University of Amsterdam

Faculty of Science

Bsc Bèta-Gamma

Major Physics

Report Bachelor Project Physics and Astronomy

Using Convolutional Neural Networks for Solar Wind Classification

by

Marleen Rijksen

10465030

18 April 2017

15 ECTS

Research conducted between 06-02-2017 and 18-4-2017

Supervisor:
Dr. Enrico Camporeale

Second assessor:
Dr. Anna L. Watts

Center for Mathematics and Computer Science

Contents

1 Summary

The prediction of space weather is important because it can cause hazardous effects on (costly) human technology and health. The term space weather refers to physical processes created by the Sun that affect human life. In this research an attempt was made to train a convolutional network on solar images to classify different types of solar wind. Solar wind is a stream of plasma and charged particles emitted by the sun. To predict the effects solar wind can have it is important to know what kind of solar plasma is carried towards earth by solar winds. A four plasma-type categorization scheme is used to distinguish the different solar plasma types. Images of EIT-171 aboard the Solar and Heliospheric Observatory-satellite were used to train the top layers of the pre-trained Inception-V3 network to classify the images into the four categories. Each of the categories is tested against the other three categories. It is found that the network could identify the different categories with accuracies ranging from 51% to 80%. Although these accuracies are not reliable enough for an accurate prediction system, taking into account the limitations of this project these results are promising for further research.

2 Popular scientific summary (Dutch)

Een algemeen bekend fenomeen is de Aurora Borealis, oftewel het noorderlicht. Dit noorderlicht is voornamelijk waar te nemen op hoge geografische breedtes en zorgt voor spectaculaire lichtshows. Het licht is het gevolg van zonnewind, een constante stroom geladen deeltjes afkomstig van de zon. Wanneer tijdens hevige zonnewind de geladen deeltjes de polen van de aarde bereiken kan men het noorderlicht waarnemen (of op de zuidpool, het zuiderlicht).

Deze zonnewind heeft niet enkel het noorder- of zuiderlicht tot gevolg. De effecten van zonnewind op en rondom de aarde kunnen nadelig zijn. Zonnewind kan gevaarlijk zijn voor elektronica. Denk hierbij aan satellieten, navigatiesystemen, elektriciteitswerken en radiocommunicatie. In 1859 was door het gevolg van hevige zonnewind de telegraafverbinding tussen Amerika en Europa uitgevallen en diverse branden werden hierdoor veroorzaakt. Naast elektronische problemen kunnen astronauten die niet beschermd worden door atmosfeer rondom de aarde zeer grote risico's lopen op het gebied van gezondheid.

Door bovengenoemde problemen zou het handig zijn als men kan voorspellen wat voor type zonnewind er op komst is. Op die manier kunnen er voorzorgsmaatregelen genomen worden om de nadelige gevolgen te beperken. Een van de manieren om dit te doen is door de satellietbeelden van de zon te analyseren. De Solar and Heliospheric Observatory (SOHO)-satelliet heeft instrumenten aan boord die de gehele dag beelden maken van de zon. Op deze beelden is te zien hoe actief de zon is en aan de mate van activiteit kan men zien welk type zonnewind op komst is. Deze beelden kunnen bekeken worden nog voordat de zonnewind op aarde aankomt gezien zonnewind ongeveer 50 tot 100 uur onderweg is van de zon naar de aarde.

Beeldherkenning kan worden gebruik om de satellietbeelden te analyseren. Beeldherkenning wordt tegenwoordig in veel toepassingen gebruikt, zelfs in smartphones (gezichtsherkenning). Om beeldherkenning mogelijk te maken worden voornamelijk 'convolutional neural networks' gebruikt. Dit zijn computer algoritmes die onder andere in staat zijn om afbeeldingen te categoriseren. In dit project is zo een algoritme gebruikt om de satelliet afbeeldingen te categoriseren in vier verschillende zonnewind categorieën. Het is belangrijk om de zonnewind te categoriseren aangezien de effecten van de verschillende soorten zonnewind ook verschillen.

Een 'convolutional neural network' is een bepaald soort kunstmatig neuraal netwerk. Deze kunstmatige neurale netwerken zijn computer modellen die gebaseerd zijn op de werking van het brein. Het interessante van deze netwerken is dat ze kunnen leren. De manier waarop deze netwerken leren is aan de hand van voorbeelden. Als een netwerk bijvoorbeeld geschreven getallen moet leren herkennen, wordt er een groot aantal afbeeldingen met labels 'gegeven' aan het netwerk. Deze labels bevatten informatie over wat er in de afbeelding staat, in dit geval het getal wat er op de afbeeldingen wordt weergegeven. Door het netwerk te trainen is het uiteindelijk in staat om afbeeldingen zonder labels te classificeren.

In dit onderzoek is een 'convolutional neural network' getraind op afbeeldingen van de zon met vier verschillende soorten labels. Deze labels bevatten de verschillende types zonnewind. Afbeeldingen van elke soort zonnewind zijn getest tegenover afbeeldingen uit de drie overige soorten zonnewind. Omdat het erg veel tijd kost om een heel netwerk te trainen is er in dit project gebruikt gemaakt van een netwerk dat al getraind is op een andere dataset. Enkel het laatste gedeelte van het netwerk is op onze dataset getraind. Het gebruikte netwerk in dit project heet Inception-V3, dit netwerk kon met de ImageNet classificatieset afbeeldingen uit 1000 categorieën categoriseren met een top-5 nauwkeurigheid van 94.4%. Het door ons getrainde netwerk kon de afbeeldingen categoriseren met nauwkeurigheden variërend van 81% tot 51%. Gezien de korte duur van dit project en de daardoor nog mogelijke verbeteringen zijn dit veelbelovende resultaten voor verder onderzoek.

3 Introduction

The term space weather refers to the physical processes created by the Sun, that have an effect on human technology and health on Earth and in space. Solar plasmas like coronal Mass Ejections can scatter large amounts of radiation and charged particles towards the Earth. The stream of charged particles released by the Sun is called solar wind. The Earth environment and geomagnetic activity is affected by the solar wind (Qahwaji & Colak 2007). Common effects that can arise due to space weather are current surges in power lines, radiation hazards in the polar regions and spacecraft anomalies due to surface charging (Feynman & Gabriel 2000). Because of these effects it is important to be able to predict space weather so that precautions can be taken, such as temporarily disabling sensitive satellite systems for the duration of the solar flare.

In order to predict space weather it is important to know which kinds of solar plasmas can be produced by the Sun. When solar wind plasma is categorized it is usually divided in two categories, either "fast wind" or "slow wind" depending on the speed of the wind (Dasso et al. 2005). In this research a four plasma-type categorization scheme developed by Xu & Borovsky (2015) is used. An explanation about this categorization scheme and why it is used in this research can be found in section 4.3.

Solar images can help to predict which kind of solar plasma is heading towards Earth. Instruments on satellites can for example observe light with different wavelengths emitted by the Sun and capture images. These images contain, amongst other things, information about solar wind. When certain solar wind events are observed, depending on the speed of the solar wind, it will take approximately 50-100 hours before this solar wind reaches Earth (Feldman et al. 2005). Solar wind can thus be observed before it arrives on Earth. In this research images captured by the EIT-171 instrument aboard the SOHO (Solar and Heliospheric Observatory)-satellite were analysed.

One of the tools which can be used to classify images are Artificial Neural Networks (ANN). Articifial neural networks are information processing computational models inspired by the biological brain. These networks can be used, for example, to classify datasets like images or audiofiles. ANNs can learn by examples, just like the human brain. In short, an ANN can learn from a dataset to classify objects not included in the dataset. Special types of ANNs are Convolutional Neural Networks (CNNs). As opposed to other types of networks, CNNs treat the input data as volumes (instead of vectors) and take into account the spatial structure of the data. CNNs also use fewer parameters then standard ANNs meaning that they are easier to train on larger datasets. CNNs have proven to achieve higher accuracies than other types of ANNs at tasks like image recognition (Krizhevsky et al. 2012). Because of the state-of-the-art results of CNNs on image recognition tasks in this research this type of network is used to classify the different types of solar wind.

Research on CNNs has made a lot of progess. The Inception-V3 network achieved for example a 5.6% top-5 error on the ImageNet test set (consisting of 150,000 images labeled with 1000 categories) (Szegedy et al. 2016). Because of the need to predict space weather and the developments in CNNs in this research the two are combined. The top-layers of the inception-V3 network where trained on a dataset consisting of solar images with four different categories (chapter 5). With this research a foundation is laid for further research towards using CNNs for solar wind classification. A general overview of neural networks is given in section 4.1, followed by a detailed discussion of one particular set of neural networks in section 4.2. The categorization scheme used in this project is discussed in section 4.3. A pre-existing trained neural network is used to analyse SOHO data in section 5. Finally, the results and discussion are discussed in section 6 and 7 to draw conclusions in section 8.

