	5964	8134	12890
4201	5850	25968	12829
4210	5933	7999	12801
4202	6352	7990	12859
4198	5893	7994	12753
4197	5855	7993	12800
4197	5934	7990	12892
Total in seconds			
84210	120607	179699	256970
Total in hours			
23,39166667	33,50194444	49,91638889	71,38055556
Total in days			
23h23min30sec	1Day9h30min7sec	2Days1h54min59sec	2Days23h22min50sec

Table 4.10. Running time for the training from the fine tuning

128x128	256x256	512x512	1024x1024
4234	6246	7980	12990
4272	6002	7978	12973
4199	6064	7978	12894
4195	5872	7979	12882
4200	5906	7971	12898
4210	5881	7977	12925
4229	6032	7980	12904
4229	6255	7976	12794
4228	5996	7980	12808
4189	5903	7995	12804
Total in seconds			
42185	60157	79794	128872
Total in hours			
11,71805556	16,71027778	22,165	35,79777778
Total in days			
11h43min5sec	16h42min37sec	22h9min54sec	1Day11h47min52sec

4.3. Discussions

In this section, the experimental results, which were displayed, previously, have been analyzed and interpreted.

4.3.1. The Classification

We have used 680 images for this task. With these 680 images, 408 were clouds images, and 272 were other images different than cloud. Indeed, we divided our data into 2 sets: training and validation. The holdout method has been used to proceed during the training. For our solution, we have chosen 73.5 % (500 images) for the training and 26.5 % (180 images) for the validation. After using the classic

parameter 80% and 20%, our above choice provided the best accuracy score. We also resized all of our input data to 227x227 format. Using the “transform_img” as a function, we were able to apply the histogram equalization on our input data and then resized them into our expectation format. After completing all the settings, we trained our model, which provided us the loss score and the accuracy score. Therefore, in order to reach this best accuracy score, we have trained our model in two ways. Firstly, our model has been trained from scratch, after 1000 iterations the highest scored accuracy was 67%. Secondly, our model has been fine-tuned in order to improve the previous accuracy score. Using the “bvlc_reference_CAFFEnet. CAFFE model,” 92% has been scored as the highest accuracy.

Since we have trained our neural network, this means that we have computed and saved the weights of our model. We used these weights during our testing and prediction phase. For this step, we have used 315 images, which contained 160 cloud images and 155 non-cloud images (others). Our goal here was to classify correctly into two different classes. The class (0) belongs to cloud images, and the class (1) belongs to other images different than cloud images. After using the weights from scratch training for the testing phase, out of 315 images, our model misclassified 14 images and classified correctly 301 images. After using the fine tuning, the model, we have got 314 images correctly classified and 1 image misclassified. The weights from scratch provided us an accuracy of 95% and these weights from the fine-tuning provided 99% as accuracy.

4.3.2. The Segmentation

In this section, we shall discuss the results obtained from image processing techniques and deep learning techniques.

4.3.2.1. Image Processing Techniques

For this section, different methods have been used such as Otsu method, edge detection, and watershed segmentation technique. Among the different segmentation techniques that exist, the automatic thresholding methods are mostly used. Otsu method is one of the thresholding methods. Otsu performs well only when the numbers of pixels in each class are close to each other. In our case, the results that we have obtained are not so great. In some cases, Otsu failed to separate the cloud pixel from the sun. With Otsu, we generated a binary image where the black color represented the background, and the white represented the segmented clouds. As we said previously, with the images which contained the sun and blue sky (without cloud), we noticed that Otsu was mapping also the sun and the blue sky as cloud pixels. Otsu performed well when our input images are fully cloudy. Otherwise, this method is not accurate for the cloud images segmentation.

Edge detection is used in image processing as a technique for finding the boundaries of objects. It works by detecting discontinuities in brightness. Edge detection can be used for image segmentation and data extraction. In this study, we applied Gradient magnitude, Roberts, Prewitt, Sobel and Canny edge detectors. In the displayed results (see section 4.1.) we can see that after we have applied Gradient Magnitude edge detector, we obtained some pixels that represented the cloud and this technique managed to draw almost 80% of the boundary of the input image. The Prewitt edge detector managed to draw the entire boundary of the input image but did not manage to represent any cloud pixel. The Sobel edge detector managed to display some cloud pixels even if they are not distinct like the gradient magnitude. Fortunately, it performed well with the boundary drawing. The Roberts managed to represent almost 80% of the boundary but did not represent any cloud pixel. Among all of the edge detection techniques, Canny edge detector appeared to be the best because it managed well to represent most of the cloud pixels and at the same time, the entire boundary has been drawn.

