Image Classification with Deep Learning

Thesis for degree in Master of Science in physics

By

ANDRIY BOROVKOV

Universität Hamburg

DER FORSCHUNG | DER LEHRE | DER BILDUNG

Fakultät für Mathematik, Informatik und Naturwissenschaften
UNIVERSITÄT HAMBURG

APRIL 2017

1. Gutachter: Prof. Dr. Peter Schleper
2. Gutachter: Jun-Prof. Dr. Gregor Kasieczka

Abstract

A deconvolutional network approach for the visualization of the learning of a galaxy image classifier is presented here. For the classification, a convolutional neural network with viewpoint extraction in preprocessing is used. The network was able to achieve a RMSE of 0.084 for the training data set and 0.082 for the validation data set. The deconvolution technique shows that the network is able to highlight the morphological features of the galaxies through the convolutional layers by isolating and enhancing different objects for different views of a galaxy.

Zusammenfassung

In dieser Arbeit wird ein Neuronales Entfaltungsnetz vorgestellt, welches die erlernten Merkmale eines Neuronalen Faltungsnetzes (engl. convolutional neural network (CNN)) visualisiert, das für die Klassifizierung von Galaxiebildern nach Morphologien trainiert wurde. In der Vorverarbeitung des CNN wird eine spezielle Bildausschnittextrahierung durchgeführt. Das Neuronale Netz erreichte einen quadratischen mittleren Fehler von 0.084 für den Trainingsdatensatz und 0.082 für den Validierungsatensatz. Durch die Entfaltung wird gezeigt, dass das CNN die morphologischen Merkmale der Galaxien in den Faltungslagen isoliert und unterschiedliche Objekte für verschiedene Bildausschnitte hervorhebt.

Table of Contents

CHAPTER 1

INTRODUCTION

Machine learning in general, and neural networks in particular, are on their way to omnipresence in several areas of our everyday lives, ranging from Google's email inbox filtering [1] to consumer credit-risk models in the financial sector [2]. They have also been employed in natural science, in particular in high energy physics. However, there is much potential in machine learning methods that has not been utilized yet. One important reason why physicists are careful with their commitment to machine learning is that these methods are not yet fully understood. There is still insufficient insight into the internal operation and behavior of neural networks and how they achieve their predictive power.

This analysis aims to provide insight into one of the most successful types of neural networks in the field of image and pattern recognition, the convolutional neural networks. To do this, a deconvolutional network is built and its output analyzed.

Chapter 2 provides a brief introduction into the important concepts of machine learning and neural networks as a part of machine learning. In this thesis, a convolutional neural network will be tasked to classify image data of galaxies into morphological categories. Chapter 3 gives insight into how the data used for this analysis were acquired and Chapter 4 describes the architecture of the neural network model. Subsequently, the hyper parameters of this network will be studied in Chapter 5. The deconvolutional network and its results will be discussed in Chapter 6. Finally, an outlook for future research and a summary are given in Chapters 7 and 8, respectively.

CHAPTER 2

DEEP LEARNING

Deep learning is an ambiguous term, as it has gone through several different meanings throughout the years. A broad, contemporary definition can be found in Goodfellow's book *Deep Learning* [3]. There, it is defined as the decomposition of complex concepts into simple ones and the recombination into new complex concepts. An algorithm therefore has to establish a hierarchy of concepts. A visualization of this hierarchy would be a multi-layered graph, that can be called "deep" in a graph theory context [4]. In this thesis, the term will be used in a more narrow sense, as proposed in Skansi's book *Introdution to Deep Learning* [5]. There, it refers to *deep artificial neural networks* as a subfield of machine learning. In this chapter, the crucial concepts for understanding deep learning are provided, following the description in [3].

2.1 Machine Learning

Machine learning is a subfield of computer science. At its core, it is the foundation for a set of statistical tools that estimate complicated functions by learning from data. Machine learning can be divided into two main approaches, supervised and unsupervised learning. Supervised learning generally means that the program is given both input and the desired output, for example, pictures of objects with corresponding labels of what is depicted. The goal of the learning (or training) is to construct a map between those two. In contrast to supervised learning, the unsupervised learning approach does not provide the program with the correct output. Here, the goal of training is to find structure inside the given input; it is used, for example, in autoencoders [6].

With this in mind, one useful definition of machine learning is given by Tom Mitchell in his book *Machine Learning* [7]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T,

as measured by P, improves with experience E."

The **task** $\boldsymbol{T}$ is the learning goal of the algorithm. It is defined by the user and is usually communicated to the algorithm by providing an example of how an event is to be processed. Here, event means a set of features describing a single data point. It is usually denoted as a vector $x \in R_n$ with entries x_i representing features of the event. In the analysis of galaxy identification presented in this thesis, each event is an image of a galaxy and its features are the values of the pixels in the image. The task is classification, where the algorithm is asked to assign a probability to the image with which it belongs to any of the k categories. To solve this task, the learning algorithm produces a function $f : R^n \to \{1, ..., k\}$, where f outputs a classifier distribution.

The **performance measure** $\boldsymbol{P}$ is the metric by which the competence of the algorithm is evaluated. The metric can be chosen by the user and generally has to be tailored to the task T. In the case of classification, the performance measure P usually employs a loss function that quantifies the disagreement between the predicted and true output. In a simple case, the loss function could be the amount of correctly classified events in proportion to the total number of events. In more sophisticated approaches, it is a continuous-valued score for each event. The loss function for the presented analysis is discussed in chapter 3.3. The performance is tested on data that were not included in the training sample. This independent sample is called the validation data set or validation sample. To estimate how well the algorithm will perform in a broader range of applications, additional performance tests not related to the chosen loss function are performed.

The **experience** $\boldsymbol{E}$ encompasses the provided data set and any additional information that the machine learning algorithm can use to learn. It is here where supervised and unsupervised algorithms can differ. They are not entirely exclusive and applications that combine both methods, called semi-supervised learning, exist. It is still a useful paradigm to categorize different machine learning goals. Unsupervised learning algorithms are presented with datasets with many features and learn properties of the structure of the dataset. They produce a probability distribution $p(x)$ of a random event x. Supervised learning algorithms are trained on datasets containing features, where each event is also associated to a predetermined output via a label. They process several events x and an associated value or vector y, then learn to predict y from x. The dataset does not have to be fixed either. Other machine learning techniques, like reinforced learning, make use of a feedback loop that provides new data in response to the learning system. As noted previously, the presented analysis is based on supervised algorithms.

2.1.1 Training

Training a machine learning algorithm can be seen as approximating two functions $y(x)$ and $\hat{y}(x)$, where the algorithm tries to find the closest distance from $y(x)$ to $\hat{y}(x)$ in a given metric. The basic principles of training can be illustrated with a linear regression:

$$\hat{y} = w^T x \ . \tag{2.1}$$

Here, w is a vector of parameters that the algorithm can optimize, which in a machine learning context, are called weights. They determine how features x_i correlate with the output $\hat{y}$; finding the closest "distance" between $\hat{y}$ and y is called predicting y from x. There are many possible ways for an algorithm to optimize the parameters. In the provided example, a possible learning method can be to minimize the mean squared error (MSE) from equation 2.2 on the training set x:

$$\mathrm{MSE} = \frac{1}{n}\sum_i (\hat{y} - y)_i^2 \ . \tag{2.2}$$

Here, n is the number of events x with features i. $\hat{y}$ is called the prediction of the model on the training set. The MSE is minimized by solving the gradient with respect to weights w for 0:

$$\nabla_w \mathrm{MSE} = 0 \ . \tag{2.3}$$

To validate the training process, the MSE is also calculated for an independent validation set x_{val} with n_{val} events, that the algorithm does not use for training:

$$\mathrm{MSE}_{val} = \frac{1}{n_{val}}\sum_i (\hat{y}_{val} - y_{val})_i^2 \ . \tag{2.4}$$

Here, y_{val} is the set of correct output values and $\hat{y}_{val}$ the algorithm prediction for y_{val} based on x_{val}. The algorithm iterates the training and validation process until the error is sufficiently small, a criterion specific to the task.

In general, the user has to define a model that describes the output y in terms of input x, like the linear regression above, and a learning method. Models that are used in deep learning are described in chapter 2.2. The learning method employed for this thesis, called Gradient-Based Learning, is described in chapter 2.2.4.

2.1.2 Overfitting and Underfitting

A successful machine learning algorithm should perform well on unseen input samples that are of the same type as the training and validation data sets. Therefore, the performance of the algorithm should not only be evaluated on its ability to minimize the

training error, but also on its ability to minimize the difference between the training error and validation error. This difference is called the generalization error and is discussed in more detail in chapter 2.1.4. Both, the training and generalization errors highly depend on the representational capacity of the algorithm, which is a measure of how well the algorithm is able to adapt to a range of outputs. In the example above, a linear regression was used as the model for training. If the user wants to model data whose underlying distribution is a higher order polynomial, the linear model will not be able to describe the desired behavior. The training error will never be sufficiently small. This problem is called underfitting the model.

In theory, if the algorithm's representational capacity is allowed to be arbitrarily high, it will eventually perfectly fit any given finite event set to the outputs. But this limits the algorithm's ability to perform on new data, since in this regime it is overly specialized to the training data. If the algorithm has the capacity to achieve a small training error but is not able to make the generalization error small, this problem is called overfitting. The key to avoiding overfitting is to find a sufficiently complex model that is still able to generalize. Figure 2.1 shows the typical relationship between capacity and error.

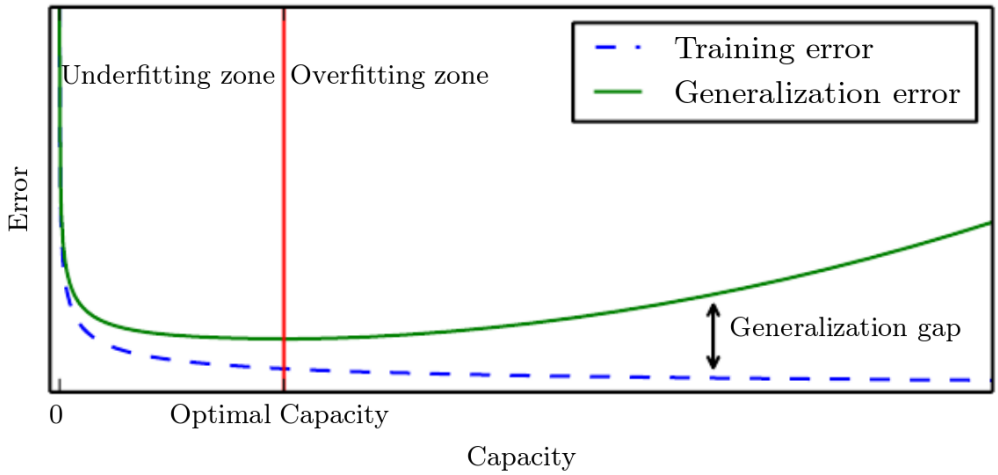


FIGURE 2.1. Training error and generalization error as functions of capacity. Between the underfitting and overfitting zone an optimal capacity exists. Figure taken from [3].

2.1.3 Bayes Error and The No Free Lunch Theorem

There are statistical constraints on the classification performance of machine learning algorithms that have to be considered.

Assume there is an ideal model that matches the true underlying distribution of a data set X. The model will still incur errors on classification for multiple reasons. The mapping from events $x \in X$ to y may be inherently stochastic or X may be incomplete in terms of

predicting y, because y may also depend on other variables outside of those that can be obtained from X. Finally, the distribution describing X may be noisy, meaning that an event x can belong to more than one class. The smallest possible error an ideal model can obtain is called Bayes error. In this limit, machine learning only offers probabilistic rules, i.e. it promises to find rules that are *probably* correct about *most* members of the set they concern.

The second constraint to consider is called the No Free Lunch Theorem [8]. It states that no one machine learning algorithm is inherently better performing on unseen data than any other algorithm if averaged over all possible data-generating distributions. This means that there is no single best algorithm to do all possible tasks. Therefore, assumptions about the given data distributions have to be made. This way algorithms can be tailored to the task.

2.1.4 Regularization

To address the aforementioned issues, the machine learning algorithm has to be designed with a set of preferences that align well with the given task. Building and modifying these preferences with the goal of decreasing the generalization error but not the training error is called regularization. Some important aspects of regularization are introduced here.

One way to efficiently adjust the capacity of the model is to examine the bias and variance of the algorithm. The bias of an algorithm A for sample size m at event x is defined as

$$\mathtt{bias}(A,m,x) = \mathtt{E}(\hat{f}(x)) - f(x). \tag{2.5}$$

Here, f is the true distribution underlying the data, $\hat{f}$ the prediction for the data and $\mathtt{E}(\hat{f}(x))$ the expectation value of $\hat{f}(x)$ at event x. The prediction is called unbiased, if $\mathtt{bias}(A,m,x) = 0$. This implies, that $\mathtt{E}(\hat{f}) = f$.

The variance (`var`) is given by

$$\mathtt{var}(A,m,x) = \mathtt{E}\left[\left(\hat{f}_S - \mathtt{E}(\hat{f}(x))\right)^2\right], \tag{2.6}$$

where S are all training samples of size m. The variance describes random variations between different training sets S, that can result from noise in the training data, varying numbers of events in classes, or random behavior in the algorithm itself, such as different initial parameters. Both of the quantities above contribute to the generalization error ($\mathtt{Err}_{gen}$) as discussed in [9]:

$$\mathtt{Err}_{gen}(A,m,x) = \mathtt{bias}(A,m,x)^2 + \mathtt{var}(A,m,x) \tag{2.7}$$

Figure 2.2 shows how this connection can be applied to find the optimal capacity to minimize the generalization error.

