Master Thesis

Fog Prediction with Deep Neural Networks

Author(s):
Kipfer, Kevin

Publication Date:
2017

Permanent Link:
https://doi.org/10.3929/ethz-b-000250658

Rights / License:
In Copyright - Non-Commercial Use Permitted
Master’s Thesis Nr. 177
Systems Group, Department of Computer Science, ETH Zurich
in collaboration with
Meteo Swiss and Nvidia

Fog Prediction with Deep Neural Networks

by
Kevin Kipfer

Supervised by
Prof. Dr. Ce Zhang

29th September 2017
Abstract

The Zurich Airport location is prone to fog and limited visibility and accurate prediction of these conditions is of major economic and safety importance. However, due to the extremely localized nature of the phenomenon compared to the resolution of the operational weather model, accurate prediction of fog is challenging. In the current operational procedure, fog is therefore predicted using the experience of human forecasters, making the quality of the predictions dependent on the level of the forecasters experience.

The goal of this project is to develop an artificial intelligence system to predict fog at the Zurich airport 24hrs into the future. Standard data products of the operational weather model are used as inputs. The quality of the result is assessed by measuring the deviation of the network forecast from the actual visibility condition.

One of the most challenging factors we had to deal with is the small amount of available data. Furthermore, due to the nature of fog the provided dataset is high dimensional, heavily unbalanced and fundamentally different from traditional datasets such as ImageNet.

The impact of several commonly used methods, including SVMs, Neural Networks, etc. were studied. We focused especially on methods known to deal with small datasets, such as transfer learning. Additionally we made attempts to artificially increase the size of the dataset with Gan Models.
Acknowledgements

Many people have contributed to this project, and I like to thank in particular:

Prof. Ce Zhang for his continuous support and especially for all the advices, ideas and encouragements throughout the entire project.

Dr Oliver Fuhrer and MeteoSwiss for providing the dataset and valuable knowledge inputs.

Dr Peter Messmer for giving me the chance to work on this project and investing time in discussing the project in general.

Zhang Hantian for helping with all administrative problems and the kind support when problems with a server appeared.

The ETH Systems Group in general and the Ds3lab in particular for providing the necessary GPUs.
# Contents

1 Introduction .............................................. 1
   1.1 Motivation for Fog Prediction ......................... 1
   1.2 Deep Learning causes no Miracles .................... 2
   1.3 Goals ................................................ 3

2 Related Work ............................................. 5
   2.1 Automatic Fog Prediction ............................. 5
   2.2 Small and Unbalanced Datasets ....................... 6

3 Dataset .................................................. 7
   3.1 Background ........................................... 7
   3.2 About Creating Samples ............................... 8
   3.3 Characteristics of the Dataset ....................... 10
   3.4 Preprocessing and Data Augmentation ................. 11
   3.5 Training, Testing and Evaluation .................... 12

4 Baselines ............................................... 15
   4.1 K-Nearest-Neighbors ................................. 16
   4.2 Logistic Regression ................................... 16
   4.3 SVM .................................................. 17
   4.4 Conclusion .......................................... 18

5 Neural Networks ......................................... 19
   5.1 Convolutional Neural Networks ....................... 19
   5.2 Different Regularization Methods .................... 22
   5.3 Different Forms of Input ............................. 23
   5.4 Different Forms of Balancing ....................... 23
   5.5 Conclusion .......................................... 24

6 Transfer Learning ....................................... 25
   6.1 Resnet ............................................... 26
   6.2 VGG .................................................. 27
   6.3 Random Weights ...................................... 28
   6.4 Conclusion .......................................... 31
Introduction

Weather deals with very chaotic systems. Solving many of the problems that appear in this context is computationally expensive and often very challenging in general. A solution could be to learn how to make predictions instead of modeling the entire system.

As a consequence of the nature of the problem, a lot of the produced data is high dimensional, noisy and unbalanced. Furthermore many of the available datasets are small and have many other unfavorable characteristics. This thesis intends to investigate two questions: Is it possible to learn how to predict something as complicated as weather? Is it possible to do that with a non-optimal dataset?

1.1 Motivation for Fog Prediction

The interest in short-term weather warnings with higher localization accuracy has increased recently. The significant and hazardous meteorological events influence many parts of the society like traffic, agriculture, tourism, power generation, construction industry and many more.

Of particular interest is the fog phenomenon to airports all around the globe, since it has a significant impact on air transport operations. Limited visibility causes considerable restrictions of landing and takeoff. Thus it affects aircraft and passengers safety. In order to prevent delays, or even air accidents or incidents due to the adverse conditions, specific safety measures are required and in some cases prescribed by law. A reliable fog forecast is necessary.

Currently fog is predicted using the experience of human forecasters, This is cumber-
some for several reasons. Besides the fact that forecaster need to gather the necessary experience over a long period of time, it makes the quality of the predictions dependent on the level of the forecasters experience. Furthermore they are only used to doing so for a small amount of locations and can probably not quickly adapt to new areas.

The development an automated approaches might not only simplify to predict fog at the Zurich airport 24hrs into the future using standard data products of the operational weather model but it might also give us the chance to verify predictions. It might therefore lead to better and more accurate predictions overall. Furthermore it might also prove that Machine Learning can solve tasks that are fundamentally different from the ones we worked with so far. By being able to predict weather we show that other similarly complex and caotic systems might also be learnable. We would have shown that not everything needs to be modeled.

Besides the extremely localized nature of the phenomenon compared to the resolution of the operational weather model, the complex relationship behind the parameters and processes causing the formation of fog makes it extremely difficult to model and forecast it accurately. An Idea might be to learn how to make predictions, instead of modeling the entire system. Recent advances in Machine Learning hint that such an artificial intelligence system could be created.

1.2 Deep Learning causes no Miracles

Traditionally computers, while being very fast, have not been very smart. This changed in recent years. Nowadays learning algorithms are frequently employed for mobile applications, as well as for data analysis in back-end data centers. A category of algorithms, called Deep Learning recently rose in prominence and has emerged as state-of-the-art across a broad range of applications. Much of the progress in developing self-driving cars can be attributed to advances in deep learning. This Technology also lead to other successful applications such as programs that can play Go, translate languages or recognize a large amount of objects.

Deep-learning software attempts to mimic the activity of a human brain. Thus another widely used term for the same type of algorithms is Neural Networks. By extracting high-level, complex abstractions as data representations (features) through a hierarchical learning process, deep learning models often produce breathtaking results. In plain English, such a model will learn the features that are important by itself, instead of requiring the data scientist to manually select the pertinent features. [49]

The advances became possible in recent years because of the large computational power of GPUs and the large amount of available data. For example the standard dataset for object recognition (ImageNet)[18] consists of 14 Million images. A Dataset called Sports-1M[36] for video recognition consists of 1.1 Million videos. Such
a large amount of inputs is necessary in order to train the millions of parameters a neural network consists of. Without it learning useful features is quite challenging.

A quote from Andrew Ng illustrates a key problem of deep learning quite well: "I think AI is akin to building a rocket ship. You need a huge engine and a lot of fuel. If you have a large engine and a tiny amount of fuel, you won’t make it to orbit. If you have a tiny engine and a ton of fuel, you can’t even lift off. To build a rocket you need a huge engine and a lot of fuel. The analogy to deep learning is that the rocket engine is the deep learning models and the fuel is the huge amounts of data we can feed to these algorithms." [23]

Unfortunately there are many small datasets. Collecting data requires both time and money, which are both scarce resources. Scientific projects such as weather prediction is in that regard no exception. However without enough data many tasks become almost unsolvable. Deep Learning is no magic after all.

1.3 Goals

For our fog prediction project, we had to deal with only about 1.4 years of data. Therefore the main focus of this thesis is to explore whether different methods may achieve a reasonable predictive performance. In a first step, the student will familiarize himself with the MeteoSwiss dataset. We will train several commonly used methods such as knn, linear regression, svm and xgboost as baselines to detect correlations between a set of operational weather model products and visibility observations. The quality of the result will be assessed by measuring the deviation of the network forecast from the actual visibility condition. It is expected that the highly non-uniform distribution of the training data will have a significant impact on the performance of the network. This impact will therefore be investigated as well.

In a second step we train several simple neural networks from scratch and investigate the training behavior and prediction performance. We will explore whether transfer learning with pretrained models such as Resnet and VGG-16 can improve the results. Furthermore we will use other methods such as using random weights or creating an encoding with a VAE. The visualization of correlations between the different model quantities and the resulting predicted visibilities is left open for future research.
In recent years Neural Networks have shown big successes. They have become a major tool in fields such as computer vision and natural language understanding. There are a large number of new publications every year. Unfortunately most of them focus on very large datasets or try to solve very different problems. In the following overview of the related works, we try to cover the papers that propose solutions to the problem of small datasets or try to predict weather in general or fog in particular.

