National Observatory of Athens

Department of Informatics and Telecommunications -University of Peloponnese

Master of Science in

Space Science Technologies and Applications

ALEXANDRA ANTONOPOULOU

Dipl.Eng.

Convolutional Neural Networks for Automated Detection of ULF Waves in Swarm Time Series

MASTER THESIS

Advisory Committee

George Balasis – Dr., Advisor

Konstantinos Koutroumbas – Dr., Member

Athanassios Rontogiannis – Dr., Member

ATHENS 2020

Abstract

Ultra-low frequency (ULF) magnetospheric plasma waves play a key role in the dynamics of the Earth's magnetosphere and, therefore, their importance in Space Weather studies is indisputable. Magnetic field measurements from recent multi-satellite missions (e.g. Cluster, THEMIS, Van Allen Probes and Swarm) are currently advancing our knowledge on the physics of ULF waves. In particular, Swarm satellites, one of the most successful mission for the study of the near-Earth electromagnetic environment, have contributed to the expansion of data availability in the topside ionosphere, stimulating much recent progress in this area. Coupled with the new successful developments in artificial intelligence (AI), we are now able to use more robust approaches devoted to automated ULF wave event identification and classification. The goal of this effort is to use a deep learning method in order to classify ULF wave events using magnetic field data from Swarm. We construct a Convolutional Neural Network (CNN) that takes as input the wavelet spectrum of the Earth's magnetic field variations per track, as measured by each one of the three Swarm satellites, and whose building blocks consist of two convolution layers, two pooling layers and a fully connected (dense) layer, aiming to classify ULF wave events in four different categories: 1) Pc3 wave events (i.e., frequency range 20-100 MHz), 2) non-events, 3) false positives, and 4) plasma instabilities. Our primary experiments show promising results, yielding successful identification of almost 90% accuracy. We are currently working on producing larger training/test datasets, by analyzing Swarm data from the mid-2014 onwards, when the final constellation was formed, aiming to construct a dataset comprising of more than 50000 wavelet image inputs for our network.

Περίληψη

Τα κύματα εξαιρετικά χαμηλής συχνότητας, γνωστά ως κύματα ULF (Ultra-Low Frequency waves) παίζουν σημαντικό ρόλο στη δυναμική της γήινης μαγνητόσφαιρας, και επομένως η σημασία τους στη μελέτη του διαστημικού καιρού είναι αδιαμφισβήτητη. Μετρήσεις μαγνητικού πεδίου από πρόσφατες δορυφορικές αποστολές (π.χ. Cluster, THEMIS, Van Allen Probes και Swarm) συμβάλουν σημαντικά στην διεύρυνση των γνώσεών μας πάνω στη φυσική που διέπει τα κύματα ULF. Πιο συγκεκριμένα, οι δορυφόροι Swarm, μία από τις πιο επιτυχημένες δορυφορικές αποστολές για τη μελέτη του ηλεκτρομαγνητικού περιβάλλοντος γύρω από τη Γη, έχει συμβάλει στη επέκταση των διαθέσιμων δεδομένων που αφορούν την ανώτερη (topside) ιονόσφαιρα, οδηγώντας πρόσφατα σε μεγάλη πρόοδο στον τομέα. Σε συνδυασμό με τη μεγάλη εξέλιξη στον τομέα της μηχανικής μάθησης (machine learning), είμαστε πλέον σε θέση να χρησιμοποιούμε προηγμένες μεθόδους που αποσκοπούν στην αυτοματοποιημένη αναγνώριση και ταξινόμηση των διάφορων ULF γεγονότων. Στόχος της παρούσας εργασίας είναι η χρήση μίας μεθόδου βαθιάς μάθησης (deep learning), για την ταξινόμηση γεγονότων ULF χρησιμοποιώντας δεδομένα μαγνητικού πεδίου από την αποστολή Swarm. Κατασκευάζουμε ένα συνελικτικό νευρωνικό δίκτυο (convolutional neural network) το οποίο δέχεται ως είσοδο μία σειρά εικόνων, που αναπαριστούν τα φάσματα της ισχύος συναρτήσει της συχνότητας και του χρόνου, οι οποίες έχουν προκύψει ύστερα από wavelet μετασχηματισμούς στις χρονοσειρές μαγνητικού πεδίου ανά δορυφορική τροχιά. Η βασική αρχιτεκτονική του δικτύου έχει ως εξής: δύο συνελικτικά επίπεδα (convolutional layers) και δύο επίπεδα χωρικής υποδειγματοληψίας (pooling layers), εναλλασσόμενα μεταξύ τους, και ένα πλήρως συνδεδεμένο επίπεδο (fully connected layer) στο τέλος προκειμένου να δημιουργήσει το μοντέλο απόφασης (classification). Σκοπός του δικτύου είναι να ταξινομήσει γεγονότα ULF σε 4 κατηγορίες: 1) κύματα ULF κατηγορίας Pc3 (εύρος συχνοτήτων 20-100 MHz), 2) θόρυβος, 3) ψευδώς θετικά γεγονότα, και 4) ηλεκτρομαγνητικές αστάθειες του πλάσματος. Τα πρωταρχικά μας πειράματα δίνουν πολύ αισιόδοξα αποτελέσματα, αποδίδοντας επιτυχή αναγνώριση γεγονότων ULF με ακρίβεια σχεδόν 90%. Μελλοντικός στόχος είναι η επέκταση της εργασίας κατασκευάζοντας μεγαλύτερα σύνολα δεδομένων (training/test datasets), με την ανάλυση δεδομένων Swarm από τα μέσα του 2014 και έπειτα, όπου πραγματοποιήθηκε η τελική διαμόρφωση του σχηματισμού των τριών δορυφόρων, για τη δημιουργία τουλάχιστον 50000 δειγμάτων εισόδου (50000 ζεύγη που το καθένα αποτελείται από μία φασματική εικόνα και την κατηγορία στην οποία αυτή ανήκει (class label)) για το δίκτυό μας.

Table of Contents

Chapter 1: Introduction

Ultra-low frequency (ULF) waves are produced by processes in the Earth's magnetosphere and solar wind. The broader class of ULF waves are quasi-sinusoidal and exist for several periods; these are classified as continuous and are further broken down into five subcategories, Pc1-5, depending upon their frequency (Jacobs et al., 1964). Their energy source is either in the solar wind or within the magnetosphere. The magnetosphere is a resonant cavity and waveguide for waves that either originate within or propagate through the system. Magnetospheric ULF waves have a profound effect on charged particle dynamics in the radiation belts (Mann, 2016). For instance, ULF waves may accelerate electrons with MeV energies in the radiation belts. These electrons may penetrate spacecraft shielding, generate a build-up of static charge within electrical components resulting in subsystem damage, and ultimately can even cause the total loss of Earth-orbiting satellites (Mann, 2016). As radiation belts variability has a direct impact on spacecraft and humans in space, ULF waves are of particular importance for space weather.

The study of ULF waves is a very active field of space research and much has yet to be learned about the processes that generate these waves. Recent studies for the analysis of ULF waves (e.g. Balasis et al., 2013, 2019) have stimulated much progress in this area. In Balasis et al. 2013, a wavelet-based spectral analysis tool has been developed for the classification of ULF wave events using data from the low-Earth orbit (LEO) CHAMP satellite, in preparation of the Swarm mission, while in Balasis et al. 2019, a machine learning technique based on Fuzzy Artificial Neural Networks has been employed in order to detect ULF waves in the time series of the magnetic field measurements on board CHAMP. The analysis of isolated ULF wave events— especially those detected in the Pc3 frequency range (20–100 mHz) that a topside ionosphere mission, like the Swarm mission, efficiently resolves—can help to elucidate the processes that play a crucial role in the generation of waves and their most defining propagation characteristics.

Machine learning techniques have been successfully introduced in the fields of Space Physics and Space Weather, yielding highly promising results in modeling and predicting many disparate aspects of the geospace environment. A promising machine learning method to solve image classification problems is Convolutional Neural Networks (CNN/ConvNets). Therefore, we want to study the capability of ConvNets to classify ULF wave events represented on images. To do so, we create an image dataset, starting from magnetic field time series from Swarm satellite measurements and applying a wavelet transform that allows us to go from one-dimensional (i.e., time series) to two-dimensional space (i.e., images).

The ConvNets are not a new idea; for example, in 1989 LeCun et al. used a ConvNet to recognize handwritten digits in mails, but the hype around ConvNets started in 2012 after the incredible winning of Krizhevsky et al. on the ImageNet challenge for classification of images into 1000 categories. The ConvNet solution of Krizhevsky et al. had a top-5 error-rate of 15.3% compared to the second place with 26.17%. Since this, the error rate is further decreased with other ConvNet designs and further research on the training of Neural Networks. ConvNets have not only reached good results on ImageNet and other image classification tasks, but they are also applied successfully on non-image tasks, e.g., in natural language processing (Collobert & Weston, 2008).

ConvNets, or rather Neural Networks (NN), are very interesting machine learning methods. They have a layered structure, and every layer could learn an abstract representation of the

input. This ability is called Feature Learning or known as Representation Learning (LeCun et al., 2015). Feature Learning is a method, that allows a machine to learn from raw data. The machine takes the raw data as input and learns automatically the needed features to solve the machine learning problem. For example, in image classification the raw data is an image, which is represented by an array of pixels. These arrays are fed to the ConvNet, and the ConvNet learns useful features from these images to solve the machine learning problem.

There are numerous variants of CNN architectures in the literature. However, their basic components are very similar. Taking the famous LeNet-5 as an example (LeCun et al., 1998), it consists of three types of layers, namely convolutional, pooling, and fully-connected layers. The convolutional layer aims to learn feature representations of the inputs. In the first layer, the ConvNet could learn to detect low-level features such as edges and curves, while in higher layers the ConvNet could learn to encode more abstract features. By stacking several convolutional and pooling layers, we could gradually extract higher-level feature representations. The last layer of CNNs is a fully connected layer (FC), which gives the final output. For classification tasks, the Softmax operator is commonly used (Gu et al., 2018).

In traditional image classification, useful features must be extracted from the image, which are then used on machine learning algorithms like a Support Vector Machine (SVM). These handcrafted features must be carefully chosen, otherwise, the classification has a bad performance. This problem of carefully chosen features is not present in NNs, but they have other difficulties. One of the difficulties is the right choice of hyperparameters and of the architecture. These have direct influence on the performance.

In this thesis, we use a Convolutional Neural Network (ConvNet) to classify ULF wave events in four different categories. We develop our own ConvNet architecture that is simple and fast, and that can be implemented with a relatively small input dataset, consisting of small images, due to the limited resource of time and computational power. We use Swarm magnetic field measurements for the year 2015. Specifically, we use the magnetic field vector, measured in the North-East-Center (NEC) frame, with 1 second sampling rate. One Swarm magnetic field data file corresponds to 24-hour measurements. We cut these data per satellite track (i.e. from -90° to +90° latitude, ~45 minutes) to zoom in the occurring events that the satellite may measure, and then apply a high-pass filter to isolate the Pc3 ULF waves (20–100 mHz). Next, by performing a wavelet transform, we get the final spectra images. The four different categories we define are: “Pc3 ULF waves”, “Non-ULF signals”, “False Positives”, and “Plasma Instabilities”. For a good classification performance, ConvNets need a dataset of enough images that are carefully labeled. Therefore, we need to spend much time for the correct separation of the wavelet spectrum images in the four different categories, and especially for the “Plasma Instabilities” class definition, which presents difficulties due to its similarity with the Pc3 ULF wave class.

In the next chapter, information about the Swarm mission is presented and the basics of Earth’s magnetic field, magnetosphere, space weather and ULF waves is introduced. In Chapter 3, the basics of machine learning is introduced, and Artificial Neural Networks and Convolutional Neural Networks are explained. After this, in Chapter 4, the preprocessing pipeline for the preparation of the images before they are applied to the ConvNet is described. In the same chapter we also present our ConvNet architecture as well as the evaluation of the developed ConvNet and accuracy results. Finally, in the last chapter a conclusion of this thesis and an outlook for future work is given.

Chapter 2: The Swarm mission, Earth's magnetosphere & ULF waves

2.1. Swarm Mission

Swarm is a satellite constellation comprising three identical spacecraft that was launched on November 22, 2013 (12:02:29 UTC) on a Rockot vehicle from the Plesetsk Cosmodrome in north-western Russia. It is ESA's first constellation mission for Earth Observation (EO). The mission is operated by ESA's European Space Operations Centre (ESOC), in Germany, via the primary ground station in Kiruna, Sweden [1].

The objective of the Swarm mission is to provide the best-ever survey of the geomagnetic field and its temporal evolution, in order to gain new insights into the Earth system by improving our understanding of the Earth's interior and environment. Each of the three Swarm satellites are making high-precision and high-resolution measurements of the strength, direction and variation of the magnetic field, complemented by precise navigation, accelerometer, plasma and electric field measurements. These observations are provided as Level-1b data, which are the calibrated and formatted time series of e.g. the magnetic field measurements taken by each of the three Swarm satellites. These Level-1b data, as well as the higher-Level Swarm data products, are distributed by ESRIN (Frascati/I) (Olsen et al., 2013).

Swarm satellites obtain simultaneously a space-time characterization of both the internal field sources in the Earth and the ionospheric-magnetospheric current systems. The research objectives assigned to the mission are: (a) studies of core dynamics, geodynamo processes, and core-mantle interaction; (b) mapping of the lithospheric magnetisation and its geological interpretation; (c) determination of the 3-D electrical conductivity of the mantle; and (d) investigation of electric currents flowing in the magnetosphere and ionosphere (Olsen et al., 2013). A more precise description of the mission objectives follows in the next section.

2.1.1. Swarm Objectives

The primary research topics to be addressed by the Swarm mission include:

• Core dynamics, geodynamo processes, and core-mantle interaction: The goal is to improve the models of the core field dynamics by ensuring long-term space observations with an even better spatial and temporal resolution. Combining existing Ørsted, CHAMP and Swarm observations allows also more generally the investigation of all magnetohydrodynamic phenomena potentially affecting the core on sub-annual to decadal scales, down to wavelengths of about 2000 km. Of particular interest are those phenomena responsible for field changes that cannot be accounted for by core surface flow models.

• Lithospheric magnetization and its geological interpretation: The increased resolution of the Swarm satellite constellation allows, for the first time, the identification from satellite altitude of the oceanic magnetic stripes corresponding to periods of reversing magnetic polarity. Such a global mapping is important because the sparse data coverage in the southern oceans has

[1] http://earth.esa.int/swarm

been a severe limitation regarding our understanding of plate tectonics in the oceanic lithosphere. Another important implication of improved resolution of the lithospheric magnetic field is to derive global maps of heat flux.

• 3-D electrical conductivity of the mantle: Our knowledge of the physical and chemical properties of the mantle can be significantly improved if we know its electrical conductivity. Due to the sparse and inhomogeneous distribution of geomagnetic observatories, with only few in oceanic regions, a true global picture of mantle conductivity can only be obtained from space.

• Currents flowing in the magnetosphere and ionosphere: Simultaneous measurements at different altitudes and local times, as foreseen with the Swarm mission, allows better separation of internal and external sources, thereby improving geomagnetic field models. In addition to the benefit of internal field research, a better description of the external magnetic field contributions is of direct interest to the science community, in particular for space weather research and applications. The local time distribution of simultaneous data fostered the development of new methods of co-estimating the internal and external contributions.

The secondary research objectives include:

• Identification of the ocean circulation by its magnetic signature: Moving sea water produces a magnetic field, the signature of which contributes to the magnetic field at satellite altitude. Based on state-of-the-art ocean circulation and conductivity models it has been demonstrated that the expected field amplitudes are well within the resolution of the Swarm satellites.

• Quantification of the magnetic forcing of the upper atmosphere: The geomagnetic field exerts a direct control on the dynamics of the ionized and neutral particles in the upper atmosphere, which may even have some influence on the lower atmosphere. With the dedicated set of instruments, each of the Swarm satellites is able to acquire high-resolution and simultaneous in-situ measurements of the interacting fields and particles, which are the key to understanding the system [2].

As well as furthering science, the measurements delivered by the three Swarm satellites can be valuable for a range of other applications. For example, the data are used to help improve the accuracy of navigation systems including those systems carried on satellites or even to improve the efficiency of drilling for natural resources [3].

Table 1: The mission's primary and secondary objectives (https://earth.esa.int/swarm)

Swarm mission main objectives	
Mission's primary objectives	Studies of core dynamics, geodynamo processes and core-mantle interaction Mapping of the lithospheric magnetisation and its geological interpretation Determination of the 3D electrical conductivity of the mantle Investigation of electric currents flowing in the magnetosphere and ionosphere
Mission's secondary objectives	Identifying the ocean circulation by its magnetic signature Quantifying the magnetic forcing of the upper atmosphere

[2] EO Portal Directory (Earth Observation information discovery platform), Satellite Missions Database, https://earth.esa.int/eoportal/swarm

[3] Swarm: ESA's Magnetic Field Mission brochure, 2013, Online, https://esamultimedia.esa.int/BR-302/

2.1.2. Swarm satellite orbit

As well as the advanced technology the satellites carry, their carefully selected orbits are essential to the success of the mission. Two of the Swarm satellites, Swarm Alpha and Swarm Charlie, are flying almost side-by-side in near-polar orbits of inclination 87.4° at an altitude of about 465 km (in November 2015) – but descending to around 300 km over the life of the mission – and above a mean radius of 6371.2 km. The East–West separation of their orbits is 1.4° in longitude, corresponding to 155 km at the equator. The third satellite, Swarm Bravo, flies at a slightly higher (about 520 km altitude in November 2015) orbit of inclination 88° (Olsen et al., 2016).

The lower the satellites are, the more sensitively they can measure small magnetic features in the crust. The Sun generates typical day and night patterns in the ionosphere between the satellites and Earth. Magnetic storms resulting from solar activity also cause irregular disturbances in the ionosphere and magnetosphere. The satellites' drifting orbits mean that all the magnetic signals originating from Earth and those caused by the Sun are captured.

