Active learning with anomaly detection in object recognition

Author: Hanne Nijhuis  
hamne.nijhuis@student.uva.nl  
0364568

Supervisor: Dr. M.W. van Someren  
m.w.vansomeren@uva.nl

Wednesday 1st May, 2013
Abstract

The main goal of this work is to investigate the possibilities of improving existing active learning strategies by looking at anomalies or outliers in a dataset. To be more specific, we introduce a sampling strategy for a pool-based scenario which uses misclassifications on the train set to find an informative instance. Furthermore, we use a stopping-condition to define that, if a sample stays misclassified for a certain number of iterations, we consider this sample to be of negative influence on our classification algorithm and we therefore remove it from our train set.

We evaluate our approach by running experiments on the VOC PASCAL challenge data from 2008. This object recognition challenge provides sufficient labeled data to apply active learning without the need for a human annotator as an oracle. The results show that the trigger we use for our approach, the misclassifications, happen very rarely. The overall results are therefore somewhat marginal. Fortunately, in the specific situations where we do run onto a misclassified sample, we can successfully show the effect of our strategy and the improvements it shows over the basis strategy.
Acknowledgements

First and foremost, I would like to thank my supervisor Maarten van Someren for his long lasting commitment to this project. Our numerous brainstorm sessions led to all the insights I hope to bring across in this thesis. Thank you for your quick responses, your availability, your helpful thoughts and inspiring ideas.

I would also like to thank Henk Zeevat and Gwenn Englebienne for taking place in my graduation committee.

After several failed attempts, I realized I would never finish this project on my own. I would like to thank Hania Kaufmann and all members of the group for their support.

10 years ago I started studying Artificial Intelligence at the UvA. I have never regretted that decision and am grateful for all the cool projects I got to do, all the knowledge I got to absorb and all the cool people I met. For their much appreciated support in these last few months I would especially like to thank: Anne Schuth and Gijs Molenaar.

Outside of the university I want to thank Bart Geleijnse and Jur Jacobs, my former roommates, for motivating me and for always being there for me.

Last but not least I would like to thank my family: Gerard, Bertine, Marijn, Jort, Margé, Noa en Jonathan. You have inspired me and supported me in every possible way throughout my study.
# Contents

1 Introduction .................................................. 1
   1.1 Machine Learning ........................................ 1
       1.1.1 Logistic Regression ................................. 3
       1.1.2 Anomaly detection ................................. 4
       1.1.3 Active Learning ................................. 4
       1.1.4 Practical Considerations ......................... 8
   1.2 Motivation .............................................. 9
   1.3 Research Questions .................................... 10
   1.4 Purpose and Contributions of this thesis ............ 10

2 Background & Literature Review ............................. 11
   2.1 Computer Vision ....................................... 11
       2.1.1 Use of context ................................... 12
       2.1.2 Challenges ...................................... 13
   2.2 Context in Image Understanding ........................ 13

3 Method ......................................................... 19
   3.1 Terminology ............................................ 19
   3.2 Query strategies ....................................... 20
       3.2.1 Uncertainty Sampling ............................. 20
       3.2.2 Certainty Sampling ......................... 21
       3.2.3 Positive & Negative Sampling ............... 22
       3.2.4 Mode & Mean Sampling ....................... 22
   3.3 Useful Data Properties .................................. 23
       3.3.1 Variables ....................................... 24
   3.4 “Something interesting” ................................ 25
       3.4.1 Misclassification in the train set ............ 26
       3.4.2 Unexpected label from the oracle ............ 27
   3.5 Approach ............................................... 28
       3.5.1 Nearest Neighbors .............................. 28
       3.5.2 The Algorithm .................................. 29
       3.5.3 How to deal with ‘something interesting’? ... 31

4 Experiments & Results ....................................... 33
   4.1 Implementation ....................................... 33
   4.2 Data ................................................... 34
       4.2.1 Feature Representation ....................... 35
   4.3 Experiments & Results ................................ 35
4.3.1 Experiment 1: A baseline for classifying on gist data . . 36
4.3.2 Experiment 2: performance of our plug-in on uncertainty and positive sampling . . . . . . . . . . . . . . . . . . . . . 40
4.3.3 Experiment 3: examples of misclassification and anomaly removal . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43

5 Conclusions, Discussion & Future Work 49
5.1 Conclusions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 49
5.2 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
5.3 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50
Chapter 1

Introduction

In this chapter we will give a description of the field with which this research project is concerned. We will discuss the basics of machine learning to provide a better understanding of the contribution we make to this field. A scientific motivation for this research, along with a formal declaration of the specific questions we want to answer is given at the end of this chapter.

Our project is also concerned with elements of computer vision, but these will be discussed in Chapter 2.

The main goal of our work is to investigate the possibilities of improving existing active learning strategies by looking at anomalies or outliers in a dataset. To be more specific, we introduce a sampling strategy for a pool-based scenario which uses misclassifications on the train set to find an informative instance. In the next section we will elaborate on what these terms mean, and how they are used within the field.

1.1 Machine Learning

Machine learning is concerned with learning from data. For example, a system could have the goal of distinguishing spam emails from non-spam emails. To be able to do so, such a system needs some sort of starting point at which the receiver labels emails as either spam or non-spam. This can be referred to as supervised learning, meaning that we have labeled data to train a model on. With this model, the system will now have the ability to predict a label (e.g., spam or non-spam) for new data (e.g., new emails). If we are concerned with data for which we don’t have any labels, but we want to infer knowledge from it anyhow, we are talking about unsupervised learning. This means for example that a system could look for clusters in a dataset, or correlations between several properties of the data. Clustering, market segmentation and social network analysis are only a few of the examples for which unsupervised learning algorithms can be used [1].

Learning means we want to find a function or model which correctly maps data as input to a label as output. The representation of the input, output and the model can be anything. In the above example of recognizing an email as being spam, we can imagine the input to be certain properties of an email, which
we call features. Useful features could be the length of the email, the occurrence of certain predefined keywords associated with spam, and the sender. As output, or label, we want to be able to tell whether an email is to be considered spam or not, maybe even expressed as a probability. The model needs to somehow map the input values to a single output. This is often done by creating a function in which all input values are added up, each multiplied by some weight defining the importance of that feature. The goal of the learning algorithm is to simply find the best possible weights for all features so that the input and output of the available data correspond. This process is referred to as model fitting.

More formally, we use the notation $x$ for an input variable, $y$ for an output value or label. We need a function to map any $x$ to a $y$, which we call a hypothesis $h$. To denote a model, we use $\theta$. A training-example can be defined as a pair $(x, y)$. Generally, we use the notation $h_\theta(x)$ for the predicted output for input $x$, according to model $\theta$.

A hypothesis is considered to be consistent with its training data if there is no empirical error. This means that if we were to use the hypothesis to classify the train set, all labels would be predicted correctly. This may seem like an obvious goal, but limitations in the representation of a hypothesis could form a restriction which makes this impossible. This may lead to misclassifications on the instances on which the model was trained.

This brings us to the problem of fitting the right model to your data. Since the train set is often just a small sample of the entire distribution, finding a model that fits perfectly to this train set might not be the best option. If the model does not generalize well enough then testing the model on a different subset of the same data distribution might show very low performance. This is called overfitting and is a very common pitfall in machine learning problems. The model has become too specific to capture the general aspects of the data distribution. It will probably perform badly on any subset other than the train set, because it is unable to cope with the variance in the data. In contrast, the model could become too general in the sense that it is not able to capture the true properties of what is being learned.

There are two types of learning problems. If the goal is to predict a value in a continues output space (say any number between -100 and 100), then we talk about regression. In regression problems we are looking for a function which can give us an output value for any input. A simple way to think about such problems is to visualize them as a plot. Here, we draw a line of values for $y$ on a vertical axis, set out against input values for $x$ on the horizontal axis. Once a line is fitted to the points, one can predict the output value for any new input value. A regression problem could also have more than one feature as input.

The other type of problems is referred to as classification problems. In classification we are concerned with predicting to which class, out of a limited set of classes, an input sample belongs. The previously described problem of detecting spam in emails, is such a classification problem. More precisely we call this problem a single-class classification problem. For any input (an email) we only define whether it does or does not belong to a class (in this case spam).

There are also multi-class problems, where an input sample can belong to one out of many classes. Sometimes a sample can even belong to more than one class. In object recognition for example, an object might belong to the class
'person', or 'car', or 'boat', but if we were to label an image with the objects it contains, that image very well belong to all three classes. We have to be aware that this specific example can actually be interpreted as a multi-class, or as multiple single-class problems. Either we try to predict the correct class or classes for each sample in a single model, or we train a separate model for every possible class, in which each model is only concerned with correctly predicting whether an object does or does not belong to that class. This is a design choice for which, in this project, we chose the latter approach.

1.1.1 Logistic Regression

Although the name may seem to suggest otherwise, logistic regression is actually a method for classification. It uses the logistic function to map values to the [0, 1] interval. It is one of the most widely used algorithms nowadays. As this is the algorithm we use in our project, we will give a short description of how it works. Logistic regression is based on linear regression, which is a regression method for fitting a linear model to data. We are working with classification, so we want the output to be binary, either 1 or 0. The logistic function (or sigmoid function) has that property, in the sense that for any \(x\), the corresponding \(y\) lies between 0 and 1. The function is defined as

\[
g(z) = \frac{1}{1 + e^{-z}}
\]

If we apply this to the standard function for linear regression \(h_\theta(x) = \theta^T x\), then we get

\[
h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}
\]

We need to map the output from the interval [0, 1] to the binary set 0, 1. To accomplish this we use \(h_\theta(x) > 0.5\) to predict \(y = 1\) and \(h_\theta(x) \leq 0.5\) to predict \(y = 0\). According to the logistic function, this corresponds to \(y = 1\) if \(x > 0\) and \(y = 0\) if \(x \leq 0\). This means we can interpret our hypothesis to represent the probability that \(y = 1\), denoted as \(p(y = 1|x; \theta)\).

To fit a model \(\theta\) to the training samples, we declare a cost-function which we will minimize. A standard way of fitting a model is to minimize:

\[
J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})
\]

Where \(m\) is the number or training samples and Cost is declared as:

\[
\text{Cost}(h_\theta(x), y) = \begin{cases} 
- \log(h_\theta(x)) & \text{if } y = 1 \\
- \log(1 - h_\theta(x)) & \text{if } y = 0 
\end{cases}
\]

We can simplify this into a single formula:

\[
\text{Cost}(h_\theta(x), y) = - y \log(h_\theta(x)) - ((1 - y) \log(1 - h_\theta(x)))
\]

Any standard optimization algorithm could be used to minimize this cost function (e.g., gradient descent, conjugate gradient, BFGS, etc.). Normally, having fewer training samples than features (especially if \(m << n\)) makes logistic regression prone to overfitting. This is solved by using regularization, which
penalizes large parameter-values. It is outside the scope of this project to discuss these properties in more detail. For this project we used an implementation based on work by Koh, Kim and Boyd, which is well explained in [2].

After the model is fitted, we can think of it as a boundary which divides the feature space into a positive part (where $h_\theta(x) = 1$) and a negative part. This boundary is called a decision boundary and is a line or an $n - 1$-dimensional (hyper)plane, depending on the amount of features.

