Cooperative Visual Object Learning

Master Thesis

Jovana Radojević

Cooperative Visual Object Learning

Master Thesis

Jovana Radojević
Student No: 4510542

Supervision:	Prof. Dr. Jens Kober,	Delft University of Technology
	Dr. Lydia Fischer,	Honda Research Institute EU

September 18, 2017

Correspondence:
radojevic.r.jovana@gmail.com

"The important thing is not to stop questioning.
Curiosity has its own reason for existing."

Albert Einstein

Acknowledgments

I wish to express my sincere thanks to Honda Research Institute in Germany (HRI-EU) for providing me the resources necessary for my thesis. The Institute has supported my research not only by giving me access to their state of the art hardware, but also through intellectual and moral support provided by its members. I would especially like to thank Dr. Lydia Fischer for sharing her expertise, dedicating her time and leading me in the right direction throughout this venture.

Furthermore, I am deeply grateful to my academic supervisor Prof. Dr. Jens Kober, from Delft University of Technology (TU Delft), who has made the collaboration between the HRI-EU and TU Delft possible. The ideas, advice and valuable guidance that I have received from Dr. Kober have shaped my thesis and inspired me to further explore this scientific field.

Finally, I must express profound gratitude to my parents, my brother and to my partner for giving me unfailing support and continuous encouragement during the years of my study and through the process of working on this thesis. This accomplishment would not be possible without them.

Abstract

A lot of attention has recently been focused on possible benefits of the cooperation between machines and humans. Taking the best from machines and humans and joining them together can produce results which exceed each collaborating partner performing separately. A common belief is that the key for good cooperation is an excellent communication. Some important aspects of communication are self-evaluation processes. When applied to humans these processes improve the communication quality between humans. Therefore, we believe that employing self-evaluation processes at machines advances the human-machine communication, and cooperation quality. Accordingly, this thesis is exploring communication strategies between machines and humans. More precisely, it examines possibilities for the communication improvement through an exploration of self-evaluation processes of classifiers.

Firstly, we introduce a baseline framework, an interactive visual category learning architecture, called Tubby at Honda Research Institute in Germany. For simplicity we consider in a first step the classification of objects rather than the more difficult task of learning multiple categories per object. We then introduce theoretical foundations used in the thesis. The background on classification, neural networks, outlier detection and assessment of classifiers is explained in depth. We outline the critical importance of self-evaluation in classification. Therefore, we propose two self-evaluation measures which are incorporated within a testing and a training strategy. The first measure captures a confidence in predictions during classification and it is used within the proposed testing strategy. The second measure denotes the quality of each training sample with respect to the generalization performance of classification and it is used within the proposed training strategy. The quality of the training sample essentially represents how different the current training sample is from all of the previously acquired training samples of the object. The confidence and the quality measure are communicated through a graph to the user of the system. Depending on whether the system is in the testing or training phase the value of the corresponding measure is provided to the user. Two ways of deriving each of the two measures are presented. Essentially, we offer two testing and two training strategies. Furthermore, we consider each proposed strategy separately and a combination of those. We then evaluate all of the considered cases against a baseline strategy. The evaluations are performed in different dimensionalities of the feature space for different numbers of training and testing objects. We present an offline simulation of the interaction between Tubby and the user. Results of the simulation provide additional insights into the working mechanisms of the proposed strategies and measures.

The proposed strategies improve the baseline performance. The absolute improvement of the average classification accuracy varies between 1% and 25%, depending on the dimensionality of the feature space and number of training and testing objects. The best results are achieved when a combination of the proposed training and testing strategy is used. The biggest improvement is observed when a lot of objects are in the learning process (≈ 100), and the dimensionality of the feature space is high (≥ 10D). In the actual application setting this is the most realistic case - a large number of objects and a high dimensionality of the feature space.