4 Related work and Background Material

In this section general information about neural networks is given. The historical background of neural networks is discussed and after that some knowledge about basic neural networks is provided. Following is a section about a special type of neural networks: convolutional neural networks.

4.1 Neural networks in general

Tasks like image recognition are very hard to solve with handcoded rules. An example of a domain in which handcoded rules fail to work properly is face recognition. A human can easily recognize a set of pictures of faces, but the ordering of the pixels between the pictures is very inconsistent. It is very difficult to write handcoded rules to tell the computer where and how to look for certain features in the image. These features are different for each image and it would take an impossible amount of handcoded rules to tell the computer all the possible pixel orientations in which a face is represented. Instead of using handcoded rules it is better to use artificial neural networks for tasks like face recognition (Muller & Guido 2017). CNNs are a special type of artificial neural network and have proven to achieve the best results in image recognition tasks.

It is important to first understand the basics of artificial neural networks in order to learn how CNNs work. In the first part of this chapter an overview is given about how single (artificial) neurons work, how a neural network is designed and how neural networks can be trained. In the second part of this chapter an introduction to Convolutional Neural Networks is provided.

4.1.1 Historical background

McCulloch & Pitts (1943) tried to understand how (complex) neural networks in the brain work using the building blocks of the brain: neurons. They made a simplified model of a neuron and showed that articifical neural networks could be used to compute arithmetic or logical functions.

Rosenblatt (1958) was inspired by the work of McCullock and Walter Pitts and invented the so called perceptron model. This network consisted of a type of artificial neurons (called perceptrons) and was able to perform pattern recognition. Widrow and Hoff also developed a simple neural network similar to the one of Rosenblatt (Widrow et al. 1960). Unfortunately, both of these networks suffered from limitations and they were not able to train more complex networks. The research towards neural networks became stagnated. One of the limitations of these networks were that the neurons could not solve "exclusive or" or "exclusive nor" functions. It was around 1980 that research in neural networks increased again because of the computational possibilities at that time and the discovery of backpropagation algorithms to train networks (Demuth et al. 2014).

In a different field of science, Hubel & Wiesel started to study the behaviour of neurons in the visual cortex of cats in 1959. They discovered that the visual cortex has an hierarchical organization ranging from simple cells towards hyper complex cells (Wiesel & Hubel 1959). The cells in the visual cortex respond to visual inputs such as the orientation of edges or colors. They discovered that the complex cells extracted more complex information from the visual inputs than the simple cells. This hierarchical organization and the extraction of increasingly complex features inspired CNNs (Schmidhuber 2015).

4.1.2 How neural networks work

Biological Anology Artificial neurons are a simplification of real neurons in the biological brain. The brain is build up of approximately 10^{11} neurons, all higly connected with each other. The basic parts of neurons are the cell body, the dendrites, the axon and synapses. The dendrites of a neuron receive input from other neurons, which are then summed by the cell body. If the sum of these input signals exceeds a certain treshold, the cell body fires a signal through its axon. The neurons receive input signals via the synapses. A synaps is the connection between the axon of a neuron and the dendrite of another neuron. The strength in which the signals are passed is controlled by a complex chemical process. The function of the neural network is thus established by the connections between the neurons and the strength with which signals are passed through the network (Demuth et al. 2014). In figure 1 a schematic overview of two neurons is given.

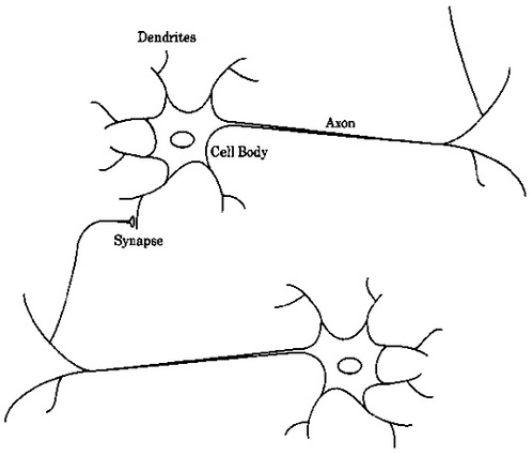

Figure 1: Schematic of two biological neurons where the cell body, the axon, the dendrites and the synapse are displayed (Demuth et al. 2014).

Artificial neurons Just like biological neurons, artificial ones receive inputs from other artificial neurons. The inputs are then evaluated, not by a cell body but by activation functions. Depending on the activation function and the input, a signal with a certain strength is passed to other articifical neurons.

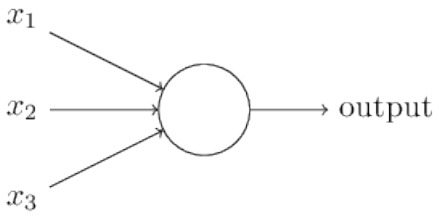

Figure 2: Schematic of an artificial neuron. x_1, x_2 and x_3 are the inputs of the neuron. The output is based on which type of activation is used in the neuron.

As in biological neurons, not all inputs of an artifical neuron are as important as other inputs. Each input is multiplied by weights, $w_1, w_2, ..., w_n$, to express the importance of input to the output. Also, a bias, b, is added to shift the activation function to the left or to the right, it is a measure of how easy it is to get a higher (or lower) signal. There are different activation functions which can be used in artificial neurons. One of these (historically often used) is the sigmoid function,

$$\sigma = \frac{1}{1 + e^{-x}} \tag{1}$$

where σ is the output and x is the input of the neuron. Taking all the weights and biases into account gives us

$$\sigma = \frac{1}{1 + \exp(-\sum_i w_i x_i - b)} \tag{2}$$

with inputs x_i, weights w_i and a bias b. Note that the bias is a horizontal shift in the activation function of an artificial neuron and that the same bias b is applied to each input. In figure 3 a plotted version of the sigmoid function is provided. When the input of the sigmoid function is a large positive number the output of the neuron is approximately 1, if the input is a large negative number the output is approximately 0. Beside the sigmoid function there are other activation functions as well, among them are for example $f(x) = \tanh(x)$, ReLu: $f(x) = \max(0, x)$ and $f(x) = x$.

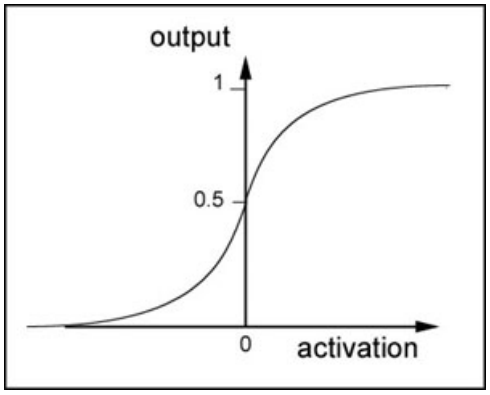

Figure 3: Plotted sigmoid function

Networks of artificial neurons Individual articifial neurons can be connected with each other to form a neural network. Neural networks consist of three different layers: an input layer, hidden layers and an output layer. All of the neurons of the previous layer are connected to neurons in the next layer. The advantage of using this architecture is that it allows to use vector and matrix multiplication (instead of treating every value seperately). The input would, for example be a vector containing all the pixel information and the weights would be ordered in a matrix. When this matrix W is multiplied with the input vector, a new vector is created which then again is multiplied with new weights in the next layer. In the last layer a product between a weight matrix and vector should output a new vector representing the scores for each class.

Neural networks can have different types of input data, ranging from images to audio files. An application of a simple neural network could for example be to recognize handwritten digits. The input would be a 28 x 28 image and the output would be an indication about which digit the network thinks the image represents. In this case the input layer would have 28 x 28 = 784 neurons, where the input of each neuron is a pixel value of the image (we assume that the image is greyscale and that each pixel corresponds to a value indicating the intensity of greyness). There can be multiple hidden layers and the output layer could have 10 neurons. When an image with the number 5 on it would be fed to this network, the 5th neuron of the output layer should give, for example, the highest output of all output neurons. Then

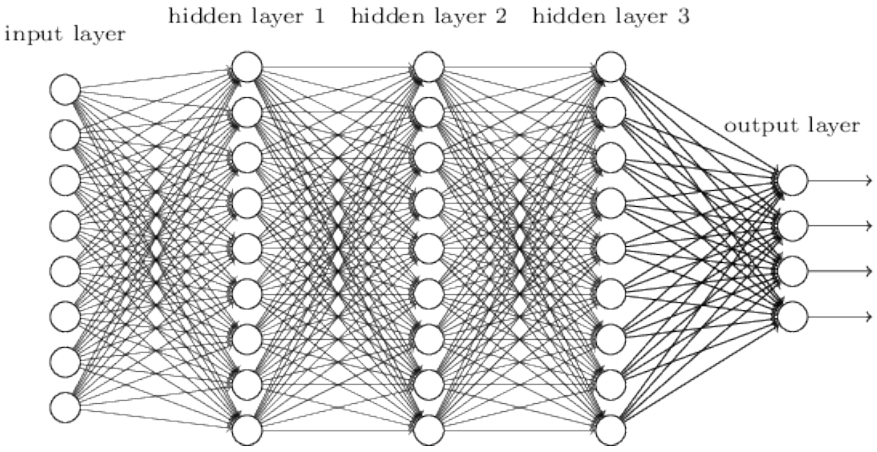

Figure 4: Example of a 5-layered fully connected neural network. This network consists of an input and an output layer and has three hidden layers. (Nielsen 2015)

the network 'thinks' the probability that the given image represents a 5 is higher than the probability the image represents another number. Whether a neural network correctly handles these images and gives the correct output signal depends on the weights and biases of all the neurons (Nielsen 2015). How these weights and biases are determined is explained in the next paragraphs.

