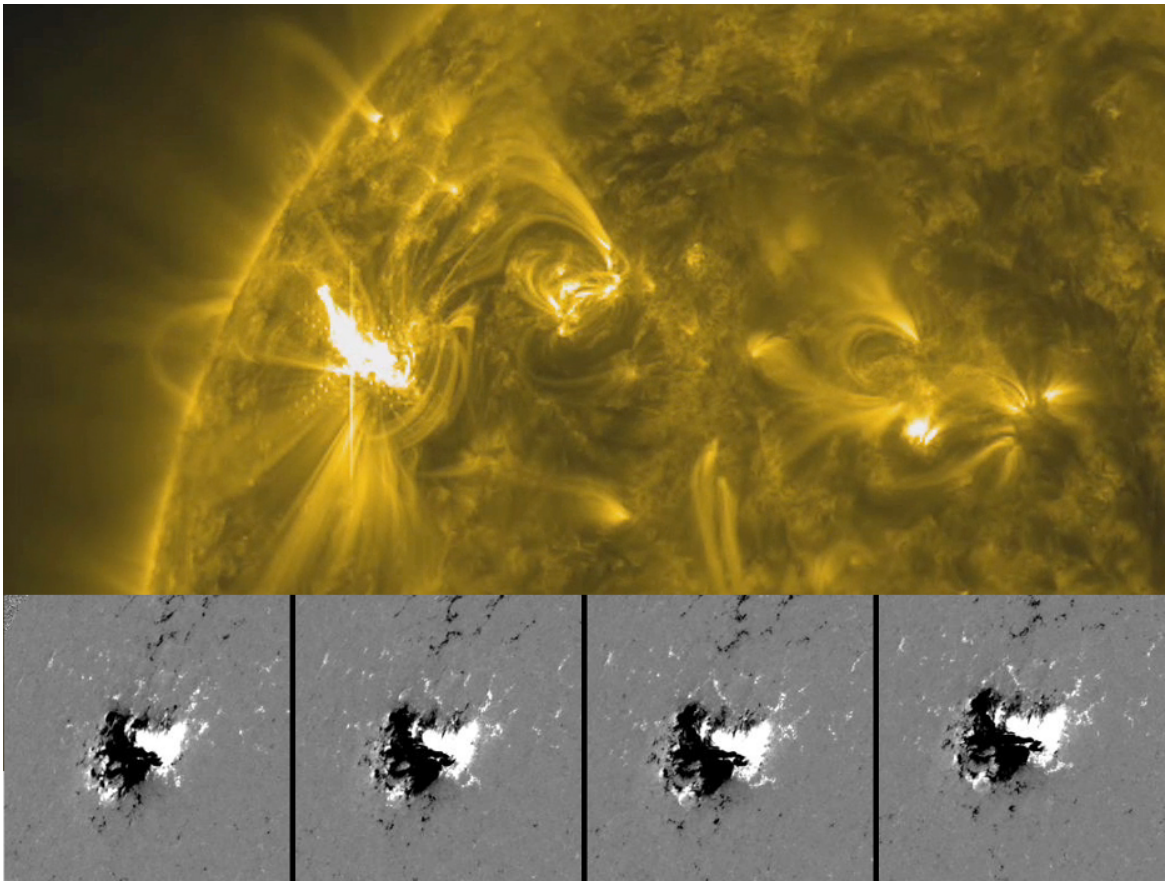
University of Applied Sciences and Arts
Northwestern Switzerland



SFP@FHNW

IP5: Solar Flare Prediction on SDOBenchmark Data

Authors

Ackermann Patrick, Siffer Florian

BCs Information Technology

Advisors

Csillaghy André, Felix Simon

Institute for Data Science | FHNW

Windisch 2019

Abstract

Solar flares are sporadic high energy events near the surface of the sun. These can produce highly energetic particle streams, which pose risks to astronauts, spacecraft and some terrestrial systems. Their accurate and timely prediction would enable potential evasive maneuvers and further study of these events and their mechanics. Currently their prediction is done by experts and partially automated systems using conventional image feature extraction.

Convolutional Neural Networks could bring various benefits to this task, over the methods used currently. Most importantly, they promise to be cheaper, more accurate and potentially even offer insights into the underlying physical systems. As an initial step, the Institute for Data Science has curated a machine learning dataset consisting of satellite image data of the sun and solar flare strength values. Our work is the next major step towards the goal of using satellite image data to predict the strongest solar flare in the 24 hours after an image was taken.

The contributions which are presented in this report, can be grouped into three parts. First, quick summaries provide all of the relevant information in regards to the problem domain, the dataset and machine learning techniques. Secondly, and this is the most important contribution, a development and evaluation framework for CNNs using the previously mentioned dataset was developed. In a final step, several models with different architectures using the previously mentioned framework were developed and evaluated. The report concludes with recommendations about how to proceed and make further progress towards the goal outlined above.

Contents

1. Introduction

In recent years, Convolutional Neural Networks (CNN) have achieved comparable performance to, and in some cases even surpassed, manual and conventional feature extraction methods in the fields of image recognition and natural language processing. The methods CNNs use will be presented in further detail in Chapter 4, Machine Learning Theory.

Many of these CNN architectures have been successfully adapted and employed in other domains, such as the playing of digital and analog games [1], the prediction of atomic and molecular interactions [2] and natural language processing. CNNs promise to be adaptable to many more domains, without requiring deep domain knowledge, thus turning many unconnected problems into machine learning problems [3]. Advances in machine learning methods can then be used to improve the performance of such systems, independent of the current understanding of the application domain. Additionally, the analysis of the trained networks can enable domain experts to discover previously unknown properties of the underlying systems.

One application domain which could benefit from CNNs is the prediction of solar flares. The streams of highly energetic particles they can produce, pose serious risks to astronauts, spacecraft and terrestrial power, navigation and communication grids. Accurate and fast solar flare prediction could enable us to aim observation instruments at the events in advance and potentially take countermeasures. Currently, the prediction is primarily done manually by experts, through the analysis of sunspots and their characteristics. More recently, there have also been efforts to develop partially, as well as fully automated systems, using conventional feature extraction methods. However, none of the currently employed methods are fast, cheap and accurate enough [4]. CNNs might be able to bring their benefits over conventional methods to this application domain.

A major challenge in applying machine learning techniques to this problem is curating the dataset used to train any potential model. One of the primary challenges to curating such a dataset is the fact that the amount of relevant data is relatively small. In this case, relevant data refers to samples of large to extreme solar flares. This also leads to a significant bias within the small dataset. Another challenge is the sensible division of the available data into training and validation datasets, as it requires a detailed analysis of the data and some understanding of the sun as a physical system. The Institute for Data Science (I4DS) has curated such a machine learning dataset and provided a zero-rule baseline prediction on that dataset [4]. The specifics of the dataset will be discussed in Chapter 3, Dataset Analysis, of this report.

Based on the success of CNN models in other domains, the availability of the dataset and the potential benefits of successfully using such a model to predict solar flares, we come to the conclusion, that it is worthwhile to pursue the possibility of predicting solar flares using CNNs.

