Gradu Amaierako Lana / Trabajo Fin de Grado
Fisika Gradua / Grado en Física

Changes on large-scale environmental conditions due to the impact of global warming on shallow cumulus convection

Semantic segmentation of shallow cumulus convection using deep learning techniques and identification of global warming effects and its consequences on them

Egilea/Autor/a:
Eneko Ruiz Fernández
Zuzendaria/Director/a:
Luís Javier Rodriguez Fuentes

Leioa, 2020ko irailaren 1a /Leioa, 1 de Septiembre de 2020

Contents

1 Introduction and aim

While looking at satellite imagery mesoscale patterns of shallow cumulus convection is a common feature, organizations on these scales have been largely ignored both in simulations and models. For the sake of simplicity, henceforth shallow cumulus convection will be reduced to SCC.

The aim of this thesis therefore is to study SCC and the role they play in the global climate, from a climate physics scope. For that purpose, the basics thermodynamics of cloudy air will be illustrated first. Then, atmospheric convection and how shallow cumulus are formed and vanished through convective processes will be understood.

After understanding the basic physics under individual shallow cumuli parcels, their pattern organization in the mesoscale will be studied. As said before, very few studies dealing with this topic have been published yet, hence those pattern identification (and naming) is still in a novice stage. We will choose the ones given by [1]: Gravel, Fish, Flower and Sugar. A more in-depth description of the visual features of the patterns will be provided later.

[2] studied relations between those clouds and large-scale environmental conditions using 19 years data east of Barbados, in the trade winds region. Their results suggest that mesoscale organization of shallow clouds may change due to global warming. Therefore, that work will be presented and results analysed.

Because of the fact that the study was geographically confined to one determined place, afterwards the work presented by [3], who extended the study to three other regions in the Pacific Ocean, will be studied. In order to extend their analysis to the whole globe, they created a deep learning algorithm to analysis clouds images taken from satellites during the whole year 2017 and to link patterns appearance to certain large-scale environmental conditions, to enable the analysis of how those pattern location and frequency change with global warming.

Since 2015, Deep Learning is increasing in popularity both in the industry and in academics, due to its outstanding ability to deal with large amounts of data. From particle physics to statistical physics, researches have taken advantage of this new tool to boost their progress in their respective fields.

Concerning Geophysics and climatological sciences, it has been applied in different studies. We will briefly summary them and introduce the main concepts of Deep Learning. Later we will develop our own algorithm to classify SCC following the four patterns. We will compare different networks (Deep Learning is based on deep neural networks or NN) to decide which one has classified better the patterns, after analysing the data available.

Lastly, we will present a recently publish unsupervised deep learning method that presents promising results that may ease a lot the work of classification clouds and that can be very useful in future studies

2 Development

First and foremost, we will define very briefly what a cloud is. Their formation (in the case for SCC) will be explainer later. According to Britannica [4], "clouds are any visible mass of water droplets, ice crystals or a mixture of both that is suspended in the air, usually at a considered height. They are formed when relatively moist air rises."

Yet we will focus our work on cumulus clouds. According to Wikipedia [5], "Cumulus clouds have flat bases and are often described as fluffy in appearance". They form at low altitudes (less than 2000m, thus in the troposphere) and can cover the entire sky, or only part of it. They are usually precursors of other clouds, for instance, cumulonimbus, when certain weather factors, such as instability, moisture and temperature gradient, are met.

Shallow cumulus are clouds with low cloud cover, between 5% and 20%, and are normally accompanied with good weather. As their vertical extent is not enough to consider them as precipitative, we will assume that they never produce precipitations, ignoring microphysics issues dealing with transportation of liquid water to ice and precipitation. We will ignore too the interaction of the precipitations with the dynamic.

Once understood what cumulus clouds are, we will introduce the basic thermodynamics of cloudy air.

2.1 Thermodynamics of clouds

First, we will introduce some variables and put them together into the law of ideal gases. This section, 2.1, as well as 2.2 are mainly based on [6].

We can express the density of an isothermal parcel as the sum of dry air, water vapour and liquid water densities as:

$$\rho = \frac{m}{V} = \rho_d + \rho_v + \rho_l \tag{1}$$

Following the law of ideal gasses, the pressures for water vapour and dry air are:

$$p_d = \rho_d R_d T \quad , \quad p_v = \rho_v R_v T \tag{2}$$

where R_x is the specific gas constant of either water vapour or dry air. According to Dalton's law, we can add the two pressures to obtain the equation of state of a moist parcel:

$$p = p_d + p_v = \rho R_d T (1 + (\epsilon^{-1} - 1)\frac{\rho_v}{\rho} - \frac{\rho_l}{\rho}) \quad where \quad \epsilon = R_d/R_v \approx 0.622 \tag{3}$$

Here, we introduce the specific humidity for water vapour q_v and liquid water content q_l, defined as:

$$q_v = \frac{\rho_V}{\rho} \quad , \quad q_l = \frac{\rho_l}{\rho} \tag{4}$$

Then, rewriting 3:

$$p = \rho R_d T_v \quad , \quad T_v = T(1 + 0.61q_v - q_l) \tag{5}$$

T_v is the virtual temperature, defined as the temperature that dry air must have in order to have the same density as the moist air. Thus, for a given pressure, T_v is inversely proportional to density and a direct measurement of the buoyancy of an air parcel (excess of virtual temperature).

Moist air ($q_v > 0$) has a higher virtual temperature and therefore a higher buoyancy than dry air ($q_v = 0$), at the same pressure. Presence of moisture increases the buoyancy of a parcel, even though liquid water decreases it.

Another import variable is q_s, the saturation specific humidity or the humidity from which water vapour condensates into droplets. We can define it in terms of total pressure and saturation vapour pressure, $e_s = \rho_s R_v T$, :

$$q_s = \frac{\rho_s}{\rho} = \epsilon \frac{e_s}{p + e_s(\epsilon - 1)} \tag{6}$$

2.1.1 Conserved variables

To study turbulence mixing of heat and moisture we will eliminate internal sources and sinks of them, absorbing them into heat and moisture variables. In this way, we will be able to express the combination of two parcels as the linear combination of the properties of them.

Ignoring again precipitation and ice phase, the only sinks and sources of humidity are condensational effects. The total water specific humidity will be then $q = q_v + q_l$. This variable will be conserved in all condensational processes and pressure changes.

The sources and sinks of heat we will convert into conserved variables are the temperature changes due to adiabatic compression and both the expansion and the temperature changes due to condensational effects. Therefore, two variables will be introduced. A first variable will be account of dry adiabatic processes and the second for moist one (processes in which condensational effects should be taken into account).

Neglecting ice phase and transition of water to precipitation and combining the first and second principles of the thermodynamics we obtain:

$$ds(p, T, q_l) = c_{pm} d \ln T - R_m d \ln p - \frac{L}{T} dq_l \tag{7}$$

where $c_{pm} = c_{pd} + q_v C_{pv} + q_l c_{pl}$ is the mean parcel value for specific heat capacity and $R_m = R_d + q_v R_v$ is the specific gas constant.

This result is exact, but some approximation can be made when working with shallow cumulus convection. First, we can approximate both specific heat and gas constant mean values to the ones of dry air. Besides that, the last term, which represents condensational effects, can be reduced to $d(L_0 q_l/T)$.

For dry processes, $q_l = 0$, the remaining terms represent the potential temperature, defined as $ds = c_{pd} d\ln\theta$. Matching this value for the simplified Equation 7 and integrating from a reference state (usually defined as $p_0 = 1000hPa$):

$$\theta = T\left(\frac{p_0}{p}\right)^{\kappa} = T\pi^{-1} \quad where \quad \pi = \left(\frac{p}{p_0}\right)^{\kappa} \tag{8}$$

Here θ is defined as the potential temperature, the temperature an air parcel would obtain if compressed or expanded adiabatically ($ds = 0$) to a pressure of $1000hPa$. Since there is no water, there will be no phase change ($dq_l = 0$).

If we assume moist air, this simplification will not work, thus we will have to work with the potential liquid water temperature θ_l, where possible phase changes are considered. Integrating Equation 7 in the reference state $p_0 = 1000hPa$ and $q_l = 0$:

$$\theta_l = \theta \exp\left(-\frac{L_0 q_l}{c_{pd} T}\right) \tag{9}$$

Since q_l is an small term in clouds, it can be Taylor expanded to:

$$\theta_l \approx \theta - \frac{L_0}{c_{pd}\pi} q_l \tag{10}$$

2.1.2 Lapse rates and atmosphere's local stability

As an alternative to θ and θ_l, static energies will be used. The only difference between this variables and the previous one is that when deriving static energies we will assume hydrostatic pressure changes. From Equation 7:

$$dh(T, z, q_l) = c_{pm} dT + g dz - L dq_l \tag{11}$$

The same simplifications as above will be taken. The result for unsaturated parcel, i.e. relative humidity less than 100% and thus $dq_l = 0$, is :

$$h_d = c_{pd} T + gz \tag{12}$$

Nevertheless, if the parcel is saturated:

$$h_l = h_d - L_0 q_l \tag{13}$$

If we consider an insulated raising parcel of air, it will cool adiabatically due to expansion. This "cooling rate", known as lapse rate, depends heavily on whether the parcel is saturated (moist) or unsaturated. In the latter case, h_d will be constant for the case under study. Then, solving Equation 12 for T and taking derivatives, we obtain the lapse rate:

$$\Gamma_d = -\left(\frac{dT}{dz}\right)_{parcel} = -\left(\frac{\partial T}{\partial z}\right)_{h_d} = \frac{g}{c_{pd}} \approx 9.76K/km \tag{14}$$

This result is not so direct for saturated parcels, because the latent heat released as vapour condensates into water decelerating this process. Before solving Equation 13 for T, we will substitute the liquid water content for the water content before saturation minus the water content as it saturates ($q_l = q_t - q_{sat}(p, T)$), in order to make T only z and p dependant. Assuming h_l will remain constant and hydrostatic equilibrium:

$$\Gamma_m = -\left(\frac{dT}{dz}\right)_{parcel} = -\left(\frac{\partial T}{\partial z}\right)_{h_l,q_t} = \Gamma_d \frac{\left(1+\frac{Lq_s}{R_dT}\right)}{\left(1+\frac{L}{c_p}\frac{\partial q_s}{\partial T}\right)} \tag{15}$$

The moist adiabatic lapse rate, despite depending heavily on temperature, will be always smaller as its counterpart for the dry case. The lower the temperature, the bigger the rate and the smaller the difference. For instance, for $T = 290K$ and $p = 1000hPa$, $\Gamma_m = 4.5K/km$.

Once introduced this rate, we will dive into local stability, a factor which influences heavily the intensity of convection. We can define the stability as the sign of the buoyancy of the atmospheric parcel displacing vertically. In order to give explain rigorously stability, we will provide it with a proper mathematical corpus, since dry and moist cases have again different treatments.

We will consider an air parcel at a height z_0 with the atmospheric properties of this height, surrounded by a far bigger "environmental" parcel with average temperature and pressure $\bar{T}$ and $\bar{P}$. If we displace infinitesimally the smaller parcel a distance δz, following second Newton's law:

$$\frac{dw}{dt} = \frac{d^2\delta z}{dt^2} = B(z+\delta z) \approx \frac{dB}{dz}\delta \tag{16}$$

B is the buoyancy force per unit mass working on the ascending parcel, defined as (taking into account Equation 5 and that parcel's pressure adjusts instantaneously with environmental's):

$$B = -g\frac{\rho_p - \bar{\rho}}{\rho_p} = g\frac{T_{v,p} - \bar{T_v}}{\bar{T_v}} \tag{17}$$

where subscript p denotes parcel properties and $\bar{T_v}$ is the mean virtual temperature. In Equation 16 an harmonic oscillation can be identified. Its solutions will be either exponentially grow or oscillate depending on RHS' sign. Consequently, as said before, stability depends on the sign of buoyancy.