1.1.2 Anomaly detection

An anomaly is defined as something that deviates from what is standard, normal or expected. Anomaly detection is common practice especially in the field of statistics. For ages, people have tried to find anomalies in their data to be able to remove them. A few strange (and maybe even unwanted) instances in a dataset may lead to completely unexpected statistical results, so it might be better to remove those, or at least be able to detect them.

Anomalies should not be confused with noise however. We can define noise as incorrectly labeled instances, while anomalies are correctly classified but unexpected instances. If we consider the ground truth of our data to be free of noise, we have to consider anomalies as either interesting or uninteresting peculiar instances of the class. In the setting of this project we might be looking for cars and running onto a toy car, which looks quite different and appears in different contextual environments, could be considered an anomaly. Obviously it would be a valid argument to say that this toy car should not have been labeled as a car, but this is unfortunately a common problem in computer vision; the ambiguity of labels and classes. But that is also the crux of what makes these anomalies so interesting. If we are able to at least detect that this is a special version of a car, within the complete set of cars, then we can use this information to improve our classifier. In section 3.4 we explain how anomalies could be used in an active learning strategy.

In comparison, we just mentioned noise can be defined as incorrectly labeled instances. This is not entirely true, because noise can also occur on a feature level. Imagine a setting in which sensors provide the feature values. If a sensor is malfunctioning it may occasionally output strange values which should be considered as noise. On the other hand, labeling may also cause noise, for example in settings where the labeling restrictions are less obvious. Even in our data from the VOC PASCAL challenge we encounter labels which are debatable. It depends on the question you ask, whether a human would classify an image as containing a person or not. Look for example at the images in figure 1.1. There is indeed a person on both images, but the pilot in the airplane is definitely less significant and also much harder to recognize.

1.1.3 Active Learning

We briefly discussed supervised and unsupervised learning, the former working with labeled data to train on and the latter with trying to learn something from unlabeled data. There is a mixture of the two which is called semi-supervised learning. The idea behind semi-supervised learning is that the system should use
strategies from unsupervised learning on top of a supervised learning setting. In most learning problems, labeled data is scarce and often difficult or expensive to obtain. Moreover, the labeling process is often tedious manual labor. Imagine a supervised learning system which has access to huge amounts of data, of which only a limited part is labeled. The system could train on the labeled data, then make predictions about the unlabeled data and take its own predictions to be true if it is really certain about them. With this newly (possibly incorrectly) labeled data, the system could retrain itself. This approach is shown to improve performance in many settings.

In a similar fashion, active learning also aims at adding unlabeled data to its train set, but instead of trusting its own predictions, it makes use of an oracle (e.g. a human annotator) to provide the correct labels for the data. In semi-supervised learning, the system makes assumptions about unlabeled data and therefore needs as level of certainty. In active learning however, the system focuses on finding samples from the unlabeled data of which it expects to learn most. The challenge is to find a way of predicting how useful a sample is for the learning process, which we refer to as the informativeness. Intuitively, the most informative samples are the ones about which it knows the least.

**Scenarios**

Active learning is an incremental process which aims to reach a certain performance level with as little training data as possible. There are different scenarios for applying active learning to a learning problem. Often the choice for any of these is based on the availability of data, the limitations in computational power and other properties of the setting.

The first scenario is called *membership query synthesis*, in which the algorithm may request labels for any unlabeled instance in the input space. This typically means the algorithm generates a query for the ideal instance de novo, rather than sampling from a given distribution of instances. Synthesizing queries like this may seem strange in the setting of this project, but it makes perfect sense if we would apply it to chemical experiments. The system can ask for the outcome of a specific experiment (e.g. “mix 3ml of fluid A with 8 ml of fluid...
CHAPTER 1. INTRODUCTION

B”), where the oracle (in this case someone who carries out the experiment) can then provide the system with feedback. This scenario is very useful for some settings, but can result in rather strange situations on others. This is mainly depending on the way the input space is defined.

Obviously this approach is not applicable to object recognition, because the oracle would have to label synthesized images which are likely to not contain any objects at all [3]. In our setting we want to focus only on existing images. The second scenario does exactly that. In Stream-based selective sampling the system takes an instance from the input distribution and decides whether or not it wants to have this instance labeled so it can be added to the train set. The key assumption behind this approach is that obtaining an unlabeled instance is free and only the labeling process is costly. This scenario might work rather well in object recognition, but the biggest risk is that it might take a very long time before we finally encounter an instance which might be useful. If we are training a model for recognizing a car, but our input distribution exists of tens of thousands of images with only a few with a car on them, then it might take a while before our selective sampling scenario is finally presented an image we can learn something from. The stream-based part of the name means we have no control over which instances are presented to us when.

To third and last scenario we will discuss is called pool-based sampling. To overcome the limitations of having instances presented one by one without control over the order in which they come in, we would like to have them available all at once, as a pool of data. In a stream-based setting this is often not possible due to technical limitations on availability of data. If such a pool is available, the task becomes picking instances from that pool based on some query strategy. Often using the complete distribution as a pool introduces new problems at the level of computational complexity, so only a certain (often normally sampled) subset of the distribution is used. The complexity arises from analyzing the pool. If we want to decide which instance in a pool seems most informative, we need to do certain calculations on all instances of the pool to be able to do so. In figure 1.2 we see a depiction of how this scenario typically works. We have a train set with labeled data on which we train a classifier. Then we use this classifier to decide which instance we would like to train on in order to improve the performance. This instance is then labeled by an oracle and added to the train set, after which the process starts over.

The most important question in any of the above described scenarios is obviously how to decide whether an instance is informative or not. How do we define informativeness? Extensive research has been done on this problem. We will briefly discuss a few approaches to querying instances from a pool, which we call query strategy frameworks. Some of them are particularly designed for a pool-based scenario, but others might also be applicable to stream-based sampling, or even in a membership query synthesis setting. Since our project is only concerned with single-class learning we focus only on those versions of strategies.
CHAPTER 1. INTRODUCTION

Figure 1.2: Depiction of a typical pool-based sampling active learning setting

Query Strategy Frameworks

When we think about informativeness of an instance, we think about how much we could potentially learn from it once we would know the correct label for that instance. In a binary learning problem we try to predict the probability that an instance belongs to a class, i.e. $p(y = 1|x)$. If our current model predicts this probability for a certain instance to be 0.5, it means the probability for not belonging to that class is just as high. In other words, the model has no clue. We can talk about the confidence of the system to be correct in its prediction and in this case that confidence is 0. Intuitively, getting a label for this instance would provide a lot of new ‘insight’ for the system, at least more than getting a label for which it is more certain about its prediction. This intuition lead to one of the most commonly used strategies called uncertainty sampling. More formally this can be defined as:

$$x_{us}^* = \arg\max_x 1 - P_\theta(\hat{y}|x)$$

where $\hat{y} = \arg\max_y P_\theta(y|x)$, $\theta$ is the current model, and $x^*$ refers to the most informative sample.

There are a lot of other strategies for picking instances from the pool. They differ greatly in their complexity (which often correlates with need for computing power) and their theoretic basis. Query-by-Committee (QBC) focuses on disagreement between multiple hypotheses, while expected model change, expected error reduction and variance reduction make predictions about what the addition of an instance would infer. Density-weighted methods look at the distribution of the data in an unsupervised learning fashion and include for example looking at clusters within the data. These methods are not used in this project and will therefore not be discussed in further detail, but the interested reader should refer to a great survey written by Burr Settles [4].

One specific research we would like to briefly discuss is a strategy which was introduced by Nguyen and Smeulders in 2004 [5]. Their approach uses
CHAPTER 1. INTRODUCTION

the concept of clustering (an unsupervised learning technique) to find dense areas in the data distribution. It then tries to find the best representative for each cluster. The clustering is adjusted using a coarse-to-fine strategy as the active learning adds more samples to the train set. The use of clusters can also prevent the repeated sampling from a certain group of similar data. A large disadvantage of this method, however, is the computational complexity. Furthermore, finding clusters in a dataset is not self-evident at all.

In Section 3.3 we explain how we consider density of data to be a useful property which we would like to take into account when choosing a sample from the pool. This is inspired on the above-mentioned research.

1.1.4 Practical Considerations

In Section 1.1.2 we explained the difference between noise and anomalies. In real-world settings it is often wise to expect at least some noise in the dataset. The distinction between anomalies and noise can then become quite hard and ambiguous, but it will still be very useful to at least be able to detect that something strange is going on in the data. For this project we will however, assume the data to be completely noise-free. The labeled images we use are provided by the PASCAL VOC challenge which we will discuss in Section 2.1.2.

Data instances are defined within the feature space by their values for the features. This space is \( n \)-dimensional, where \( n \) is the number of features. This dimensionality of the data is an important property, because it tends to have influence on the variance in the data. In our experiments we will use the gists of images (which we will explain in Section 2.1.1), which consist of 560 values. This means the data is 560-dimensional. Within such a high-dimensional feature space, data is often very sparse. The distribution of the data over the space can be an interesting aspect for finding anomalies or outliers.

Another aspect of data is the skewness, which refers to the relative amount of positive versus negative samples for a class. One of the pitfalls with skewed data is that sampling informative instances can become more complex. To fit a model, a classifier algorithm generally needs at least a few positive and a few negative samples. If there are very little positive samples, there is simply insufficient data to find parameters that correlate to the class.

A more technical consideration we have not yet discussed is that the computational complexity of active learning may defeat the purpose of the strategy. Think of calculations as something that costs time and money, just like the labeling process does. One could imagine a system becoming so complex that the effort of applying an active learning strategy becomes higher than the effort of simply labeling enough data to reach the desired level of performance. The main ‘cost’ of the iterative process of active learning comes from the updating or retraining of the model after an instance has been added to the train set. To overcome this problem, Hanoi et al. introduced batch mode active learning [6]. The idea is that instead of picking instances from the pool one by one, we take several at each iteration. It is less obvious then it may seem, because it is unclear how to assemble such a batch. In the case of uncertainty sampling we could simply take \( n \) instances with highest uncertainty, but this is probably not the most resourceful batch we can create. If we take a diverse group of instances...
(some with high uncertainty, some with low uncertainty, some with the focus on variance reduction, etc.) we probably get more joint informativeness out of a single batch. In this project we introduce a strategy which focuses on single instances which seem interesting because of their abnormality, so batch-mode sampling is not applicable to our method.

Finally, we will briefly discuss the challenges of balancing between exploration and exploitation. Once we introduce interactivity for a learning algorithm, we have give the system a say in what the next step should be. A goal in machine learning is to find an optimal model to be able to make correct predictions about new data. But the only reason we want to be able to do so, is because apparently there was a need for labeling data in the first place. We can separate these two goals and do them sequentially; learn a model first, then apply it to the setting you designed it for. In many real-world problems however, it would be nice to be able to do both at the same time. We call this online learning, because the model will be trained and used at the same time. This is very common in the field of reinforcement learning, which is often used in robotics. In such a setting the system learns by getting rewards or punishments for actions it takes. If the system does not try and new learn things it may be punished a lot for not knowing how to act. On the other hand, if it never uses the knowledge it has gathered, it wont get as much rewards as it could by acting on that knowledge.