2.1.3. Swarm satellite structure & instruments

The three identical satellites have a rather unusual shape: trapezoidal with a long boom that was deployed once they went in orbit. Developed on behalf of ESA by an industrial consortium led by EADS Astrium GmbH, each satellite is about 9 m long, including the boom, with the surface at the front only measuring about 1 m^2. This is to reduce the effect of air drag and to cut down on the amount of propellant needed to stay at the correct altitude. Below around 500 km, air drag tends to slow satellites down and lower the orbit. The boom, which accounts for almost half the length of the satellite, trails at the back. This is because the front surface is needed for the electric field instrument so that it can collect and measure the speed and direction of incident ions along the orbital path. Once the boom has been deployed, the satellite has no moving parts. This ensures that there are no vibrations that could influence the measurements made by the accelerometer, which is fixed at the very centre of the satellite. Likewise, the solar panels are fixed, forming the satellite 'roof'. Magnetic cleanliness is of paramount importance to the mission, so the sensitive scalar magnetometers are mounted at the end of the boom, far away from any magnetic disturbance that the electrical units on the body may cause. The optical bench holding the VFM and the three startrackers is mounted halfway along the boom.

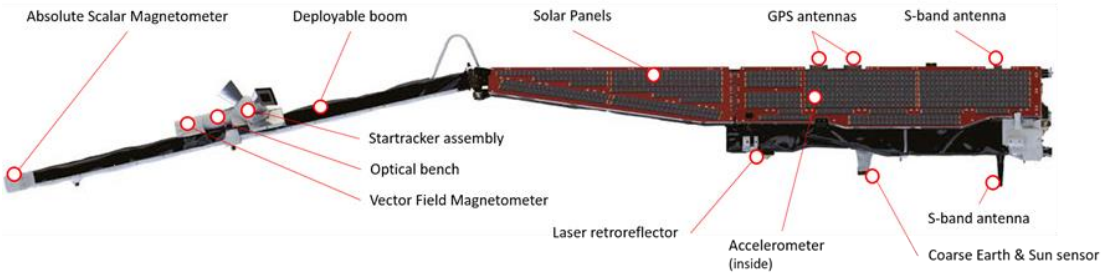


Figure 1: Side-view of instrumentation on the Swarm satellites. The satellite flies with the boom trailing at the rear. (http://earth.esa.int/swarm)

Table 2: The on-board sensors of the Swarm satellites. (https://earth.esa.int/swarm)

Swarm satellite sensors	
Vector Field Magnetometer (**VFM**)	Provides vectorial measurements of the magnetic field
Advanced Scalar Magnetometer (**ASM**)	Provides the total magnetic field (used to calibrate vector measurements); experimental vector mode
Electric Field Instruments (**EFI**)	Provides (after processing) vectorial electric field measurements based on ion temperature & density, electron temperature
Accelerometer (**ACC**)	Provides 3 degree-of-freedom measurements of non-conservative forces; used to derive atmospheric density and winds
GPS Receiver (**GPSR**)	Provides 8-channel positioning data for precise orbit determination (positioning)
Laser Retro-Reflector (**LRR**)	Provides positioning information through ground-based satellite laser ranging (via ILRS)
Star Tracker (**STR**)	Provides attitude data that determines the orientation of the optical bench

Each of the three Swarm satellites is equipped with magnetic sensors, which can be thought of as a 3D compass. Each provides precise and detailed measurements of the strength and direction of the magnetic field. Magnetic sensors measure a tangle of the core field with other signals from magnetised rocks in the crust, electrical currents flowing in the ionosphere, magnetosphere and oceans, and currents inside Earth induced by external fields. The challenge here is to separate the different sources of magnetism. GPS receivers, an accelerometer and an electric field instrument deliver supplementary information to study the interaction between Earth's magnetic field and the solar wind. [4]

Specifically, each of the three satellites carries an Absolute Scalar Magnetometer (ASM) measuring Earth's magnetic field intensity, a Vector Fluxgate Magnetometer (VFM) measuring the magnetic vector components and a three-head Star Tracker (STR) mounted close to the VFM to obtain the attitude needed to transform the vector measurements of the VFM magnetometer to a known coordinate frame. Time and position are obtained by on-board GPS (Olsen et al., 2016). All the sensors onboard the Swarm satellites are presented in Table 3. A more precise description of the Swarm magnetometers (VFM and ASM) follows on the next paragraphs.

Vector Field Magnetometer: This magnetometer is the mission's core instrument. It makes high-precision measurements of the magnitude and direction of the magnetic field, i.e. the field's vector. The orientation of the vector is determined by the startracker assembly, which provides attitude data. The VFM and the startrackers are both housed on an ultra-stable structure called an *optical bench*, halfway along the satellite's boom. The design of the bench and carefully selected material make it possible to keep the instruments aligned to 1 arcsec.

Absolute Scalar Magnetometer: This novel instrument measures the strength of the magnetic field to greater accuracy than any other magnetometer. The ASM is an 'optically-pumped metastable helium-4 magnetometer', developed and manufactured by CEA-Leti in France

[4] Swarm: ESA's Magnetic Field Mission brochure, 2013, Online, https://esamultimedia.esa.int/BR-302/

under contract with the French space agency, CNES. It provides scalar measurements of the magnetic field to calibrate the VFM [5].

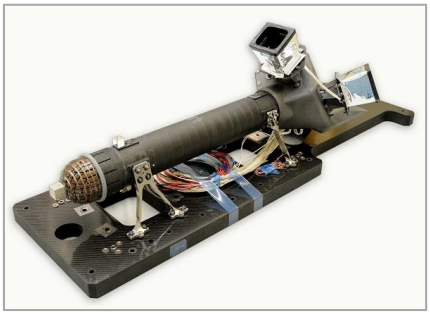


Figure 2: The Vector Field Magnetometer (VFM) measures the magnetic field vector at the tip of the optical bench on the boom. (https://earth.esa.int/swarm)

2.1.4. Swarm Data Products

The Swarm Level 1b and Swarm Level 2 data products include Swarm magnetic field models, ionospheric and thermospheric products, and Precise Orbit Solutions including supporting information.

The Swarm Level 1b data products are the corrected and formatted output from each of the three Swarm satellites. By a complex assimilation of these individual satellite measurements into one set of products for the satellite constellation, the Swarm Level 2 Processor ensures a very significant improvement of the quality of the final scientific data products.

In accordance with ESA Earth Observation Data Policy, all Swarm Level 1b and Level 2 products are freely accessible to all users via anonymous access. They can be downloaded:

- via any HTTP browser at http://swarm-diss.eo.esa.int
- directly via an ftp client at ftp://swarm-diss.eo.esa.int

The top-level structure is divided into the 'Level1b', 'Level2daily', and 'L2longterm' and 'Advanced' directories. Within the 'Level1b' and 'Level2daily' directories, the structure presents first the 'Latest baselines' and 'Entire_mission_data' folders, then a Simplified datatype list, and finally, the satellite ID, if applicable. A short explanation on these data follows in the next paragraphs.

- LATEST BASELINES FOLDER: the user will find all the interoperable products generated in a consistent way with the application of all significant data quality improvements, but not necessarily covering the entire mission.
- ENTIRE MISSION DATA FOLDER: the users will find the full data coverage of the entire mission, regardless of any consideration of interoperability among the same product type. In other words, the data might have been produced with strategies or knowledge that does not make the products necessarily interoperable for certain types of science. This information can still be captured by looking at the Product Baseline number.
- PRODUCT BASELINE: is a number associated with a specific product type and satellite. The Product Baseline is identified by the first two of the four digits placed at the end of the file name, i.e. the first two digits of the File_Version field represent the Product Baseline, while the last two represent an incremental File Counter. Here is an example: "*SW_OPER_STRCATT_1B_20140218T000000_20140218T235959_0301_MDR_SAT_AT.cdf*" where 03 is the Product Baseline number, while 01 is only an incremental File Counter [6].

[5] Swarm: ESA's Magnetic Field Mission brochure, 2013, Online, https://esamultimedia.esa.int/BR-302/

[6] https://earth.esa.int/swarm/data-handbook

2.2. The Earth's Magnetic Field

As already mentioned, the Swarm mission aims to unravel one of the most mysterious aspects of our planet: the magnetic field. Although invisible, the magnetic field and electric currents in and around Earth generate complex forces that have immeasurable impact on everyday life. The field can be thought of as a huge bubble, protecting us from cosmic radiation and the charged particles that bombard Earth through the solar wind. Without this protective shield, the atmosphere as we know it would not exist, rendering life on Earth practically impossible.
Even as it is, strong solar storms have the potential to cause power and communication blackouts and can also damage satellites orbiting Earth. A visible display of what happens when charged particles collide with atoms and molecules in the upper atmosphere can be seen as waves of luminous green light in the polar skies – the *aurora borealis* in the north and *aurora australis* in the south.
Earth's magnetic field is in a permanent state of flux. Magnetic north wanders, and every few hundred thousand years the polarity flips so that a compass would point south instead of north. Moreover, the strength of the magnetic field constantly changes – and it is currently showing signs of significant weakening.
By analysing the different characteristics of the observed field, the Swarm mission has led to new insight into many natural processes, from those occurring deep inside the planet, to space weather caused by solar activity. In turn, this information can yield a better understanding of why the magnetic field is weakening.
In simple terms, Earth's magnetic field behaves as if there were a powerful dipolar bar magnet at the centre of the planet, tilted at about 11° to the axis of rotation. In reality, however, the processes involved in generating the field are far more complex. The magnetic field is thought to be largely generated by an ocean of superheated, swirling liquid iron that makes up the outer core 3000 km under our feet. Acting like the spinning conductor in a bicycle dynamo, it generates electrical currents, which in turn, generate our continuously changing electromagnetic field. Other sources of magnetism are the minerals in Earth's mantle and crust, while the ionosphere and the magnetosphere also play an important role. Since salt water is conductive, oceans make an additional, albeit weak, contribution to the magnetic field.
The geomagnetic field is far from static and varies both in strength and direction. For example, recent studies have shown that the position of the north magnetic pole is changing rapidly. Furthermore, over the last 200 years, the magnetic field has lost around 9% of its strength on a global average. A large region of reduced magnetic intensity has developed between Africa and South America and is known as the *South Atlantic Anomaly* [7].

[7] Swarm: ESA's Magnetic Field Mission brochure, 2013, Online, https://esamultimedia.esa.int/BR-302/

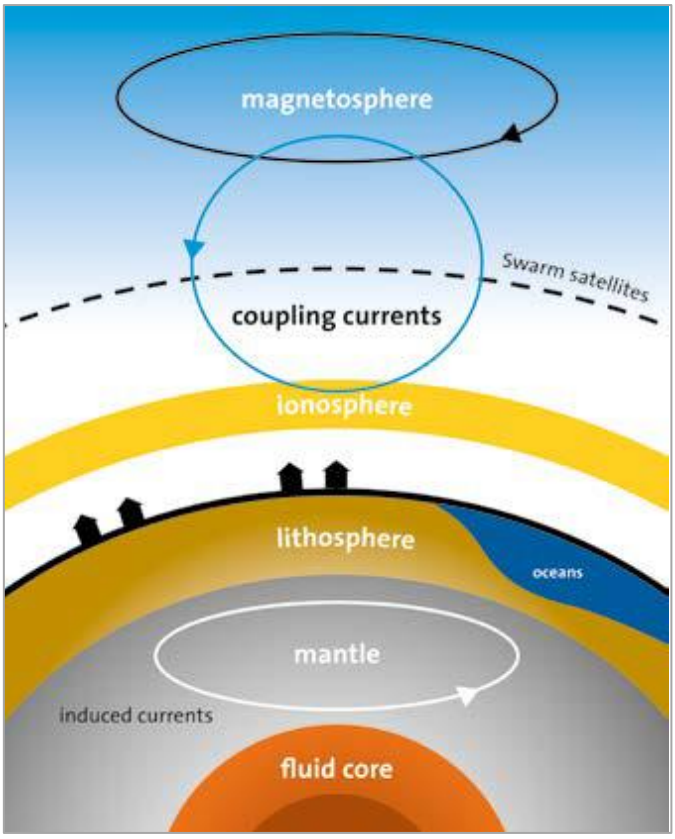


Figure 3: The different sources that contribute to the magnetic field measured by Swarm. The coupling currents or field-aligned currents flow along magnetic field lines between the magnetosphere and ionosphere. The ionosphere is 85–600 km above Earth, while the magnetosphere is 60 000–120 000 km from Earth. (ESA/DTU Space)

2.2.1. Structure of Earth's magnetosphere

The geospace environment extends from the Sun's surface to the Earth's ionosphere. The dynamics of the Geospace region are governed by the interaction between the Earth's magnetic field and the solar wind. In this region all matter is in the plasma state[8]. The region over which the geomagnetic field is a dominant influence is the magnetosphere. Earth's magnetosphere is never in a truly steady state. It is a dynamic and constantly changing structure (Walker et al., 2005). The individual parts of the magnetosphere are described below.

Bow shock: It forms the outermost layer of the magnetosphere; the boundary between the magnetosphere and the ambient medium.

[8] In physics, the word *plasma* designates a fully or partially ionized gas consisting of electrons and ions. The term plasma was introduced 80 years ago by Irving Langmuir (1881–1957) to describe the charge-neutral part of a gas discharge. David A. Frank-Kamenezki identified plasma as the fourth state of matter. From a phenomenological point of view, the identification of plasma as a new state of matter can be justified because the splitting at high temperature of neutral atoms into electrons and ions is associated with a new energy barrier, the ionization energy. Today we know that plasma is not only the hot, disordered state of matter described above. Rather, we have learned during the last 20 years that plasma systems can attain gaseous, liquid and even solid phases. The plasma state, as an electrically conductive medium, possesses a number of new properties that distinguish it from neutral gases and liquids. The plasma state is a gaseous mixture of positive ions and electrons. Plasmas can be fully ionized, as the plasma in the Sun, or partially ionized, as in fluorescent lamps, which contain a large number of neutral atoms (Piel, 2017).

Magnetosheath: It is the region of the magnetosphere between the bow shock and the magnetopause. It is formed mainly from shocked solar wind, though it contains a small amount of plasma from the magnetosphere. It is an area exhibiting high particle energy flux, where the direction and magnitude of the magnetic field varies erratically.
Magnetopause: It is the area of the magnetosphere wherein the pressure from the planetary magnetic field is balanced with the pressure from the solar wind. It is the convergence of the shocked solar wind from the magnetosheath with the magnetic field of the object and plasma from the magnetosphere. Because both sides of this convergence contain magnetized plasma, the interactions between them are complex. The magnetopause changes size and shape as the pressure from the solar wind fluctuates.
Magnetotail: Opposite the compressed magnetic field is the magnetotail, where the magnetosphere extends far beyond the Earth. It contains two lobes, referred to as the northern and southern tail lobes. The northern tail lobe points towards the Earth and the southern tail lobe points away. The tail lobes are almost empty, with few charged particles opposing the flow of the solar wind. The two lobes are separated by a *plasma sheet*, an area where the magnetic field is weaker and the density of charged particles is higher [9].

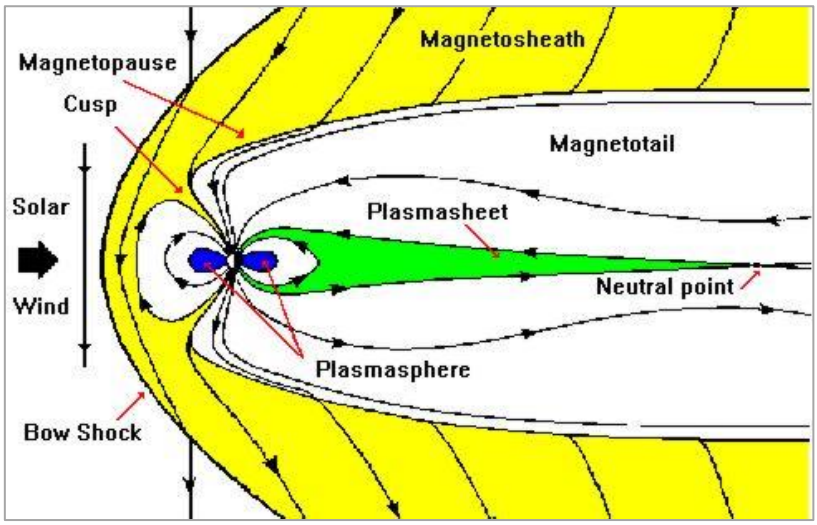


Figure 4: A simplified representation of the structure of Earth's magnetosphere (High Altitude Observatory (HAO), National Center for Atmospheric research (NCAR), Colorado, www2.hao.ucar.edu)

2.2.2. Solar Wind & Space Weather

The solar wind consists of plasma streaming outwards from the Sun. It can be regarded as the outer part of the Sun's atmosphere. The shape of the Earth's magnetosphere is the direct result of being blasted by solar wind [10]. It carries with it a frozen-in magnetic field originating from the Sun. This field is variable in magnitude and direction. It has been extensively observed by in situ satellites which provide information on the plasma composition, velocity, temperature, and density, and the three components of the magnetic field. The irregular and gusty nature of the solar wind observed near the Earth means that magnetic field, plasma velocity, density, and temperature vary on a wide range of spatial and temporal scales (Walker et al., 2005).
The term "space weather" is to describe the dynamic conditions in the Earth's outer space environment, in the same way that "weather" and "climate" refer to conditions in Earth's

[9] Space Environment lecture notes 2019, National Observatory of Athens, MSc in SSTA
[10] https://www.nasa.gov/sunearth/magnetosphere

lower atmosphere. Space weather includes all conditions and events on the Sun, in the solar wind, in near-Earth space and in our upper atmosphere that can affect space-borne and ground-based technological systems and through these, human life and endeavor.
Modern society depends on a variety of technologies susceptible to the extremes of space weather. Strong electrical currents driven along the Earth's surface during auroral events disrupt electric power grids and contribute to the corrosion of oil and gas pipelines. Changes in the ionosphere during geomagnetic storms interfere with high-frequency radio communications and Global Positioning System (GPS) navigation. During polar cap absorption events caused by solar protons, radio communications can be compromised for commercial airliners on transpolar crossing routes. Exposure of spacecraft to energetic particles during solar energetic particle events and radiation belt enhancements cause temporary operational anomalies, damage critical electronics, degrade solar arrays, and blind optical systems such as imagers and star trackers.
Human and robotic explorers across the solar system are also affected by solar activity. Research has shown, in a worst-case scenario, astronauts exposed to solar particle radiation can reach their permissible exposure limits within hours of the onset of an event. Surface-to-orbit and surface-to-surface communications are sensitive to space weather storms [11].