In this project we made major progress towards the long term endeavour of predicting the size of the largest solar flare flux in the next 24 hours after a satellite image was taken, as discussed in Chapter 7.1, Accomplishments. Most importantly, we developed a framework

that significantly eases the configuration, training, evaluation and comparison of CNNs. We present this framework in more detail in Chapter 5, Development & Evaluation Framework. Additionally, we used this framework to develop and evaluate several model architectures, which we present in Chapter 6, Models & Evaluation. The remaining chapters are a summary of the information we gathered concerning the application domain, the dataset and machine learning. They serve as an introduction to the project and provide the knowledge necessary to understand our contributions.

2. Domain Overview: Solar Flare Prediction

To understand the goal of long term endeavour and the need for it, this chapter will present a short overview of the problem domain. More concretely, the following chapters will first explain what solar flares are, why it's important to predict them reliably and finally what the current state of solar flare prediction is.

2.1 Space Weather

Space weather refers to various temporary or permanent conditions and events within our solar system. Some of which can have an adverse effect on human systems on earth or within space. Moreover, for some of these events, it would be scientifically beneficial to be able to study them with precise instruments for their entire duration. Ideally instruments would be targeted at these events, even before they occur. [5]

2.2 Solar Flares

Solar flares are a source of space weather events of particular interest. They are a sudden moment of significantly increased brightness on the sun. Usually, solar flares occur in close proximity of a sunspot group and often, but not always, coincide with a coronal mass ejection. They produce extremely high energy particle streams, which can damage spacecraft and astronauts through radiation. Additionally, very powerful solar flares, accompanied by massive coronal mass ejections, can fully disable satellites and terrestrial power, navigation and communication grids for extended periods of time. These and various other risks posed by solar flares are of growing concern, as humanity is expanding its sphere of influence and habitation in the 21st century. Therefore, the fast, cheap and precise prediction of these events is increasingly important. Such predictions would allow us to aim observation instruments at these events in advance, to study them in greater detail. Eventually they could also enable possible countermeasures or evasive maneuvers. [4], [6]

2.3 Solar Flare Prediction

Currently, most solar flare predictions are almost exclusively done manually by experts, through the analysis of sunspots and their characteristics. However, such manual methods are relatively expensive, slow and imprecise. Recently, there have been efforts to develop partially and fully automated systems. These systems feature high complexity and require extensive domain knowledge. Partially automated systems employ conventional feature

extraction and image analysis methods in an initial step. In a second step an expert then uses these features and the original images to make a prediction. Alternatively neural networks have been applied to take these features and make a final prediction. These techniques are comparable to the methods employed by automated terrestrial weather prediction systems and early image recognition systems. [4]

3. Dataset Analysis

This chapter investigates the dataset that the Institute for Data Science provided. The dataset is fairly complex, as will be explained in the next subchapter. The distribution of the available data samples will also be analyzed. Lasty, a few specific data samples will be showcased, in order to gain some intuition and a deeper understanding of the dataset.

3.1 Description

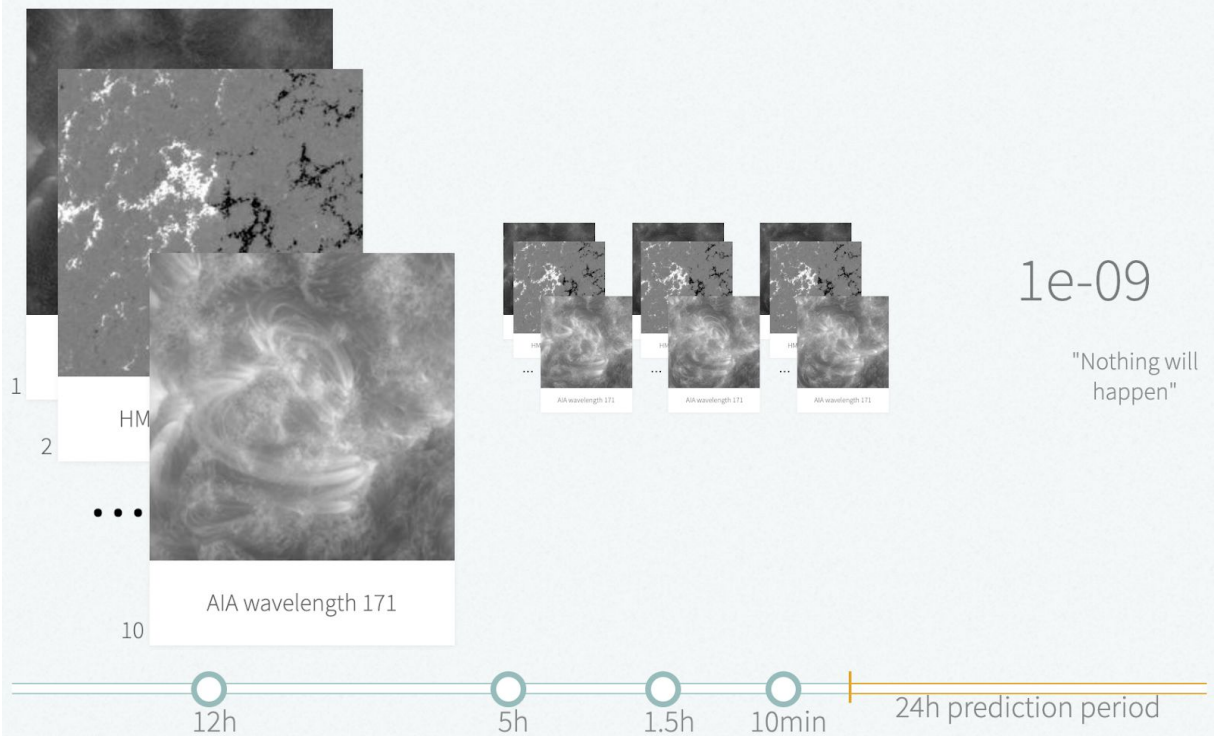


Figure 1: each sample consists of four time steps (→) with ten images (↓) [4]

Figure 1 is a visualization of a single sample in the dataset. The dataset consists of about 8’300 training samples and about 900 test samples. Each sample contains ten different image types at four different points in time before the targeted 24 hour prediction period. Each set of images is already cropped to a small region of interest on the sun. The ten different image types are made by the AIA (Atmospheric Imaging Assembly) and HMI (Helioseismic and Magnetic Imager) detectors onboard the SDO [7] (Solar Dynamics Observatory) satellite mission, targeting various different wavelengths. The four points in time are approximately twelve hours, five hours, one and a half hours and finally ten minutes before the previously mentioned 24 hour prediction period. The labels of the samples, and therefore the numbers that should be predicted by a machine learning model, are the peak

flux values on the sun within the 24 hour prediction period. Flux, in this case, refers to the rate of flow of high energy particles ejected from the sun. [4]

Solar flares are classified using the letters A, B, C, M and X, in accordance to their peak X-ray flux. X-rays having wavelengths of 100 to 800 picometre are considered. The peak flux is measured in watts per square metre (W/m^2), by the GOES spacecraft. Table 1 indicates which classification is assigned to which peak flux ranges. [6]

Classification	**Peak flux range** at 100–800 picometre (watts/square metre)
A	< 1e-7
B	1e-7 - 1e-6
C	1e-6 - 1e-5
M	1e-5 - 1e-4
X	> 1e-4

Table 1: classification for solar flares [6]

3.2 Dataset Imbalance

Using the classification system described above, we can analyze the bias in the training data set. Table 2 shows all the samples grouped by class. It is of particular note that only 6.5% of the samples are in the classes M and X, while class A makes up almost 44% percent of the samples on its own.