Derivating Equation 17 with respect to height and assuming linearity of differentiation, the atmosphere will be stable for $\frac{d\bar{T_v}}{dz} > \frac{dT_{v,p}}{dz}$ and unstable when $\frac{d\bar{T_v}}{dz} < \frac{dT_{v,p}}{dz}$. Thus, we have to compare the lapse rate of T_v of a parcel with the vertical gradient of the atmospheric profile of T_v. We can define the virtual temperature lapse rate, or environmental lapse rate, as $\Gamma_e = -\left(\frac{\partial T}{\partial z}\right)$.

When differentiating moist and dry air parcel, we introduce conditional instability, which is illustrated in Figure 1. There are lustrated the adiabats of the ascending parcel (both in dry and moist cases) and the surrounding environmental one too. The adiabat is the curve that describes the changes in temperature and/or pressure of a gas.

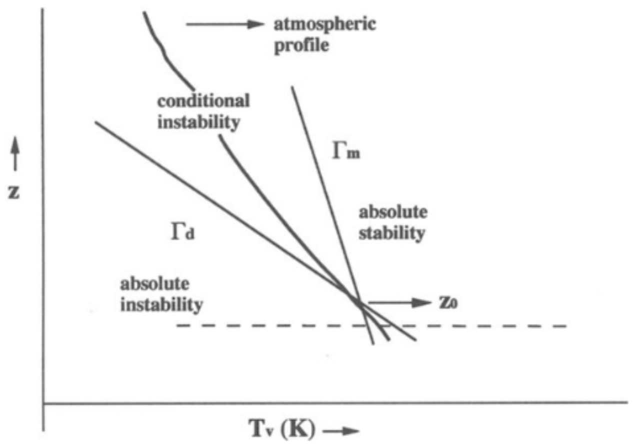

Figure 1: Conditional unstable profile, as well as dry and moist adiabats. Image taken from [6].

If $\Gamma_e > \Gamma_{d,m}$, either saturated or unsaturated parcels will be warmer than the environment and will continue to ascend. We describe this situation as absolute instability. However, if $\Gamma_d > \Gamma_e > \Gamma_m$, the saturated parcel will still be warmer and ascend, but unsaturated one will be cooler and descend (conditional instability). Lastly, if $\Gamma_{d,m} > \Gamma_e$, both parcels will be cooler and hence sink (absolute stability).

2.2 Cloud convection

Now, we will study how the stability linked to vertical displacement affects moist convection. For this purpose, we will consider an ascending air parcel, called thermal, due to the heat in the surface layer produced by sunlight radiation. This process is illustrated below in Figure 2:

The parcel raises through the dry convective boundary layer, cooling with a constant Γ_d lapse-rate until it gets to the lifting condensation layer or LCL, the level where clouds form. When the air is cooled adiabatically, e_s decreases exponentially while q_v remains constant. Therefore, relative humidity (RH) increases and eventually the thermal will get oversaturated (RH = 100%).

At that point, the thermal has negative buoyancy and it will start falling unless it has enough kinetic energy. If it does have it, it will continue rising, yet with moist lapse rate Γ_m because droplets have started to form, due to the growth of RH above 100%. Since the cloud layer is conditionally unstable, the moist adiabat will intersect the atmospheric profile at the level of free convection (LFC), when the parcel becomes positively buoyant respect to the environment.

It will continue rising until moist adiabat crosses again the profile in the inversion layer, height called the level of neutral buoyancy LNB. Again, owing to the fact that the thermal

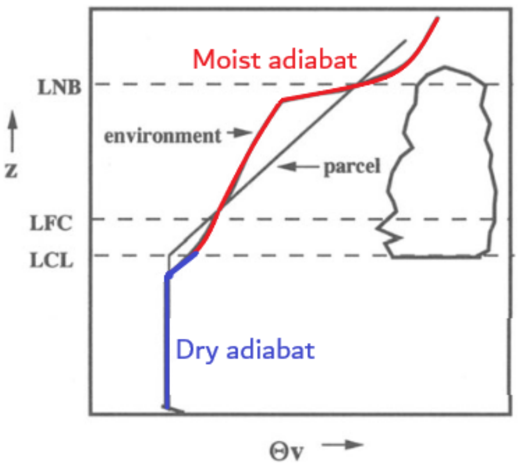

Figure 2: Height against $\hat{\theta}_v$ for both environmental and parcel adiabats. Coloured by the author of the present work. Image taken from [6].

has kinetic energy, it may penetrate slightly in that inversion layer. In this model, the cloud base can be found at LCL and the top at LNB.

In order to know how much energy is available for moist convection, we can define CAPE (convective available potential energy) as the buoyancy of the parcel. Hence, the stability of the parcel depends on where it is released. CAPE, as the function of the position, is defined as:

$$CAPE(z, z_0) = \int_{z_0}^{z} B dz = R_d \int_{p(z)}^{p(z_0)} T_{v,p} - \dot{T}_v d\ln p \tag{18}$$

This CAPE will only be consumed if the thermal has enough kinetic energy and overcomes the "potential barrier" between LFC and LCL. In that case, this potential energy can be consumed into kinetic energy by cloud dynamics.

2.2.1 Cloud life cycle

Figure 3 shows the stages of the cloud life cycle. The cloud is born due to the thermal rising in the dry convective layer. Eventually, it will reach LFC. If the cloud top is between LFC and LCL the cloud is called "forced cloud" since it is still negative buoyant and it is forced mechanically by the subclouds thermals below.

However, if it has enough energy, as commented above, it will reach the LCL and become positively buoyant, starting the so-called active phase. In this phase, rapidly growing turrets with sharp interfaces can be seen. The cloud will continue developing until it reaches the LNB (or a slightly higher height, due to kinetic energy). At this point, if the fuelling of moist air from the subcloud layer ends, the clouds enter in the passive phase. In this phase, the cloud base disappears, the interface becomes fuzzy and the clouds dissolves completely by evaporation.

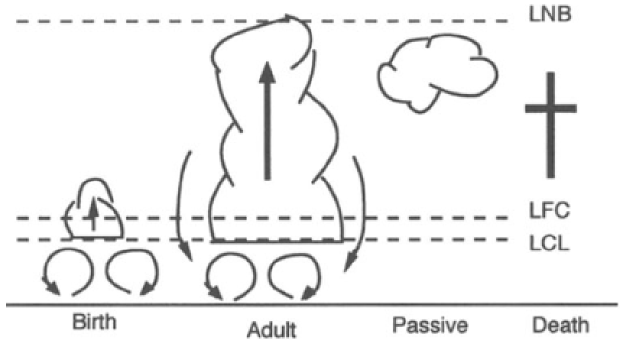

Figure 5. **Schematic life cycle of a cumulus cloud.**

Figure 3: Cumulus cloud life cycle. Image taken from [6].

If this were the actual life cycle of the cloud, the convection of moisture and heat would be explained by the parcel concept introduced above. This is not nevertheless the case. It has been proved during many penetrations of cumulus clouds at different heights that the water content q_l is lower than the value predicted by the parcel method. Indeed, it decreases with height.

This phenomenon is due to entrainment, which describes the turbulent mixing of the cloud with its environment and affects its top height, vertical velocity and in-cloud fields. However, this is not the topic of the present work and the interested reader is referred to the pages 454-478 of [6], where this phenomenon is more in-depth explained.

2.3 Mesoscale organization of SCC

We will now study shallow convective clouds in the mesoscale. To this scale belong weather systems ranging from around five kilometres to several hundreds of kilometres, but smaller than the synoptic scale. Inside this scale we distinguish again three categories : meso-α scale ($200 - 2000$km), meso-β scale ($20 - 200$km) and meso-γ scale ($2 - 20$km).

In a pair of paper published in 2019, [2] and [1], a group of 13 researchers supported by the International Space Science Institute (ISSI) started a cloud classification activity, in order to determine if those patterns could be linked to large-scale environmental conditions. The following sections are therefore mainly based on them.

They identified around 900 images from a region windward of Barbados in North Atlantic (13°N, 58°W) during the boreal winter. They looked for patterns in NASA's Worldwide's MODerate-resolution Imaging Spectroradiometer (MODIS) dataset [1]. From it, four patterns were extracted: Sugar, Gravel, Fish and Flower.

As noted by [2], Sugar patterns are formed by very fine cumulus clouds extended along

[1] https://modis.gsfc.nasa.gov/

wide-spread areas. They are the only patterns that do not present mesoscale organization. Neither are they very reflective nor have large pockets of cloud-free regions. The structure comes from the large-scale flow in which they are embedded. Flower patterns, which belong to meso-β scale, are formed by circular cloud structures, each area with a diameter between 50 and 500 km. Wide cloud-free regions can be found between those areas.

In the other hand, Fish patterns are formed by skeletal and elongated structures. Since those structures can extend up to $1,000$ km, they can be found in the meso-α scale. Lastly, meso-β scale belonging Gravel patterns form fields of granular structures. They are featured by arcs and rings, that can span around 20 kilometres. They can be seen in Figure 4:

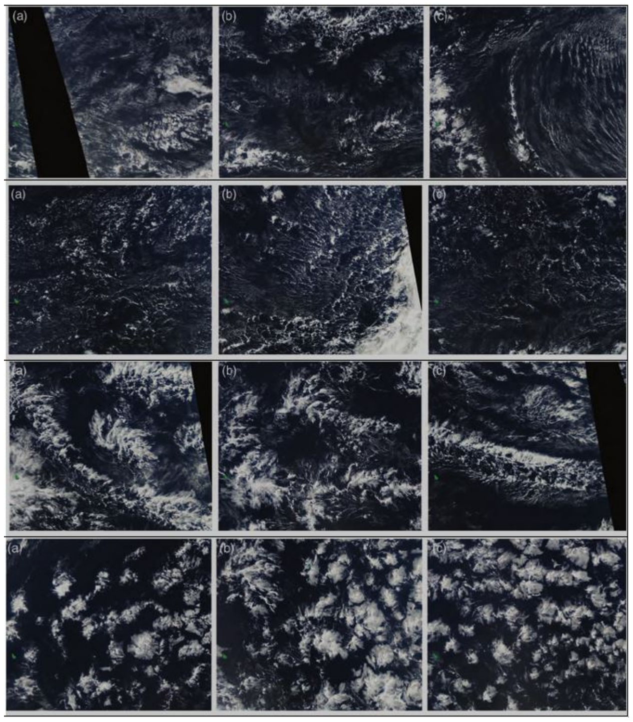

Figure 4: The four different patterns. Images in the first row belong to Sugar, in the second to Gravel, in the third to Fish and in the fourth to Flower. Image taken from [1].

The three images in row one of Figure 4 are Sugar clouds, three in row two are Gravel clouds. Fish and Flower mesoscale patterns can be found, respectively, in rows three and four.

Due to the restricted area under study, the obtained result should be taken with caution, as they may contain some bias, as it will be proved later, and might well not be representative of those patterns outside Barbados.

Once classified, the second paper [2] tried to link those patterns with large-scale environmental changes. As said before, shallow cumuli appearance is quite common in satellite images. Therefore, they can heavily influence Earth's radiation budget, the difference between the energy that is received from Sun and the energy that is radiated back, and climate sensitivity.

Despite being quite common, the response of trade-wind cumuli to global warming is a major source of uncertainty in climate models estimates of cloud feedback, in spite of the advantages made last 10 years. Likewise, the role of mesoscale organization of clouds has been ignored and not included in models, although it was known long ago that organization did exist.

In order to shed light on mesoscale organization SCC role in large-scale environmental conditions and radiation budget of Earth, [2] studied daily and interannual variability of some environmental variables in the appearances of the patterns. They tried to give the patterns a more robust classification too, rather than the current purely visual one.

2.3.1 Cloud classification

The data analysed by [2] was based on 3-hourly infrared ($11 \mu m$) brightness temperature (T_b), gridded ($0.7°$) data from the GridSat-B1 dataset from the tropical Atlantic Ocean east of Barbados ($48 - 58°$W, $10 - 20°$N) during boreal winter from 2000 to 2019. Patterns will be identified and classified by trained scientists, being each image shown to more than one classifier.

To characterize this population, due to the daily variation of clouds organization, two metrics have been chosen: the mean object size S and a clustering measure I_{org}. S is defined as $S = \frac{A}{N} \times 10^4$, where A is the total fractional area covered by shallow clouds and N the number of cloud objects.