2.1 Automatic Fog Prediction

There are several projects that apply neural networks to weather prediction. For example neural networks have been used to predict quantitative rainfall for the Dallas-Ft. Worth area [29]. Some other project tried to process the output of a numerical weather prediction model with neural networks. The goal was to give more accurate and localized rainfall predictions in four separate regions in the mid-Atlantic United States [45]

Systems for automatic fog prediction at airports have been proposed for many areas around the globe. e.g the research group from the Italian Aerospace Research Center developed several fog classifiers based on Bayesnets [84]. In order to identify parameters with a strong influence for the formation of fog at the International Airport of Rio de Janeiro neural network were used together with rules extraction algorithms[19]. A historical dataset with 20 years of data was used to build a classification model which obtained an error of about 6 percent. A downside of this work is that they do not take in account the highly non-uniform distribution of fog occurrences.
A similar approach was used in [82] to classify collected historical data at the Melbourne airport. They use two classes, fog and no fog. Based on The Combinatorial Rule Model [4] they tried to extract interesting associative rules that can be stored in a knowledge base. Another project tried to predict fog at the Canberra International Airport based on 44 years of standard meteorological observations [21]. The trained neural network produced predictions up to 3 hours into the future with 0.937 cross-validated mean value.

k-Nearest-Neighbor was used in [35] for climate prediction. They had two datasets containing 40 thousand or 80 thousand records to predict around 17 different climatic attributes, including fog. They claimed to be able to predict fog with a 96 percent accuracy for the smaller dataset and 98 percent for the larger one. Tardif et.al [78] investigated fog events in New York City, with a dataset of 20 years of hourly data. They identified fog occurrences in 17 different locations. The events in each site were ranked according to an algorithm based on the fog formation process.

All of the presented systems predict fog only a few hours in advance. None of them attempt to make a prediction 24 hours in advance.

### 2.2 Small and Unbalanced Datasets

There are two main directions a dataset could be balanced: oversampling and undersampling. The standard method to balance heavily unbalanced dataset is called SMOTHE[13]. The main idea is to oversample the minority classes by adding to each of their samples their nearest neighbors. An alternative method is called ADASYN[30]. It uses a weighted distribution for different minority class examples according to their level of difficulty in learning, where more synthetic data is generated for minority class examples that are harder to learn compared to those minority examples that are easier to learn.

Transfer Learning is a common method to help dealing with small datasets by transferring knowledge, aka weights, to a new model. This technique has shown success in a study of Razavian et.al. [71]. They showed on different tasks the usability of transfer learning. Ciresan et.al applied transfer learning on various character recognition tasks [15]. DNNs trained on digits are perfectly capable of recognizing uppercase letters with minimal retraining. Rothe et.al showed the applicability of transfer learning to regression tasks such as age recognition [65].

A recent paper of Shrivastava et.al. suggests to simulate new synthetic data and refine them with a Generative Adversarial Network[72]. They found a significant improvement over using synthetic images, and achieved state-of-the-art results on the MPIIGaze dataset without having any real labeled data.
One of the main ingredients in building a successful artificial intelligence system is to either provide a lot of the necessary knowledge or giving it enough resources to learn what ever there is to learn. A common method to do just that is to throw as much data as possible at the system and having it analyze said data and making conclusions based on it. The amount of data is especially key to let deep learning methods yield staggering results. Without it, learning becomes quite a challenge if it is possible at all.

An interesting question to explore is how to deal with a not optimal dataset. For this investigation we chose a project that intends to predict fog at the Zurich airport 24 hours into the future using standard data products of the operational weather model. All experiments in the following chapters will try to solve this problem.

### 3.1 Background

In general Fog consists of visible cloud water droplets or ice crystals suspended in the air at or near the Earth’s surface[1]. It is formed as a result of air cooling, humidification, and by mixing contrasting air portions. One of the most common forms of fog occurs when its formation on land involves nightly radiative cooling under low wind conditions. It usually dissipates again a few hours after sunrise because of the warmth of the sun [64] [63] [67]. In general the temperature seems to be an important factor, since fog forms when the difference between air temperature and dew point is less than 2.5 C [74].

Although it is not an absolute rule, fog normally occurs at a relative humidity near 100 percent. [24] This occurs either from added moisture in the air, or falling am-
bient air temperature.[24] The necessary water vapor is often generated locally,[20] for example through daytime heating evaporating water from the surface of nearby water bodies such as a lake, moist ground or marshes [81]. Other sources are for example wind convergence into areas of upward motion;[58] cool or dry air moving over warmer water;[70] and lifting air over mountains.[60] As a result fog is heavily influenced by the topography, and changing wind conditions.

The phenomenon does not only have an extremely localized nature but it can also form suddenly and dissipate just as rapidly. Often fog is lifted up and forms a continuous layer of low stratus clouds, referred to as Hochnebel (high fog) in German. During periods of fog and low stratus over the Swiss Plateau direct radiation is blocked until afternoon or even for the entire day.[68]

In Switzerland a typical example for a location prone to fog and limited visibility is the Zurich Airport. In this Area the season with the highest probability to experience fog is fall. For example in October you have to expect fog almost every third day. However at higher elevations or south of the alps fog is far less common. In summer fog is in general quite rare.[77]

An interesting fact seems to be that the amount of days with fog seems to be decreasing. A study of the University of Bern in cooperation with MeteoSwiss shows that fog has become less frequent since 1971. In the years 1971-1975 there were 41 days with fog recorded. But in the years 2000-2004 there were only 25. That indicates that the amount of fog decreased by 5 days per decade. [83][68] Such a result may indicate that the climate in Switzerland is constantly changing. We are dealing with a very complex and dynamic system after all.

### 3.2 About Creating Samples

Fog is usually a very local phenomenon. Its creation is heavily influenced by the geography. Different places can look very differently and thus other weather conditions might be required to have the same effects. If you only care about the outcome at one particular location, though, the topology will remain constant. Therefore it might be possible to learn what kinds of conditions are necessary to produce fog. We may assume that local effects often correlate with the large scale conditions. Thus predictions could potentially be maid despite the resolution of the operational weather model.

The samples of the dataset are basically the predicted weather maps for different kinds of readings 24 hours into the future. One may call them an overview of the weather situation in all of Switzerland at the target time of the fog prediction. This samples are automatically generated by an already existing weather model of MeteoSwiss. The quality of the result will be assessed by measuring the deviation of the network forecast from the actual real world visibility condition. Simply put we label
each time step represented by a model sample with the actually measured visibility value.

For each time step there are seven predicted quantities: The cloud coverage, the humidity, the wind direction, the wind speed, the pressure above sea level, the total precision and the solar radiation. For each of those quantities we have a 256x256 pixel image. Each pixel represents one kilometer in reality. Figure 3.2 shows some examples. We combine different time steps together, hoping to get valuable information about the dynamics of the system. Whether the humidity is increasing or decreasing is information that might be quite important. Furthermore it might compensate for missing information outside the scope of the images.

Originally it was planned to combine images taken in a one hour interval. This approach has several downsides, though. The most important one is that weather changes quite slowly. The gain of information is probably quite limited and might not justify the massive increase of dimensionality. Instead there are other options. In fact we are quite free in how we arrange the dataset.

Another option would be to combine several time stamps with a much larger interval. In practice we chose a three hour interval instead. The question is now how many time steps to include. We will make experiments with one, three [+3hrs, -3hrs] and six [+3hrs, -12hrs] time steps in chapter six. We need to find the optimal trade off between a larger dimensionality and some additional information.

**Distribution of Fog-Days**

![Figure 3.1](image.png)

Figure 3.1: This diagram was taken from Scherrer et.al. [68]. It shows the average number of days with fog per month.
3.3 Characteristics of the Dataset

The most obvious characteristic of the fog dataset is its small size. It contains data coming from only 1.4 years, which gives us 15924 samples in total. 11944 samples in the training set and 3980 in the testing set. Compared to other deep learning datasets that consists of hundred thousands of samples this is an incredibly small amount of data. Moreover since samples were generated in a one hour interval, they are heavily correlated. As we can see in figure 3.2, samples within a range of several hours from each other often look very similar. In some extreme cases even data augmentation causes them to be more different. As a result we effectively need to consider the dataset to be even smaller.

Another characteristic of the dataset is its unbalancedness. The training set consists of 8933 clear sky samples, 1452 samples with a visibility within 5-10km, 1024 in the range 600m-5km and 515 below 600m. Therefore 75 percent of all the samples belong to the same class. Such a distribution usually proves to be challenging for many of the standard methods.

Images are by definition high dimensional. In our case we have to use seven differ-
ent weather references and up to six different time steps. Each channel consists of 256x256 pixels. Calculated together a sample consists of 2.7 million features. This is a lot larger than any input of state-of-the-art object recognition models deal with. Because of the curse of dimensionality we require even more data to avoid over-fitting given the high dimensionality of the feature space. This is especially the case since the problem is probably non-trivial. Because of this we will probably not be able to make overly fine grained predictions. Instead we will focus on predicting at most 4 classes.