Such a biased dataset makes this machine learning problem significantly harder. However, as discussed in Chapter 4.6, Evaluation of Models, the Mean Absolute Error weights percentage errors for the underrepresented classes (M and X) much more heavily than for other classes, due to the logarithmic nature of the peak flux label. This should mitigate this imbalance at least partially. [4]

It is also notable, that the samples stem from very few regions on the sun. This had to be taken into account when creating the dataset, to prevent the dataset from being biased against individual regions. For example, if too many flaring samples came from the same region on the sun, the network might learn to recognize that region, instead of learning to correctly predict flare size for any region on the sun. [4]

Classification	# of Samples	% of Samples	# of Different Regions on Sun	% of Different Regions on Sun
A	3652	43.8 %	450	29.5 %
B	1210	14.5 %	106	7.0 %
C	2936	35.2 %	407	26.7 %
M	500	6.0 %	111	7.3 %
X	39	0.5 %	17	1.6 %

Table 2: training sample distribution over solar flare classes and regions of the sun [8]

3.3 Examples

Traditionally, only the magnetogram image type has been used to predict solar flares, for manual forecasting as well as for feature extraction based methods. To gain a better understanding of the data and how it differs for different peak flux strengths, we will showcase a few magnetogram data samples in this chapter.

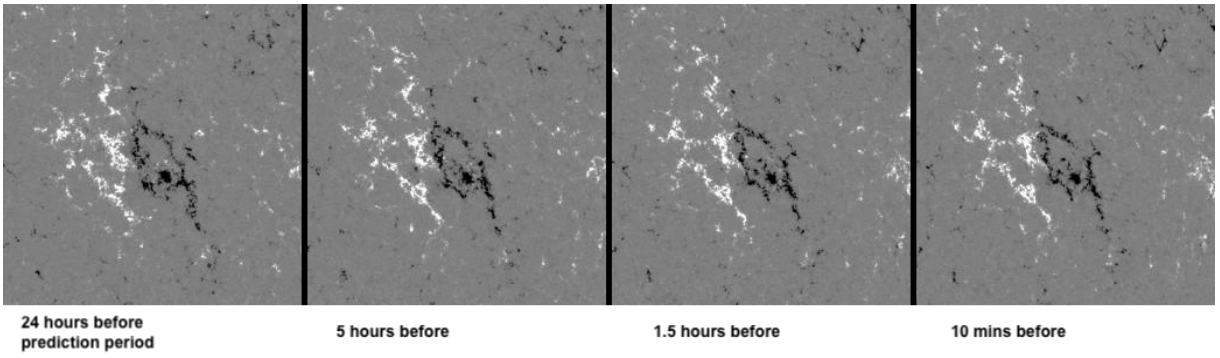


Figure 2: Magnetograms before a solar flare with minimal flux

Figure 2 is assembled from a data sample with minimal flux, i.e. 1.0e-9, which indicates there was no solar flare. It shows, from left to right: twelve hours, five hours, one and a half hours and ten minutes before five o'clock on the 05.20.2014.

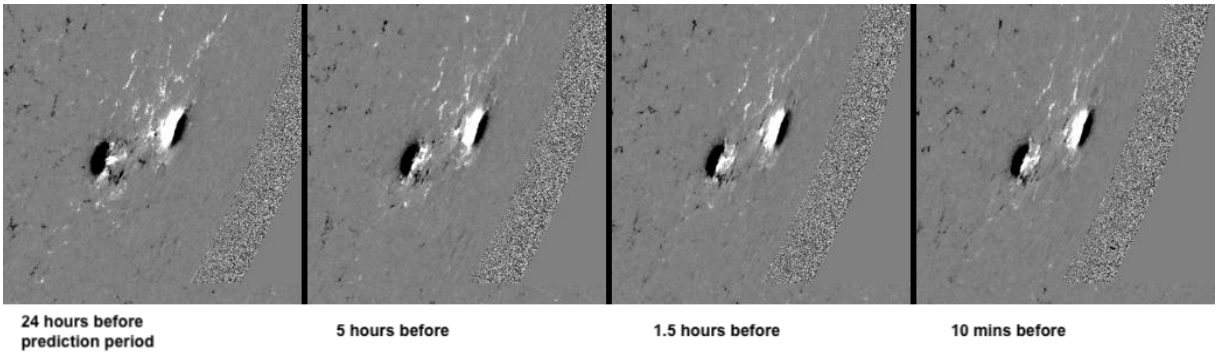


Figure 3: Magnetograms before a solar flare with medium flux

Figure 3 is showing a series of magnetograms before a prediction period which had a roughly median peak flux value of 5.2e-07, placing this flare in the B class. The images were taken before the prediction period starting at eight o'clock on the 17.08.2014. This region has bigger and better defined "spots" than the first sample, which experts use to predict the peak flux. Note that in this image, the edge of the sun can be seen in the lower right corner of the image.

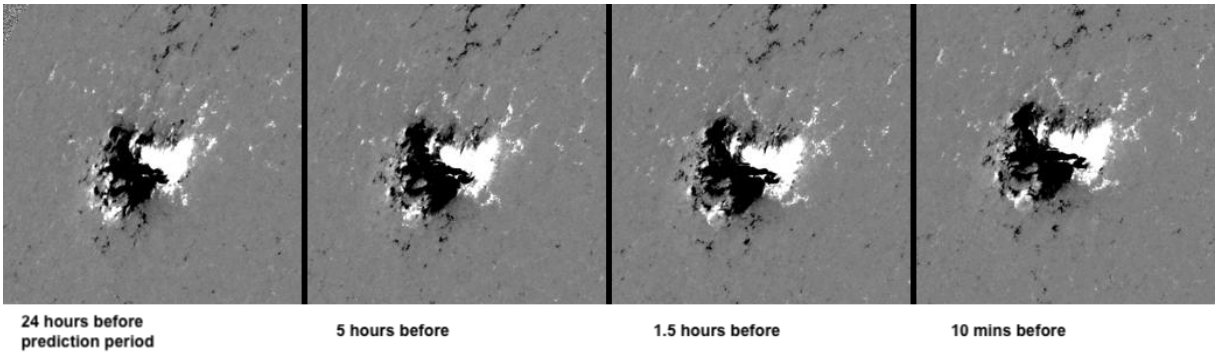


Figure 4: Magnetograms before the solar flare with the maximum flux

This last example, figure 4, shows the series before a solar flare of class X with maximum peak flux in the training dataset of 6.4e-4. It is showing a region of the sun before the prediction period starting half past midnight on the 06.03.2012. It is notable that the spots are now even bigger and much more complex than in the second example. Moreover, their edges have become irregular.

4. Machine Learning Theory

In this chapter we will provide an overview of existing machine learning techniques and practices, provided they are relevant for the understanding of the developed framework and experimental models, mentioned in Chapter 5 and 6 respectively. More concretely, the following subchapters will cover the reasons to use machine learning at all, the preprocessing of data, two specific machine learning techniques, two architectures and finally the evaluation of models.

4.1 Benefits of Machine Learning

One of the most important aspects of machine learning models is that they can solve problems for which it would be infeasible to develop an algorithm consisting of specific instructions [9]. The goal of the overarching endeavour that this project is part of, which is predicting the size of the largest solar flare flux in the next 24 hours after a satellite image was taken, is an example of such a problem.

