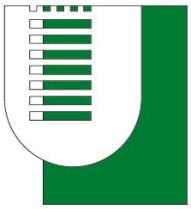


TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Big Data & Physics of Complex Systems

AUTOMATIC SOLAR FLARE DETECTION USING A CONVOLUTIONAL NEURAL NETWORK

written by:
Piero Paialunga

Year 2020/2021

Contents

Abstract

Solar flares detection is a center role challenge for Earth communication and space missions. Several automatic solar flare detection techniques have been built in the past, mostly using image pre-processing techniques to make the analysis process cheaper [4][15]. The approach of this work is different as it applies convolutional neural network directly to the raw pixels to classify the input image, without building any hand designed feature. A first classification has been applied to distinguish whether or not the Sun has active regions. A final classification has then been performed to distinguish active regions and flare state regions. An high performance value (accuracy) has been reached for both the classification algorithms($\approx 96\%$)

1 Introduction

Solar flares are sudden flashes of brightness on the Sun. These events are accompanied by an energy release (from 10^{20} J to 10^{25} J). When a solar flare appears, its radiation cannot pass human atmosphere and directly affect humans on Earth. Nonetheless, solar flares can disturb the atmosphere where GPSes and communication signals travel [12].
Moreover, solar flares are usually observed in correspondence of sunspots, that are crucial events in considering Solar Cycle and Solar activity.
Lastly, sometimes their disappearance is linked with the appearance of large **coronal mass ejections** that can disturb terrestrial communication, power systems and space missions [11]. All these phenomena clearly express the necessity of solar flare periodical observation and detection.

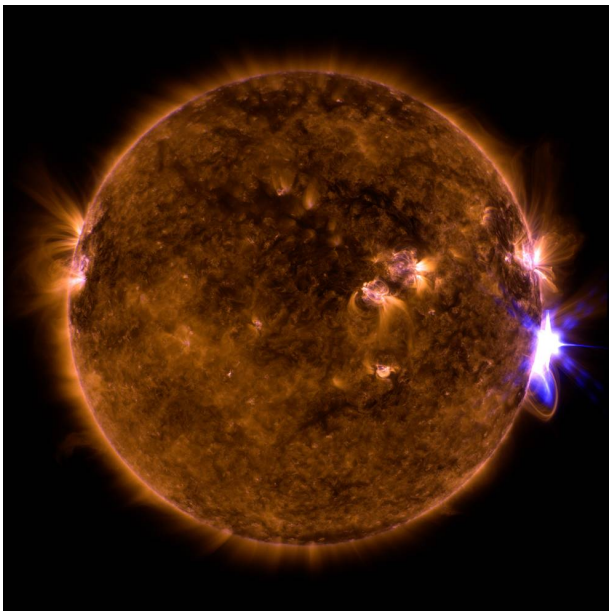


Figure 1.1: **Sun in a Solar flare state**: NASA's Solar Dynamics Observatory captured this image of a solar flare – as seen in the bright flash on the right side – on Sept. 10, 2017. The image shows a combination of wavelengths of extreme ultraviolet light that highlights the extremely hot material in flares, which has then been colorized.[12]

2 Machine Learning theoretical background

Unequivocally, the possibility of automatizing the entire process seems appealing. **In particular, it is useful to build a tool that, given an image in input, is able to distinguish if the sun is or is not in a solar flare state**.

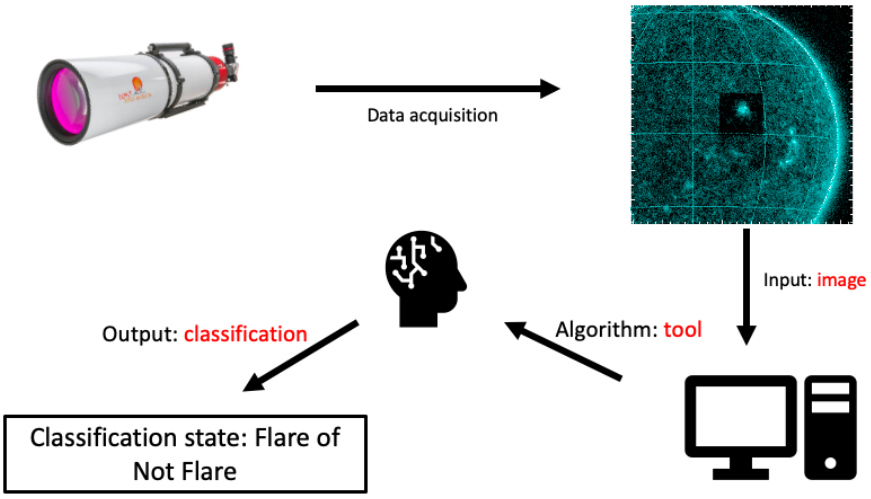


Figure 2.1: **Process**: Brief scheme that explains the process that the tool needs to execute.

Following this idea, a powerful way to execute it is by using a **machine learning approach**.

2.1 An overview

Machine Learning is the study of algorithms that are able to teach a machine to execute a particular task without being explicity programmed to do so. **A machine learning process starts with data that are feed to the computer**. These data will be, where appropriate, pre-processed and taken to the learning algorithm, that will use the data to learn how to execute its task efficiently. When the learning process is completed, the software is ready to take a new unknown input and to output its prediction.

In our specific case, two different algorithms have to be implemented, with two different tasks:

- **First algorithm**: Distinguish whether or not the Sun has active regions
- **Second Algorithm**: Distinguish whether or not the Active Region has solar flares

In our specific case, each single data is accompanied by a desired output value known as **target**:

- 1 if the data is a **Magnetically Active** (First Algorithm)/**Flare State data** (Second Algorithm)
- 0 if the data is a **Non Magnetically Active** (First Algorithm)/**Non Flare State** data (Second Algorithm)

The algorithms like the one described above use labeled data and are known as **supervised learning algorithms**. In a supervised learning algorithm data are usually split in two different section:

- **Training set**: Usually 70%-90% of the data, with the target values. The algorithm learns the best way to assign targets to data.
- **Test set**: Usually 30%-10% of the data. Targets values are now unknown to the algorithm. The algorithm uses its best parameters (obtained after the training) to predict the targets of the test set data. The predictions are eventually compared with the targets to get the performance index.

In our specific task, due to the small dataset used, **K-fold cross validation has been used**. In this approach, the training and test set rotate iteratively K times in order to test the optimum values of the algorithms on the whole dataset, thus getting more robust results. The resulting **accuracy** (i.e. the ratio between the good classification and the total element of a class) will then become a **mean accuracy** between the K accuracy obtained.

Moreover, it is important to notice whether or not the dataset we are considering is acceptably balanced. For example if we are considering a binary classification task, it is important to notice if both the classes we want to distinguish are almost equally present in the dataset (e.g. 50% of ones and 50% of zeros, 60% of ones and 40% of zeros...)

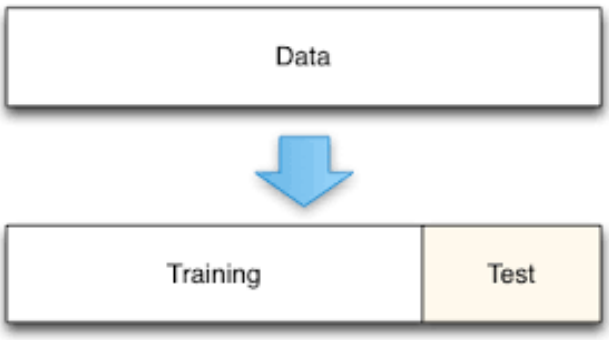


Figure 2.2: **Train-test split**: In the training set the algorithm tries to learn the best way to assign targets to data. The algorithm uses its best parameters (obtained after the training) to predict the targets of the test set data. The predictions are eventually compared with the targets.[5]

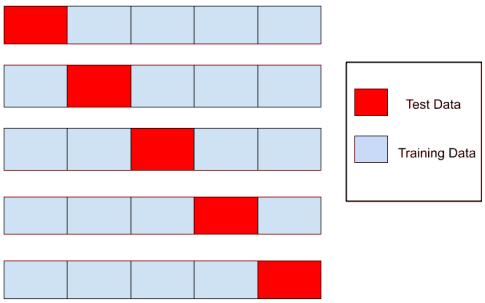


Figure 2.3: **K-fold cross validation**: An illustrative explanation of how K-fold cross validation works. The training set and the test set rotates, so that the algorithm can be tested on the whole dataset and the results become more robust .[20]

2.2 Traditional Machine Learning vs Deep Learning

Depending on the usage of data and the approach that is applied to them, two different kind of Machine Learning algorithms can be indentified:

- **Traditional Machine Learning**
- **Deep Learning**

Traditional machine learning processes tend not to use raw data as input, but they start building hand-designed features for each example in the dataset. For example, let's pretend we want to apply a Machine Learning

Traditional Algorithm to classify whether or not a movie is an horror movie. A traditional machine learning process will start by extracting hand designed features like the director, the actors, the production company and so on and so forth.

The main problem with this approach is that the performance of our algorithm dramatically depends on how the features have been designed. It is then crucial to **properly design the features** and this may require specific domain competences. Moreover sometimes the task is simply too complex to design these features.
For example let's consider a machine learning eye detection in a human face image. We can try to design specific features to represent the eyes, but the variability of the data (e.g. the brightness of the sun, the position of the eye in the image or the shape of the eye itself) makes the hand-designing of these features extremely complicated and probably inefficient.

A more modern idea is to imprint the process of understanding in terms of **hierarchy of concepts** [7]. Therefore, the computer will start by learning simple concepts, and will manage to learn more complex concepts building them from the simpler ones. This approach does not require any hand designed features of the data and the learning process can be seen as a deep, layered, structure. This is the reason why this approach is known as **deep learning**. A deep learning model is also known as a **multilayer network model** as the learning process is defined by a sequence of transformation on the input data. These sequence of transformation of the data require an higher number of parameters than the traditional machine learning techniques. It is thus necessary to use an high number of data to avoid overfitting, and apply regularization techniques like the K-fold cross validation technique described above.