In this project we don’t take this balance into consideration while designing the method. Our research focuses on ‘offline active learning’. However, it is important to be aware of this balance in case one were to apply our method in an online setting.

In section 4.1 we discuss in more detail how these considerations lead to the choices we made for our implementation.

1.2 Motivation

In the active learning strategies, the main challenge is to find data instances that will bring about an optimal model as quickly and efficiently as possible. This is often redefined as looking for a measure of informativeness. We propose a twofold strategy which switches between a basis strategy and a special ‘plug-in’ which takes over if it finds a misclassification in its train set. A misclassification should be considered as a point of interest as we will explain in Section 3.4. To the best of our knowledge, there is no previous work on this observation in an active learning setting. We will therefore explore the potentials and test our theory with an implementation to do experiments. Finally, we use Skinners first principle as more of a methodological motivation:

When you run onto something interesting, drop everything else and study it

In 1956, the American psychologist, behaviorist and social philosopher B.F. Skinner wrote an article titled “A case history in scientific method”. In this paper he studies the principles of performing research and how these can be taught to those who want to become scientist. The quote is above is what he
describes in this paper as the first principle to be not formally recognized by scientific methodologists up to that point.

1.3 Research Questions

The main goal of our work is to investigate the possibilities of improving existing active learning strategies by looking at anomalies or outliers in a dataset. To be more specific, we want to answer the following questions:

1. Are misclassifications in the train set useful for detecting informative data?

2. Can we show improvement on existing query strategies, like uncertainty sampling, by removing data from our train set?

We chose to restrict ourselves to these two main questions. The first is defined to focus on the sampling aspect of our approach and the second is to investigate the effects of removing data, in our case anomalies, from the train set.

1.4 Purpose and Contributions of this thesis

The main contribution of this thesis is the development of a new query strategy framework for active learning. Our approach is actually more like an additional ‘plug-in’ for any existing strategy.

Since we use data from an object detection challenge in this project, we will discuss the basics of computer vision in general, and more specifically the use of contextual information in object recognition in chapter 2.

A formal description of how we propose to use anomalies in an active learning strategy will be given in chapter 3. In chapter 4 we will briefly discuss the technical details of our implementation and our data set. Several experiments will be set out and their results will be analyzed and discussed.

Finally, we will provide answers to the research questions as stated in the previous section, based on our findings and experimental results in chapter 5. We will discuss these answers and make some suggestions for future work on the subject.
Chapter 2

Background & Literature

Review

In our project we focus on active learning, which we apply to a computer vision setting. In this chapter we will discuss some of the aspects of computer vision that are relevant to our research. In particular, we will focus on a project by Divvala et al. which makes use of contextual information to improve object recognition, since we rely heavily on their research.

2.1 Computer Vision

Vision enables us to recognize people, animals, and inanimate objects reliably. In Computer Vision we refer to this process as object recognition. This includes determining the class of particular objects that have been imaged, as well as recognizing specific objects. If we do so in still images or photographs, we can only use the visual data that is available to us. In current technology this visual data is almost always represented by a grid of pixels, each with their own color, often represented by 3 values describing the intensity of the colors red, green and blue (RGB-values).

By looking at this visual data we can focus either on detecting whether an object is present (object detection), or include with this detection the goal of localizing the object within the image (object localization). Many research has been done on both, where the latter obviously yields a bigger challenge.

Generally there are two main philosophies in machine learning: generative and discriminative algorithms. The former is concerned with learning a model of the joint probability of input (an image) and a label \( p(x,y) \). Using Bayes’ rule we could then derive \( p(y|x) \). In contrast, the latter, a discriminative model learns directly the posterior \( p(y|x) \) and tries to map from input to labels: \( y = f(x) \). In object recognition we can state that, more informally, generative models can be thought of as trying to predict the probability of finding object \( o \) in image \( i \) by comparing that image to many other images that either do or do not have \( o \) on them. Discriminative models try to make this prediction by looking for preliminaries for detecting object \( o \), based on what can be learned from other images with \( o \) on it.
Extensive research has been done on both types of models. The most important challenge is to find a representation for the visual data. Model-based approaches try to represent an object as a collection of three dimensional geometrical primitives (boxes, spheres, etc.), whereas appearance-based models focus mainly on two dimensional views of the object. Calculations can be done on the visual data to find certain local or global features in the image (edge detection, SIFT features, etc.). These can then be used to train a model of the object on.

What most approaches don’t take into consideration is the importance of context (at least not explicitly). Say we have established that a detected object belongs to the class ‘vehicle’, but we are not sure whether it is a bus, a plane or a boat. If we would be able to correctly determine the surroundings of the object, we could probably easily find the right answer. Based on finding either a road, air or water, we would instantly know the right class. This is just one example of how context might help in the field of object recognition.

2.1.1 Use of context

Context plays an important role in how humans interpret images, and more specifically in how humans detect objects in images [7]. To be able to make use of such contextual information in computer vision, we need a way to define in more detail what we mean by context and find a way to incorporate it in existing object recognition techniques.

Types of Context

A very useful distinguish between several types of contextual information has been proposed by Divvala et al [8]. We will discuss these in more detail in Section 2.2. Some of the contexts they use need external sources to be computed, while others rely completely on visual data from the image itself. We will now focus on what is called the 2D Scene Gist Context in a little more detail, since this is this the type of context we use to predict object presence in our project.

Gist

As mentioned before, biological and psychological research has shown that humans make great use of contextual information when interpreting an image. One way of getting a quick overview of an image and determining the scene, is looking at what is called the “gist” of an image. For example, even if an image is only presented for a fraction of a second, humans show the ability to report that they were looking at an indoor kitchen scene with numerous colorful objects on the counter top. These high level observations can be of huge importance in priming the further object recognition process.

There are several approaches to implementing a representation for a gist in computer vision. In this research we use an approach developed by Oliva and Torralba [9] because it is the version used in previous work by Divvala et al., which we used as a starting point for our project. This choice can be further motivated by looking at a comparison of four different gist models in which the
one we use performed best in terms of overall error rate on a one-shot scene classification task [10].

2.1.2 Challenges

To stimulate researchers around the world to engage in the field of object recognition, a big annual challenge is organized since 2005 called the PASCAL VOC Challenge [11]. VOC stands for Visual Object Classes. Nowadays the challenge exists of multiple competitions among which a classification and a detection competition, as well as a segmentation and an action classification challenge. The organization provides competitors with a large labeled train set.

This challenge has functioned well as a tool for comparing the current state-of-the-art approaches within these competitions. Again, since our project is based on the research by Divvala et al., this is the initial reason for choosing to work with the PASCAL VOC data. But just like with the choice for the gist model, this choice of data makes perfect sense in the broader prospect of our research and its purpose.

2.2 Context in Image Understanding

As mentioned before in section 2.1 this project relies greatly on the framework and methodology of Divvala et al [8]. However, this is definitely not the first time someone tried to use context to improve performance of object recognition. In the early nineties Strat and Fischler talked about Context-based vision, mainly focusing on scenes one could encounter in nature outdoors. Instead of looking at individual objects, they emphasized the importance of taking into account the whole scene made up by those objects [12].

In 2003 Murphy, Torralba and Freeman published a paper on how to use a mixture of global and local features in an image to solve the task of object detection and scene recognition, with the beautiful title ‘Using the forest to see the trees’ [13]. They got their inspiration from the research that had been done on the topic in the field of psychology. Mandler and Johnson, with another great paper title ‘Some of the thousand words a picture is worth’, described how only certain settings could be recognized as a scene [14]. In contrast, putting objects together in an unrelated way, even though they may be semantically correlated (e.g. kitchen tools), did not trigger the same recognition capabilities in subjects.

In 2009 Divvala et al published a paper with the intent of comparing the potential of several uses of context. Up till then, most approaches for using context were tested on home-grown data. Moreover, the focus was always on one context versus none. Instead, they wanted to investigate applying more than one context in a single system. In Table 2.1 we see a description of the types of contexts they appointed. Some of these need external sources to be computed, while others rely completely on visual data from the image itself. Local pixel context is the most simple type of context which simply incorporates a little border of pixels around an object into the recognition process. Temporal and especially cultural context are still very broad and it is unclear how to effectively use them in general.
Apart from the use of multiple contexts, Divvala et al wanted to run experiments on data that would provide some way of comparing their performance to other state-of-the-art computer vision systems. They chose the PASCAL VOC Challenge for this purpose. As mentioned before in Section 2.1.2, this challenge has become a ‘standard’ in the field over the past couple of years, with a large number of competitors. Among the top-performing approaches of 2008, UoCTTI, developed by Felzenszwalb, McAllester and Ramanan, was the only one publically available [15]. Divalla et al used this local detector as the basis to which they would apply their contextual improvements. They re-score the output of this detector (1000 bounding boxes per image) based on several classifiers which predict the likelihood of these bounding boxes based on a context. In Figure 2.1 we see a depiction of their architecture. We will now discuss the components in a little more detail.

The main idea behind the system is that every context gets its own classifier, producing a probability for object presence in an image $P(o|I)$ according to that context. For some of these contexts it is not just the image itself, but also the location of the object on the image. It is important to keep in mind that this architecture is for binary classification, so we look at only 1 object class at a time. All these classifiers can be trained independently and only when used for classifying we need to combine their predictions.

At the top we see the UoCTTI classifier, this is the previously mentioned local detector which provides a thousand bounding boxes per image. The blue lines represent this output and show which other components take the bounding
Figure 2.1: Architecture of the system as developed by Divvala et al.
The gist is always based on an entire image, so it only gets the image as input (no need for the bounding boxes). Based on the calculated gist, as described in Chapter 1, it will make a prediction for finding an object of the specific class on this image.

The next two components we can see that external data is used. In the PASCAL VOC Challenge this is not allowed, so these components could not be used if they were to compete in the real challenge. But for the purpose of research it is very interesting to see what can be achieved with this external contextual data.

The geographic context is based on a system called im2gps by Hays and Efros in 2008 [16]. They estimate geographic properties for a novel image by finding matching scenes within a database of approximately 6 million geotagged Flickr photographs. Based on the estimated GPS-location of where the novel photo may have been taken, Divvala et al compute 15 geographic properties such as land cover probability (e.g. forest, cropland, barren or savanna), vegetation density, light pollution, and elevation gradient magnitude. These properties correlate somewhat to object appearance (e.g. a boat is often found in water scenes and a person is more probable in areas with high population density). It should also be noted that this type of context was not very well suited for dealing with the indoor classes in the VOC dataset.

The same Flickr data was also used for the semantic context. The 500 most popular words appearing in the tags and titles of these photos were manually divided into categories corresponding to the 20 VOC classes and 30 additional categories. For instance, bottle, beer and wine all fall into one category, while church, cathedral and temple fall into another. For a novel image they build a histogram of the keyword categories that appear among the 80 nearest neighbor scenes. A classifier is then trained to predict the appearance of an object given this histogram.

The last three contexts, the location context, the height context and the spatial support context, are very complex and of fewer interest to this project and will therefore not be discussed in further detail. In short, they use the location of an object (based on a 5 x 5 grid) and the height of the bounding box to compute a probability for the object belonging to a certain class. Spatial support is best explained by an example. With very complex computations they try to predict the probability of finding a person above a horse. This is obviously an example which could definitely occur and may be helpful in the distinction between the lower object being either a horse or a dog.