Softmax Classifier The output of the neural network described above is difficult to interpret. To handle the output of the last layer of the network classifiers can be used to convert the outputs into, for example, probabilities. The Softmax classifier turns the output of the neural network into normalized class probabilities and it treats the output values as unnormalized log probabilities for each class (where the classes are numbers from 0 to 9, in the example described above). The conversion from arbitrary numbers to probabilities is done with the Softmax function,

$$f_i(y_i, x_i) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{3}$$

where y_i is the correct label given the image x_i, f_{y_i} is the class score for the correct label and f_j is the j^{th} element of the vector containing the class scores. This function returns a vector of values between 0 and 1 which can be interpreted as normalized probabilities.

Training of neural networks When using neural networks it is important to choose the architecture of the network (for example the number of neurons in each layer or the number of layers). When an architecture is chosen, the network needs to 'learn' the correct weight parameters in order to give the desired output. The input of a network is fixed, but the weights can be changed in order to make the network perform better.

An important term when training networks is the so called loss, which gives a measure of how well the network is performing. When a network is trained it is given input in the form of images (or other data types, like audio files) with corresponding labels. These labels contain the correct output, in our digit example the label would be one of the ten digits. When a network returns a certain output, this output is compared to the correct output via the loss function. There are different types of loss functions, the softmax classifier uses cross-entropy loss. Loss functions give the network a measurement of how well it performs, when the loss is

high the network probably does not give the desired results. The cross-entropy loss is described by

$$L_i = -\log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} \tag{4}$$

where f_{y_i} is the class score for the correct label and f_j is the j^{th} element of the vector containing the class scores (as described in formula 3). The output of this function is the negative log likelihood of the correct class. We want this function to be minimized (or, we want the log likelihood to be maximized). In other words, we want to train our network in such a way that the loss is minimized and that the output value gives the highest probability for the correct class. A way of calculating the minimum of the loss function would be to analytically find the minimum of the loss function by computing the derivatives of the loss function with respect to the weights. This would quickly become very difficult though, in neural networks there are often many variables on which the loss function depends. Instead of analytically determining the minimum of the loss function, the backpropagation algorithm is used.

Backpropagation In neural networks, the backpropagation algorithm is used to change the weights and biases in such a way that the loss function gets minimized. This backpropagation algorithm is introduced by Rumerhart et al. (1986). Backpropagation is a recursive application of the chain rule which computes gradients throughout the whole network. When an input image is propagated through the network we get an output score. With the help of this output score the loss can be calculated, we want this loss to be minimized. The loss is a function of the input data and the weights and biases of the neurons. The gradients of the loss function, with respect to the weights, can be calculated for each neuron to determine the contribution each neuron has on the loss. When each contribution is known the weights are updated in such a way that the network attempts to minimize the loss function. When training a network data gets propagated through the network multiple times and weights will be updated according to the outcomes of the loss function.

There are different ways to update the weights, among them is the RMSprop method (Tieleman & Hinton 2012). RMSprop is an optimizer which is not introduced in a publication, yet it is often used and shows to be a favorable optimizer. In this research the inception-V3 network is used (section 5.2) and this network is trained with the RMSprop optimizer as well (Szegedy et al. 2016). Although a mathematical derivation of this optimizer is not provided here it is important to know what the concept of learning rate is. The learning rate of a network is a parameter that controls the amount by which the weights and biases are changed. When the learning rate is too high, the network cannot converge to a certain accuracy. When the learning rate is too low it might take a lot of time before the network converges to its optimal accuracy.

4.2 Convolutional Neural Networks

A neural network as described above would need a lot of weights if the input would be a much larger (and colored) image. If the input would be a 300 x 300 x 3 (where the 3 stands for the three color channels: red, green and blue) each neuron in the first hidden layer would have $270,000$ weights. If a network has a few layers containing neurons with this many weights, the computational costs would become huge. CNNs assume the input is a volume and instead of 'squashing' the pixels of the image into a vector it treats the images as a volume. The layers of a convolutional neural network are also arranged in three dimensions (figure 5). Instead of fully connecting all the neurons, in CNNs the neurons are only connected to a small part of the layer before it. Every layer in the network transforms the input volume in a three dimensional output volume, containing neural activations.

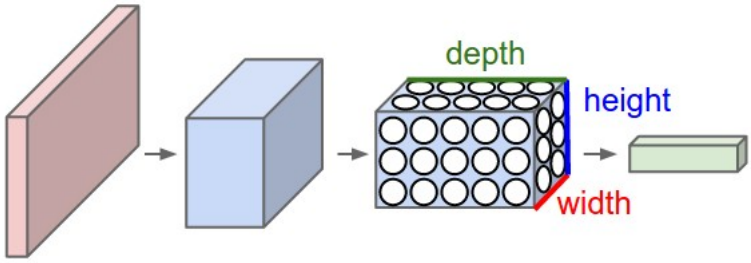

Figure 5: Schematic overview of a convolutional neural network. The neurons are arranged in three dimensions and in each layer of the network an input volume is transformed into an output volume of neuron activations. The red input layer represents the image, with a certain width and height depending on the dimensions of the image and the depth could for example be three (red, green and blue color channels) (Li et al. 2016).

Layers A simple CNN would have about 5 different types of layers. The input layer would represent the input image, for simplicity, assume the input is a 28x28x1 (greyscale) image. A convolutional layer computes dot products between their weights and a small part of the input volume. Unlike ordinary neural networks, not all pixel values will be connected to all neurons in the first layer. Instead, connections are made in small and localized regions of the input volume. The size of these localized regions is called the local receptive field for the hidden neuron, for example 5x5x1. The first neuron in the first hidden layer will be connected to a local receptive field (figure 6). The second neuron in the first hidden layer will also be connected to a 5x5x1 field. If we shift the local receptive field 1 pixel to the right, we can see the local receptive field for this second neuron (figure 7).

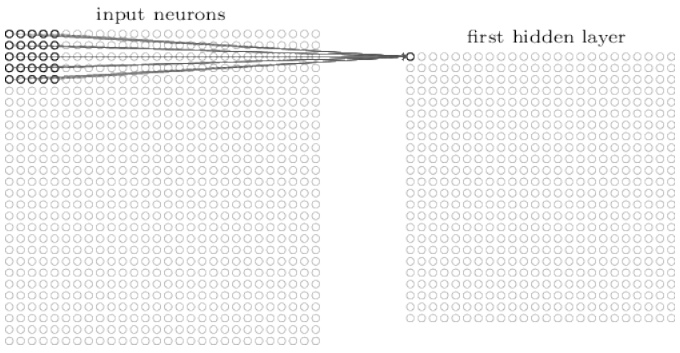

Figure 6: Local receptive field for the first neuron (Nielsen 2015).

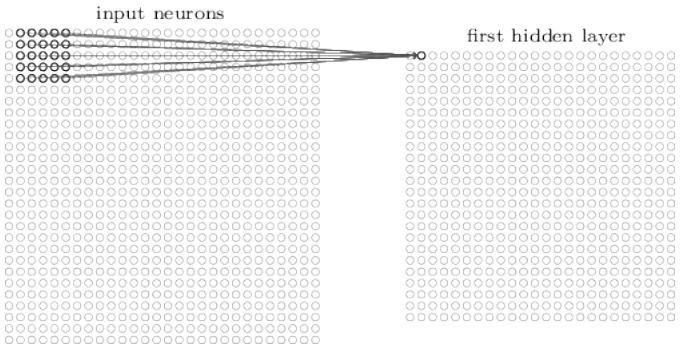

Figure 7: Local receptive field for the second neuron (Nielsen 2015).