Machine learning models outperform humans at most specific tasks they have been adapted to in several key metrics. They are significantly faster, cheaper and most of the time more accurate [10]. Once a model has been successfully developed for the goal outlined above, it will likely feature these same improvements over manual prediction methods.

Since machine learning techniques are fairly domain independent, advances in one domain can be adopted to another relatively easily. Therefore, as the field of machine learning advances, it is likely that new techniques can be transferred to this problem as well and thus improving the performance on this problem without requiring further discoveries about the mechanisms of solar flares. The most popular example of this are CNNs. [11]

Finally, in some areas, trained machine learning models have been analysed to better understand what the network is doing [12]. This has allowed humans to discover previously unknown properties of the underlying physical systems. Since the sun as a whole and solar flares specifically are an active area of research, any such insights based on the results of continuations of this project would prove immensely valuable.

4.2 Preprocessing & Augmentation

An important part of any machine learning pipeline is data preprocessing. It mainly involves dealing with outliers and incomplete data. The specific techniques to be used heavily depend on the dataset. For example, samples with missing values have to be removed from the dataset or completed with sensible defaults. [13]

When datasets are smaller than desired, which is almost always the case since more training data increases almost any models performance, data augmentation is employed. This commonly done with images, and can involve random cropping, zooming, rotations and changes to the brightness. However, the specific set of operations performed to increase the amount of available data has to be considered carefully for each machine learning problem individually. It is very important to select augmentation techniques in a way that does not change the structure within the image that the model should learn to recognize. In this project for example, horizontal flipping would result in the sun turning in the reversed direction in the flipped images and it is thus unlikely that this additional data would help the network make better predictions in general. [4]

4.3 Regression by Classification

Previous work by Weiss & Indurkhya clearly showed, that turning regression problems into classification problems and then applying leading machine learning classification techniques results in excellent predictions, oftentimes beating identical models with a direct regression output. This is not surprising, as machine learning research has been focused on classification problems and has generally been more successful at solving them, compared to regression problems. [14]

Turning a regression problem into a classification problem requires an encoding and decoding function. The encoding function should take the true labels of the data samples as an input and return a list of values representing the true label. For example, a one-hot encoding into equally sized bins over the value range of the true labels would be such an encoding function. The decoding function should then take such a list of values, either being the output from the model or directly from the encoding function, and transform it back into a

single value. Our framework supports using encoding functions and provides a few predefined ones, see Chapter 5.5, Customization. [14]

4.4 Convolutional Neural Networks

Conventional neural networks do not scale well to larger images, due to them requiring a weight for each pixel in each channel for each neuron on the first layer. They also do not make use of the three dimensional nature of images with multiple channels.

Using convolution and pooling layers, CNNs overcome these shortcomings. They also change the logical architecture of the neurons into a rectangular box, thus mimicking the shape of the input data. Convolution layers use rectangular image filters, that is matrices of weights, to extract a single value from a region in the input data. This is done using the well known convolution operation in mathematics. These filters are moved across the input data in order to compute the value of each neuron of a single dimension in the next layer. Additional filters result in additional dimensions in the output, while additional dimensions in the input lead to filters with greater depth. Pooling layers are similarly constructed, but use different operations to extract a value from a given region of the input data. The goal is to reduce the height and width of the next layer, so the filters are moved more than one pixel to calculate the value for the next neuron, thus resulting in fewer values per dimension than in the input. Commonly used operations for pooling layers are selecting the maximum or the average value in the input region. [15]

4.5 Architectures

In recent years, there have been various relatively successful CNN architectures, some of which translate fairly well into non image classification tasks. There are two key reasons to adopt existing architectures. Most importantly, developing a completely new architecture is immensely time intensive. Moreover, using or adapting an already existing architecture enables transfer learning. Transfer learning refers to a learning technique in which the model is initialized with weights from an already trained model. [16]

4.5.1 Alexnet

The AlexNet architecture was created for the ImageNet Large-Scale Visual Recognition Challenge 2010 contest. The goal of the Imagenet contest is to classify images into 1’000 object classes. To achieve this, the AlexNet model first applies a 11x11 convolution layer to the input with a stride of four. This indicates that the eleven by eleven filter of is moved by four pixels in the input image for each pixel in the output image. The output of the first convolution layer is thus about one quarter of the size of the input. The next layer is a max pooling layer that further reduces the size, followed by another convolutional layer and so on. All together, the Alexnet model has five convolution layers followed by three conventional fully connected layers. There are around 60 million parameters in total [17]. Figure 5 provides a visual overview of this fairly simple but, for the time, unusually deep architecture.

In addition to the architecture and its depth, there was another important difference between Alexnet and previously popular CNNs. Concretely, it was one of the first popular and

influential networks to use the ReLu (Rectified Linear Unit) activation function. For each neuron, after summing up its inputs and adding its bias, the value is run through a activation function. The results of that function are then forwarded to the next layer. In comparison to the previously used sigmoid function, ReLu is not only faster to compute [17], but also tackles the vanishing gradient problem more effectively [18]. Nowadays, the ReLu activation function is widely used.

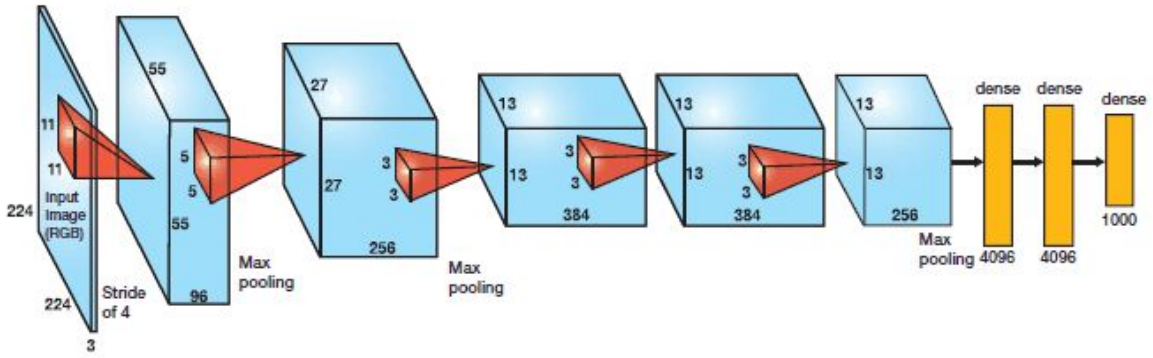


Figure 5: Alexnet architecture [18]

4.5.2 VGG16

VGG was created for the same ImageNet Challenge, but a few years later for the 2014 edition.The most important difference is its massively increased size in terms of trainable parameters. It is not just one architecture though, instead it is a collection of models with different depths. The number after ‘VGG’ indicates the network depth, which is the number of layers with trainable weights. The most popular choice for this is 16, which has around 138 million parameters. That model, VGG16, achieved an error rate as low as 7.5% on the ImageNet challenge [19]. The weights for this performance can be used for transfer learning, as we demonstrate in Chapter 6.4.