Contents

1
Introduction

Over the past decades there has been an interest in possibilities of making machines intelligent like humans. Intelligent machines like robots can replace humans in many areas of work. Robots can be utilized in carrying out laborious, repetitive and time-consuming tasks efficiently. For example, robots can be used in manufacturing industry and increase workplace safety. Workers can be moved to supervisory roles where they no longer have to perform dangerous applications in hazardous settings. Recently, endeavors of making machines intelligent were focused on the development of machine learning algorithms. Machine learning (ML) is a subfield of computer science that gives machines the ability to learn directly from data. ML allows to find hidden insights into complicated structures. The inspiration for a creation of such algorithms came from the specific way human brains learn and process data. One of the examples of how the human brain works is recognizing objects. Humans learn objects based on the understanding that each object has a set of common relevant features [12]. We have the ability to perceive an object's physical properties and apply semantic attributes to it. Regardless of an object's position, illumination or even partial occlusion, humans can effectively identify and classify an object. Inspired by this amazing ability of the human visual system, researchers were interested in possibilities of building machines which use ML algorithms and which are able to distinguish objects efficiently like humans do. Exploration of ML algorithms which are able to cope with object recognition can potentially improve driver assistance systems, robot assistance systems, etc.

Another area of interest is to make machines able to collaborate with humans. A collaboration between machines and humans is a purposeful coupling in which both partners cooperate in order to achieve shared or overlapping objectives through the fulfillment of joint tasks. Machines and humans present complementary parts for the development of common tasks. On the one hand, machines complete those subtasks which require extreme precision or accuracy better than humans. On the other hand, humans perform those subtasks which cannot be executed by machines because of their complexity. For instance, humans are able to perform specialized tasks which require intelligence and a high level of abstraction. This synergy between humans and machines enables the development of more flexible and complex tasks which cannot be performed individually by a human or a machine. Structured methods of collaboration employ a good communication, a process in which a flow of information is processed between the machine and the human through a common system, e.g. voice, symbols, signs, images. In order to achieve a good communication the most informative data must be shared between the collaborative partners, the machine and the human.

Self-evaluation, as the process of judging on our own actions, attitudes, or performances, can improve the communication and increase chances of a success in a human-human collaboration. In communication with people an evaluation of the received information from the collaborative partner is performed. It is noted to the other collaborating partner if something communicated is not clear or understood. More information is required if the originally provided information is not enough for a decision. Information about the evaluated confidence in our decision is given to the other party. Therefore, the collaborating partner can adapt its own actions accordingly to the feedback obtained from us, and by doing so increase the chances of a successful collaboration. Likewise, in human-machine interaction self-evaluation of machines can enrich the information flow and advance the communication between the machine and the human. Thereby, the collaboration quality increases, and the successful fulfillment of shared objectives is a more likely outcome.

By self-evaluation of classifiers, a process of evaluation of the quality of decisions or performances of the classifier is considered. A subcategory of these processes is the evaluation of the quality of predictions and the quality of data obtained before the prediction. Finally, having this type of evaluation in machines which collaborate with humans can increase the likelihood of the successful objective achievement, and prepare the human party for an adaptation in collaboration with the machine, depending on the self-evaluation outcome.

This thesis is exploring communication strategies between the machine and the human. It investigates possibilities for the communication improvement through exploration of self-evaluation of classifiers. The targeted application is object recognition. Firstly, a baseline visual learning architecture is introduced in Chapter 2. A discussion on important aspects of the architecture, which are tackled in the thesis is provided, along with a research target. Chapter 3 introduces theoretical foundations on classification, neural networks, outlier detection and assessment of classifiers, which are necessary for a good understanding of the project. This is followed by Chapter 4 which outlines the critical importance of self-evaluation and proposes new self-evaluation methods. Chapter 5 presents the evaluation of the proposed methods, experiment details and a discussion on the results. The thesis concludes with Chapter 6, which summarizes the project, the most important findings, and provides directions for the future research.

2 System Architecture

A discussion of a visual category learning architecture which serves as a main motivation for the project development is presented in this chapter. The description of the architecture along with a discussion on important aspects thereof. The chapter concludes with the research target of the master thesis.

2.1. Description of the Architecture

An amazing capability of the human visual system is the ability to learn an enormous number of visual categories which are incrementally acquired during life. Object categorization is one of the primary tasks of the visual system. Sensory processing of visual stimuli, along with prior visual experience, leads to judgments of categories of objects around us. Each object is assigned to a set of labels. The object is described with its features, e. g., color, shape. Humans are capable of assigning labels to an object which is not previously seen, accordingly to the features which the object has, based on their previous experience. Inspired by these human abilities Kirstein et al. [29] have developed an interactive visual category learning architecture at Honda Research Institute Europe (HRI-EU) which is capable of real-time incrementally learning several visual categories based on hand-held objects. The system is called Tubby at HRI-EU and it will be referred as Tubby in the following text.