This shifting can be done for each neuron in the first hidden layer. The amount of pixels by which this local receptive field is slided is called the stride which could also be 2 or 3 for example. All these connections between the local receptive field and the neuron represent dot products between the values of the input volume and the weights of that connection. This 5x5x1 local receptive field can be moved 23 times before the right side of the input volume is reached. The receptive field can also move down 23 times in the same way as the horizontal shift. Therefore, the first hidden layer consists of 24x24 neurons. Each neuron in a certain layer has shared weights and biases, so for the i, jth neuron in the hidden layer the output of the activation function will be

$$\sigma = f(b + \sum_{l=1}^{5} \sum_{m=1}^{5} w_{l,m} x_{i+l,j+m}) \tag{5}$$

where σ is te output, $w_{l,m}$ represents the 5x5x1 array of shared weights, b is the shared value of the bias and $x_{j+l,k+m}$ is the input volume activation at the position $i+l, j+m$. The output function can be any activation function, for example the sigmoid function. Each neuron 'looks' at a different part of the input volume, but they have shared weights and biases. The advantage of sharing parameters is that the total number of parameters is greatly reduced, which reduces computational cost. This sharing means that each neuron in a hidden layer extracts the same features from the input, only at different locations. Such a feature could for example be a horizontal edge. Then each neuron looks at each local receptive field whether it detects a horizontal edge. The hidden layer represents a new map, this map from input layer to hidden layer is called a feature map. A convolutional layer often contains more than one feature map (figure 8). With the setup described as in figure 8 the network can detect 3 different features throughout the whole input volume, the output volume of this layer is a volume with size 24x24x3.

Another type of layer is called a pooling layer. In a pooling layer the input volume is down-sampled along the spatial dimensions. It is often the case that pooling layers are periodically inserted after convolutional layers in a network. Pooling layers reduce the amount of parameters (and thus computational cost) and control overfitting. There are different types of pooling layers, among them max pooling layers. Max pooling layers output the maximum activation in a certain input region, for example a 2x2 region. When this kind of pooling is applied to our 24x24x1 output layer we get a new map of 12x12x1 neurons. A schematic overview of max pooling applied to each feature map as described above is given in figure 9.

Although it might look as if information is thrown away, pooling layers act as a way to distill information. The pooling layer does not throw away information about whether a given feature is found anywhere in the input volume, it only throws away the exact positional information.

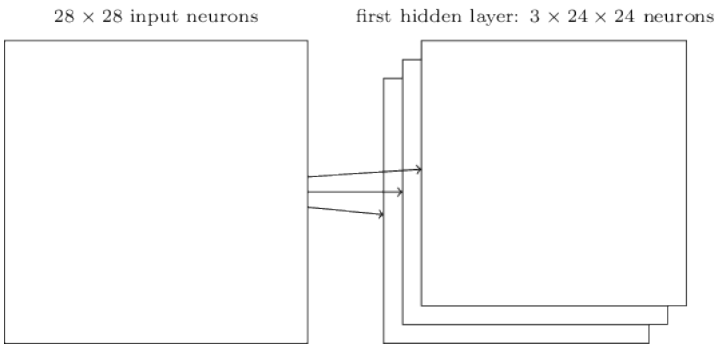

Figure 8: Convolutional layer with 3 feature maps. The input is a 28x28 greyscale image, which is then transformed to three feature maps by the first hidden layer which consists of 3x24x24 neurons (Nielsen 2015).

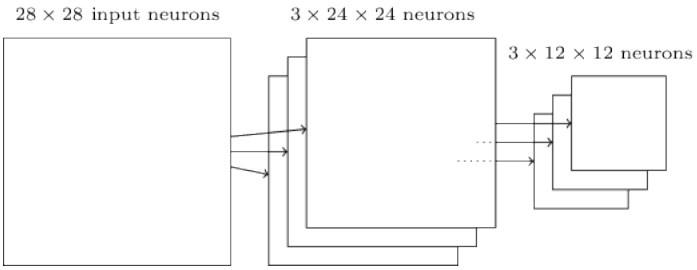

Figure 9: The max pooling receives an input of 3x24x24 which is then transformed to a volume of 3x12x12 (Nielsen 2015)

.

The positional information relative to other features is more important than the exact location, this relative positional information is not thrown away. Another type of pooling which is frequently used is L2 pooling. L2 pooling uses the square root of the sum of the squares of the activations in a certain region. Like max pooling, L2 pooling distills information from the previous layer.

The last layer(s) of a CNN are fully connected layers which work in the same way as the ordinary neural networks described earlier. In the last fully connected layer class scores are computed for each category.

4.2.1 Training a Convolutional Neural Network

Although the architecture of a CNN is quite different than that from an ordinary neural network, it is still made out of simple neurons with weights and biases and the overall goal is the same. Just like ordinary neural networks, this network needs to be trained using backpropagation. When training a CNN the overall process is the same as in ordinary neural networks. Data is propagated through the network and depending on the loss the parameters of the network are updated. Backpropagation works slightly different in CNNs, the modifications are mainly about the fact that CNNs do not have fully connected layers.

Transfer learning Training a CNN requires a lot of data and computational power. Fortunately, it is not necessary to train a CNN from scratch and pre-trained networks are available. Pre-trained networks are networks which are already trained on a large dataset, for example the ImageNet dataset (Deng et al. 2009). These pre-trained networks already 'learned' the best

parameters and weights in order to identify images as accurate as possible. Features extracted by CNNs which are already trained are proven to be usable for other types of categorization (Sharif Razavian et al. 2014). The idea of working this way is that a CNN can act as an extractor of mid-level image representations, which can be pre-trained on one dataset and then re-used on other datasets and tasks (Oquab et al. 2014). To train the network on a new dataset, the last fully connected layers can be removed and trained on the new dataset. When the dataset is medium sized part of the convolutional layers could be trained as well. Not training all the layers of (deep) networks saves a lot of computational power and time.

4.3 Solar Wind Categorization Scheme

So far we have discussed neural networks to later apply them to solar observations to predict space weather. In the current section we briefly digress to discuss different classifications of solar wind. In order to predict the consequences space weather can have on Earth, it is important to categorize the different types of solar plasma emitted by the Sun. There are multiple ways to categorize solar wind. In this research four categories are used to distinguish different types of solar wind. This categorization scheme is proposed by Xu & Borovsky (2015). The categories are based on a three-parametric algebraic scheme based on measurements of solar wind proton density and temperature, the solar wind speed and the solar wind magnetic field strength. The measurements needed to apply this scheme are the proton number density n_p, the proton temperature T_p, the magnetic field strength B and the solar wind speed v_{sw}. An advantage of this type of categorization is that it does not require measurements of heavy ions of the solar wind, as opposed to other schemes.

The four plasma categories of the scheme are coronal-hole-origin plasma, streamer-belt-origin plasma, sector-reversal-region plasma and ejecta. The categorization scheme was developed by using collections of known solar wind plasma types, such as unperturbed coronal hole plasma from constant velocity high-speed streams, published magnetic clouds and collected pseudostreamers. This categorization scheme is applied to the OMNI2 dataset ranging from 1963 - 2013. This dataset contains hourly resolution solar wind magnetic field and plasma data from multiple spacecrafts around the Sun. It also contains data about fluxes of energetic protons, sunspot numbers and geomagnetic activity indices.

The scheme was assessed against the known collections of the plasma types described above. The fraction of correctly categorized points for ejecta was 87.5%, for coronal-hole-origin plasma 96.9%, for streamer-belt-origin plasma 69.9% and for sector-reversal-region plasma 72.0%. Note that no solar wind categorization is exact because there is a lack of knowledge about the exact origins of solar wind from the Sun and of how the plasma origin behaves within the solar wind. Still this categorization scheme is found to be quite accurate and an improvement compared to other categorization schemes.

5 Description of work

The previous sections addressed (training of) neural networks and the categorization scheme. This section describes how we apply the pre-existing trained neural network to SOHO images to predict space weather. We first describe the dataset used in this research.

5.1 Solar and Heliospheric Observatory Dataset

The used solar images in this project came from the EIT-171 instrument aboard the SOHO-satellite. This instrument is used to obtain solar images of the Sun in the 171 Ångstrom bandpass (Fe IX/X) (figure 10). Images captured by the EIT-17 instrument show the solar corona at a temperature of approximately 1.3 million K. There are three other EIT instruments which image the Sun in other selected bandpasses. The EIT instruments can image active regions, coronal holes and a variety of other solar features. An overview of dates on which the Sun emitted the four different solar plasmas was provided by dr. E. Camporeale. Based on these dates the images were downloaded from the Virtual Solar Observatory. In this way a database was created with solar images matching a certain solar plasma category. The original images had a size of 512 x 512 pixels and were resized to a size 299x299 pixels since the inception V3 network was pre-trained on images with this resolution as well (Szegedy et al. 2016). Note that only the channel with a bandpass of 171 Ångstrom is used in this research.