What differentiates the VGG model from most of the previous architectures is its simplicity in terms of hyperparameters for the convolution layers and the general arrangement of the layers. Concretely, all convolution layers use a filter size of 3x3 and a stride of one. To reduce the height and width, max pooling layers with a size of 2x2 and a stride of two are used twice after two convolution layers and then after every third convolution layer. The last three layers are fully connected layers, similar to the architecture of most CNNs, as for example with Alexnet. This architecture, specifically for VGG16, is visualized in figure 6. [20]

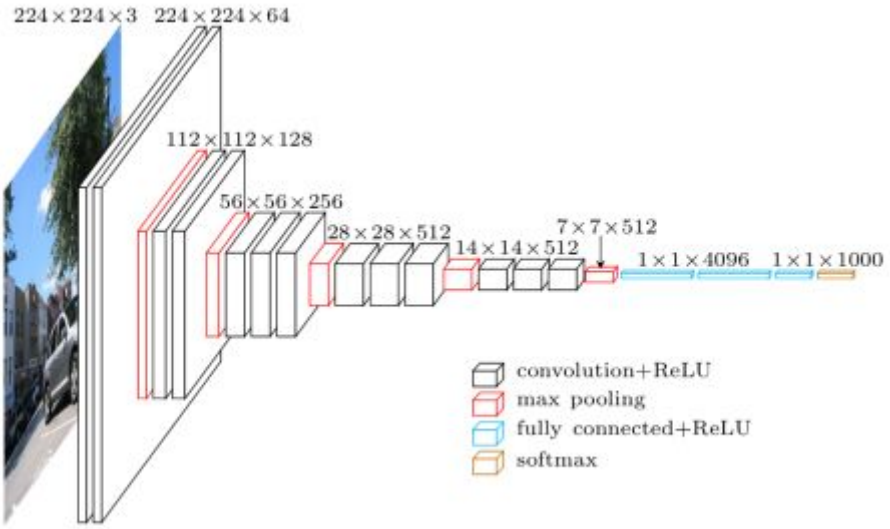


Figure 6: VGG16 architecture [21]

4.6 Evaluation of Models

Most importantly, the evaluation of machine learning models enables the informed selection of the best model among several options. Moreover, during the development and refinement of a machine learning model, evaluation metrics can provide valuable insights to uncover weaknesses and potential opportunities to improve the performance.

To those ends, our framework calculates and logs various different metrics to TensorBoard, as described in Chapter 5.4, Usage, Evaluation & Weights. However, in the following we will only cover the mean absolute error, as its characteristics make it the most appropriate metric for the specific dataset used in this project. The other metrics are documented directly in the framework code, which is referred to in Chapter 8.1, Source Code.

The MAE (Mean Absolute Error) is used to evaluate and compare the overall accuracy of a machine learning model, for which a lower value is better. It is calculated using the evaluation data set as described in equation 1.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n} \tag{1}$$

In equation 1, n is the number of samples in the validation data set, $\hat{y}_i$ is the prediction the model made based on the i-th validation sample and finally, y_i is the true label for that same sample.

The interpretation of the MAE is simple and straightforward, since the average absolute deviation from the true value is easy to conceptualize. Even more so, as the MAE uses the same scale as the predictions themselfs.

However, some caution has to be exercised when the true labels vary strongly in their size: a ten percent deviation for a very large value will affect the MAE more strongly than the same

ten percent deviation for a much smaller value. Whether or not this property of the MAE is desirable depends on the domain and specifics of a task. [4]

5. Development & Evaluation Framework

5.1 Overview

To enable continuations of this project and future strides towards the previously described goal, we built a software framework. Concretely, it trains and evaluates Keras machine learning models on the previously described dataset.

The primary design goal of this framework was to fully separate the definition of a model from all of the code required to train and evaluate it. Now, new models can be trained and evaluated, and therefore compared to other models, with minimal overhead.

The framework provides various options regarding the initialization of the environment, the loading and preprocessing of data and the level of detail of the evaluation. Still, all of the complexity required in these areas is encapsulated and hidden away by sensible defaults.

5.2 Pipeline Architecture

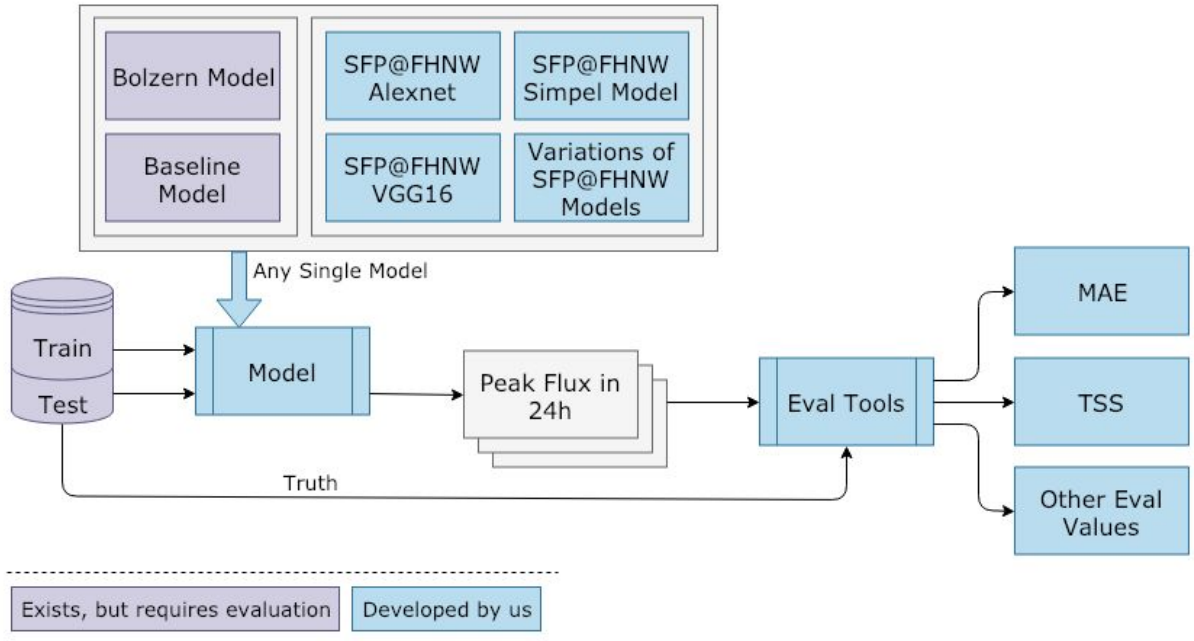


Figure 7: architecture overview

Our pipeline shown in figure 7 shows the primary model development and evaluation pipeline that our framework was created for. The primary idea is that data flows from the left to the right, through the model and the evaluation tools, to finally yield evaluation metrics. Then the model can be switched out, without changing other parts of the pipeline. This means that the

evaluation metrics, and thus the performance of the different models, can be compared with minimal effort.

The flow from left to right includes the training data used to train the model, the evaluation data to make predictions and the true labels of the evaluation data in order to calculate the metrics. The true labels are forwarded directly to the evaluation tools.