"I_{org} compares the distribution of the nearest neighbour distances among the centroids of objects to that expected for a random distribution objects"[2]. Therefore, $I_{org} = 0.5$ means that the centroids are randomly distributed. Values less than 0.5 correspond to regular distribution and higher than 0.5 agree to clustered (and thus organized) distributions.

From those images, we will define 4 quadrants, which match upper and lower terciles of S and I_{org} and compared them to robustly classified images by [1], as seen in below Figure 5. The grey points belong all to a class, but they are not the best representative of it. We understand robustly classified as classified equally by at least 4 people.

Following I_{org}, Fish and Sugar are more clustered cloud patterns and Gravel and Flower more randomly distributed. Thus, in Quadrant A occurs mainly Flowers, in B Fish, in C Gravel and in D Sugar. From Figure 6 can be seen that patterns are separated reasonably well. Sugar and Gravel, linked with small-scale cloud features, fall into lower tercile of S, while Fish and Flowers (more extended features) in the upper.

These results were compared to data obtained from higher resolution MODIS (1km) channel 31 and GOES-16 (2km). Despite higher resolution changes the number of clouds objects N and therefore the mean cloud object size S, as well as the absolute value of I_{org}, the data correlates well.

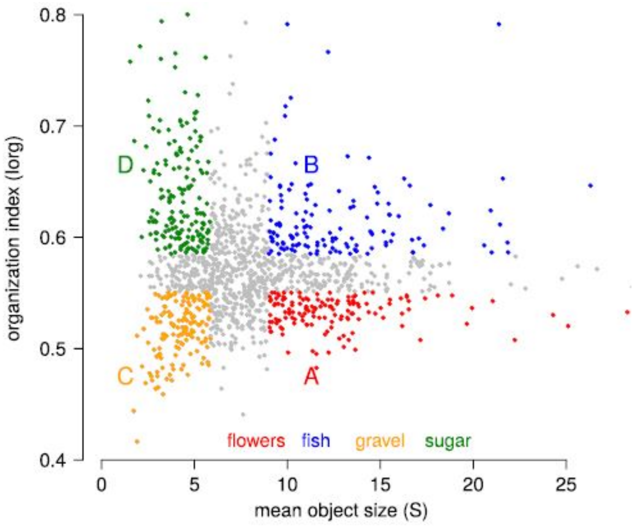

Figure 5: Classification of SCC mesoscale patterns using I_{org} and S metrics. Image taken from [2].

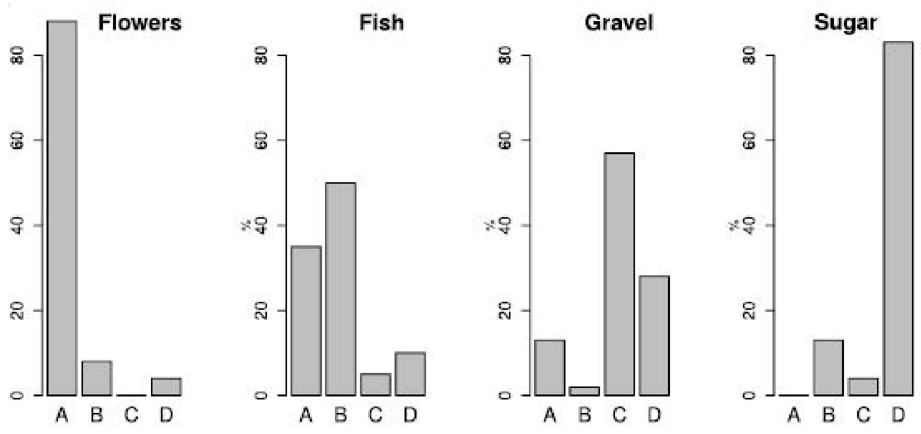

Figure 6: Histograms showing in which quartile of 5 was each pattern more frequent . Image taken from [2].

2.3.2 Patterns and large-scale environment

For identification of large-scale environmental conditions and their changes, all data from ERA-interim product [7] will be reanalyses, for each boreal winter or DJF from 2000 to 2019. The large-scale environmental conditions under study are : SST (sea surface temperature), near-surface wind speed V_S, the zonal and meridional components of the surface wind u_s and v_s, the zonal wind shear between $700hPA$ and the surface, the large-scale vertical velocity at $700hPa$, the lower-tropospheric stability LTS (defined as $LTS = \theta_{700} - \theta_{1000}$) and estimated inversion strength EIS (defined as $EIS = LTS - \Gamma_m^{850}(z_{700} - LCL)$, when the surface RH is 80%). SST, V_s and EIS can be seen in Figure 7:

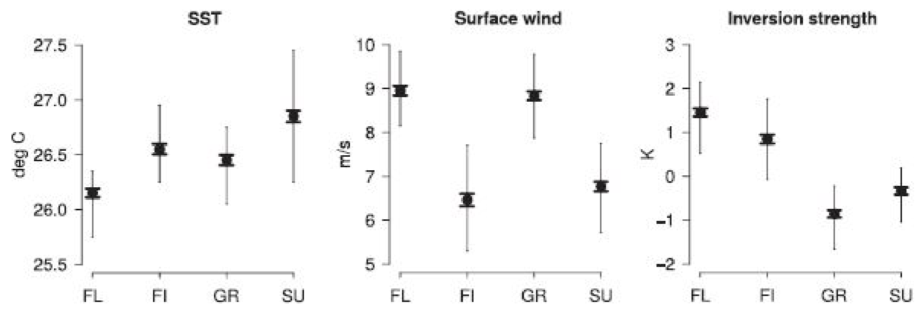

Figure 7: Daily mean of SST, V_s and EIS over the 200-2019 period. Image taken from [2].

From the above listed variables, only V_S and EIS differ significantly among patterns in daily a scale.

As it can be seen in Figure 7 and 8, Flower and Gravel are associated with a strong surface wind V_s and Sugar and Fish with weaker one. On the other hand, Sugar and Fish could be found in regions with strong stability (the greater the EIS, the bigger the stability) and the resting ones in regions with low EIS. Lastly, Flower was associated with cold SSTs, Fish and Gravel with moderate SSTs and Sugar with warm SSTs.

To summarize, EIS discriminates very well clouds based on their size S. Large patterns (Flower and Fish) occur in a stable lower troposphere, since they are linked with high stability.

However, V_s successfully discriminates patterns based on convective organization I_{org}. More organized patterns ($I_{org} > 0.5$), Sugar and Fish, tend to happen when trades are weak, whereas more randomly organized patterns, such as Flowers and Gravel, (I_{org} around 0.5) appear for strong winds, that is, $V_s > 8m/s$. This data was compared with the one obtained from MODIS, receiving very similar results, as can be seen in Figure 8:

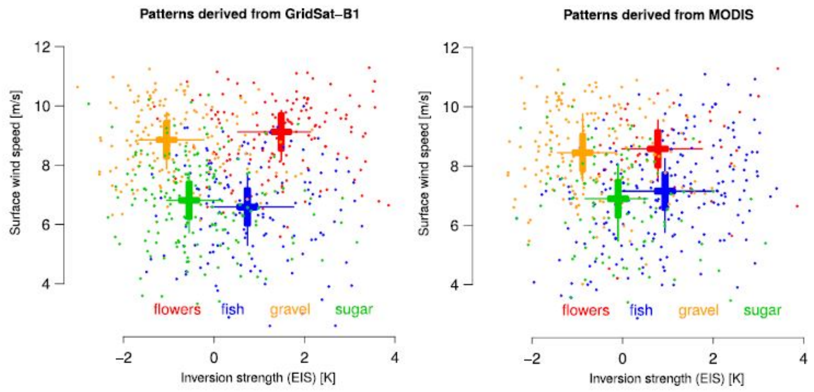

Figure 8: Daily mean values of EIS and V_s during the 2000-2019 period for both GridSat and MODIS observations. Image taken from [2].

Resulted derived from daily observations can be generalized to interannual results too, with the usual variabilities due to abnormal values measured some years. For instance, the 2009-2010 season exhibited unusually weak V_S and strong EIS and thus a more often Fish patterns. However, 4 years later in the 2013-2014 season, strong V_S were usual and consequently, Gravel and Flower patterns were easier to spot. Lastly, in 2017-2019 DJF came with Gravel and Sugar patterns, due to weak stability.

2.3.3 Radiative effects

To check whether it is a connection between clouds' radiative feedback and patterns data from CERES (Clouds and Earth's Radiant Energy System) geostationary enhanced temporally interpolated data set [8] in the 2001-2017 period will be used.

Cloud radiative effect (CRE) is defined as the difference in radiation budget of (average) cloudy sky and cloud-free one. Clouds have influence in both incoming radiation and the radiation emitted back by Earth. It can be mathematically defined as:

$$\Delta R_{TOA} = R_{Average} - R_{Clear} = \Delta Q_{abs} - \Delta OLR. \tag{19}$$

where the former term is short-wave effect and the latter the long-wave effect. ΔQ_{abs} and ΔOLR can be discomposed as:

$$\Delta Q_{abs} = S_0/4(2 - \alpha_{clody} - \alpha_{clear}) \qquad\qquad \Delta OLR = \sigma T_z^4 - F_{clear}^{up} \tag{20}$$

where S_0 is the solar constant, and α_x is the albedo for cloudy and clear day. In the second equation σ is the Stefan-Boltzmann constant, T_z is the temperature at the given height and F_{clear}^{up} is the upward flux in clear conditions.

Clouds reflex around the $15 - 30\%$ incoming solar radiation, decreasing by around $44W/m^2$ the radiation absorbed by Earth. However, they contribute too to the greenhouse effect, reducing the outgoing long-wave radiation by about $31W/m^2$. Thus, the net contribution is the loss of $13W/m^2$, so clouds cool up Earth.

It is clear that the bigger the cloud coverage, the more negative radiation budget will be. In our model, low-level coverage doubles across different patterns and net CRE of Flower patterns is two times bigger than the one of Sugar. However, it is cloud-coverage and not patterns which influence CRE since not difference across patterns for a given amount of low-cloud coverage has been observed. This can be seen in Figure 9.

The question then is how will global warming affect low-clod coverage and hence cloud-radiative feedback for trade-wind cumuli. In experiments run with IPSL climate model [9] a rise of EIS over the tropical western Atlantic with global warming has been noticed. This surge fluctuates from 0.1 to $0.7KK^-1$, depending on the model. V_s does not change in a noticeable way. If EIS increases, assuming that only V_s and EIS control mesoscale organization of shallow cumuli, Fish and Flower patterns will be more common.

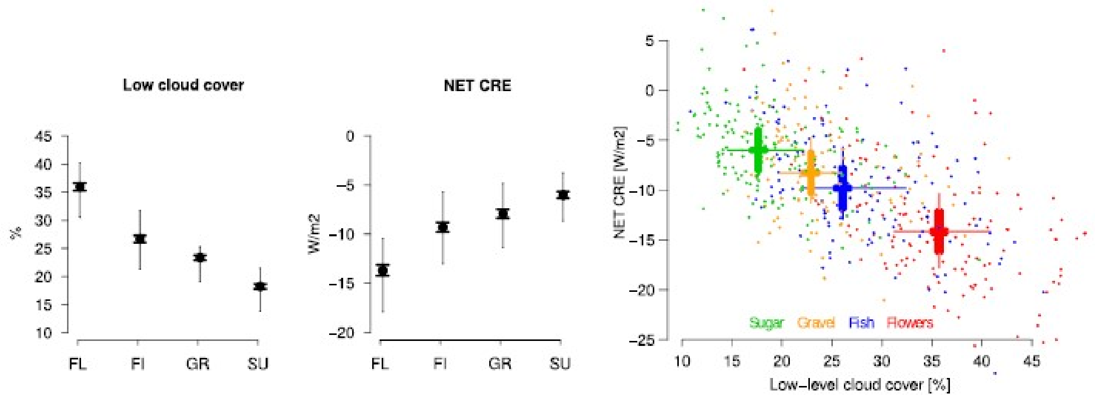

Figure 9: Low cloud cover, NET CRE and the relationship between those patterns for each pattern. Image taken from [2].