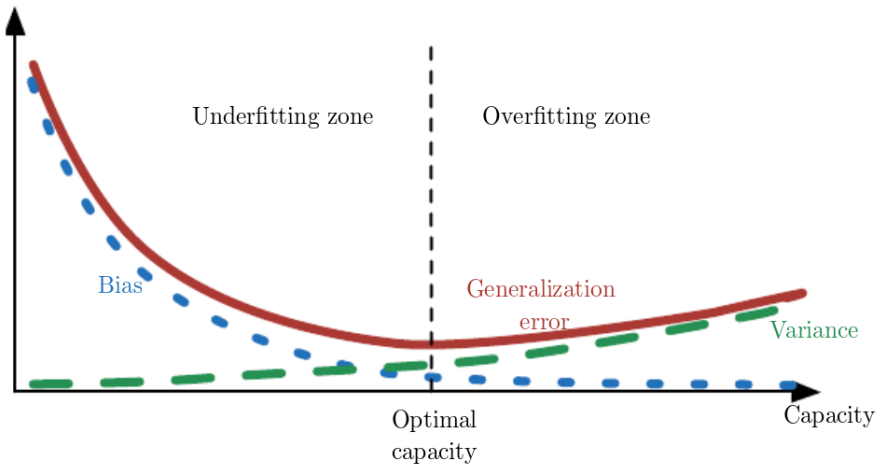


FIGURE 2.2. Generalization error as a function of capacity, bias and variance [3].

Commonly used regularization methods to avoid overfitting include Dropout [10], which is also employed in this thesis and described in chapter 2.3.4, and introducing variations into the training data during preprocessing, as described in chapter 4.1.1.

2.2 Artificial Neural Networks

To build a machine learning algorithm, the user must define a model, a cost function, and an optimization procedure that befit the structure of the data. This chapter deals with the most prominent kinds of models in machine learning, artificial neural networks, and learning algorithms that can be used with these models.

As mentioned before, a machine learning algorithm is used to approximate a given function f. In the case of a classification, f maps an input x to a category y. An algorithm will derive $\hat{f}$ with weights w such that f approximates

$$y = \hat{f}(x; w). \tag{2.8}$$

In general, $\hat{f}$ can be a composite function of any number of functions $\hat{f}_i$ in the form

$$\hat{f} = \hat{f}_n \circ ... \circ \hat{f}_3 \circ \hat{f}_2 \circ \hat{f}_1. \tag{2.9}$$

This structure is called a network, where $\hat{f}_1$ corresponds to the first layer, also called input layer, $\hat{f}_2$ to the second layer, and so on. The final layer $\hat{f}_n$ is called the output layer. The $n-1$ layers between input and output layers are called hidden layers, because their behavior is only implicitly constrained by the training data, as the data do not show the desired output for each of those layers. The number of layers n defines the depth of the network. This simple chain structure is only one possible way to build a network and was first introduced in 1958 by Frank Rosenblatt [11]. In general, any connection structure between the different functions is possible. Some such alternative examples are deep residual networks [12], long/short term memory networks [13], and self-organized networks, that can decide which of its layers suit a given task [14].

All of the above networks are often referred to as artificial neural networks because they are loosely inspired by neuroscience. It is an established name used by the community. However, the goal of neural networks is not to model the human brain. Neural network research is fundamentally based on and driven by mathematical and engineering disciplines rather than biological brain function.

2.2.1 Types of Units and Activation Functions

Each layer in a neural network consists of many units that act in parallel, where each unit represents a vector-to-scalar function. Most units can be described as accepting a vector of inputs x, computing the affine transformation

$$z = W^T x + b, \tag{2.10}$$

where the matrix W describes the mapping from x to z with bias b applied to the transformation. Afterwards, a nonlinear function $g(z)$, called the activation function, is applied element-wise on z. The design of units is an active area of research and is not yet supported by definitive guiding theoretical principles. Most hidden units are distinguished from each other only by the choice of the form of the activation function $g(z)$.

ReLU

The default choice for an activation function in modern neural networks is the max function

$$g(z) = \max\{0, z\}\,. \tag{2.11}$$

A unit that employs this function is called a rectified linear unit (ReLU) [15]. ReLUs are quickly optimized since the derivative is either 0 or a positive constant value through the domain. This makes the gradient direction far more useful for learning than it would be with activation functions with non-vanishing and higher order derivatives. One drawback to ReLUs is that they cannot learn via gradient-based methods on examples for which the activation is zero.

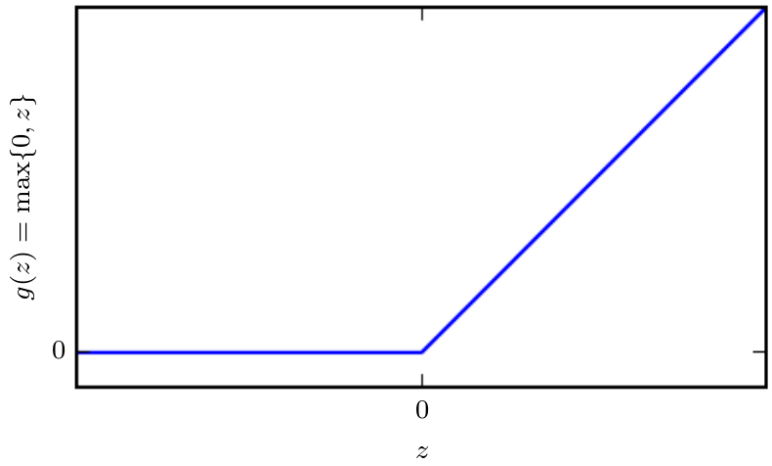


FIGURE 2.3. A rectified linear unit outputs zero across half its domain [3].

Maxout

Maxout units are generalized rectified linear units [16]. Instead of applying an element-wise function $g(z)$, maxout units divide z into groups of k values. Each maxout unit then outputs the maximum element of one of these groups:

$$g(z)_i = \max z_j,\, j \in \mathbb{G}^{(i)}\,. \tag{2.12}$$

Here, $\mathbb{G}^{(i)}$ is the set of indices for group i , $\{(i-1)k+1, ..., ik\}$. This provides a way of learning a piecewise linear function that responds to multiple directions in the input x

space. A maxout unit can facilitate learning for a piecewise linear, convex function with up to k pieces. Since each maxout unit is parametrized by k weight vectors instead of just one, it needs more regularization than an ReLU.

Sigmoid

The sigmoid function is used to represent a probability distribution over a binary variable. It is defined as

$$\sigma(z) = \frac{1}{1+\exp(-z)}. \tag{2.13}$$

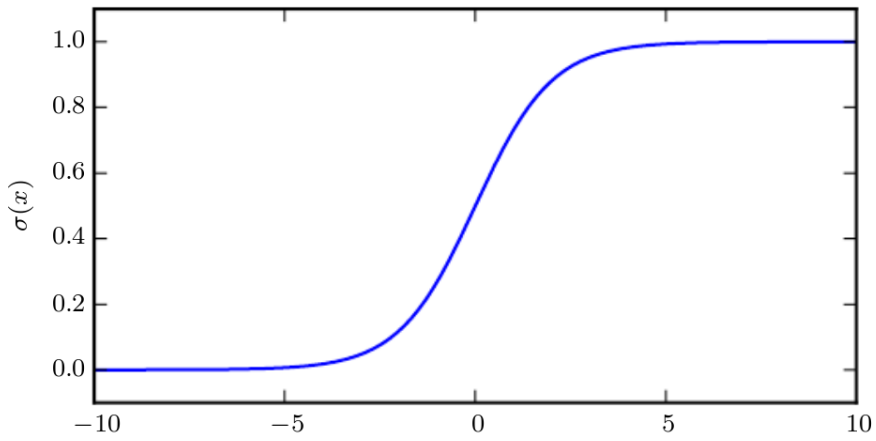


FIGURE 2.4. The sigmoid function [3].

This activation function saturates to 0 when z becomes very negative and saturates to 1 when z becomes very positive. For large absolute values of z, the gradient can become too small to be useful for learning even if the training data abundantly populate these regions.

Softmax

The softmax function of z, is a generalization of the sigmoid function that represents a probability distribution over a discrete variable with n possible values. Softmax functions are often used as the output units of a classifier. Formally, the softmax function is given by

$$\texttt{softmax}(z) = \frac{\exp(z_i)}{\sum_j \exp(z_i)}. \tag{2.14}$$

Cost functions that do not use a log to undo the exp of the softmax cause a failure to learn when the argument to the exp becomes very negative, causing the gradient to vanish.

2.2.2 Back-Propagation

By default, information in a neural network propagates in a forward direction from the input nodes, through the hidden layers, and into the output node. This is called forward propagation. During training, forward propagation can continue until it produces a cost in an output node. The back-propagation algorithm [17] allows the information from the cost to then flow backward through the network in order to compute the gradient. The back-propagation algorithm computes the gradient of scalar z with respect to one of its ancestors x in the graph. First, the gradient with respect to z is taken. This is simply $\frac{dz}{dz} = 1$. From here on, the next gradient with respect to the next parent of z in the graph is obtained by multiplying the current gradient by the Jacobian of the operation that produced z. This is repeated backward through the graph until x is reached. When two or more paths can be chosen, the gradients of those paths are simply summed up.

2.2.3 Cost Functions

As mentioned before, a learning algorithm needs a metric with which to evaluate the distance between two function. Such a metric is called a cost function or loss function in neural networks. Similar to the euclidean metric being the default choice of metric in euclidean space, the first choice for a cost function in a neural network is the cross-entropy between the training data and the predictions of the model, which makes use of the principle of maximum likelihood.

In most cases, parametric models define a distribution. Let $p_{model}(x;\theta)$ be a family of probability distributions over the same space indexed by θ that maps any configuration x to a real number estimating the true probability $p_{data}(x)$. The maximum likelihood θ_{ML} is then defined as

$$\theta_{ML} = \arg\max_{\theta} p_{model}(\mathbb{X};\theta). \tag{2.15}$$

Other popular cost functions are quadratic functions, such as the mean-square-error (MSE) and the root-mean-square-error (RMSE), which are shown in 3.3.

2.2.4 Gradient-Based Learning

Neural networks are usually trained using iterative optimizers based on Cauchy's gradient-descent [18]. Such methods merely drive the cost function to a very low value, in contrast to linear equation solvers used to train linear regression models, as well as convex optimization algorithms with global convergence guarantees used to train logistic regression. Convex optimization converges starting from any initial parameters. Stochastic gradient descent applied to non-convex loss functions has no such convergence guarantee and is sensitive to the values of the initial parameters. For feed-forward neural networks, it is

important to initialize all weights to small random values. The biases may be initialized to zero or to small positive values.

Important for this work is the momentum learning method introduced in [19]. It is designed to accelerate learning, especially in the case of high curvature, small but consistent gradients, or noisy gradients. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. A variant of the momentum algorithm that was inspired by Nesterov's accelerated gradient method [20] was introduced in [21]. The difference between Nesterov momentum and standard momentum is where the gradient is evaluated. The former can be interpreted as attempting to add a correction factor to the standard method of momentum.

2.2.5 Data Structure

Frequently, data sets are too large be processed as a whole. Therefore, the data are provided to the network in batches. These batches take the form of multidimensional arrays. That way, the data can be treated as a matrix $X_{i,j} \in \mathbb{R}^{i \times j}$, where rows i correspond to events x and columns j correspond to features of those events. The labels are provided in a vector y, where y_i contains the label for event i. The above only holds if the data are homogeneous, meaning all the events are of the same dimensionality. Methods for handling heterogeneous data can be found in [3]. In the presented analysis of galaxy data, the input is homogeneous.

Any type of input can be represented by a flattened vector [22], but in this case the data may lose intrinsic structure. An RGB image, for example, contains two order-sensitive axes for width and height, and one axis that is used to access different views of the data (i.e. the red, green, and blue channels of the color image), for which the ordering does not matter. This must be considered when choosing different layer types for the network, as some of the layers do not preserve this structure.

2.3 Important Layer Types

The key layers employed by the models in the presented work are the *convolutional* layer, the *pooling* layer, and the *dense* layer. A network composed of these layers is very typical for image recognition and is based on networks developed by Krizhevsky et al. for ImageNet [23]. In addition, the dropout layer is introduced because it has proven to be an important means of regularization.

2.3.1 Dense Layer

The dense layer uses the linear set of connections between input and output, described in equation 2.10. All units of a dense layer are fully connected to all units in neighboring layers in such a way that the units take every output from all units in the former dense layer as their input and pass their output to all units in the following layer.

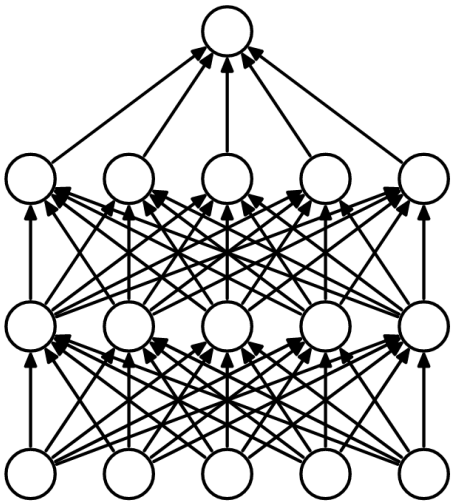


FIGURE 2.5. A densely connected neural net with two hidden layers.