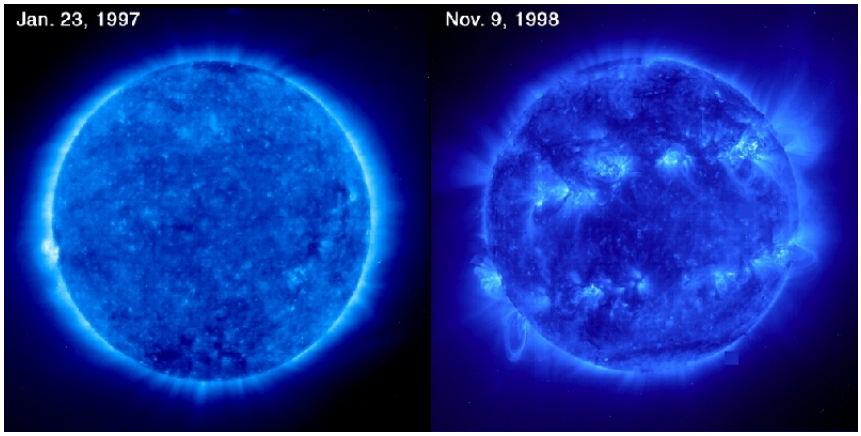

Figure 10: Two images captured by the EIT-171 instrument, taken almost two years apart. In the right image more active regions and magnetic loops can be observed (Solar and Heliospheric Observatory, 2017).

5.2 Architecture of the network

In this research a pre-trained network is applied to the solar images dataset. There are a lot of pre-trained networks available, most of them trained on the ImageNet dataset (Deng et al. 2009). The Inception-V3 network is a CNN published in 2016 which is benchmarked on the ILSVRC 2012 classification challenge validation set. The network achieved a 21.2% top-1 error (Szegedy et al. 2016). Although this network is deep (it consists of 42 layers), the computational costs are relatively low compared to other networks with a similar performance. They replaced part of the convolutional layers with large filters (for example 5x5 or 7x7 filters) by other mini networks with less parameters while keeping the input volume and output volume are the same. A 5x5 convolution is 2.78 times more computationally expensive than a mini network with

Table 1: Overview of the Inception-V3 network as proposed by Szegedy et al. (2016). The output size of each module is the input size of each next one. Included are the inception modules discussed in this section which reduce the amount of parameters and computational costs.

Layer type:	Input size:
conv	299x299x3
conv	149x149x32
conv padded	147x147x32
pool	147x147x64
conv	73x73x64
conv	71x71x80
conv	35x35x192
3x Inception module	35x35x288
5x Inception module	17x17x768
2x Inception module	8x8x1280
pool	8x8x2048
linear	1x1x2048
softmax	1x1x1000

smaller filters which can replace the 5x5 convolution. These mini networks are called inception modules. An overview of the architecture is provided in table 1.

5.3 Training Methodology

Because of its simple interface and easy prototyping in this research Keras is used as application programming interface (Chollet 2015). Keras is a deep learning library (written in python) which runs on top of either Tensorflow or Theano. In this research TensorFlow is used as backend. TensorFlow is an open source library used for machine learning (Abadi et al. 2015). The InceptionV3 network with weights from the ImageNet could be loaded from the Keras library.

Since this network was pre-trained on the ImageNet dataset it was not necessary to train all the layers of the network. The fully connected layers (the last three layers in table 1) were removed and replaced by the same types of layers. The only difference was that the last layer had a different output size since there are two categories instead of 1000 categories. The remaining (convolutional) layers were set to be non-trainable. In this way the network used the pre-trained weights for the convolutional part of the network and the fully connected layers were trained on our dataset. An overview of the code used to train the inception-V3 network is provided in appendix A.

The dataset is preprocessed in the same way as the pre-training data, as proposed in the original inception paper (Szegedy et al. 2015). The dataset was zero centered and resized to a range of -1 to 1 (see appendix A). To examine how the network performs on each category the network is tested on one category versus the other three categories. First, an experiment was done on a small dataset to examine how the learning rate affects the loss and accuracy during the training of the network. This is done on a sample of 150 images in total. 75 images were from one of the four categories, the other 75 were picked from the other three other categories (25 images from each of the other three categories). This experiment was done for all four categories with six different learning rates ranging from 1×10^{-5} and 1. The ratio between training and validation data was 80% and 20% and the network was trained for 20 epochs.

The learning rate at which the network performed best was used to do a final training of the network with a larger amount of data. In the final training 540 images were used in both categories in the same ratio as described above. Again, this was done for all four categories. In this final training the network was trained for 30 epochs. All of the selected images for the experiments were stored in numpy arrays containing the training set, the test set and the labels. The code written to select the images (randomly) from the different categories can be found in appendix B.

6 Results

In this research the top-layers of the Inception-V3 network were trained on images captured by the EIT-171 instrument in order to predict solar weather. Each category was tested against the other three categories to measure how the network performs on each category. To determine the learning rate at which the network performed best, the network was trained on a smaller dataset with different learning rates. In appendix C the results of this experiment can be found. It was observed that the network performed best with a learning rate of 1×10^{-5}. With this learning rate the loss is observed to be less than with the other learning rates. This is important since the loss is a measure of error while training and testing the network.

In the final experiment a larger part of the dataset is used to train the network (540 images for both categories). It was found that the network performed best at classifying Ejecta. Ejecta could be classified with a validation accuracy of 80% (figure 11). The network was not able to distinguish Streamer-Belt-Origin Plasma from the other three categories, the validation accuracy converged to a value of 51% (figure 13). Coronal-Hole-Origin Plasma and Sector-Reversal-Region Plasma could be classified with a validation accuracy of 67% (figure 12) and 72% (figure 14) respectively. In figures 11-14 the losses are displayed as well. It can be observed that the validation loss for the Ejecta category is lower than the other categories. This is as expected since the loss is a measure of error and the accuracy with which ejecta is predicted is the highest. The loss for the other categories converged to values ranging from 0.62 - 0.79 while the loss of the Ejecta category converged to a value of 0.45.

In addition to the validation loss and accuracy also the training loss and accuracy where monitored during the training of the network. The validation loss and accuracy can be compared to the the training values to see how the network performs on new data versus the data on which it is trained. In all the categories it can be seen that the training accuracy is higher than the validation accuracy and that the training loss is lower than the validation loss.

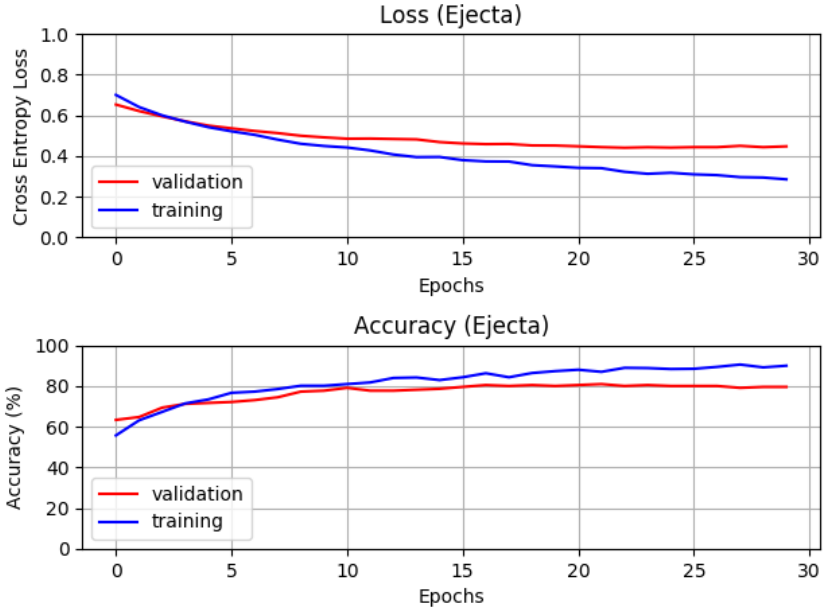

Figure 11: Accuracy and loss of the experiment in which the ejecta category is classified against the other three categories. The accuracy of the validation set converged to 80% and the cross entropy loss converged to a value of 0.45.