2.2.3. ULF waves

Ultra-low frequency (ULF) waves are magnetohydrodynamic (MHD) plasma waves in the frequency range of approximately 1 mHz ≤ f ≤ 10 Hz. Generated by a variety of instabilities, ULF waves transport and couple energy throughout the system, and are readily recorded throughout the Earth's magnetosphere and on the ground.
ULF waves provide a convenient probe and diagnostic monitor of the magnetosphere. The availability of multipoint measurements from spacecraft, ionospheric sounders and ground magnetometer arrays and the increasing sophistication of modeling tools have stimulated much recent progress in this area. Nevertheless, fundamental questions remain regarding the generation, propagation and consequences of these waves (Menk et al., 2011).
ULF waves are also of great importance for Space Weather. They play a critical role in magnetospheric dynamics due to wave-particle interactions in radiation belts, responsible for electron acceleration and loss [11].
Magnetospheric ULF waves are large-scale phenomena – they can be excited by the solar wind with a scale size of the whole magnetosphere and can have a strong impact on charged particle dynamics in the radiation belts (Mann, 2016). For instance, radial diffusion caused by ULF waves may accelerate electrons with MeV energies in the radiation belts. These electrons may penetrate spacecraft shielding, generate a build-up of static charge within electrical components resulting in subsystem damage, and ultimately can even cause the total loss of Earth-orbiting satellites (Mann, 2016). Therefore, studies of the excitation, propagation and global characteristics of ULF waves, as well as their impact on MeV electron dynamics remain an active area of research (Mann, 2016). There has been also a number of studies addressing the monitoring of ULF waves from low-Earth orbit (LEO) satellites (for recent reviews see Balasis et al., 2015; Papadimitriou et al., 2018) as well as from the ground (Bogoutdinov et al., 2018).
In principle, simultaneous observations at many locations are needed to understand in depth their generation and propagation. In particular, continuous pulsations with periods in the range 0.2 to 600 s, denoted as Pc 1-2 (0.2–10 s), Pc 3 (10–45 s), Pc 4 (45–150 s), and Pc 5 (150–

[11] https://www.nasa.gov/sunearth/spaceweather

600 s), have been extensively studied using measurements from both space-borne and ground-based instruments for many years [12].

ULF plasma waves are broadly of two types, depending on whether their energy source originates in the solar wind or from processes within the magnetosphere. Evidence for the former comes from the dependence of daytime power in the Pc3 (20–100 mHz), Pc4 (7–20 mHz) and Pc5 (1.7–7 mHz) ranges on solar wind speed and interplanetary magnetic field (IMF) clock angle (e.g. Odera 1986; Engebretson et al. 1987; Mathie and Mann 2001; Kessel et al. 2004; Francia et al. 2009). Solar wind density also plays an important role in controlling Pc3 activity (Heilig et al. 2010). Substorms and other instabilities in the tail form an important source of ULF waves on the nightside (Menk et al., 2011).

Table 4: Classification of magnetospheric pulsations wrt their frequency (Jacobs et al., JGR 1964)

ULF pulsations		Period range (s)	Frequency (mHz)
Continuous pulsations	Pc1	0.2 - 5	200-500
	Pc2	5 - 10	100-200
	Pc3	10 - 45	22-100
	Pc4	45 – 150	7-22
	Pc5	150 - 600	2-7
Impulsive pulsations	Pi1	1 - 40	25-200
	Pi2	40 - 150	7-25

[12] Space Environment lecture notes 2019, National Observatory of Athens, MSc in SSTA

Chapter 3: Deep Learning & Convolutional Neural Networks

3.1. Introduction

In recent years, deep learning has garnered tremendous success in a variety of application domains. This new field of machine learning has been growing rapidly, and has been applied to most traditional application domains, as well as some new areas that present more opportunities. Different methods have been proposed based on different categories of learning, including supervised, semi-supervised, and unsupervised learning. Experimental results show state-of-the-art performance using deep learning when compared to traditional machine learning approaches in the fields of image processing, computer vision, speech recognition, art, medical imaging, bio-informatics, natural language processing (NLP), and many others (Alom et al., 2018). This chapter starts with an introduction of the general concepts related to Deep Neural Networks (DNN) and concludes focusing on Convolutional Neural Networks (CNN), which is the core methodology used in this thesis.

Machine learning (ML) in general is in its golden age today for the simple reason that methods, algorithms, and tools, studied and designed during the last two decades have started to produce unexpectedly good results, exploiting the historically unique combination of big data availability and cheap computing power. Practically, we can think of machine learning as the set of methods and algorithms that can be used for the following problems: (1) make predictions in time or space of a continuous quantity (regression); (2) assign a datum to a class within a pre-specified set (classification); (3) assign a datum to a class within a set that is determined by the algorithm itself (clustering); (4) reduce the dimensionality of a dataset, by exploiting relationships among variables; and (5) establish linear and nonlinear relationships and causalities among variables (Camporeale et al., 2018).

Learning is a key importance concept in ML. More specifically, the learning procedure estimates the parameters of a certain model that describes the task under study, so that the learned model (the model having as parameter values, those estimated by the learning process) can perform a specific task. For example, in Artificial Neural Networks (ANN), the parameters are the weights of the synapses between nodes. A part of increased significance of Machine Leanring (ML) developed largely from 2006 onward, is that of Deep Learning (DL). The associated with DL architectures are very complex, a fact that gives them the ability to learn very complex tasks, such as, feature learning and pattern classification (Schmidhuber, 2014; LeCun et al., 2015).

DL on the other hand consists of several layers in between the input and output layer which allows for many stages of non-linear information processing units with hierarchical architectures to be present that are exploited for

ML and its newer subfield, DL, has been utilized in space weather applications at least since the 1990s. For example, several attempts have been made to use neural networks and linear filters for predicting geomagnetic indices and radiation belt electrons (Baker, 1990; Valdivia et al., 1996; Sutcliffe, 1997; Lundstedt, 1997, 2005; Boberg et al., 2000; Vassiliadis, 2000; Gleisner and Lundstedt, 2001; Li, 2001; Vandegriff, 2005; Wing et al., 2005). A feature that makes space weather very remarkable and perfectly fit in the machine learning framework is

the huge amount of data that is usually publicly available and that the released datasets are often of very high quality and require only a small amount of preprocessing.
The machine learning applications to space weather and space physics can generally be divided into the following categories (Camporeale et al., 2018):

• *Automatic event identification:* Space weather data is typically imbalanced, with many hours of observations covering uninteresting/quiet times, and with only a small percentage of data of useful events. The identification of events is still often carried out manually, following time-consuming and non-reproducible criteria. As an example, techniques such as convolutional neural networks can help in automatically identifying interesting regions like solar active regions, coronal holes, coronal mass ejections, and magnetic reconnection events, as well as to select features.

• *Knowledge discovery:* Methods used to study causality and relationships within highly dimensional data, and to cluster similar events, with the aim of deepening our physical understanding. Information theory and unsupervised classification algorithms fall into this category.

• *Forecasting:* Machine learning techniques capable of dealing with large class imbalances and/or significant data gaps to forecast important space weather events from a combination of solar images, solar wind, and geospace in situ data.

• *Modeling:* This is somewhat different from forecasting and involves a higher level approach where the focus is on discovering the underlying physical and long-term behavior of the system. Historically, this approach tends to develop from reduced descriptions based on first principles, but the methods of machine learning can in theory also be used to discover the nonlinear map that describes the system evolution.

3.2. The image classification problem

Since our task in this thesis is to classify a set of images, we are solving an image classification problem: the task of assigning to an input image a label out of a fixed set of categories. This is one of the core problems in Computer Vision that, despite its simplicity, has a large variety of practical applications. Moreover, many other seemingly distinct Computer Vision tasks (such as object detection, segmentation) can be reduced to image classification.
An image classification model takes a single image and assigns probabilities to a number of labels. Hence, our task is to turn a considerable amount of numbers (which represent the intensity of the pixels of the image) into a single label, such as "*ULF-wave Event*".
The approach we follow is referred to as a *data-driven approach*, since it relies on first accumulating *a training dataset* of labeled images. The complete image classification pipeline can be formalized as follows[13]:
Input: Our input consists of a set of *N* images, each labeled with one out of *K* different classes. We refer to this set as the *training set*.
Learning: Our task is to use the training set to learn what every one of the classes looks like (this implicitly assumes that a sort of knowledge concerning each class will be built). We refer to this step as *training a classifier*, or *learning a model*.
Evaluation: In the end, we evaluate the performance of the classifier by asking it to predict labels for images with a priori known labels, which have not been used for the training of the

[13] *F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class notes CS231n: Convolutional Neural Networks for Visual Recognition" (2020), http://cs231n.stanford.edu/.*

model (they constitute the test set). We will then compare the true labels of these images to the ones predicted by the classifier. Intuitively, we're hoping that a lot of the predictions match up with the true answers (which we call the *ground truth*). Specifically, high classification rates implies that the model generalizes well on images that were not used for its training, and thus, it can be trusted for operational purposes (where we do not know the label of a certain image and we rely on the classification performed by the classifier).

3.3. (Parameterized) Mapping from Images to Label Scores

We could start by defining a simple classifier for image classification that we will eventually naturally extend to entire Neural Networks and Convolutional Neural Networks. The approach will have two major components: a ***score function*** that maps the raw data to class scores, and a ***loss function*** that quantifies the agreement between the predicted scores and the ground truth labels. We will then cast this as an optimization problem in which we will minimize the loss function with respect to the parameters of the score function.

3.3.1. The score function

The first component of this approach is to define the score function that maps the pixel values of an image to confidence scores for each class. We assume a training dataset of images $x_i \in \mathbb{R}^D$, each associated with a label y_i with $i = 1 \dots N$ and $y_i \in \{1 \dots K\}$. That is, we have *N* examples (each with a dimensionality *D*) and *K* distinct categories. We will now define the score function $f: \mathbb{R}^D \longrightarrow \mathbb{R}^K,\ f(x_i\ ; W),$ that maps the raw image pixels to *class scores*, i.e., a k-dimensional vector containing the scores associated with the classes for the image under study (hopefully, the class with the highest score will be the correct one for the image at hand).

The simplest possible function is a ***linear mapping***: [14]

$$f(x_i\ ; W, b) = Wx_i + b \tag{2.1}$$

In the above equation, we are assuming that the image x_i has all of its pixels flattened out to a single column vector of shape [*D* x 1]. The matrix ***W*** (of size [*K* x *D*]), and the vector ***b*** (of size [*K* x 1]) are the **parameters** of the function. The parameters in ***W*** are often called the **weights**, and ***b*** is called the **bias vector** because it influences the output scores, but without interacting with the actual data x_i.

Our goal will be to set these parameters in such way that the computed scores match the ground truth labels across the whole training set. Specifically, we wish the correct class to have a score that is higher than the scores of incorrect classes.

Looking at the above equation, we can say that a *linear score function* computes the score of a class as a ***weighted sum*** of all of the pixel values of the image across all of its color channels. Convolutional Neural Networks will map image pixels to scores exactly as shown above, but the mapping (f) will be more complex and will contain more parameters.

3.3.2. The loss function

In the previous section we defined a function from the pixel values to class scores, which was parameterized by a set of weights (*W* and *b*). Now, we want to set those weights so that *the*

[14] In contrast to the general notation used before, in this specific case the parameters are contained in the matrix *W* and the vector *b*.

predicted class scores are consistent with the *ground truth labels* in the training data. To do so, we define a *loss function*, through which we will compare the score $f(x_i; W, b)$ with the true label y_i. Intuitively, high loss values indicate poor quality data classification, while low loss values indicate good classification ability. Hence, in order to find the best possible values for these weights, we need to minimize our loss function. This can be done through certain optimization methods, as will be discussed later.

Summary		
Training data:	$\{x_i, y_i\}$	$i = 1, \dots, N,\ x_i \in \mathbb{R}^D,\ y_i \in \{1 \dots K\}$
Prediction:	$f(x_i, W, b)$	W, b - learned parameters
Loss:	$L(f(x_i, W, b), y_i)$	small loss $\Rightarrow$ good predictions

Cross-entropy loss, or log loss, which measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. A perfect model would have a log loss of 0 [15].

The cross-entropy formula takes in two distributions, $p(x)$, the true distribution, and $q(x)$, the estimated distribution, defined over a discrete variable x, and is given by

$$H(p, q) = -\sum_{x} p(x)\, log(q(x)) \tag{2.2}$$

For a neural network, we usually see the equation written in a form where $\boldsymbol{y}$ is the ground truth vector and $\hat{\boldsymbol{y}}$ is the estimate. For a single example, it would look like this:

$$L = -y \cdot log(\hat{y}) \tag{2.3}$$

where · is the inner product. We often see this equation averaged over all examples as a ***cost function***. It is not always strictly adhered to in descriptions, but usually a loss function is lower level and describes the degree of accuracy the adopted model responds to a single instance or component, whilst a cost function is higher level and describes the degree of accuracy of the adopted model over the while data set. A cost function based on multiclass log loss for a data set of size *N* might look like this[16]:

$$J = -\frac{1}{N}\left(\sum_{i=1}^{N} y_i \cdot log(\hat{y}_i)\right) \tag{2.4}$$

3.4. Artificial Neural Networks (ANNs)

Artificial neural networks are models inspired by the way the human brain is constructed. Their building block is the artificial neuron, which try to mimic the behavior of the human brain cells. The basic structure of a node is shown in Figure 1. Specifically, the node receives inputs from external sources, and has some internal parameters (including weights and biases that are learned during training) which adjust the outputs. This computational unit is called a perceptron (Alom et al., 2018). Perceptrons are simple examples of the so-called *learning machines*, that is, structures whose free parameters are updated by a *learning algorithm*, in order to "learn" a specific task, based on a set of training data (Theodoridis & Koutroumbas, 2003).

[15] https://ml-cheatsheet.readthedocs.io/en/latest/
[16] https://datascience.stackexchange.com/

Returning to Figure 1, let us describe the basic operation of **a single neuron**. We can see two distinct operations: the one is a *linear* combination of input features and parameters, $z = w^T x + b$, and the other is a *nonlinear* operation, performed by an activation function, $g(z)$, such as a sigmoid. The most well-known class of ANNs are the Multilayer Perceptrons (MLP), which contain one or more hidden layers with multiple hidden units (neurons) in each of them.

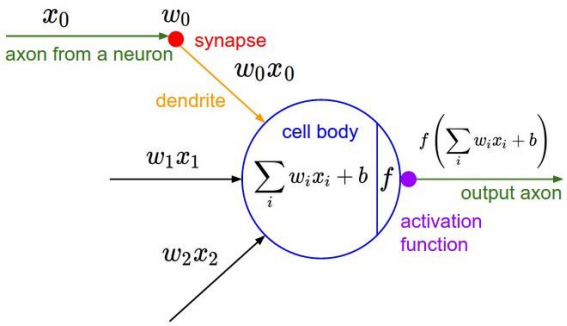


Figure 5: Basic model of an artificial neuron and its relationship with the human brain. (http://cs231n.stanford.edu/)

The notation and indices used in Neural Networks is the following[17]: $foo^{[\ell]}$ with brackets denoting anything associated with layer ℓ, $x^{(i)}$ with parenthesis refers to the i^{th} training example, and $a_j^{[\ell]}$ referring to the activation of the j^{th} unit in layer ℓ.

For example, the *first hidden unit* in the *first hidden layer* will perform the following computation:

$$z_1^{[1]} = W_1^{[1]} x + b_1^{[1]} \quad \text{and} \quad a_1^{[1]} = g\left(z_1^{[1]}\right) \tag{2.5}$$

where W is a matrix of parameters and W_1 refers to the first row of this matrix. The parameters associated with the first hidden unit is the vector $W_1^{[1]} \in \mathbb{R}^N$ and scalar $b_1^{[1]} \in \mathbb{R}$. The *second hidden unit* in the *first hidden layer* will perform:

$$z_2^{[1]} = W_2^{[1]} x + b_2^{[1]} \quad \text{and} \quad a_2^{[1]} = g\left(z_2^{[1]}\right) \tag{2.6}$$

and so on. Each hidden unit has its corresponding parameters W and b.

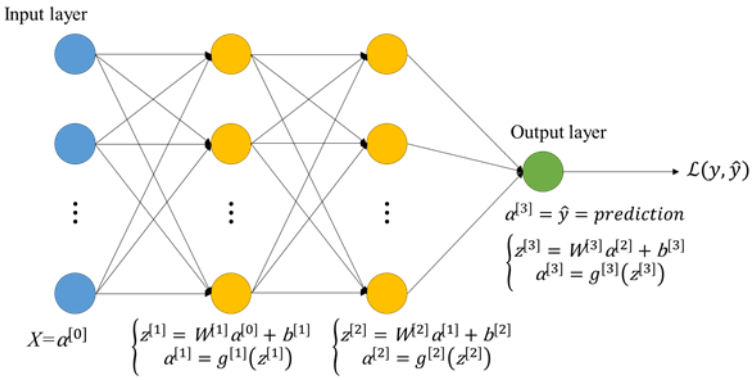


Figure 6: Basic architecture of a Neural Net.

In practice, when implementing the algorithm, we want to avoid for-loops in order to perform efficiently the neural network computations. The solution to this is to transform the input data, as well as the parameters and the activation functions in a vectorized form.