These patterns are linked with a higher low-level cloud cover. Thus, against previous models which do not take into account mesoscale organization, low-level cloud coverage will peak with global warming.

In their analysis, SST was proved to have no influence in mesoscale organization on a daily and interannual basis, even though it may affect climate warming.

2.3.4 Problems of the current method

The main problem of the previous approach was the fact that the study was only conducted in one specific location, which may introduce some bias in the analysis of the result. In order to avoid that, the same study should be conducted in a larger geographical area. However, this is greatly time-consuming and may distract trained scientist from more important tasks.

To avoid that need of humans classification images based on visual patterns, [3] introduced an alternative: training a Machine Learning ML or Deep Learning (a category of ML) DL algorithms to automate that task.

This method has many advantages when compared with the previous one, based solely on human labelled examples. However, the human labelling is still necessary, in order to have enough labelled data that would enable to the algorithm to make accurate predictions, which is why two days were set up (2nd and 29th November 2018) in the Max Plack Institute for Meteorology in Hamburg and in The Laboratoire de Métérologie Dynamique in Paris.

They labelled 30,000 images (although less will be used in the algorithm). Unlike [2], they used different geographical areas and seasons ([2] only classified images upwind of Barbados during the boreal winter). They classified images during DJF (boreal winter), MAM (spring) and SON (autumn). The regions were chosen in the Pacific based on climatological similarities to the original study and can be seen in the map of Figure 10, as well as the area fraction of each pattern.

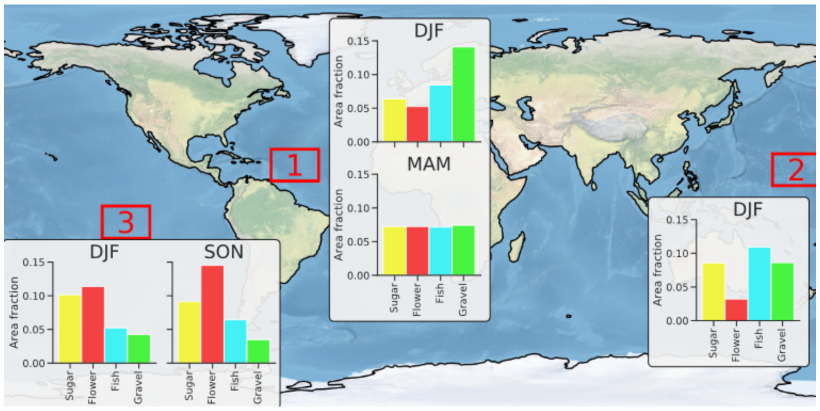

Figure 10: World map showing the area from which images and the period were taken and the area fraction of each pattern. Image taken from [3].

They performed a study to check if the four patterns correspond to meaningful cloud regimes based on their physical characteristic. To do that, they took data from ERA-Interim reanalyses corresponding to each pattern. It suggested that the patterns do appear in different climatological environments, a statement supported but the standard error being smaller than the difference between patterns.

However, this work of finding physical characteristics of each pattern, should not be confused by the one done by [2], which tried to justify the four-pattern-classification based on physical variables.

Flowers were found in a relatively dry and cold boundary layer with very strong inversion. On the other hand, Sugar was linked to warm and humid boundary layer with a strong downward motion. Fish and Gravel had similar large-scale environmental conditions, with weak inversion and downward motion. Inversion is defined as a deviation of the normal behaviour of a certain atmospheric property with altitude. In our case, we speak about temperature inversion, i.e. the temperature increases with height instead of decreasing, as it should do following lapse-rates. These characteristics can be seen in Figure 11.

Once presented the data, we will program an algorithm to detect and segment images based on satellite images, as [3] did, to later apply it worldwide to check if large-scale environmental conditions pointed out by [3] and [2] can be confirmed. However, we will present the first machine and deep learning and its techniques.

2.4 Machine and Deep Learning

When people speak about artificial intelligence (AI), they usually think about self-driving cars, instant voice recognition or robots working autonomously without any human supervision. Nevertheless, they are only thinking of a subfield of AI, Machine Learning (ML).

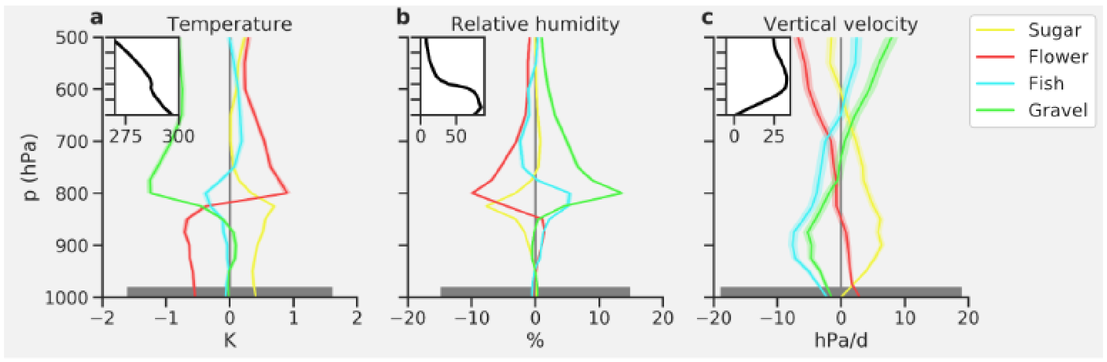

Figure 11: Median of large-scale environmental conditions for each pattern. Image taken from [3].

In this century we have already seen a fast development and application of those techniques in our daily life. For instance, Google's mail service, Gmail, uses a spam filter based on ML and Amazon's (or Apple's Siri) Alexa uses DL techniques to recognize voice commands. In the future, more challenging applications, such as self-driven cars are expected. This technology is a reality now though, at least if you live in Palo Alto.

In the following sections, we will introduce applications of ML and Deep Learning (DL), a subfield of ML, in Physics, to focus later on Geophysics and Climatological Physics, the area of our problem.

After presenting them, we will approach our problem, segmentation of shallow cumulus convection, and the techniques that will be used. Lastly, we will analyse the impact that different techniques have on our model's performance.

2.4.1 Physical application of Machine and Deep Learning algorithms

The main drawback of ML and DL techniques is that they need huge amounts of data, especially DL, to enhance their performance. Indeed, the more data we have, the more accurate our prediction will be.

Therefore, experimental physics, and particularly particle physics, is the area where ML has been applied for longer. In LHC, the data flow from detectors is around 25 GB/s. Yet all of this data is not saved, because much of it is noise. To discriminate between actual signal and noise, CERN developed its own ML Toolkit inside their data analysis software ROOT [10].

However, the topic of this work is climate physics, lest address it. In this field, DL algorithms have been mainly used to detect and extract visual features in images (i.e. taken from satellites or aircraft). However, as said before, the main disadvantage of this method is the vast amount of labelled data (data in which visual features have been extracted previously) needed to train them. This problem has greatly limited likely studies in the area. In order to face this issue, other ML algorithms, unsupervised ones, which cluster image based on similar features have been used, such as [11]. As we will see later, this approach can be useful to classify shallow phenomena too.

Another example of DL techniques in climatological sciences is [12], who successfully (with 89-98% accuracy) detected extreme weather events such as Tropical Cyclones, Atmospheric Rives and Weather Fronts on images.

However, the more similar work is the study carried out by [13]. They used 1000 hand-labelled image to train a neural network to classify different types of clouds. They trained the algorithm to distinguish between "No Mesoscale Cellular Convection (MCC)", "Closed MMC", "Open MMC" and "Cellular, but disorganized".

They studied the link existing between two characteristics of clouds and their variation depending on the mesoscale organization category, in order to link associate them with large-scale meteorology. The characteristics under study where cloud fraction CF (this was one of the parameters, among with I_{org} used by [1] to classify cloud into the patterns) and the moments of LWP, or liquid water path, defined as the total amount of liquid water present between two points in the atmosphere, or

$$LWP = \int_0^\infty \rho_{air} r_L dz \tag{21}$$

where ρ_{air} is the density of air including water and r_L is the liquid water mixing ratio.

Their network obtained an $85\% - 90\%$ classification accuracy. However, in the absence of the code and the data used (it is nowhere available), due to the success ratio of the algorithm, reasonable doubts about the quality of this classification may emerge. First, the studied is from 2005, 7 years before AlexNet[14].

This network won the 2012 ImageNet ILSVRC Challenge with an error rate of 17%, and it is considered the first modern neural network for Deep Computer vision task. In 2005 ANN where rudimentary and only achieve good accuracy rates when identifying handwritten digits.

Second, the authors used only two sets (one for training and other for verifications) and trained (and it assumed that tuned network parameters too) until the above accuracy was reached. Thus, it is possible that they may be suffering a problem of overfitting, because of the limited number of training images and the fact that they do not test their algorithm in an independent set, neither used to tune parameters nor to train the network.

The concept of overfitting can be defined as the generalization error which occurs when the gap between training error and test error is very big [15], i.e., our model is too specialised on the data we are training with. NN are prone to overfitting so that it is a recurrent problem in the area. In our model, this risk is taken into account, and measures to avoid it are implemented.

It could be too that they did obtain a non-overfitting algorithm that performed well because the patterns, as seen in the Figure 12, were far easier to detect and classify than the ones used by [1].

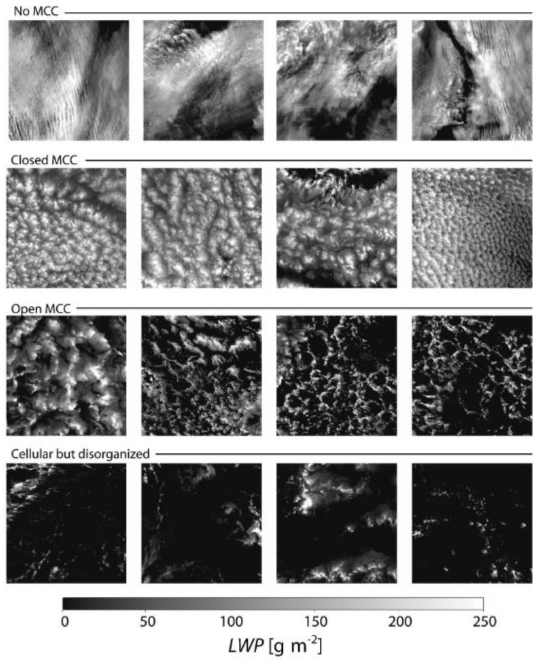

Figure 12: Cloud patterns classified by the neural network programmed by [13]. Image taken from [13].

[3] did find the algorithm, and applied it to the dataset obtained by [1]. The result was mainly ”Cellular, but disorganized”, therefore they decided to create a modern algorithm (with all the improvement from 2006 to 2019) to classify the patterns proposed by [1]. This is the task of the next section, build our very own DL algorithm to classify those images. We will use data proved by Kaggle [16] to train and test our algorithm.

2.4.2 Artificial Neural Networks

First models of Artificial Neural Networks or ANN brought inspiration from human's brain in order to build intelligent machines. However, this field has developed incredibly fast (especially the last decade) since firsts model were proposed back in 1943, and modern ANN are no longer similar to their biological counterparts.

This subsection, 2.4.2, and 2.4.3, as well as the part concerning optimization method in subsection 2.5.1, are mainly based on [15] and [17].