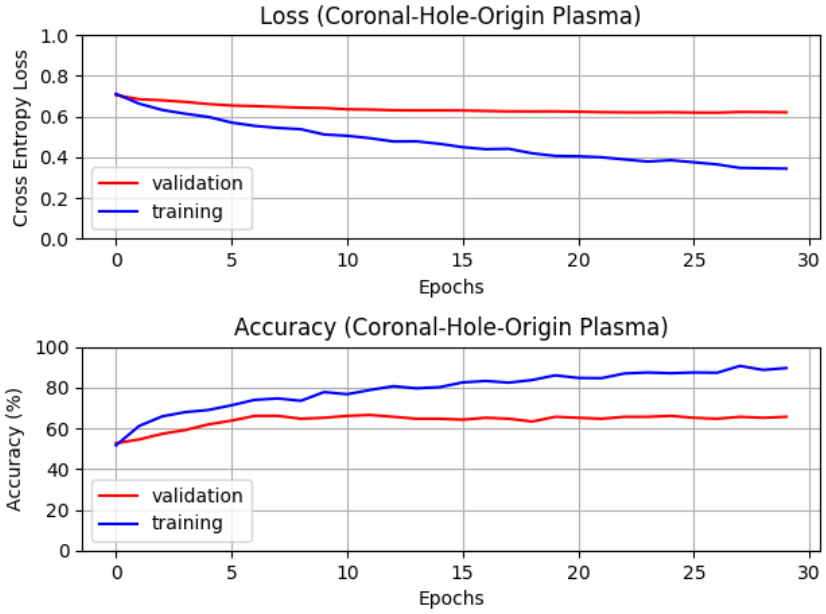

Figure 12: Accuracy and loss of the experiment in which the Coronal-Hole-Origin Plasma category is classified against the other three categories. The accuracy of the validation set converged to 67% and the cross entropy loss converged to a value of 0.62.

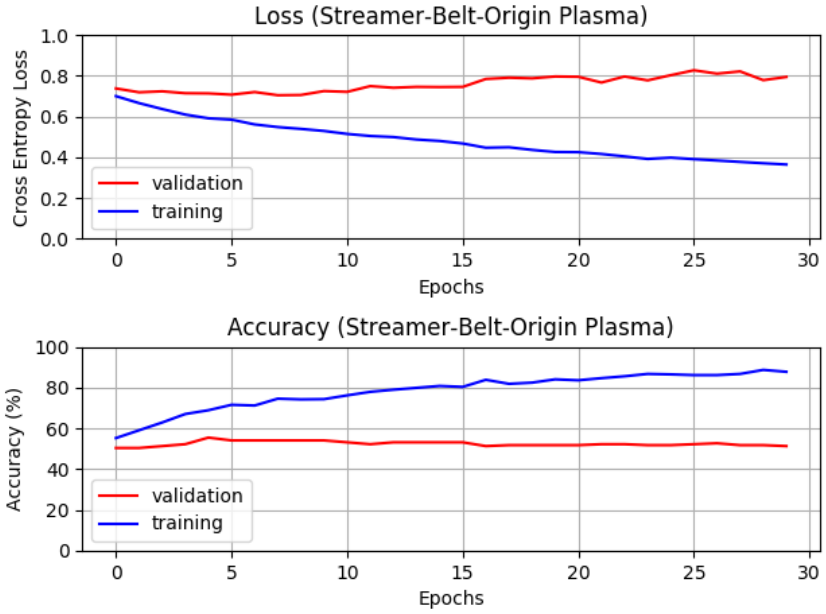

Figure 13: Accuracy and loss of the experiment in which the Streamer-Belt-Origin Plasma category is classified against the other three categories. The accuracy of the validation set converged to 51% and the cross entropy loss converged to a value of 0.79.

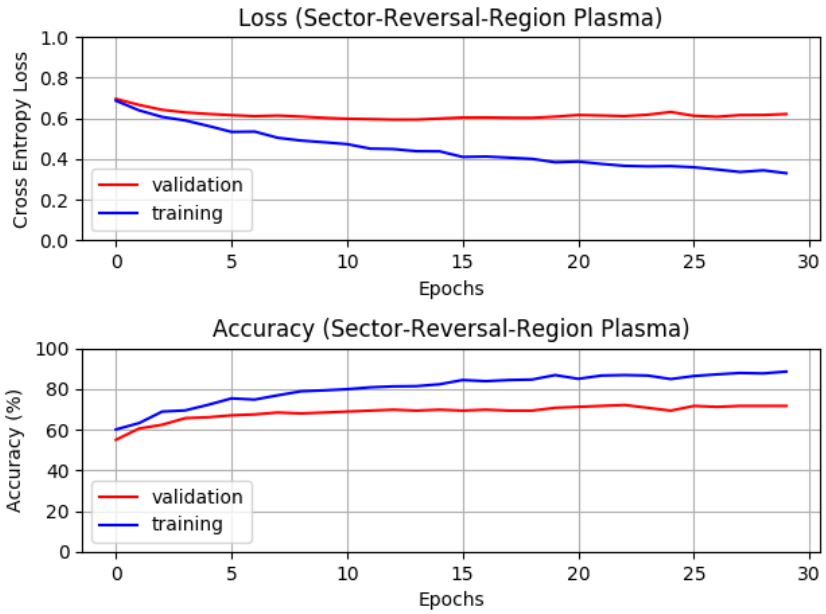

Figure 14: Accuracy and loss of the experiment in which the Sector-Reversal-Region Plasma category is classified against the other three categories. The accuracy of the validation set converged to 72% and the cross entropy loss converged to a value of 0.62.

7 Discussion

As observed in the results section, it is found that the training accuracy in all graphs is higher than the validation accuracy. This strongly suggests that the network is overfitting on the training data. Overfitting here means that the model is trained very well on predicting the training data but cannot handle new data correctly. There are multiple factors which can contribute to this overfitting. First of all, the network is pre-trained and there is a possibility that this pre-trained network extracts the wrong features from the images for our goal. This means that the network categorizes the training images based on the wrong features. The network then learns patterns from the training set that are not applicable in the validation set. This problem can possibly be solved by adding the fully connected layers earlier in the network where more basic features are extracted.

Another aspect leading to overfitting might be the size of the used dataset, the dataset used (with 540 images in each category) is small compared to similar researches (Krizhevsky et al. 2012). Also, The EIT-171 instrument captures images every six hours meaning that part of the images in the dataset look very similar. The network may not have distilled the correct information from the data it got because of the datasize. This might have affected the accuracy in a negative way. In further research it would therefore possibly be better to use a larger dataset and to make a selection with respect to timelapses. When a larger dataset is used it is also possible to train convolutional parts of the network, which can lead to better feature extraction.

In this research, for simplicity, only images from the EIT-171 instrument were used meaning that only one channel was used in the images. The different types of categories can probably be made more visible when all channels are used. Since the images with different wavelengths are not captured at exactly the same time, it is needed to design a method where images of different wavelengths are correctly added together into a more complete image. In our research Ejecta could be predicted with an accuracy of 80% while for example the Streamer-Belt-Origin Plasma could be predicted with an accuracy of 51%. Ejecta might be more visible in the 171 Ångstrom wavelength than the other categories and the usage of only one channel might have caused the difference in accuracies of different categories. It is a possibility that the other categories are better visible at other wavelengths.

In addition, there are more variables that could be altered than only the learning rate when training the network. Due to time constraints only the effects on altering the learning rate was examined. Other variables, like the batch size (the number of images propagated through the network, after which the weights are updated) also affect the performance of the network. There are more variables that will not be discussed in this thesis. It is critical that these variables are taken into account in further research.

Another important factor which influences the accuracy of the predictions is the categorization scheme which is used. No categorization scheme is exact because of the lack of knowledge about the solar wind origins. In this research a categorization scheme is used which showed to be more accurate than other existing schemes, the fraction of correctly categorized points of this scheme ranged from 96.9% to 69.9%. A more-detailed study of solar wind is needed to develop a categorization scheme with a higher accuracy which also might lead to better accuracies in predicting space weather.

Lastly, the solar cycle could influence predictions as well. The solar cycle is the approximately 11 year periodic change in the magnetic activity and the sun's luminosity (Willson & Hudson 1991). The Sun can appear very different in different stages of the solar cycle, which can affect the ability for the network to predict the correct category. The four categories used in this

research might for example look different in these different solar cycle stages as well. When applying a (new) larger dataset on a CNN it is important to take in account the effects the solar cycle has on the images. To investigate the effect the solar cycle has on the outcomes of training a CNN one could for example train the network on images of one stadium of the solar cycle at a time. When it is found that the solar cycle has a significant effect on the outcomes of training a network, the network has to be adapted to this effect or the categorization scheme could be adapted to the effects the solar cycle has.

8 Conclusions

In this research the top layers of the Inception-V3 network are trained on the solar images captured by the EIT-171 instrument aboard the SOHO satellite. The network is trained on four different categories of solar plasma: Ejecta, Coronal-Hole-Origin Plasma, Streamer-Belt-Origin Plasma and Sector-Reversal-Region Plasma. Each of those categories is tested against the other three categories. Ejecta could be classified with an accuracy of 80%, Coronal-Hole-Origin Plasma with an accuracy of 67%, Streamer-Belt-Origin Plasma with an accuracy of 51% and Sector-Reversal-Region Plasma with an accuracy of 72%. Although the accuracies are not yet desirable for reliable space weather prediction, when the dataset size, the usage of only one channel and the accuracies of the used categorization scheme are taken into account these results are promising for further research.