Tubby's aim is to learn object categories in a similar way humans do. It builds up its knowledge about categories over time. The objects are presented by the human teacher in front of the system, like shown in Figure 2.1. Tubby is then able to either recognize categories of the object or to learn its features. When the new object is introduced to Tubby, it updates its classification model of categories to learn the new object.

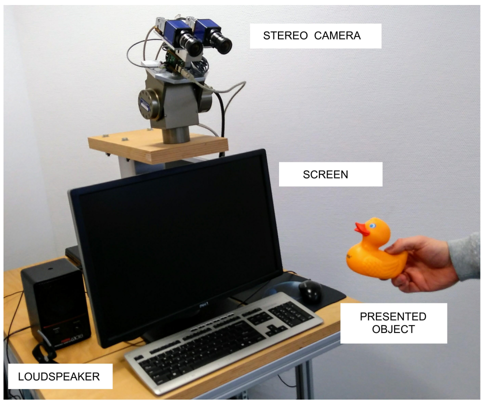

Figure 2.1: Tubby: The user is presenting the object to the system. The presented object is rotated in front of the stereo camera. The loudspeaker and the screen serve for communication with the user. [1]

[1]Source: HRI-EU

In order that the system learns categories the human teacher rotates an object in hand in front of the stereo camera and tells the system a set of corresponding visual categorization labels, e. g., "red, white ball". Based on extracted color and texture features the system tries to learn the objects. The phase during which the system is learning objects is called training. In the next step the user can present a previously seen or a new object. The system identifies the new visual input and tells the user the detected object's categories or responds with "unknown object". This phase during which the system tries to recognize the object is called testing. The user confirms or corrects the labels.

Tubby is composed of a combination of hardware and software components. The software is explained in the following paragraphs, whereas the hardware components are a stereo camera, a microphone, a speaker and two laptops which serve as a main computational unit.
Figure 2.2 shows the functional blocks, and the interaction setting between the human user and the system.

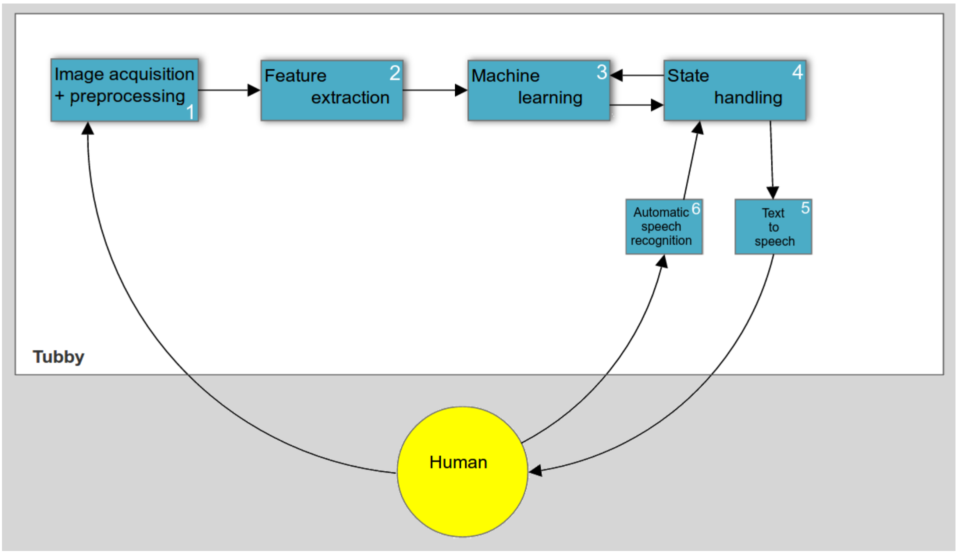

Figure 2.2: Functional block diagram of Tubby system. The human shows an object to the system. Block 1 extracts the segment of an image with the presented object. Block 2 extracts features of the object. Block 3 uses features to learn the object. In the testing phase this block predicts the labels of the object. Block 4 organizes the interaction timing and translates predicted labels into text. Block 5 translates those labels into speech. The user confirms or corrects the labels by providing speech labels. Block 6 translates the speech labels into text. Block 4 then delivers these labels to block 3. If the user corrects the labels, block 3 changes its internal classification model.

Block number 1 of Figure 2.2 is in charge of an image acquisition and the preprocessing of images. This block collects images from the stereo camera. Based on the distance information the block extracts the part of the input image with the object presented by the user. The closest object to the camera is selected and a region around it is cropped, creating a segment of the image which is used in further processing, like shown in Figure 2.3.