The results of all the classifiers are being combined into a single probability of the object occurring at the bounding box \( P(o|BB) \). They each have their own weight in this final probability. Then the 1000 bounding boxes will be re-scored based on this weighted probability. Divalla et al showed impressive improvements on the original UoCTTI performance with their use of contexts.

A completely different way of tackling the limitations of using only visual data, is by trying to translate some of the knowledge about the world to the recognition process. This knowledge is often only available as text and lacks sufficient structure to be applied in a straightforward fashion. Successful tex-
tual attempts like Wordnet however, have encouraged people to try and build ontologies for vision purposes as well. Many research on this topic has been done by Li Fei-Fei [17–21].
Chapter 3

Method

In this chapter we look in more detail at the proposed approach. We will first go over the terminology and give formal definitions of the terms we use. In section 3.2 we introduce several new query strategies and compare them to uncertainty sampling. After that we will look at how we can use interesting observations on our data to improve existing strategies or even formulate a new one in section 3.4 and 3.3. And finally in section 3.5 we explain how we combine these concepts into a single approach for active learning.

3.1 Terminology

In this section we will concisely explain the terms that are used in this chapter. Most of the notations will be conform the standard in the field of research, but it is important to define clearly what is represented how. The definitions stated in this section are specific to this project and for example partly determined by our focus on binary learning problems and the use of logistic regression as our learning algorithm.

We use the notation \( x \) for an instance in our data, which may sometimes also be referred to as a sample, example or (data) point. Every \( x \) is made up of values for the features we use to train on. \( x = \langle x_1, \ldots, x_n \rangle \) where \( n \) is the number of features and every \( x_i \) is a value for one such feature. Data instances can have labels, which in our case are binary. The label represents whether an instances belongs to a certain class, where 1 means it does belong that class, and 0 means it does not. For a label we use the notation of \( y \). \( x \)'s and \( y \)'s can have subscripts and superscript to denote a certain indexing or belonging to a certain group or set.

A hypothesis \( h \) is a mapping function. When applied to an input instance \( x \), it will give an output in the desired output space. To do so it needs a model \( \theta \). These two terms are often intertwined. Training a model means finding the right values for \( \theta \) which consists of \( \langle \theta_1, \ldots, \theta_n \rangle \) where \( n \) is the size of the model (which is not necessarily the same size as the data instances although in our case it is). Every \( \theta_i \) is called a parameter (or also sometimes weight). Choosing an hypothesis \( h \) from a set of hypotheses \( H \) (or the hypothesis space \( H \)) is another way of looking at the this problem, but obviously choosing from a pre-made set
is different from actually fitting the parameters of the model.

Because we use logistic regression, the output of $h_\theta(x)$ will always be in the range $\{0, 1\}$. This also means we can interpret this output as the probability of $x$ belonging to a class $P(y = 1|x; \theta)$. If $h_\theta(x) = P(y = 1|x; \theta) \geq 0.5$, then we predict $y = 1$ and if $h_\theta(x) = P(y = 1|x; \theta) < 0.5$ then we predict $y = 0$. This is what we refer to if we use the term *prediction*, for example “the system’s prediction for $x$”.

When we talk about the *certainty* of a prediction, we talk about how close the probability is to 0 or 1. More formally, we can define certainty as $2 \times |h_\theta(x) - 0.5|$, where the factor 2 is only to make the outcome in the range $[0, 1]$.

A *decision boundary* is an interpretation of the model $\theta$ which separates instances with label 1 from instances with label 0. In a linear model such a boundary is a line or hyperplane (depending on the number of dimensions) in the feature space. Certainty can now also be explained as the distance from an instance to the decision boundary. Lying close to the boundary means low certainty, further away means high certainty.

Finally we define a *train set* to be a set of instances with a label (thus pairs of the form $⟨x, y⟩$). The test set has the same properties but serves a different purpose. These data instances will be used to test the model on. We need the labels to check whether the prediction are correct. Then we also have a *pool* which consists of only instances and no labels.

In section 1.1.3 we talked about *query strategy frameworks* (or simply *query strategies*). With this term we refer to a strategy that is used for choosing an instance from the pool. This instance will then get its appropriate label from an *oracle*. The instance that is chosen by the strategy will be denoted as $x^∗$.

## 3.2 Query strategies

We have already discussed the wild variety of strategies that have been developed in section 1.1.3. One of the most intuitive and most commonly used strategies is *uncertainty sampling* which we also briefly touched upon in that same section. We will here explain in further detail how this strategy works and show how we have developed two new strategies in the same fashion.

### 3.2.1 Uncertainty Sampling

The idea behind uncertainty sampling is best explained by analyzing a simple example. In figure 3.1 we see two groups of instances in a 2 dimensional feature space. The instances of the lower left group have the label 0, and the others have label 1, but since the system does not know the labels yet, they are depicted as gray dots. We start with a train set consisting of 2 instances, 1 positive (label 1) blue instance and 1 negative (label 0) red instance. The initial model is trained and the line represents the *decision boundary*. Uncertainty sampling looks for the instance in the pool about which it is least certain, which corresponds to the point nearest to the decision boundary. This point $x^*$ is the point with a square around it.

The key idea behind uncertainty sampling is that, intuitively, if we know little about an instance, then getting a label for that instance will give us a
CHAPTER 3. METHOD

21

(a) Initial train set and $x^*$ (b) After 50 iterations

Figure 3.1: Visual example of how uncertainty sampling works

lot of information about the data in general. Uncertainty corresponds to high informativeness. By taking points close to the decision boundary we expect that boundary to become optimal as quickly as possible. Surely, taking such a point will have great influence on the boundary after it has been added to the train set. And after a few iterations of great influence we expect convergence to occur quickly.

Based on our definition for certainty, we can define uncertainty sampling as:

$$x^*_{\text{US}} = \arg\min_{x \in P} |P(y = 1|x) - 0.5|$$

3.2.2 Certainty Sampling

As the complete opposite of uncertainty sampling, we introduce a strategy that is inspired by semi-supervised learning. Instead of looking for instances about which we are uncertain, we look for instances with a probability very close to 0 or 1, i.e. a very high certainty. This may seem very counter intuitive in correspondence to what we explained in the previous section on the influence of points on the shifting of the boundary. However, this can very well be considered as an exploration strategy, because instead of trying to further refine the boundary we will now take a look at ‘the other end’ of the class. In figure 3.1 this would imply taking the point furthest away from the boundary. In this 2-dimensional simple case with two well separated groups of instances this may not make a lot of sense, but in many real-world learning problems the feature space is much more complex in the sense that is has more dimensions and the data is probably less separable.

Another specific reason why we want to explore the applicability of certainty sampling is that is more likely to trigger what we define as ‘something interesting’ in section 3.4. If the prediction for an instance has a high certainty but the oracle gives us the unexpected label, then it is probably worth looking into that instance (and maybe its surroundings). We will elaborate on this theory in the previously mentioned section.

We define certainty sampling as:

$$x^*_{\text{CS}} = \arg\max_{x \in P} |P(y = 1|x) - 0.5|$$
CHAPTER 3. METHOD

22

Notice the subtle difference of taking the argmax (the $x$ with the highest value) instead of argmin in uncertainty sampling.

3.2.3 Positive & Negative Sampling

In a similar fashion as looking for instances for which we are very certain or uncertain about our prediction, we could also be interested in instances with a specific label, i.e. either positive or negative examples. The percentage of a data set that belongs to a class may be very limited. In our case, if we look at objects on images, there are only a few images with a ‘sheep’ on them, while there is a ‘person’ on nearly half of them.

Particularly in cases with very limited positive examples it seems like a useful strategy to eagerly put those in our train set. So we define positive sampling as a new strategy which looks for instances about which we expect highest probability of getting a positive label ($y = 1$) from the oracle. This is not a trivial challenge however, since we are technically looking for the best representative of the class (e.g. the best picture of a sheep).

For the sake of brevity and because it is unclear how useful this strategy will be, we will take the most simple approach for picking the positive sample. Technically it is a specific version of certainty sampling where we look at only one subclass of certain samples, namely the ones for which we predict $y = 1$. So for positive sampling we define:

$$x_{PS}^* = \operatorname{argmax}_{x \in P} P(y = 1|x)$$

As with uncertainty and certainty sampling, here we could also have a look at the complete opposite, meaning we would focus on finding the image which best represent the negative samples (e.g. the best picture to not represent a sheep). This seems very counter intuitive, but we think it is none the less interesting to see what this strategy will yield in experiments. Negative sampling can thus be defined as:

$$x_{NS}^* = \operatorname{argmax}_{x \in P} P(y = 0|x) = 1 - P(y = 1|x)$$

3.2.4 Mode & Mean Sampling

Finally we want to look at using the distribution of the predictions for the instances in the pool to base our choice for $x^*$ on. In figure 3.2 we see an example of how these predictions might be distributed. This is a histogram for the values for $P(y = 1|x)$ for 1000 instances. Apparently there are more than 30 instances for which the prediction lies between 0.12 and 0.13, but more importantly we can see the overall distribution, which we will from now on refer to as the pool distribution. In a more abstract sense we can say that the goal of a learning algorithm could be redefined as getting high values near 0 and 1 and low values in the middle. Figure 3.3 shows us abstract forms of what the pool distribution may look like (a) at the beginning of a train process and (b) what we want it to become. Obviously we could make the same histograms for the predictions over the train set or the test set, but we are now concerned with picking $x^*$ from the pool and therefore focus on the pool distribution.

The reason to look at these distributions is that, instead of simply taking the instance with the most uncertain prediction, we want to take an instance from
the peak of the histogram. If we look closely at the our definition of uncertainty sampling, we see that it correspond to taking the instance with a $P(y = 1|x)$ closest to 0.5. It is very easy to implement a different value for this strategy, so we propose two new strategies with a dynamic sampling value. The first focuses on the ‘peak’ of a distribution, i.e. the mode. We calculate this by dividing the range $[0,1]$ into 100 bins and count the number of instances that for which the $P(y = 1|x)$ falls into those bins. The second dynamic value we use is simply the mean of the distribution. The mode is more likely to radically change per iteration, while the mean is expected to shift more gradually.

For both strategies the formalization look like:

$$x_{MS}^* = \arg\max_{x \in P} |P(y = 1|x) - \alpha|$$

Where $\alpha$ is the value for which we enter either the mode or the mean of the current pool distribution.

### 3.3 Useful Data Properties

It is important to note that the above mentioned strategies have different effects and therefore differ in applicability to numerous circumstances. These circumstances are primarily dependent on the properties of the data (number of features, skewed distribution, density, etc.).

In an attempt to introduce a new strategy which is less sensitive to these properties the data, we have to define what makes an instance valuable in an active learning setting. Which properties of an instance can we use to calculate an informativeness score?
CHAPTER 3. METHOD

3.3.1 Variables

We define four variables that we think play an important role in finding $x^*$. For every data-point in the feature space we define:

- **density**: The amount of samples located within a certain distance $d$ of $x$.