One of the simplest ANN architectures is the Perceptron. It is rooted on a neuron called threshold logic unit or TLU and both the input and output are tensors. The Perceptron associates each input with a weight $\boldsymbol{w}$ and a bias term $\boldsymbol{b}$. Then, a step function is applied to that connection such as:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = h(\boldsymbol{x}; \boldsymbol{w}, \boldsymbol{b}) = Heavside(\boldsymbol{x}^T\boldsymbol{w} + \boldsymbol{b}) \quad where \quad Heavside(x) = \begin{cases} 0 & \text{if z} < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \tag{22}$$

Perceptron can only be used to classify linear data. Therefore, it does not differentiate much from other, more rudimentary, ML techniques as logistic regressions or support vector machines (SVM). To catch this non-linearity, we pill up layers of neurons (for instance, layers of Perceptrons) to form the Multilayer Perceptron (MLP) or Deep Feedforward Network. An example of them can be seen in Figure 13:

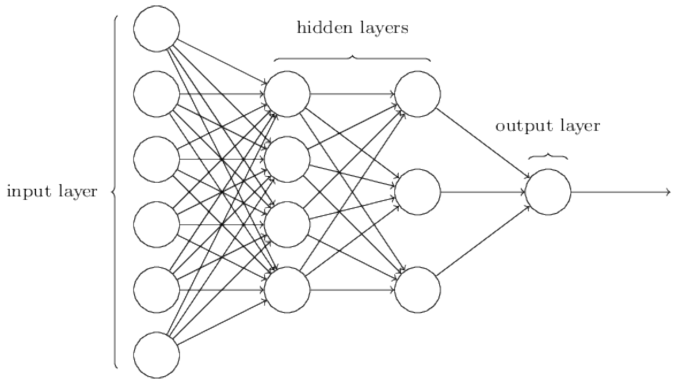

Figure 13: Ordinary architecture for MLP. Image taken from [2]

MLPs are formed by an input layer, where data is fed, one or more layers of TLU called hidden layers and an output layers. Each neuron in each layer has different weights and biases. In order to achieve our goal (multi-class classification, object identification, binary classification, etc.), this weights have to be trained. When an ANN contains many hidden layers, we talk about Deep Neural Network (DNN). Then, Deep Learning is the field of ML that studies DNN.

The process to train DNN was introduced in 1986 and it is called backpropagation. However, before diving into it, we will briefly introduce a first-order optimization technique called Gradient Descend or GD. It is nowadays one of the most used optimization technique (but not the unique one) in DL.

To explain how this algorithm works, the analogy of the mountain is used. First, suppose that you are on a mountain and your goal is to reach the bottom of it. The most logical process is to start walking in the direction of the steepest slope. This is what actually GD does, go step by step until it reaches the bottom or the local minimum. For this, it computes the gradient of the cost or loss function $J(\boldsymbol{\theta})$, the function we are trying to minimize, finding the "steepest" direction.

Examples of those cost functions are the minimum likelihood estimator and the mean

[2] `http://neuralnetworksanddeeplearning.com/chap1.html`

squared error, a special case of the former for normal distributions. However, in our case, other cost function, specific for segmentation problems, will be implemented.

Depending on the size of the step, you may spend more time to reach it (if the step is small) or never reach it (if the step is too big). This is clearly shown in Figure 14:

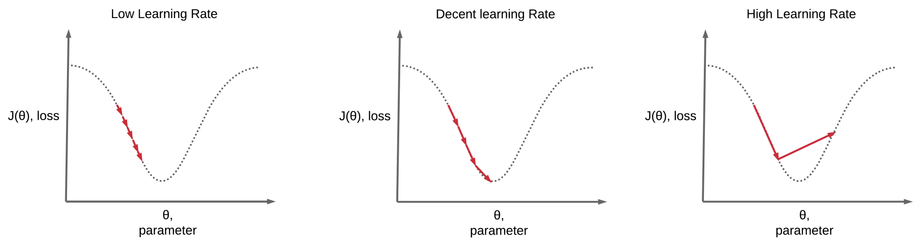

Figure 14: Step size or learning rate dependence in the optimization process. Image taken from [3]

In GD, the step is called learning rate (LR), and it is a hyperparameter of the NN that will have to be tuned. Once the gradient is computed, the parameters (weights and bias in our case) will be computed, the GD will continue forward to the next layer and the process will be repeated. We can define mathematically this as :

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla J(\boldsymbol{\theta}) \tag{23}$$

where := is the assignment operator, $\boldsymbol{\theta}$ the tensor containing both weights and bias terms and α the LR. However, depending on the dimensionality of the problem (in DL, it is in the order of magnitude of around 10^7) computing at the same time gradients for all data can be very time-consuming and we will probably never reach the global minima.

There are three main GD strategies. In the first strategy (or batch GD), the NN is feed with the whole dataset, compute the gradients and update the weights. In the second one, called SGD or stochastic gradient descent, one example is picked randomly from the dataset, iterated through the whole NN and the weights are updated. The last strategy is called mini-batch GD and it is located halfway between the previous methods. Instead of picking only one training instance, it picks a "batch" of them (usually 16, 32 or 64), iterates the NN and updates the weights. This process takes advantage of hardware optimizations, especially when using GPUs, where these operations are performed parallelly.

The first strategy is unfortunately impossible for modern datasets that are not able to fit in the RAM memory. The second nevertheless is time-consuming and due to its stochastic nature may oscillate around the minima and actually never reaching it. It is the last one the one we will choose to train our network.

In all of these three strategies, the whole dataset is introduced in the network, but it varies in how many steps the iteration is completed. In batch GD, it is completed in one

[3] https://www.deeplearningwizard.com/deep_learning/boosting_models_pytorch/lr_scheduling/

step, in SGD the steps are the number of training examples and in mini-batch GD the number of steps is given by $number\ of\ steps = \frac{number\ of\ training\ instances}{mini-batch\ size}$. Each pass of the entire dataset is called an epoch, a hyperparameter that we will have to adjust to increase performance.

Once defined GD, we introduce backpropagation. In two passes of each mini-batch thorough the network (one forward, one backward), the algorithm computes the cost function (or the error) of each neuron, finding which weight and bias should be tweaked to reduce it.

First, the algorithm introduces one mini-batch into the network. Then, the algorithm computes all the outputs of all the neurons of all layers, saving them. This is called a forward pass.

Next, the whole network's output error is computed, calculating the contribution of each output (neuron) to the error. This is done measuring how much each connection in the layer below contributes to that error (i.e. from which neuron), applying the chain rule iteratively. So, this backward pass works out the error gradient across all the connections(weights) in the layer.

Finally, the algorithm use GD to tweak all the connection weights of the network, using the error gradient just calculated, as shown in 23.

In order to work the algorithm, two things have to be taken into account. First, all the weights have to be initialized independently to different random values. If they would be initialized to the same given value, 0 for instance, the backpropagation would have the same effect in all the neurons in the same layer. Second, instead of using a step function, another activation function has to be used. A popular choice is ReLU, defined as:

$$ReLU(z) = max(0, z) \tag{24}$$

As you have already noticed, there are lots of hyperparameters to optimize, such as the number of layers, the number of neurons in each layer, the learning rate, the optimizer (here we have used gradient descent, but there are much more), batch size, the number of epochs ... However, since we will use fixed architectures, the number of layers and the neurons in each layer will not longer be a problem. Yet we will have to choose the remaining parameters.

2.4.3 Deep Convolutional Neural Networks

Despite being able to perform CV tasks with these MLPs, their accuracy is quite low when they work with more complex data. To overcome this obstacle, in 1980s convolutional neural networks were introduced, thanks to the study of the brain's visual cortex. However, the last few years, owing to the increasing amount of data and computational powers, (Deep) Convolutional Neural Networks (CNN) have improved human results in many tasks of CV, such as image classification. In this section, we will explain why these networks are so successful and we will introduce the most popular architectures.

The main building block of CNN is the convolutional layer and, as their name suggest, perform convolutional operations of two functions, i.e.:

$$\begin{cases} s(t) = (x * w)(t) = \int x(a)w(t-a)da & \text{When the functions are continious} \\ s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) & \text{When the functions are discrete} \end{cases} \tag{25}$$

In CNN, we called the first argument (usually, and in our case, an image) the input, the second is convolutional kernel (or weights, like in previous nets) and the last is the feature map (layer of neurons with the same filters, thus the same weights and bias). In the case of colour image (width × height × colour channel), this convolutional operation is performed for the (i,j) position in 3D tensor as:

$$S(i,j) = \sum_l \sum_m \sum_n I(i-m, j-n, l)K(m,n,l) \tag{26}$$

CNN are so successful because their implementation includes three key concepts for ML: sparse interactions, parameter sharing and equivariance representation. In traditional NN every output unit or neuron interacts with every input unit. However, in CNN, each neuron is not connected to each pixel of the input image (or each pixel of the previous kernel), but to pixels in their receptive fields (see Figure 15).

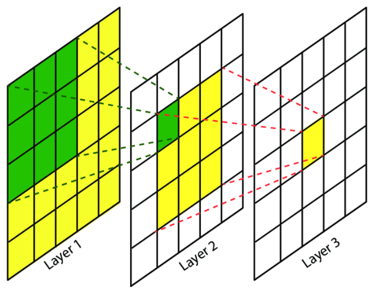

Figure 15: The four different patterns. Image taken from [4].

In the picture, the neurons of the second convolutional layer are connected only to a small rectangle (the receptive field) in the first layer and so on. This enables the network to focus on small low-level features in the first hidden layer, and to ensemble them in larger level features in the following layers. This process requires far fewer operations to calculate the output and less memory since the number of parameter decreases with the depth of the network.

However, since the second layer focuses only in a small area of the first layer, problems may arise due to the different dimensions of two connected layers. To avoid that, it is common to add zeros to the input, in a process called zero-padding.

Parameter sharing refers to the fact that each all neurons in a feature map (i.e. a layer) share the weights. This is because the work as "filters", since each layer focus on different small low level features.

[4] https://dev.to/jai00271/cnn-general-terms-and-their-meaning-42nc

This can easily be seen in Figure 16. These filters have not be defined manually, the network itself will decide which filter is more useful depending on the data we are introducing.

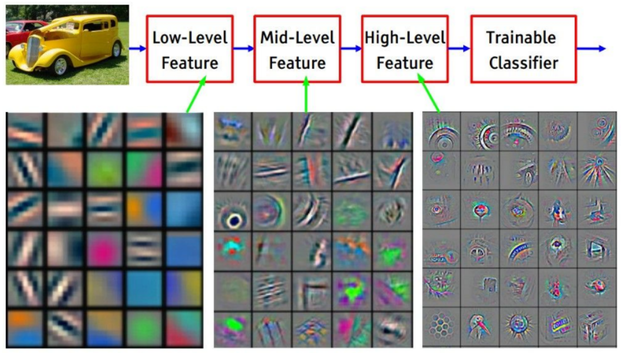

Figure 16: The role each feature map play when extracting features from images. Image taken from [18] and [5].

Lastly, equivariance or, more accurately, equivariance to translation means that if the input changes, the output changes in the same way, a key necessity in some task such as segmentation.

One other feature that makes CNN so successful is pooling. The goal of this operation is to "subsample" the input to reduce the computational load, the memory usage and the number of parameters. More formally, "a pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs" [15].

In CNN, in pooling layers, each neuron is connected to a small rectangular receptive field of neurons in the previous layer. Then, depending on the type of pooling operations, one value is extracted while the other is destroyed. In CNN, usually max pooling is used, where the maximal value within the rectangle is preserved, as seen in Figure 17. There, we are performing a max pooling on a 2×2 pooling kernel with a stride of 2 and no padding.

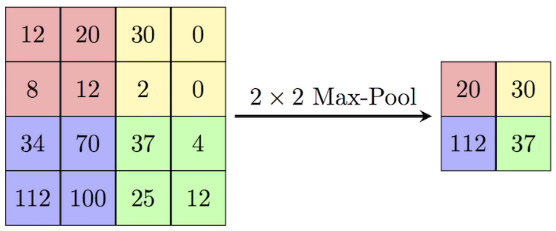

Figure 17: Max pooling operation on a 2×2 pooling kernel. Image taken from [6]

Pooling makes our network invariant to small translations, hence translating the input a little will produce no change in pooled outputs, which is useful in many applications. However, it has its disadvantages too. The destructive nature of pooling (if we pool a 3×3

[5] https://indico.cern.ch/event/510372/attachments/1245509/1840815/lecun-20160324-cern.pdf

[6] https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

area, we will lose all 9 values but 1) makes that we will necessarily lose information of the picture.

The invariant nature of the operations is not nonetheless useful for our application, segmentation. If we translate one pixel to the right, it is expected to show that translation in the output too, which is exactly the equivariance property. Despite those drawbacks, pooling is necessary since it reduces the RAM usage exponentially

In CNN typical architectures, few convolutional layers (filters) are stacked, usually 3, (after each layer generally goes a ReLU layer), then a pooling layer, then convolutional layers again and so on. The image gets then smaller and smaller (pooling layers) and deeper and deeper (convolutional layers). In the end, a regular feed-forward NN is added, composed by a small number of fully connected layers (followed by ReLUs) and a final layer to make the predictions.