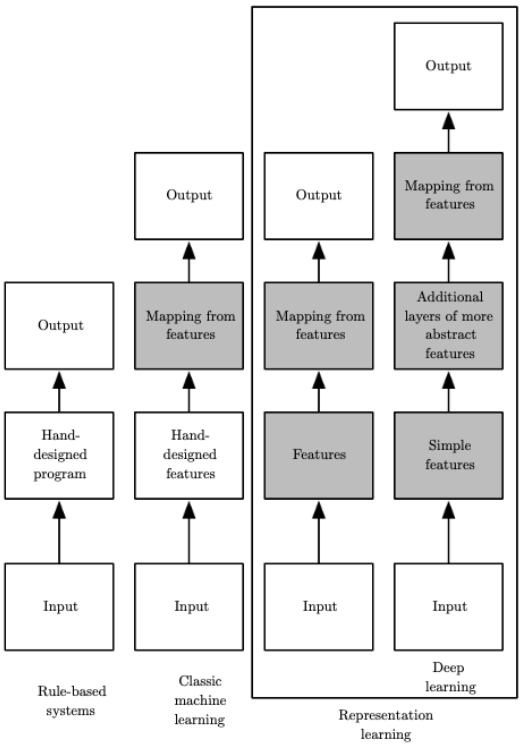


Figure 2.4: **Deep Learning vs Traditional (Classic) Machine Learning**: This scheme shows the difference between a Traditional (Classic) Machine Learning Approach, where features are hand designed by the programmer and Deep Learning, where features are obtained by the learning structure itself.[7]

2.3 Feed Forward Neural Network

As it has already been said, neural networks use a layered structure to process the input and obtain the final result. If we consider a d-dimensional input vector $x = (x_1, x_2, ..., x_d)$ the first layer derives $m_1 > 0$ activations $a_1^{(1)}, a_2^{(1)}, ..., a_{m_1}^{(1)}$ through suitable linear combination of $x_1, x_2, ..., x_d$:

$$a_j^1 = \sum_{i=1}^{d} w_{j,i}^{(1)} x_i + w_{j,0}^{(1)} = \bar{\boldsymbol{x}}^{\boldsymbol{T}} \boldsymbol{w_j} \tag{2.1}$$

Where $\bar{\boldsymbol{x}} = (1, x)$. Each activation $a_j^{(1)}$ is tranformed by means of a non-linear activation function $h^{(1)}$ to provide an unit $z_j^{(1)}$ (and a vector $\mathbf{z^{(1)}} = (\mathbf{z_1^{(1)}}, \mathbf{z_2^{(1)}}, ..., \mathbf{z_{m_1}^{(1)}})$) as output from the layer, as follows:

$$z_j^{(1)} = h^{(1)}(a_j^{(1)}) = h^{(1)}(\bar{\boldsymbol{x}}^{\boldsymbol{T}} \boldsymbol{w_j}) \tag{2.2}$$

Here $h^{(1)}$ is some threshold function like a sigmoid or a hyperbolic tangent. In this way we have a m_1 units and a m_1 dimensional vector $\mathbf{z^{(1)}}$. This vector furnishes the input for a new layer, where the equation (2.1) remains valid by substituing:

- $a_j^{(1)}$ with $a_j^{(2)}$
- m_1 with m_2
- x with z
- d with m_1
- $w_{j,i}^{(1)}$ and $w_{j,0}^{(1)}$ with $w_{j,i}^{(2)}$ and $w_{j,0}^{(2)}$

This equation together with another threshold function $h^{(2)}$ will provide a second layer defined by the m_2 dimensional vector $\mathbf{z^{(2)}} = (\mathbf{z_1^{(2)}}, \mathbf{z_2^{(2)}}, ..., \mathbf{z_{m_2}^{(2)}})$. After a certain number of layer (let's say $k-1$ layers), the structure will end with an output layer (layer number k). The dimensionality of the output layer (m_k) depends depend on the task. **In our specific case, $m_k = 1$, and the output value will be the probability that our data is or is not a solar flare**. Moreover **the final threshold function h^k will be a sigmoid, as it is necessary to have the final value between 0 and 1**. Again we will have the following expression:

$$a_1^{(k)} = \bar{\mathbf{z}}^{(\mathbf{k-1})\mathbf{T}} \bar{\mathbf{w}}_{\mathbf{1}}^{\mathbf{k-1}} \tag{2.3}$$

And eventually:

$$z_1^k = h^k(a_1^{(k)}) \tag{2.4}$$

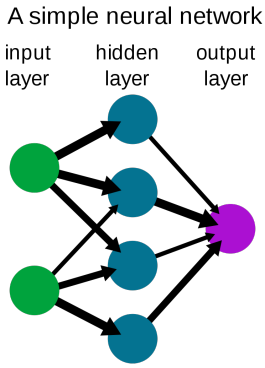


Figure 2.5: **Neural Network** Brief scheme of a 3-layered Neural Network. [21]

As it has already been said, a part of the dataset, namely the training set, is used to train the network and optimize the whole set of parameters $\mathbf{W} = (\mathbf{W^{(1)}}, \mathbf{W^{(2)}}, \mathbf{W^{(3)}}, ..., \mathbf{W^{(k)}})$ where we indicate with $\mathbf{W^{(k)}}$ the matrix containing the vectors $\mathbf{w_j^{(k)}} = (\mathbf{w_{j,0}^{(k)}}, \mathbf{w_{j,1}^{(k)}}, ..., \mathbf{w_{j,\dim(z^{(k-1)})}^{(k}})$ with j that goes from 1 to m_k. The goal of the training using the training set is to find the optimum parameters of our set $\mathbf{W}$. The **optimum set** is obviously related to a task dependent **loss function** that needs to be defined.
As it has already been said, in our specific class, namely **binary classification**, we can consider a single output whose activation function is a logistic sigmoid:

$$y = \sigma(a) = \frac{1}{1 + \exp(-a)} \tag{2.5}$$

So that $0 \leq y(\mathbf{x}, \mathbf{W}) \leq \mathbf{1}$. We can see this number as a probability value that a certain point $\mathbf{x}$ is belonging to the flare class (C_1 class). The complementary class (not belonging) is known as C_2 class, so that:

$$p(C_1|\mathbf{x}, \mathbf{W}) = \mathbf{y}(\mathbf{x}, \mathbf{W})$$
$$p(C_2|\mathbf{x}, \mathbf{W}) = \mathbf{1} - \mathbf{y}(\mathbf{x}, \mathbf{W})$$

We are thus describing a **probabilistic model**, and an important function to optimize is the **likelihood function**. The likelihood function estimates, as the name itself suggests, the likelihood that the model has generated the target value we want to predict.
Speaking in terms of probability, the likelihood is the probability that some target value $t \in \{0, 1\}$ occurs, stated that its correspondent point (element of the dataset) $\mathbf{x}$ and the parameters set $\mathbf{W}$ have occurred: $p(t|\mathbf{x}, \mathbf{W})$. Of course this probabiliy is equal to $p(C_1|\mathbf{x})$ if $t = 1$ and it is equal to $p(C_2|\mathbf{x})$ otherwise. In a more compact manner, we can use the following equation:

$$p(t|\mathbf{x}, \mathbf{W}) = p(C_1|\mathbf{x}, \mathbf{W})^t p(C_2|\mathbf{x}, \mathbf{W})^{1-t} \tag{2.6}$$

On the entire training set the likelihood can be computed as follows:

$$L(t|\mathbf{x}, \mathbf{W}) = \prod_{i=1}^{N} y(\mathbf{x_i}, \mathbf{W})^{t_i} (1 - y(\mathbf{x_i}, \mathbf{W}))^{1-t_i} \tag{2.7}$$

where we have defined t_i as the target value of the element number i, and $\mathbf{x_i}$ as the vector representing the element number i. Again, obviously, $0 \leq L(t|\mathbf{x}, \mathbf{W}) \leq 1$. Nevertheless, it is highly preferable to have this number the closest as possible to 1, as it represents the likelihood that the model is able to generate the real target using its parameters on the element $\mathbf{x}$. For numerical

reasons, and for a convenient property of logarithms ($\log(bc) = \log(b)+\log(c)$) we can define the error function in the following way:

$$E(\mathbf{W}) = -\ln L(\mathbf{t}|\mathbf{x}, \mathbf{W}) = -\sum_{i=1}^{N}(t_i \ln y(\mathbf{x_i}, \mathbf{W}) + (1 - t_i) \ln (1 - y(\mathbf{x_i}, \mathbf{W}))) \tag{2.8}$$

Again, as the final goal is to have $L(\mathbf{t}|\mathbf{x}, \mathbf{W})$ the closest as possible to 1, it is equivalent to say that $E(\mathbf{W})$ needs to be as closest as possible to 0.
Anyway, **the number of parameters in a neural network tends to be very high, thus making practically impossible to solve the optimum problem analytically and find the global optimum**. Nevertheless, **it is possible to use some numerical approach to the optimum problem to find the local optimum**. A well known way to do so is by using a **gradient descent method**, and it is based on changing the parameters set iteratively :

$$\mathbf{W^{(k+1)}} = \mathbf{W^{(k)}} + \Delta\mathbf{W^{(k)}} \tag{2.9}$$

The gradient descent method states that:

$$\Delta\mathbf{W^{(k)}} = -\eta \frac{\partial E_i(\mathbf{W})}{\partial \mathbf{W}}\bigg|_{W^k} \tag{2.10}$$

where:

$$E_i(\mathbf{W}) = -t_i \ln y(\mathbf{x_i}, \mathbf{W}) - (1 - t_i) \ln (1 - y(\mathbf{x_i}, \mathbf{W}))$$

and η is a real positive hyperparameter known as **learning step**.