- **certainty**: The certainty of the prediction $P(y = 1|\mathbf{x})$ from the classifier, defined as $2 \times |P(y = 1|\mathbf{x}) - 0.5|$. This means that the certainty is close to 1 if the prediction is close to 0 or close to 1, and that the certainty is close to 0 if the prediction is close to 0.5.

- **distance**: As a distance measure between two points $\mathbf{x}_i$ and $\mathbf{x}_j$ is we simply use the euclidean distance defined as $|\mathbf{x}_i - \mathbf{x}_j|$.

- **correctness**: The correctness of the prediction $P(y = 1|\mathbf{x})$ from the classifier is defined as $1 - |P(y = 1|\mathbf{x}) - y|$. This means that the correctness is close to 1 if the absolute difference between the predicted label and the label is close to 0, and vice versa.

Note that for the last variable correctness we need the ground truth label to be able to compute this. With these variables we want to define rules about which points should be valuable to the learning algorithm. These rules should then lead to the formulation of a score for each point in the pool.

Probably the most important notion is the following: Assume we have an unlabeled instance $\mathbf{x}_P$ from the pool which is close to a labeled instance $\mathbf{x}_T$ from the train set. If the classifier is certain about its prediction for $\mathbf{x}_T$, but is in fact wrong, then $\mathbf{x}_P$ should be considered as an interesting instance. Especially when both instances are in a dense area, and even more so when the classifier also has a high certainty about the unlabeled point. Apparently the labeled point to which the unlabeled point lies closely is messing up the classifier, so it is important to investigate into other points in this neighborhood. We will elaborate on this in the next section.
On a more abstract level we have looked at all the possible combinations of the values for the variables being either low or high (e.g. ‘dense’ or ‘not dense’, ‘certain’ or ‘uncertain’ etc.) and tried to roughly estimate whether or not such a combination would make an instance informative. This lead to the observations in table 3.1.

<table>
<thead>
<tr>
<th>Certain</th>
<th>Near labeled</th>
<th>Incorrect</th>
<th>Not near labeled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense</td>
<td>-</td>
<td>+++</td>
<td>+</td>
</tr>
<tr>
<td>Not dense</td>
<td>-</td>
<td>x</td>
<td>-</td>
</tr>
<tr>
<td>Uncertain</td>
<td>Dense</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Not dense</td>
<td>+</td>
<td>x</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.1: Importance of a sample based on the four variables

Now we are faced with the challenge of combining the variables for an instance into a single formula which should provide us with an informativeness score. It could for example look somewhat like:

\[
\text{score}(x) = \alpha \text{certainty}(x) \times \beta \text{density}(x) \times \gamma \left( \frac{\sum_{x_i \in T} \text{distance}(x, x_i) \times 1 - \text{correctness}(x_i)}{\sum_{x_i \in T} \text{distance}(x, x_i)} \right)
\]

The last term in this equation refers to the number of misclassified labeled points near \( x \), divided by the total number of labeled points near \( x \). The intuition is that if this number is high, then the unlabeled point \( x \) is of high interest.

It should become clear that unfortunately this approach will not lead to any useful metric. First and far most it is completely unclear how these variables relate to each other and what their influence should be on the final score. In the formula above we take the product of the terms, but we may as well sum them. How to choose values for the weights (\( \alpha, \beta \) and \( \gamma \)) is completely ambiguous. And why don’t we include, while we are at it, some measure for diversity of a point compared to other points in the train set?

To summarize, we can state that this way of trying to define a score so explicitly is not the right way to go. There is however one insight we came across which might be useful and implementable after all and which we will discuss in the next section.

3.4 “Something interesting”

“When you run onto something interesting, drop everything else and study it.” Inspired by this first principle for scientific methodology as stated by Skinner in 1956 [22], we introduce a ‘plug-in’ on top of the previously discussed strategies. Our framework is twofold in the sense that we we apply a certain strategy as our basis, but we ‘drop everything’ and switch to a special algorithm if case we
‘run onto something interesting’. In an active learning setting we define two situations to be potentially interesting:

1. a misclassification in the train set with high certainty
2. receiving an unexpected label from the oracle for $x^*$

These situations can be used as a trigger for our special plug-in algorithm. In this section we will elaborate on when we might encounter them and then our approach of dealing with these situations will be explained in section 3.5.

### 3.4.1 Misclassification in the train set

To find a misclassification in the train set $T$, we have to apply our model to this set. This means we will get predictions $P(y = 1|x; \theta)$ for every $x \in T$. If the certainty of $x$ is high, but the prediction turns out to be wrong none the less, then we can state that it is probably worth investigating what is going on with that sample (and possibly the space around it). More formally we state:

$$\forall x \in T | h_\theta(x) - y | > \zeta \implies M \leftarrow x$$

Where $T$ is the train set, $\zeta$ is a threshold parameter for the certainty and $M$ is the set of all misclassifications in $T$. Since $h_\theta(x) \in [0,1]$ and $y \in [0,1]$, the difference between them is also in $0,1$. In this case 0 means the prediction is completely correct and 1 means completely incorrect. The threshold $\zeta$ provides the opportunity to set the ‘strength’ of the misclassification. As described in section 1.1.1, $h_\theta(x)$ can be interpreted as $P(y = 1|x; \theta)$ and therefore setting $\zeta = 0.5$ corresponds to what is called the misclassification error. From now on we will assume $\zeta = 0.5$ unless stated differently. A misclassified instance will be denoted as $\hat{x}$.

So let’s have a look at the possible situations of encountering a misclassified sample in our train set. We start with very few samples in our train set. If there is a consistent hypothesis we can assume that our classification algorithm will find it. This means that if we have a misclassification, either:

1. The classification algorithm is unable to learn a correct model for this data because of limitations in the algorithm or hypothesis representation.
2. The sample is an anomaly and is therefore hard to classify correctly.
3. The sample is incorrectly labeled and therefor, technically, the classification is actually correct, but we don’t know this.

In this project the last option can be discarded since we assume perfect labeling. However, in many real-world problems there is reason to expect at least some noise in the data, as discussed in section 1.1.2. It is important to be aware of this fact when applying our approach to other data sets.

The first option is probably the most interesting, but also the most difficult problem to solve. Say we are trying to fit a linear model to our data. If the separation between positive and negative examples turns out to be non-linear, we will keep misclassifying samples in our train set. A simple example of such a situation is shown in figure 3.4. However, in our iterative way of learning, how
Figure 3.4: Example of data set for which there is no consistent linear model

can we figure out that this is indeed the case? And if we are able to decide that this is the situation at hand, then how should we deal with it?

To answer the first question: we probably can’t. At least not in an automated fashion. There are so many aspects of the data which can cause this problem, that there is no simple way of finding out without investigating manually on these aspects. The best option we have to find out if we are dealing with this situation is by investigating the area around or misclassified point \( \hat{x} \). In section 3.5.1

The second option for encountering a misclassification is when we are dealing with an anomaly in our data. In section 1.1.2 we explained what an anomaly is and why it may be interesting. In our approach we need to find a way to deal with these instances. In the object recognition case we are concerned with, best way to think of an anomaly is ‘a strange example of an object’. For example, we have the class ‘couch’ and on a lot of images we can clearly see the full object. However, there are some images where there are people on the couch, sometimes so many people that the couch itself is hardly visible. When such an instance is in our train set, it may have an unwanted effect on the decision boundary. Again, the best option we have for figuring out if our misclassified instance \( \hat{x} \) is indeed such an anomaly, is to look at surrounding images, i.e. images that are very similar to \( \hat{x} \) and therefore close to \( \hat{x} \) in the feature space.

3.4.2 Unexpected label from the oracle

The other situation in which we may have run onto something interesting is when we get a label from the oracle which was unexpected, based on the prediction of our current model. In this case we have already picked \( x^* \) based on some strategy \( s \). We would have to look at the certainty of the prediction for \( x^* \) to
see how ‘unexpected’ the ground truth label actually is. For example, if the prediction $P(y = 1|x^*) = 0.498$ and the true label turns out to be $y = 1$, this can hardly be considered as unexpected. We can now use the previously defined parameter $\zeta$ to define the minimal prediction error before considering a label to be unexpected. Intuitively this value could be set to $\zeta = 0.8$, but it is best to leave this open and find a useful value through experiments.

An important observation is that the strategies as discussed in section 3.2 all have a different probability of getting an unexpected label for their $x^*$. If we take $\zeta = 0.8$ then the prediction for $x^*$ has to be at least below 0.2, or above 0.8. This means that certainty sampling has a much higher chance of incurring this situation than uncertainty sampling.

But even more important is the question of what we do in a situation where we get an unexpected label. Because the answer is that we don’t want to act on it at all. We should probably be happy that our strategy chose this particular $x^*$ for it is likely to be very informative. But apart from that we cannot really use this information in a straightforward fashion. So in the next section, where we explain our approach of dealing with the potentially interesting situations, we only use the misclassification of a sample in our train set as a trigger.

3.5 Approach

Now that we have defined the trigger for our approach, we can look at how we want to handle the situation of finding a misclassification in the train set.

3.5.1 Nearest Neighbors

As mentioned before, we will look at points around $\hat{x}$ in our pool and pick those to be labeled by the oracle and added to the train set.

There is however a big risk in doing so. Let’s say we are dealing with a situation in which the misclassified sample is an anomaly. If we add nearby points, they probably have the opposite label of our misclassified point. The model will therefore become more certain about this area and will keep misclassifying our point, even with increasing certainty.

To end this process of becoming ‘more incorrect’ about this point, we have to come up with a stopping condition. We decide that at some point, enough points in the area surrounding the misclassified point $\hat{x}$ have been added to the train set. This could be done by counting the number of points within some distance $d$, but it is unclear how to pick a value for $d$. Another option is to look at the distance of the nearest neighbor to $\hat{x}$ in the train set and in the pool. Let $x^{NN}_P$ denote the nearest neighbor of $\hat{x}$ in the pool, and $x^{NN}_T$ the nearest neighbor of $\hat{x}$ in the train set. We can now calculate the euclidean distances from $\hat{x}$ to those points and compare them. In the most simple case we could say that if there is a point in the pool nearer to $\hat{x}$ then there is in the train set, then we want our strategy to pick that point.

$$x^* = x^{NN}_P \iff \|x - x^{NN}_P\| \leq \|x - x^{NN}_T\|$$
CHAPTER 3. METHOD

Figure 3.5: Comparison of the effect of the stopping condition

This fairly strict stopping condition implies that we only look at one nearest neighbor for a misclassified point and then disregard it in subsequent iterations. We can easily relax this rule by looking not at the first nearest neighbor, but the $k$th nearest neighbor in the train set.

In figure 3.5 we see an example of a $\hat{x}$ with points $p_1, \ldots, p_4$ from the pool and points $t_1$ and $t_2$ from the train set. If we use the strict stopping condition, we take $p_1$ in our first iteration (because it is closer to $\hat{x}$ than the nearest point from the train set $t_1$). In the next iteration we would stop, because there is no point in the pool within the red circle. However, if we were to take the $k$th nearest neighbor from the train set as our stopping condition, and set $k = 4$, we could add $p_1$, $p_2$ and $p_3$ before we stop. After the first iteration $p_1$ becomes the nearest neighbor in the train set, but the fourth nearest neighbor is still not even in the image. Once $p_2$ is added, $t_2$ becomes the fourth nearest neighbor, and even then $p_3$ is still closer. So by increasing $k$ we can relax our stopping condition.