The Watershed technique appeared to be the best image processing technique among all of the others. Indeed, it detected almost 75% of the cloud pixels and drew perfectly the boundary of the input image. It seems to be more accurate than the previous image processing techniques. As we can see in section 4.1, on the figure describing the result obtained after using the watershed, only 25% of the clouds are not segmented. This technique managed to segment and clearly display almost 75% of the cloud pixels.

4.3.2.2. Deep Learning Methods

This section describes the results obtained through the U-NET and the FCN models. Indeed, in this section, for both (U-NET and FCNs) we have used 613 cloud images for the training phase and 300 images for the testing phase. We have also used the classic holdout (80% for training and 20% for validation).

4.3.2.2.(a). U-NET

Four different models of U-NET have been implemented. Each of them has different input shape (U-NET128×128, U-NET256×256, U-NET 512×512, and U-NET1024×1024).

- ✓ U-NET128×128: Here our input image has been resized to 128×128 format. Then up sampling and down sampling have been applied as mentioned in section 4.1. In this model, we have seen that the initial filters have 64×64 as size and 1024×1024 as central filters. That is why the contracting path filter sizes are: 64, 128, 256, 512, and 1024. After applying the dice coefficient formula, this model provided an accuracy of 85%, and the loss was 48% during the training phase from scratch. After the fine-tuning, we obtained 87% as accuracy and 45% as a loss.

- ✓ U-NET 256×256: Here our input image has been resized to 256×256 format. Then up sampling and down sampling have been applied as mentioned in section 4.1. In this model, we have seen that the initial filters have 32×32 as size and 1024×1024 as central filters. That is why the contracting path filter sizes are: 32, 64, 128, 256, 512, and 1024. After applying the dice coefficient formula, this model provided an accuracy of 85%, and the loss was 50% during the training phase from scratch. After the fine-tuning, we obtained 86% as accuracy and 46% as loss.
- ✓ U-NET 512×512: Here our input image has been resized to 512×512 format. Then up sampling and down sampling have been applied as mentioned in the section 4.1. In this model, we have seen that, the initial filters have 16×16 as size and 1024×1024 as central filters. That is why the contracting path filter sizes are: 16, 32, 64, 128, 256, 512, and 1024. After applying the dice coefficient formula, this model provided an accuracy of 85%, and the loss was 49% during the training phase from scratch. After the fine-tuning, we obtained 86% as accuracy and 46% as loss.
- ✓ U-NET 1024×1024: Here our input image has been resized to 256×256 format. Then up sampling and down sampling have been applied as mentioned in the section 4.1. In this model, we have seen that the initial filters have 8×8 as size and 1024×1024 as central filters. That is why the contracting path filter sizes are: 8, 16, 32, 64, 128, 256, 512, and 1024. After applying the Dice coefficient formula, this model provided an accuracy of 87%, and the loss was 46% during the training phase from scratch. After the fine-tuning, we obtained 86% as accuracy and 45% as loss.

Our different models were trained twice with a maximum of 20 epochs for the training from scratch, and 10 epochs for the fine-tuning training. Among all of the training, our fine-tuning models have shown better results than these one from scratch. Indeed, we have scored 87% as the highest testing accuracy and 45% as the lowest loss. Since we have trained our models using the CPU we spent hours and sometimes days during the training phase. The longest training model was U-NET 1024×1024 with 2 days 23 hours 22minutes for the training from scratch and 1 day 11 hours 47 minutes for the fine-tuning training.

4.3.2.2.(b). FCN

Two different FCN architectures were fine-tuned with 3 different optimization models (Adam, Nesterov, and Stochastic gradient descent). The utilized architectures were FCN-32 and FCN-8. Both architecture includes seven convolution layers with different up sampling approaches. While FCN-32 includes two deconvolution layers, FCN-8 includes 3 deconvolution layers which provide details in segmented areas. The best results for both models were obtained with Adam optimizer. The best result achieved with FCN architectures is 63.12% with FCN-8.