For each point (i) of the training set, the equation (2.9) is used to update the parameter. Once this is done, the entire process is repeated a certain number of times known as **epochs**. The reason why this updating choice works can be graphically seen in Figure 2.6.
The process starts with an initial random value. We can see that **subtracting the value of the derivative (or the gradient) of the cost function, the parameter w will change in the direction of the minimum of the cost function itself**. **The hyperparameter η is responsable of the magnitude of the step done during the optimization process**. Nevertheless, this hyperparameter plays a crucial role in the optimization problem. In fact:

- If η is too big it is possible that the local optimum is never met as the step may "overcome" the optimum.

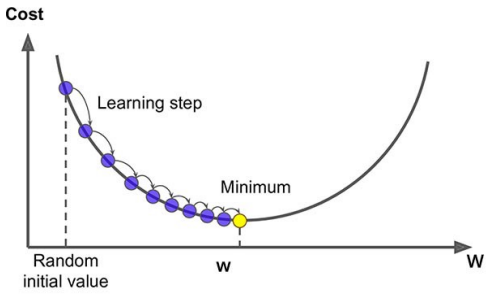


Figure 2.6: **Gradient descent** A simple 2-D dimensional plot showing the updating process using gradient descent [17].

- If η is too small, the optimum may never be reached as well as the number of steps required to reach it could be too big with respect to the number of the epochs.

A standard solution to overcome this problem is to lower η during the optimization, as shown in Figure 2.6.
As it has already been stated that the training set needs to be wide to overcome the possible overfitting scenario, it may result unconvenient to adopt the equation (2.10) and use one point at the time to change the parameters. An alternative and more efficient way to solve the optimization problem is the **batch gradient descent**. This method consists in using a certain subset B of the training set to update the parameters set:

$$\mathbf{W^{(k+1)}} = \mathbf{W^{(k)}} - \eta \sum_{\mathbf{x_i} \in B} \frac{\partial E_i(\mathbf{W})}{\partial \mathbf{W}}\bigg|_{\mathbf{W^{(k)}}} \tag{2.11}$$

This subset B, whose dimension is $\leq$ the dimension of the whole training set, will be changed iteratively until the entire training set is used to update the parameters. Again, the entire process will be repeated one time for each epoch.
To apply the (batch) gradient descent it is thus necessary to compute the derivative of the cost function. To do so let's start by considering a 3 layered (input-hidden-output) feed forward neural network for binary classification, like the one shown in Figure 2.5. Moreover, let's denote $y(\mathbf{x_i}, \mathbf{W}) = y_i$. As the final activation function is a sigmoid, $y = \sigma(a^{(2)})$, the following equation is valid:

$$\frac{\partial E(\mathbf{W})}{\partial a^{(2)}} = -\sum_{i=1}^{N} (t_i \frac{1}{y_i} \frac{\partial y_i}{\partial a_i^{(2)}} - (1 - t_i) \frac{1}{1 - y_i} \frac{\partial y_i}{\partial a_i^{(2)}}) \tag{2.12}$$

Where we have indicated with $a_i^{(2)}$ the element number i of the vector $a^{(2)}$, that is the vector of the prediction y before the application element-wise of the sigmoid function. By construction:

$$\frac{\partial y}{\partial a^{(2)}} = y(1-y) \tag{2.13}$$

As a consequence:

$$\frac{\partial E(\mathbf{W})}{\partial a^{(2)}} = \sum_{i=1}^{N}(y_i - t_i) \tag{2.14}$$

This equation will be a crucial one during the compute of the derivative of the cost function. In fact, these derivatives have to be computed for each dimension of the total set of parameters $\mathbf{W}$. As it has already been stated, all these parameters are connected to each other through the underlying network structure. For this reason, the computation of the derivative with respect to a particular parameter $w_k^{(j)}$ has to start from the last layer and follow the network back to the interested layer (in our case the layer number j). This mechanism is obviously in the opposite sense of the natural flow of the network. For this reason, it is called **back-propagation algorithm**.

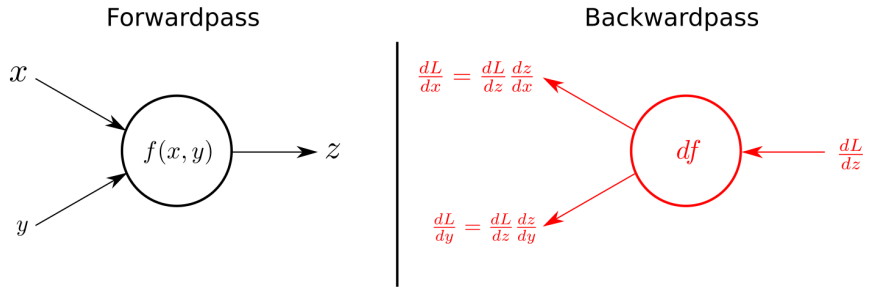


Figure 2.7: **Feed-Forward Neural Network vs Backpropagation Algorithm** A brief scheme showing the difference between the Feed Forward Neural Network and its optimization technique that has to flow in the opposite sense to backpropagate the compute of the derivatives. [8]

Let's assume that for each element of the training set $\mathbf{x_i}$, and the correspondent target t_i, the ouput value has been computed (forward propagation). We wish to evaluate the derivative of E_i with respect to the parameter $w_{j,l}^{(k)}$. We know that:

$$a_j^{(k)} = \sum_{r=1}^{m} w_{j,r}^{(k)} z_r^{(k-1)} \tag{2.15}$$

Then, denoting:

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}}$$

and stated that:

$$\frac{\partial a_j^{(k)}}{\partial w_{j,l}^{(k)}} = z_l^{(k-1)}$$

We have that:

$$\frac{\partial E_i}{\partial w_{j,l}^{(k)}} = \frac{\partial E_i}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial w_{j,l}^{(k)}} = \delta_j^{(k)} z_l^{(k-1)} \tag{2.16}$$

Considering the highest k value ($k = k_{\max}$), in our case we simply have, given the element number i and its correspondent target and prediction:

$$\delta_1^{(k)} = y_i - t_i \tag{2.17}$$

Thus obtaining for the last layer the following expression:

$$\frac{\partial E_i}{\partial w_{1,l}^{(k)}} = z_l^{(k-1)}(y_i - t_i) \tag{2.18}$$

When the update has to be done for the parameters of the previous layer (from $k-1$ to the 1), we need that any change that is done on the k layer affects E_i by inducing changes for all the variables $a_l^{(k+1)}$. In fact, using the chain rule, we can notice that:

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \frac{\partial E_i}{\partial a_r^{(k+1)}} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} \tag{2.19}$$

Since by definition:

$$a_r^{(k+1)} = \sum_l w_{r,l}^{(k+1)} z_l^{(k)} \tag{2.20}$$

$$z_j^{(k)} = h_k(a_j^{(k)}) \tag{2.21}$$

It results:

$$\frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \frac{\partial a_r^{(k+1)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial a_j^{(k)}} = w_{r,j}^{(k+1)} h_k'(a_j^{(k)}) \tag{2.22}$$

Consequently:

$$\delta_j^{(k)} = h_k'(a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} w_{r,j}^{(k+1)} \tag{2.23}$$

And as a final result:

$$\frac{\partial E_i}{\partial w_{j,l}^{(k)}} = z_l^{(k-1)} h_k'(a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} w_{r,j}^{(k+1)} \tag{2.24}$$

This final equation represents the updating that can be done for each point of the training set and for each parameter of the set $\mathbf{W}$.

2.4 Convolutional Neural Network

A specific kind of Feed Forward Neural Network that is able to get high performances in terms of image object detection and image classification is known as **convolutional neural network (CNN)**.
Convolutional Neural Networks are very similar to feed forward neural network, as they are made of neurons with weights and biases that have to be learned in the most efficient way. Moreover, at the same way of feed forward neural networks, each neuron receives some input, performs a dot product and optionally follows it with a non-linear operation. A powerful property of convolutional neural network is that the task is performed by taking as input the raw pixels of the image. Moreover, in our case the task is binary image classification, so that the last fully connected layer represents the probability that the image belongs to the C_1 class.
As we could expect, when treating an image (with $\geq 10^2$ pixels) if all the layers are fully connected like the feed forward neural network shown in Figure 2.5, the number of parameters explodes and the overfitting scenario becomes highly probable. Moreover the computational cost of such a network results extremely high.
In fact, important elements of a convolutional neural network are the **convolutional layers**, that use a different approach. As the name itself suggests, convolutional layers are based on a **convolutional operation**, with a matrix, called **kernel**, "moving" on the output of the previous layer. The dimensions $(w \times h)$ of the kernel are smaller than the dimension of the output of the previous layer. The elements of this matrix are multiplied (element-wise) by the elements of the first $w \times h$ window (starting by the highest pixel to the extreme left) of the output of the previous layer. This computation furnishes the first element (first row and first column) of the output matrix of the convolution. Moving one pixel to the right on the output of the previous layer, thus considering the second $w \times h$ window starting by the immediate right of the extreme left highest pixel, we perform the same operation on the new window obtaining the second element (first row and second column) of the output matrix. Moving down instead of moving right, and performing

the same operation, we obtain the elements of the lower rows of the output matrix. The entire process is summarized by the illustration in Figure 2.8.