This leads to a very simple matrix-vector-computation, but it also leads to a very large set of trainable parameters. A neural network that consists of dense layers can be very successful for low dimensional data, but computation can become very expensive for high dimensional data like images. To handle such tasks, layers with weight sharing are introduced.

2.3.2 Convolutional Layer

A convolution is a special kind of linear operation on two functions of a real valued argument. A one-dimensional convolution $y(t)$ of two functions $x(t)$ and $w(a)$ is defined as the integral of the pointwise multiplication of these two functions:

$$y(t) = (x \star w)(t) = \int_{-\infty}^{+\infty} x(a)w(t-a)da \ . \tag{2.16}$$

The function $y(t)$ can be seen as a modified version of $x(t)$ weighed by $w(a)$. In a neural network context, $x(t)$ is the input, $w(a)$ the kernel, and $y(t)$ the output or feature map. Since algorithms only compute in discrete steps, the convolution operation has to be discretized:

$$y(t) = (x \star w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \ . \tag{2.17}$$

Here, t is an integer. A convolution can be generalized for multiple dimensions, where functions x and w are defined on a set t of integers. With a 2-dimensional input $x(i,j)$, for example images with width and height coordinates i and j, the convolution can be written in the following form:

$$y(i,j) = (x \star w)(i,j) = \sum_m \sum_n x(i-m, j-n) w(m,n) \; . \tag{2.18}$$

The first prominent implementation of convolutions in neural networks was carried out by Le Cun et al. [24], proving that the discrete convolution has important properties for use in image recognition. It is an operation that preserves the notion of ordering. Only a few input units feed into a given output unit, and parameters are systematically reused, such that the same weights are applied to multiple locations in the input.

The convolution operation can be visualized with the example in figure 2.6. The light blue grid is called the input feature map. The kernel (shaded blue area) slides across the input feature map. In this example, the Kernel is

$$w = \begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix} \; . \tag{2.19}$$

At each location, the product between each element of the kernel and the input element it overlaps with is computed and the results are summed to obtain the output at the current location. The result is called the output feature map (green grid).
The following parameters can be adjusted in a convolutional layer:

- the kernel size, also called filter size in neural networks;
- the step size, which is the distance between two consecutive positions of the kernel;
- zero padding, which is the number of zeros concatenated at the beginning and end of an axis.

The resulting effects of those parameter changes on the output are discussed in detail in [25].

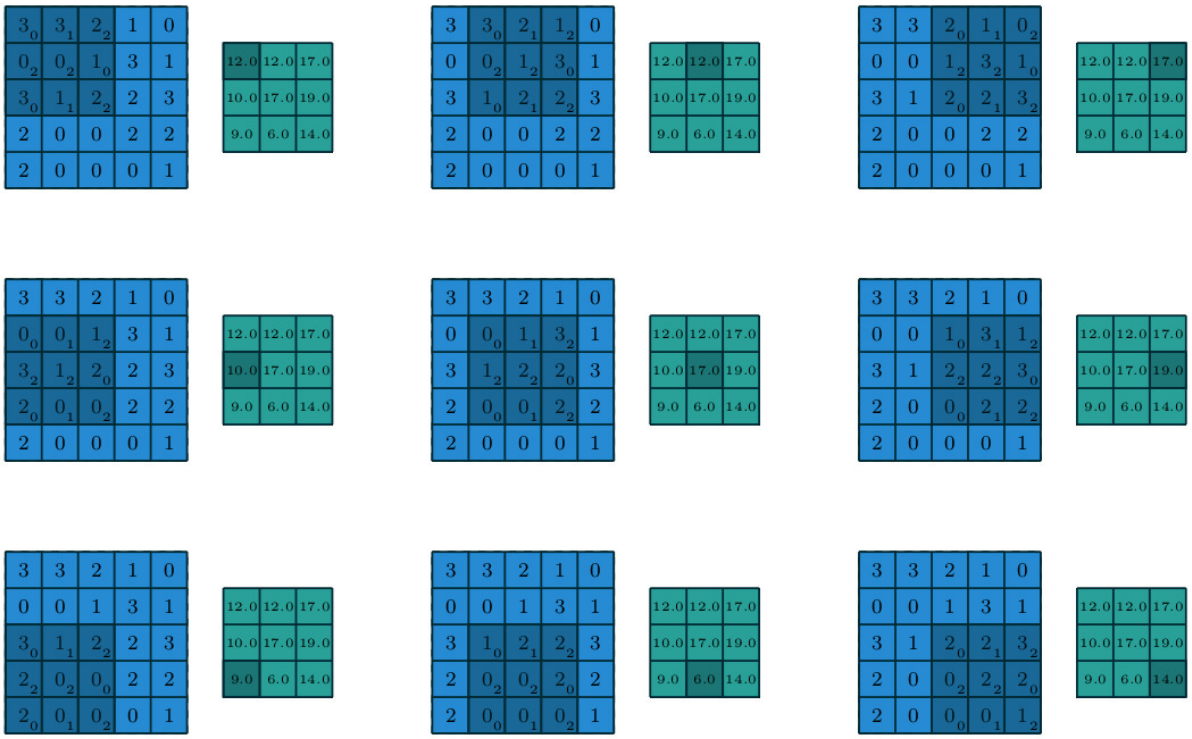


FIGURE 2.6. The convolution operation [25].

2.3.3 Pooling Layer

The pooling layer is a tool that ensures invariance of y under small translations of the input, as well as for down sampling. It is very often used in combination with convolutional layers. The pooling operation computes a summary statistic of nearby inputs using an arithmetic function, and can be freely defined. One of the most popular pooling operations is called max pooling [26], which returns the maximum input within a rectangular neighborhood.

The pooling operation is similar to discrete convolution, but replaces the linear combination with some other function, such as the `maximum` or `average` of the input of the neighbors. Figure 2.7 shows the max pooling operation applied to the same input as in figure 2.6 (shown in blue) and its output (shown in green) for the computation in nine steps.

Similar to the convolutional layer, the pooling layer can also have different kernel sizes, strides and zero padding.

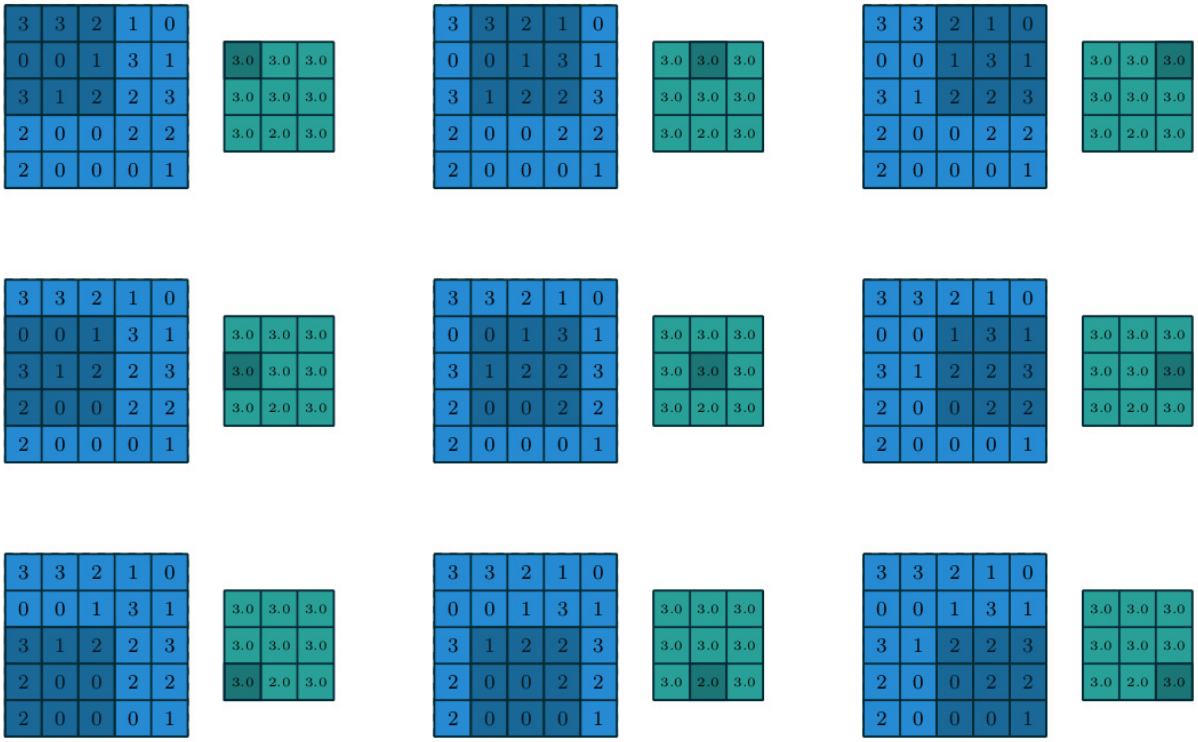


FIGURE 2.7. The max pooling operation [25].

2.3.4 Dropout

Dropout is a method for regularizing of machine learning algorithms. The key idea of a dropout layer is to randomly disable input units after each iteration of the training. A neural network with n units that employs dropout can be seen as a collection of 2^n possible neural networks. These possible networks have a smaller number of units but still share weights such that the total number of parameters is unchanged. Training a network with dropout can be seen as training a collection of 2^n smaller networks with extensive weight sharing.

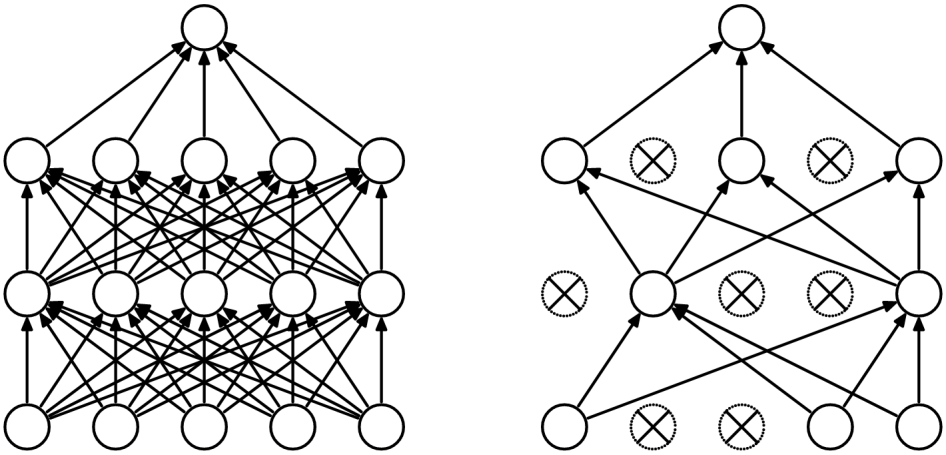


FIGURE 2.8. The same densely connected neural net as in figure 2.5 (left). An example of a smaller network produced by applying dropout (right). Crossed units have been deactivated by dropout [10].

At test time, it is easy to approximate the effect of averaging the predictions of the smaller networks by using a single network without dropout that has smaller weights. This significantly reduces overfitting and gives significant improvements over other regularization methods [10].

CHAPTER 3

GALAXY IMAGE DATA AND THE KAGGLE COMPETITION

The neural networks used in this thesis were trained on galaxy image data that were produced by the Sloan Digital Sky Survey [27] and then categorized by Galaxy Zoo [28]. Finally, the data were processed to be suitable for training and evaluation for use in the Kaggle competition "Galaxy Zoo – The Galaxy Challenge". A broad overview of relevant astronomical units and methods can be found in Ian S. McLean's *Electronic Imaging in Astronomy* [29]. The following chapter describes the data acquisition and processing.

3.1 The Sloan Digital Sky Survey

The Sloan Digital Sky Survey (SDSS), named after the Alfred P. Sloan Foundation, is a multi-spectral imaging and spectroscopic redshift survey of the sky [27]. SDSS has gone through several data taking phases. The first phase started in 2000, with the objective to image a large area of the North Galactic Cap and three stripes in the South Galactic Cap. Objects found in those scans were then evaluated in a spectroscopic survey. Sub-surveys and additional phases focused on certain aspects of the already mapped area, like star distributions and spectroscopy in our own galaxy. Currently the SDSS is in phase IV, with the prime goal of making precision cosmological measurements. This survey will run until 2020.

The relevant data for this thesis was taken in phases I and II as part of the Sloan Legacy Survey. They were published in Data Release 7 (DR7) of the SDSS [30]. The full distribution of the data is shown in Figure 3.1.

This survey used a dedicated 2.5 m wide-angle optical telescope and two digital spectrographs at Apache Point Observatory in New Mexico, United States. Photometric calibration was done using observations of a network of standard stars established by the United States Naval Observatory. For the astrometry, an array of astrometric charge-coupled

devices (CCDs) [29] in the imaging camera was used.