This is the idea behind the most popular architectures. In our algorithm, we will use Residual Networks ResNet[19] architectures, that have four variants, with 34, 50, 101 or 152 layers, EfficientNet[20] and DenseNet[20].

The idea behind ResNet is skipping connections, as seen in Figure 18. The goal when training a NN is to model a target function $f(x)$. If you skip a connection and add directly the input to the output, you will force the model to learn $g(x) = f(x) - x$. This is called residual learning. Additionally, if many skip connections are added, the net can start making progress, even though the input has not reached them yet. The ResNet can be seen as a stack of residual unit RU, being this unit small NN with skip connection.

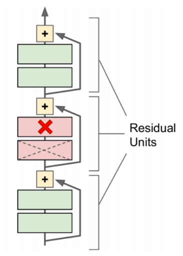

Figure 18: Deep residual network, ResNet, example. Image taken from [17].

The main problem when designing CNN is model scaling, i.e. deciding when increasing model depth will rise performance. Increasing model size has to be done with great cautious, since if available data does not increase too, we will be facing a problem of overfitting. DenseNet proposed a more thin network based on ResNet. In DenseNet, each layer obtains inputs from all preceding layers, unlike ResNet that got only from the previous one. This enables us to build thinner networks with increased accuracy.

Lastly, EfficientNet, created by Google, presented the compound scaling method, a method that ”uniformly scales the network width (number of neurons in each layer), depth (number of layers) and resolution with a set of fixed scaling coefficients”. Those coefficients were calculated empirically, after many trial-error experiments.

2.5 Segmentation of shallow cumulus convection patterns in the mesoscale

Once presented the "basics" of Deep Learning applied to computer vision, we will tackle our problem. We face a semantic segmentation problem, where we have to assign each pixel of the image to a category. In our problems, the categories are five: Flower, Gravel, Fish, Sugar and None.

For this task, we cannot use directly the architectures presented above, but a U-Net network [21]. U-Net contains two paths, a contraction (encoder) path and an expansion (decoder) path. Although the contraction path can be built "manually", stacking blocks of convolutional layers followed by ReLU layers and Pooling layers, usually one of the above-presented architectures are taken, removing the fully connected layers of the tail. In our model, we will use different encoders, in order to test which has a better performance. The main purpose of this encoder is to capture the context in the image, i.e. a set of feature maps that correctly detect mesoscale cloud organizations.

In a "normal" CNN, at the end of the network comes the fully connected network and the output layer to make predictions. In U-Net we replace this by a fully convolutional network or FCN. Here, pooling operators have been replaced by upsampling ones, to localize high-resolution features from the contracting path. Then, a successive convolutional layer will learn to assemble a more precise output based on this information. This block, build by upsampling operator layers, followed by convolutional layer and ReLU layers, are repeated roughly the same times the block of the contracting path, obtaining a U shaped network, as seen in Figure 19:

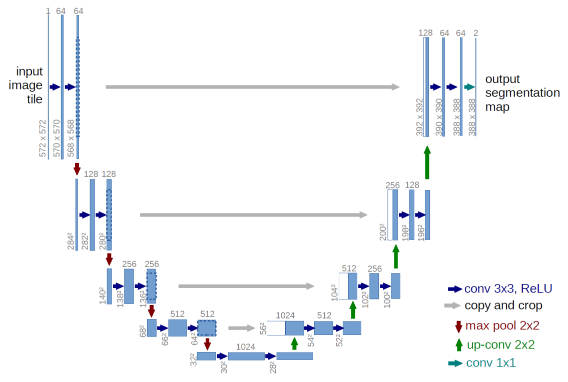

Figure 19: Sketch form U-Net architecture. Image taken from [21].

We will build our model in Python 3.7 programming language, using Google's TensorFlow 2.0 and Keras libraries. To train our network, we will use Kaggle's kernel, which provides 30 hours of free GPU usage per week. Had we trained our network without GPU, we would have spent more than 90 minutes per epoch (15 epochs, then around 22 hours each network). GPU plummets this, spending at the most 5 minutes per epoch and 120 to train the network.

We will program a Data Generator too. The main problem of current datasets, especially in CV, is that they are too big and they do not fit in computers RAM. To avoid that, we will load a batch of instances (images), perform SGD or any other optimizers used, and repeat until the NN is trained with the whole dataset (epoch). The size of the batch depends on our RAM size. The RAM of Kaggle's kernel is 16 GB, and problems raised when training very deep networks (ResNet-154) with a 16 batch, hence we will train with a batch size of 14 instances.

One of the advantages of programming a Data Generator as an input pipeline is the ease to set a data augmentation pipeline. Data augmentation techniques generate extra training instance applying transformations to current ones. Those transformations can be rotating, shifting and resizing the actual image, or even adding noise or changing pictures illumination. Data augmentation enables us to train for longer (more epochs) decreasing the risk of overfitting of NN. For this task, we will use the Albumentations library [7].

All augmentation techniques nevertheless are not suited for semantic segmentation. For instance, destructive transformations, such as "Crop" or "CenterCrop", may ensue in losing a part of a pattern, which can plunge accuracy. Neither we can use transformations that change image size (like transposing) because the networks expect images of a certain size (this is we preprocess first our images). In our work, we will use "HorizontalFlip", "VerticalFlip" and "ShiftScaleRotate".

Our dataset consists on 5546 labelled train images (around 3.5 GB of images) and 3698 test ones (around 2.3 GB). We will divide the train set into two different and independent sets, one for training (the 80% of the data) and other for validating purposes (the remaining 20%). If we explore our data, we obtain the histograms of Figure 20:

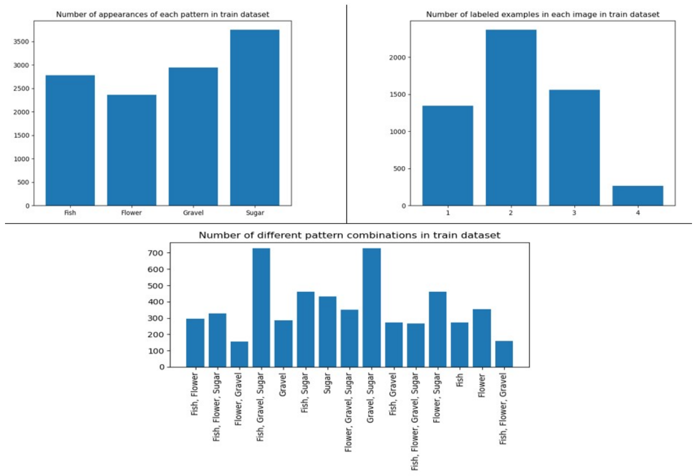

Figure 20: Histogram showing each patterns' appearance frequency, the number of labels in each image and the combination frecuency for patterns. Image produced by author's algorithm

[7] https://albumentations.ai/

Thus, the most often pattern is Sugar, in the majority of images there are between 1 and 3 cloud patterns (being two patterns per image the most numerous one) and the two more usual combinations of patterns are both "Fish", "Gravel" and "Sugar" and "Gravel" and "Sugar". Now, in Figure 21, we will explore 4 randomly chosen images:

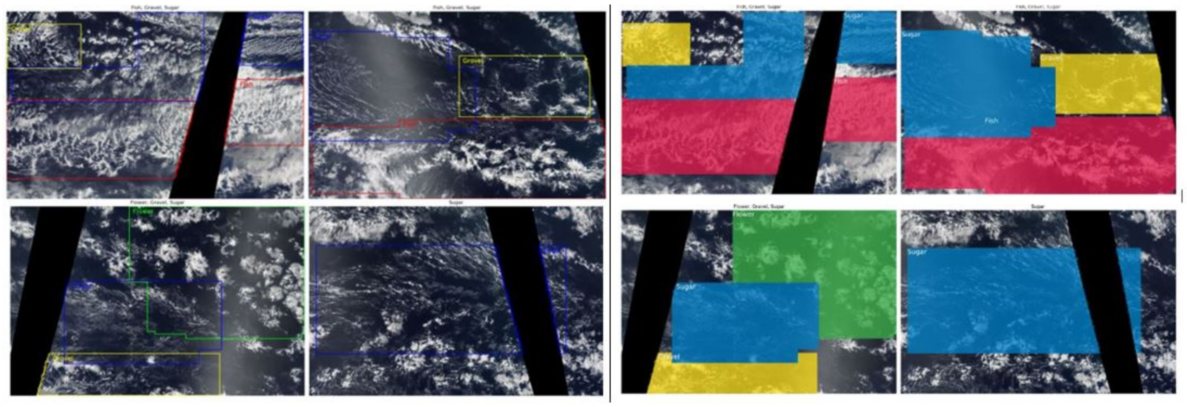

Figure 21: Segmentation map and identification of four randomly taken images. Images produced by author's algorithm.

It can be noticed that some times patterns overlap. After getting a grasp of the data we will feed our NN, we will start building, training and testing it.

2.5.1 Network architecture and metrics

As said before, we will use U-Net with different encoders (or backbones). Instead of building them "by hand" in TensorFlow stacking Convolutional Layers, Upsampling layers, pooling layers ... we will use a Python library that has already defined those models [8]. Different encoder required different input shapes, so we will have to preprocess our images to enable the network to load them. However, it provides its own preprocessing tools. Therefore it is a quite handy library that will considerably ease our work.

Next, we will load pre-trained models for each encoder. This means that our encoder (ResNet, EfficientNet ...) has already been trained in a very large benchmark dataset (usually, this dataset is ImageNet, which contains more than 14 million images). This is called Transfer Learning (TL) and it has been proved that in certain applications improves classification (in our case segmentation) accuracy, compared to randomly initialized weights, in both the first epoch and the last one.

Since the dataset in which the encoder has been trained and our dataset are not very similar (in our dataset only clouds can be found, while in ImageNet there are more than 20,000 classes ranging from dogs or cats to aeroplanes and ships) and our dataset it is not

[8] https://github.com/qubvel/segmentation_models

small, we will have to train again our whole model, which, depending on the computer, may take up to 12 hours.

In TL, we have to be very careful when choosing the learning rate, since very large LR leaps the risk of losing previous knowledge. However, making the LR small is not a problem, and it will only improve the accuracy of our net. However, the smaller the LR, the more time we will spend training our network and a trade-off between accuracy and training time should be reached.

For certain optimizers, we will use a function that changes dynamically our LR. This function, that change parameters of our network during training are called callbacks. We will implement two more callbacks, one to stop the training if the loss functions do not fall for a certain number (in our case 5) epochs, called EarlyStopping, and other to restore the weights to the epoch with smaller loss function, called ModelCheckpoint, which will prevent the model from overfitting. Therefore, the number of epochs will not be longer a critical hyperparameter to optimize.

Before starting the training, we will set out model metrics. Since we have not labelled test images, our result has to be upload to the Kaggle's competition website to be evaluated there. Hence, we have to use the metrics proposed by them. It is the Sørenson-Dice coefficient [22], defined as:

$$DSC = \frac{2|X \cap Y|}{|X| + |Y|} \tag{27}$$

where X is the predicted set of pixels and Y is the ground truth. However, we will use Intersection over Union IoU or Jaccard index metric too for validation purposes, defined as:

$$IoU = \frac{|X \cap Y|}{|X \cup Y|} \tag{28}$$

For both metrics, the bigger, the more accurate. Defined the metrics, we start our training.

2.5.2 Training and testing

In this subsection, we will first present the optimizers, apart from SGD, we will use for our training. Then, strategies to tune the learning rate dynamically will be presented. Following this, we will present the performance results of our algorithm. Lastly, the conclusion raised by the application of the similar algorithm developed by [3] to the globe will be presented.

SGD method presented above, combined with backpropagation, is the standard algorithm for training NN. Yet the main problem of SGD is that it spends a lot of time to converge, even if it reaches a better convergence rate than other optimizers. To leap the training speed we will add momentum optimization method and Nesterov Accelerated Gradient (NAG) to our vanilla SGD.