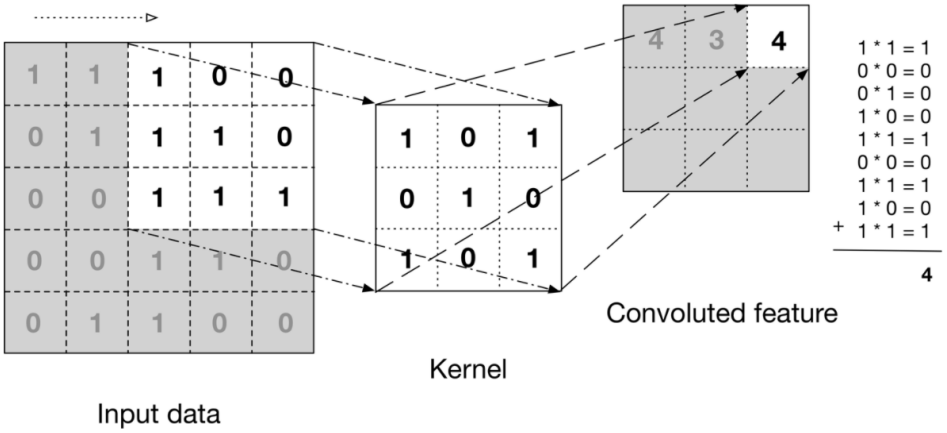


Figure 2.8: **Convolutional Layer** An example of how convolutional layers perform on an Image. In this example the Input Data is a 5×5 matrix, while the kernel is a 3×3 matrix. The kernel moves on the input data and outputs the result building the convoluted feature. [13]

Convolutional layers are characterized by 5 **hyper-parameters**:

- **Width**: The width of the kernel matrix
- **Height**: The height of the kernel matrix
- **Depth**: The depth of the kernel matrix/tensor. Each layer is meant to identify a specific feature of the input data (e.g. shapes,shades,lines...)
- **Zero-Padding**: The number of zero borders that are added to the input image in order to weight each element of the input in the same way. In fact if there is no border, the border elements of the image are considered less times than the others. If Zero-Padding= 1 a border of 0 is added, if Zero-Padding= k, k borders of 0 are added to the input image
- **Stride**: The step of the filter slide over the output of the previous layer. When the stride is 1 filters are moved one pixel at a time. When the stride is k then the filters jump k pixels at a time as they are slided around.

As it was anticipated before, the application of Convolutional Layers in place of simple Feed Forward Neural Network (dense) Layers implies two important advantages:

- **Parameters sharing**: all the parameters of a same convolutional layer are the ones that appear in the kernel
- **Non-Dense layer**: the dimension of the kernel does not correspond with the one of the input. That means that some units have weight= 0

These advantages make the convolutional operation cheaper than a simple feed forward neural network step and the number of parameters smaller, thus speeding the training process and preventing the overfitting scenario.
Convolutional Neural Networks usually apply another kind of layers, known as **pooling layers**.
Pooling layers are meant to decrease the total number of parameters in order to prevent overfitting. The function of a pooling layer is to focus on a determined window and summarize its content by using the mean (**average pooling**), the max value (**max pooling**) or the min value (**min pooling**) of the window itself. Unlike convolutional layer, pooling layers do not require any parameters. As the convolutional layers, the width, height, depth, and stride remain valid hyperparameter of the pooling layers, while usually Zero-padding=0.

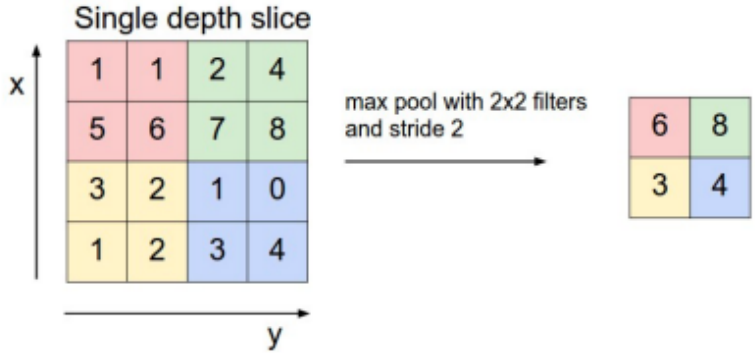


Figure 2.9: **Max Pooling** An example of how max pooling layers perform on an Image. In this example the Input Data is a 4×4 matrix, while the pooling window is a 2×2 matrix with stride= 2. The pooling takes only the higher value of the window that has been selected. [6]

A third important structure to consider is the **RELU layer**. In fact, as the backpropagation method is applied, it is necessary to compute the derivative values $\frac{\partial E_i}{\partial w_{j,l}^{(k)}}$. These values tend to monotonically decrease as k decreases during the backpropagation, thus not updating the first layers parameters. This problem is known as **vanishing gradient** and it is solved by applying element wise the RELU function:

$$h(x) = \max(0, x) \tag{2.25}$$

A RELU layer simply applies this relu function to the output of the previous layer.

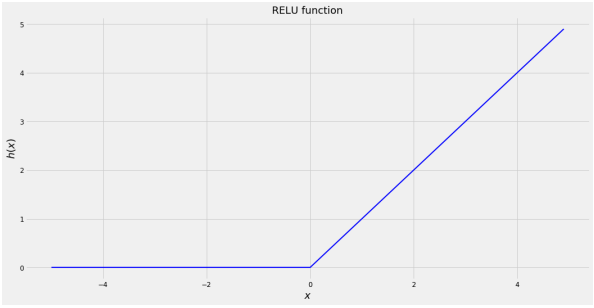


Figure 2.10: **RELU function** Plot of the RELU function.

These three kind of layers (convolutional, relu, and pooling) are used in sequence to build the convolutional neural network. Layer by layer, the network will analyze the input data, and will process it, in order to distinguish the features of the input image and perform its task. As it has already been stated, the last layer needs to be fully connected, in order to use the total amount of "knowledge" that has been collected during the network.

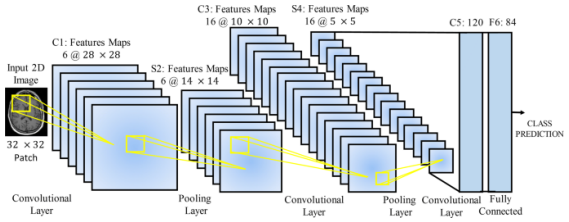


Figure 2.11: **Example of a CNN for medical image classification** Typical structure of a CNN used for image classification. A sequence of convolutional, RELU and max pool layers, followed by a fully connected layer has been used. [1]

3 Data

All the data that has been used are available in open source manner from the following sources:

- **Space Weather Archive:** `https://www.spaceweatherlive.com/en/archive`
- **Big Bear Solar Observatory (BBSO):** `ftp://ftp.bbso.njit.edu/pub/archive/`
- **Lockheed Martin Solar and Astrophysics Laboratory (LMSAL):** `https://www.lmsal.com/`

In particular **the first two sources have been used for the first algorithm classification dataset, that classifies whether or not the Sun has active regions**. Specifically, the Space Weather calendar, that reports day by day the solar activity, has been used during the data labeling process. The GONG Full Disk Magnetograms images have been extracted from the BBSO archive. **106 Images and events happened in 2013, 2018 and 2019 have been collected**. In particular 62 pictures of the Sun with active regions 44 pictures of the Sun without active regions have been extracted, obtaining an acceptably balanced dataset:

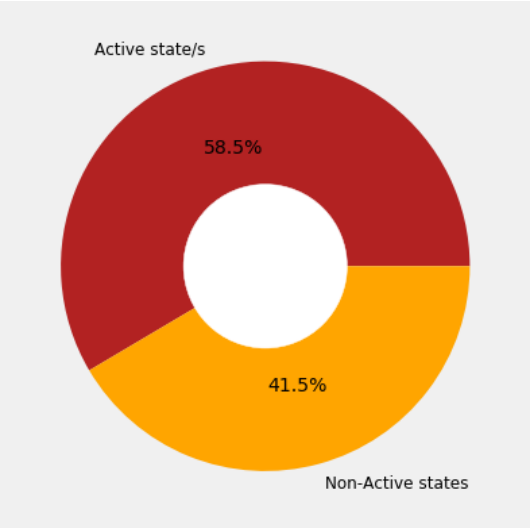
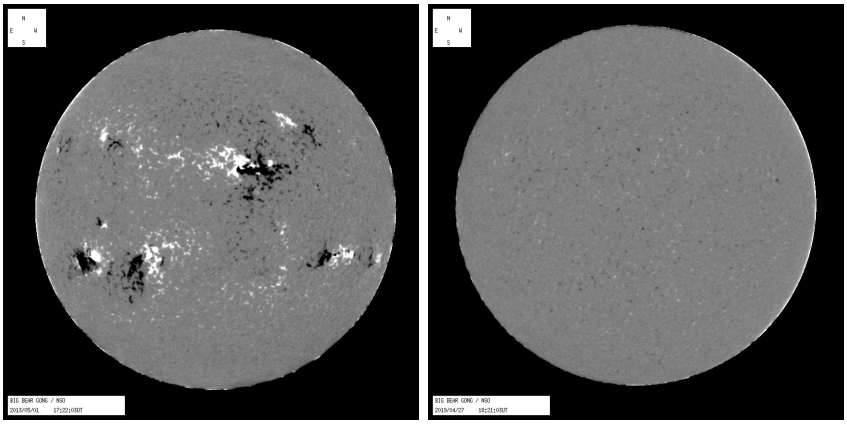


Figure 3.1: **Pie plot showing the first algorithm dataset** As it is possible to observe the dataset is acceptably balanced. [1]



(a) (b)

Figure 3.2: **First Dataset example**: An example of the first dataset image has been reported. Figure (a) represents the Sun in an active state, while (b) represents the Sun in a non active state (i.e. with no active regions) [2][3]

The last source has been used to extract and simultaneously label detailed images of solar flares and active regions for the second. 150 Atmospheric Imaging Assembly [9] images from 2019 have been collected. In particular, 80 images represent Solar Flares, while the other 70 represent active regions with Non-Solar Flares:

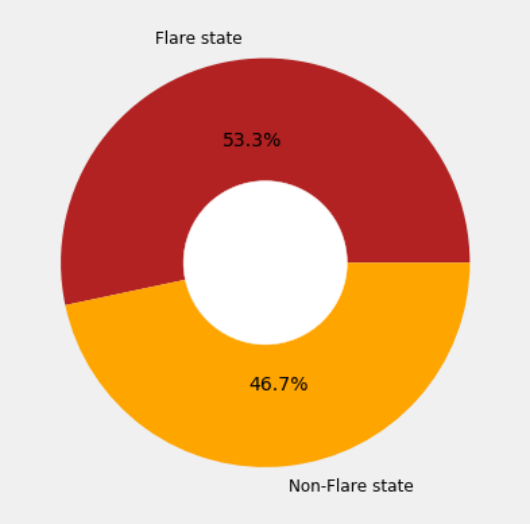
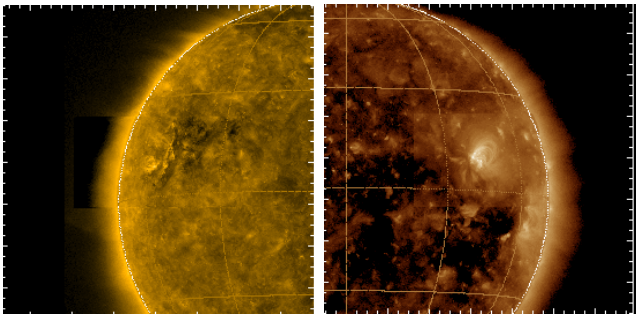


Figure 3.3: **Pie plot showing the second algorithm dataset** As it is possible to observe the dataset is acceptably balanced. [1]



(a) (b)

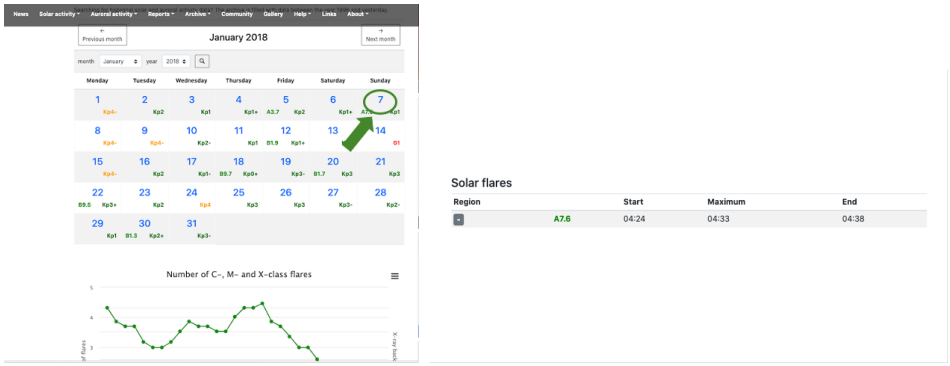
Figure 3.4: **Second Dataset example**: An example of the second dataset image has been reported. Figure (a) represents the Sun in a non Flare state, while (b) represents the Sun in a Flare state. [18]

3.1 Data extraction and data labeling

As it has already been stated, all the data that has been used are available in an open source manner. Nevertheless, both the first and the second dataset were not available in a single downloadable file. Moreover, as the algorithm that have been used are part of the supervised learning algorithms, it has been necessary to label the data accurately. Previous works completed this task hand-labeling the images that have been collected [16]. This approach can cause misclassifications and can be very long to apply. **The alternative approach used in this work is to consider the archives (Space Weather and LMSAL) to label the images in a more robust and rapid manner**.

3.1.1 First Dataset

For the first dataset, it has been necessary to collect full disk solar images both when the Sun was devoid of active regions and when the Sun was magnetically active. To do so, using web scraping technique on the Space Weather Archive site, **the records, day by day, of the solar activity has been extracted and stored**:



(a) (b)

Figure 3.5: **Space weather archive**: A quick look at the Space Weather Archive site. Day by day the report of the solar flares that occurred is furnished. [18]

In particular, **the solar flares start, peak, and end time have been collected for 2019,2018 and 2013**:

	Date	Start	Maximum	End
0	2018/01/07	04:24	04:33	04:38
1	2018/01/12	13:08	13:09	13:10
2	2018/01/18	07:28	07:35	07:39
3	2018/01/18	08:08	08:13	08:16
4	2018/01/18	09:51	09:57	10:06
5	2018/01/20	04:48	04:55	05:00
6	2018/01/20	17:42	17:52	17:58
7	2018/01/22	02:37	02:57	03:20
8	2018/01/30	17:35	17:39	17:46
9	2018/02/04	19:06	19:22	19:27
10	2018/02/04	19:46	19:51	19:55

Figure 3.6: **Web scraping algorithm result**: 10 entrance of the resulting dataset output from the Web Scraping algorithm. [1]

As this specific part of the work has the preliminary function to distinguish whether or not active regions are present in the Sun, the images it is classifying are the full image disk of the BBSO archives, so this flare dataset has been used to pick the images that present active regions. Nevertheless, as the BBSO archive only collects images at certain hours and minutes, not every active Sun image has been collectable. The day and the time of the collection from the BBSO archive has been extracted from the image name itself:

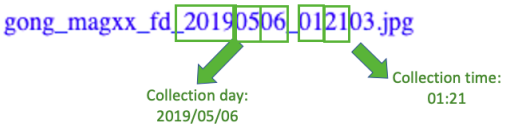


Figure 3.7: **Web scraping algorithm result**: 10 entrance of the resulting dataset output from the Web Scraping algorithm. [1]

Comparing time and day from the dataset shown in figure 3.6, all the available active Sun images have been collected. In this way the solar active images have been collected. The non solar active images have been picked from the BBSO site in different times and days with respect to the ones that were indicated by the solar flare dataset. In particular, as the algorithm is analyzing the full disk image, only the images that didn't present any solar active regions have been used in the training part to help the algorithm in its task.

3.1.2 Second Dataset

The data extraction and labeling in this part of the work was easier than the first. In fact, no comparison has been made, and only one source has been used. Using the appropriate query, all the active regions and solar flares can be displayed. The web scraping techniques have been here applied only to acquire the data and their labels.

4 Algorithms

Two different, but very similar, algorithms have been used in this work. Both of them are **convolutional neural network** for binary classification:

- **The first algorithm** is meant to distinguish whether or not the input image represents a magnetically active Sun.
- **The second algorithm** is meant to distinguish active regions from solar flares.

Both the networks follow a similar structure structure of the network Figure 2.10.

4.1 First Algorithm

As the input of the first network is a black and white magnetogram image (BBSO archive), the input consists of a single layer. This layer is composed by a matrix with numbers between 0 and 255, indicating the intensity of white of the single pixel:

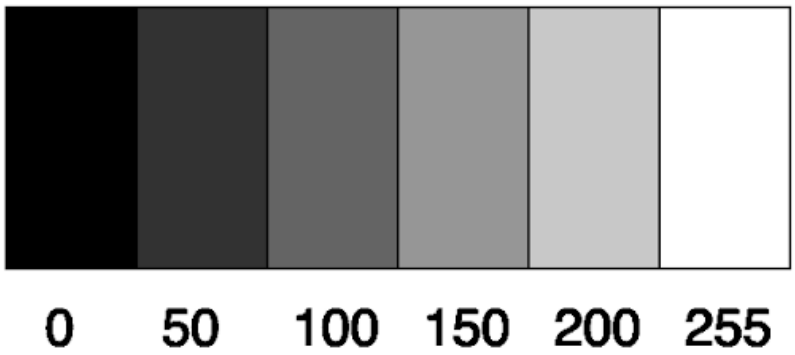


Figure 4.1: **Color Scale**: Assignment of the pixel number in correspondence of the color of the image pixel. [19]

Firstly, the input image matrix has been resized in order to obtain, for each input image, a 300×300 matrix. Subsequently, two sequences of a convolutional layer and of max pooling layer has been applied with dimensions shown in Figure 4.2. A "flatten filter" is then applied to flatten the resulting output and prepare it for the last dense layer. The output of the last layer is the class prediction:

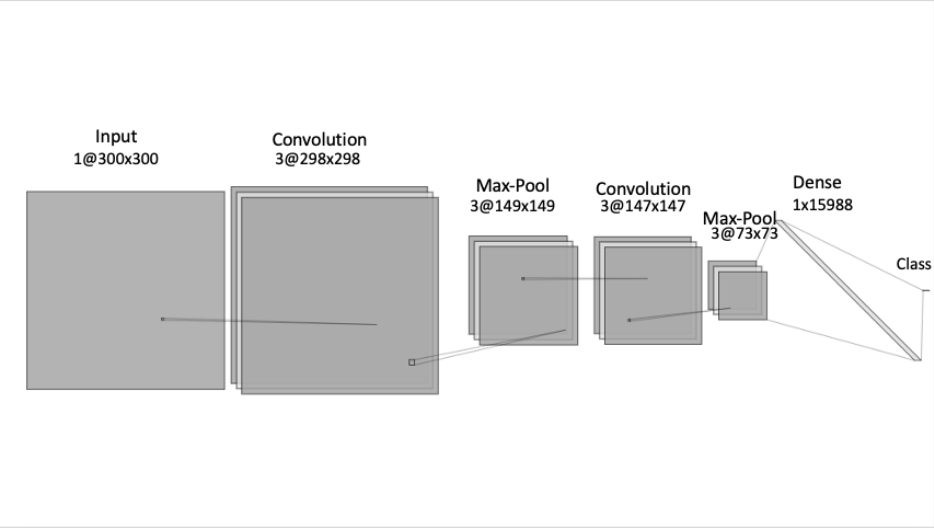


Figure 4.2: **First algorithm/Convolutional Network**: An intuitive illustration of the second Convolutional Neural Newtork. The 300×300 input passes through a first 3-dimensional convolutional layer (each dimension is a 298×298 matrix. Subsequently a Max Pooling layer is applied, thus reducing the dimension to a three 149×149 matrices. Subsequently, a new 3-dimentionsal convolutional layer has been applied with dimensions 147×147. Another 3-dimensional Max Pooling layer is applied to reduce the dimension of each matrix of the previous output, thus obtaining 73×73 matrices. A "flatten" layer is applied to flatten the previous output, thus obtaining a $73 \times 73 \times 3 = 15987 + 1$ bias parameter$= 15988$ dimensional vector. Finally, the last, dense, layer furnishes an ouput (0/1) that represents the class prediction for the input image. [10]

4.2 Second Algorithm

The second algorithm is really similar to the first. The only two differences are the following:

- **Images are represented using the RGB color model**: the images are not magnetogram but AIA images. The input of the network is not a single matrix anymore, but a tensor built with three matrix with values between 0 and 255, indicating the intensity of red, green and blue in the image for that pixel.
- **Images are specific of a certain portion of the Sun**: as the second algorithm needs to distinguish active regions from solar flares, the images are focused on a specific portion of the Sun. For this reason, **all the images have been resized to** 250×250 **images**.