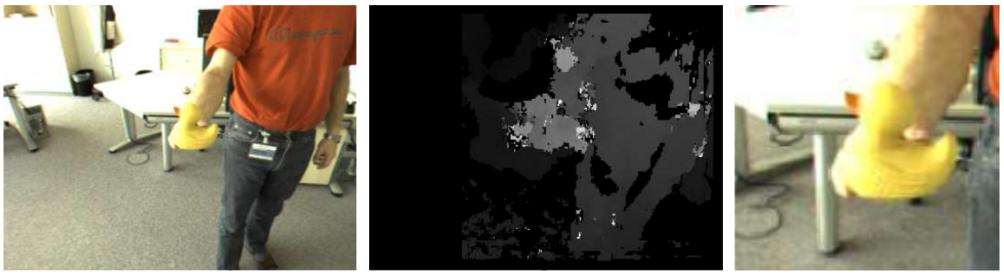

Figure 2.3: Block 1 performs image processing. The left image is the input of the stereo camera. The middle image shows the distance information. Based on this image the segment with the closest object to the stereo camera is selected [29] which is shown on the right image.

Additional processing of the extracted part of the image is performed. For details we refer to [29]. Block number 2 of Figure 2.2 extracts characteristics of the presented object. Specifically, it extracts color, shape and texture features of the object which are used in block number 3 for learning. Block 3 is the processing unit of the system which learns the objects. This block works in collaboration with block number 4, which controls the system-user interaction. When the object is presented to the system for recognition, block 3 predicts a set of categories, and block 4 determines when the labels are reported to the user. This predicted set of category labels is in text form. Block number 5 is transferring written labels from text form into speech and reports them to the user. Then the user confirms or corrects the label. Recognition of the user's speech is performed by block 6. If the user confirms the labels, the next object can be presented. However, if the user corrects the labels, block 3 changes its internal classification model in order to learn the presented categories properly.

Important aspects of the communication strategy implemented on the Tubby platform are outlined in the following section.

2.2. Discussion on Important Aspects

An important aspect of the Tubby platform is the interaction between the user and the system. Tubby is build to be used in cooperative online scenarios. The key component of good cooperation between any two partners is good communication. By providing more information to the partner during cooperation the probability of the successful fulfillment of the shared goal is increased.

The currently implemented interaction strategy on the Tubby platform is based mainly on the exchange of speech information: the predicted labels which the system gives to the user, and the user's response. This communication strategy is very simple and rather limited. The user does not know anything about the platform's internal states, which leads to higher expectations and higher disappointment in case of failure. Possibly the user could perceive the system as annoying and irritating and therefore turn it off. On the other hand the system's performance is limited by the human behaviour. The speed and quality of information extracted is heavily dependant on the user's behaviour, e. g., how the user rotates an object, which is again linked to the understanding of the system's needs.

When communicating with people, humans (usually) provide feedback on the partner's actions. They say if they do not understand a question or a command. Humans ask for additional information if the originally provided information is not enough for a decision, or the probability of a mistake is high. They have the ability to provide information about the confidence of decisions if the chances of success are below expectations. Humans note when they already know something which the partner is teaching them. They note as well when something which they did not know was shown to them, and whether it is useful to know that.

We believe that the implementation of a communication strategy which mimics the human's natural behaviour on the platform would increase the understanding between the user and the system. Scenarios of special interest are listed here. It is stated how the improved system should act in the ideal case.

1. Tubby has an internal confusion about some specific objects, which is the reason why the predictions are wrong for these objects sometimes. They are difficult to recognize, and the probability of a wrong prediction is high. The system is sometimes able to distinguish properly between these objects but only if special views of these objects are shown to the system. The user neither knows that there is the confusion nor whether the confusion can be solved.

   Ideal scenario: The system evaluates whether some objects are difficult for the distinction, and which objects exactly. It evaluates as well which views of those objects are suitable for the recognition. This information is delivered to the user.

2. Tubby is in the process of learning a new object or category. Views of this object are shown to the camera. Tubby has already seen these views before. Therefore, this is redundant information provided to the system. On the other hand, some other views of that object are not shown to

the system at all. Those views represent useful information. The user does not know this.

Ideal scenario: The system evaluates whether some views are already seen and suggests that to the user.