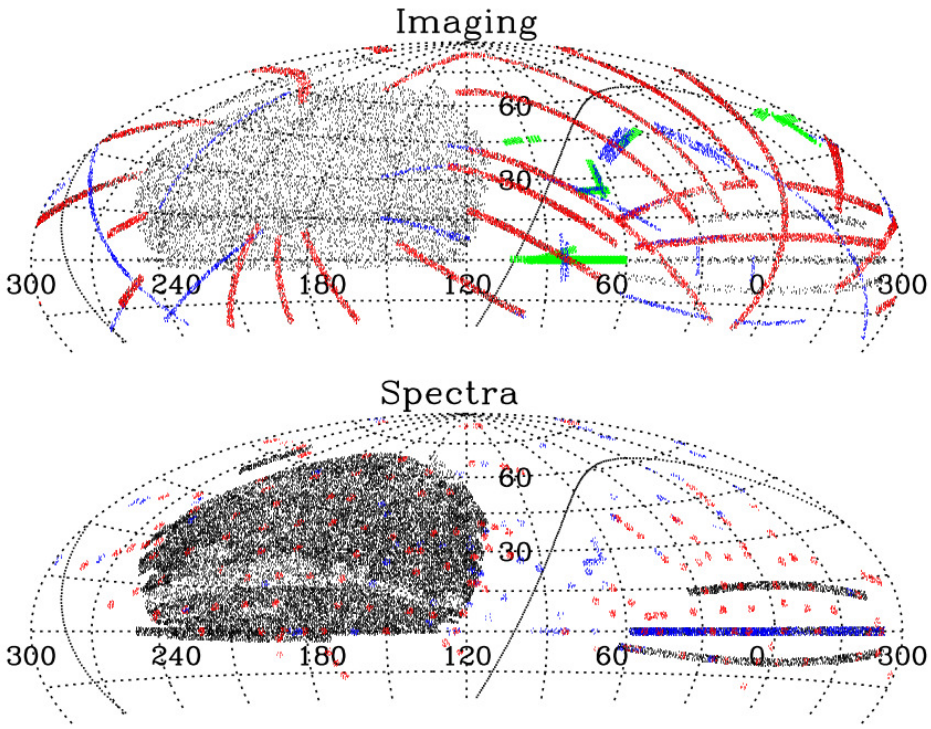


FIGURE 3.1. The distribution of the complete DR7 data for both imaging and spectroscopy, including sub-surveys plotted in different colors. The sky is shown in an Aitoff projection [31] in J2000 Equatorial Coordinates [32]. The sinuous line that goes through each panel is the galactic equator. The plots are cut at declination $\delta = -25°$. The SDSS did not cover areas below that. Figure taken from [30].

3.2 Galaxy Zoo

The Galaxy Zoo project is a crowd sourced science (also called citizen science [28]) project, where volunteers are asked to classify galaxies into morphological categories. For that, a multi-step decision tree via a web-based interface [33] was launched in 2007 with a data sample of nearly one million galaxies drawn from the SDSS database. The data used in this thesis are from the second edition of Galaxy Zoo. Here, more morphological categories have been implemented compared to the original Galaxy Zoo release. There are more than 16 million morphological classifications of roughly 300.000 of the largest and brightest galaxies taken from the SDSS. The following criteria were used to select the galaxies:

- magnitude $m_r < 17$
- angular size $r_{90} > 3''$
- redshift $0.0005 < z < 0.25$

The classification process is shown in figure 3.2 with the set of corresponding questions and answers in table 3.1.

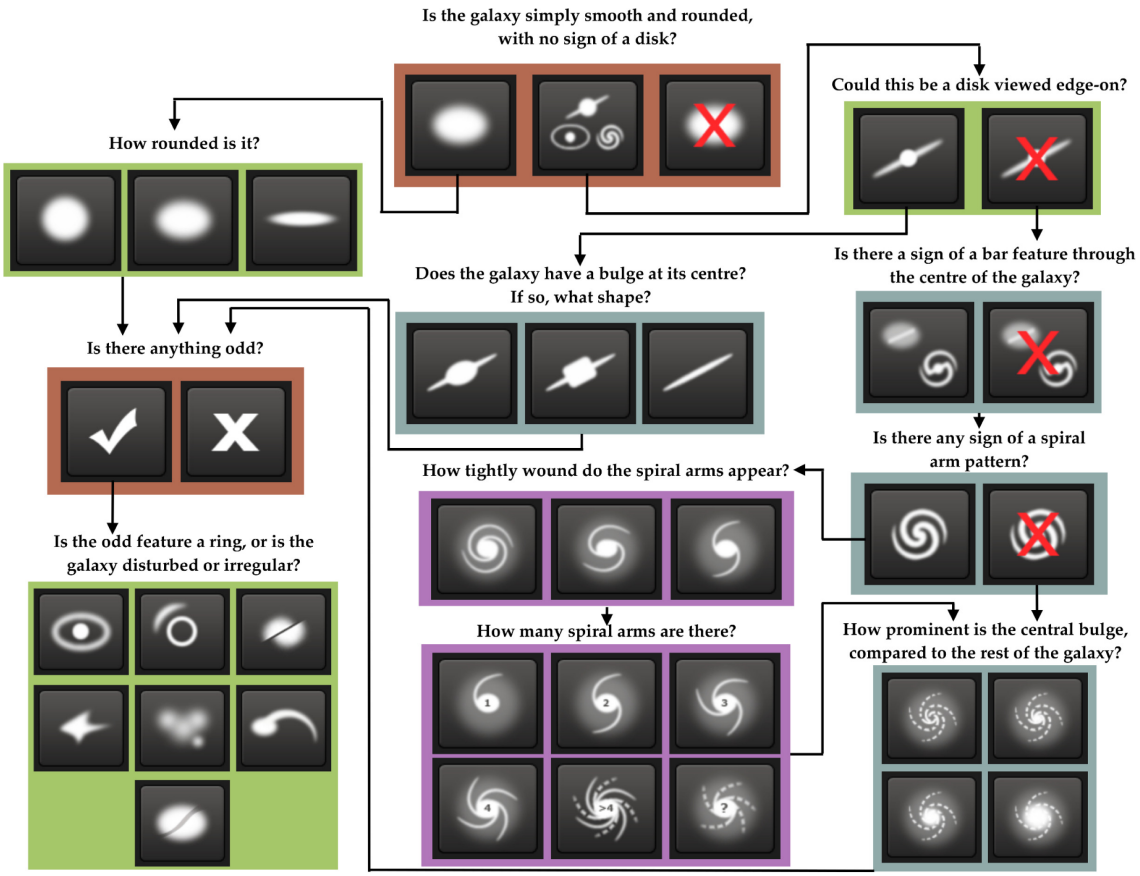


FIGURE 3.2. Galaxy Zoo decision tree with color-coded queries starting at the top center. Queries outlined in brown are asked of every galaxy. Queries outlined in green, blue, and purple are (respectively) one, two or three steps below branching points in the decision tree [34]

To estimate the accuracy of this procedure, the results have been compared with several morphology catalogues that contain SDSS images, especially *Nair & Abraham (NA10)* [35], *EFIGI* [36] and *Huertas-Company et. al. (HC11)* [37]. The first two are expert classification catalogues, while the third used an automatic classification method. The classification was found to be in good agreement with those catalogues, with a lower bound on accuracy of 82.1% for early type galaxies compared to the HC11, and up to 100% agreement for merger objects in NA10 and EFIGI. A slight deviation resulting from crowd sourced voting can be observed and data taken from Galaxy Zoo 2 should be adjusted for the biases for scientific use [34]. Since 2016, Galaxy Zoo strongly advises the use of a debiased table developed by Hart et. al. [38]. It has not been used in this thesis, since the image data used here is from 2014.

	Question	Response	Next
Q1	Is the galaxy simply smooth and rounded, with no sign of a disc?	Smooth	Q7
		Features or disc	Q2
		Star or artifact	End
Q2	Could this be a disc viewed edge-on?	Yes	Q3
		No	Q3
Q3	Is there a sign of a bar feature through the centre of the galaxy?	Yes	Q4
		No	Q4
Q4	Is there any sign of a spiral arm pattern?	Yes	Q10
		No	Q5
Q5	How prominent is the central bulge, compared with the rest of the galaxy?	No bulge	Q6
		Just noticeable	Q6
		Obvious	Q6
		Dominant	Q6
Q6	Is there anything odd?	Yes	Q8
		No	End
Q7	How rounded is it	Completely round	Q6
		In between	Q6
		Cigar-shaped	Q6
Q8	Is the odd feature a ring, or is the galaxy disturbed or irregular?	Ring	End
		Lens or arc	End
		Disturbed	End
		Irregular	End
		Other	End
		Merger	End
		Dust lane	End
Q9	Does the galaxy have a bulge at its centre? If so, what shape?	Rounded	Q6
		Boxy	Q6
		No bulge	Q6
Q10	How tightly wound do the spiral arms appear?	Tight	Q11
		Medium	Q11
		Loose	Q11
Q11	How many spiral arms are there?	1	Q5
		2	Q5
		3	Q5
		4	Q5
		More than four	Q5
		Can't tell	Q5

TABLE 3.1. The Galaxy Zoo decisions tree with all possible question that can be asked about an image with the corresponding responses. The column 'Next' indicates what question will be queried after the corresponding response. This means the questions will not occur in the indicated order and some questions will not be asked for certain images [34].

3.3 The Kaggle Competition

Kaggle is a web-based platform for competitions in statistics and machine learning methods [39]. The neural network models used in this thesis are based on the winning model of the Kaggle competition "Galaxy Zoo – The Galaxy Challenge" [40]. This contest was organized by Galaxy Zoo and sponsored by Winston Capital. It was held from December 20, 2013 to April 4, 2014. The goal was to develop an algorithm that was able to predict galaxy morphology from images from the Galaxy Zoo 2 project. A requirement on the algorithm is that it must be general and possibly applicable to future surveys [41]. To produce an

unbiased training sample for this purpose, the Galaxy Zoo 2 sample had to be constrained in several ways. While the images had to cover the full observed range of morphology, color, and size, galaxies with large uncertainties were eliminated. To discourage the algorithms from training on color instead of morphologies, galaxy images were chosen to be evenly distributed in color among the categories.

The training data set consisted of 61,578 JPEG color galaxy images, 424 by 424 pixels in size, and morphological data labels for each of the 37 categories. Those labels were a modified version of the weighted vote fractions in the Galaxy Zoo 2 catalogue, where the fractions were transformed into 'cumulative' probabilities that gave larger weights to more fundamental morphological categories earlier in the decision tree. Images were also stripped from their SDSS metadata to prevent contestants from using it to train the network. For the evaluation a set of 79,975 JPEG color images was provided.

The performance of a network was measured by computing the root-mean-square error (RMSE) $e(\hat{p_k}, p_k)$ between predictions p_k on the evaluation set and the corresponding crowd sourced probabilities $\hat{p_k}$ as shown in (3.1), where $k = (1, ..., 37)$ is the index number of the category and $N = 37$ the total number of categories:

$$e(\hat{p_k}, p_k) = \sqrt{\frac{1}{N} \sum_{k=0}^{N} (\hat{p_k} - p_k)^2} \tag{3.1}$$

CHAPTER 4

SETUP

The following chapter describes the key features of the winning solution of the Kaggle "Galaxy Zoo – The Galaxy Challenge" competition that the presented analysis is based on. A more detailed discussion can be found in [41].

4.1 Winning Model

As mentioned in Chapter 2.3, the best performing neural network used in the winning solution is based on networks developed by Krizhevsky et al. for ImageNet [23]. As such, it employs convolutional layers, pooling layers, dense layers and dropout. The architecture is described in Chapter 4.1.2.

The considered network has roughly 4×10^7 trainable parameters. This is in stark contrast with the 5×10^4 images that the training set contained, which leads to a high risk of overfitting. The network may tend to memorize the training examples, and thus, may not generalize well to new data. Several strategies to avoid overfitting are used. Besides the already mentioned dropout, the data is preprocessed as described in Chapter 4.1.1.

For the learning method the Nesterov momentum, introduced in Chapter 2.2.4, is used. The method introduced in the winning solution combines the prediction of an ensemble of models and is able to reproduce the high agreement answers from Galaxy Zoo participants with an accuracy above 99%. The RMSE achieved through this method is 0.07491. Only the best performing model from the ensemble is used as a starting point in this thesis.

4.1.1 Preprocessing

The images described in chapter 3.3 are first cropped from 424×424 pixels to 207×207. This can be done without training performance loss, because the object of interest is in the middle of the image with a large amount of sky background. Afterwards, the image is

downscaled to 69×69 pixels to reduce the dimensionality of the input and speeding up the training process.

Due to the limited size of the training set, performing data augmentation to artificially increase the number of training examples is instrumental. Each training example is randomly perturbed in five ways, which are shown in Fig. 4.1:

(a) rotation: random rotation with an angle sampled uniformly between 0° and 360°, to exploit rotational symmetry in the images;

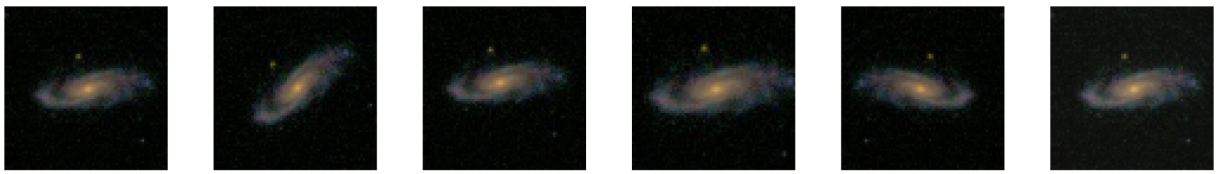
(b) translation: random shift sampled uniformly between -4 and 4 pixels (relative to the original image size of 424 by 424) in the x- and y-directions. The size of the shift is limited to ensure that the object of interest remains in the center of the image;

(c) scaling: random rescaling the image size with a scale factor sampled log-uniformly between 1.3^{-1} and 1.3;

(d) flipping: the image is flipped with a probability of 0.5;

(e) brightness adjustment: the pixel values are adjusted equally in the three color channels by a parameter α obtained from a Gaussian distribution in range $[0, 1]$, centered at $\hat{\alpha} = 0.5$. This procedure is described in more detail in [23].