An innate characteristic of clouds is to appear in ever new forms of shapes. There overall appearance is probably complete irrelevant for the prediction of fog. As a result the samples are full of noise. The relevant information might very well be hidden somewhere in a large heap of irrelevant information. To make it worse, the factors that influence weather are distributed over a very large scale. Therefore a lot relevant information could be hidden somewhere outside of our scope. In Fact since we do not deal with real world data but with data generated by a model there is a certain probability that a lot of the relevant information is not contained in any of the images. Therefore we might sooner or later hit some upper boundary, that we can not overcome because of a lack of information in the input samples.

A big problem that we will face is that the fog dataset is completely different from other datasets like ImageNet. There are no shapes at all. view points and poses are irrelevant while the absolute location in an image matters. We could easily extend this list. Since many properties and assumptions of ImageNet do not hold in the fog dataset we need to expect out models to learn very different patterns. The consequence is ultimately that we might have a hard time reusing pretrained knowledge.

3.4 Preprocessing and Data Augmentation

Standardization of a dataset is a common requirement for many machine learning estimators. It usually consists of centering and scaling. In Object Recognition they often use a per image standardization. This is especially important for small datasets. Unfortunately we can not apply it in our case. Each value has a separate meaning. To standardize per image would destroy valuable information. We apply a simple scaling that ensures that all value are in [0,1].

Data Augmentation[39][12] faces similar problems. Standard augmentations such random brightness, etc. would destroy the meaning of a value. Furthermore it is important to keep in mind that the location is important. Mindlessly moving, cropping or rotating images is not possible. Although movement is an innate characteristic of clouds and thus used in the weather models of MeteoSwiss, it might be possible that a neural network will be trained to focus on landmarks in order to center images instead of looking for something that causes fog. Instead we propose some other things. Adding noise seems to work. According to MeteoSwiss up to ten percent should be
An easy way to reduce the dimensionality, speed up the calculations and getting rid of a lot of unwanted noise is to down sample the raw images. Although fog is a very local phenomenon, we don’t assume that there are fundamental changes within a few kilometers. That’s especially the case since the airport is about 64 square kilometers tall, while at the same time we only got one label for all of it.

Something else we do is to take the average values of the center 5 times 5 pixels of each channel. We add those as additional features to almost every model that we use. For the Neural Networks we simply feed those features to the classifiers at the top. The rational is simply to add some prior information to lessen the need for data a bit. The place where we will find the most information is the place where fog will occur, namely the Airport, which is located exactly in the center of an image. Another feature we add is the time.

### 3.5 Training, Testing and Evaluation

We decided to take 20 percent of the data as the testing set and use the rest for training. Since we have so little data we can not spare any for a proper validation set. Instead while building a model and doing parameter tuning we use the first fog season as a testing set and the rest as the training set. For the validation we use the second fog season as a testing set instead and the rest as the training set. This will hopefully limit over-fitting to the test set.

Since weather experts classify their predictions, we shall do the same. Whenever the visibility falls into a certain range they will give it the corresponding label. Unfortunately the classes used by experts are too fine, such that there are too little samples in certain classes to make reasonable predictions. We will thus choose the following classes instead based on the amount of visibility there is: below 600 meter; between 600 and 5000 meter; between 5000 and 10000 meter; everything above 10000 meter. We will use these scheme to evaluate and compare the results of all models in the following chapters.

While we were experimenting we recognized that we were able to achieve an accuracy of above 90 percent with just sampling some random points and feeding those into an svm or xgboost. What happened? It seems that by randomly assigning samples to the different sets the training and the test set became correlated. The reason is that since the weather changes so slowly, there were samples in the test set that looked a lot like some others in the training set despite having a slightly different time stamp. It is therefore important to sort the samples by season before assigning them to one of the sets.

For unbalanced datasets it would be misleading to just focus on the standard accuracy. Let’s give an example: If we classify all the labels in our dataset with the classes
we just mentioned and compare that with some random prediction we can expect to get an accuracy of about one forth. If on the other hand we came up with some model that always predicts the class with the most samples, the accuracy would increase to about 43 percent. Keep in mind that the testset contains far less non-fog days than the training set since it contains just a fog season. If the samples were not sorted by season and the testset contained samples from all over the year we would have to expect an accuracy of about 80 percent. We propose to use other metrics, that are far more resistant to unbalancedness, such as the macro F1 score and the mean per class accuracy. They basically work as follows: calculate a score for each class and average the results.

![Confusion Matrix](image)

Figure 3.3: The confusion matrices for both a random prediction and a predictor that always yields the same result. The measured macro F1 for random is 0.23 and accuracy is 0.249. The measured macro F1 for one-class-prediction is 0.15 and accuracy is 0.43.
Figure 3.4: both plots shows classifications of the visibilities into 8 and 4 classes. The main difference between the upper and the lower plot is that all the classes 4-8 were aggregated into one class. The classes are 0) <600, 1) 600-5000, 2) 5km-10km, 3) 10km-15km, 4) 15-20km, 5) 20km, 6) 20km-40km, 7) 40km-50km
A central research topic is object recognition. For many years now different kinds of models tried to find and determine objects in digital images. Before the advent of neural networks, models such as knn and SVM were used for this task. They often tried to find good features and make predictions based on them. This methods are still used a lot for the many really small datasets. For such use cases a simple model can often yield better results than a far more powerful one, such as deep neural networks, that have a strong tendency to over-fit.

In this chapter we will discuss several commonly used methods and apply them to our fog dataset. The main goal of this chapter is to get baselines and to learn some characteristics of the dataset.

Results of the Baselines

<table>
<thead>
<tr>
<th>Methods</th>
<th>macro F1</th>
<th>accuracy</th>
<th>kappa</th>
</tr>
</thead>
<tbody>
<tr>
<td>knn</td>
<td>0.329</td>
<td>0.435</td>
<td>0.15</td>
</tr>
<tr>
<td>logistic regression</td>
<td>0.27</td>
<td>0.44</td>
<td>0.23</td>
</tr>
<tr>
<td>log. Reg. (just center)</td>
<td>0.21</td>
<td>0.33</td>
<td>0.12</td>
</tr>
<tr>
<td>SVM (features)</td>
<td>0.3</td>
<td>0.4</td>
<td>0.15</td>
</tr>
<tr>
<td>SVM(sampling)</td>
<td>0.38</td>
<td>0.45</td>
<td>0.21</td>
</tr>
</tbody>
</table>

Figure 4.1: Table with the best results of all the methods in this chapter.
4.1 K-Nearest-Neighbors

As our first baseline, we use the k-nearest-neighbor classifier.[7] This method is very rarely used in practice in parts because of its undesirable performance at test time. Nonetheless it will give us a baseline that is easy to implement and to understand and that will allow us to get an idea about the basics of the problem at hand.

In kNN an object is classified by a majority vote of its k nearest neighbors in the feature space. The object is thus assigned to the class that is most common in its neighborhood. We feed the entire images into the classifier and compare thus samples pixel by pixel. We are well aware that images are high dimensional and that in such spaces distance functions can behave counter-intuitive. Our main objective is to find out how much importance the location of a pixel has and how similar the similar looking samples really are based on their class. The hypothesis is that it should behave better than random since similar weather situations should lead to a similar result. On the other hand we can not expect too much since the immense amount of noise will probably easily dominate.

The Experiment yielded for 5 neighbors a macro F1 score of 32 percent and an accuracy of 43 percent. While the accuracy is the same as the one class predictor in section 3.4. We assume that there is too much noise in a sample to make accurate predictions in a high dimensional feature-space. We thus just get the most common sample as a prediction. It seems necessary to reduce the dimensionality in order to avoid the effects of the curse of dimensionality.

4.2 Logistic Regression

For the second Baseline we choose to use logistic regression. This method is usually one of those that a data scientist starts to experiment with. This is especially the case if there is only little data. In order to fight over fitting we add a l1 regularizer (lasso)[80]. The parameters were found with grid search. One way to reduce the dimensionality is to shrink the image. The most dominant patterns prevail while a lot of the noise disappears. As the name suggests we do regression and classify the predictions and the label before evaluation in order to have comparable results.

As can be seen in table 4.1 the results are not convincing. In fact the prediction is pretty much comparable to random guessing. The model heavily under-fits and just returns the most common label. There are several reasons for this failure. First of all there are still too many features that the model has to deal with despite the regularization, while on the other hand important information such as the situation in the center are missing. Secondly the unbalancedness of the dataset hurts the performance. It will be important to find means to balance the dataset.

An other option to do feature reduction is to just take the mean of the center pixels
where the airport is located, hoping that the weather model provides all necessary information correctly. This second option fails too since it puts too much faith into the weather model.