This new algorithm (with its new dimensions) is shown in Figure 4.3:

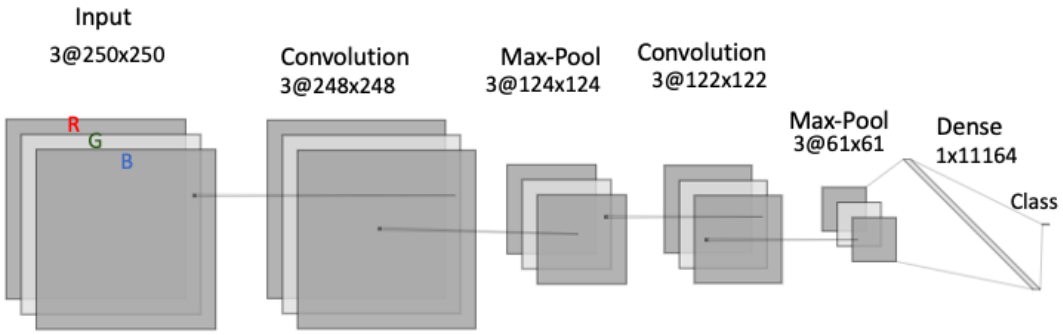


Figure 4.3: **Second algorithm/Convolutional Network**: An intuitive illustration of the second Convolutional Neural Newtork. The $250 \times 250 \times 3$ input passes through a first 3-dimensional convolutional layer (each dimension is a 248×248 matrix. Subsequently a Max Pooling layer is applied, thus reducing the dimension to a three 124×124 matrices. Subsequently, a new 3-dimentionsal convolutional layer has been applied with dimensions 122×122. Another 3-dimensional Max Pooling layer is applied to reduce the dimension of each matrix of the previous output, thus obtaining 61×61 matrices. A "flatten" layer is applied to flatten the previous output, thus obtaining a $61 \times 61 \times 3 = 11163 + 1$ bias parameter$= 11164$ dimensional vector. Finally, the last, dense, layer furnishes an ouput (0/1) that represents the class prediction for the input image. [10]

5 Software Implementation

The programming language that has been used in this work is **Python 3.7**. In particular, the following libraries have been used:

- **Selenium** and **time** for automatic web browsing and web scraping
- **Pandas** for data handling and processing
- **Keras**, **Sklearn** and **TensorFlow** for Machine Learning
- **Numpy** for vector handling and computing
- **Matplotlib** for data visualization
- **PIL** for its image to vector functions
- **os** for data storing

The most important lines of the code regarding web scraping, data analaysis and machine learning are reported in the next paragraphs.

5.1 Space Weather Web Scraping

The following lines of codes have been used to scrape the Space Weather archive [14] and extract a table with all the solar flares start, maximum and peak of 2019. The same code has been used for 2018 and 2013 by simply changing the variable "year":

```
1
2 #This first part of the code wants to build a total calendar
     for 2019.
3 year=2019
4 MONTHS=[]
5 months_number=[]
6 for i in range(1,13):
7     if i<10:
8         MONTHS.append('0%s/'%(i))
9         months_number.append('0'+str(i))
10    else:
11        MONTHS.append('%s/'%(i))
12        months_number.append(str(i))
13 twentyeight_day=[]
14 twentyeight_day_number=[]
15 for i in range(1,29):
16    if i<10:
17        twentyeight_day_number.append('0'+str(i))
```

```
18          twentyeight_day.append('0%s/'%(i))
19      else:
20          twentyeight_day_number.append(str(i))
21          twentyeight_day.append('%s/'%(i))
22  thirty_day=[]
23  thirty_day_number=[]
24  for i in range(1,31):
25
26      if i<10:
27          thirty_day.append('0%s/'%(i))
28          thirty_day_number.append('0'+str(i))
29      else:
30          thirty_day.append('%s/'%(i))
31          thirty_day_number.append(str(i))
32  thirtyone_day=[]
33  thirtyone_day_number=[]
34  for i in range(1,32):
35      if i<10:
36          thirtyone_day.append('0%s/'%(i))
37          thirtyone_day_number.append('0'+str(i))
38      else:
39          thirtyone_day.append('%s/'%(i))
40          thirtyone_day_number.append(str(i))
41  DAYS=[thirtyone_day,twentyeight_day,thirtyone_day,thirty_day,
        thirtyone_day,thirty_day,thirtyone_day,thirtyone_day,
        thirty_day,
42        thirtyone_day,thirty_day,thirtyone_day]
43  DAYS_NUMBER=[thirtyone_day_number,twentyeight_day_number,
        thirtyone_day_number,thirty_day_number,
        thirtyone_day_number,thirty_day_number,
        thirtyone_day_number,thirtyone_day_number,
        thirty_day_number,
44        thirtyone_day_number,thirty_day_number,
        thirtyone_day_number]
45  #This is the scraping part
46  path='/Users/pierohmd/Desktop/University/Magistral/To do
        tests/Neural Networks Sun/chromedriver' #The path where
        the chromedriver file is present
47  CONTENT=[] #This is a list that will be filled with all the
        content, day by day, of 2019
48  driver = webdriver.Chrome(path)# Go to your page url
49  driver.get('https://www.spaceweatherlive.com/en/archive
        /2019/01/01/xray')#go to the first site (first day of the
        year)
50
51  time.sleep(4) #Sleep for 4 seconds in order to "humanize" the
         process
52  content=driver.find_element_by_xpath('//*[@id="
        Archive_SolarFlare_table"]/table[2]').text #Find the table
```

```
    with the info and its text
CONTENT.append(content) #Fill the Content list with all these
    values
for i in range(len(months_number)): #Do that for each day of
   the year
    for j in range(len(DAYS_NUMBER[i])):
        driver.get('https://www.spaceweatherlive.com/en/
   archive/2019/'+months_number[i]+'/'+DAYS_NUMBER[i][j]+'/'+
   'xray')
        element = WebDriverWait(driver, 10).until(EC.
   presence_of_element_located((By.ID, '
   Archive_SolarFlare_table')))

        content=driver.find_element_by_id('
   Archive_SolarFlare_table').text

        time.sleep(4)
        CONTENT.append(content)
        print('Scraping done for %s/%s/%s'%(year,
   months_number[i],DAYS_NUMBER[i][j]))
#Separe the text in content to analyze each word
TOK_CONTENT=[]
for c in CONTENT:
    TOK_CONTENT.append((word_tokenize(c)))
#Clean the TOK_CONTENT content
NEW_TOK_CONTENT=[]
for data in TOK_CONTENT:
    try:
        start=data.index('End')
        new_data=[]
        for i in range(start+1,len(data)):
            new_data.append(data[i])
        NEW_TOK_CONTENT.append(new_data)
    except:
        NEW_TOK_CONTENT.append(['Not available'])
#A function that will be used to display the result
def isSolarFlare(event_list):
    issolarflare=[]
    for e in event_list:
        if e[0]=='None' or e[0]=='Not available':
            issolarflare.append(0)
        else:
            issolarflare.append(1)


    return issolarflare
#All these information are stored in a CSV file:
month_data=[]
for i in range(len(DAYS_NUMBER)):
```

```
    for j in range(len(DAYS_NUMBER[i])):
        month_data.append('%s'%(i+1))
day_data=[]
for i in range(len(DAYS_NUMBER)):
    for j in range(len(DAYS_NUMBER[i])):
        day_data.append(DAYS_NUMBER[i][j])
date=[]
for m in range(len(months_number)):
    for d in range(len(DAYS_NUMBER[m])):
        date.append('2019'+'/'+ months_number[m]+ '/' +
   DAYS_NUMBER[m][d])
label_data=pd.DataFrame()
label_data['Date']=date
label_data['Month']=month_data
label_data['Day']=day_data
label_data['Solar Flare Event']=isSolarFlare(NEW_TOK_CONTENT)
#Now we take this information and we build a final CSV file
   with start, maximum and peak of each solar flare
SOLAR_FLARE_H=[]

for event in SOLAR_FLARE_TOK:
    hour_day=[]
    for element in event:
        hour=[]
        for character in element:
            if character==':':
                hour.append(element)
        hour_day.append(hour)
    SOLAR_FLARE_H.append(hour_day)
NEW_SOLAR_FLARE_H=[]
for event in SOLAR_FLARE_H:
    flat_list = [item for sublist in event for item in
   sublist]
    NEW_SOLAR_FLARE_H.append(flat_list)
how_many=[]
for day in NEW_SOLAR_FLARE_H:
    how_many.append(int(len(day)/3))
date=[]
for m in range(len(months_number)):
    for d in range(len(DAYS_NUMBER[m])):
        date.append('2019'+'/'+ months_number[m]+ '/' +
   DAYS_NUMBER[m][d])
solarflare_date=label_data[label_data['Solar Flare Event'
   ]==1].Date.tolist()
new_solar_flare_date=[]
for h in range(len(how_many)):
    for i in range(how_many[h]):
        new_solar_flare_date.append(solarflare_date[h])
label_detailed_data=pd.DataFrame()
```

```
label_detailed_data['Date']=new_solar_flare_date
def chunkIt(seq, num):
    avg = len(seq) / float(num)
    out = []
    last = 0.0

    while last < len(seq):
        out.append(seq[int(last):int(last + avg)])
        last += avg

    return out
FINAL_SOLAR_FLARE_H=[]
for i in range(len(NEW_SOLAR_FLARE_H)):
    FINAL_SOLAR_FLARE_H.append(chunkIt(NEW_SOLAR_FLARE_H[i],
   how_many[i]))
START=[]
PEAK=[]
END=[]
for i in range(len(FINAL_SOLAR_FLARE_H)):
    for j in range(len(FINAL_SOLAR_FLARE_H[i])):
        START.append(FINAL_SOLAR_FLARE_H[i][j][0])
        PEAK.append(FINAL_SOLAR_FLARE_H[i][j][1])
        END.append(FINAL_SOLAR_FLARE_H[i][j][2])
label_detailed_data['Start']=START
label_detailed_data['Maximum']=PEAK
label_detailed_data['End']=END
```