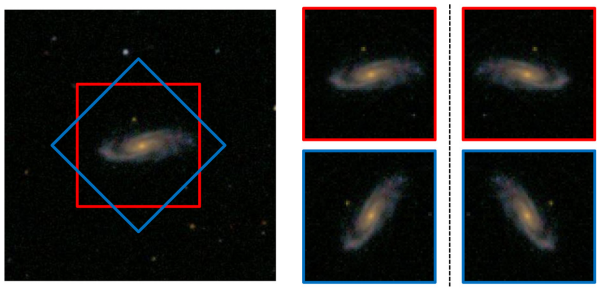
To maximize the effect of data augmentation, the images are randomly perturbed during training, so the models have a low probability to be presented with the exact same training example more than once.



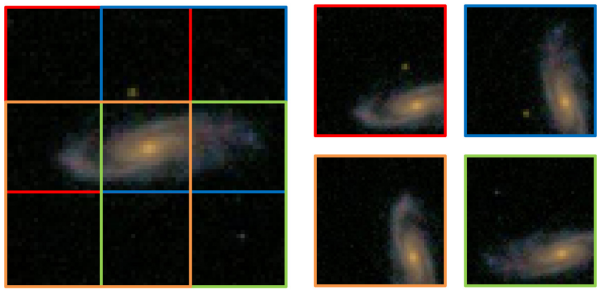
(a) none (b) rotation (c) translation (d) scaling (e) flipping (f) brightness

FIGURE 4.1. Data augmentation [41].

After augmentation, different viewpoints are extracted by rotating, flipping and cropping the input images. This way, 16 different viewpoints are created for each image: first, two square-shaped crops are extracted from an input image, one at 0° and one at 45°. These crops are also flipped horizontally to obtain four crops in total. Each of these crops is 69×69 pixels in size. Then, four overlapping corner patches of 45×45 pixels are extracted from each crop, and rotated so that the center of the galaxy is in the bottom right corner of each patch. These 16 rotated patches constitute the viewpoints, shown in Fig. 4.2. In practice, these are array indexing operations. This means the loss of image fidelity after augmentation and viewpoint extraction is minimal.



(a) 4 crops from an image



(b) 4 viewpoints from each crop

FIGURE 4.2. Crops and viewpoints [41].

4.1.2 Architecture

All viewpoints shown in Fig. 4.2 are presented to the network as $45 \times 45 \times 3$ arrays of RGB values, scaled to the interval $[0, 1]$, and processed by the same convolutional architecture. The resulting feature maps are then concatenated and processed by a stack of three dense layers to map them to the 37 answer probabilities. The architecture is visualized in Fig. 4.3. There are four convolutional layers, with filter sizes 6, 5, 3, and 3, respectively. A ReLU is used as the activation function for each convolutional layer. Max-pooling with kernel size 2×2 is applied to the first, second, and fourth convolutional layers. The concatenated feature maps from all 16 viewpoints are processed by a stack of three fully connected layers, consisting of two maxout layers with 2048 units with two linear filters each, and a linear layer that outputs 37 real numbers. The 37 values that the network produces for an input image are converted into a set of probabilities. The set of probabilities is rescaled by the probabilities of the answers that preceded them in the decision tree, shown in Fig. 3.2. Since the users give the probability of an answer conditional on its associated question being asked, but each Galaxy Zoo user is only asked a subset of the questions, some questions have a lower probability of being asked. Therefore the probabilities of the answers to these questions should be scaled down to obtain unconditional probabilities.

Table 4.1 lists the settings for the architecture. The initialization strategy for all layers is shown in the last two columns of Table 4.1. All biases were initialized to positive values to decrease the risk of units getting stuck in the ReLU saturation region. The same strategy was used for the dense layers.

	Type	No. of features	Filter size	Non-linearity	Initial biases	Initial weights
1	Convolutional	32	6×6	ReLU	0.1	$\mathcal{N}(0, 0.01)$
2	Convolutional	64	5×5	ReLU	0.1	$\mathcal{N}(0, 0.01)$
3	Convolutional	128	3×3	ReLU	0.1	$\mathcal{N}(0, 0.01)$
4	Convolutional	128	3×3	ReLU	0.1	$\mathcal{N}(0, 0.1)$
5	Dense	2048	–	maxout(2)	0.01	$\mathcal{N}(0, 0.001)$
6	Dense	2048	–	maxout(2)	0.01	$\mathcal{N}(0, 0.001)$
7	Dense	37	–	Constraints	0.1	$\mathcal{N}(0, 0.01)$

TABLE 4.1. Model Architecture. For the maxout activation square filters with sizes 2×2 are used. Table taken from [41].

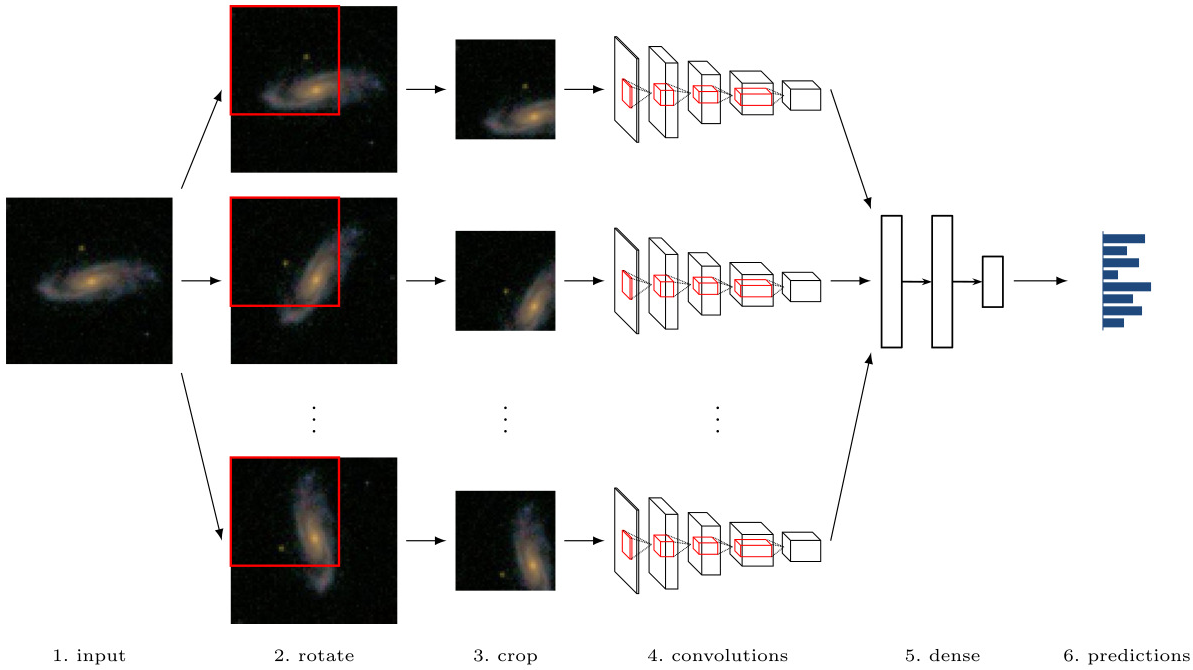


FIGURE 4.3. Model overview. The max pooling is not shown. Figure taken from [41].

4.2 Training for this Analysis

For the presented analysis, the model was implemented using PYTHON 2.7 [42] and the Keras API [43] with the THEANO V.0.9 backend [44].[1] An RMSE of 0.084 for training and 0.082 for validation as shown in Fig. 4.4 . The validation error is lower than the training error because of the implemented dropout [10].

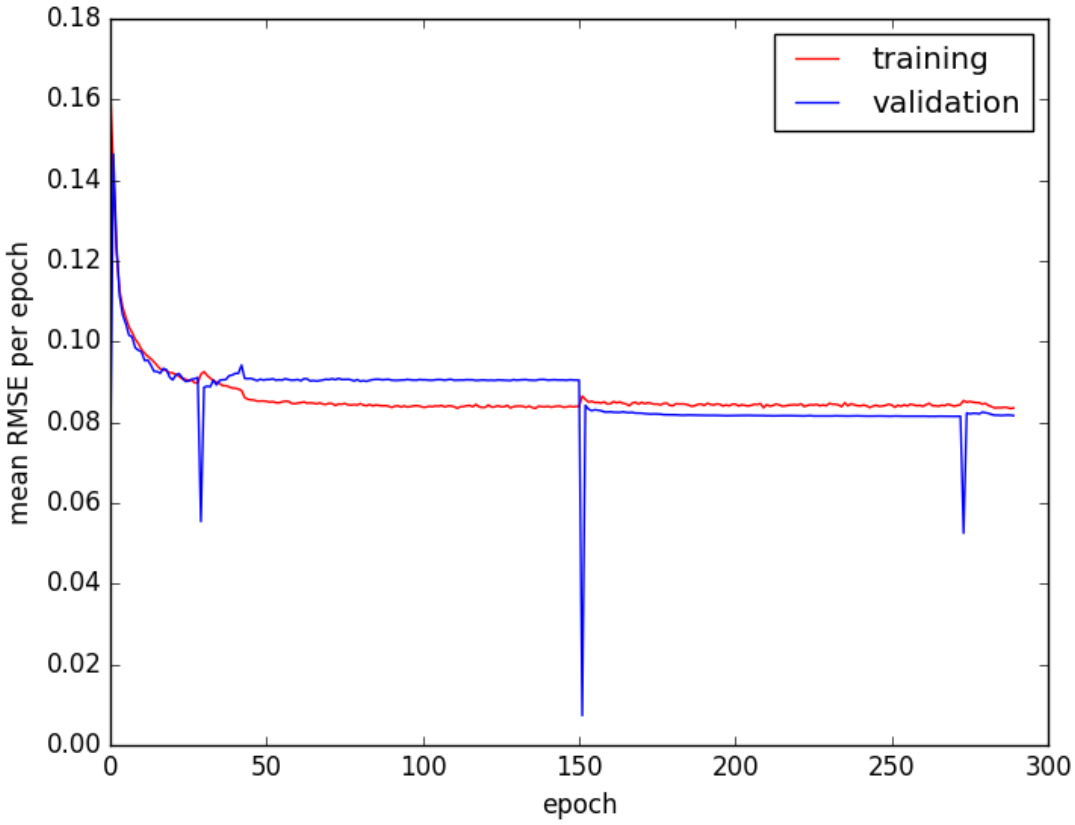


FIGURE 4.4. Training results of the reproduction of the winning model in Keras. The gaps in the validation error appear where the training was interrupted and continued later, because of incomplete batches that were validated for the new starting points.

[1]The original model was build using only PYTHON and the THEANO libraries. Importing the model into Keras was done by Christoph Garbers.

CHAPTER 5

HYPER PARAMETER OPTIMIZATION

The Kaggle competition "Galaxy Zoo – The Galaxy Challenge" took place in the year 2014. In a quickly evolving field, such as deep learning, new tools and methods are introduced frequently. One of the goals of this thesis is therefore to study if the overall performance of the presented network can be improved further, meaning not only the final predictive power and accuracy of the model, but also the training process. This chapter introduces the modifications that have been applied in time as a result of the optimization studies.

5.1 Initialization

Initialization of weights with Gaussian noise is very popular and was also used in the winning model, as shown in Table 4.1. But there are difficulties in training very deep networks from scratch with it [45], because of the activation (and/or) gradient magnitude in final layers. If each layer scales input by k and is not properly initialized, the final scale would be k^L, where L is a number of layers. Values of $k > 1$ lead to extremely large output values, $k < 1$ leads to a diminishing signal and gradient. Two initialization approaches have been studied:

Pre-Training

To exploit basic geometry that could be useful for morphological classification, the network is pre-trained on a generated geometry data training set, i.e. it learns the basic categories rectangle, circle and ellipse. Some examples of the different variations of the categories are shown in Fig. 5.1 . The main training is then initialized with the preconditioned weights.

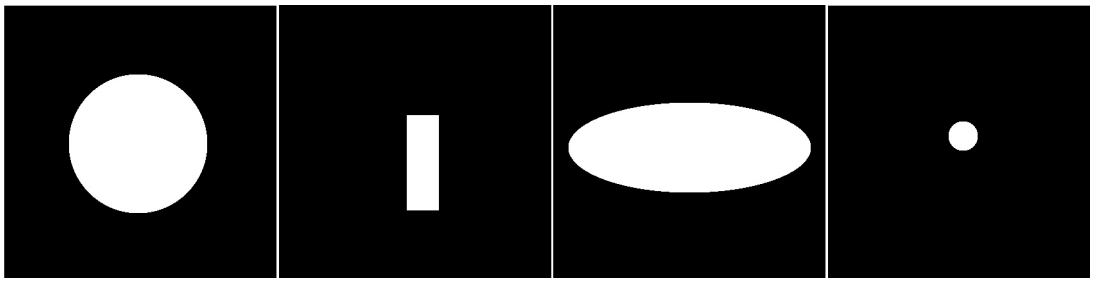


FIGURE 5.1. Examples of images of shapes used for pre-training.