4.3.3. Deep Learning Techniques vs. Image Processing Techniques

As we can see in section 4.1, the image processing techniques are less accurate than the Deep learning techniques. From the edge detector, Otsu method, up the Watershed technique, the cloud segmentation has been done with several cloud pixels missing in the final output images. In fact, in these techniques and most of the time, we have used some mathematical operations such as convolution operators, thresholding, erosion and dilatation operators on the input images, and then an output has been displayed and saved. Here the input image has been processed ones. There was not any control system trying to calculate the error and

then back propagate in order to reduce the processing errors that usually appear. But on the other hand, the implementation and the running time of the image processing techniques were pretty simple and short. Unfortunately, the image processing techniques are not performing well when the different features in the data are not similar.

Nevertheless, deep learning techniques provide segments and display the cloud features. We obtained most of the time a good representation of the cloud and non-cloud. Also, with the deep learning techniques, we were able to implement a control system to check and back propagate the learning error. In fact, the deep learning solutions offered more autonomy because the idea is to make the machine think like a human being. Usually, in the classic computer systems, computers make decisions according to mathematical logics. However, in deep learning, it is a little bit different, given that not only mathematical logics are used. The computers learned and are able to learn from their mistakes. That is why deep learning techniques offer more reliability and perform well than any other technique. The main success behind the deep learning, unlike image processing techniques it extracts features with a learning algorithm. On the other hand, deep learning solutions are not easy to implement, due to some hardware resources (if we want to use the GPU in our implementation NVIDIA is requested as graphic card or running times (when we ran using the CPU and for the deeper neural network).

5.CONCLUSION

Over the past few years, several methods and solutions have been implemented in the field of machine learning. This thesis proposed a deep learning solution for cloud images classification and segmentation. To achieve our goal, three different frameworks were used (CAFFE, KERAS and TENSORFLOW). DIGITS and JUPYTER tools were utilized to run these frameworks.

Indeed, a "two classes" system has been implemented, using CAFFE framework. We built a neural network, which was able to classify cloud images from non-clouds images. In fact, our neural network learns how to extract cloud features from cloud images. And based on these features our network separated the cloud images from the other images. During this step, the highest learning accuracy was 92%, and the highest testing accuracy was 99%. These accuracies were scored after an optimization process (fine tune).

On the other hand, two different neural networks were also implemented for the segmentation of the cloud images. The first one has been implemented using KERAS and TENSORFLOW as frameworks. In fact, a U-net neural network has been implemented for the cloud images' segmentation. This neural network mapped the cloud regions and generated an output image where we can clearly observe the cloud's part and the background. Using the same method, a dice coefficient and the loss were calculated. In addition, in order to have optimized results, four different models of U-NET have been implemented. Each of them has different input shape (U-NET128×128, U-NET256×256, U-NET 512×512, and U-NET 1024×1024). Among these models 87% were scored as dice accuracy.

The second cloud segmentation solution (FCN) has been implemented for the segmentation with different architectures: FCN-8 and FCN-32, and optimization techniques, such as Stochastic gradient descent, Adam, and Nesterov.

We have chosen DIGITS tool because of the parallelism in computing. DIGITS allows us to use the GPU during the running step. We were able to get the results of our models within some few hours. While FCN32s has provided maximally 58.24%, FCN8s has provided maximally 63.12% accuracy by using Adam's optimization.

As we saw in the previous sections of this study, we have proved that a computer through machine learning techniques can learn how to detect and recognize cloud from images like a human being. From the above results, we have seen that deep learning solutions performed better than image processing solutions. And among the deep learning techniques, we have also noticed that the U-NET (87%) offered better prediction and training accuracy than the FCN8s (63.12%). Using a system that allows us to classify and at the same time will enable us to make the segmentation of the cloud images is pretty good in the field of weather forecast. Nowadays everything is linked to artificial intelligence that is why our study may benefit in several fields. In our future studies, we will try to focus on 3D images segmentation using deep learning techniques, or on real-time image segmentation using deep learning techniques.