3.5.2 The Algorithm

Let us now formulate the approach in pseudo code. In algorithm 3.1 we see two things not previously discussed. Our system works with a budget $b$ which is used as a stopping condition for the overall process. In active learning it is common to allow a system a certain amount of time, computation power or number of iterations which can be defined as the cost. Once that cost reaches the budget, the system delivers its final model. In section 1.1.4 we briefly discussed the issue of comparing performance of an active learning system to its non-active supervised learning version. The main goal is to use fewer labeled data samples to reach the same performance level, but if the calculations take much more time or the system requires much more computational power, then sometimes this time or power could have been spent on labeling more data instead. Active learning may thereby defeat its own purpose, so using a budget is an easy and safe way to limit this risk.
The other technicality not mentioned before is the sorting of all $\hat{x}$’s in $M$ based on their certainty before looping over them. The system can only pick one $x^*$, so if we have more misclassified instances, we are most interested in the ones with a higher certainty. The algorithm will therefore explore the misclassified instances in that order and break out of the loop as soon as a suitable $x^*$ has been found.

If the $M$ is empty, i.e. there are no misclassifications in the train set, then we apply one of the standard query strategies as discussed in section 3.2.

Algorithm 3.1 Overall procedure

**Input:** a train set with labeled data $T$, a pool with unlabeled data $P$

**Output:** final classifier $\theta^*$

**Given:** a budget $b$, stopping condition parameter $k$, a basis strategy $s$

1: **Initialize:**
   - Train initial classifier model $\theta$ on $T$
   - $\text{cost} \leftarrow 0$

2: **repeat**

3: $M \leftarrow \emptyset$

4: $x^* \leftarrow \emptyset$

5: for each sample $x \in T$ do $\triangleright$ look for misclassification in train set

6: if $h_\theta(x) \neq y$ then

7: add $x$ to $M$ $\triangleright$ create set of misclassifications $M$

8: end if

9: end for

10: if $M \neq \emptyset$ then

11: Sort $M$ based on certainty (descending)

12: for each $x \in M$ do

13: Select nearest neighbor $x^{NN}_P \in P$

14: Select $k$th nearest neighbor $x^{NN}_T \in T$

15: if $||x - x^{NN}_P|| \leq ||x - x^{NN}_T||$ then

16: $x^* \leftarrow x^{NN}_P$

17: break

18: end if

19: end for

20: if $x^* = \emptyset$ then

21: Select $x^* \in P$ based on strategy $s$ $\triangleright$ See section 3.2

22: end if

23: Get label for $x^*$ from Oracle

24: Add $x^*$ with label to $T$

25: Update/Retrain classifier model $\theta$

26: $\text{cost} \leftarrow \text{cost} + 1$

27: until $\text{cost} \geq \text{budget}$

28: Return $\theta^* \leftarrow \theta$
3.5.3 How to deal with ‘something interesting’?

The big question remains what we want to do with the misclassified instances that stay misclassified even after investigating their surroundings. We have identified that they may have a ‘negative’ influence on the decision boundary, so the best option seems to remove them from our train set. We will briefly explore three scenarios and their possible solutions.

Scenario 1: a single anomaly

We have an instance in our data which a true anomaly (without any similar ‘anomalies’ around it). It is a positive example \((y = 1)\) but lies in an area surrounded by negative examples. It will at some point be added to the train set and detected as a misclassification. After we have reached our stopping condition for looking into this instance, it will just stay there and becomes of less influence with every ‘normal’ sample that will be added to the train set.

In this case the best solution seems to just remove the misclassified instance from the data. It should also be noted that there is only a limited chance of the instance being sampled by a basis strategy in the first place.

Scenario 2: a group of anomalies

There is a (relatively small) group of positive examples in the data which is located far from the rest of the positive data, but close to each other. Let’s assume we have a 3 dimensional space and about a 100 positive samples lay close to the origin as a group. Then there is the small group of about 20 samples which is located, also as a group, straight above the origin at some substantial distance from the other group. There are negative samples throughout the rest of the feature space, but a linear model can be found for the complete data set. At some point the train set consists quite a few positive samples from the first group, none of the second and only a few negative samples. The basis strategy picks a sample from the small positive group, but because of the way the other instance in the train set are distributed, this new point gets misclassified after the model is retrained. In the next iteration it is detected as misclassified and a neighbor will be added. Since this neighbor belongs to the same small group of positive examples, the model will be retrained with a ‘stronger’ influence from this area. After retraining, the instances are correctly classified.

This is a perfect example of how our approach is supposed to work. Two observations should be made about this scenario. First of all, the chance of encountering this situation is again very small. In most cases, if there is a linear model which would correctly classify the train set, the classifier will find this model. The model is fitted by minimizing a cost function over the parameters, so the goal of the classifier is to be as correct as possible about all instances in the train set at the same time. This means that it may sometimes happen that one or two instances by themselves get misclassified. Also, having an instance from the small group picked by the basis strategy is not at all probable per se.

Second, the way the data is distributed in this scenario is very specific. It is difficult, if not impossible, to make assumptions about how often this occurs in real-world learning problems.
Scenario 3: a non-linear problem

Similar to scenario 2, we have positive samples divided over two groups, but this time there exists no linear model that will correctly classify the data set. In this case the system will encounter misclassifications repeatedly and the decision boundary will change drastically in the first few iterations and eventually converge to some ‘optimal’ in which it classifies as many instances in the train set as possible correctly.

As stated in section 3.4.1, we are probably not able to detect what is going on here. There are two solutions to this scenario, but we don’t know which one we should apply. The first solution is to divide the positive class into two separate classes. The system should now train two separate classifiers, one for each subclass. It is very well possible that for these subclasses the is a linear consistent model.

The second solution would obviously be to try to fit a non-linear model, either by choosing a different classifier algorithm or by changing the feature space.

What is left

From the above mentioned scenarios it should become clear that the options for dealing with ‘something interesting’ are very limited and the expected effect on our active learning strategy rather slender. Nonetheless we implemented the removal of anomalies and will show the results thereof in chapter 4.

The implementation of this removal approach is rather straight forward. In algorithm 3.2 we see an excerpt of algorithm 3.1 with a new else statement on line 18 and 19. This means that if \( x \) is misclassified and there are no nearest neighbors in the pool that are closer then the \( k \)th nearest neighbor in the train set, then \( x \) will be removed from the train set \( T \).

\begin{algorithm}
\begin{algorithmic}[1]
\REQUIRE \( M \neq \emptyset \)
\STATE Sort \( M \) based on certainty (descending)
\FOR{each \( x \in M \)}
\STATE Select nearest neighbor \( x_{\text{NN}}^P \in P \)
\STATE Select \( k \)th nearest neighbor \( x_{\text{NN}}^T \in T \)
\IF{\( |x - x_{\text{NN}}^P| \leq |x - x_{\text{NN}}^T| \)}
\STATE \( x^* \leftarrow x_{\text{NN}}^P \)
\STATE \text{break}
\ENDIF
\STATE \( T \leftarrow T - x \)
\ENDFOR
\IF{\( x^* = \emptyset \)}
\STATE Select \( x^* \in P \) based on strategy \( s \)
\ENDIF
\end{algorithmic}
\caption{Removal procedure}
\end{algorithm}
Chapter 4

Experiments & Results

In this chapter we will elaborate on the numerous experiments we have done. First we will briefly look at a few details of the implementation in section 4.1 and discuss the data in section 4.2. Finally, the experiments and their results will be analyzed and discussed in section 4.3.

4.1 Implementation

As the basis for our implementation we took the software developed by Divalla et al.\(^1\) for their project on context in object detection [8]. The software is written in Matlab. As discussed in section 2.2, Divalla studied a wide variety of contexts which they applied to an object detector for re-scoring the predictions. One of these contexts is the gist, in this case an implementation by Oliva and Torralba [9]. This context turned out to be a highly predictive feature for the classifiers they were training.

Our approach is concerned with active learning, so we wanted to make Divalla’s system iterative and apply our strategies to it. Ideally we would hereby use a classifier which can update its model after receiving a new training example, instead of retraining a completely new model every time. The logistic regression implementation which was part of the supplied software unfortunately did not support this option. This means that a run can become quite computationally intensive, depending on the number of iterations we use.

We refer to a run as the iterative process of training a classifier for a single class, starting with a train set \( T \) and running our active learning strategy for \( q \) iterations. The budget as described in the previous chapter is not implemented explicitly. Instead we simply limit our run by the amount of iterations. In most experiments we repeat the runs a number of times (mostly \( r = 10 \)) and take the average of the results. Every repeat starts with a different train, validation and test set (and optionally also a different pool).

For our project we use an automated oracle. As we will explain in the next section, we have ground truth labels for all our data. So instead of manually supplying a label when our systems asks for it, we simply look up the available

\(^1\)http://www.cs.cmu.edu/ santosh/projects/context.html
label. We can then add the labeled instance to the train set. This means we can run thousands of iterations without any human effort.

4.2 Data

In section 2.1.2 we shortly touched upon the VOC PASCAL challenge. Since this the data that Divalla et al. used for their project, we adopted their choice. The images are from the challenge of 2008 and are publicly available online\(^2\). There are 20 object classes divided over 4 categories:

- **Person:** person
- **Animal:** bird, cat, cow, dog, horse, sheep
- **Vehicle:** aeroplane, bicycle, boat, bus, car, motorbike, train
- **Indoor:** bottle, chair, dining table, potted plant, sofa, tv/monitor

We were unable to get a hold of the official test set, so we have gathered the training and validation data which makes for a total of 5096 annotated images. It is important to note that the data is heavily skewed. The object that appears most, ‘person’, has 2530 positive examples, while ‘sheep’ appears on only 96 images. Table 4.1 shows a full list of all the classes and their number of positive samples.

<table>
<thead>
<tr>
<th>Class</th>
<th>positive samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘person’</td>
<td>2530</td>
</tr>
<tr>
<td>‘bird’</td>
<td>348</td>
</tr>
<tr>
<td>‘cat’</td>
<td>372</td>
</tr>
<tr>
<td>‘cow’</td>
<td>108</td>
</tr>
<tr>
<td>‘dog’</td>
<td>433</td>
</tr>
<tr>
<td>‘horse’</td>
<td>242</td>
</tr>
<tr>
<td>‘sheep’</td>
<td>96</td>
</tr>
<tr>
<td>‘aeroplane’</td>
<td>281</td>
</tr>
<tr>
<td>‘bicycle’</td>
<td>234</td>
</tr>
<tr>
<td>‘boat’</td>
<td>247</td>
</tr>
<tr>
<td>‘bus’</td>
<td>142</td>
</tr>
<tr>
<td>‘car’</td>
<td>549</td>
</tr>
<tr>
<td>‘motorbike’</td>
<td>250</td>
</tr>
<tr>
<td>‘train’</td>
<td>191</td>
</tr>
<tr>
<td>‘bottle’</td>
<td>320</td>
</tr>
<tr>
<td>‘chair’</td>
<td>498</td>
</tr>
<tr>
<td>‘diningtable’</td>
<td>242</td>
</tr>
<tr>
<td>‘pottedplant’</td>
<td>245</td>
</tr>
<tr>
<td>‘sofa’</td>
<td>264</td>
</tr>
<tr>
<td>‘tv’</td>
<td>272</td>
</tr>
</tbody>
</table>

Table 4.1: Number of positive examples for all classes in our data set

In section 1.1.4 we have explained the consequences of skewed data and how to deal with this situation. In the experiments, we will clearly see some of the

\(^2\)http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc2008/
effects it has on several evaluation methods for certain classes. This is not a problem per se, as long as we are aware of it. Keep in mind that in real-world problems, the skewness of the data is often unknown.