In order to investigate whether the listed improvements are possible and to evaluate the usefulness for the system, the following setting is used. Instead of categorization judgments we use the simpler task of object recognition. The main difference is that only one label is assigned to each object, as opposed to categorization where a set of labels is assigned to each object. The categorization process is a difficult problem by itself and by choosing object recognition instead the focus can be put on the communication improvement. Another change introduced to the original system is a change of the process of learning objects. Section 3.3 describes in detail the newly introduced process. This process is chosen since it highlights the important aspects of why internal confusion happens and enables deep insight into the prediction making. This setting simplifies the evaluation of the advancements in communication.

Additionally, in a recent experiment [14] it is shown that there exists a better process of generating features than the feature extraction process which is implemented on the platform. A more detailed description will follow up in the next chapter.

Figure 2.4 highlights in red the functional blocks (presented in Figure 2.2) which will be advanced. The above mentioned better feature extraction is targeted improvement in block 2. All internal evaluations are performed in block 3. Of special interest is the information exchange between blocks 3 and 4. Since block 3 is the main block for learning of objects and block 4 organizes interaction with the user, the information shared between these blocks is crucial for a high quality communication strategy. The additional block number 7 delivers the outcome of the self-evaluation process. This block suggests that delivering a feedback to the user about the outcome of the self-evaluation processes can be both speech and visual.

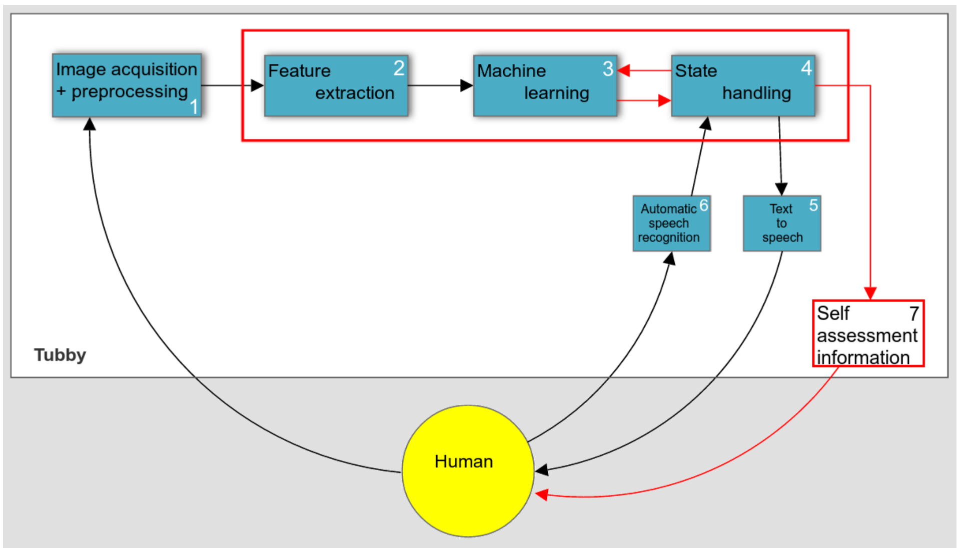
Image acquisition + preprocessing 1
Feature extraction 2
Machine learning 3
State handling 4
Text to speech 5
Automatic speech recognition 6
Self assessment information 7
Tubby
Human

Figure 2.4: Functional block diagram of the Tubby system with blocks intended for improvement marked in red.

The following section extracts the research target according to the discussion above.

2.3. Research/Technical Target

The main goal of this master thesis is the development and investigation of new communication strategies which enrich the information flow between the Tubby and the user by the deployment of self-evaluation processes. The following improvements are targeted:

- Improvement of the control loop between the user and the system.
  The aim is to develop methods which provide an understandable feedback of the system. The human should have an insight into the learning process, and into the self-evaluation of classification performance. At the same time, the system should be able to optimize the response of the human, that the overall performance of the system is increased.
  - Testing strategies
    Investigation of new testing strategies which evaluate the confidence of predictions. The aim here as well is to show that adequate communication with the user can increase the performance and lead to better and more informative predictions.
  - Training strategies
    Investigation of new training strategies which evaluate the quality of the received training information. The aim is to show that communication with the user can increase the performance and lead to a representation of higher quality of an object.
- Exchange of the existing feature extraction process with deep neural networks visual feature generation.

In order to fulfill the mentioned targets, the next chapter introduces fundamental theory on the topics of classification and image recognition, online learning, neural networks, and the assessment of classifiers.