LSUV

Layer-sequential unit-variance (LSUV) initialization introduced in [46] generalizes the orthonormal initialization for linear units presented in [47] to convolutional layers. The main idea is to have random orthogonal initial conditions on weights and subsequently applying an estimator for the output variance of each convolution and inner product layer in order to scale the weight to make variance equal to one.

Both methods significantly accelerate the learning process as shown in Fig. 5.2 , with the pre-training performing slightly better than LSUV. Considering the computational costs, however, the LSUV initialization is chosen as the preferred method of initialization for further analysis.

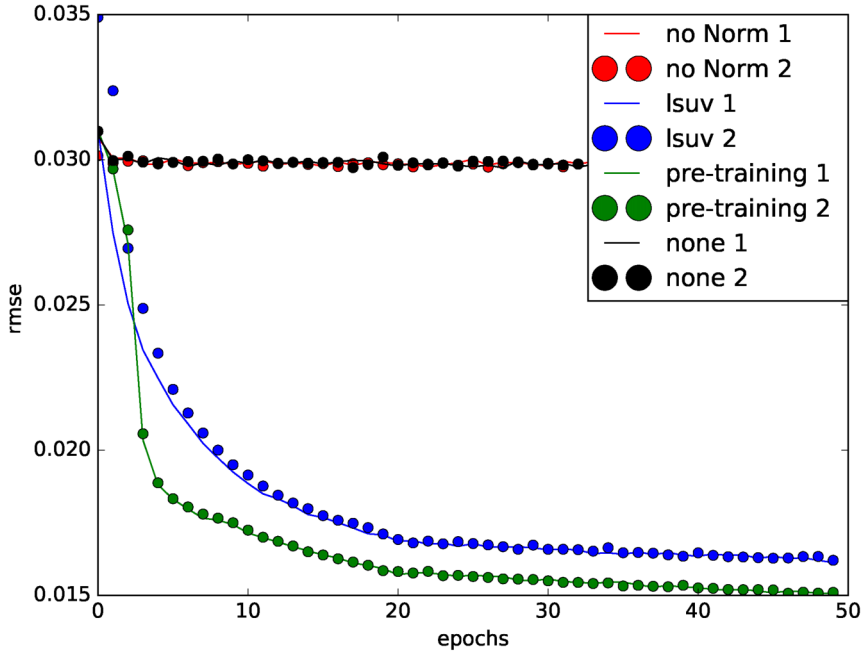


FIGURE 5.2. Loss functions for different initialization methods evaluated with the root mean square error (RMSE). Both introduced methods, pre-training and LSUV, are compared to Gaussian initialization, with no modification to the original method (none) and without normalization (no Norm), for two independent trainings.

5.2 Reduction of Parameters

To evaluate if the model's capacity is too high with respect to the task, the convolution filters are studied. The maximum activation for every filter in the fully trained four convolutional layers is computed. Figure 5.3 shows how an ideal image for the filters would look to maximize the filter's activation. Several filters seem to remain unactivated, which can be evidence for redundancy. The number of filters was reduced and little to no performance loss was observed, which further supports the hypothesis. However, additional statistical studies have to be made to ensure that the seemingly inactive filters only return random noise.

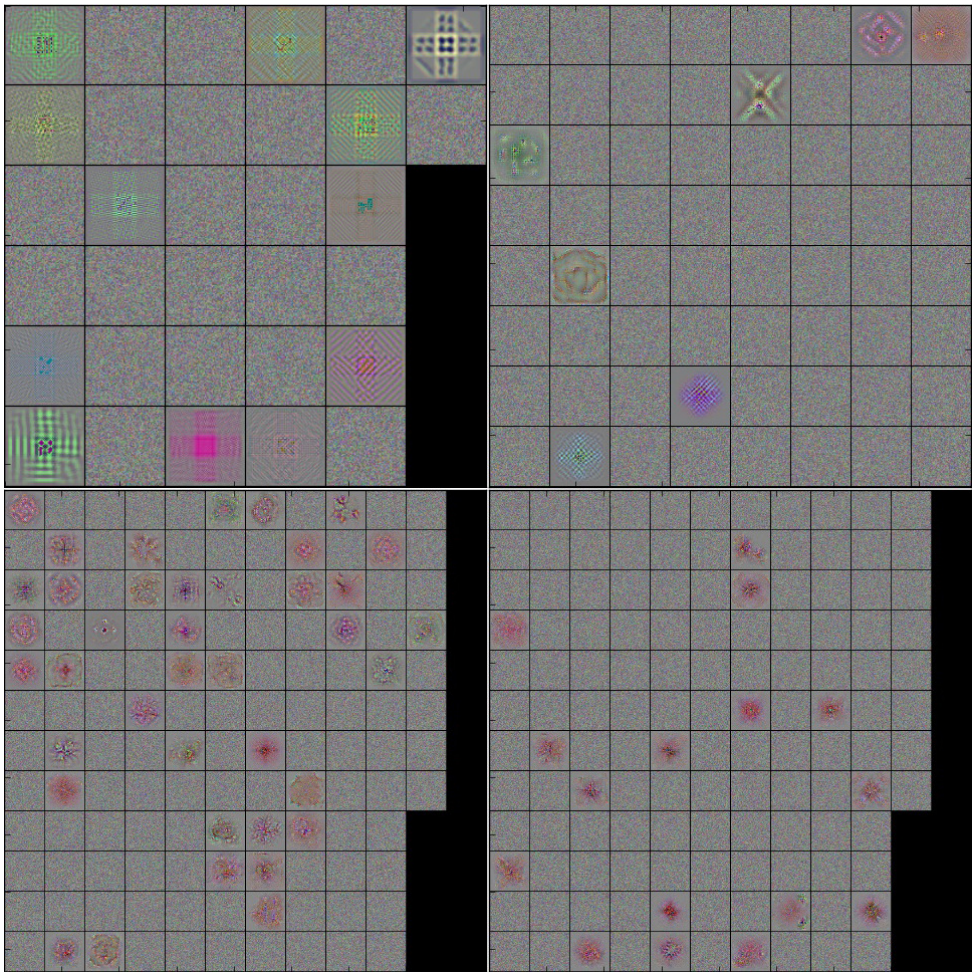


FIGURE 5.3. Maximum filter activation for the first (top left), second (top right), third (bottom left), and fourth (bottom right) convolutional layers.

5.3 Reduction of the Number of Categories

To simplify the work flow and to be able to apply clean cuts to the categories for later analyses, specifically the deconvolution analysis presented in Chapter 6, the number of categories is reduced from 37 overlapping to ten exclusive categories. Table 5.1 shows how the new categories combine the old answers taken from Table 3.1

Having exclusive categories allows for the use of the categorical cross-entropy, which has been introduced in Chapter 2.2.3, as the loss function. Another advantage is the higher number of images per category, which leads to faster convergence during training. Table 5.2 shows the training settings and results for independently trained weight sets `w0`, `w1`, and `w2` after the aforementioned modifications to the model. The further analysis makes use of these weight sets.

New Category	Answer Combinations
round	(Q1: A1) + (Q6: A1)
broad ellipse	(Q1: A1) + (Q6: A2)
small ellipse	(Q1: A1) + (Q6: A3)
edge without bulge	(Q1: A2) + (Q2: A1) + (Q9: A3)
edge with bulge	(Q1: A2) + (Q2: A1) + (Q9: A1, A2)
disc	(Q1: A2) + (Q2: A2) + (Q4: A2)
spiral with 1 arm	(Q1: A2) + (Q2: A2) + (Q4: A1) + (Q11: A1)
spiral with 2 arms	(Q1: A2) + (Q2: A2) + (Q4: A1) + (Q11: A2)
spiral other	(Q1: A2) + (Q2: A2) + (Q4: A1) + (Q11: A3, A4, A5, A6)
other	(Q1: A3)

TABLE 5.1. New exclusive categories (left column) and the corresponding answer combinations, which they include (right).

Weights	No. of Epochs	Training Time	RMSE	Categorical Cross-Entropy	Categorical Accuracy
`w0`	150	$9\,h$	0.043	0.829	0.674
`w1`	150	$9\,h$	0.043	0.825	0.677
`w2`	200	$12\,h$	0.043	0.829	0.674

TABLE 5.2. Different training settings and results for classification in ten categories with the modified network using LSUV.

CHAPTER 6

DECOMPOSITION OF THE NEURAL NETWORK

The main goal of this analysis is to gain a better understanding of the internal processes inside a convolutional neural network. The methods introduced in the last chapter help by finding a better set of parameters for the model, but provide little insight beyond fairly abstract optimization. This chapter introduces an approach called a deconvolution network to decompose and visualize the network's ability to learn by reconstructing an input image while enhancing the parts of the image that are pivotal for the classification by the network. The deconvolutional network was first introduced in [48] and further expanded in [49]. The idea is to allow the information to propagate backwards through the network and therefore reconstruct the information from deeper layers by unfolding the convolution operation and inverting any other kinds of layers in the network. To achieve this, the network is expanded in such a way that every layer, save for the convolutional layers, and their respective activation functions in the original network are mirrored with their (pseudo-)inverse counterparts.

The key element of this method is the deconvolution operation, also called transposed convolution, that can be derived from Equation 6.1 by applying the transposition operator y^T to the equation:

$$(y(t))^T = (\sum_{a=-\infty}^{\infty} x(a)w(t-a))^T = \sum_{a=-\infty}^{\infty} x^T(a)w^T(t-a)\,. \tag{6.1}$$

The resulting new operation retains all its former properties, with the exception of a transposed kernel matrix with inverse domain and image spaces. This can be used to build a layer where the data input propagates in a backwards direction, allowing the network to reconstruct the original properties of the input. The transposition is not a true inversion of the convolution operation, but this is intentional, since the goal is to find a way to have a reconstructed image amplified by the networks' vision. The deconvolutional layer uses the same set of weights that is used by the corresponding convolutional layer.

The other layers, i.e. the pooling and rectifier operations, need to be inversed for the reconstruction to work. While the rectifier operation is invertible without any complications

by simply using another rectifier operation, the max pooling operation is non-invertible. An approximate inverse can still be obtained by recording the locations of the maxima within each pooling region in a set of flag variables. In the deconvolutional network, the unpooling operation uses these flags to place the reconstructions from the layer above into the original activation location.

Finally, dropout is only used during training as a regularization method and is not active during actual classification of the network. Therefore, dropout can simply stay deactivated in a deconvolutional network.

Figure 6.1 shows how a typical composition of convolutional layer, pooling layer, and ReLU is mirrored to reconstruct the input image.

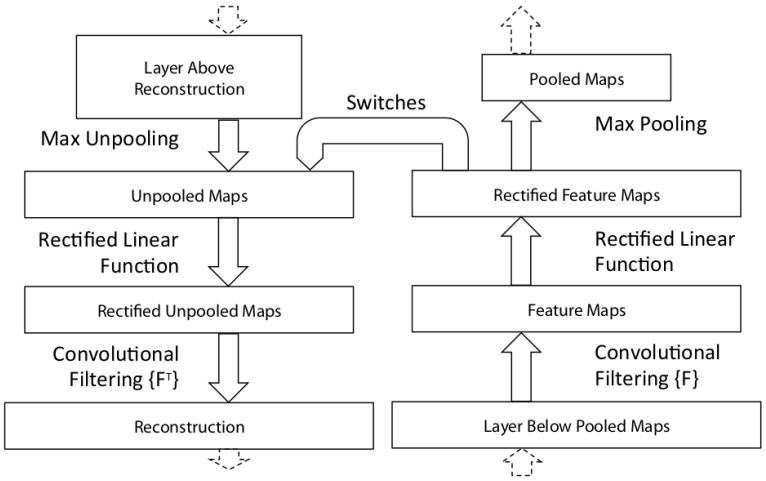


FIGURE 6.1. Architecture design for a convolutional layer, a ReLU, and a max pooling operation. On the right hand side, a part of the original convolutional network is shown. It takes input in the lowest layer. On the left hand side, the corresponding deconvolutional network is responsible for reconstruction of the image, taking input in its upper layer. Figure taken from [49].

6.1 Deconvolution Setup

The architecture of the deconvolutional network for this analysis follows the rules established in Fig. 6.1. The layer arrangement of the full architecture is shown in Table 6.1. The numbers of the deconvolutional and unpooling layers correspond to the hierarchy of the convolutional and pooling layers respectively. To reconstruct an image for a given convolutional layer, the deconvolutional network is only employed up to that layer. This means, the following four setups are implemented in this analysis:

Deconvolution Setup	Convolutional Network
Deconvolutional Layer (d4)	Convolutional Layer (c4)
Deconvolutional Layer (d3)	Convolutional Layer (c3)
Unpooling (u2)	Pooling Layer (p2)
Deconvolutional Layer (d2)	Convolutional Layer (c2)
Unpooling (u1)	Pooling Layer (p1)
Deconvolutional Layer (d1)	Convolutional Layer (c1)

TABLE 6.1. Deconvolution Setup.