REFERENCES

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/ 4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

AMPLab,"Image Classification with Pipelines", available in the internet atCheng, H.Y., and Lin, C.L., 2016 "Cloud Detection in All-Sky Images via Multi-scale Neighborhood Features and Multiple Supervised Learning Techniques" under review for jornal Atmos.Meas.Tech. At http://ampcamp.berkeley.edu/5/exercises/image-classification-with-pipelines.html

Barta, A., Horvath, G., Horvath, A., Egri, A., Blaho, M., Barta, P., Bumke, K., and macke, a., 2015 Testing a Polarimetric Cloud Imager aboard Research Vessel Polarstern: Comparison of Ten Different Cloud Detection Algorithms, APPLIED OPTICS / Vol. 54, No. 5 / 10 February 2015

Başlar, I., (2012), Cloudıness Analysıs By Usıng All Sky Camera Images, TüRkiYe BiLiMsel Ve TeknolojıK AraşTırma Kurumu Ulusal Gözlemevi Müdürlüğü.

C. Szegedy, A. Toshev, and D. Erhan. Deep neural networks for object detection. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 2553–2561. Curran Associates, Inc., 2013. At http://papers.nips.cc/paper/5207-deep-neural-networks-for-object-detection.pdf.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014. Athttp://arxiv.org/abs/1409.4842.

Chen, L.C, Papandreou, G., Kokkinos, I., Murphy, K., and Yuille, A.L., 2016, Semantic image segmentation with Deep Convolutional Nets and Fully Connected CRFs, arXiv: 1412.7062v4 [cs.CV] 7 jun 2016.

Chen, L.C, Papandreou, G., Kokkinos, I., MURPHY, K., and Yuille, A.L., 2016, DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, arXiv: 1606.00915v1 [cs.CV] 2 jun 2016.

Chow, C.W., Belongie, S. and Kleissl, J., 2015 Cloud motion and stability estimation for intra-hour solar forecasting, Solar Energy 115(2015)645-655

Clay, R.W., Pace, R.T., Riordan, D. S., Smth, A. G. K. & Wild N. R., 1999, Cloud monitoring for large cosmic ray sites, Proceedings of the 26th International Cosmic Ray Conference., (IUPAP). Vol:5, Salt Lake City, Utah, p:421, (1999)

Dev, S., Wen, B., Lee, Y.H., and Winkler, S., 2016 Machine Learning Techniques and Applications For Ground-based Image Analysis IEEE Geoscience and Remote Sensing Magazine

Difraction limited, The boltwood cloud sensor At http://www.cyanogen.com/cloud_main.php

Everingham, M., L.,Van Gool, C. K. I., J, Williams., and Winn, A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results.

Fua, C.L.,and Cheng, H.Y., 2013 Predicting solar irradiance with all-sky image features via regression, Solar Energy, vol. 97, pp. 537–550, Nov. 2013.

Gao, H.Z.W, Chen, X., and Zhao, D., 2006 Object detection using spatial histogram features, Image and Vision Computing 24 (2006) 327–341

Ghonima, M. S., Urquhart, B., Chow, C. W., Shıelds, J. E., Cazorla, A., and Kleissl, J.,2012 A method for cloud detection and opacity classification based on ground based sky imagery Atmos. Meas. Tech., 5, 2881–2892, 2012

Gonzalez, R.C., Woods, R.E., and Eddins, S.T., (2009) Digital Image Processing using MATLAB Second Edition. United States of America: Gatesmark, LLC

Heinle, A., Macke, A. and Srivastav, A., 2010, Automatic cloud classification of whole sky images, Atmos.Meas.Tech.,3, p:557–67, (2010).

Hong, Y., Chiang, Y.M., Liu, Y., Hsu, K.L., and Sorooshian, S., 2006 Satellite-based precipitation estimation using watershed segmentation and growing hierarchical self-organizing map. International Journal of Remote Sensing Vol. 27, No. 23, 10 December 2006, 5165–5184

IBM Research, 2015,"Staring at the Sun" available in the internet at https://www.ibm.com/blogs/research/2015/10/staring-at-the-sun/

Jedlovec, G., 2010 Automated Detection of Clouds in Satellite Imagery Geosciences and Remote Sensing.

Kartal, D., (2006) "Kuantum Kuyulu Kızılötesi Fotodedektörler" MSc Thesis, Cumhuriyet Üniversitesi, Sivas.

Kleissl, J, 2013 Solar Energy Forecasting and Resource Assessment. San Diego, USA: Academic Press.