5.3 Framework Architecture

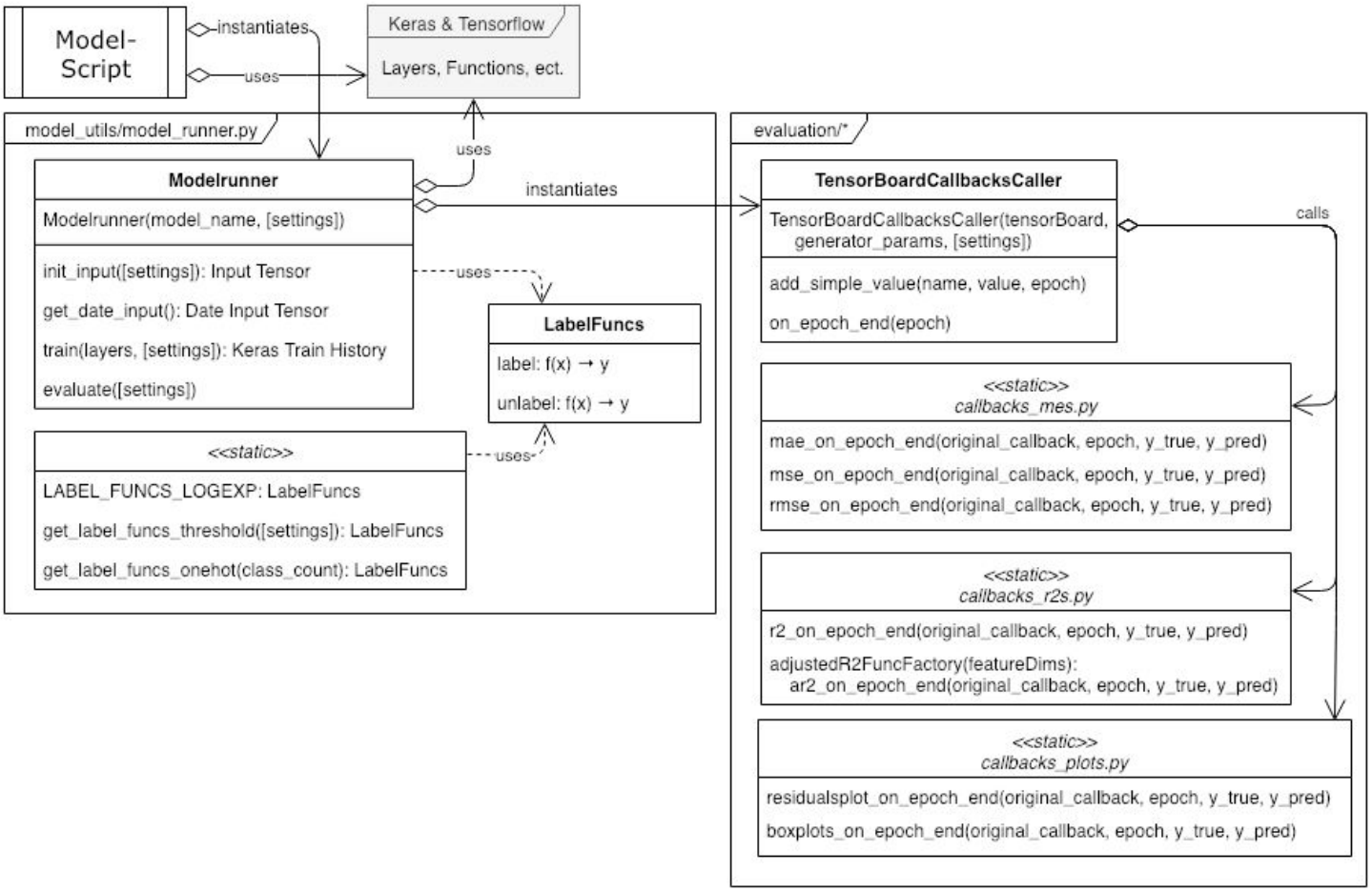


Figure 8: Class diagram of the Framework

Figure 8 is a class diagram of the architecture that we developed. The most important building block of the framework is the "`ModelRunner`" class. It initiates the environment, prepares the data generator objects for training and evaluating and constructs and trains the model. It also runs evaluations before and after the training. The same file statically provides access to three labeling function pairs, "`LABEL_FUNCS_LOGEXP`", "`get_label_funcs_threshold([options])`" and "`get_label_funcs_onehot(classes_count)`", which can be used to transform the labels that should be predicted before training. See Chapter 5.5, Customization.

The class "`TensorBoardCallbackCaller`" does not perform any evaluations itself, instead it runs the evaluation data through the current state of the model and then provides the other

callbacks, which actually perform the evaluations, with the true and predicted maximum flux values.

The model script itself is only expected to instantiate a "`ModelRunner(model_name, [settings])`" instance, use "`init_input([settings])`" to get the main data input tensor, stack any number of Keras layers on top of it and call "`train(layers, [settings])`" with that stack.

5.4 Usage, Evaluation & Weights

The usage of this framework is demonstrated using an example script.

Script	**Explanation**
`from model_utils.model_runner import ModelRunner` `from keras.layers import Dense, Flatten, TimeDistributed, Conv2D, Activation, MaxPooling2D`	Import the "`ModelRunner`" class and any layers required for the model from Keras.
`mr = ModelRunner('simple_cnn_model')`	Instantiate the "`ModelRunner`".
`main_input = mr.init_input(augment_data = False, sample_data_only = True)`	Get the primary data input layer from the "`ModelRunner`" instance. In this case, the default data augmentation is disabled and only a smaller sample dataset is used.

`layers = TimeDistributed(Conv2D(8, (5, 5), strides=(2, 2)))(main_input)`
`layers = TimeDistributed(Activation('relu'))(layers)`
`layers = TimeDistributed(MaxPooling2D(pool_size=(4, 4), strides=(4, 4)))(layers)`
`layers = Flatten()(layers)`
`layers = Dense(1, activation='linear')(layers)`
	Use the Keras layers to assemble a neural network on top of the previously mentioned primary input layer.

`mr.train(layers, epochs=5, batch_size=1)`	Finally, use the "`ModelRunner`" instance to train the assembled stack of layers and perform a evaluation of the networks performance. In this example, no mini-batching (batch_size = 1) is performed.

Table 3: example model script

After executing a model-script, like the one shown and explained in table 3, the console will be filled with various evaluation metrics and other debugging information. This information is printed after each epoch. If this behaviour is turned off, there is still the option to print a final evaluation after the training complets. Additionally, to facilitate the comparison of different models and training sessions with different parameters a TensorBoard log file is created and saved to "`../logs/[model_name]/[date & time]`". It can be viewed and compared to other runs by executing "`tensorboard --logdir=../logs/`". TensorBoard plots the

changes in the various metrics over the training epochs of all models in the logdir against each other.

Finally, four custom plots are drawn and saved to "`../logs/[model_name]/[date & time]/plots`". These plots are made after initialization, after each epoch and one last time after the training has completed. There are two different kinds of plots, one of which is a box plot of the actual values and the predictions. The other kind is a scatter plot of the residuals (i.e. errors) of each prediction against the actual values. Two versions of each plot are created. One version is limited to the value range of sensible predictions in order to see more details. The other one has no limits, thus showing all data, even in the case of very high or low predictions. These plots help discover systematic errors in the predictions of a model.

The best performing weights for the trained model are saved to "`../models_trained/[model_name]_[date & time].hdf5`".

5.5 Customization

There is a lot of customizability built into the framework through optional parameters of functions, as indicated in Chapter 5.3, Framework Architecture. The various options are documented in the code. Additionally, the framework has been designed with readability and extensibility in mind. Its modular design thus supports the addition of further settings and options with minimal effort. Important existing options are explained in their usage and effect within the source code referred to in Chapter 8.1, Source Code.