### 4.2.1 Feature Representation

When looking at the components from Divalla’s system, the gist was the first context-feature that came up as being easily implementable and readily available. We started our project by developing our experiments for this representation of the data. Eventually it turned out to be outside the scope of this project to include other types of contexts, so all our experiments are done on gist representations of images.

The gist is calculated in the standard way Divalla et al. did in their project, which results in feature vectors with 560 values. What these values represent is explained in the paper by Oliva and Torralba and is of no further concern for the rest of this research. It is worth mentioning that the gists were not normalized at first. After discovering that this lead to ambiguous and unusable distance measures for finding the nearest neighbor, we chose to convert the complete data set once and work with the normalized version of the gist.

The main observation is that apparently the gist is well suited for predicting the occurrence of an object class on an image. More formally we learn a model to predict:

\[
P(o | \text{gist}(i))
\]

Where \(o\) is the presence of an object of a particular class and \(i\) is an image.

### 4.3 Experiments & Results

We will discuss 3 experiments, all serving a different purpose. The first experiment will investigate the suitability for a gist predicting the object presence in general and the possibility of apply active learning strategies to this problem. The focus of experiment 2 will be to compare the strategies we introduced in section 3.2, along with a comparison of how much our plug-in can improve the performance of these strategies. Finally, we will investigate in more detail some examples of when our plug-in comes into action. We will show images that get misclassified and thereby trigger our algorithm.

When choosing a desired size for the train set, validation set, test set and pool, we have to take into consideration the computational complexity of our system. For the train set we can state that smaller is better if we want to see the effect of active learning to the full extend. The test set should generally be large for a more sophisticated evaluation of our model. But the most important size we have to define is that of the pool. We want the pool to be large enough for the system to have a lot of choice. The pool must represent the total data distribution. But if increase pool size, we will also increase the amount of calculations our system has to do at every iteration for choosing \(x^*\).

For most experiments, our train set consists of only two instances, one positive and one negative example. In general we use a large test set of 2000 images and our validation set consists of 500. For the pool we found a size of 1000 to be reasonable. These last three sets are randomly sampled from the complete
data. The pool will stay the same for all runs. To get good average results over 10 runs, we create a new test and validation set. We run different strategies with the same starting conditions, meaning the result are comparable.

Also, the initial train sets are re-used when comparing different strategies to each other. In practice this means that we prepare 1 pool, \( r \) validation and test sets, and \( c \times r \) train sets, where \( c \) is the number of classes and \( r \) the number of repeats.

### 4.3.1 Experiment 1: A baseline for classifying on gist data

In this first experiment we analyze the applicability of our logistic regression classifier to the normalized gist data. We use the standard settings as declared above and \( q = 30 \). We look at all 20 classes of objects and as our basis query strategy we use uncertainty sampling. We will not apply our plug-in for this experiment.

In figure 4.1 we see a plot of the error on the test set. On the x-axis we see the number of samples in the train set, which increases by 1 at every iteration. This image seems to show very promising results, but we have to be aware that the error on the test set may not be the best measure of performance for this type of problem. In fact it is quite uncommon to use error on heavily skewed data. If we have very little positive samples in our data, and our model would simply predict every sample to be negative, the error would be very small.

![Figure 4.1: Error on the test set for all 20 classes, using uncertainty sampling without our plug-in. We sample 1 instance at 30 iterations and repeat this 10 times with different data. These results are averaged over those 10 runs.](image-url)
A more sophisticated measure for these type of problems is called the \textit{F-score} or \textit{F-measure}. It combines \textit{precision} and \textit{recall} into a single measure, defined as:

\[ F\text{-score} = 2 \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \]

Figure 4.2 shows the \textit{F}-score, again for all classes over 30 iterations, averaged over 10 runs.

The iterative process of active learning can be computationally exhaustive because the model has to be re-trained after every iteration. To give a view of what would happen to the error and the \textit{F}-score when many more samples would be used, we did a special run which makes use of batches. Instead of sampling samples one by one, we take a group of 20 at once. Normally one would apply a strategy which tries to maximize the informativeness of the group, but we simply sample the samples that lie closest to the decision boundary. Even if we would have used random sampling though, the effect of increasing data on the performance would still be very clear. We only did one run, so the fluctuation in the plot is larger than in the other plots (which are averaged over 10 runs). At the final iteration, all samples in the pool have been added, because we sample 20 instances over 50 iterations (i.e. \( q = 50 \)). The results are shown for the error in Figure 4.3 and for the \textit{F}-score in Figure 4.4.

Based on these results we can conclude that predicting object presence given the gist of an image has its limitations. The class ‘person’ has the highest \textit{F}-score, but also the largest error on the test set. The large error can partly be
Figure 4.3: Error on the test set for all 20 classes, using uncertainty sampling without our plug-in. This is a single run of adding 20 instances over 50 iterations with the purpose of showing the effect of a large train set.

explained by the fact that ‘person’ is the largest class. Another explanation is the diversity of visual settings in which a ‘person’ may occur.

Another fairly high $F$-score is measured for the ‘aeroplane’ class. Again, it is one of the larger classes, but apparently its presence is much more predictable based on a gist. The error for this class is fairly low if we take the size of the class into account.
Figure 4.4: $F$-score for all 20 classes, using uncertainty sampling without our plug-in. This is a single run of adding 20 instances over 50 iterations with the purpose of showing the effect of a large train set.
4.3.2 Experiment 2: performance of our plug-in on uncertainty and positive sampling

Now that we know the baseline of how our logistic regression classifier performs on the data, we can dive into comparing the query strategies. We have applied uncertainty sampling and positive sampling to all classes. For both these basis strategies we look at performance increase when our plug-in is applied to them. We will show and analyze the performance on some specific classes and show a table with performance on all classes.

Although error showed to be a inconvenient measure for judging the performance of our classifier on this skewed data, it does provide us with valuable insight when comparing several strategies with each other. The effect of this measure is the same for all strategies and every run, so a comparison between them remains fair.

![Figure 4.5: Plots showing the error on the test set for 4 different strategies: ‘unc’ is regular uncertainty sampling, ‘uncMCR’ is uncertainty sampling enhanced with our plug-in, ‘max’ is positive sampling and ‘maxMCR’ is positive sampling with our plug-in applied to it.](image)

In Figure 4.5 we see plots for the classes ‘cow’, ‘aeroplane’, ‘car’ and ‘sofa’. We will discuss them briefly. For (a) ‘cow’ we can see that all 4 strategies perform almost equally. This is mainly due to the fact that ‘cow’ is one of the most skewed classes, which makes it very difficult to do anything smart or useful with the way we sample. In general it turns out that positive sampling works rather well on data with few positive samples.
In comparison, for (b) ‘aeroplane’ positive sampling shows a much worse performance in the first few iterations. This shows that the applicability of the strategies is indeed highly depending on the skewness of the data, because ‘aeroplane’ is one of the larger classes. In the end (after 30 iterations) the strategies are again hardly differing in their error. This perfectly illustrates how strategies may outperform each other at different points in the iterative process.

In the plot of (c) ‘sofa’ we see the opposite effect, where positive sampling performs better than uncertainty sampling in the early stage of learning. After 30 iterations there is still quite a large improvement, but the addition of more data will eventually lead to convergence where all strategies will have the same error.

Finally, (d) ‘car’ show us a nice example of how our plug-in has different effect on different basis strategies. The previous 3 plots show hardly any discrepancy for whether or not we use our plug-in, but in this one we can clearly see the positive effect it has on positive sampling and the negative effect of uncertainty sampling. We can explain this difference by looking at the probability of running onto a misclassification. If the basis strategy chooses samples for which it has high certainty, even though the train set is very small and the model itself is very unstable, there is a higher chance of sampling instances which incur misclassifications.

From the comparison of performance on these classes we can see that the applicability of our plug-in is highly dependent on the skewness of the data, as well as the ‘boldness’ of the basis strategy. To give an overview of the performance in general we show the mean drop in error between start (2 samples in the train set) and end point (31 samples in the train set) of our experiments in Table 4:2. For most classes, although with very small difference, positive sampling with our plug-in yields the best results.
### Table 4.2: Differences in error on the test set (2000 samples) between the start and end of a run \((q = 30)\), averaged over 10 runs. 4 strategies are compared: uncertainty sampling without (unc) and with our plug-in (uncMCR) and positive sampling without (max) and with our plug-in (maxMCR).

<table>
<thead>
<tr>
<th>Class</th>
<th>unc</th>
<th>uncMCR</th>
<th>max</th>
<th>maxMCR</th>
</tr>
</thead>
<tbody>
<tr>
<td>'person'</td>
<td>-0.0318</td>
<td>-0.0244</td>
<td>0.0006</td>
<td>0.0032</td>
</tr>
<tr>
<td>'bird'</td>
<td>-0.3822</td>
<td>-0.3759</td>
<td>-0.3864</td>
<td>-0.3911</td>
</tr>
<tr>
<td>'cat'</td>
<td>-0.3834</td>
<td>-0.3956</td>
<td>-0.3936</td>
<td>-0.3999</td>
</tr>
<tr>
<td>'cow'</td>
<td>-0.4942</td>
<td>-0.4987</td>
<td>-0.4999</td>
<td>-0.4989</td>
</tr>
<tr>
<td>'dog'</td>
<td>-0.3425</td>
<td>-0.3430</td>
<td>-0.3533</td>
<td>-0.3620</td>
</tr>
<tr>
<td>'horse'</td>
<td>-0.5206</td>
<td>-0.5301</td>
<td>-0.5277</td>
<td>-0.5308</td>
</tr>
<tr>
<td>'sheep'</td>
<td>-0.5022</td>
<td>-0.5028</td>
<td>-0.5089</td>
<td>-0.5071</td>
</tr>
<tr>
<td>'aeroplane'</td>
<td>-0.2320</td>
<td>-0.2321</td>
<td>-0.2197</td>
<td>-0.2348</td>
</tr>
<tr>
<td>'bicycle'</td>
<td>-0.3805</td>
<td>-0.3844</td>
<td>-0.3891</td>
<td>-0.3892</td>
</tr>
<tr>
<td>'boat'</td>
<td>-0.3571</td>
<td>-0.3617</td>
<td>-0.3799</td>
<td>-0.3895</td>
</tr>
<tr>
<td>'bus'</td>
<td>-0.4861</td>
<td>-0.4894</td>
<td>-0.4761</td>
<td>-0.4897</td>
</tr>
<tr>
<td>'car'</td>
<td>-0.3581</td>
<td>-0.3448</td>
<td>-0.3611</td>
<td>-0.3839</td>
</tr>
<tr>
<td>'motorbike'</td>
<td>-0.3655</td>
<td>-0.3656</td>
<td>-0.3740</td>
<td>-0.3812</td>
</tr>
<tr>
<td>'train'</td>
<td>-0.3825</td>
<td>-0.3787</td>
<td>-0.3923</td>
<td>-0.3848</td>
</tr>
<tr>
<td>'bottle'</td>
<td>-0.4206</td>
<td>-0.4192</td>
<td>-0.4405</td>
<td>-0.4427</td>
</tr>
<tr>
<td>'chair'</td>
<td>-0.3872</td>
<td>-0.3818</td>
<td>-0.3902</td>
<td>-0.3940</td>
</tr>
<tr>
<td>'diningtable'</td>
<td>-0.5059</td>
<td>-0.5101</td>
<td>-0.5090</td>
<td>-0.5164</td>
</tr>
<tr>
<td>'pottedplant'</td>
<td>-0.5366</td>
<td>-0.5362</td>
<td>-0.5413</td>
<td>-0.5460</td>
</tr>
<tr>
<td>'sofa'</td>
<td>-0.5124</td>
<td>-0.5199</td>
<td>-0.5395</td>
<td>-0.5391</td>
</tr>
<tr>
<td>'tv'</td>
<td>-0.4360</td>
<td>-0.4343</td>
<td>-0.4482</td>
<td>-0.4477</td>
</tr>
<tr>
<td>average</td>
<td>-0.4009</td>
<td>-0.4014</td>
<td>-0.4065</td>
<td>-0.4113</td>
</tr>
</tbody>
</table>
4.3.3 Experiment 3: examples of misclassification and anomaly removal