### 4.3 SVM

For the third baseline we use a Support Vector Machine[17]. This algorithm was successfully applied in many image classification projects. [22][10][11] The main idea is to constructs a hyperplane in a high-dimensional feature space, that separates the different classes. This ability to handle high-dimensional input is of interest to us. We may thus expect a far better result than the other two baselines so far. We use once again the down sized samples from 4.2 as input.

For imbalanced data sets we typically use a method mentioned by Edgar Osuna et.al. [56]. For the so called class-weighted SVM we change the misclassification penalty per class. Essentially this is equivalent to oversampling the minority classes. In order to get results we also additionally under-sampled the majority class. In other words we only kept one forth of all the samples with a visibility of 20 kilometers. All the parameters were found with grid search. As a side note the best result that we found used a linear kernel instead of a rbf kernel. Even though rbf is supposed to be more powerful [37] Chih-Wei et.al. [33] state that If the number of features is large, one may not need to map data to a higher dimensional space. That is, the nonlinear mapping does not improve the performance. Using the linear kernel is good enough. This definitely is the case for our dataset.

If we run this experiment we find a Macro F1 of 38 percent and an accuracy of 45 percent. This is better than the baselines we had so far. With better features a better result should be possible, though. The confusion matrix is given in table 4.2.

<table>
<thead>
<tr>
<th>real</th>
<th>predicted</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>146</td>
<td>122</td>
<td>47</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>243</td>
<td>325</td>
<td>307</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>208</td>
<td>158</td>
<td>304</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td>214</td>
<td>273</td>
<td>1159</td>
</tr>
</tbody>
</table>

Figure 4.2: The confusion matrices for SVM experiment. We measured a mean per class accuracy of 0.38 and an accuracy of 0.45. We investest some effort to balance and weight the input.
4.4 Conclusion

In this chapter we tried to get some baselines for a high dimensional and small dataset. A well known fact became obvious: You need to have good features if you want to make accurate predictions. However doing feature engineering is tricky, given the fact that it is currently unknown what exact circumstances are required to create fog in Zurich Kloten.

A simple procedure such as taking an estimate for each area plus the values in the very center, where the airport is located already gives us features that we can work with. In future chapters we need to include more fine grained information without increasing the dimensionality too much. Furthermore we learned that it is important to keep the information about where the different values are coming from, since translation independent features did not perform well.

A second conclusion we can make is that balancing a dataset is of utter most importance. Many models do not perform well if facing highly unbalanced inputs. A reasonable way to do such is to discard two thirds of all the samples with a visibility of 20km. The strong correlation of the samples allows us to do such. Therefore we still seem to have enough examples to learn properly.

On a side note we need to mention that we also tested other models such as xgboost and other kinds of features such as hog, lbp etc. Since all of those results were worse than the SVM baseline and since we no longer had the time to rerun the experiments on the validation set we chose to not report them.
The last couple of years were an exciting time for the deep learning community. Every now and then new and sometimes surprising results are published that push artificial intelligence to new heights. Unfortunately these models are not just tools that always work by them self. In this chapter we intend to explore the possibilities of neural networks in regards to the fog dataset. We study different forms of architectures and regularizations and try to find out how we can augment our samples. All the networks are trained from scratch and are implemented in Tensorflow.

We only present the results measured while doing validation. Furthermore all models in this section show higher accuracies within the first 1000 steps. Later it gets traded against a higher mean per class accuracy. We perform early stopping when the models obviously is saturated.

### 5.1 Convolutional Neural Networks

Convolutional Neural Networks are a category of Neural Networks that have proven to be very effective in areas such as image recognition and have become an important tool for most machine learning practitioners today. The main components are convolutions that learn to detect more generic features such as edges and blobs at the bottom layers and more sophisticated patterns like shapes higher up in the network. In between the convolutions there is often some max pooling layer that downsizes the images. At the top there is a classifier, for example a fully connected layer, which makes the predictions. The entire network is trained by minimizing some loss function. For classification this loss is usually the cross entropy loss. Andrew Ng mentioned in some of his course lecture notes that using minibatches is helpful[50].
Averaging the losses over a minibatch smoothen the gradients while at the same time it prevents the magnitude of the weights to grow out of proportion. All the networks in this chapter use batch normalization[34]. Ioffe et.al mentioned [34] that it not only reduces the covariant shift but also acts as a regularizer and can in some cases eliminate the need for dropout. For the activation function we experimented with both relu[48] and elu[16] but could, in this context at least, not find any noteworthy difference.

Figure 5.1 shows our simple network. It's basically a much smaller version of VGG. The size of the network was found in experiments that tried to find the optimal tradeoff between bias and variance. This proved to be quite challenging given the small size of the dataset. In order to deal with the unbalancedness of the training set we under-sample the samples with a visibility of 20000 meters. We do this by guaranteeing that there are no more than five of those samples in a minibatch.

Table 5.2 contains the results. Using the network as it is without any additional regularization does not yield satisfying performances. The accuracy converges at 0.30 and the mean per class accuracy converges at 0.29. Therefore the network performs even worse than kNN. The reason is probably that on one hand the network is too small to solve the task. On the other hand it would overfit if we made it any larger.

A standard method used to lessen this problem is regularization in the form of dropout[76]. As we can see in table 5.2 the mean per class accuracy increases to 32 percent thanks to adding an additional dropout layer with a dropout probability of 50 percent. Instead of predicting most of the time the label of the largest class the model sincerely tries now to solve the task even though it does not seem have understood the problem yet. In the next section we will check whether we can take the idea of using regularization any further.

So far we used classification to solve the task even though visibility is implicitly continuous. In order to test how regression would perform we build a network similar to Rothe et.al.[65]. The architecture is basically the same as before. In the final layer there are 8 neurons with a softmax as the activation function. Each of those neurons is weighted with a weight between 1 and 8. The sum of them yields the predicted value. We minimize the mean squares loss. The benefit of this approach is among others that we could use it for transfer learning later on. The reason is especially that the activations inside the network do not need to get artificially large in order to make up for the difference between the different visibilities. Instead they remain comparable to the ones derived while doing classification.

The result shows similar scores to the ones we had before. Van den Oord et.al [54][55] showed that a softmax distribution tends to work better than a continuous one, even when the data is implicitly continuous. One of the reasons they mentioned is that a categorical distribution is more flexible and can more easily model arbitrary distributions because it makes no assumptions about their shape. For this reason we
will hence force attempt to solve the fog prediction task as a classification problem. In order to make up for the samples close to a boundary we add label smoothing[59]. This ensures that the network does not get too confident about its’ predictions.

**Simple Neural Network**

![Simple Neural Network Diagram](Image)

Figure 5.1: The architecture of the network used in section 5.1. It only consists of several convolution and maxpooling layers. At the top there are two fully connected layers. We use dropout with a 50 percent drop-rate. The additional features were taken in the center of the image.

**Results of the Simple Neural Networks**

<table>
<thead>
<tr>
<th>Methods</th>
<th>mPCA</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>regression (like in fig.5.2)</td>
<td>0.32</td>
<td>0.28</td>
</tr>
<tr>
<td>class. (like in fig.5.2)</td>
<td>0.32</td>
<td>0.29</td>
</tr>
<tr>
<td>no regularization</td>
<td>0.29</td>
<td>0.3</td>
</tr>
<tr>
<td>maxout</td>
<td>0.33</td>
<td>0.3</td>
</tr>
<tr>
<td>fmax pool</td>
<td>0.27</td>
<td>0.25</td>
</tr>
<tr>
<td>Whiteout (0.05) everywhere</td>
<td>0.25</td>
<td>0.13</td>
</tr>
<tr>
<td>Dropout everywhere</td>
<td>0.26</td>
<td>0.2</td>
</tr>
<tr>
<td>dilated convolution</td>
<td>0.29</td>
<td>0.3</td>
</tr>
<tr>
<td>Maxout + ensemble of nets</td>
<td>0.34</td>
<td>0.31</td>
</tr>
<tr>
<td>global pool</td>
<td>0.2</td>
<td>~0.43</td>
</tr>
<tr>
<td>bilinear attention</td>
<td>0.31</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Figure 5.2: Table with the best results of all the methods in this chapter. We discarded a long list of results with similar or worse results.
5.2 Different Regularization Methods

Nielsen explains [51] the problem of over-fitting as follows: "The main problem is that models with a large number of free parameters can perfectly describe an amazingly wide range of phenomena. Even if such a model agrees well with the available data, it may just mean there’s enough freedom in the model that it can describe almost any data set of the given size, without capturing any genuine insights into the underlying phenomenon. When that happens the model will work well for the existing data, but will fail to generalize to new situations. The true test of a model is its ability to make predictions in situations it hasn’t been exposed to before." Table 5.2 contains the best results of a series of experiments presented in this section.