6. Experimental Models

We developed and evaluated several models with various architectures and differing hyperparameters, using the framework described above. One important reason to do so, was to test against and orient the development of our framework according to real world needs. Moreover, these models are another steps towards the development of a CNN outperforming the predictions made by experts and conventional image feature analysis in the field of solar flare prediction. The following chapter will quickly describe these models and present their MAE. All of the models mentioned here and their detailed evaluations, as well as some additional model variations, can be found in the repository mentioned in Chapter 8.1, Source Code. The evaluation logs there contain various additional metrics in addition to the MAE that is mentioned here.

6.1 Baseline

The baseline is a zero-rule model, which means it makes a static prediction independent of any input it is provided with. The optimal static value for the MAE metric is the median of the goes flux values in the training data. With this dataset, that means the model always predicts a peak flux of 5.3e-07. Using this value, the model scored a MAE of *1.5e-05* on the evaluation data set. So far, beating the performance of this baseline model remains an open opportunity [22].

6.2 Bolzern Competitive Model

Roman Bolzern developed the first CNN architecture using this dataset [8]. The model is fairly wide and deep, using many advanced machine learning techniques. Table 4 details a few important design parameters of the model. Note that the model scored a MAE of *3.6e-5* [4], which is about two times worse than the baseline.

MAE	3.6e-5
Channels	magnetogram, 304, 131, 1700, date
Points in Time	4
Encoding	One-Hot over 7 Value Ranges
Weighting	Sum of Samples / Samples per Range
Layers	11 layers with weights. Arrangement of Conv2D, BatchNormalization, SeparableConv2D, MaxPooling2D, GlobalMaxPooling2D and Dense Layers.

Table 4: Bolzern model MAE and design parameters

6.3 Simple CNN

The Simple CNN model is, as its name suggests, a small convolutional model with only one convolution layer. It uses the magnetogram images of all of the four available points in time as input. The first layer is a convolution layer with a filter size of 5x5 and a stride of two. That means for each pixel in the output image, the filter is moved two pixels in the input image, thus reducing the width and height of the image by half. All four points in time are analysed using the same filters, which means all four input images are treated the same during the convolution and the following max pooling layer with a filter size of 4x4. Finally everything is combined linearly into a single output neuron.

This model achieved a MAE of *2.2e-5*. Despite its simplicity it outperforms the Bolzern model by about 30%, while just slightly missing the baseline MAE. This performance makes sense, because, as indicated by our framework, this model converges to predicting the same static value for almost all input images. It still does not meet the performance of the baseline exactly, as the value it converges to is not quite the mathematical optimum of the median across all training data.

MAE	2.2e-5
Channels	magnetogram
Points in Time	4
Encoding	No Encoding
Weighting	No Weighting of Samples
Layers	2 layers with weights. Arrangement of Conv2D, MaxPooling2D and Dense layers.

Table 5: Simple CNN model MAE and design parameters

6.4 Alexnet

Our next experimental model is based on the Alexnet architecture laid out in Chapter 4.5.1. It was designed for RGB images. However, our image data is grayscale. To be able to more closely mimic the successful Alexnet architecture, we used additional satellite image channels in place of the color channels. Moreover, just like in the Bolzern model, we decided to add the date after the flattening layer after the last max pooling layer. This model reaches a MAE of *3.8e-3*, which is about two orders of magnitude worse than the Baseline, the Bolzern model or the Simple CNN model. Table 6 gives a brief overview of the model, the details of which can be inspected in the codebase referred to in Chapter 8.1, Source Code.

MAE	3.8e-3
Channels	magnetogram, 131, 1700, date
Points in Time	4
Encoding	No Encoding
Weighting	No Weighting of Samples
Layers	10 Layers with weights. Arrangement of Conv2D, MaxPooling2D and Dense Layers.

Table 6: Alexnet MAE and design parameters

6.5 VGG16 with Transfer Learning

The last experimental model that is presented in this paper is an adaptation of the VGG16 architecture describe in Chapter 4.5.2. For the same reasons as with the Alexnet adaptation, that is to be able to more closely imitate the original architecture, we are using additional satellite image channels instead of RGB channels. And just like with the Alexnet model, we are adding the date of the last image before the prediction period.

Compared to the Alexnet adaptation, in addition to the obviously different architecture, there is another significant difference. That is, in this model transfer learning is used. This machine learning technique describes the process of using an existing network architecture, and then initializing the weights using pretrained weights from a different dataset. After initialization, multiple approaches for the learning phase exist. For example, all neuron weights could be open to adjustments, or only those in the last few layers. This transfer learning technique has been shown to not only work well in general, but also specifically for small datasets. [23]

For our network, we have chosen to only train that last three layers of the network after the initialization with the pretrained weights. As with previous models, we are adding the date. However, unlike previous models we are only using the last datapoint in time before the prediction period. This model achieves a MAE of *3.4e-3* and therefore outperforms the Alexnext model described earlier just barely. But just like the Alexnet model, it is about two order of magnitudes behind all other models. Table 7 outlines a few important design parameters of our VGG16 adaption with transfer learning.

MAE	3.4e-3
Channels	magnetogram, 131, 1700, date
Points in Time	The last before the predicted period
Encoding	No Encoding
Weighting	No Weighting of Samples
Layers	Predefined VGG16 architecture (including Conv2D, MaxPooling2D, ect.). Plus 4 Dense layers with weights.

Table 7: VGG16 MAE and design parameters

7. Accomplishments & Outlook

In the following chapter we take a look back and list our contributions towards the long term project and its goals. Additionally, we also consider the future and list opportunities that would likely be worthwhile to pursue in order to make further progress.

7.1 Accomplishments

We have made major strides towards the goal of using CNNs to predict the peak flux in the next 24 hours after satellite images were taken. More concretely, we:

- Gathered and consolidated extensive information about the application domain, the dataset and convolutional neural network theory. See Chapter 2, Domain Overview: Solar Flare Prediction, and 4, Machine Learning Theory.
- Investigated the dataset and previous work. As part of this we discovered and fixed breaking bugs in existing source code. Additionally, we were able to confirm that the

dataset and the goal of predicting the peak goes flux is suited to the application of convolutional neural networks. See Chapter 3, Dataset Analysis.
- Developed and extensively used a debugging and evaluation framework for Keras machine learning models on the previously mentioned dataset. This is our most important contribution to the continuation of this project. See the Chapter 5, Development & Evaluation Framework, for further explanation and the source code referred to in Chapter 8.1, Source Code.
- Using this framework, we considered, developed and evaluated several different models to solve the task of predicting the peak goes flux value over a 24 hour prediction period. These models can inform further work, since they show what avenues where already explored. Moreover, they can also serve as jumping off points for further development. See Chapter 6, Experimental Models, the source code referred to in Chapter 8.1, Source Code.

7.2 Outlook

As discussed in Chapter 3, Dataset Analysis, we conclude that the dataset and task specifics pose a significant challenge, but are still suited to a solution using a convolutional neural network. Considering the benefits of machine learning solutions in general and in this specific application domain, laid out in Chapter 2, Domain Overview: Solar Flare Prediction, and 4.1, Benefits of Machine Learning, we recommend investing further resources into the continuation of this project. In the following chapter we will present several approaches to do so.

7.2.1 Data Collection

One of the aspects that makes this project challenging, is the comparatively small dataset for a regression problem on images. It follows therefore, that the problem would be significantly easier if there was a larger dataset available.

However, we cannot simulate or generate solar flares. Thus we are limited to selecting from the available satellite images and solar flare data. For the existing dataset, this has been done with considerate effort and care, as discussed in Chapter 3, Dataset Analysis.