9 Acknowledgements

First of all I would like to thank dr. Enrico Camporeale for supervising me and for giving me the opportunity to do this project. Before this project I did not know a lot of things about machine learning and I really appreciate the things I learned. I definitely think it will help me in the future when I proceed to a master in Computational Sciences. I would also like to thank dr. Anna L. Watts for being my second examinator, Mandar Chandorkar for helping me with the data acquisition and Timo Halbesma for the support during the writing of my thesis and helping me with LaTeX related issues. Lastly, I would like to thank Olaf van Waart & Chris Olberts for their never ending, unconditional support throughout my project.

10 References

Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from tensorflow.org

Chollet, F. 2015, Keras, https://github.com/fchollet/keras

Dasso, S., Milano, L., Matthaeus, W., & Smith, C. 2005, The Astrophysical Journal Letters, 635, L181

Demuth, H. B., Beale, M. H., De Jess, O., & Hagan, M. T. 2014, Neural network design (Martin Hagan)

Deng, J., Dong, W., Socher, R., et al. 2009, in Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 248–255

Feldman, U., Landi, E., & Schwadron, N. 2005, Journal of Geophysical Research: Space Physics, 110

Feynman, J. & Gabriel, S. 2000, Journal of Geophysical Research: Space Physics, 105, 10543

Krizhevsky, A., Sutskever, I., & Hinton, G. E. 2012, in Advances in neural information processing systems, 1097–1105

Li, F.-F., Johnson, J., & Yeung, S. 2016

McCulloch, W. S. & Pitts, W. 1943, The bulletin of mathematical biophysics, 5, 115

Muller, A. C. & Guido, S. 2017, Introduction to machine learning with Python (O'Reilly Media)

Nielsen, M. 2015, Neural Networks and Deep Learning (Determination Press)

Oquab, M., Bottou, L., Laptev, I., & Sivic, J. 2014, in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

Qahwaji, R. & Colak, T. 2007, in Recent Advances in Space Technologies, 2007. RAST'07. 3rd International Conference on, IEEE, 739–742

Rosenblatt, F. 1958, Psychological review, 65, 386

Rumerhart, D., Hinton, G., & Williams, R. 1986, Nature, 323, 533

Schmidhuber, J. 2015, Neural networks, 61, 85

Sharif Razavian, A., Azizpour, H., Sullivan, J., & Carlsson, S. 2014, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 806–813

Solar and Heliospheric Observatory,. 2017, Comparison of two EIT-171 images, https://sohowww.nascom.nasa.gov/gallery/images/eit171cf.html, online; accessed 27 March 2017

Szegedy, C., Liu, W., Jia, Y., et al. 2015, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1–9

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. 2016, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2818–2826

Tieleman, T. & Hinton, G. 2012, COURSERA: Neural networks for machine learning, 4

Widrow, B., Hoff, M. E., et al. 1960, in IRE WESCON convention record, Vol. 4, New York, 96–104

Wiesel, D. & Hubel, T. 1959, J. Physiol, 148, 2

Willson, R. & Hudson, H. 1991, Nature, 351, 42

Xu, F. & Borovsky, J. E. 2015, Journal of Geophysical Research: Space Physics, 120, 70

Appendices

Appendix A

Below an overview of the used code is provided. The full code of the Inception-V3 network kan be found on https://github.com/fchollet/deep-learning-models/blob/master/inception_v3.py (Chollet 2015).

```
# -*- coding: utf-8 -*-
"""
Implementation of the inception-V3 network.

@author: Marleen Rijksen
"""

# utilities
from keras.applications.inception_v3 import InceptionV3
from keras.models import Model
from keras.utils import np_utils
from keras.layers import Dense, GlobalAveragePooling2D
import numpy as np
import pandas

#=============================================
# Data preparation
#=============================================

# for reproducibility
np.random.seed(123)

def preprocess_data( train_images_file , test_images_file , train_labels_file , test_labels_file ):

    # load data from directory
    train_images = np.load( train_images_file )
    test_images = np.load( test_images_file )
     train_labels = np.load( train_labels_file )
     test_labels = np.load( test_labels_file )

    # turn data into correct format
    train_images = train_images.astype('float64')
    test_images = test_images.astype('float64')

    # normalize and zero center the data
    train_images = np.divide(train_images, 255)
    test_images = np.divide(test_images, 255)
    train_images -= 0.5
    test_images -= 0.5
    train_images *= 2
    test_images *= 2
```

```
    # turn category list into binary class matrices
    train_labels = np_utils.to_categorical(train_labels, 4)
    test_labels = np_utils.to_categorical(test_labels, 4)

    return train_images, test_images, train_labels, test_labels

#=============================================
# Finetuning the top layers of the Inception-V3 network
# Part of this code is inspired by the code as can be found on
# https://keras.io/applications
#=============================================

def inception_v3(train_images, test_images, train_labels, test_labels):

    # load the inception model without top layers, pretrained on imagenet
    conv_layers = InceptionV3(weights='imagenet', include_top=False)

    # add pooling layer
    x = conv_layers.output
    x = GlobalAveragePooling2D()(x)

    # add fully connected layer
    x = Dense(1024, activation='relu')(x)

    # add fully connected layer with two classes with a softmax clasisfier
    output_layer = Dense(2, activation='softmax')(x)

    # construct final model
    final_model = Model(input=conv_layers.input, output=output_layer)

    # only train the top layers, freeze the convolutional layers
    for layer in conv_layers.layers:
        layer.trainable = False

    # compile the model
    final_model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
                        metrics=['accuracy'])

    # specify learning rate
    lr = 0.00001

    from keras.optimizers import RMSprop
    model.compile(optimizer=RMSprop(lr=lr), loss='categorical_crossentropy')

    # specify number of epochs
    nb_epoch = 30

    # train the network and save results in hist variable
    hist = model.fit(train_images, train_labels, nb_epoch=nb_epoch, batch_size=32,
                     validation_data=(test_images, test_labels))
```

```
    # write data to csv  file
    pandas.DataFrame(hist.history).to_csv("history"+"lr{}".format(lr)+".csv")

    return hist
```

Appendix B

Below the code is provided to store random images from each category in numpy arrays. Also an array of labels in the same order as the images is created. These arrays can be fed to the network (appendix A).

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 19 11:30:35 2017

@author: Laptop
"""

import os
from scipy.misc import imread, imresize
import random
import numpy as np
from numpy import array

#=============================================================
# Functions
#=============================================================

# resize images and store them in a list
def resize_append(category, width, height):

    image_list = []

    # first list all filenames in specified directory
    filenames = os.listdir('C:/Users/Laptop/documents/Bachelorproject/Dataset/Textfiles/eit_171/'
                           + category)

    for filename in filenames:

        # read and resize images
        im = imread('C:/Users/Laptop/documents/Bachelorproject/Dataset/Textfiles/eit_171/'
                    + category + '/' + filename)
        im = imresize(im, (width, height))

        # store image in list
        image_list.append(im)

    # return list with numpy arrays of images in it
    return image_list

# pick random images from a list of images, and divide them in test/train data
def pick_rand_img(images, nb_img, nb_train):
    train_images = []
    test_images = []
    count = 0
```

```
    for index in random.sample(range(len(images)), nb_img):
        if count < nb_train:
            train_images.append(images[index])
        else:
            test_images.append(images[index])
        count +=1
    return train_images, test_images

# make a list of all categories, corresponding to the list of images
def category_list(nb_cat, nb_train, nb_test):
    train_labels = []
    test_labels = []
    for i in range(nb_cat):
        train_labels.extend([i for x in range(nb_train)])
        test_labels.extend([i for x in range(nb_test)])
    return train_labels, test_labels

#=====================================================
# Example of the usage of the functions described above, in this example 540 images
# of the coronal-hole-origin plasma are selected and 540 images from the other three
# categories are selected (180 from each of the three other categories).
#=====================================================

# specify image size in which the images have to be resized
width = 299
height = 299

# resize images in specific directory
strahl = resize_append('strahl', width, height)
pseudo = resize_append('pseudo', width, height)
lepping = resize_append('lepping', width, height)
coronal = resize_append('coronal', width, height)

# specify number of images for the category which is tested
nb_img = 540
dif_categories = 4

# take nb_images / 3 images from each of the other three categories
nb_img_other = round(nb_img / (dif_categories - 1))

# define amount of training/test data
train_part = 0.8
nb_train = round(train_part * nb_img)
nb_train_other = round(train_part * nb_img_other)

# divide data in train/test data for each category
coronal_train, coronal_test = pick_rand_img(coronal, nb_img, nb_train)
pseudo_train, pseudo_test = pick_rand_img(pseudo, nb_img_other, nb_train_other)
lepping_train, lepping_test = pick_rand_img(lepping, nb_img_other, nb_train_other)
strahl_train, strahl_test = pick_rand_img(strahl, nb_img_other, nb_train_other)
```