[17] Ng A. & Katanforoosh K. (2018), "Stanford lecture notes CS229: Deep Learning", http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

So, for a single training example, it is represented by (x, y), where $x \in \mathbb{R}^{n_x}$ and label $y \in \mathbb{R}^{n_y}$, where $\mathbb{R}^{n_x}$ and $\mathbb{R}^{n_y}$ are sets of real n-vectors (nx1 matrices). But for the training set m, with training examples $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$, we define a $n_x \,\mathrm{x}\, m$ dimensional matrix, by stacking them all together in columns:

$$X = \begin{bmatrix} | & | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | & | \end{bmatrix} \quad , \; X \in \mathbb{R}^{n_x \,\mathrm{x}\, m} \tag{2.7}$$

The labels y are usually converted to *one-hot encoding*, e.g., in the case we have 4 classes,

Labels y	0	1	2	3
one-hot encoding	$\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

As for the parameters, for each layer, $W^{[\ell]}, b^{[\ell]}$, we are stacking them as follows:

$$W^{[\ell]} = \begin{bmatrix} - & W_1^{[\ell]T} & - \\ - & W_2^{[\ell]T} & - \\ & \vdots & \\ - & W_n^{[\ell]T} & - \end{bmatrix}, \qquad b^{[\ell]} = \begin{bmatrix} b_1^{[\ell]T} \\ b_2^{[\ell]T} \\ \vdots \\ b_n^{[\ell]T} \end{bmatrix} \tag{2.8}$$

We can then combine this into a single unified formulation:

$$Z^{[\ell]} = \begin{bmatrix} | & | & | & | \\ z^{[\ell](1)} & z^{[\ell](2)} & \dots & z^{[\ell](m)} \\ | & | & | & | \end{bmatrix} = W^{[\ell]}X + b^{[\ell]} \tag{2.9}$$

where in this case, we refer to the first layer where the input is the training examples ($\ell = 0$, $a^{[0]} = X$). To compute the activation $a^{[1]}$ without a for-loop, we can leverage vectorized libraries in Matlab or Python which compute $a^{[1]} = g(z^{[1]})$ very fast by performing parallel element-wise operations. For example, we defined the sigmoid function $g(z)$ as:

$$g(z) = \frac{1}{1+e^{-z}} \quad \text{where } z \in \mathbb{R} \tag{2.10}$$

However, the sigmoid function can be defined not only for scalars but also for vectors. In a Matlab-like pseudocode, we can define the sigmoid as:

$$g(z) \; = \; 1 \,./\, (1 + exp(-z)) \quad \text{where } z \in \mathbb{R}^n \tag{2.11}$$

where ./ denotes element-wise division. With this vectorized implementation, $a^{[1]} = g(z^{[1]})$ can be computed very quickly.
Finally, for any layer ℓ, its output can be represented mathematically as follows:

$$a^{[\ell]} = g^{[\ell]}(W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}) \tag{2.12}$$

Where $a^{[\ell-1]}$ represents the concatenation of the activations computed by all the neurons of the previous layer.

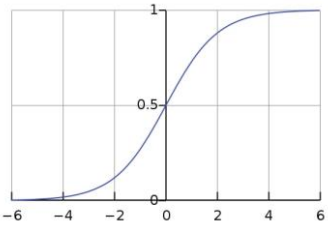
3.5. Output Activation Functions

The activation functions used in ANNs play an important role in the convergence of the learning algorithms (Gomes et al., 2011; Dabal Pedamonti, 2018), and enhance their representation capabilities. As we already saw, these functions transform the output score vectors before the loss computation in the training phase. An activation function introduces the non-linearity in the model. Without non-linear activation functions, the neural network will simply perform linear regression [18]. An activation has to be a continuous differentiable function. As we will see later, differentiation enters into the scene as a requirement for performing cost function minimization (Theodoridis & Koutroumbas, 2003).

a) Sigmoid

A popular family of continuous differentiable functions, which approximate the step function, is the family of *sigmoid functions*. A typical representative is the *logistic function* (Theodoridis & Koutroumbas, 2003). Its domain is the set of real numbers while its range of values is the interval (0,1). It is often used in ANNs in binary classification problems. Sigmoid is applied independently to each element of the output vector, z_j, of a certain layer of nodes:

$$f(z_j) = \frac{1}{1 + e^{-z_j}} \tag{2.13}$$



One of the problems of using sigmoid functions during the training, arises in the regions where the slope of the function is nearly zero, meaning that learning becomes really slow when gradient descent is implemented. In other words, the gradient in these regions is nearly zero and the parameters change very slowly yielding to very slow learning (*gradient vanishing* problem).

b) Softmax

Softmax takes as input a vector of arbitrary real-valued scores (input vector *z*, of *K* real numbers) and *squashes* it into a vector of normalized values between zero and one that **sum to one.** In other words, it normalizes the vector *z* into a probability distribution consisting of *K* probabilities proportional to the exponentials of the input numbers.

Hence the output can be viewed as a probability distribution, since it fulfills the relative requirements. Therefore, the cross-entropy loss can be applied. For a given score z_j, the Softmax function gives:

$$\sigma_j(z) = P(y = j|z) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \dots, K \text{ and } z = (z_1, \dots, z_K) \in \mathbb{R}^K \tag{2.14}$$

where K is the number of classes. We should note that the Softmax computation, for a given score z_j, depends on all the scores in z, i.e. it is not applied independently to each element z_j.

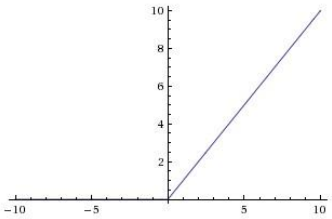
[18] Ng A. & Katanforoosh K. (2018), "Stanford lecture notes CS229: Deep Learning", http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

In other words, this function will calculate the probabilities of each target class over all possible target classes. Moreover, looking at the equation, we can see that the exponential function (numerator) combined with the normalization (denominator) results high scores in z to become much more probable than low scores.
The softmax function is often used in the final layer of a neural network-based classifier. For a binary classification, using sigmoid is same as softmax and hence we could say that softmax is a generalization of sigmoid. For multi-class classification, we use softmax with cross-entropy.

c) ReLU

The Rectified Linear Unit has become very popular in the past few years. Despite its name and appearance, it is not linear in its domain and provides the same benefits as Sigmoid but with better performance. It is often used within hidden layers of a neural network model, and in fact, it is the most popular activation function for deep neural networks.

$$R(z) = \max(0, z) \tag{2.15}$$



In other words, the activation is simply thresholded at zero (see image above). There are several pros in using the ReLUs, some of them are the following [19]:

1. It was found to greatly accelerate (e.g. a factor of 6 is reported in Krizhevsky et al. 2012) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.
2. Sparse activation: For example, in a randomly initialized network, only about 50% of hidden units are activated (have a non-zero output).
3. Efficient computation: Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.
4. Better gradient propagation: Fewer *vanishing gradient* problems compared to sigmoidal activation functions that saturate in both directions.

3.6. Parameter learning – Backpropagation

As we already saw, the two steps to define a neural network to learn a specific task are the following: (i) establishments of the network architecture, which defines how many layers, how many neurons, and how the neurons are connected and (ii) the estimation of the involved the parameters (that is, the weights and the biases of the nodes of the network). In this section, we will see how to adjust the parameters so that the network to learn the task under study. This task involves a parameter initialization phase and the main optimization phase.

[19] *F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class notes CS231n: Convolutional Neural Networks for Visual Recognition" (2020), http://cs231n.stanford.edu/.*

3.6.1. Parameter Initialization

Before we start training the neural network, we must select an initial value for each of the parameters. We do not use the value zero as the initial value because the output of the first layer will always be the same and this will cause problems later on when we try to update these parameters (i.e., the gradients will all be the same). Usually, we randomly initialize the parameters to small values (e.g., normally distributed around zero; $\mathcal{N}(0; 0.1)$).
In practice, it turns out there is something even better than random initialization. It is called **Xavier/He initialization** and initializes the weights:

$$w^{[\ell]} \sim \mathcal{N}\left(0, \sqrt{\frac{2}{n^{[\ell]} + n^{[\ell-1]}}}\right) \tag{2.16}$$

where $n^{[\ell]}$ is the number of neurons in layer ℓ. This acts as a mini-normalization technique. For a single layer, if the variance of the input to the layer is $\sigma^{(\text{in})}$ and the variance of the output (i.e., activations) of the layer $\sigma^{(\text{out})}$, Xavier/He initialization encourages $\sigma^{(\text{in})}$ to be similar to $\sigma^{(\text{out})}$ [20]. Once the parameters have been initialized, we can begin training the neural network with gradient descent.

3.6.2. Optimization

Optimization is the process of finding the set of parameters $\theta = \{W, b\}$ that minimize the loss function. A popular general optimization algorithm that, among others, have been used for the training of neural networks, is that of gradient descent (GD), which is briefly described below.

- ***Gradient descent (GD):***

The gradient descent approach is a first order optimization algorithm which is used for finding the local minima of an objective function. *Algorithm I* explains the concept of gradient descent:

Algorithm I. Gradient descent

Inputs: loss function $\mathcal{L}$, learning rate a, dataset X, y and the model $h(\theta, x)$

Outputs: Optimum θ which minimizes $\mathcal{L}$

REPEAT until converge:

$$\hat{y}_i = h(\theta, x_i); \qquad (i = 1,\ldots,N)$$

$$\theta := \theta - a \cdot \frac{1}{N}\sum_{i=1}^{N} \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial \theta}$$

End

However, gradient descent requires a long training time, especially in the case where the size of the data is very high. An alternative of GD that deals with this problem is the stochastic gradient descent scheme.

---

[20] Ng A. & Katanforoosh K. (2018), "Stanford lecture notes CS229: Deep Learning", http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

- ***Stochastic Gradient Descent (SGD):***

The SGD approach is a well-established technique from the 50s, which nowadays is used for training Deep Neural Networks (DNN) (Bottou, 2012). *Algorithm II* explains SGD in detail:

Algorithm II. Stochastic Gradient Descent (SGD)

Inputs: loss function $\mathcal{L}$, learning rate a, dataset $X = \{(x_i, y_i),\ i = 1, \ldots, N\}$, and the model $h(\theta, x)$

Outputs: Optimum θ which minimizes $\mathcal{L}$

REPEAT until converge:

For **each example**, select randomly a sample;

Compute $\hat{y}_i = h(\theta, x_i)$;

$$\theta := \theta - a \cdot \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial \theta}$$

End

Stochastic gradient descent attempts to *approximate* the gradient from (full) gradient descent.

In practice, a compromise between GD and SGD is used, giving rise to the **mini-batch gradient descent**. In the mini-batch gradient descent case, the cost function J_{mb} is defined as follows[21]:

$$J_{mb} = \frac{1}{\mathrm{B}} \sum_{i=1}^{\mathrm{B}} \mathcal{L}^{(i)} \tag{2.17}$$

where B is the number of examples in the mini batch.

3.6.3. Back-propagation algorithm

Deep neural networks (DNNs) are trained with the popular backpropagation (BP) algorithm (Rumelhart et al., 1998). Actually, this is a GD algorithm, which can be viewed under either the classical GD or the SGD framework. The pseudocode of the basic backpropagation is given in *Algorithm III*.

In the case of MLPs, we can easily represent NN models using computation graphs which are directive acyclic graphs. We can use the **chain-rule** to efficiently calculate the gradient from the top to the bottom layers with BP as shown in *Algorithm III* for a single path network. For example (Alom et al., 2018):

$$y = f(x) = g\left(W^{[L]} \ldots g\left(W^{[2]} g\left(W^{[1]} x + b^{[1]}\right) + b^{[2]}\right) \ldots + b^{[L]}\right) \tag{2.18}$$

This is the composite function for L layers of a network. In case of $L = 2$, then the function can be written as

$$y = f(x) = f\left(g(x)\right) \tag{2.19}$$

According to the chain rule, the derivative of this function can be written as

$$\frac{\partial y}{\partial x} = \frac{\partial f(x)}{\partial x} = f'\left(g(x)\right) \cdot g'(x) \tag{2.20}$$

[21] Ng A. & Katanforoosh K. (2018), "Stanford lecture notes CS229: Deep Learning", http://cs229.stanford.edu/summer2020/cs229-notes-deep_learning.pdf

Algorithm III. Backpropagation (*SGD version*)

Input: A network with ℓ layers, the activation function $g(z^{[\ell]})$, the outputs of hidden layer $\alpha^{[\ell]} = g\left(W^{[\ell]}a^{[\ell-1]} + b^{[\ell]}\right)$ and the network output $\hat{y} = \alpha^{[\ell]}$

Compute the gradient: $\delta \leftarrow \frac{\partial \mathcal{L}(y_i, \hat{y}_i)}{\partial y}$

For $i \leftarrow \ell$ down to 1 **do**

Calculate gradient for present layer:

$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial W^{[\ell]}} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial W^{[\ell]}} = \delta \frac{\partial a^{[\ell]}}{\partial W^{[\ell]}}$$

$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial b^{[\ell]}} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial b^{[\ell]}} = \delta \frac{\partial a^{[\ell]}}{\partial b^{[\ell]}}$$

Apply gradient descent using $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial W^{[\ell]}}$ and $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial b^{[\ell]}}$

Update the parameters:

$$W^{[\ell]} \coloneqq W^{[\ell]} - a \frac{\partial \mathcal{L}(y,\hat{y})}{\partial W^{[\ell]}}$$

$$b^{[\ell]} \coloneqq b^{[\ell]} - a \frac{\partial \mathcal{L}(y,\hat{y})}{\partial b^{[\ell]}}$$

Back-propagate gradient to the lower layer:

$$\delta \leftarrow \frac{\partial \mathcal{L}(y,\hat{y})}{\partial a^{[\ell]}} \frac{\partial a^{[\ell]}}{\partial a^{[\ell-1]}} = \delta \frac{\partial a^{[\ell]}}{\partial a^{[\ell-1]}}$$

End

Actually, the name backpropagation indicates the way the updating of the parameters takes place. Thus, the gradient terms corresponding to the nodes of the output layer are computed first, then the gradient terms corresponding to the nodes of the previous layer are computed and so on, until the first layer parameters is reached.

3.6.4. Learning rate (a)

The learning rate is an important component for training DNN (as explained in *Algorithm I* and *II*). The learning rate is the step size considered during training which controls the training process. The performance of BP is very sensitive to the choice of the learning rate. For example: if a larger value for a is chosen, the network may diverge or oscillate, instead of converging. On the other hand, if a smaller value for a is chosen, it will take more time for the network to converge. In addition, it may easily get stuck in local minima. The typical solution for this problem is to reduce the learning rate during training (Bottou, 2012).

3.7. Cross-validation

Cross-validation is mainly used as a more sophisticated technique for hyperparameter tuning (it can be used also for a more accurate error rate estimation), especially in cases where the size of the training data (and therefore also the validation data) might be small. The idea is that instead of splitting arbitrarily once the whole data set we have at our disposal to a training

and a validation set (e.g. arbitrarily picking the first e.g. 1000 datapoints to be the validation set and rest training set), we can get a better and less noisy estimate of how well a certain value of a hyperparameter *k* works by iterating over different validation sets and averaging the performance across these. For example, in 5-fold cross-validation, we would split the training data into 5 equal folds, use 4 of them for training, and 1 for validation. We would then iterate over which fold is the validation fold, evaluate the performance, and finally average the performance across the different folds[22].

3.8. Convolutional Neural Networks (CNNs / ConvNets)

The Convolutional Neural Network structure is a special neural network architecture that was first proposed by Fukushima in 1988. It was not widely used however due to limits of hardware for training the network. In the 1990s, LeCun et al. applied a gradient-based learning algorithm to CNNs and obtained successful results for the handwritten digit classification problem (LeCun et al., 1998). After that, researchers further improved CNNs and reported state-of-the-art results in many recognition tasks. CNNs have several advantages over DNNs, including being more similar to the human visual processing system, being highly optimized in structure for processing 2D and 3D images, and being effective at learning and extracting abstractions of 2D features. Moreover, being composed of sparse connections with tied weights, CNNs involve significantly fewer parameters than a fully connected (FC) network of similar size (Alom et al., 2018).

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product, which (optionally) passes through a non-linearity. The whole network still expresses a single differentiable *score function*: from the *raw image pixels* on one end to *class scores* at the other. And they still have a *loss function* (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks for learning regular Neural Networks still apply.

The difference is that ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network [23].

Images can be represented as a matrix with number of elements equal to the number of pixels. Color images are digitally represented as a volume (e.g., three-channels or three matrices stacked on each other for an RGB image). Hence, in general, an image is of size (x, y, c), where x, y are its spatial and c its spectral dimensions. To input such an image in a neural network, it must be flattened into a single vector containing $x * y * c$ elements.

Convolutional Neural Networks take advantage of the fact that the input consists of the raw images, of size (x, y, c), and therefore they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth. The neurons in a CNN layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would be one-dimensional, because by the end of the

[22] *F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class notes CS231n: Convolutional Neural Networks for Visual Recognition" (2020), http://cs231n.stanford.edu/.*

[23] *F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class notes CS231n: Convolutional Neural Networks for Visual Recognition" (2020), http://cs231n.stanford.edu/.*

ConvNet architecture, the full image is mapped into a single vector of class scores, arranged along the depth dimension.

3.8.1. Layers used to build ConvNets

A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. Three main types of layers are used to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). These layers are stacked together to form a full ConvNet architecture.

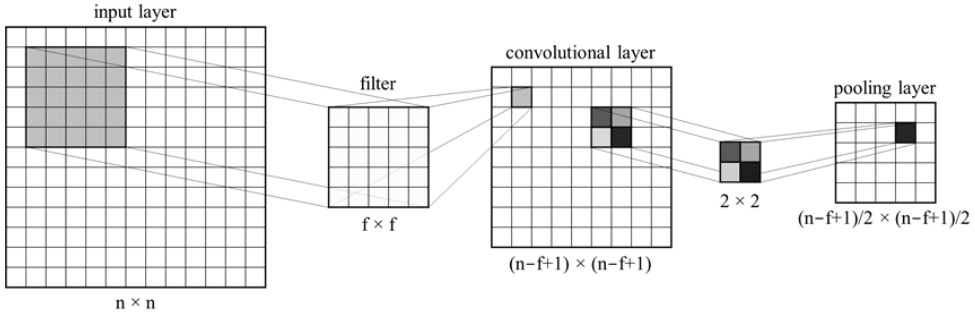


Figure 7: The two major processes in a Convolutional Neural Net: convolution & max pooling.