This is an example of the output:

	Date	Start	Maximum	End
0	2019/01/02	01:44	01:50	01:55
1	2019/01/02	18:32	18:38	18:44
2	2019/01/03	16:40	17:14	17:33
3	2019/01/04	11:04	11:35	11:46
4	2019/01/06	01:56	02:02	02:09
5	2019/01/06	03:35	03:41	03:47
6	2019/01/06	04:23	04:52	05:14
7	2019/01/06	05:32	05:51	06:01
8	2019/01/06	08:35	08:41	08:48
9	2019/01/06	10:31	10:51	11:00
10	2019/01/06	13:33	13:37	13:39

Figure 5.1: **Space Weather Web Scraping output**: The first 10 rows of the output dataset of the above code.

5.2 BBSO Web Scraping

The BBSO Web Site has been scraped to check the availability of the files. Firstly, the date and the time of all the events have been obtained and extracted. Subsequently, this availability has been cross-checked with the previous dataset, to check if images have been collected between the start and the end of each solar flare. Finally, the available images have been used to get images of full disk magnetically active Sun (labelled with 1). The other images have been pick randomly with a similar code, trivially avoiding the date and time of solar flares.

```
#The dates of the previous dataset have been obtained in unique entries and the web url has been defined
web_url='ftp://ftp.bbso.njit.edu/pub/archive/'
research_date=label_detailed_data.Date.drop_duplicates().tolist()
#The available dates for the Sun images have been extracted
TOT_TIME_TOK=[]
for j in range(len(research_date)):
    driver=webdriver.Chrome(path)
    print('Scraping done for %s , another %i day missing' %(research_date[j],len(research_date)-(j+1)))
    driver.get(web_url+research_date[j])
    try:
        text=driver.find_element_by_xpath('//*[@id="tbody"]').text
        TOK=word_tokenize(text)
        NEW_TOK=[]
        for i in range(len(TOK)):
            counter=0
            for j in range(len(TOK[i])):
                if TOK[i][j]!='x':
                    counter=counter+1
            if counter!=len(TOK[i]):
                NEW_TOK.append(TOK[i])
        FINAL_TOK=[]
        for i in range(len(NEW_TOK)):
            LIST=NEW_TOK[i].split('.')
            if LIST[1]=='jpg':
                FINAL_TOK.append(LIST[0])
        HOUR_TOK=[]
        for i in range(len(FINAL_TOK)):
            LIST=FINAL_TOK[i].split('_')
            HOUR_TOK.append(LIST[len(LIST)-1])
        TIME_TOK=[]
        sep=':'
        for i in range(len(HOUR_TOK)):
            hour=HOUR_TOK[i][0]+HOUR_TOK[i][1]
```

```
            minutes=HOUR_TOK[i][2]+HOUR_TOK[i][3]
            TIME_TOK.append(hour+sep+minutes)
        TOT_TIME_TOK.append(TIME_TOK)
    except:
        TOT_TIME_TOK.append(['Not available'])
    driver.quit()
for i in range(len(TOT_TIME_TOK)):
    if len(TOT_TIME_TOK[i])==0:
        TOT_TIME_TOK[i]=['Not available']
for i in range(len(TOT_TIME_TOK)):
    try:
        inde=TOT_TIME_TOK.index('Not available')
        TOT_TIME_TOK[inde]=['Not available']

    except:
        break
# To understand if a picture time is between the start and
    the end time for that day
# a conversion from hours and minutes to minutes only have
    been used:
def datetominute(date):
    try:
        hour=int(date[0]+date[1])
        min_hour=hour*60
        tot_min=min_hour+int(date[3]+date[4])
    except:
        tot_min='Not available'
    return tot_min
# A formatting order has been used:
MIN_TO_TIME_TOK=[]
for day in TOT_TIME_TOK:
    min_day=[]
    for hour in day:
        min_hour=datetominute(hour)
        min_day.append(min_hour)
    MIN_TO_TIME_TOK.append(min_day)
how_many=label_detailed_data.groupby('Date').count().Start.
    tolist()
REPEAT_TOT_TIME_TOK=[]
for i in range(len(how_many)):
    for j in range(how_many[i]):
        REPEAT_TOT_TIME_TOK.append(MIN_TO_TIME_TOK[i])
Start_minute=[]
Peak_minute=[]
End_minute=[]
Start_normal=minute_labeled_data['Start'].tolist()
Peak_normal=minute_labeled_data['Maximum'].tolist()
End_normal=minute_labeled_data['End'].tolist()
for s in range(len(Start_normal)):
```

```
    Start_minute.append(datetominute(str(Start_normal[s])))
    Peak_minute.append(datetominute(str(Peak_normal[s])))
    End_minute.append(datetominute(str(End_normal[s])))
minute_labeled_data=label_detailed_data
minute_labeled_data['Start']=Start_minute
minute_labeled_data['Maximum']=Peak_minute
minute_labeled_data['End']=End_minute
repeat_TOT_TIME_TOK=[]
for i in range(len(how_many)):
    for j in range(how_many[i]):
        repeat_TOT_TIME_TOK.append(TOT_TIME_TOK[i])
AVAILABLE=[]
TOT_DATA=[]
TOT_HOUR_DATA=[]
for i in range(len(minute_labeled_data)):
    start=minute_labeled_data.Start.loc[i]
    end=minute_labeled_data.End.loc[i]
    count=0
    DATA=[]
    HOUR_DATA=[]
    for j in range(len(REPEAT_TOT_TIME_TOK[i])):
        data=REPEAT_TOT_TIME_TOK[i][j]
        hour_data=repeat_TOT_TIME_TOK[i][j]
        if data=='Not available':
            continue
        else:
            data=int(data)

            if data>=int(start) and data<=int(end):
                count=count+1
                DATA.append(data)
                HOUR_DATA.append(hour_data)
    if count==0:
        AVAILABLE.append('No')
        TOT_DATA.append('-')
        TOT_HOUR_DATA.append('-')
    else:
        AVAILABLE.append('Yes')
        TOT_DATA.append(DATA[0])
        TOT_HOUR_DATA.append(HOUR_DATA[0])
#The previous csv file has been used and modified, adding an
   'AVAILABLE' column
        #which say if the file is or not available
label_detailed_data=pd.read_csv('SolarFlareDetailedData.csv')
   .drop(columns=['Unnamed: 0'])
label_detailed_data['Available']=AVAILABLE
label_detailed_data['Data']=TOT_HOUR_DATA
label_detailed_data[label_detailed_data['Available']=='Yes'].
   to_csv('Availabledata.csv')
```

```
DATE=label_detailed_data.Date.tolist()
TIME=label_detailed_data.Data.tolist()
#The Codes have been prepared for the scraping part
CODES=[]
for i in range(len(TIME)):
    CODES.append('gong_magxx_fd_'+DATE[i].replace('/','')+'_'
   +TIME[i].replace(':','')+'03.jpg')
#These very simple lines of codes are used for the scraping
   part
base_string='ftp://ftp.bbso.njit.edu/pub/archive/'
for i in range(len(DATE)):
    date=DATE[i]
    driver = webdriver.Chrome(path)# Go to your page url
    driver.get(base_string+date)
    driver.find_element_by_link_text(CODES[i]).click()
```

5.3 LMSAL iSolSearch web scraping

Heliophysics Event Registry (`https://www.lmsal.com/`) provided a specific csv with all the url reporting to the image of the single events. For this reason, the Web Scraping part of this code is simpler than the other two. The web scraping techniques here only regard the automatic labelling (from the file encoding) and the storage by saving the images from the url.

```
#The Helyophysics Event Registry has been stored as a .csv file named work
convert=pd.read_csv('work.csv')
data=pd.DataFrame()
#Only few columns have been actually helpful
data['Event']=convert[convert.columns.tolist()[0]]
data['Image']=convert[convert.columns.tolist()[11]]
data['Type']=convert[convert.columns.tolist()[18]]
data=data.dropna()
#Scraping technique here adopted:
path='/Users/pierohmd/Desktop/University/Magistral/To do tests/Neural Networks Sun/chromedriver'
driver=webdriver.Chrome(path)
for i in range(len(data)):
    time.sleep(3)
    driver.get(data.Event.tolist()[i])
    try:
        img=driver.find_element_by_xpath('/html/body/img')
        actionChains = ActionChains(driver)
        actionChains.move_to_element(img).context_click().perform()
        pyautogui.typewrite(['down','down','enter'])
        time.sleep(2)
        pyautogui.typewrite(['delete'])
        pyautogui.write(data.Type.tolist()[i]+str(i))
        pyautogui.typewrite(['enter'])
        time.sleep(2)
    except:
        print('Not available')
        continue
```