In momentum optimization, we take into account previous gradients. Instead of updating the parameter directly, we subtract the local gradient from the momentum vector $\boldsymbol{m}$, to add it to the current parameter. To prevent the momentum from growing too much, we add a kind of "friction" β, a hyperparameter called momentum. It ranges between 0 (high friction) and 1 (no friction), being the typical value and the one chosen by us $\beta = 0.9$. Mathematically, it can be defined as:

$$\boldsymbol{m} := \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{29}$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \boldsymbol{m} \tag{30}$$

where $\boldsymbol{\theta}$ represents both the weights and the bias.

The other method, NAG, (which is usually combined with momentum) measures the gradient, not at the local position (i.e. for the local weights), but slightly ahead in the direction of the momentum, such as:

$$\boldsymbol{m} := \beta \boldsymbol{m} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta} + \beta \boldsymbol{m}) \tag{31}$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \boldsymbol{m} \tag{32}$$

This works because usually gradient points in the right direction, and thus will be a bit more accurate to measure the gradient a bit further than the actual position.

Besides those methods based on SGD, we will be used another optimization method, adaptative moment estimation or Adam. This method combines the advantages of RMSProp and SGD with momentum.

Adaptative learning rates methods rely on separating learning rate for each parameter and then adapt automatically those LR during learning. Thus AdaGrad, Equation 33, automatically and "individually adapts LR for all parameters of the models scaling them inversely proportional to the square root of the sum of all historic squared values of the gradients"[15]. Due to this, the learning rate of parameters with a larger partial derivative of the cost function will decrease faster than the ones with a small one, decreasing training time. This can be seen in Equation 33:

$$\boldsymbol{s} := \boldsymbol{s} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{33}$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \varepsilon} \tag{34}$$

where $\otimes$ is element-wise multiplication, $\oslash$ element-wise division and ε is set to a very small value (usually 10^{-10}) to avoid divisions by zero.

However, AdaGrad accumulates the squared of all gradients since the beginning of the training, which may result in a premature and excessive dip in the effective LR. This is changed by RMSProp which discards extremely old gradients with an exponentially decaying average, as shown in Equation 35, where β takes into account the decay.

$$\boldsymbol{s} := \beta \boldsymbol{s} + (1 + \beta) \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{35}$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\boldsymbol{s} + \varepsilon} \tag{36}$$

Lastly, Adam combines both SGD with momentum and RMSProp to obtain:

$$\boldsymbol{m} := \beta_1 \boldsymbol{m} - (1 - \beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{37}$$

$$\boldsymbol{s} := \beta_2 \boldsymbol{s} + (1 + \beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{38}$$

$$\hat{\boldsymbol{m}} := \frac{\boldsymbol{m}}{1 - \beta_1^t} \tag{39}$$

$$\hat{\boldsymbol{s}} := \frac{\boldsymbol{s}}{1 - \beta_2^t} \tag{40}$$

$$\boldsymbol{\theta} := \boldsymbol{\theta} + \eta \hat{\boldsymbol{m}} \oslash \sqrt{\hat{\boldsymbol{s}} + \varepsilon} \tag{41}$$

where t represents the iteration number. The algorithms presented above soar both the precision and the learning rate, compared with RMSProp and AdaGrad. However, as it will be seen later, the NNs optimized by SGD get better result than the ones based on Adam, despite being slower.

Apart from choosing the right optimizers, learning rate schedule may help to dip convergence time and to soar convergence rate. This technique could be used in both SGD and Adam. Even though Adam set different LR for different parameters, varying the global LR can help too.

Instead of choosing between a small one that will spend much time until converge, and a very big one, that may damage weights from TL, learning rate scheduling chooses a learning rate that changes with the time. Despite the existence of many learning rate scheduling approaches, we will choose 1cycle scheduling.

This algorithm operates in two different phases. In the first phase, called cycle phase, the LR increases from the initial LR to a final one (both set by the users). This takes around $30-40\%$ per cent of the training. Then, the decay phase comes, and the LR starts to decay until the end of the training, reaching smaller values than the initial one. It was shown to enhance both training time and accuracy in certain experiments.

Once presented the different optimizers, SGD with momentum and NAG and Adam, we will use during our training and a technique to change LR during training, we will start the actual training of our Deep Learning-based model.

We will use ResNet(ResNet-34 and ResNet-152), EfficientNet(EfficientNet-B2) and DenseNet(DenseNet-121) backbones (No free lunch theorem[9]). Besides that, before posting our result for submission to Kaggle, we have post-process them. When labelling in the original photos, one of the requirements for labellers was that the minimum size of the label

[9]In this theorem, proved by [23] in that paper, is assumed that there is no reason to prefer one model over other, if you make no assumption about the data, i.e. there is no model that will work better for all data and therefore all have to be evaluated. Due to the restricted amount of computation power and time, we will only test some of the possible models.

was 10% of the whole image. The results for SGD-based optimizer can be seen in Table 1 and in Table 2 the ones for Adam:

	Post.	**No Post.**	**Epochs**	**Train.(min)**	**Algorithm(min)**
ResNet-34	0.656	0.625	14	81	111
ResNet-152	0.657	0.627	16	152	180
EfficientNet-B2	0.654	0.615	18	122	153
DenseNet-121	0.653	0.624	12	71	105

Table 1: Post processing (Post.) accuracy, Accuracy without post processing (No Post.), number of epochs, training time for the NN (train.) and whole execution time of the algorithm (Algorithm) for **SGD optimizer**.

	Post.	**No Post.**	**Epochs**	**Train.(min)**	**Algorithm(min)**
ResNet-34	0.639	0.601	16	111	140
ResNet-152	0.636	0.587	16	121	160
EfficientNet-B2	0.646	0.596	16	105	139
DenseNet-121	0.489	0.367	16	60	101

Table 2: Post processing (Post.) accuracy, Accuracy without post processing (No Post.), number of epochs, training time for the NN (train.) and whole execution time of the algorithm (Algorithm) for **Adam optimizer**.

The reason beneath the quickness in training in DenseNet-121 is that in both cases, SGD and Adam, triggered EarlyStopping and ModelCheckpoint callbacks, in order to avoid overfitting. It is easy to stop that SGD training outperformed in all cases Adam, despite being a little bit slower. In the following Figure 22 we can see how the curves for the loss function, the IoU and DSC fluctuated during the training (for SGD ResNet-34).

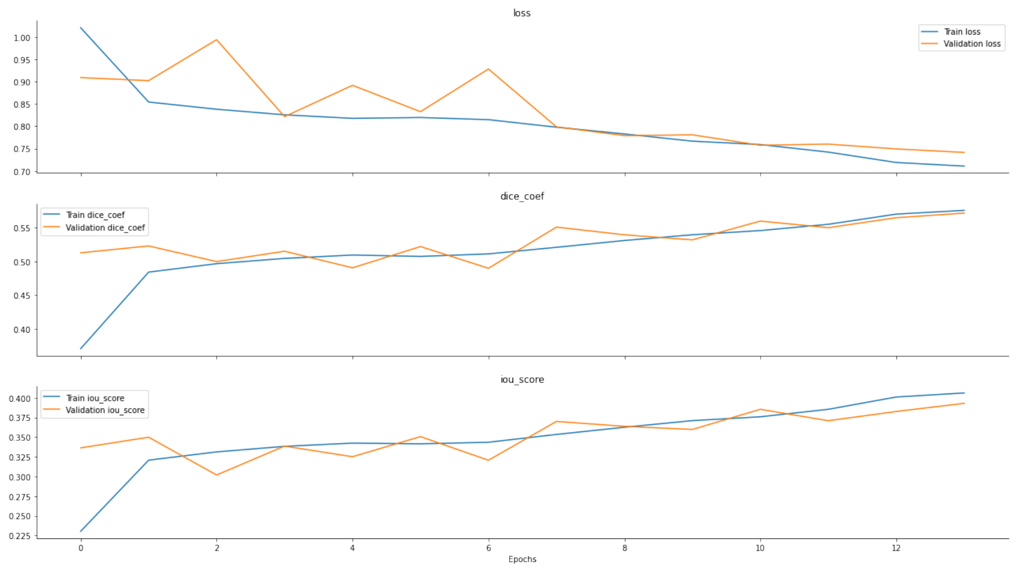

Figure 22: Loss, Dice Coefficient and IoU values' fluctuation through training. Image produced by author's algorithm.

The advantages of DL algorithm is that they are faster than human labelling, more accurate and it can be automatized. Thus [3] applied their algorithm globally for the entire year 2017, as seen below in Figure 23. From the study of heatmaps' hotspots, some question raised by [2] can be answered.

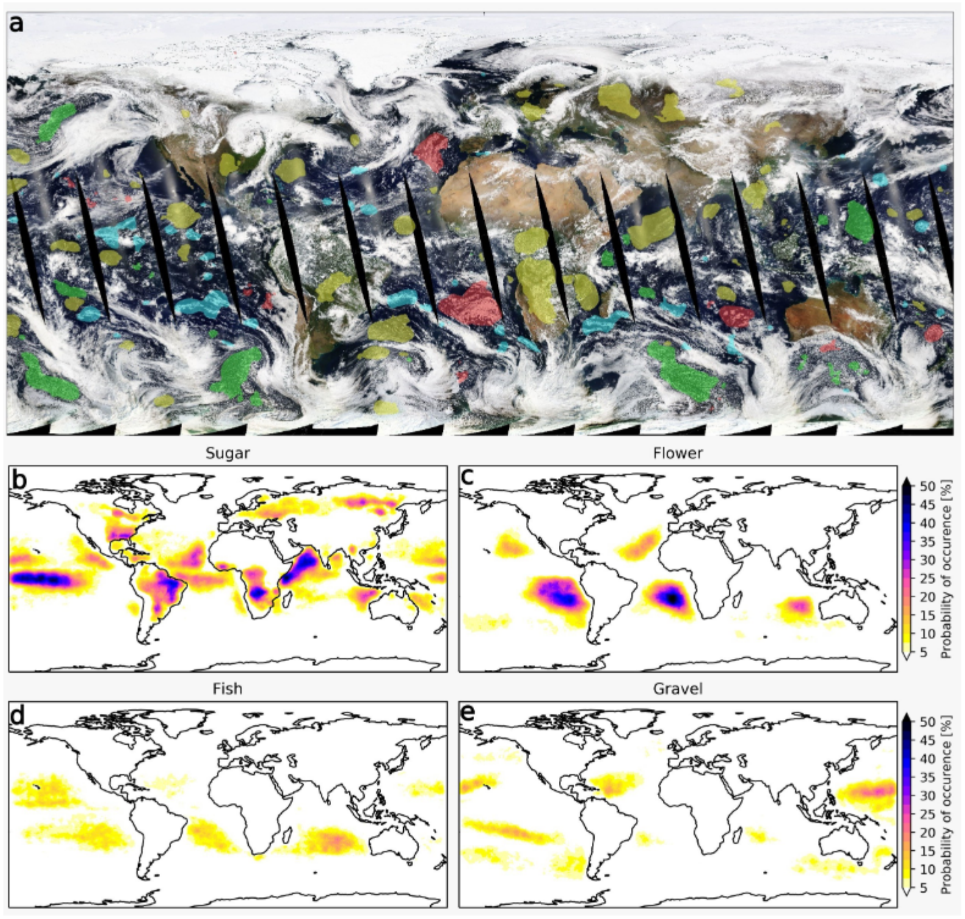

Figure 23: (a) Predictions from the [3] algorithm in May 1, 2017 and (b) Heatmaps of the four patterns for the whole 2017. Image taken from [3].

First, Sugar (the only pattern without mesoscale organization) is the only pattern that can be seen over land. However, since the algorithm was not trained with land images, the algorithm may have some bias, hence this result must be dealt with caution. Sugar was too the most common pattern, followed by Flower. Since large Sugar patterns are rare near Barbados, [2] results are partially biased, due to their study was confined determined geographical area.

Prevalence of sugar in trades adjacent to deep tropics, as well as in regions such as Arabian see are consistent with 11, i.e. strong low-level subsidence and drier cloud layer, which may indicate that they are favoured in regions where, instead of convection, strong subsidence from neighbouring regions of active convection can be found.