- **Convolutional Layer**

The Conv layer is the core building block of a Convolutional Network that does most of the computational heavy lifting. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full spectral depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because RGB images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we produce a 2-dimensional *activation map* that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them produce a separate 2-dimensional activation map. We stack these activation maps along the depth dimension and produce the output volume. Some important issues of the CONV layer are now presented in the next paragraphs.

***Local Connectivity**:* When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, we connect each neuron to only a *local region* of the input volume. The spatial extent of this connectivity is a *hyperparameter* called the **receptive field** of the neuron (equivalently this is the **filter size**). The extent of the connectivity along the depth axis (spectral axis) is always equal to the depth of the input volume. It is important to emphasize again this asymmetry in how we treat the spatial dimensions (width and height) and the depth dimension: The connections are local in space (along width and height), but always full along the entire depth of the input volume.
For example, suppose that the input volume has size [32x32x3]. If the receptive field (or the filter size) is 5x5, then each neuron in the Conv Layer will have weights to a [5x5x3] region in

the input volume, for a total of 5*5*3 = 75 weights (and a bias parameter). The extent of the connectivity along the depth axis must be 3, since this is the depth of the input volume.

Spatial arrangement: We saw the connectivity of each neuron in the Conv Layer to the input volume, but we haven't yet seen how many neurons there are in the output volume or how they are arranged. Three *hyperparameters* control the size of the output volume: the **depth**, **stride** and **zero-padding**.

1. First, the **depth** of the output volume is a hyperparameter: it corresponds to the **number of filters** we would like to use, each learning to identify different aspects in the input. For example, if the first Convolutional Layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edges, or blobs of color. We refer to a set of neurons that are all looking at the same region of the input as a **depth column**.
2. Second, we must specify the **stride** with which we slide the filter. When the stride is 1 then we move the filters one pixel at a time (that is, from the current pixel we move to its neighboring one). When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as we slide them around. This will produce smaller output volumes spatially.
3. Sometimes it is convenient to pad the input volume with zeros around the border. The size of this **zero-padding** is a hyperparameter. The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes (most commonly we use it to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

We can compute the spatial size of the output volume as a function of the input volume size (W), the receptive field size of the Conv Layer neurons (i.e., the filter size, F), the stride with which they are applied (S), and the amount of zero padding used (P) on the border. The number of neurons that "fit" is given by $(W - F + 2P)/S + 1$. For example, for a 7x7 input and a 3x3 filter with stride 1 and zero padding, we would get a 5x5 output. With stride 2 we would get a 3x3 output. In general, setting zero padding to be $P = (F - 1)/2$ when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially.

Constraints on strides: The spatial arrangement of hyperparameters have mutual constraints. For example, when the input has size $W = 10$, no zero-padding is used $P = 0$, and the filter size is $F = 3$, then it would be impossible to use stride $S = 2$, since $(W - F + 2P)/S + 1 = (10 - 3 + 0)/2 + 1 = 4.5$, i.e. a fractional value, indicating that the neurons don't "fit" neatly and symmetrically across the input. Therefore, this setting of the hyperparameters is considered to be invalid, and a ConvNet library could e.g. throw an exception or zero pad the rest to make it fit, or crop the input to make it fit. Sizing the ConvNets appropriately so that all the dimensions "work out" can be a real headache, which the use of zero-padding around the border might significantly alleviate.

Parameter Sharing: Parameter sharing scheme is used in Convolutional Layers to control the number of parameters. It turns out that we can dramatically reduce the number of parameters by making the following reasonable assumption: if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2). In other words, denoting a single 2-dimensional slice of depth as a depth slice, we are going to constrain the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the first Conv Layer in our example would now have only one unique set of weights for each depth slice. Alternatively, all neurons in each depth slice will now be using the same parameters. In practice during backpropagation, every neuron in the

volume will compute the gradient for its weights, but these gradients will be added up across each depth slice and only update a single set of weights per slice.
So if all neurons in a single depth slice are using the same weight vector, then the forward pass of the CONV layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume (Hence the name: Convolutional Layer). This is why it is common to refer to the sets of weights as a filter (or a kernel), that is convolved with the input.

Summary for the Conv Layer

Accepts a volume of size $W_1 \times H_1 \times D_1$

Requires four hyperparameters:
- Number of filters K,
- their spatial extent F,
- the stride S,
- the amount of zero padding P.

Produces a volume of size $W_2 \times H_2 \times D_2$ where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
- $D_2 = K$

With *parameter sharing*, it introduces $F \cdot F \cdot D1$ weights per filter, for a total of $(F \cdot F \cdot D1) \cdot K$ weights and K biases.

In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

- **Pooling Layer**

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation (other operations are also used; see below). The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 *downsamples* every depth slice in the input by 2 along both width and height, *discarding 75% of the activations*. Every MAX operation would in this case determining max over 4 numbers (small 2x2 region in some depth slice). The depth dimension remains unchanged.

Summary for the pooling Layer

Accepts a volume of size $W_1 \times H_1 \times D_1$.

Requires two hyperparameters:
- their spatial extent F,
- the stride S.

Produces a volume of size $W_2 \times H_2 \times D_2$ where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$
- $D_2 = D_1$.

Introduces zero parameters since it computes a fixed function of the input.

For pooling layers, it is not common to pad the input using zero-padding.

General pooling: In addition to max pooling, the pooling units can also perform other functions, such as *average pooling* or even *L2-norm pooling*. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been shown to work better in practice.

- **Fully-connected layer**

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.
It is worth noting that the only difference between FC and CONV layers is that the neurons in the CONV layer are connected only to a local region in the input, and that many of the neurons in a CONV volume share parameters. However, the neurons in both layers still compute dot products, so their functional form is identical.

3.8.2. Layer Sizing Patterns

The common rules of thumb for sizing the architectures are the following:
The **input layer** (that contains the image) should be divisible by 2, many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.
The **conv layers** should be using small filters (e.g. 3x3 or at most 5x5), using a stride of $S = 1$, and crucially, padding the input volume with zeros in such way that the conv layer does not alter the spatial dimensions of the input. That is, when $F = 3$, then using $P = 1$ will retain the original size of the input. When $F = 5$, $P = 2$. For a general F, it can be seen that $P = (F - 1)/2$ preserves the input size. It is not common to use larger filter sizes (such as 7x7 or so), except the very first conv layer that is looking at the input image.
The **pool layers** are in charge of downsampling the spatial dimensions of the input. *The most common setting is to use max-pooling with 2x2 receptive fields (i.e. $F = 2$), and with a stride of 2 (i.e. $S = 2$).* This discards exactly 75% of the activations in an input volume (due to downsampling by 2 in both width and height). Another slightly less common setting is to use 3x3 receptive fields with a stride of 2 (this is also called overlapping pooling). It is very uncommon to see receptive field sizes for max pooling that are larger than 3 because the pooling is then too lossy and aggressive. This usually leads to worse performance.
The use of stride = 1: Smaller strides work better in practice. Additionally, stride 1 allows us to leave all spatial down-sampling to the POOL layers, with the CONV layers only transforming the input volume depth-wise. Stride > 1 may apply on data with high resolution.
The use of padding: In addition to the benefit of keeping the spatial sizes constant after CONV, doing this actually improves performance. If the CONV layers were to not zero-pad the inputs and only perform valid convolutions, then the size of the volumes would reduce by a small amount after each CONV, and the information at the borders would be "washed away" too quickly[24].

[24] *F.-F. Li, A. Karpathy, and J. Johnson, "Stanford CS class notes CS231n: Convolutional Neural Networks for Visual Recognition" (2020), http://cs231n.stanford.edu/.*

3.9. The k-Nearest Neighbors (kNN) and the Support Vector Machine (SVM) classifiers

To validate our methodology and the accuracy we obtained through the ConvNet classifier implementation, we had to compare it with other popular and competitive classifiers. For this, we used k-Nearest Neighbors (kNN) and Support Vector Machines (SVM). Here, we briefly describe the theory behind those two methods.

- **k-Nearest Neighbors**

The algorithm for the so-called *nearest neighbor rule* is summarized as follows (Theodoridis & Koutroumbas, 2006). Given an unknown feature vector $\boldsymbol{x}$ and a distance measure, then:

- Out of the *N* training vectors, identify the k nearest neighbors, *irrespective* of class label. k is chosen to be odd for a two-class problem, and in general not to be a multiple of the number of classes *M*.
- Out of these *k* samples, identify the number of vectors, k_i, that belong to class ω_i, $i = 1, 2, \ldots, M$. Obviously, $\sum_i k_i = k$.
- Assign $\boldsymbol{x}$ to the class ω_i with the maximum number k_i of samples.

Remarks

- Various distance measures can be used, including the Euclidean and Mahalanobis distance. The simplest version of the algorithm is for $k = 1$, known as the *Nearest Neighbor* (*NN*) rule. In other words, a feature vector $\boldsymbol{x}$ is assigned to the class of its nearest neighbor. Provided that the number of training samples is large enough, this simple rule exhibits good performance.
- A serious drawback associated with (*k*)NN techniques is the complexity in search of the nearest neighbor(s) among the *N* available training samples. The problem becomes particularly severe in high-dimensional feature spaces follows (Theodoridis & Koutroumbas, 2006).

- **Support Vector Machines**

This algorithm searches for a hyperplane that best separate two classes that are linearly separable. Due to some interesting properties of this method, we can also extend its application to non-linear classification problems. It can be also extended in multi-class classification problems. A very sensible choice for the hyperplane classifier would be the one that leaves the maximum margin from both classes. A hyperplane is defined by its direction (determined by $\boldsymbol{w}$) and its exact position in space (determined by w_0). *Our goal is to search for the direction that gives the maximum possible margin.* For each $\boldsymbol{x}_i$, we denote the corresponding class indicator by y_i (+1 for ω_1, -1 for ω_2).

Initially, we can define an optimization problem as follows: Compute the parameters $\boldsymbol{w}$, w_0 of the hyperplane so that to (Theodoridis & Koutroumbas, 2006):

- minimize $J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2$
- subject to $y_i(\boldsymbol{w}^T\boldsymbol{x}_i + w_0) \geq 1, \ i = 1, 2, \ldots, N$

Obviously, minimizing the norm makes the margin maximum. This is a nonlinear (quadratic) optimization task subject to a set of linear inequality constraints.

The vector parameter $\boldsymbol{w}$ of the optimal solution is a linear combination of $N_s \leq N$ feature vectors which are associated with $\lambda_i \neq 0$, where λ_i are the *Lagrange multipliers*. That is, $\boldsymbol{w} = \sum_{\boldsymbol{i=1}}^{N_s} \lambda_i y_i \boldsymbol{x}_i$. These are known as *support vectors* and the optimum hyperplane classifier as a *support vector machine* (SVM). The margin is finally defined as the distance between the pair of parallel hyperplanes described by: $\boldsymbol{w}^T \boldsymbol{x} + w_0 = \pm 1$.

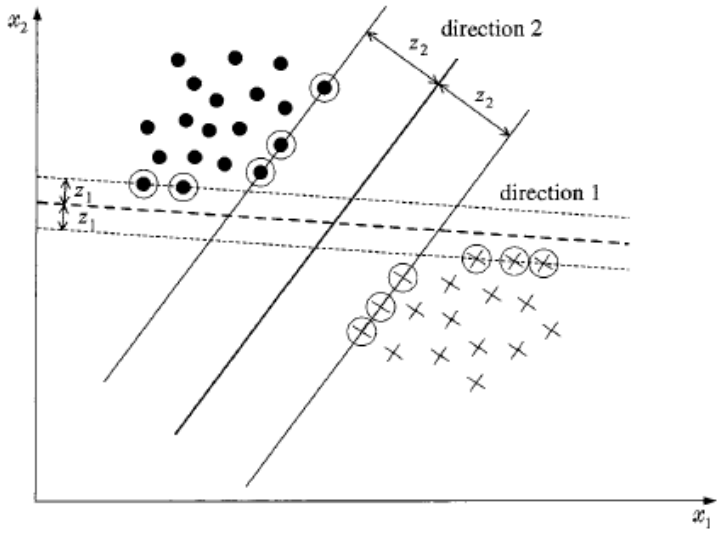


Figure 8: An example of a linearly separable two-class problem with two possible linear classifiers. The margin for direction 2 is larger than the margin for direction 1 (Theodoridis & Koutroumbas, Pattern Recognition, 2006).

After a bit of algebra our optimization task can be summarized as (Theodoridis & Koutroumbas, 2006):

$\max_{\lambda} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i{}^T \boldsymbol{x}_j \right)$, subject to $\sum_{i=1}^{N} \lambda_i y_i = 0$, $\boldsymbol{\lambda} \geq 0$, where $\boldsymbol{\lambda}$ the vector of the Lagrange multipliers.

Remarks

- The training vectors enter into the game in pairs, in the form of inner products. This is most interesting. *The cost function does not depend explicitly on the dimensionality of the input space.* This property allows for efficient generalizations in the case of nonlinearly separable classes.
- Although the resulting optimal hyperplane is unique, there is no guarantee about the uniqueness of the associated Lagrange multipliers λ_i (Theodoridis & Koutroumbas, 2006).

Chapter 4: Preprocessing steps & ConvNet Model Implementation

All the preprocessing steps, from the first familiarization with Swarm data to the generation of the final product, i.e. the wavelet spectrum images, were implemented in MATLAB, while the input dataset (i.e., the labeled images) and the CNN model were implemented using Python and its ML/DL framework TensorFlow[25].

4.1. Preprocessing

The preprocessing includes:

- Familiarization with Swarm satellite measurements:
  - Plots of magnetic field measurements using data from VFM and ASM instruments of the three Swarm satellites;
  - Comparison between "quiet" time periods and periods when a magnetospheric storm had occurred (e.g., March 2015);
  - Use of CHAOS-6 geomagnetic field model [26] on Swarm time series, in order to isolate each magnetic field contribution, i.e., magnetic field source from Earth's core, crust, magnetosphere;
  - Comparison with various indices of magnetic activity such as Dst and Kp [27].

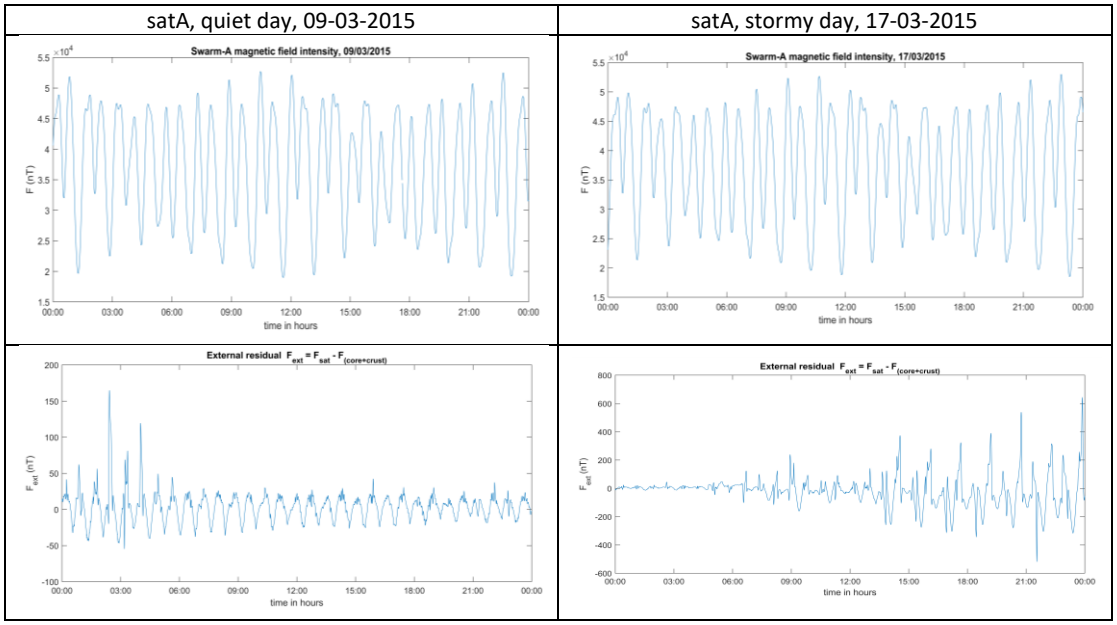


Figure 9: Example plots of Swarm-Alpha magnetic field measurements (total magnitude calculated from VFM instrument) per day (first row), and subtracting CHAOS-6 model internal sources (core & crust contributions), for two different days of March 2015, a quiet day (left column) and a stormy day (right column).