5.4 First algorithm Neural Network

: The practical implementation of the algorithm shown in figure 4.2 has been here reported:

```
#The images have been imported in the Python File
size=300
entries = os.listdir('/Users/pierohmd/Desktop/University/
    Magistral/To do tests/Neural Networks Sun/Notebook')
entries.sort(key=natural_keys)
del entries[entries.index('.ipynb_checkpoints')]
del entries[entries.index('.DS_Store')]
data=[]
IMAGES=[]
for entry in entries:
    try:
        image = Image.open(entry)
        IMAGES.append(image)
        new_image = image.resize((size,size))
        data.append(np.array(new_image.convert('L')))

    except:
        #print(entry)
        del entries[entries.index(entry)]

data=np.array(data)
labels=np.zeros(40).tolist()
#The labeling part has been obtained by the order of the
    images
for i in range(41,len(data)+1):
    labels.append(1)
LABELS=[]
for i in range(len(labels)):
    LABELS.append(int(labels[i]))
labels=LABELS
classifier = Sequential()

# Step 1 - Convolution
classifier.add(Conv2D(3, (3, 3), input_shape = (size,size, 1)
    , activation = 'relu'))

classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Conv2D(3, (3, 3), input_shape = (size,size, 1)
    , activation = 'relu'))

classifier.add(MaxPooling2D(pool_size = (2, 2)))
classifier.add(Flatten())

#classifier.add(Dense(units = 32, activation = 'relu'))
```

```
classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN
#Accuracy has been computed using cross validation
ACC=[]
for i in range(10):
    classifier = Sequential()

    # Step 1 - Convolution
    classifier.add(Conv2D(3, (3, 3), input_shape = (size,size, 1), activation = 'relu'))

    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Conv2D(3, (3, 3), input_shape = (size,size, 1), activation = 'relu'))

    classifier.add(MaxPooling2D(pool_size = (2, 2)))
    classifier.add(Flatten())

    #classifier.add(Dense(units = 32, activation = 'relu'))
    classifier.add(Dense(units = 1, activation = 'sigmoid'))

    # Compiling the CNN
    classifier.summary()
    train_images, test_images, train_labels, test_labels = train_test_split(
        data, labels, test_size=0.15)
    train_images=np.array(train_images)
    test_images=np.array(test_images)
    train_images, test_images = train_images / 255.0, test_images / 255.0
    Train_images=[]
    Test_images=[]
    for i in range(len(train_images)):
        a=train_images[i].reshape(size,size,1)
        Train_images.append(a)
    Train_images=np.array(Train_images)
    for j in range(len(test_images)):
        b=test_images[j].reshape(size,size,1)
        Test_images.append(b)
    Test_images=np.array(Test_images)
    train_images,test_images=Train_images, Test_images
    classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    history = classifier.fit(train_images, train_labels, epochs=15,
                     validation_data=(test_images, test_labels),batch_size=10)
```

```
    ACC.append(history.history['val_accuracy'][len(history.
    history['val_accuracy'])-1])
#
```

5.5 Second algorithm Neural Network

: The practical implementation of the algorithm shown in figure 4.3 has been here reported:

```
#The images have been imported in the Python File
size=250
entries = os.listdir('/Users/pierohmd/Downloads/Solar Flares
    And Active Regions/')
entries.sort(key=natural_keys)
del entries[entries.index('.ipynb_checkpoints')]
del entries[entries.index('.DS_Store')]
data=[]
IMAGES=[]
for entry in entries:
    try:
        image = Image.open(entry)
        IMAGES.append(image)
        new_image = image.resize((size,size))
        data.append(np.array(new_image.convert('L')))

    except:
        #print(entry)
        del entries[entries.index(entry)]

data=np.array(data)
labels=[]
for entry in entries:
    SPLITTED=entry.split('.')
    filename=SPLITTED[0][0]
    if str(filename)=='A':
        labels.append(0)
    else:
        labels.append(1)

# Compiling the CNN
#Accuracy has been computed using cross validation
ACC=[]
for i in range(10):
    classifier = Sequential()

    # Step 1 - Convolution
    classifier.add(Conv2D(3, (3, 3), input_shape = (size,size
    , 3), activation = 'relu'))
```

```
38
39      classifier.add(MaxPooling2D(pool_size = (2, 2)))
40      classifier.add(Conv2D(3, (3, 3), input_shape = (size,size
        , 3), activation = 'relu'))
41
42      classifier.add(MaxPooling2D(pool_size = (2, 2)))
43      classifier.add(Flatten())
44
45      #classifier.add(Dense(units = 32, activation = 'relu'))
46      classifier.add(Dense(units = 1, activation = 'sigmoid'))
47
48      # Compiling the CNN
49      classifier.summary()
50      train_images, test_images, train_labels, test_labels =
        train_test_split(
51          data, labels, test_size=0.15)
52      train_images=np.array(train_images)
53      test_images=np.array(test_images)
54      train_images, test_images = train_images / 255.0,
        test_images / 255.0
55      Train_images=[]
56      Test_images=[]
57      for i in range(len(train_images)):
58          a=train_images[i].reshape(size,size,3)
59          Train_images.append(a)
60      Train_images=np.array(Train_images)
61      for j in range(len(test_images)):
62          b=test_images[j].reshape(size,size,3)
63          Test_images.append(b)
64      Test_images=np.array(Test_images)
65      train_images,test_images=Train_images, Test_images
66      classifier.compile(optimizer = 'adam', loss = '
        binary_crossentropy', metrics = ['accuracy'])
67      history = classifier.fit(train_images, train_labels,
        epochs=15,
68                       validation_data=(test_images, test_labels
        ),batch_size=10)
69      ACC.append(history.history['val_accuracy'][len(history.
        history['val_accuracy'])-1])
70  #
```

6 Results

In the previous works [4][15], complex features have been hand designed to obtain useful information from the image. This approach may be incautious for the following reasons:

- The performance of the algorithms crucially depends from the designing of the features.
- An high domain knowledge is required, and sometimes even the highest domain knowledge could be insufficient when the task is too complicated.

In this work **Convolutional Neural Networks** has been used to detect solar flares directly from raw pixels of images. **BBSO**, **Space Weather Archive** and **LMSAL** data has been used together with these two different algorithms:

- The First Neural Network is trained to distinguish whether or not the Sun has active region by analyzing full disk magnetogram images (HMI).
- The Second Neural Network is trained to distinguish whether or not the active regions have solar flares by analyzing specific regions of the Sun itself (AIA).

For both the algorithms, **K-fold cross validation** has been used with $K = 10$, obtaining 94.7% of accuracy for the first algorithm and 96.5% accuracy for the second one. As a test, the first algorithm has been used for the second task, obtaining comparable but slightly inferior performance with respect to the second one (95.8% of accuracy). Several web scraping techniques have been adopted to extract, analyze, and compare datasets.

After a brief overview of what Deep learning and Machine Learning are, how data can be used to gain information, and how the training of the parameter is performed, Convolutional Neural Network are described. Subsequently, the datasets extraction and sources have been explained. In section number 4, the algorithms, starting from the row data and ending with the binary classification, has been specifically explained. The software specifics and the codes used to implement the web scraping techniques and the networks have been finally reported in sectio n.5.

References

[1] Syed Muhammad Anwar et al. "Medical image analysis using convolutional neural networks: a review". In: *Journal of medical systems* 42.11 (2018), p. 226.

[2] BBSO. *BBSO archive*. URL: ftp://ftp.bbso.njit.edu/pub/archive/2019/04/27/.

[3] BBSO. *BBSO archive*. URL: ftp://ftp.bbso.njit.edu/pub/archive/2013/05/01/.

[4] Roberto A Fernandez Borda et al. "Automatic solar flare detection using neural network techniques". In: *Solar Physics* 206.2 (2002), pp. 347–357.

[5] Scott Fortmann-Roe. *Train Test Split*. URL: http://scott.fortmann-roe.com/.

[6] Giorgio Gambosi. *Deep Learning*. URL: https://tvml.github.io/ml1920/note/deep-notes.pdf.

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

[8] Cheang Weng Hang. *Neural Network with BackPropagation*. URL: https://mc.ai/an-introduction-to-gradient-descent-2/.

[9] James R Lemen et al. "The atmospheric imaging assembly (AIA) on the solar dynamics observatory (SDO)". In: *The solar dynamics observatory*. Springer, 2011, pp. 17–40.

[10] Alexander Lenail. "NN-SVG". In: *NN SVG, alexlenail. me/NN-SVG* (2018).

[11] Petrus C Martens and Cornelis Zwaan. "Origin and evolution of filament-prominence systems". In: *The Astrophysical Journal* 558.2 (2001), p. 872.

[12] Nasa. *Sun erupts with significant flares*. 2017. URL: https://www.nasa.gov/feature/goddard/2017/active-region-on-sun-continues-to-emit-solar-flares.

[13] O'Reilly. *Chapter 4. Major Architectures of Deep Networks*. URL: https://www.oreilly.com/library/view/deep-learning/9781491924570/ch04.html.

[14] Solar Cycle Progression. "Space Weather Live". In: *Modified image. url: https://www. spaceweatherlive. com/en/solar-activity/solar-cycle (visited on 08/22/2018)* (2019).

[15] Ming Qu et al. “Automatic solar flare detection using MLP, RBF, and SVM”. In: *Solar Physics* 217.1 (2003), pp. 157–172.

[16] Frank Y Shih[1] et al. “Application of Machine Learning in Solar Physics”. In: (2008).

[17] Charis Sng. *An introduction to Gradient Descent*. URL: https://mc.ai/an-introduction-to-gradient-descent-2/.

[18] Lockheed Martin Solar. *Astrophysics Laboratory (LMSAL)—Science—Last Events—Solar Soft*.

[19] Ria Stroes. *Creative Coding Course Lesson 2: Black and White*. URL: http://riastroes.nl/CCC/lesson2.html.

[20] ML Tut. *K Fold Cross-Validation in Machine Learning? How does K Fold Work?* URL: https://www.mltut.com/k-fold-cross-validation-in-machine-learning-how-does-k-fold-work/.

[21] Wikipedia. *Rete neurale artificiale*. URL: https://it.wikipedia.org/wiki/Rete_neurale_artificiale.