Flower patterns are consistent too with 11, as they are associated with more pronounced lower tropospheric stability and drier free troposphere.

Further downstream in the trade winds, Gravel and Fish can be found. These patterns are more geographically intertwined, as seen in Figure 23, and there is no strong hotspot of them anywhere in particular, which agrees with 11.

The fact that Gravel is confined to the Barbados regions, the west of Hawaii and the southern Pacific near regions can be the reason why [2] results claim that this pattern is prevalent in the trade-winds. However, as seen before, the Barbados area is not representative of the whole trade-winds region and their result should not be directly extrapolated to it.

2.6 Alternatives to human chosen labels

Classifying cloud patterns based on subjectively chosen patterns may result in losing information. For instance, between Fish and Gravel other patterns, not so clear to be distinguished by bare human eye, can exist. [24] proposed the first unsupervised NN to autonomously discover cloud organizations patterns from satellite images in shallow convective clouds in trade winds.

Unsupervised machine learning techniques, unlike supervised ones that as the one we have used before, do not need labelled data to work. Indeed they cluster images with similar structures. The authors used a ResNet-34 NN, but by the time I am writing these lines (2020/08/18) the code is not available yet.

To train the network, they introduced a triplet of image tiles, called anchor, neighbour and distant tile. Anchor and neighbour tiles are taken from the same image, displacing the former half tiles width in a random direction, so the overlap is around the 50%. The third tile is taken from another image.

They trained the network with $10,000$ triplets taken from Tropical Atlantic ($60°W$ to $20°W$ and $10°N$ to $20°N$) during boreal winter. They perform hierarchical clustering, where points are merged into a progressively larger cluster. In each merge, two clusters which minimizes a given metric, Ward Metric[25], are combined. The study is performed in the last 12 merges. In those 12 merges, an spatial structure organization and radiative properties study was performed.

The results of the study were that, despite the large degree of overlap (in both spatial structure and radiative properties), the mean radiative properties and clusters mean were clearly separated. They were promising, not only because the model was able to identify and separate different cloud structures, but also generated a representation (hierarchical clustering using Ward metric) where similarities between structures cloud are studied.

3 Conclusions

In the present work deep learning techniques have been used to try to identify and segment different mesoscale shallow cumulus convection patterns. After briefly introducing the physics of shallow cumulus convection and how are they formed, we analyse the work done by the ISSI team [2].

They proposed four brand new patterns for SCC seen in satellite images, solely based in visual features. However, they successfully classified those images using robust metrics, rather than pure visual and subjective classification, such as mean object size and organization index I_{org}.

Apart from visual features, they discriminated the patterns based in two physical variables of the clouds, such as surface wind V_s and inversion strength EIS, and studied their variations both in daily and annual basis. Later, they confirmed that the more the cloud coverage, the less the Net CRE. In their study of changes of the two physical characteristics, they discovered that EIS was increasing due to global warming. Since EIS is related to Fish and Flowers, global warming may produce increasing low-level cloud coverage and hence decreasing net radiative budget, which may cool the Earth.

Despite the promising results of that work, it was physically confined to windward of Barbados. In order to solve that, [3] analysed and labelled images from other parts of the Pacific Ocean, with similar climatological characteristics as the Barbados one. They discovered that in all three regions, the changes in temperature, relative humidity and vertical velocity for with altitude for each pattern were noticeable.

To be able to classify SCC all over the globe, they built a DL algorithm, which would classify those patterns from the labelled data. We follow their path and built our own DL algorithm too.

After introducing the main techniques for DL, we discovered that optimizers based on Adam were considerably faster on average than the ones based on SGD, despite having a worse convergence rate. Thus, the advantages of using SGD or Adam should be weight up.

Both algorithms reached nevertheless very acceptable results, around 0.65, close to the one obtained by the winner of the competition, 0.67. They used an ensemble of neural networks, which reduces generalization error by combining several models in a technique called bootstrap aggregating.

The reason this ensemble model works is that usually different models do not make the same error on the test set. At least, the ensemble will perform as well as any member of it if the errors are perfectly correlated, and enhancing the global performance if not. This can be expressed in other, more mathematical, words.

First, we will suppose a multivariate normal distribution with zero mean ($\mu_{i,j} = 0$). If $\mathbb{E}[x^n]$ is the nth algebraic momentum of x, defined as $\mathbb{E}[x^n] = \sum x_i^n f(x_i)$, $f(x)$ is the probability distribution function (PDF) of x. Remember that the case n = 1 is zero.

Once clarified that, we define the second momentum or variance as $\mathbb{E}[\epsilon_i^2] = v$, covariance $\mathbb{E}[\epsilon_i \epsilon_j] = c$ and the averaged-error of the ensemble (assuming independent events) $\frac{1}{k}\sum_i \epsilon_i$. Thus, the expected mean squared error of the ensemble is:

$$\mathbb{E}\left[\left(\frac{1}{k}\sum_i \epsilon_i\right)^2\right] = \frac{1}{k}\,\mathbb{E}\left[\sum_i \left(\epsilon_i + \sum_{j\neq i} \epsilon_i \epsilon_j\right)\right] = \frac{1}{k}\left(v + (k-1)c\right) \tag{42}$$

If the correlation coefficient, defined as $\rho_{ij} = \frac{c}{v}$ is one, i.e. are perfectly correlated, the MSE reduced to v, which was the error of the single model. However, if they are perfectly uncorrelated, the expected squared error will be, $\frac{v}{k}$, and hence the bigger the ensemble, the smaller the MSE.

Building different models is straightforward. In our model, for instance, many different parameters can be chosen, such as the backbone of U-Net, the learning rate, the number of epochs. Even variations of U-Net, such as U-Net++, or other semantic segmentation networks could be used.

This model averaging is an extremely handy and powerful tool to reduce generalization error. Indeed, machine learning contest, such as the one the data are using, are won by this kind of models.

Due to our computational restrictions imposed by Kaggle, this model was out of our current scope, because only 9 hours of GPU training were available. The only way to overcome that time restriction would be using Microsoft's Azure or Amazon Web Service (AWS), which provided unlimited GPU usage after paying. The winner of the contest receive monetary awards, thus investing in those services is usually a good idea.

Other options would be to restructure the whole algorithm, including U-Net ant its backbones, to train it using Google's Tensor Processing Units or TPUs, very fast machines build only for Deep Learning applications. However, this would be an incredibly time-consuming transform and again out of the scope of the current work. Besides that, the data input pipelines provided by TensorFlow, necessary if using TPU, are not yet well prepared for multi-class semantic segmentation problems.

Lastly, an unsupervised neural network to cluster cloud images was proposed. The introduction of hierarchical clustering of cloud patterns unveils numerous applications that can incredibly increase our current knowledge of mesoscale organization of shallow cumuli because we could tweak our segmentation algorithm to work with different merges of the unsupervised cluster algorithm in the entire planet for a certain amount of time (since it is automatized and requires very little human supervision, we could run it since registers exist) to check if any physical variable changes considerably, as it seems according to [2].

The code is available attached, in authors' GitHub[10] and in authors' Kaggle Kernel

[10] https://github.com/ruizeneko/fisica_TFG

(or Notebook) [11]. However, it anyone want to test the correct running of the program, I encourage to use Kaggle's Kernel, in order to take advantage of its GPU accelerator.

[11] https://www.kaggle.com/ruizeneko/eneko-tfg

References

[1] Bjorn Stevens, Sandrine Bony, and et al. Brogniez. **Sugar, gravel, fish and flowers: Mesoscale cloud patterns in the trade winds**. *Quarterly Journal of the Royal Meteorological Society*, 146(726):141–152, 2020.

[2] Sandrine Bony, Hauke Schulz, Jessica Vial, and Bjorn Stevens. **Sugar, Gravel, Fish, and Flowers: Dependence of Mesoscale Patterns of Trade-Wind Clouds on Environmental Conditions**. *Geophysical Research Letters*, 47(7):e2019GL085988, 2020.

[3] Stephan Rasp, Hauke Schulz, Sandrine Bony, and Bjorn Stevens. **Combining crowd-sourcing and deep learning to explore the meso-scale organization of shallow convection**, 2019.

[4] The Editors of Encyclopaedia Britannica. **Cloud**. https://www.britannica.com/science/cloud-meteorology, 2020.

[5] Wikipedia. **Cumulus clouds**. https://en.wikipedia.org/wiki/Cumulus_cloud, 2010.

[6] Viegas D.X. Wyngaard J.C. (eds) late E.J., Fedorovich E.E. ***Buoyant Convection in Geophysical Flows***. Springer, Dordrecht, 1998. https://doi-org.ehu.idm.oclc.org/10.1007/978-94-011-5058-3_19.

[7] Dick P Dee, S M Uppala, AJ Simmons, Paul Berrisford, P Poli, S Kobayashi, U Andrae, MA Balmaseda, G Balsamo, d P Bauer, et al. **The ERA-Interim reanalysis: Configuration and performance of the data assimilation system**. *Quarterly Journal of the royal meteorological society*, 137(656):553–597, 2011.

[8] Bruce A. Wielicki, Bruce R. Barkstrom, Edwin F. Harrison, III Lee, Robert B., G. Louis Smith, and John E. Cooper. **Clouds and the Earth's Radiant Energy System (CERES): An Earth Observing System Experiment**. *Bulletin of the American Meteorological Society*, 77(5):853–868, 05 1996.

[9] Jean-Louis Dufresne, Marie-Alice Foujols, and month = 02 pages = 2123–2165 title = "**Climate change projections using the IPSL-CM5 earth system model: from CMIP3 to CMIP5**" volume = 40 journal = Climate Dynamics doi = 10.1007/s00382-012-1636-1 Denvil, et al. year = 2013.

[10] A Bagoly, A Bevan, A Carnes, Sergei Gleyzer, L Moneta, A Moudgil, Simon Pfreundschuh, T Stevenson, S Wunsch, and O Zapata. **Machine Learning Developments in ROOT**. *Journal of Physics: Conference Series*, 898:072046, 10 2017.

[11] Evan Racah, Christopher Beckham, Tegan Maharaj, Samira Ebrahimi Kahou, Prabhat, and Christopher Pal. **ExtremeWeather: A large-scale climate dataset for semi-supervised detection, localization, and understanding of extreme weather events**, 2016.

[12] Yunjie Liu, Evan Racah, Prabhat, Joaquin Correa, Amir Khosrowshahi, David Lavers, Kenneth Kunkel, Michael Wehner, and William Collins. **Application of Deep Convolutional Neural Networks for Detecting Extreme Weather in Climate Datasets**, 2016.

[13] Robert Wood and Dennis L. Hartmann. **Spatial Variability of Liquid Water Path in Marine Low Cloud: The Importance of Mesoscale Cellular Convection**. *Journal of Climate*, 19(9):1748–1764, 05 2006.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **ImageNet Classification with Deep Convolutional Neural Networks**. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. ***Deep Learning***. MIT Press, 2016. http://www.deeplearningbook.org.

[16] Kaggle. **Understanding clouds from satellite images**. https://www.kaggle.com/c/understanding_cloud_organization, 2020.

[17] Aurlien Gron. ***Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems***. O'Reilly Media, Inc., 2nd edition, 2019.

[18] Matthew D Zeiler and Rob Fergus. **Visualizing and Understanding Convolutional Networks**, 2013.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep Residual Learning for Image Recognition**. *CoRR*, abs/1512.03385, 2015.

[20] Mingxing Tan and Quoc V. Le. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks**. *CoRR*, abs/1905.11946, 2019.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. **U-Net: Convolutional Networks for Biomedical Image Segmentation**, 2015.

[22] Lee R. Dice. **Measures of the Amount of Ecologic Association Between Species**. *Ecology*, 26(3):297–302, 1945.

[23] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[24] L. Denby. **Discovering the Importance of Mesoscale Cloud Organization Through Unsupervised Classification**. *Geophysical Research Letters*, 47(1):e2019GL085190, 2020. e2019GL085190 10.1029/2019GL085190.

[25] Joe H. Ward Jr. **Hierarchical Grouping to Optimize an Objective Function**. *Journal of the American Statistical Association*, 58(301):236–244, 1963.