[25] https://www.tensorflow.org/
[26] http://www.spacecenter.dk/files/magnetic-models/CHAOS-6/
[27] https://omniweb.gsfc.nasa.gov/

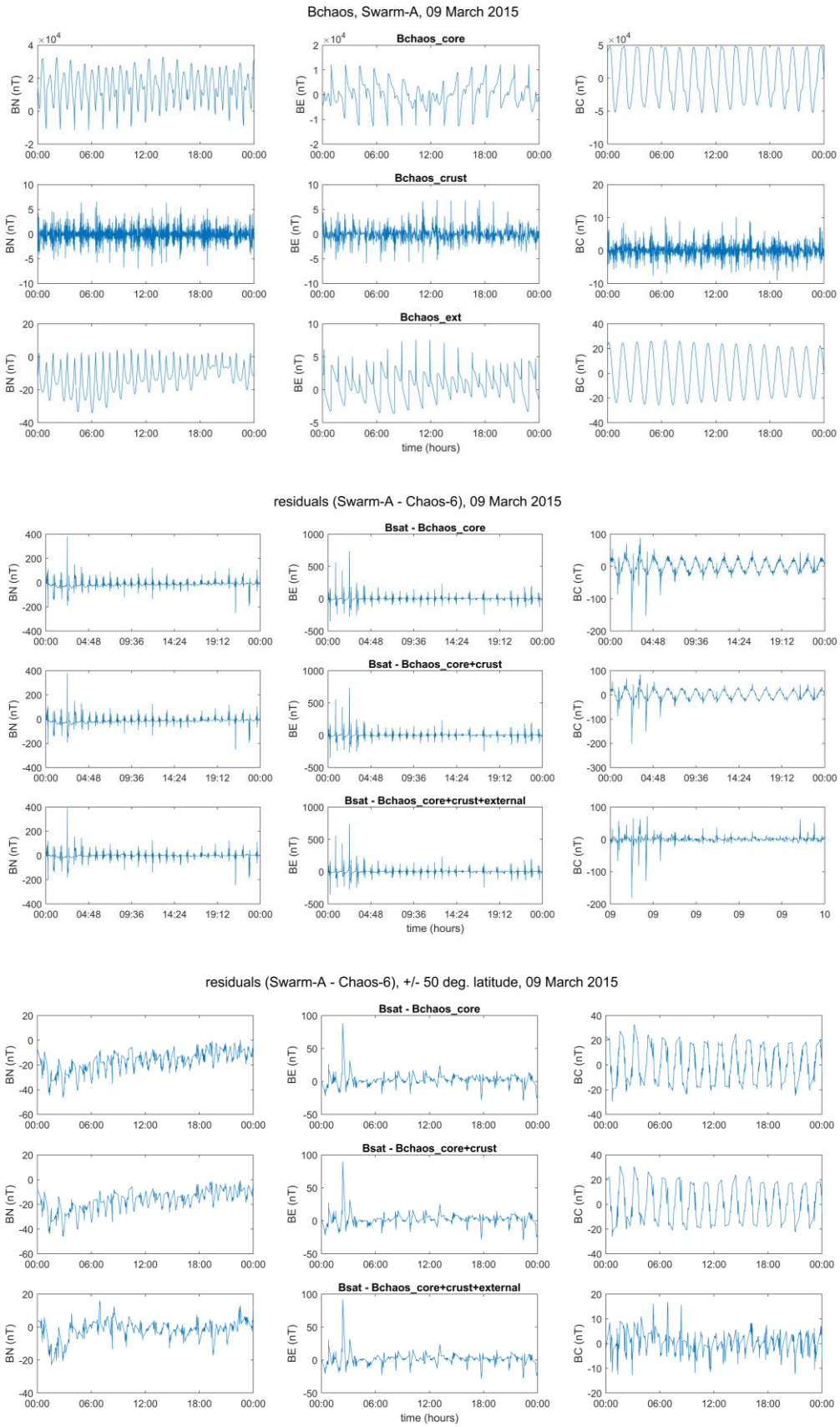


Figure 10: Magnetic field vector components, NEC frame. Core, crust and external sources derived from CHAOS-6 model (up), subtracted from Swarm-A total magnetic field vector components, NEC frame (middle), and "zoomed-in" ±50 deg. latitude (down), for the 9th of March 2015 ("quiet" day).

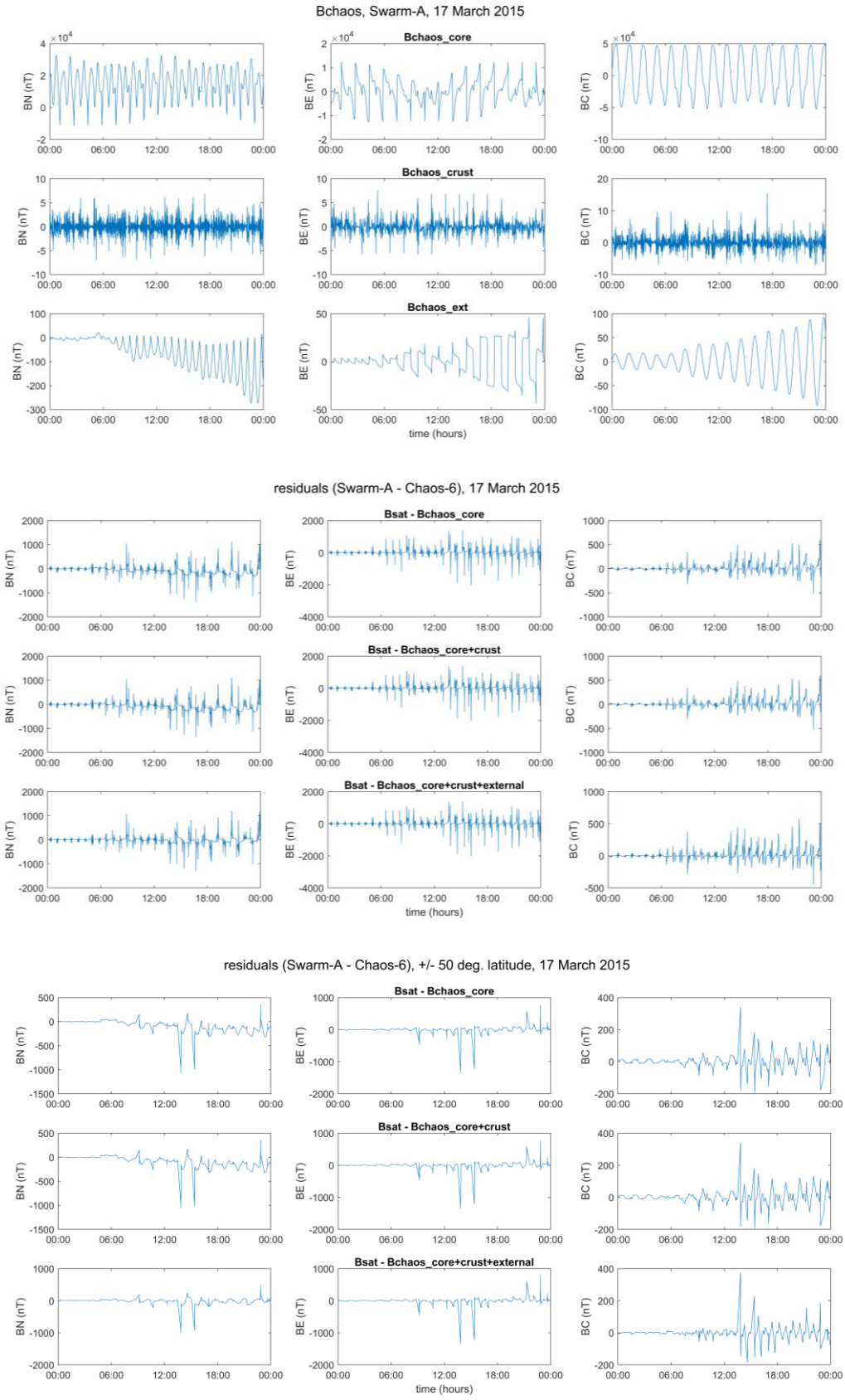


Figure 11: Magnetic field vector components, NEC frame. Core, crust and external sources derived from CHAOS-6 model (up), subtracted from Swarm-A total magnetic field vector components, NEC frame (middle), and "zoomed-in" ±50 deg. latitude (down), for the 17th of March 2015 ("stormy" day).

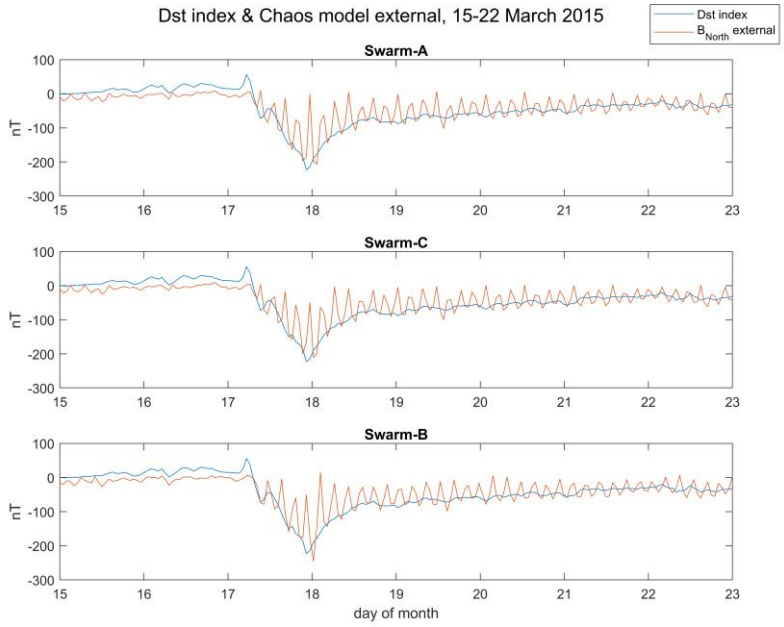
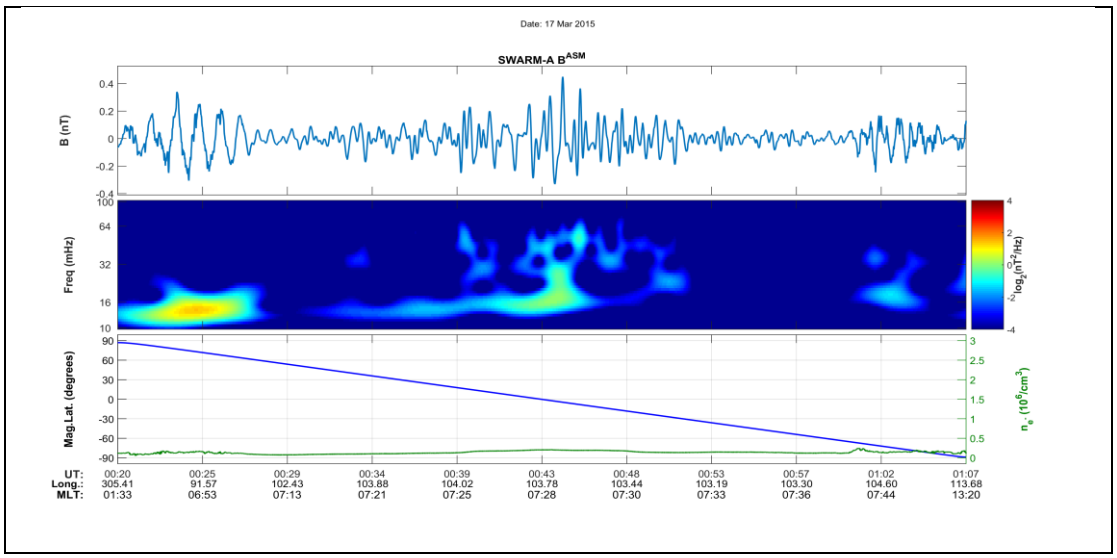


Figure 12: Example plots of the Earth's external magnetic field source using the CHAOS-6 model in combination with Swarm data, and comparison with the Dst index, for the period 15-22 March 2015, where a magnetic storm has occurred (started on 17 of March).

- Familiarization with various methods on time series analysis, such as filtering, Fourier and Wavelet transform;
- Plots of magnetic field time series per day for each satellite and then cut per satellite track (i.e., -90° to +90° latitude), in order to "zoom in" the occurring events;
- Segmented into mid-latitudinal tracks (i.e., -45° to +45° latitude), in order to exclude the influence of polar FACs that might affect the measurements;
- High pass (HP) filtering, using Butterworth filter, with a cutoff frequency of 16 mHz, in order to isolate the Pc3 ULF pulsations;
- Wavelet analysis on the produced time series, which gives us the final 3D images.



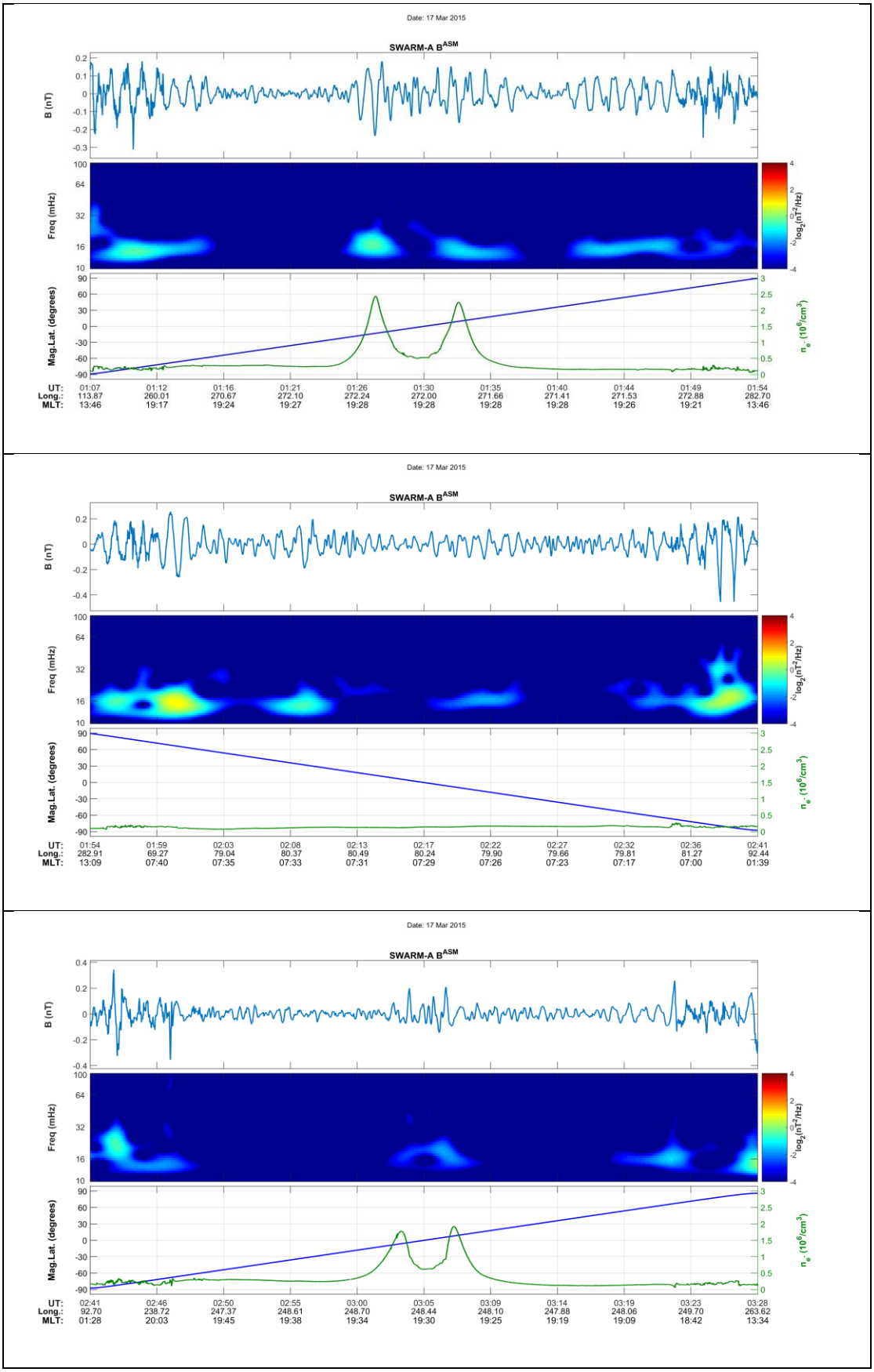


Figure 13: Example plots of the filtered magnitude (derived from Swarm ASM instrument) per satellite track, its wavelet transform, and the Swarm electron density measurement along with satellite latitude, wrt time.

4.2. CNN Model Implementation

The CNN model implementation includes the following steps:

- The creation of the **input dataset (*X*, *y*)**, where *X* the wavelet spectrum images and *y* their labels. To do so, we split the wavelet images in four different categories and give each category a different label, as follows:
    - *label y* = 0: ***"Pc3 ULF wave events"*** → *wavelet spectrum image x represents a Pc3 ULF wave event* (16-100 mHz)*;*
    - *label y* = 1: ***"non-ULF signals"*** → *wavelet spectrum image x doesn't show any significant wave activity (mainly background noise);*
    - *label y* = 2: ***"False Positives"* (*FP*)** → *wavelet spectrum image x shows anomalous signals due to e.g. spikes or discontinuities in the measurements;*
    - *label y* = 3: ***"Plasma Instabilities" (PI)*** → *wavelet spectrum image x represents plasma instabilities, i.e., events that are influenced or caused by Equatorial Spread F (ESF) irregularities (Stolle et al., 2006; Park et al., 2013) or in general by other, unclassified anomalies in the ionosphere, in near-equatorial, night-side areas.*

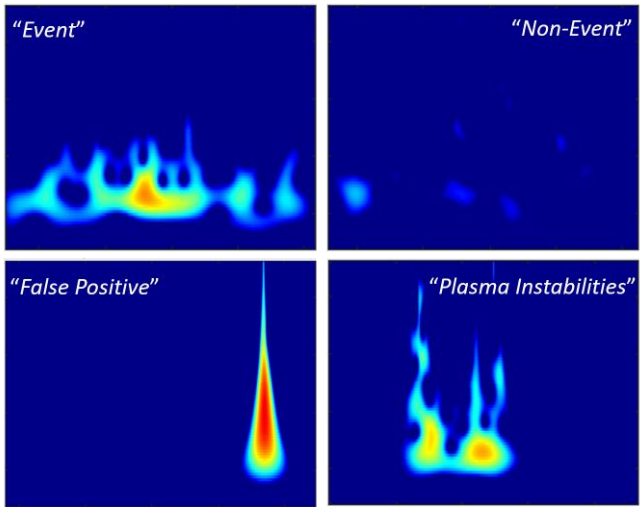


Figure 14: Examples of the four categories used in our classification problem.

- The construction of the CNN building blocks. An abstract representation of the CNN architecture is the following:

$$CONV1 \rightarrow RELU \rightarrow POOL1 \rightarrow CONV2 \rightarrow RELU \rightarrow POOL2 \rightarrow$$

$$FLATTEN \rightarrow FULLYCONNECTED \rightarrow SOFTMAX.$$

  In the first stage of the CNN model, each image from the input data *X*, (of size (n_H x n_W x n_C x m), where n_H and n_W the image height and width respectively (spatial dimensions), n_C the number of channels (spectral dimensions), and m the number of training examples) passes through convolution operations with 8 filters of size f = 4, stride *s* = 1, and zero-padding with *p* = (f−1)/2 ("*same*" *padding*) in order to give the *same* spatial size output. The result then passes through a non-linear operation (ReLU), which doesn't change the input dimensions. The output of the non-linear activation function gets through a pooling layer, which implements a max-pooling

operation using a kernel of size 8x8, stride 8 and zero-padding, to give an output volume with smaller spatial size and same spectral size.
Next, we are moving to the second stage of the CNN model where we have the same sequence of layers with slight different parameters: convolution operations with 16 filters of size 2x2, stride 1 and "same" zero-padding, followed by a ReLU activation function, followed by a max-pooling operation with kernel size 4x4, stride 4 and "*same*" zero-padding.
The output volume then passes through a flattening process, which converts it into a 1D feature vector. The top layer is a fully connected layer (FC), with one output unit per class label (Ciresan et al., 2011), i.e. four neurons, since we have four different classes. The non-linearity in the FC layer is introduced through the Softmax activation function.