One way to make our networks better at generalizing beyond the training data is to increase the amount of training data. Artificial additional data can be found by augmenting the data. Since this is not possible for this project, we require other methods. A technique used by Resnet and many other architectures is known as weight decay or L2 regularization. The idea of L2 regularization is to add an extra term, namely the mean squared sum of the weights, to the cost function in order to penalize overly complex models. We use this method for almost all the neural networks in this thesis. The decay parameters were found experimentally.

Another widely used idea called dropout was introduced by Srivastava et.al.[76]. It can be viewed as a stochastic version of model averaging. The main idea is to throw away a certain amount of all the activations. This prevents the nodes from co-adapting too much and therefore helps to reduce the generalization error. Many state-of-the-art networks usually apply dropout to the last one or two layers in a network. However Srivastava et.al [76] showed that dropout in the lower layers still helps because it provides noisy inputs for layers near above which prevents them from over-fitting. We have seen in the last section that adding dropout to the last two layers can improve the scores. The question is now whether adding dropout to other layers might help us aswell. As can be seen in table 5.2, this is not the case. Adding dropout to most convolutional layers only slowed down training but did not help much in preventing over-fitting. A reason might be that the approximation of the weight compensation for disabling dropout at test time is only accurate near the top[5].

Techniques such as whiteout [42] and noise injection [85] simply add Gaussian noise to the activations. This techniques have several characteristics: it smoothens the gradients, regularizes the networks by forcing it to generate more distinct activations and it augments the activations which enlarges the total number of activations that a network will see. Furthermore a big plus of this methods is how easy they are to implement. Although this method seemed to work during our experiments, we can only give the advice to only use this technique at some very specific locations. Using it for every layer did not work at all.

Other methods such as MaxOut [27], L1, fractional max pooling [28], etc. did not...
yield any convincing results either. In all our experiments with different regularization parameters and methods we were never able to get a mean per class accuracy much above 32 percent. Although a better hyper parameter tuning might have led to better results, in the end regularization despite its usefulness can not magically undo all the bad effects of a far too small dataset. Regularization is but one component of an architecture.

5.3 Different Forms of Input

A network is only as good as its input. So far a sample consisted of 6 timesteps and thus was a cube of 64x64x42 after the initial average pooling. In this section we alter the input and measure the changes. To begin with we rerun the best network so far with two time steps and one time step. The additional features contained the means of all six time steps plus the time and the month.

The hypothesis was that by reducing the number of channels we reduce the dimensionality of the input space and thus we might need less data to train the model. Unfortunately we seem to loose valuable information instead. Stronger models and less regularization didn’t help either.

The same problem applies to just focusing to the center and ignoring every thing else by cropping the image. Weather depends on large scale information. Even a local phenomenon such as fog still needs information about the situation far away. Discarding information by cropping can not be suggested, as can be seen in table 5.2.

5.4 Different Forms of Balancing

Since the dataset is heavily unbalanced we often have the problem that a minibatch consists of just one class. Therefore the network quickly learns to always predict the same label. The first idea that might come to mind is to selectively augment the dataset in order to generate a more balanced version of it. This might turn out to hurt the performance instead since the network might start looking for the augmented images instead of trying to get some genuine insight. Using methods akin to SMOTHE will not help either since the dataset is already heavily correlated.

As we already mentioned the dataset is dominated by samples with a visibility of 20km. The easiest method one can apply is to under-sample this kind of images in order to make sure that they do not appear too often in a minibatch. This alone is not enough though. Images with a visibility above 10km still heavily dominate. One option would be to just oversample the fog classes. This results in heavy over-fitting though. We suggest a different option instead. Collecting all the images with a large visibility into one class seem wasteful since it destroys a lot of information about the underlying visibilities.
Moreover as we can see in figure 3.4 in absolute numbers the visibility is a lot better balanced. The problem with the unbalancedness comes from the fact that we only care about a very small range, namely just the samples with a visibility of less than 5km. If we could use a lot more classes the problem that a batch gets dominated by just one label would no longer be that important. We choose 8 classes such that each class consists of at least 500 samples. This amount is comparable to many other widely used datasets. Now a new problem arises: the more fine grained a prediction is supposed to be the more data is required.

This can be circumvented by using as little labels as possible during evaluation. Therefore we aggregate together all the classes with a visibility larger than 10km into one single class. In other words we use 8 classes for training and only 4 for evaluation.

5.5 Conclusion

Neural Networks are data hungry. They need a decent amount of data in order to learn something useful. Training Neural Networks from scratch with very little data eventually results in over-fitting. On the other hand trying to learn only a small network with very little layers only results in having a large bias since the problem of interest is non trivial.

After many experiments the importance of balancing became obvious. Without having the guarantee that a minibatch in general consists of more than just one class, training is almost impossible. The techniques that worked best include oversampling per minibatch, using more labels than needed and aggregating some of them together before evaluation.

The need to regularize the networks also became evident. We experimented with many different methods of regularization. Hence force we will use weight decay, dropout and noise injection where ever it is possible.
Transfer Learning

Traditional supervised learning requires a sufficient amount of labeled data for generalizing well on a given task. Over-fitting is often the inevitable consequence of having a small dataset. This problem can be circumvented by leveraging the already existing labeled data of some related task. This is known as Transfer Learning. In general we reuse the learned knowledge and apply it to our problem of interest. In the case of neural networks we simply reuse the weights of a state-of-the-art CNN pretrained on ImageNet by training a new model on these extracted features.[57] In practice, we keep most of the pretrained parameters fixed and fine tune the rest with a small learning rate.

Edge and blob detectors are usually located at the bottom, while the layers at the top contain more dataset-specific features. Therefore Li et.al. [41] suggest to discard the top layers and train some other classifier using just the rest of the network as a feature extractor. Unfortunately this advice is not directly applicable since we lack a usable bottom layer. The original models expect rgb images instead of our fog samples with 42 channels. As a solution we simple learn a new layer that is capable to understand the input data.

In this chapter we study the application of Transfer Learning to two well known Networks; Resnet50 and VGG-16. Even though the fog dataset and ImageNet are fundamentally different, the hope was that we might still perform better than with training a network from scratch. For our experiments we used the checkpoints provided by Google Tensorflow[1]. The last section studies the use of totally random weights instead of pretrained ones. For all the models in this chapters we use the following parameters: In order to reduce noise and speed up the calculations we resize all input samples to 64x64x42. A Minibatch consists of 4 samples with a visibility of 20km.
and 28 belonging to some other class. We use 8 classes during training and combine them to 4 classes during evaluation. The model were trained on 2 GPUs and the gradient were averaged between them.

### Results of Transfer Learning and Random weights

<table>
<thead>
<tr>
<th>Methods</th>
<th>mPCA</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resnet50 (as is, tb)</td>
<td>0.27</td>
<td>0.4</td>
</tr>
<tr>
<td>Resnet50 (as is, all)</td>
<td>0.25</td>
<td>0.6</td>
</tr>
<tr>
<td>Resnet50 (fc, tb)</td>
<td>0.25</td>
<td>0.16</td>
</tr>
<tr>
<td>VGG-16 (3top + b for 10k steps)</td>
<td>0.3</td>
<td>0.52</td>
</tr>
<tr>
<td>VGG-16 (all for 1k steps)</td>
<td>0.5</td>
<td>0.37</td>
</tr>
<tr>
<td>VGG+AE (3top+b)</td>
<td>0.27</td>
<td>0.38</td>
</tr>
<tr>
<td>Random + autoencoder</td>
<td>0.41</td>
<td>0.48</td>
</tr>
<tr>
<td>Random + elu</td>
<td>0.41</td>
<td>0.4</td>
</tr>
<tr>
<td>Random no autoencoder</td>
<td>0.45</td>
<td>0.54</td>
</tr>
<tr>
<td>Random + variance scaling</td>
<td>0.28</td>
<td>0.4</td>
</tr>
<tr>
<td>Random + just top</td>
<td>0.38</td>
<td>0.48</td>
</tr>
</tbody>
</table>

Figure 6.1: Table with the best results of all the methods in this chapter. We discarded a long list of results with similar or worse results.

### 6.1 Resnet

Deep convolutional neural networks have led to a series of breakthroughs for image classification. The trend led to building ever larger, ever deeper networks, to solve more complex tasks. But as we go deeper, training a network becomes more difficult. Kaiming He et.al presented an architecture that tries to solve these problems and finally led to the victory in the ILSRVC 2015 competition.[31] ResNet is a short name for Residual Network.

In general, instead of learning several levels of features like convnets do, a ResNet tries to learn some residuals. In other words each subsequent layer is only responsible for, fine tuning the output from a previous layer by just adding a learned “residual” to the input. This is implemented by simply adding a shortcut connection which directly connects the input of a layer to its output. It has been shown [31] [32] that this form of architecture not only requires far less weights but is also easier to train and less prone to over-fitting. A variant of this is ResNet50. A residual network consisting of 50 layers.