Lukac, R., and Plataniotis, N.K., 2007 Color Image Processing: Methods and Applications. United States of America: CRC Press

Mangan, A.P., and Whitaker, R.T., 1999, Partitioning 3D Surface Meshes Using Watershed Segmentation, journal of Ieee Transactıons On Vısualızatıon And Computer Graphıcs, Vol. 5, No. 4, October-December 1999

Moujahid, A., 2016, "figure of the training and prediction". Adilmoujahid.com Post, June 26th, 2016, http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/?utm_source=top.caibaojian.com/111120

Neuberg, B., 2016 “Cloudless: open source Deep Learning Pipeline for orbital Satellite Data”. At http://codinginparadise.org/ebooks/html/blog/introducing_cloudless.html

Nikumbh, A., Harikishan, G., and Padmakumari, B.,2016 Cloud detection and Opacity classification from ground based Sky Imagery

Peng, Z., Yu, D., Huang, D., Heiser, J., Yoo, S., and Kalb, P., 2015, 3D Cloud Detection and Tracking System for Solar Forecast Using Multiple Sky Imagers, journal of Solar Energy Volume 118, August 2015, Pages 496–519.

Pfister, G., Mckenzie, R. L., Liley, J. B., Thomas, A., Forgan, B. W., and Long, C. N., J., 2003, Cloud coverage based on all-sky imaging and its impact on surface solar irradiance, Appl. Meteorol., 42, 1421–34.

Pratt, W.K., 2001 Digital Image Processing: PIKS Inside, Third Edition. New York: John Wiley and sons, Inc.

Ronneberger, O., Fischer, P., and Brox, T., 2015, “figure of U-Net: Convolutional Networks for Biomedical Image Segmentation”, Medical Image Computing and Computer-Assisted Intervention (MICCAI), Springer, LNCS, Vol.9351: 234--241, 2015, http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a

Roy, G., Hayman S., Julian W., 2008, “Sky Modelling from Digital Imagery”. At http://www.cadplan.com.au/Reports/dsmreport.pdf. University of Sydney Murdoch University.

S. Dev, F. Savoy, Y. H. Lee, and S. Winkler, WAHRSIS: A low-cost, high-resolution whole sky imager with near-infrared capabilities, in Proc. IS&T/SPIE Infrared Imaging Systems, 2014.

Smith, S. and Toumi, R., 2008 Measuring Cloud Cover and Brightness Temperature with a Ground-Based ThermalInfrared Camera Applied Meterology and Climatology Volume 47

Tyantov, E., 2017, “Figure of FCNs segmentation”, Post, Jan 21, 2017, https://www.slideshare.net/Eduardyantov/ultrasound-segmentation-kaggle-review

Viblo, 2017, “Figure of Fully connected layer”. Post, Nov 26th, 2017, 10:58 PM, https://viblo.asia/p/intro-to-deep-learning-3P0lPkmPZox

Wang, Y.H., 2010 Tutorial: Image Segmentation

Wikipedia, 2016, “Cloud Coverage”. At http://en.wikipedia.org/wiki/Cloud cover

Y. LeCun, Y. Bengio, G. Hilton. Deep Learning. Nature 251, pages 436-444, May 2015.

Zhen, S., Jayasumana, S., Romera-Paredes, B., Vineet, V., Su, Z., Du, D., Huang, C., and Torr, P.H.S, 2015, Conditional Random Fields as Recurrent Neural Networks, International Conference on Computer Vision (ICCV)

Zheng, C., Pulido, J., Thorman, P., and Hamann, B., 2015, An improved method for object detection in astronomical images, MNRAS 451, 4445–4459.

BIOGRAPHY

Bole Wilfried TIENIN was born in Ouagadougou, Burkina Faso, in 1989. He has three sisters. In June 2001, he completed his elementary education at Ecole Primaire Nakeb-Zanga, Ouagadougou. He studied at College Prive Wend Manegda for his high school, where he obtained his junior degree (BEPC) in June 2005 and his Baccalaureate in June 2009. In October 2009, he started his academic studies in the department of Computer Engineering, at Bobo-Dioulasso Polytechnic University, where he obtained his Bachelor degree in computer engineering in 2013. And then in 2014, he has been granted a scholarship to study in Turkey, by The Turkish Government through Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı (YTB). He started his Master program at the Department of Computer Engineering, Cukurova University, Adana, since 2015 with Assist. Prof. Dr. Buse Melis ÖZYILDIRIM as his supervisor.