- Training the model - Backpropagation:

  We train our model (update the parameters and minimize the cost) using ***Adam optimization***, instead of the classical stochastic gradient descent procedure, with 64-sized mini-batches (64 examples per batch) and a cross-entropy loss function. The initial learning rate of the optimization is 0.009. The model is trained for 100 epochs, i.e. we update the parameters in the optimization loop 100 times.

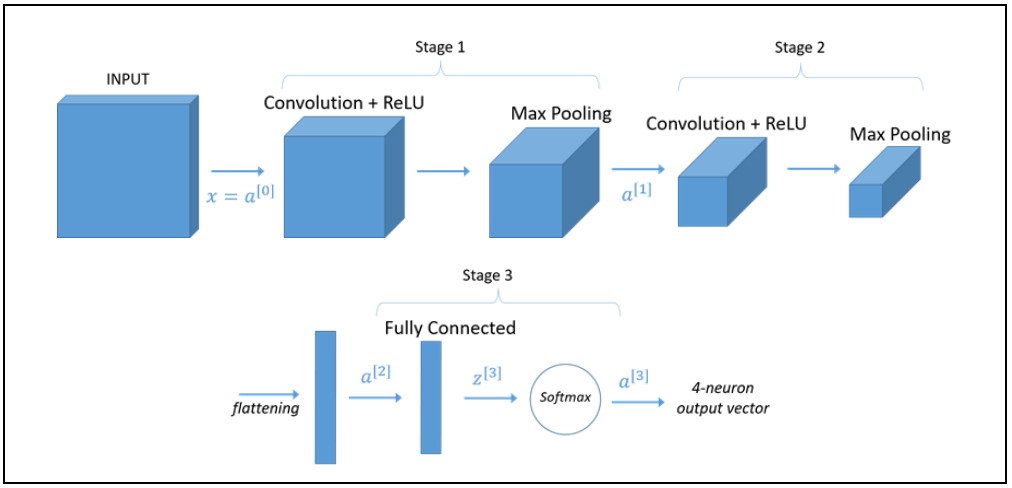


Figure 15: A simple representation of the Convolutional Neural Net architecture: Two convolutional followed by max pooling layers and one fully connected layer were used.

In the FC layer we use a regularization technique in order to avoid overfitting. Overfitting can be reduced by using "Dropout" (see Figure 15) to prevent complex co-adaptations on the training data (Hinton et al., 2012).
In general, when a large feedforward neural network is trained on a small training set, it typically performs poorly on held-out test data. This "overfitting" is found that can be greatly reduced by randomly omitting half of the feature detectors on each training case. More precisely, on each presentation of each training case, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present (Hinton et al., 2012).
We also use the Batch Normalization technique, which normalize layer inputs, allowing us to use much higher learning rates and be less careful about parameter initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. The normalization is

performed *for each training mini-batch*, before the activation function (Ioffe & Szegedy, 2015).

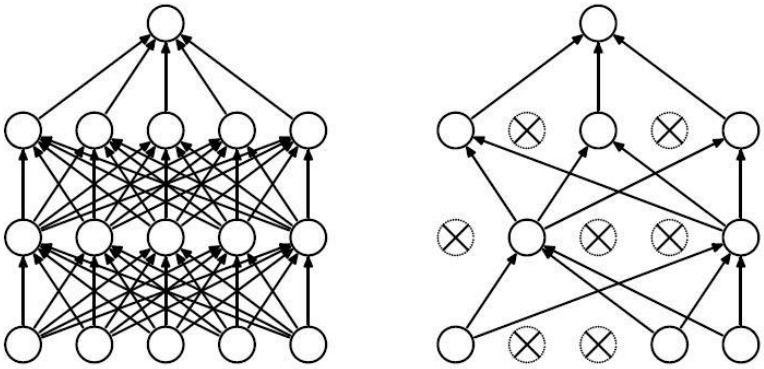
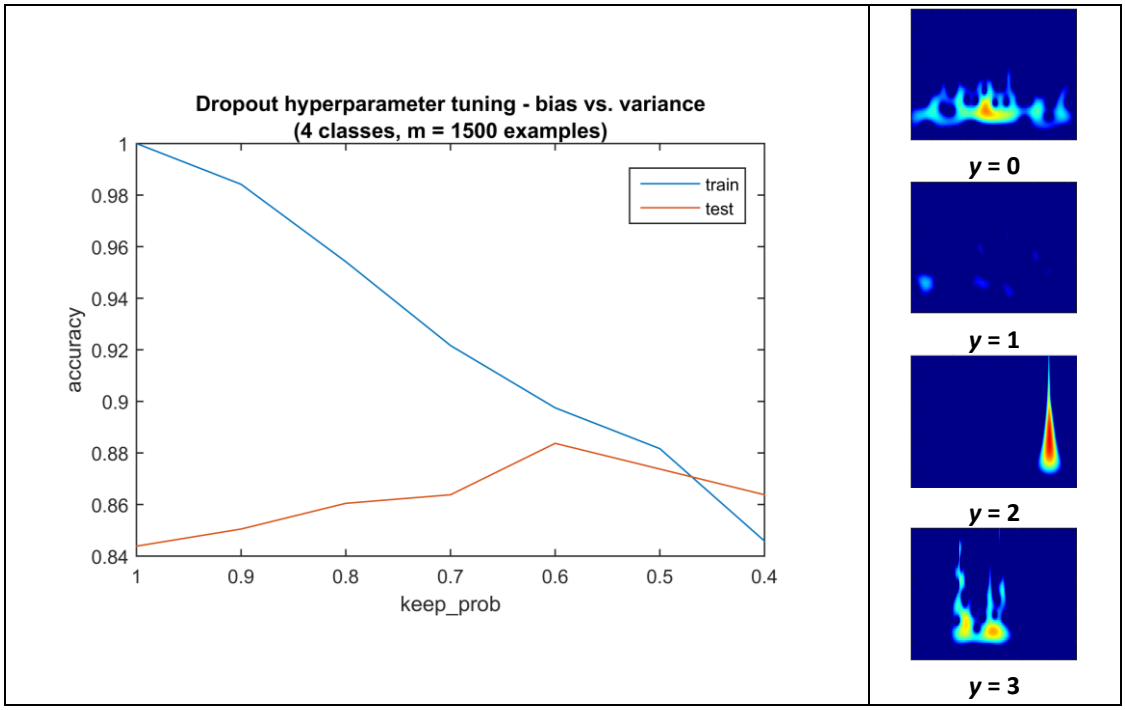
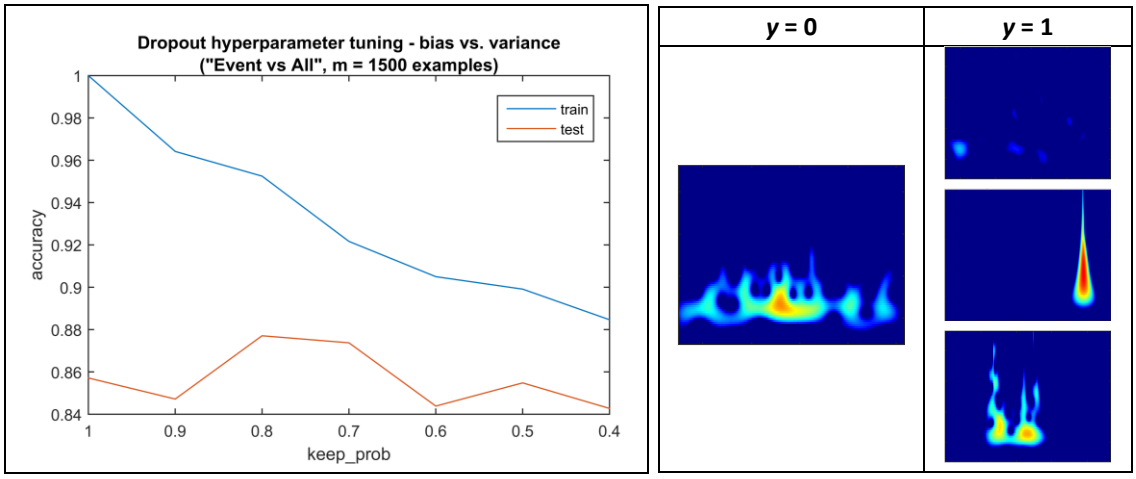


Figure 16: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped. (Srivastava et. al., JMLR 2014)

Table 5: Details of the ConvNet model architecture.

Layer		**Layer Operation & Details**		**Notation & Dimensions**	
ℓ=1	CONV	Input $x = a^{[0]}$	*Wavelet spectrum image*	$n_H^{[0]}$ x $n_W^{[0]}$ x $n_C^{[0]}$	85 x 109 x 3
		Weights $W^{[1]}$	*8 filters, 4x4 kernel, stride = 1, zero-padding, "same"*	$f^{[1]}$ x $f^{[1]}$ x $n_C^{[0]}$ x $n_C^{[1]}$	4 x 4 x 3 x 8
		activation $a^{[1]}$	ReLU($z^{[1]}$)	$g^{[1]}(W^{[1]}x)$	85 x 109 x 8
	POOL	*maxpooling*	*8x8 kernel, stride = 8, zero-padding, "same"*	$n_H^{[1]}$ x $n_W^{[1]}$ x $n_C^{[1]}$	11 x 14 x 8
ℓ=2	CONV	Weights $W^{[2]}$	*16 filters, 2x2 kernel, stride = 1, zero-padding, "same"*	$f^{[2]}$ x $f^{[2]}$ x $n_C^{[1]}$ x $n_C^{[2]}$	2 x 2 x 8 x 16
		activation $a^{[2]}$	ReLU($z^{[2]}$)	$g^{[2]}(W^{[2]}a^{[1]})$	11 x 14 x 16
	POOL	*maxpooling*	*4x4 kernel, stride = 4, zero-padding, "same"*	$n_H^{[2]}$ x $n_W^{[2]}$ x $n_C^{[2]}$	3 x 4 x 16
ℓ=3	FLATTEN	*flattening*	*convert to 1D feature vector*	$n_H^{[2]}$ x $n_W^{[2]}$ x $n_C^{[2]}$ $\rightarrow$ $\left(n_H^{[2]} \cdot n_W^{[2]} \cdot n_C^{[2]}\right)$ x 1	192 x 1
	FC	Weights $W^{[3]}$	*fully connected, dense layer*	4 x $n^{[2]}$	4 x 192
		activation $a^{[3]}$	*Classifier, Softmax* ($z^{[3]}$), *Dropout prob = 0.6*	$g^{[3]}(W^{[3]}a^{[2]})$	4 x 1
		Final output	*Label*	$n^{[3]}$ x 1	4 x 1





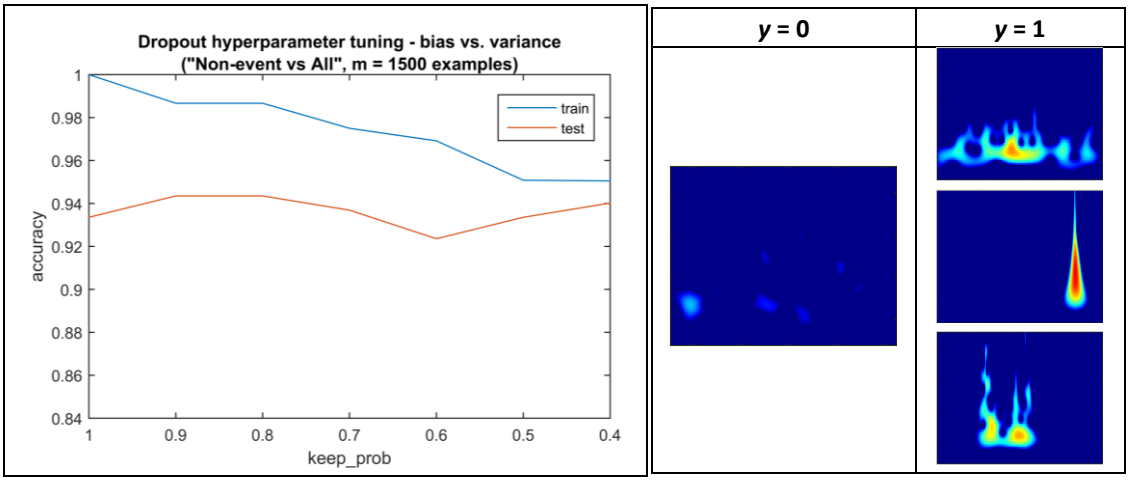


Figure 17: Dropout hyper-parameter tuning, in various cases where different types of classes were used.

4.3. Coding Steps

- Pre-processing (MATLAB)

1. Load multiple cdf files in MATLAB. Each cdf file corresponds to one day of Swarm satellite measurements.
2. Cut the loaded data in satellite orbits (~32 orbits/day) using the latitude information.
3. Discard latitudes that are out of our area of interest, i.e., out of ±45° latitude. Hence, the final magnetic field time series that we used in the next steps correspond to mid-latitudinal satellite tracks only.
4. Apply a Butterworth high pass (HP) filter on the magnetic field measurement, with cutoff frequency of 16 mHz.
5. Implement the wavelet transformation on the filtered magnetic field measurement.
6. Split the wavelet images in the four classes and create a vector containing their corresponding labels.

- Model Implementation (Python – TensorFlow)

7. Load training and test datasets (X_{train}, y_{train}, X_{test}, y_{test}).
8. Convert labels to one-hot encoding and normalize the intensity of each image pixels by dividing it with 255.
9. Initialize the parameters (weights/filters) of each layer using Xavier initializer.
10. Forward propagation, to build the following model: CONV2D → RELU → MAXPOOL → CONV2D → RELU → MAXPOOL → FLATTEN → FULLYCONNECTED, using TensorFlow's built-in functions [28].
11. Compute the Softmax entropy loss and sum the losses over all the examples to get the overall cost.
12. Back-prop: define an optimizer using TensorFlow's AdamOptimizer that minimize the cost. Initial learning rate defined at 0.009.
13. Train the model for a number of epochs using a for-loop, for one mini-batch of data each time using another for-loop.
14. Calculate the accuracies on training and test sets through the comparison of the known labels (ground truth) and the predicted labels.

- Extra notes

15. The Dropout function is used only in the fully connected layer, before the Softmax (Park and Kwak, 2017), with a probability p = 0.6. If we want to apply Dropout in the CONV layers also, it should be applied with much smaller p, after every convolution layer (before pooling).
16. The batch normalization can be applied inside the activation function, e.g., relu(batch_normalization(x)).

[28] https://docs.w3cub.com/tensorflow~python/tf/

4.4. Results

Table 6.1a: Summary information about the training of the network

Data used:	Total magnitude, extracted from Swarm Vector Field Magnetometer (VFM) measurements, NEC frame, 1s sampling rate, (data extracted from the year 2015).
Number of total samples:	1500 samples, *manually annotated* in 4 classes
Input:	Wavelet power spectra images with their annotation (class label)
Training – Test set:	80% – 20%
Layers:	2 convolutional, 2 pooling, 1 fully connected (FC)
Parameter initializer:	Xavier Initialization
Activation functions:	ReLU, Softmax
Cost function:	Cross -entropy (Log Loss)
Optimizer:	Adam Optimization
Extra:	Batch Normalization, Dropout Regularization

Table 6.1b: Summary information about the layers of the network

Layers	**Details**
Conv1	8 filters, $f = 4$, $s = 1$
Pool1	$f = 8$, $s = 8$
Conv2	16 filters, $f = 2$, $s = 1$
Pool2	$f = 4$, $s = 4$
FC	*Classifier*, *4-neuron output*

The final dataset consists of 1500 samples manually annotated, i.e. 1500 pairs of spectral images with their annotation (class label). The whole dataset was split in training/test sets at an 80/20% ratio, respectively. The training was performed for 100 epochs. The obtained accuracy, 89.7% on the training set and 88.5% on the test set, is presented in Table 7. The cost function and the accuracies at every training epoch are shown in the next figure.

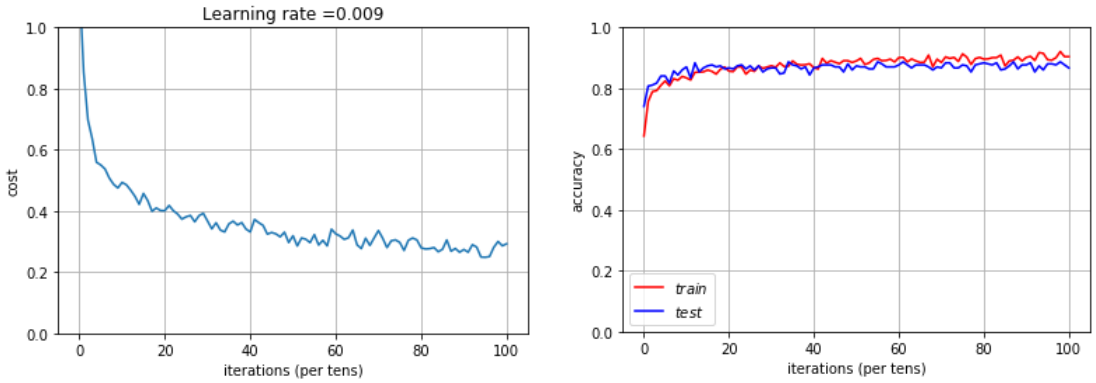


Figure 18: The cost function of the training set (left), and the accuracies of the training and test sets (right), with respect to the iterations of the optimization loop.

Table 7: The ConvNet performance after 100 epochs of training.