```
# now add coronal data and all the 'other' categories in one list
total_train = coronal_train + strahl_train + pseudo_train + lepping_train
total_test = coronal_test + strahl_test + pseudo_test + lepping_test

# make a list with the correct labels
nb_test = 108
nb_train = 432
nb_cat = 2
train_labels, test_labels = category_list(nb_cat, nb_train, nb_test)

# store everything in numpy arrays
np.save('train_images_file.npy', array(total_train))
np.save('test_images_file.npy', array(total_test))
np.save('train_labels_file.npy', array(train_labels))
np.save('test_labels_file.npy', array(test_labels))
```

Appendix C

In this appendix the results of the experiment with different learning rates is provided.

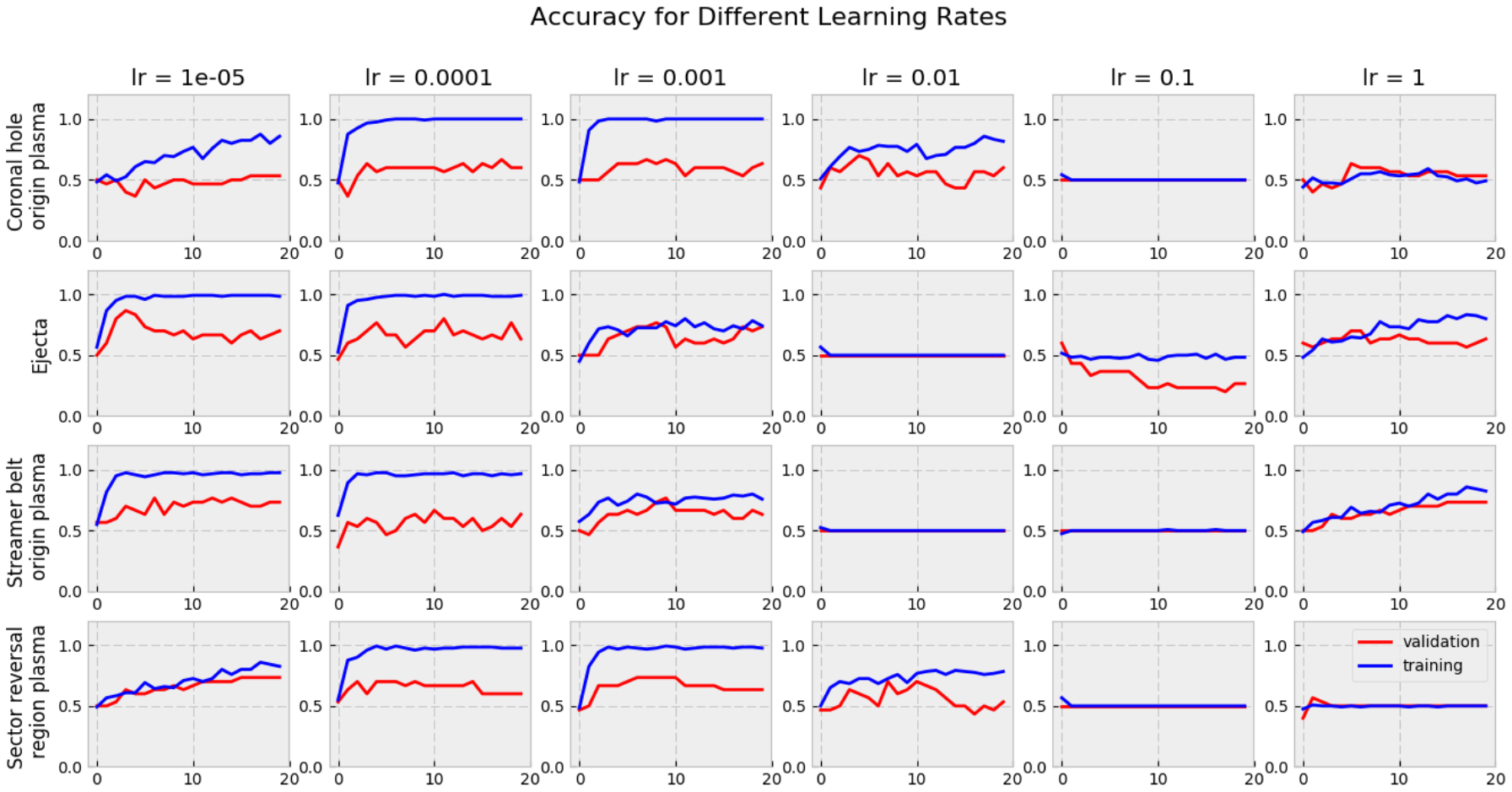

Figure 15: Accuracies for each category for different learning rates. The y-axis displays the accuracy and the x-axis the epochs.

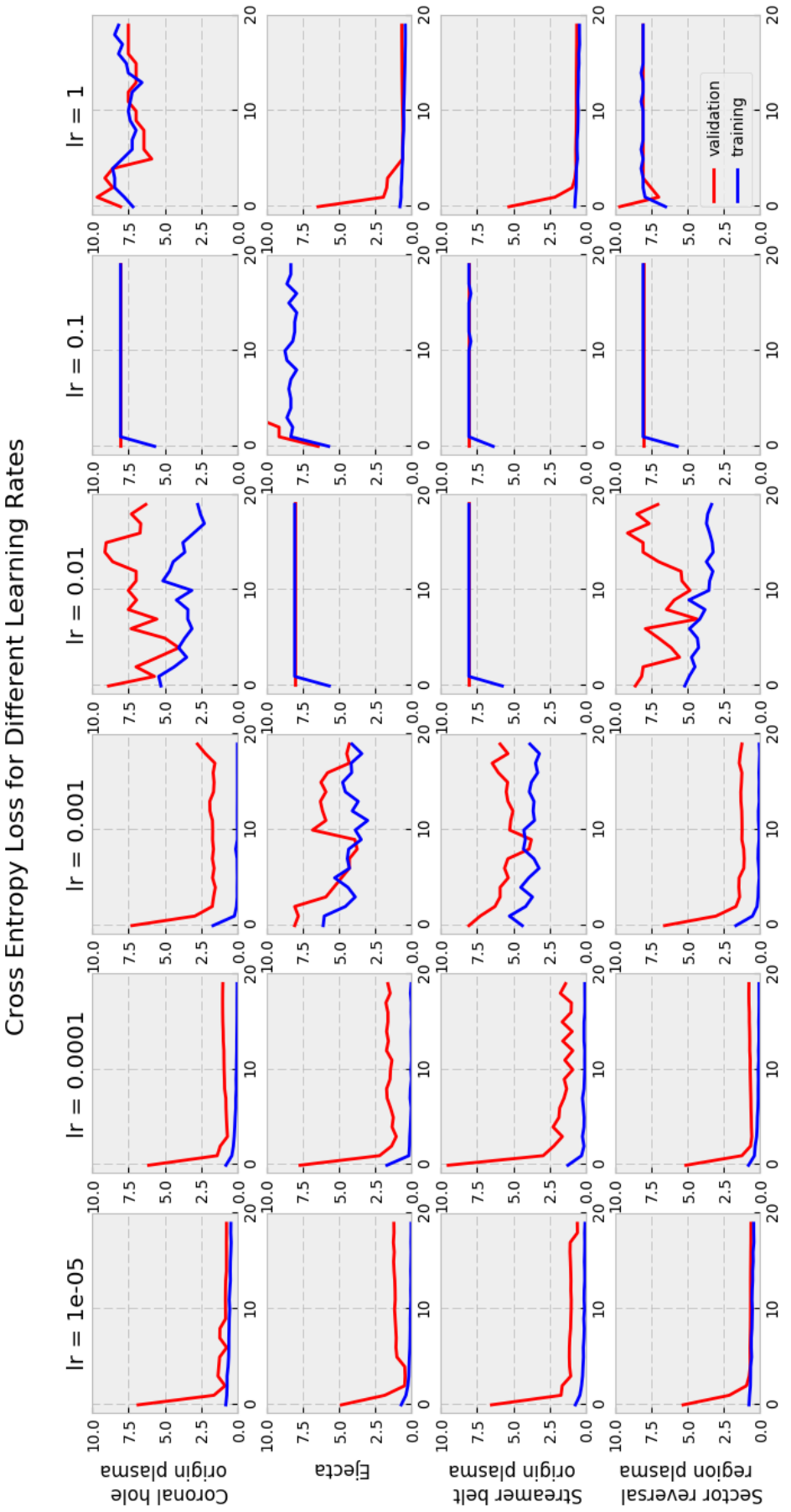

Figure 16: Loss for each category for different learning rates. The y-axis displays the cross entropy loss and the x-axis the epochs.