If however, a possible continuation of this project comes to the conclusion, that the amount of data is preventing better results, it should be possible to extend the timeframe of the dataset. As it stands, data from 2012 to 2017 is included.

7.2.2 Data Augmentation

Another solution to the problem of not having enough data would be employing additional data augmentation methods, to get more training out of the same base data. However, this has to be done very carefully, because, as mentioned previously, the network might just learn to recognize the areas that have flared in the past instead of learning to predict solar flares.

Even more importantly, if the data augmentation methods, such as is the case with horizontal flipping over more than one point, alter the underlying physical indicators and properties of the sun, it is unlikely that this additional data will help the network make better predictions in

general. Augmentation methods will have to be carefully selected and tested one at a time, to make sure the performance of the model on the evaluation data set actually improves. [4]

One concrete augmentation method that we have not tried but would like to mention at this point, is making minor adjustments to the brightness of the images. The idea is to prevent the network from learning about or being confused by the sensor degradation [24] onboard the SDO and the solar cycle [25]. Both of these factors lead to darker images over time, which however does not seem to have an effect on the observed peak flux values.

7.2.3 Model Development

There are many yet untested model architectures, that could be tried. For example, there are more VGG variations and different contemporary approaches, such as ResNet, that we have not yet tried. Moreover, custom models might prove successful as well. As this field is developing extremely fast, there are likely even newer promising architectures that could be employed, such as ensemble architectures. [26] [27]

It is not just about architectures though, it is also worthwhile to try some of the different optimizers Keras provides, or change the activation functions used.

7.2.4 Encoding Functions

Our architecture supports the use of different encoding and decoding functions. The true labels are encoded before they are fed to the network and the outputs of the network are decoded to perform the evaluation. It might be worthwhile to come up with additional encoding methods or try different parameters for the existing ones.

7.2.5 Implementing Regression by Classification

As described in Chapter, 4.3 Regression by Classification, there is a strong possibility of improving the performance of the existing models by applying the regression by classification technique discussed in Chapter 4.3. Our framework has built in support for the required encoding and decoding functions, as indicated in Chapter 5.3, Framework Architecture.

Using this functionality to develop models using the regression by classification technique is in our opinion the most promising and worthwhile area to work on.

8. Appendix

8.1 Source Code

Any and all source code can be found on the following GitLab page: https://gitlab.fhnw.ch/patrick.ackermann/SFPatFHNW. The repository also contains the final version of this report.

8.2 Data

All data that was used during this project, along with additional information about the dataset can be found at the following address: https://i4ds.github.io/SDOBenchmark/#

8.3 References

[1] Henry, “A brief introduction of AlphaGo and Deep Learning: How it works,” *Medium*, 03-Oct-2017. [Online]. Available: https://medium.com/@kinwo/a-brief-introduction-of-alphago-and-deep-learning-how-it-works-76e23f82fe99. [Accessed: 21-Jan-2019].

[2] J. Gomes, B. Ramsundar, E. N. Feinberg, and V. S. Pande, “Atomic Convolutional Networks for Predicting Protein-Ligand Binding Affinity,” *ArXiv170310603 Phys. Stat*, Mar. 2017.

[3] A. Bhandare, M. Bhide, P. Gokhale, and R. Chandavarkar, “Applications of Convolutional Neural Networks,” vol. 7, p. 10, 2016.

[4] Roman Bolzern, Michael Aerni, “SDOBenchmark - Solar flare prediction image dataset,” *SDOBenchmark*. [Online]. Available: http://i4ds.github.io/SDOBenchmark/. [Accessed: 15-Jan-2019].

[5] “Space weather,” *Wikipedia*. 13-Jan-2019.

[6] “Solar flare,” *Wikipedia*. 05-Jan-2019.

[7] “Solar Dynamics Observatory,” *Wikipedia*. 31-Oct-2018.

[8] “SDO machine learning dataset.,” 06-Jan-2019. [Online]. Available: https://github.com/i4Ds/SDOBenchmark. [Accessed: 16-Jan-2019].

[9] “Machine learning,” *Wikipedia*. 14-Jan-2019.

[10] “6 areas where artificial neural networks outperform humans,” *VentureBeat*, 09-Dec-2017. [Online]. Available: https://venturebeat.com/2017/12/08/6-areas-where-artificial-neural-networks-outperform-humans/. [Accessed: 21-Jan-2019].

[11] “Convolutional Neural Networks,” *Coursera*. [Online]. Available: https://www.coursera.org/learn/convolutional-neural-networks. [Accessed: 21-Jan-2019].

[12] L. Hulstaert, “Interpreting machine learning models,” *Towards Data Science*, 20-Feb-2018. [Online]. Available: https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f. [Accessed: 16-Jan-2019].

[13] M. Sharma, “What Steps should one take while doing Data Preprocessing?,” *Hacker Noon*, 25-Jul-2018. [Online]. Available: https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa. [Accessed: 21-Jan-2019].

[14] L. Torgo and J. Gama, “Regression by classification,” in *Advances in Artificial Intelligence*, vol. 1159, D. L. Borges and C. A. A. Kaestner, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 51–60.

[15] “CS231n Convolutional Neural Networks for Visual Recognition.” [Online]. Available: https://cs231n.github.io/convolutional-networks/. [Accessed: 16-Jan-2019].

[16] “Machine Learning - Logistic Regression,” *Coursera*. [Online]. Available: https://www.coursera.org/learn/machine-learning/home/week/6. [Accessed: 21-Jan-2019].

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing*

Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.
[18] “ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks,” *CV-Tricks.com*, 09-Aug-2017. .
[19] “Very Deep CNNS for Large-Scale Visual Recognition.” [Online]. Available: http://www.robots.ox.ac.uk/~vgg/research/very_deep/. [Accessed: 20-Jan-2019].
[20] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” *ArXiv14091556 Cs*, Sep. 2014.
[21] D. Frossard, “VGG in TensorFlow · Davi Frossard,” 34:54 - EDT-400AD. [Online]. Available: https://www.cs.toronto.edu/~frossard/post/vgg16/. [Accessed: 17-Jan-2019].
[22] “The SDOBenchmark Dataset.” [Online]. Available: https://kaggle.com/fhnw-i4ds/sdobenchmark. [Accessed: 19-Jan-2019].
[23] P. Marcelino, “Transfer learning from pre-trained models,” *Towards Data Science*, 23-Oct-2018. [Online]. Available: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751. [Accessed: 20-Jan-2019].
[24] “Many of the Images Are Darker and Darker in Comparison with Previous Releases and with the SDO Page · Issue #136 · Helioviewer-Project/helioviewer.org,” *GitHub*. [Online]. Available: https://github.com/Helioviewer-Project/helioviewer.org/issues/136. [Accessed: 16-Jan-2019].
[25] “Solar cycle,” *Wikipedia*. 05-Jan-2019.
[26] V. Fung, “An Overview of ResNet and its Variants,” *Towards Data Science*, 15-Jul-2017. [Online]. Available: https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035. [Accessed: 17-Jan-2019].
[27] “Ensemble learning,” *Wikipedia*. 26-Dec-2018.

8.4 Declaration of Honesty

We hereby confirm that the present Report is solely our own work and that if any text passages or diagrams from books, papers, the Web or other sources have been copied or in any other way used, all references – including those found in electronic media – have been acknowledged and fully cited.

Signature: _____ Date & Place: _____

Signature: _____ Date & Place: _____