Training set accuracy %	89.7
Test set accuracy %	88.5

Chapter 5: Discussion on the results & further possible improvements

5.1. Summary of the method

In this study, we used a wavelet power spectrum image dataset, created with data from the Swarm satellite mission, to classify ULF wave events in the Earth's magnetosphere, with Convolutional Neural Networks. We have manually created a 4-class dataset, by using time series magnetospheric data from all the three Swarm satellites. Specifically, we used the total magnitude, extracted from the Swarm Vector Field Magnetometer (VFM), which is measuring in the NEC frame with 1 second sampling rate, for the year 2015.

Furthermore, we showed our preprocessing pipeline for the preparation of an image, before it is applied on our ConvNet. The data were first segmented into mid-latitudinal tracks, i.e., -45° to +45° latitude, and then filtered using a HP Butterworth filter with a cutoff frequency of 16 mHz. Next, we performed wavelet analysis on the produced time series. Hence, from 1-D time series features we went to 2-D image features (time & frequency).

These wavelet images were split manually in 4 different classes. That is, instead of creating only two classes, "ULF wave events" and "non-ULF signals", we defined two more classes, "False Positives" (FP) and "Plasma Instabilities" (PI). We have seen in the evaluation that the "PI" class is not so well defined and has therefore increased the classification error. A possible solution to this is to add extra information during preprocessing to help on the accurate definition and separation of the "PI" class images.

Having prepared our 4-class image dataset, we went to the development of our ML model. We used a ConvNet architecture with 2 convolutional stages, composed of convolutional and pooling layers, and one fully-connected (FC) stage giving the final output, i.e. the predicted class. To initialize the model's parameters, we used Xavier initialization. To learn the model's parameters during Backprop, we used the Adam optimization method to minimize our cost function, implemented in mini-batches of the data.

The Swarm data analysis was performed in the MATLAB environment, while the ConvNet model was implemented in Python using TensorFlow framework. In general, our developed ConvNet is well designed. The Dropout technique in the FC layer helped to improve the overall performance, reducing overfitting. But with such a small dataset and a class that has not been perfectly defined (the "PI" class), the error could not further decrease. Growing the dataset and adopting a more robust definition of the classes, the overall performance of the method can be further improved.

As a final step, we need to compare our ConvNet model with other machine learning classifiers, using exactly the same input dataset. For this, we have implemented four different versions of the popular k-Nearest Neighbors (kNN) classifier, for k = 1, 3, 5, 7, and the very competitive Support Vector Machine (SVM) classifier. The accuracies obtained with these methods compared with our ConvNet accuracy of 88.5% (see Table 8), are a good indication and proof that ConvNets are a very powerful ML method that can be definitely used for the ULF wave analysis.

Table 8: Comparison of classifiers: various versions of k-Nearest Neighbors, Support Vector Machines, and our Convolutional Neural Network. Overall accuracy on the test set.

ML Classifier	Accuracy (%)
kNN (k = 1)	54.8
kNN (k = 3)	56.5
kNN (k = 5)	57.5
kNN (k = 7)	54.5
SVM	80.7
ConvNet	88.5

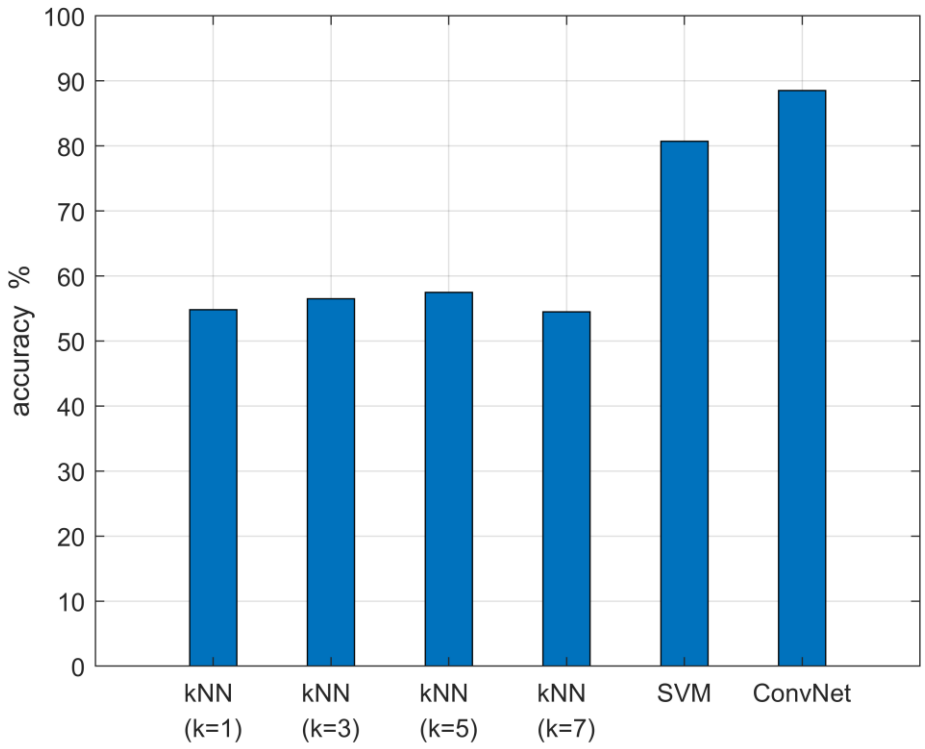


Figure 19: Bar chart showing the accuracy achieved by each one of the classifiers. Clearly our ConvNet model shows the best result.

5.2. Further notices on processing & results, Future directions

- Accuracy

For an input dataset of m = 1500 examples and a 4-classes output, the achieved accuracy on the test set is acc_{test} = 88.5% while the accuracy on the training set is acc_{train} = 89.7%, using Dropout regularization with probability p = 0.6. When the Dropout probability is larger (i.e. we keep more hidden units on the training phase), then we have higher variance and lower bias (i.e. higher accuracy on the training set and lower accuracy on the test set). For example, for p = 0.8, acc_{train} = 95.5% and acc_{test} = 86.5%. Therefore, for a relatively small input dataset,

the Dropout regularization technique seems to work, reducing overfitting. Apart from the overall accuracy, it is crucial to calculate more quantities, such as a *confusion matrix*, in order to better understand which classes are not well separated.

- Cross-Validation

Another way to achieve better performance of the estimator, is to insert a cross-validation step in the training phase. That is, split the training set in a number of smaller batches and evaluate different settings (i.e. different values for the hyperparameters of the model) on each batch. Then, use the hyperparameters that achieved the highest accuracy in a specific batch, to train the model (using the whole training set). This way, we could better tune hyperparameters such as the kernel size or the number of kernels in the CONV layer, kernel size of the POOL layer, etc., and achieve better performance on the test set. Another reason why cross-validation is crucial to be inserted in our methodology, is to decrease the statistical fluctuation in the final error estimations.

- Dimensions of input images

In order for our CNN model to work more efficiently, we should take better care of the dimensions of the input images. We should try: (1) use 1-D grayscale images that represent the "power" matrix of the wavelet transform output, instead of 3-D RGB images which might contain redundant information, slowing down the model's performance; (2) reshape the spatial dimensions to a more appropriate size for the CNN model. As already been said, the input layer (that contains the image) should be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512. Hence, instead of using images of size 85 x 109 x 3, we should try to create rectangular images of size e.g. 96 x 96 x 1.

- The "Plasma Instabilities" class

As a further development and examination of the method, we performed the following experiments: training of the network by having split the data into only 2-classes datasets, (1) "ULF Events vs. All" and (2) "Non-ULF events (i.e. background noise) vs. All" (see Figure 16), and comparing the results with the ones obtained by using our initial 4-classes dataset. The results show an indication that the class "Plasma Instabilities" (PI) seems to "damage" the model performance, unsurprisingly, because many "PI" wavelet spectrum images are very similar optically with many "ULF wave Events" spectrum images, making it difficult to distinguish between them and annotate the images with the correct class label. To solve this, we should consider inserting some useful information during preprocessing that will help us differentiate PIs from ULF Events, such as the *electron density product* from Swarm measurements, or the *nightside* and *dayside* location information of the satellite track. Equatorial Spread-F (ESF) or plasma bubbles are a post-sunset phenomenon (i.e., 18:00-06:00 MLT or even 20:00-06:00 MLT) and also a seasonal effect (i.e., Fall/Spring have more ESF events than Winter/Summer (Stolle et al., 2006)).

- Dataset

Finally, when we consider all of the possible improvements mentioned above, the most crucial thing to do is to re-train our network using a much larger dataset, exploiting all the Swarm magnetic field data, from the beginning of the mission onwards. This will give us a dataset with almost 50000 examples. Training a network with a larger dataset may help improve the obtained accuracy and the generalization performance of our deep learning model whilst the whole methodology will be much more well-established.

References

Alom, Md. Z., T. Taha, C. Yakopcic, S. Westberg, M. Hasan, B. Esesn, A. Awwal, and V. Asari (2018), The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches, Preprint.

Baker, D. N. (1990), Linear prediction filter analysis of relativistic electron properties at 6.6 RE, J. Geophys. Res. Space Phys. 95 (A9), 15133–15140.

Balasis, G., S. Aminalragia-Giamini, C. Papadimitriou, I. A. Daglis, A. Anastasiadis, and R. Haagmans, A machine learning approach for automated ULF wave recognition, J. Space Weather Space Clim., 2019, 9, A13, doi:10.1051/swsc/2019010

Balasis, G., C. Papadimitriou, I. A. Daglis, and V. Pilipenko, 2015, ULF wave power features in the topside ionosphere revealed by Swarm observations, Geophys. Res. Lett., 42, 6922–6930, doi:10.1002/2015GL065424.

Boberg, F., P. Wintoft, and H. Lundstedt (2000), Real time Kp predictions from solar wind data using neural networks. Phys. Chem. Earth Part C 25 (4), 275–280.

Bogoutdinov, Sh. R., N. V. Yagova, V. A. Pilipenko, and S. M. Agayan (2018), A technique for detection of ULF Pc3 waves and their statistical analysis, Russ. J. Earth. Sci., 18, ES6006, doi:10.2205/2018ES000646.

Bottou, L. (2012), "Stochastic gradient descent tricks." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 421-436.

Camporeale, E., S. Wing, and J. Johnson (2018), Machine Learning Techniques for Space Weather, p 454, Elsevier, Amsterdam, The Netherlands, doi: 10.1016/C2016-0-01976-9.

Ciresan D. C., U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber (2011), Flexible, High Performance Convolutional Neural Networks for Image Classification, IJCAI-11, Barcelona, Spain, AAAI Press.

Collobert, R., and J. Weston, 2008, A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning, *in: Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008)*, Helsinki, Finland, June 5-9, 2008, pp. 160–167.

Engebretson M, Zanetti L, Potemra T, Baumjohann W, Lühr H, Acuna M (1987) Simultaneous observations of Pc3-4 pulsations in the solar wind and in the Earth's magnetosphere. J Geophys Res. doi:10.1029/JA092iA09p10053.

Francia P, De Laurentis M, VellanteM, Paincatelli A (2009) ULF geomagnetic pulsations at different latitudes in Antarctica. Ann Geophys 27:3621–3629.

Fukushima, K. (1988), Neocognitron: A hierarchical neural network capable of visual pattern recognition, Neural networks 1.2: 119-130.

Gleisner, H., and H. Lundstedt (2001), A neural network-based local model for prediction of geomagnetic disturbances, J. Geophys. Res. Space Phys. 106 (A5), 8425–8433

Gomes, G. S., T. B. Ludermir, and L. M. R. Lima (2011), Comparison of new activation functions in neural network for forecasting financial time series, Neural Computing and Applications, doi: 10.1007/s00521-010-0407-3.

Gu, J., Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, 2018, Recent Advances in Convolutional Neural Networks, Pattern Recognition, vol. 77, pp. 354-377, arXiv: 1512.07108v6.

Heilig B, Lotz S, Verö J, Sutcliffe P, Reda J, Pajunpää K, Raita T (2010) Empirically modelled Pc3 activity based on solar wind parameters. Ann Geophys 28:1703–1722.

Hinton G. E., N. Srivastava, A. Krizhevsky, I. Sutskever and R. R. Salakhutdinov (2012), Improving neural networks by preventing co-adaptation of feature detectors, arXiv:1207.0580v1 .

Ioffe S. and C. Szegedy (2015), Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, eprint:1502.03167, arXiv:1502.03167.

Jacobs, J. A., Y. Kato, S. Matsushita, and V. A. Troitskaya, 1964, Classification of geomagnetic micropulsations. J. Geoph. Res., 69(1), 180–181. doi:10.1029/jz069i001p00180.

Kessel RL, Mann IR, Fung SF, Milling DK, O'Connell N (2004) Correlation of Pc5 wave power inside and outside the magnetosphere during high speed streams. Ann Geophys, 21:629–641.

Krizhevsky, A., I. Sutskever, and G. Hinton (2012), ImageNet classification with deep convolutional neural networks. In *Proc. Advances in Neural Information Processing Systems 25* 1090–1098.

Kingma D. P. and J. Ba (2014), Adam: A Method for Stochastic Optimization, 3rd International Conference for Learning Representations, San Diego, 2015, eprint:1412.6980, arXiv:1412.6980.

Park J., M. Noja, C. Stolle, and H. Lühr, 2013, The Ionospheric Bubble Index deduced from magnetic field and plasma observations onboard Swarm, Earth Planets Space, 65: 1333–1344.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998), Gradient-Based Learning Applied to Document Recognition, Proc. of the IEEE, vol. 86, no. 11, pp. 2278-2324, doi: 10.1109/5.726791.

LeCun, Y., Bengio, Y., and Hinton, G. (2015), Deep Learning, Nature, **521**: 436–444. doi: 10.1038/nature14539.

Li, X. (2001), Quantitative prediction of radiation belt electrons at geostationary orbit based on solar wind measurements, Geophys. Res. Lett. 28 (9), 1887–1890.

Lundstedt, H. (1997), AI techniques in geomagnetic storm forecasting. In: Tsurutani, B.T., Gonzalez, W.D., Kamide, Y., Arballo, J.K. (Eds.), Magnetic Storms. American Geophysical Union, Washington, D.C.

Lundstedt, H. (2005), Progress in space weather predictions and applications, Adv. Space Res. 36 (12), 2516–2523.

Mann IR. 2016, Waves, particles, and storms in geospace: an introduction, in: Waves, particles, and storms in geospace. Balasis G, Daglis IA, Mann IR (Eds.), Oxford University Press, Oxford, pp.1–14.

Mathie RA, Mann IR (2001) On the solar wind control of Pc5 ULF pulsation power at mid-latitudes: Implications for MeV electron acceleration in the outer radiation belt. J Geophys Res. doi:10.1029/2001JA000002.

Menk, F. W., 2011, Magnetospheric ULF Waves: A Review, in The Dynamic Magnetosphere, IAGA Special Sopron Book Series, Vol. 3, Part 4, 223–256, Springer, Netherlands, doi: 10.1007/978-94-007-0501-2_13.

Olsen, N., C. C. Finlay, S. Kotsiaros, et al., 2016, A model of Earth's magnetic field derived from 2 years of Swarm satellite constellation data, Earth Planets Space, 68, 124, doi: 10.1186/s40623-016-0488-z

Olsen, N., E. Friis-Christensen, R. Floberghagen, P. Alken, C. D. Beggan, A. Chulliat, E. Doornbos, J. T. da Encarnacao, B. Hamilton, G. Hulot, … , The Swarm Satellite Constellation Application and Research Facility (SCARF) and Swarm data products, 2013, Earth Planets Space, 65, 1189–1200, doi: 10.5047/eps.2013.07.001

Papadimitriou, C., G. Balasis, I. A. Daglis, and O. Giannakis, 2018, An initial ULF wave index derived from two years of Swarm observations. Ann. Geophys., 36, 287–299, doi:10.5194/angeo-36-287-2018.

Park, S., and N. Kwak (2017), Analysis on the Dropout Effect in Convolutional Neural Networks, in: Computer Vision – ACCV 2016, Revised Selected Papers, Part II, LNCS 10112, Lai S.-H., Lepetit V., Nishino K., and Sato Y. (Eds.), pp. 189–204, Springer International Publishing AG, Switzerland, doi: 10.1007/978-3-319-54184-6_12.

Pedamonti, D. (2018), Comparison of non-linear activation functions for deep neural networks on MNIST classification task, arXiv: 1804.02763v1.

Piel A. (2017) Definition of the Plasma State. In: Plasma Physics. Graduate Texts in Physics. Springer, Cham. https://doi.org/10.1007/978-3-319-63427-2_2

Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1988), "Learning representations by back-propagating errors." Cognitive modeling 5.3: 1.

Schmidhuber, J. (2015), Deep Learning in Neural Networks: An Overview, Neural Networks, **61**: 85–117.

Sutcliffe, P.R. (1997), Substorm onset identification using neural networks and Pi2 pulsations, Ann. Geophys. 15 (10), 1257–1264.

Srivastava N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014), Dropout: A Simple Way to Prevent Neural Networks from Overfitting, Journal of Machine Learning Research, 15, 1929-1958.

Stolle C., H. Lühr, M. Rother, and G. Balasis, 2006, Magnetic signatures of equatorial spread F as observed by the CHAMP satellite, J. Geophys. Res., 111: A02304, doi: 10.1029/2005JA011184.

Theodoridis, S., and K. Koutroumbas (2003), Pattern Recognition, 2nd Ed., p. 689, Elsevier, Amsterdam, The Netherlands, ISBN: 0-12-685875-6.

Valdivia, J. A., A. S. Sharma, and K. Papadopoulos (1996), Prediction of magnetic storms by nonlinear models, Geophys. Res. Lett. 23 (21), 2899–2902.

Vandegriff, J. (2005), Forecasting space weather: predicting interplanetary shocks using neural networks, Adv. Space Res. 36 (12), 2323–2327.

Vassiliadis, D. (2000), System identification, modeling, and prediction for space weather environments, IEEE Trans. Plasma Sci. 28 (6), 1944–1955.

Walker, A. D. M., 2005, Magnetohydrodynamic Waves in Geospace: The Theory of ULF Waves and their Interaction with Energetic Particles in the Solar-Terrestrial Environment, Series in Plasma Physics, Institute of Physics Publishing, Bristol and Philadelphia, ISBN: 0750309105

Wing, S., et al. (2005), Kp forecast models, J. Geophys. Res. Space Phys. 110, A4.