We used this kind of network for transfer learning with our fog dataset. As mentioned above we trained the bottom and the top layer first in order to adapt to the new input
CHAPTER 6. TRANSFER LEARNING

and output domain. The experiment showed that the model over-fits easily. After only 10000 steps the test error was already a factor 450 times larger than the training error, while the the mean per class accuracy was comparable to random guessing. Training it for a longer amount of time did not help at all. Neither did training other parts of the network for only a few thousand steps (Its not suggested to fine tune on a small dataset for too long).

Global pooling assumes the input to be invariant to translation. Since this is not the case for our fog dataset we exchanged, in a second set of experiments, global pooling with a max pooling and fed it to a fully connected layer. We added dropout in order to prevent over-fitting since the numbers of weights increased. Despite those changes, the results remained comparable to the ones we had before.

We see two reasons for the experiments failures: First of all, traditional transfer learning is trained from the top to the bottom. Therefore a classifier treats all lower layers as a fixed feature extractor for the new dataset. The possibility to over-fit is thus limited. On the other hand if we train the bottom layer as well the network can learn to project the input to a space in which the given weights make the most sense. This includes the projection to a space where over-fitting becomes easy. The second, probably more important reason we can think of is that networks trained on ImageNet tend to detect different kinds of shapes. However in our dataset shapes are of no consequence. In fact the ability to do so anyway seems to simplify recognizing already seen images too much. For a small and heavily correlated dataset this can be deadly.

6.2 VGG

Another well known and widely used network architecture is VGG[73]. It is basically a large traditional convnet with dropout. Like before we have to exchange the bottom layer and the top layer. Furthermore we have to change the size of the last convolution from a 7x7 kernel to a 4x4 kernel since the input samples are smaller than the images that the original got. To sum it all up: We chose to train the bottom and the top three layers. Since those layers store very dataset specific information we wont loose much by removing them. In order to avoid over-fitting we made them slightly smaller. Thus we experimented with 512, 1024, 2048 neurons per layer instead of 4096.

Table 6.1 shows the results of our efforts. The mean per class accuracy converges after 10000 steps at about 29 percent. If we now keep training all the weights for only 1000 steps the mpc accuracy increases to about 37 percent. Training them any longer only leads to over-fitting and a reduced accuracy. Using an SVM afterwards as an alternative for the fully connected layer did not help either.

In a second experiment we tried to regularize the bottom layer by adding a second objective in the form of an auto encoder. The objective is to not allow projections
to overly beneficial spaces in which over-fitting becomes too easy. Furthermore we trained the autoencoder and the model simultaneously such that the output of the bottom layer can be read by the rest of the network. As can be seen in table 6.1 the results of the experiments are not convincing. The mean per class accuracy remained about the same, while the overall accuracy decreased.

We experimented with other training policies as well but could not find any changes that yield performances worthy to be reported. Overall the experiments failed mainly because the patterns learned for imageNet do not help to solve weather problems. This is true even though we removed all the top layers that usually store very dataset specific patterns. Ultimately the two datasets are just too different. We did not investigate in detail what kind of weights the network should learn but we believe that it should put more emphasis in recognizing different kinds of distributions instead of focusing to detect edges and shapes. The exact boundaries of clouds are irrelevant after all.

Furthermore since we have to exchange the bottom layer all the layers above are affected. It would require a lot of effort to retrain a decent bottom layer. The amount of available data might simply not be enough for that. The result is that a lot of weights might have lost its’ original purpose and thus no longer provide any useful information. Using an autoencoder to train the bottom layer might have failed because a pixel wise loss might learn different features than the classifier requires.

### 6.3 Random Weights

During our previous experiments we recognized that our models behave well when loaded with random weights. In this section we intend to explore this discovery by training different parts of the VGG network presented above without having loaded any pretrained weights. In other words we initialize the weights and leave them unchanged afterwards. The only layers we train are the top three and the bottom layer. Experiments show that this kind of setting produces significantly better results than any other models presented in this thesis. Overall the models trained with random weights showed a far smaller tendency to over-fit.

Before we take a look at the different experiments lets get some background knowledge first. Neural Networks are usually initialized with random weights. Not doing that would prevent the network to learn anything at all. Xavier Glorot et.al [25] proposed to sample weights from a distribution with zero mean and variance. This ensures that signals, as they pass through a large number of layers, do neither grow too large nor become too small. A more recent approach used by most state-of-the-art architectures is called variance scaling [79] and attempts to keep the scale of the input variance constant. During most of our experiments, if we do not explicitly state otherwise we will use Xavier initialization.
In all the experiments presented here we use the very same architecture as in the last section. The results of are listed in table 6.1. All differences from the standard architecture are listed as well. The very first experiment that we conducted used fully connected layers with 512 neurons. The first layer was regularized with an autoencoder. The activation function was relu. Even though we ran the experiment three times with different seeds we always measured a maximal mean per class accuracy of about 41 percent. This indicates that random weights are useful regardless of the used seed.

A better result was achieved with 1024 neurons in the fully connected layers and a large dropout probability of 70 percent. We trained once again the top three and the bottom layers for about 35e4 steps and regularized with an autoencoder. With this settings we achieved a macro F1 of 45.9 percent and an accuracy of 54.8 percent. The confusion matrix is given in table 6.2.

![Confusion Matrix of the best Random Experiment](image)

Figure 6.2: The confusion matrices for the best experiment. Here we only trained the top three and the bottom layer (no autoencoder). We measured a mean per class accuracy of 0.459 and an accuracy of 0.547.

To show the importance of randomness we trained all the weights for only about 2000 steps. The performance instantly collapsed. The mean per class accuracy dropped to less than 30 percent. This small amount of learning brought some order into the weights, which hindered the model to work properly. Another experiment that we conducted initialized the weights with the often used variance scaling initialization instead of the Xavier initialization that we used so far. If we use randomness it might matter a lot from which distribution we sample the weights. As can be seen in table 6.1, variance scaling is not capable to surpass a mean per class accuracy of 28 percent. The result clearly shows that the weights need to be sampled from the right distribution or the model will not be able to generalize well.

Experiments that used different activation functions such as elu instead of relu yielded at most a mean per class accuracy of 41 percent. Getting rid of negative activations seems to help avoiding to over-fit. This result is comparable to experiments that used a slightly smaller dropout probability or slightly larger fully connected layers. This might indicate that elu made the model too powerful such that it generalized a bit worse due to the small amount of available data.
Learning Curves for Random Weights without an Autoencoder

Figure 6.3: The test loss and the mean per class accuracy learning curves for the random experiment without an autoencoder. Here we only trained the top 3 layers and the bottom layer. As we can see the model does not overfit much. The model reaches a maximum at about 400k steps, but begins to saturate a lot earlier.

Next we tried different training protocols. We ignore the autoencoder regularizer this time and train just the bottom layer without any external influence. The result did not change much. We received once again a mean per class accuracy of 45. The only difference that can be recognized is that the training process takes a lot more time.

In the last experiment that we present in this section we only train the top layers. We can see in table 6.1 that the model still performs as well as a SVM but it no is longer capable to generalize as well as before. This might be because we train one layer less than before. One thing that we noticed regarding this model is that it trains incredibly slowly. We need about one million steps until it saturates.

The question that remains is why random weights work at all? More experiments are required to find out whether this effect also appears in other settings with very different datasets. One option is that this phenomenon only appears in datasets that are quite noisy and chaotic. Sets where recognizing shapes, boundaries and edges has no relevance. In this case we found reasonable weights for the fog dataset by coincidence. An alternative option is that we stumbled onto something that works
in general. If the paradigm can indeed be used for other sets a possible explanation would be that having enough random weights leads to a similar effect as random projections do for a rbf kernel in SVM training.

6.4 Conclusion

It proved to be challenging to transfer knowledge between networks using fundamentally different datasets. Even feeding the data is not possible without retraining the base layer. We would require a lot of data to properly overcome this hurdle. If we had more data though, it would probably be better to retrain large portions, if not all, of the network since the learned patterns are simply far too different. In other words doing transfer learning between datasets like ImageNet and the fog dataset is not possible.

This point is supported by the fact that it is obviously better to use random weights than the ones we got from pretraining. In future work we need to find out whether the discovered fact only applies to this particular dataset or if it might be generalizable. The connection to rbf kernels needs to be studied.
In the last chapters we found that none of the standard methods yielded satisfying results. In this chapter we will experiment with alternative methods and sometimes very unconventional architectures.

We need to mention that many of the presented models contain a long list of little hacks that allow us to improve the results. This is particularly true for the adversarial models.

7.1 Generating an Encoding with a VAE

The performance of algorithms such as SVMs depend on finding good features. Given our limited knowledge about weather and its underlying processes it was not possible to do proper feature engineering. Although neural networks can to some degree learn features by themselves, they require large amounts of data to do so.