1. D1 = d1 ∘ c1
2. D2 = d1 ∘ u1 ∘ d2 ∘ c2 ∘ p1 ∘ c1
3. D3 = d1 ∘ u1 ∘ d2 ∘ u2 ∘ d3 ∘ c3 ∘ p2 ∘ c2 ∘ p1 ∘ c1
4. D4 = d1 ∘ u1 ∘ d2 ∘ u2 ∘ d3 ∘ d4 ∘ c4 ∘ c3 ∘ p2 ∘ c2 ∘ p1 ∘ c1

The first network setup D1 is tested with a test image, to see if any errors occur during deconvolution, which would be harder to detect using the galaxy images. An example is shown in Fig. 6.2 . For randomly initialized weights, the convolution is expected to behave like a weak superposition of image processing techniques, e.g. blurring, sharpening, and edge detection [50] , but differently for each color channel. This is indeed seen in Fig. 6.2 .

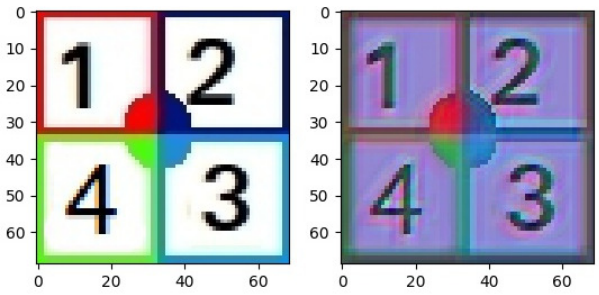


FIGURE 6.2. Test image that features high contrasts and distinct shapes and border regions (left), reconstruction after the first convolutional layer using D1 with untrained weights (right).

6.2 Visualization of the Deconvolution

With the method described above, the following galaxy images have been reconstructed. Figure 6.3 shows the results for galaxy image number 121410 after the deconvolutional layers D1, D2, D3, and D4.

The reconstructed images are dominated by the green color channel. This can be a random effect caused by the initialization that just happens to favor a parameter region

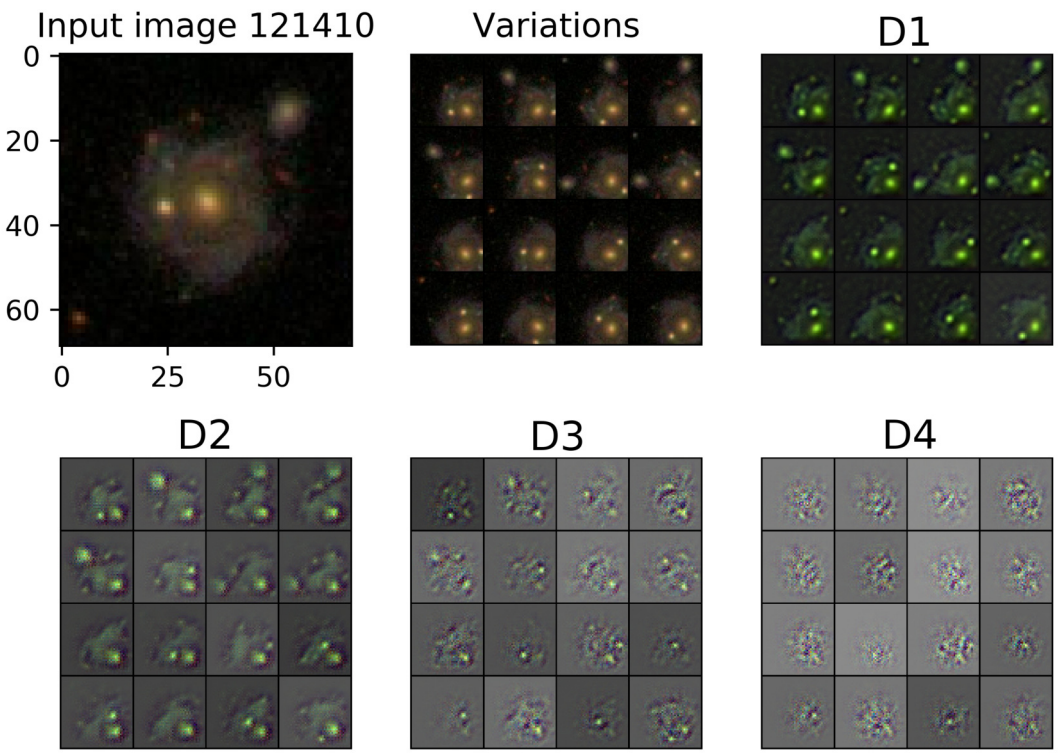


FIGURE 6.3. Deconvolution of image number 121410: the original image (top left), the pre-processed image set that is used as input for the main network (top middle), as described in Chapter 4.1.1, and the deconvolution for the pre-processed image set after the first (top right), second (bottom left), third (bottom middle), and fourth (bottom right) convolutional layers.

where green has a higher activation than the other channels. To study this effect, the network was trained independently multiple times with different starting conditions. The result is shown in Fig. 6.5 and discussed later.

Unlike the convolution with a random kernel, as shown in Fig. 6.2, the trained kernel enhances the galaxy features from the input image. In D1, an increased overall contrast compared to the input image with an increase in brightness for distinct features can be observed. This resembles an embossing operation with additional Gaussian filters from digital image processing [50]. The embossing is even more prominently visible in Fig. 6.4.

The pooling operation between D1 and D2 reduces the image parameter space. After reconstruction, this can be observed in the pixelation and noise increase in the image. The trend to isolate the galaxy features continues, flattening the background to a gray color with less features outside the main object. Deconvolution from deeper layers, using D3 and D4, suffers under information loss even more. Although this does not mean that the networks performs any worse using those layers, the pixelation and noise make it hard for the human eye to recognize any pattern that might be underlying the convolution

operation at these deep levels.

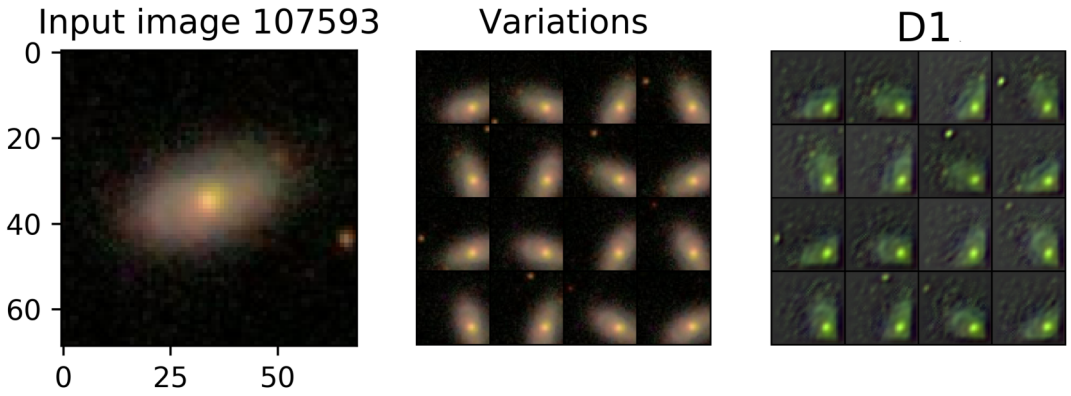


FIGURE 6.4. Deconvolution of image number 107593: As before, the original image (left), the pre-processed image set (middle), and the deconvolution D1 for the pre-processed image set after the first convolutional layer (right). The embossing after D1 is very visible.

Figure 6.5 shows the deconvolution for another galaxy using different weight sets, as introduced in Table 5.2 . The same behavior as in Figures 6.3 and 6.4 can be observered. Besides the expected feature enhancement, the network has a tendency to optimize in the same direction, where green is the dominant channel, despite having randomized initial weights. The number of different weight sets for this study is very low and not enough to verify the hypothesis, but training the network often enough to have statistical significance is a very expensive task in both computation power and time.

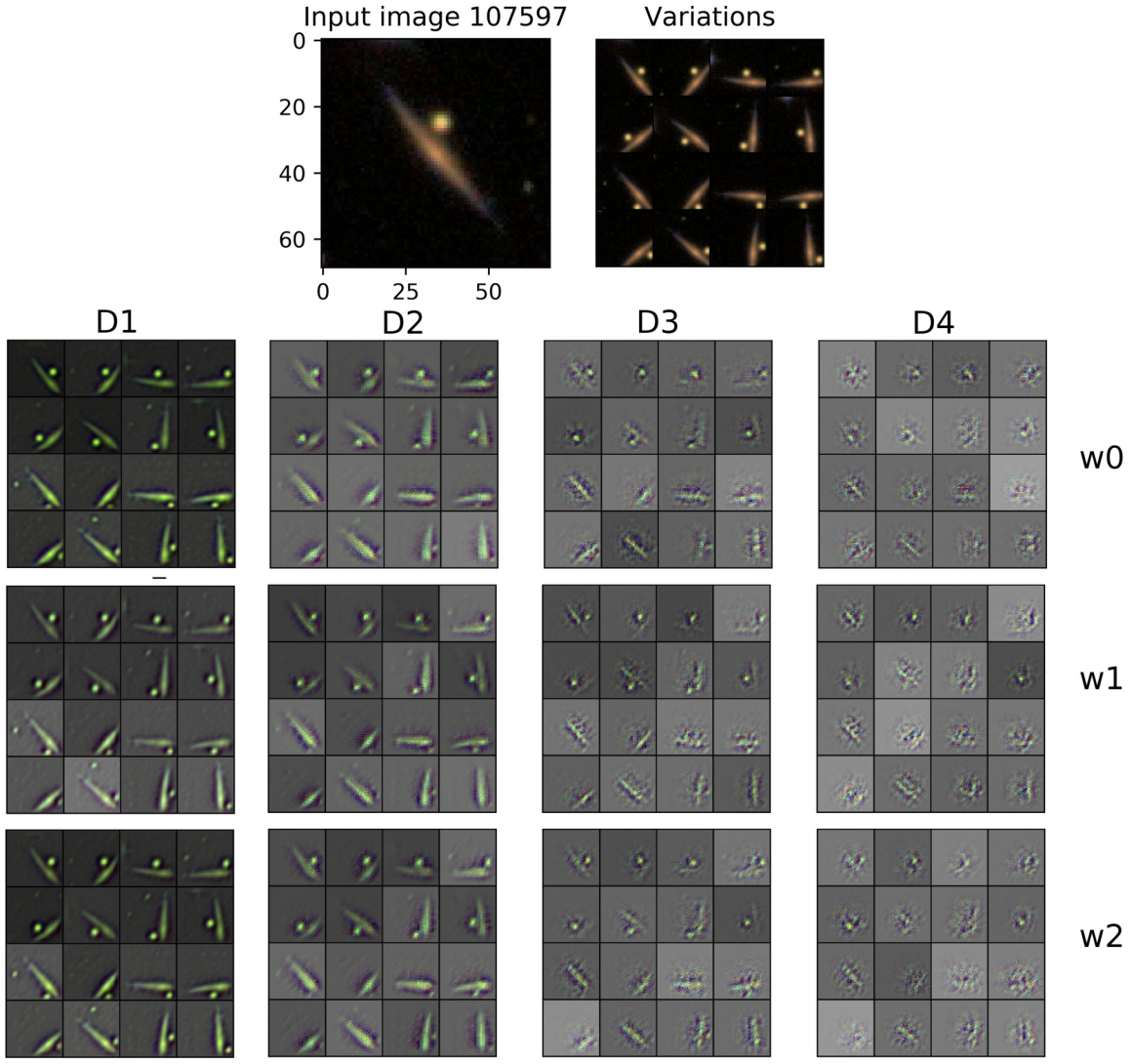


FIGURE 6.5. Deconvolution of input image 107597 with its pre-processing (first row) using the `w0` (second row), `w1` (third row), and `w2` (fourth row) weight sets as introduced in Table 5.2. The columns D1 to D4 correspond to the deconvolutional layers.

6.3 Conclusion

The results show that the network is able to highlight the morphological features of the galaxies through the convolutional layers. For different views of the galaxy different objects in the image are isolated and enhanced. In Fig. 6.5, the cigar shaped galaxy and the star in the foreground appear alone in several images in the D3 reconstruction. This is plausible, because due to the preprocessing, the views change in such a way that different parts of the image are in the center region, which has a higher impact in a convolution operation compared to the peripheral regions. This can be a hint that the viewpoint extraction preprocessing allows this network to outperform other models.

The deconvolution reconstructions have only been shown for a couple of images. To corroborate the above hypothesis, a proper statistical analysis has to be made. The deconvolutional network output needs to be investigated for each category with both high and low confidence images.

CHAPTER 7

OUTLOOK

The presented analysis shows promising insight, but several aspects need improvement to clarify ambiguities and additional studies are needed to test the proposed hypotheses. In the following chapter, a few suggestions for improvement and further work are presented.

7.1 Dense Layer Inversion

So far, only the convolutional layers with their activation function and the pooling layers have been implemented in the deconvolution setup. The final dense layers and their maxout activation also need to be reversed in order to have a complete setup. That way, the deconvolution may be reconstructed from a single category entry at the output layer of the network.