In this experiment we will investigate the performance of our plug-in. As mentioned in Section 3.5, our approach is designed to include the removal of anomalies. However, the two effects can easily be separated and tested independently. We look at the error on the test set and use uncertainty sampling as our basis strategy.

In the following results, ‘unc’ refers to basic uncertainty sampling without our plug-in, ‘uncMC’ means we apply our plug-in for sampling and in ‘uncMCR’ the removal of anomalies is applied on top of that. Whereas in the previous experiments we looked at the average results over 10 runs, we will now look at single runs to be able to show the images that were chosen from the pool ($x^*$), the misclassifications in the train set ($\tilde{x}$) and the images that were removed. One technical implication of removing samples from the train set is that the interpretation of the y-axis changes. The size of the train set is no longer corresponding directly to the iterations.

Class: Bus, Repeat: 2

In Figure 4.6 we see a plot of the error on the test set for the ‘bus’ class, repeat 2. At iteration 17 we can clearly see the effect of the removal of an anomaly. We can also see that at the beginning of the run, in the first iteration, our plug-in shows a huge improvement over the basis strategy. We will first explain why this improvement should be considered a coincidence instead of proof for the effectiveness of our approach. After that we will look into the anomaly removal in iteration 17.

At the beginning of this run, with a train set of 1 positive and 1 negative sample, there is a misclassification on the positive sample. Figure 4.7 shows (a) the image that was misclassified, (b) the image that our plug-in chose from the pool and (c) the image that the basis strategy picked. The misclassification may seem obvious at first (the ‘bus’ on this image is hardly recognizable), but remember that it is not about what we see in this image. All the algorithm does is fit a model to the train set. If there is to little discerning information to train on, the model will predict a probability of 0.5 for every sample. In Section 1.1.1 we stated that $y = 1$ if $h_{\theta}(x) > 0.5$. This is merely a design choice (0.5 becomes either positive or negative, but we have to choose), but it will yield a misclassification nonetheless. This could be prevented by choosing a higher value for $\zeta$ (see Section 3.4.1). But since our experiments have been carried out with $\zeta = 0.5$, this is what happens.

In iteration 17, a sample is removed from the train set. This results in a better model, as can be derived from the lower error for ‘uncMCR’. It turns out that the anomaly that was removed, is actually the same image as we previously discussed for the beginning of this run. Figure 4.8 shows the image.
Figure 4.6: A plot of the error on the test set for uncertainty sampling without our plug-in (unc), with our plug-in (uncMC) and with the anomaly removal applied (uncMCR). In the first iteration our plug-in picks a much more informative sample. Later, in iteration 17, the removal of an image leads to better performance in subsequent iterations. The related images are shown in Figure 4.7 (for iteration 1) and Figure 4.8 (for iteration 17).

Figure 4.7: Images that play a role in the performance of our plug-in in run 2 for the ‘bus’ class.
Figure 4.8: In iteration 17 of run 8 for the ‘bus’ class, this image is removed from the train set. Apparently there were no more nearest neighbors in the pool that could be picked according to our algorithm. The removal leads to better performance in subsequent iterations as can be seen in Figure 4.6.
Class: Car, Repeat: 8

Another positive example of removing an anomaly can be seen in run 8 of the ‘car’ class. Again, a plot of the error on the test set is provided in Figure 4.9. We can tell that applying only the sampling part of our strategy does not incur any improvement. When the misclassified sample is removed, the retrained model does perform better. This means that the removed sample was apparently not a great representative for the ‘car’ class in the overall data distribution. Intuitively, from a human judgment perspective, we could see that this image (Figure 4.10(c)) is indeed not a particular example of a setting in which we would say it is very likely to find a car.

![Figure 4.9: A plot of the error on the test set for uncertainty sampling without our plug-in (unc), with our plug-in (uncMC) and with the anomaly removal applied (uncMCR). In iteration 17, the plug-in picks a different sample than the basis strategy. By itself this does not improve performance (uncMC), but with the additional removal of a misclassified sample it does (uncMCR). The related images are shown in Figure 4.10.](image)

In Figure 4.10 we see the images that were chosen for $x^*$ by respectively uncertainty sampling and our plug-in. We do not need a full understanding of the features that the gist provides us with, to sense that (b) has a higher probability of showing a car than (a). However, neither image actually has a car on it, so the retrained model has little to learn from. Choosing the image on which we might expect a car (b) seems rather inconvenient in that sense, because we add a negative sample to the train set with a gist that could be a good representative for pictures of cars. In contrast, adding image (a) as a negative
sample seems less dramatic because this image is indeed a bad representative for pictures of cars. This effect is indeed visible in that ‘unc’ outperforms ‘uncMC’ in iteration 18.

(a) $x^*$ according to uncertainty sampling

(b) $x^*$ according to uncertainty sampling with our plug-in

(c) $\hat{x}$, the misclassified image that was removed

Figure 4.10: Images that play a role in the performance of our plug-in in run 8 for the ‘car’ class.

Finally, a few more examples of images that were removed during the runs for experiment 3 (all runs, all classes) are depicted in Figure 4.11. The overall sense for these examples is that they stand out from the rest of the data because of very explicit properties. A white border, heavy vignetting or a very strong color or the lack thereof is apparently of substantial influence on the gist. This overview should also provide some insight in the diversity of the data set and complexity of the task of object recognition.
Figure 4.11: Examples of images that have been removed during the runs for experiment 3. The overall sense is that these images have a specific visual property that makes them stand out from the rest of the data, like a border or strong vignetting.
Chapter 5

Conclusions, Discussion & Future Work

In this chapter we will draw conclusions based on our experiments and results. During our research, we ran onto several interesting aspects which we were unable to incorporate in our investigation. These will be discussed in Section 5.2. Finally, in Section 5.3 we will briefly discuss the possibilities that came forth in during this project for further investigation.

5.1 Conclusions

We have introduced a new strategy for sampling data in a pool-based active learning scenario. Our approach uses misclassified instances in the train set as a trigger for investigation. If there are no misclassifications we apply a basis strategy (e.g. uncertainty sampling). However, if a misclassification occurs, the algorithm will sample data from around that misclassified point. This is done by looking for nearest neighbors in the pool. When some stopping-condition is reached after sampling one or more instances in the area, our algorithm will consider this point to be an unwanted anomaly. To prevent this anomaly from any further negative influence on the model, the system will remove the point from its train set. This last, rather resolute step, which is very uncommon in machine learning in general, yields promising results.

To be able to investigate the effect of our strategy thoroughly, we also introduced several new basis query strategies which are based on uncertainty sampling. We have discussed their properties extensively and for at least one of them, positive sampling, we show its stand-alone performance, as well as its performance when used within our approach.

For our experiments we used a logistic regression classifier in an object recognition learning problem. We have shown that our approach performs well in certain specific situations. However, we have to conclude that misclassifications in the train set, the trigger for our method, does not occur very often. Fortunately, our approach is computationally cheap and does not cause any negative effects on the basis strategy when it is not triggered.

We can now answer the research questions as stated in Chapter 1:
1. Are misclassifications in the train set useful for detecting informative data?

2. Can we show improvement on existing query strategies, like uncertainty sampling, by removing data from our train set?

We have shown the potential of using misclassifications in the train set as a trigger for sampling data around them. Although the experimental results are somewhat marginal, we believe that the question can be answered positively. The extent to which the misclassifications are useful is subject to many properties of the learning problem setting.

For the second question we also propose a confirming answer. Not only the experiments in Chapter 4, but also the theoretical background and discussion of scenarios in Chapter 3 should provide the reader with a good feel for when removing data from the train set is a feasible strategy.

5.2 Discussion

In Chapter 3 we discuss the effects for choosing a value for $\zeta$. This parameter defines the minimal error for triggering our plug-in. In our experiments we used a value of 0.5, because this is the standard for considering misclassification. Unfortunately, because of design choice, this means that when a positive and a negative sample both get $P(y = 1|x) = 0.5$, then one of them is considered as misclassified. We were unable to experiment with other values within the scope of this project, but it should be noted that finding an optimal value for this parameter is very dependent on the properties of the data set and the learning algorithm.

The potential of our approach would probably be better shown by using data in which we have more anomalies. The choice for our data however, was made at the beginning of the project, when the main focus was on using contextual information in computer vision. When we ran onto the research by Divvala et al. the goal shifted gradually to applying active learning to their system. We had the ambition to use more than one ‘context’ from their system, but this turned out to be unrealistic within the scope of this project. In the end there is no real problem with our data, any type of data could be used to run experiments on. However, ideally we would have investigated into other data sets in which the trigger for our algorithm may occur more frequently.

5.3 Future Work

As mentioned in the previous section, applying our approach to different data sets could potentially show a much greater effect, assuming that there are data sets in which the trigger occurs more often. This is obviously also depending on the learning algorithm. If we would replace our logistic regression classifier with a Support Vector Machine (SVM), other rules would apply which will lead to different effects.

This brings us to, in our opinion, the most interesting extension of this research. In Section 3.4.1 we explained that there are 3 possible causes for a misclassification in the train set. The first possible cause is that the classification
algorithm is simply unable to fit a correct model to the data, because of its bias (e.g. fitting a linear model to non-linear data). If we could correctly detect such a situation automatically, then we can introduce a strategy which will switch between classification algorithms if necessary. We would think of such a strategy as relaxing the bias of the classifier, either by choosing another algorithm or by redefining the features on which we classify.

Parallel to this strategy, we think that looking into the options for splitting an ‘unlearnable’ class into two or more ‘learnable’ subclasses could be a very useful proceeding. We have touched upon this subject briefly in scenario 3 in Section 3.5.3.
Bibliography