An alternative are Autoencoders. They are capable to learn a lower dimensional representation in an unsupervised manner. Instead of relying on a single label per image, they learn to reconstruct images on a per pixel level. Therefore a smaller amount of data is necessary to get a decent result. The hope is now that they are capable to learn a representation that we can use as an input to an SVM. After all with appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

A variant of an Autoencoder that is capable to learn latent features instead of just some random encodings is called Variational Autoencoder (VAE)[38]. This architecture has an interesting characteristic: it is possible to generate new samples. We would like to find out whether we can use this to balance our dataset.
7.1. GENERATING AN ENCODING WITH A VAE

Architecture: VAE + Classifier

Figure 7.1: The architecture is basically an ordinary VAE with the exception that we use an additional Classifier to force the latent to be helpful during classification. We train this model just like an adversarial autoencoder.

Sampling becomes possible because VAEs add a constraint on the encoding network, that forces it to generate latent vectors that roughly follow a unit Gaussian distribution. The loss function is given by the sum of two separate losses: a loss that measures the goodness of the reconstruction (usually the pixel wise negative cross entropy or the mean squared error), and a latent loss, which is the Kulmar Leibler divergence that measures how closely the latent variables match a unit Gaussian.[2]

We need to apply a simple reparametrization trick to optimize the latent loss: the encoder predicts both the mean and the standard deviation that are used to sample the latents. The decoder tries to reconstruct the original image based on the sampled latents. In practice it proved to be challenging to reconstruct the original images. We had to do some changes to the original architecture.

A phenomenon VAE are known for is that the variational regularization term causes some of the latent units to become inactive during training [44]. Sonderby et.al [75] state that the collapse of many units can hinder a VAE to learn a useful representation. Bowman et.al [8] proposed a solution that seems to help in some cases. He suggested to slowly increases the strength of the latent loss from 0 to 1.

The second change we introduced is a learnable map that gets added as an additional channel in the decoder. Since the geography never changes adding a layer that allows the decoder to store additional more localized patterns lead to a better reconstruction. Furthermore it limits the amount of information that needs to be passed through the bottleneck. The learned channels can be seen in figure 7.2. It seems like it learned a
representation of the mountains in Switzerland.

**Learned Additional Channels**

![Learned Additional Channels](image)

Figure 7.2: The learned additional channels in the decoder. They learn images that look similar to the mean of all channels. If we compare them with an actual map we can recognize the main structure of the Alps. In other words it learned something about the geography.

After some failed experiments we decided to leverage the given label information to regularize the latent representation of the autoencoder more heavily in order to push the latents in a more useful direction. We added two fully connected layers with the sole task to incorporate the given labels. Following the training principle introduced by the Adversarial Autoencoder[46] both the adversarial network and the autoencoder are trained jointly with ADAM in two phases. First we update the encoder and decoder to get a good reconstruction. Then we update the encoder and the classifier to find features that are fit to be used for classification. It can be observed that introducing an additional classifier leads to dramatically better reconstructions. This change was also recognizable on the reconstructions of images of the test set. This may indicate that the introduced chaos by the classifier activates weights that otherwise would be turned off. Our conclusion is basically that whenever there is a chance to incorporate label information, you should use them. A reconstruction can be seen in figure 7.3.

After we trained our VAE on our fog data for about 500k steps we sampled a more balanced dataset and trained a SVM with it. Each sample consist of the 700 derived latents plus the time and the mean values in the center of an image. As a result we measured a mean per class accuracy of 41 percent and an accuracy of 51 percent. We managed to improve on the score of the SVM baseline. This shows the importance of having the right features and a balanced dataset. A question that remains is how diverse the different samples are. We did not have the time to answer this question properly yet.
7.2 Multi Task Learning

Human learning is more complicated than traditional supervised learning. Ruder gives the following intuitive example: "For learning new tasks, we often apply the knowledge we have acquired by learning related tasks. For instance, a baby first learns to recognize faces and can then apply this knowledge to recognize other objects. From a pedagogical perspective, we often learn tasks first that provide us with the necessary skills to master more complex techniques." [66]

Multi Task Learning allows us to apply this principles to neural networks. The most commonly used method was introduced by Caruna et.al. [9] and shares the hidden layers between all tasks. Only the output layers are task-specific. This so called Hard parameter sharing greatly reduces the risk of over-fitting. In fact, [3] showed that the risk of over-fitting the shared parameters is an order \( N \) – where \( N \) is the number of tasks – smaller than over-fitting the the output layers. The more tasks we are learning simultaneously, the more our model has to find a representation that captures all of the tasks and the less is our chance of over-fitting on our original task. [66]

For the fog dataset we could not get any other labels within the time span of this thesis. It would have been interesting to also predict other parameters such as the temperature among others. The learned features would have been helpful when predicting fog. The only option left was to use an unsupervised method. The idea is basically to use the decoder of an autoencoder as a second task. The encoder encodes the input for both the decoder and the classifier. Figure 7.4 shows the chosen architecture. Similar to Adversarial Autoencoders there are two phases that are run by taking turns: First we update the encoder and the encoder, then we update the encoder and the classifier. The hope is that the autoencoder regularizes the classifier enough to prevent over-fitting, while at the same time the classifier is flexible enough to choose proper features. In particular we hope that the autoencoder helps us to train all the important low-level features.

In practice this approach proved to be quite tricky to train. Small changes to the architecture are enough to either horribly overfit or not learn at all. After many experiments we achieved a mean per class accuracy of 38 percent on the validation set. This result is comparable to the SVM baseline. On the testing set however the
measured values were higher. This clearly indicates that we over-fitted on the test set. Nonetheless we improved the score over learning a standard network from scratch by about 18 percent.

In retrospect it would have been better to use other classification tasks instead of an autoencoder. Focusing on the reconstruction of a picture at the pixel level might not help to learn interesting features of the kind that supervised learning induces and thus harm the performance. [52] Furthermore it would be advisable to find a less fragile training policy.

7.3 Artificial Dataset

Children play many silly games. Sometimes neither the purpose nor the actual rules of their games can easily be understood by adults. They might consider their actions to be childish. However we argue that children can train many useful abilities by playing seemingly useless games. The condition is simply that abilities are trained that can help to solve other task.

Applied to neural networks this would imply that we could use almost anything for training as long as it shares the main characteristics of the real dataset. Simply put we claim that it should be possible to use totally synthetic data without any practical application for Transfer Learning and still learn useful features.

Shrivastava et.al [72] showed that synthetic data can improve the performance of a
38 7.3. ARTIFICIAL DATASET

Some Artificial Cloud Images

Figure 7.5: Some artificial images. Those three should represent cloud coverage. Although the results have a high variability all of them look somewhat similar. It was not possible to get a better reconstruction despite having tried various architectures.

classifier. There is one catch though: they already know beforehand what kind of gesture will cause which label. Thus they can easily simulate new images. For the fog dataset this is not possible. For us it is not clear how a fog image differs from a clear sky image. As an alternative one could try to learn to produce similar looking images with a GAN model and later on train the network to predict the underlying random number.

During the course of this thesis we experimented with many different frameworks: Vanilla GAN[26], CGAN[47], BEGAN[6], to just name a few. None of them seem to work. The problem is an innate characteristic of the dataset. It is far too noisy and has no obvious structure. In computer vision, a simple method to produce images that look like clouds is to take some random noise, to blur it and to scale it up. This process is repeated a few times. The generator of a GAN has a similar protocol. You take a random vector, you apply a convolution, scale it up and repeat the process (using resize convolution or transposed convolution makes no difference since the principle remains the same). As a result its far too easy for the generator to fool the discriminator. A proper Nash equilibrium can not be established.

The benefit of ACGAN[53] and InfoGan[14] is that the networks have additional tasks to solve. Namely predicting the labels in the case of the ACGAN and trying to predicted the used random number that was used to generate the image for the InfoGan. This additional task allowed us to finally establish an equilibrium. Figure 7.5 contains the results. Although the fakes kind of look like clouds, the result is not truly convincing. The structure of the fakes seems to look different and far noisier than the original. This might hint that the discriminator focused more on solving the other tasks and did not really recognize the difference between the real cloud images and the fake ones.

Feeding the generated images into a classifier didn’t allow it to learn proper features. Instead it learned to correctly predict the fake labels within a few thousand steps. It
made no difference whether we used a mixture of gaussians or some other method to produce the pair of underlying noise and label. It is possible that the generator hid some patterns that a strong enough classifier can instantly recognize and thereby conclude how the underlying noise might have looked like. The classification itself would be easy.

## 7.4 Conclusion and Remarks

There are several conclusions we can make: First of all we tried to reduce the dimensionality with a vae in order to provide decent features for a SVM. While doing so it became obvious that autoencoders of any kind tend to focus on very different things. It might focus on patterns that are completely irrelevant for the prediction of fog, and treat everything else instead as noise that can be discarded. Therefore it might oppose the very fine grained nature of the fog phenomenon. This problem can be circumvented by adding an additional classifier that forces the latents to be more in line with the needs of the classification task.