This may provide additional insight. There are, however, several problems that need to be solved to build the full setup. The concatenation of the final `conv4` layer output that is fed into the dense layers as well as the maxout activations are not easily invertible. A set of flag variables, similar to those used to unpool the max pooling operation, would need to be implemented to save the index groups positions in the maxout activation function.

7.2 Unsupervised Learning

The network has been trained to classify galaxy images into predetermined morphological categories. An unsupervised learning approach, where the network finds patterns in the data and tries to produce its own categories, might provide additional insight. Such a task could be done using an autoencoder [6]. Since the building blocks for the structure of the deconvolutional network are largely the same as for an autoencoder, this extension idea comes naturally. The network would need to be completely mirrored, i.e. the above mentioned dense layer inversion has to be implemented for this approach to work. The two

networks would then be treated as one big network that encodes information in the first half and decodes the information in the second half of the network and trained accordingly.

The information that can be obtained from such autoencoders is which features are the most dominant and provide the highest separation power in galaxy images. These can then be compared to the original network and the features it highlights.

7.3 Image Analysis

A proper image analysis with tools from digital image processing may be applied to the reconstructed images from the deconvolutional network to quantify several important aspects that have only been discussed qualitatively so far. Some interesting questions are:

1. How does the RGB balance in the image change after deconvolution?
2. Is this RGB balance shift approximately equal for different trained weight sets?
3. Is there still valuable information contained in the seemingly dark areas of the image?

7.4 Application in Physics

The presented analysis has been performed on image data, but many concepts and techniques introduced here can be applied to several types of data. Most importantly, any type of data can be seen as an abstract image projection. This means, similar techniques can be applied, for example, to event data from particle colliders in high energy physics.

CHAPTER 8

SUMMARY

A machine learning approach for galaxy image classification with a convolutional neural network with viewpoint extraction in preprocessing has been presented. For each galaxy image, 16 viewpoints are generated through rotation and flipping of 4 image cuts of the original image. The main layers of the network are four convolutional layers that train individually on each of the 16 viewpoints of the galaxies and two dense layers with maxout activation at the end of the network, that concatenate the output of the convolutional layers. The neural network model was able to achieve a RMSE of 0.084 for the training data set and 0.082 for the validation data set. The model has been studied with the deconvolutional network approach with the goal of understanding and visualizing internal processes in the architecture.

To save computational time, the hyper parameters of the network have been studied. Plotting the maximum activation for convolutional filters revealed seemingly inactive filters, which could then be cut from the network, resulting in a parameter space reduction. For the initialization, the LSUV algorithm and a pre-training approach were studied. Both methods were found to lead to faster convergence during main training, with the pre-training beating the LSUV algorithm by a small margin. The further analysis was still done using the LSUV initialization due to the additional time the pre-training requires before the main training process.

The classification task of the model has been downscaled from 37 overlapping categories to 10 exclusive categories to allow for possible future data analyses. The deconvolution approach was done for the four convolutional layers and their accompanying pooling layers. The images that have been reconstructed by the deconvolutional network show that the network is able to highlight the morphological features of the galaxies through the convolutional layers by isolating and enhancing different objects for different views of a galaxy. Further studies were proposed that need to be performed to learn more about the model's ability to classify with those isolated features.

BIBLIOGRAPHY

[1] D. Aberdeen, O. Pacovsky, and A. Slater, "The learning behind gmail priority inbox", in *in NIPS 2010 Workshop on Learning on Cores, Clusters and Clouds*, 2010.

[2] A. Ehsan Khandani, A. Kim, and A. Lo, "Consumer credit-risk models via machine-learning algorithms", in *Journal of Banking and Finance*, vol. 34, Nov. 2010, pp. 2767–2787.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[4] H. Bodlaender, J. Gilbert, H. Hafsteinsson, and T. Kloks, "Approximating treewidth, pathwidth, frontsize, and shortest elimination tree", *Journal of Algorithms*, vol. 18, no. 2, pp. 238–255, 1995, ISSN: 0196-6774. DOI: https://doi.org/10.1006/jagm.1995.1009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0196677485710097.

[5] S. Skansi, *Introduction to Deep Learning*. Springer, Cham, 2018.

[6] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes", *ArXiv e-prints*, Dec. 2013. arXiv: 1312.6114 [stat.ML].

[7] T. Mitchell, *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997.

[8] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms", *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, Oct. 1996, ISSN: 0899-7667. DOI: 10.1162/neco.1996.8.7.1341.

[9] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma", *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992, ISSN: 1.

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting", *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[11] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review*, pp. 65–386, 1958.

[12] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *ArXiv e-prints*, Dec. 2015. arXiv: 1512.03385 [cs.CV].

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[14] T. Kohonen, "Self-organized formation of topologically correct feature maps", *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, Jan. 1982, ISSN: 1432-0770. DOI: 10.1007/BF00337288. [Online]. Available: https://doi.org/10.1007/BF00337288.

[15] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?", in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153. DOI: 10.1109/ICCV.2009.5459469.

[16] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout Networks", *ArXiv e-prints*, Feb. 2013. arXiv: 1302.4389 [stat.ML].

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 533, Oct. 1986, ISSN: 10.1038/323533a0. DOI: http://dx.doi.org/10.1038/323533a0.

[18] A. Cauchy, "Méthode générale pour la résolution des systèmes d'équations simultanées", *Comptes Rendus Hebd. Séances Acad. Sci.*, vol. 25, pp. 536–538, 1847.

[19] B. Polyak, "Some methods of speeding up the convergence of iteration methods", in *USSR Computational Mathematics and Mathematical Physics*, vol. 4, Dec. 1964, pp. 1–17.

[20] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O(1/k2)", *Soviet Mathematics Doklady*, vol. 27, pp. 372–376, 1983.

[21] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning", in *Proceedings of the 30th International Conference on Machine Learning*, S. Dasgupta and D. McAllester, Eds., ser. Proceedings of Machine Learning Research, vol. 28, Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1139–1147. [Online]. Available: http://proceedings.mlr.press/v28/sutskever13.html.

[22] S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek, and D. Kostrzewa, *Beyond Databases, Architectures, and Structures*. Springer, Cham, 2014.

[23] 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. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[24] Y. Le Cun, L. Bottou, and Y. Bengio, "Reading checks with multilayer graph transformer networks", in *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, Apr. 1997, 151–154 vol.1. DOI: 10.1109/ICASSP.1997.599580.

[25] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning", *ArXiv e-prints*, Mar. 2016. arXiv: 1603.07285 [stat.ML].

[26] Y. T. Zhou and R. Chellappa, "Stereo matching using a neural network", in *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, Apr. 1988, 940–943 vol.2. DOI: 10.1109/ICASSP.1988.196745.

[27] D. G. York *et al.*, "The Sloan Digital Sky Survey: Technical Summary", *Astron. J.*, vol. 120, pp. 1579–1587, 2000. DOI: 10.1086/301513. arXiv: astro-ph/0006396 [astro-ph].

[28] M. J. Raddick, G. Bracey, P. L. Gay, C. J. Lintott, P. Murray, K. Schawinski, A. S. Szalay, and J. Vandenberg, "Galaxy Zoo: Exploring the Motivations of Citizen Science Volunteers", *Astronomy Education Review*, vol. 9, no. 1, p. 010 103, 2010. DOI: 10.3847/AER2009036. arXiv: 0909.2925 [astro-ph.IM].

[29] I. S. McLean, *Electronic Imaging in Astronomy*. Springer-Verlag Berlin Heidelberg, 2008.

[30] K. N. Abazajian *et al.*, "The Seventh Data Release of the Sloan Digital Sky Survey", *Astrophys. J. Suppl.*, vol. 182, pp. 543–558, 2009. DOI: 10.1088/0067-0049/182/2/543. arXiv: 0812.0649 [astro-ph].

[31] J. P. Snyder, *Map projections: a working manual*. University of Chicago Press, 1987.

[32] J. B. Kaler, *The ever-changing Sky: A Guide to the Celestial Sphere*. Cambridge University Press, 1996.

[33] Galaxy Zoo, *Categories*, https://www.galaxyzoo.com, [Online; accessed 25-April-2017], 2007.

[34] K. W. Willett *et al.*, "Galaxy Zoo 2: detailed morphological classifications for 304,122 galaxies from the Sloan Digital Sky Survey", *Mon. Not. Roy. Astron. Soc.*, vol. 435, p. 2835, 2013. DOI: 10.1093/mnras/stt1458. arXiv: 1308.3496 [astro-ph.CO].

[35] P. B. Nair and R. G. Abraham, "A Catalog of Detailed Visual Morphological Classifications for 14,034 Galaxies in the Sloan Digital Sky Survey", *The Astrophysical Journal Supplement*, vol. 186, pp. 427–456, Feb. 2010. DOI: 10.1088/0067-0049/186/2/427. arXiv: 1001.2401.

[36] A. Baillard, E. Bertin, V. de Lapparent, P. Fouqué, S. Arnouts, Y. Mellier, R. Pelló, J.-F. Leborgne, P. Prugniel, D. Makarov, L. Makarova, H. J. McCracken, A. Bijaoui, and L. Tasca, "The EFIGI catalogue of 4458 nearby galaxies with detailed morphology", *Astronomy and Astrophysics*, vol. 532, A74, A74, Aug. 2011. DOI: 10.1051/0004-6361/201016423. arXiv: 1103.5734.

[37] M. Huertas-Company, J. A. L. Aguerri, M. Bernardi, S. Mei, and J. Sánchez Almeida, "Revisiting the Hubble sequence in the SDSS DR7 spectroscopic sample: a publicly available Bayesian automated classification", *Astronomy and Astrophysics*, vol. 525, A157, A157, Jan. 2011. DOI: 10.1051/0004-6361/201015735. arXiv: 1010.3018.

[38] R. E. Hart, S. P. Bamford, K. W. Willett, K. L. Masters, C. Cardamone, C. J. Lintott, R. J. Mackay, R. C. Nichol, C. K. Rosslowe, B. D. Simmons, and R. J. Smethurst, "Galaxy Zoo: comparing the demographics of spiral arm number and a new method for correcting redshift bias", *Monthly Notices of the Royal Astronomical Society*, vol. 461, pp. 3663–3682, Oct. 2016. DOI: 10.1093/mnras/stw1588. arXiv: 1607.01019.

[39] Kaggle, *Kaggle: Your Home for Data Science*, https://www.kaggle.com, [Online; accessed 25-April-2017], 2017.

[40] Kaggle Galaxy Zoo, *Galaxy Zoo – The Galaxy Challenge*, https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge, [Online; accessed 26-April-2017], 2014.

[41] S. Dieleman, K. W. Willett, and J. Dambre, "Rotation-invariant convolutional neural networks for galaxy morphology prediction", *Monthly Notices of the Royal Astronomical Society*, vol. 450, pp. 1441–1459, Jun. 2015. DOI: 10.1093/mnras/stv632. arXiv: 1503.07077 [astro-ph.IM].

[42] G. Rossum, "Python reference manual", Amsterdam, The Netherlands, The Netherlands, Tech. Rep., 1995.

[43] F. Chollet *et al.*, *Keras*, https://keras.io, 2015.

[44] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: A cpu and gpu math expression compiler", in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, vol. Vol. 4, Jan. 2010.

[45] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", *ArXiv e-prints*, Sep. 2014. arXiv: 1409.1556 [cs.CV].

[46] D. Mishkin and J. Matas, "All you need is a good init", *ArXiv e-prints*, Nov. 2015. arXiv: 1511.06422 [cs.LG].

[47] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks", *ArXiv e-prints*, Dec. 2013. arXiv: 1312.6120.

[48] M. D. Zeiler, G. W. Taylor, and R. Fergus, "Adaptive deconvolutional networks for mid and high level feature learning", in *2011 International Conference on Computer Vision*, Nov. 2011, pp. 2018–2025. DOI: 10.1109/ICCV.2011.6126474.

[49] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901.

[50] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001, ISBN: 0130307963.

Dedication and Acknowledgments

Contrary to popular belief, I do have many people to thank. I will, however, keep things simple and informal. First and foremost, I would like to thank Peter, not only for the opportunity to work on my thesis in his group, but also for the countless talks we had beforehand and the guidance he has given me to find my place in the science landscape. My thanks go to Gregor, Hartmut, and Johannes, for helping with my work, supervising parts of it, or simply giving me food for thought. A very special thank you goes to Klitz and Sam for their invaluable contribution during the writing phase. I would very much like to thank my friends who endured me during the final part of my work - whatever their reason was. Amongst them especially Phillip, who put a lot of effort into distracting me from actual work, and Ronja, who always managed to keep my morale high even in very bad times. Many thanks go to my parents and my siblings, Wladi, Valeriy, Sascha, and Helga. It only took me eight years for my Master's degree. Top that, you nerds! Finally, thank you, Mimi, for anything you may already know and so much more than that.

This work is dedicated to Mika. I hope that one day you will have the curiosity to read this and forgive the time I spent writing this instead of being with you.

Author's declaration

Hiermit bestätige ich, dass die vorliegende Arbeit von mir selbständig verfasst wurde und ich keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt habe und die Arbeit von mir vorher nicht einem anderen Prüfungsverfahren eingereicht wurde. Die eingereichte schriftliche Fassung entspricht der auf dem elektronischen Speichermedium. Ich bin damit einverstanden, dass die Masterarbeit veröffentlicht wird.

Hamburg, Datum: ..... Unterschrift: .....