The problem of having two totally different objectives also surfaced during the experiments in section 7.2. Although we were indeed capable to get results similar to the SVM baseline, the model proved to be quite fragile and hard to train. Small changes in the architecture or with any of the parameters lead to the model failing to generalize. The conclusion we can make is basically that we should not use a autoencoder as a second task, unless we have no other choice.

Training an artificial dataset might help but if you have no clue what kind of constellation leads a particular label we can not suggest to try it. This is especially the case if the dataset is incredibely noisy and without any clear shape. In such situation GAN seems not to be applicable yet.

We also made some more experiments we did not report in this chapter. For example we tried to do some kind of artificial semi-supervised learning. We took different crops and used them as additional images. The labels were assigned similar to Lee et.al [40], meaning we took what ever the classifier predicted as a pseudo label that we could use for training. Although this approach yielded good results on the training set, we were not capable to repeat the results on the validation set.

Another method we were experimenting with is oneshot learning following the example of facenet[69]. We do not report about it since we did not repeat those experiments on the validation set yet. The model seems to have overfitted though. As far as we understand it, oneshot learning allows to use very little images per label but it asks for a large number of labels instead. We only have very little data and very few labels. Thus our dataset seems not to be the right usecase for oneshot learning.
We will first briefly discuss the success of the different methods we used before we will talk about what the best results that we got signify.

8.1 Used Methods

In this thesis we tested a large number of different models for predicting fog with a non-optimal dataset. This task proved to be quite challenging. Most standard methods fail for different reasons. The most important one is the limited number of really different samples in the minority classes.

First we tested some simple standard methods. Especially SVM is usually a good choice when facing high dimensional data. On the other hand many of them perform badly when facing unbalanced data. Over- and undersampling can help solve this problem. Another difficulty is the need of having useful features. Given that no one truely seems to understand the domain yet, feature engineering is challenging. The results were accordingly. Nonetheless SVM performed quite well compared to other methods, despite not having ideal features. This showes that it is often benefitial to use as simple methods as possible.

Retraining a neural network from scratch would require to have a large amount of data. Since we do not have such we tried to only train a small network. Reducing degrees of freedom hinders the network to overfit too quickly. The downside of this is of course that we will begin to underfit instead. The fog dataset proved to be particularly difficult. The networks that we experimented with were on onehand too small to solve the difficult task and on the other hand too large to not overfit. This is probably not only caused by the small total size of the dataset but also by the heavy temporal
correlation between the samples. Regularization did not help much either. It only prevents overfitting after all.

Doing Transfer Learning is usually a powerful tool. It is often only possible though, if you have a dataset that is somewhat similar to the original domain. If the input format is entirely different and there are almost no shared characteristics then it might be better to retrain the entire network. This proves to be challenging though if you have only very little data.

We also experimented with some unconventional methods. Using an autoencoder might have helped to train the lower layers. This is particularly helpful in cases where all other methods fail. On the other hand since it has a tendency to learn a very different kind of features than a classifier it might also hurt the model enough to prevent it truly learn. This reasoning also stands for our second best approach, generating an encoding with a VAE. This method helped us to reduce a lot unnecessary noise. The downside of it is of course that it is really tricky to not discard important features. This is especially a problem when predicting fog. It is likely that we lost a lot valuable information and could have performed much better with a better trained encoder.

The method that performed best is actually surprisingly simple. Using random weights seems to work especially well. The big plus of the method is its’ simplicity. The architecture is basically just VGG. Thus this method could be used without having to program a lot. Just training several top layers and the bottom layer also simplifies the training procedure. Many weights do no longer have to be updated after all. Of course the network looses degrees of freedom which might result in lower performances for some specialized tasks but since we need far less data for training, it will be worth it. The main question that remains is why does it work at all. If our findings can not be generalized to other fundamentally different datasets this might indicate that we found reasonable weights by coincidence. Weights that can deal well with heavily noisy datasets like the fog dataset. We could therefore only use it for related problems.

The more interesting case is if our findings can be generalized. We believe that might actually be the case. Using random weights is basically comparable to multiplying the inputs with a random matrix. Thus we basically extract random features and the linear classifiers at the top will do the entire classification alone. Similar methods have already proven to be very effective, especially in the context of rbf kernels. For example Rahimi et.al proposed to map input data to a randomized low-dimensional feature space and then apply existing fast linear methods [61][62]. The similarity is obvious. More experiments need to be made in order to check this.
8.2 Can we predict fog?

We still need to check how well we performed in the end. With the random weights method we achieved our best result. As mentioned before we achieved a macro F1 score of 45.9 percent and an accuracy of 54 percent. The question that remains is whether that is any good. Let’s assume we classify everything with a visibility of less than 5km as fog. If we look at the covariance matrix we can calculate probability of having fog when we predicted fog of 68 percent. On the other hand the probability of having predicted fog when there actually is fog is 63 percent. Although this result is not good enough for practical use, we are still a lot better than random.

This indicates that the task is indeed solvable. You can make predictions based on standard data products of the operational weather model. This is still possible even though data is not optimal. We are confident that the result could be improved a lot if we would get more data. There might be an upper boundary though. Our input was produced by a weather model that simulated the situation 24 hours into the future. Therefore it is not guaranteed that the model does contain all the information that real world data would have. Thus relevant information might be missing from the very start. Furthermore the model itself is not perfect thus some of the information might be completely wrong. With more data we could improve the score but since we are no domain experts we can not predict what the maximum possible score might be.
8.2. CAN WE PREDICT FOG?
the main goal of this bachelor thesis was to investigate, whether it is possible to solve a complex task like weather prediction with a non optimal dataset. For this we experimented with many different models such as learning neural networks from scratch, transfer learning, oneshot learning and multi task learning. We experimented with new models including networks relying on random weights. Furthermore we tried to create artificial datasets with GANs.

9.1 Conclusion

The main conclusion of this thesis is that Data Science and Deep Learning in particular is no magic. If you dont provide a useful dataset and follow certain rules, even the best algorithms will fail for sure. An optimal dataset would be incredibly large, well balanced and as distinctive as possible. Unfortunately many real world datasets can not guarantee any of those. In this thesis we tried to apply as many different models as possible in order to test how they behave with a dataset that is anything but optimal. After many experiments we reached the following conclusions:

We do not have to learn everything. Instead it might be worthwhile to leave some parts of a network untrained. Random weights can improve a score a lot if done right. Furthermore things that are not trained generally do not over-fit. It is important though to use the right distribution when initializing the weights. xavier initialization proved to be helpful.

Any model, including SVMs, require good features or they will not work properly. Doing Dimensionality Reduction with a VAE can yield such features. In practice it is tricky to make sure that the right kind of information is discarded.
Gan Models have a serious problem with heavily noisy input that lacks proper shapes. Such inputs lead to a situation where the generator can fool the discriminator far too easily.

Multitasking can improve your score. Even though an auto encoder may not be the best choice since it learns features that differ from the kind of features that a semi-supervised classifier tends to learn, it can still improve the score for the fog dataset. We would not suggest to use our presented method though.

Given that we were capable to make predictions that were a lot better than random, it seems that weather can indeed be learned. This might hint that many problems with a large complexity do not need to be explicitly modeled. It should never be forgotten though that Machine Learning and Deep Learning in particular follow their own set of rules too. This algorithms can not cause miracles. But they can yield staggering results if their conditions are fulfilled. One of those is to provide them with a reasonable dataset.

9.2 Future Work

There is a lot of future work that can be done. The most obvious is of course to repeat the task with a lot more training data. The authors are convinced that it is indeed possible to successfully predict fog. We were already capable to get results out of the dataset we had. An optimal one might yield much better results. Moreover it might be worth it to try to learn other very complex problems instead of attempting to model them. This might not only dramatically limit the required amount of computation but we might also be capable to solve problems that were so far unsolvable. Of course this would be the optimal case, but it is definitely worth a shot.

The project showcased the need to find architectures and techniques that would allow us to train with very little data. Many areas of research can only provide very non optimal datasets. Be it because of the difficulty to produce the imagery or labels, the inability to produce a large enough amount of data hinders us to solve a great deal of problems. A broad range of applications would benefit from such a solution.

The biggest surprise of the thesis was ultimately that random weights can perform so well. We need to learn more about the conditions necessary to learn properly and about the limitations of the process. In order to find out whether this approach only responds to this particular dataset. If we could generalize it to other kinds of inputs, there would be an immense number of applications. We would propose to use it for compression. The random weights are produced by a pseudo random generator. Thus it might not be necessary to store so many weights because we could just compute them on the fly.


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Fog Prediction at Zurich Airport with Deep Neural Networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):  
Kevin Kipfer

First name(s):

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

28.9.2017

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.