3 Fundamentals

This chapter presents a theoretical background necessary for the understanding of the project. We provide an introduction to the following topics: Section 3.1 introduces Classification, Section 3.1.1 presents Online Learning, Section 3.2 explains Convolutional Neural Networks, Section 3.3 presents k-Nearest Neighbour classifier, Section 3.4 introduces Outlier detection and Section 3.5 explains Assessment of Classifiers. The role of each topic in the thesis project is outlined. Chapter 4 utilizes the theory presented in this chapter to propose new measures for self evaluation of classifiers, while implementation and experiment details are given in Section 5.1.

3.1. Classification

This section explains classification from the general and mathematical point of view. Classification is an important part of the thesis foundations. The first reason is that a single label classification is a special case of the classification which finds an application in object recognition. The setting which we use for the improvement of the communication between the user and Tubby system is the interactive object learning scenario. Therefore, the object recognition is an important component. The second reason is that online learning is a subfield of classification. The scenario described in Chapter 2 and discussed in the thesis is an online cooperative learning of objects. Images of the objects arrive in sequential order (as a function of time) during the interaction of the system and the user. Therefore, the data generation process is online and the whole process is suitable for online learning. General introduction to classification is provided next.

All organisms assign objects and events in the environment to separate classes or categories. Is the plant edible or poisonous? Is the person friend or foe? Was the sound made by a predator or by the wind? This allows them to respond differently, for example, to nutrients and poisons, and to predators and prey. Any species that lacked this ability would quickly become extinct [43].

Classification is the process in which ideas and objects are recognized, differentiated, and understood. In this particular case we define the classification as a process of assigning an object to a class which can be defined by a property that all its members share. A classification task usually involves separating data into the training and testing set. Each object in the training set contains one or more labels and several properties. The goal of the classifier is to produce a model which predicts one or more labels of the test data [43].

A single label classification is the special case of classification where each instance (e. g., object) is assigned to only one class (e. g., apple), and it is associated with only one label. Each instance $\mathbf{x} \in \mathbb{R}^n$ is assigned to a single label $l_i \in L$, $i \in \{1, ..., h\}$ from a set of disjoint labels $L = \{l_1, \dots, l_h\}$.

Application of Classification

Object recognition is a technology in the field of computer vision for finding and identifying objects from a sensor data, e. g., an image or video sequence. Humans have the ability to perceive an object's physical properties (e. g., shape, colour and texture) and apply semantic attributes to it (e. g., identifying the object as an apple). Regardless of an object's position or illumination, humans can effectively identify and classify an object. Objects can even be recognized when they are partially occluded. Recognition of objects is still a challenge for computer vision systems.

3.1.1. Online Learning

Online machine learning is a method of ML in which data are available in a sequential order and is used to update the best predictor for future data at each step. This ML method is opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once. Online learning gained more attention [20, 40] especially in the context of big data and learning from data streams. Online machine learning is a common technique used in areas of ML where it is computationally infeasible to train over the entire dataset. It is also used in situations where it is necessary for the algorithm to dynamically adapt to new patterns in the data, or when the data itself is generated as a function of time (e. g., presentation of objects, stock price prediction). Online learning has numerous advantages over offline methods like memory requirements are much lower because samples do not need to be stored, a huge amount of available data can be exploited by online methods and the training is usually much faster. Offline methods are not applicable if the data generation process is online or the underlying distribution changes over time [43]. However, batch techniques can exploit a global view of the data to generate more robust classifiers. Online algorithms, on the other hand, attempt to incorporate every single instance into the model, becoming more prone to over-fitting. Moreover, the *Stability-Plasticity dilemma*, introduced at the end of this section, explains the paradigm which affects the online learning systems.

In online systems the data is observed in a sequential manner. Streams of data are n-dimensional points $\mathbf{x} \in \mathbb{R}^n$. After each observation, the algorithm predicts one or more labels $l_i \in L$ from a set of h disjoint labels $L = \{l_1, l_2, \dots, l_h\}$. Once the algorithm has made a prediction, it receives feedback indicating the correct label. Then, the online algorithm may modify its prediction mechanism, presumably improving the chances of making an accurate prediction on subsequent rounds [43].

Stability-Plasticity Dilemma

The stability-plasticity dilemma [36] is a paradigm for artificial and biological online systems. The idea is that online learning requires *plasticity* for the integration of new knowledge, but also *stability* in order to prevent the forgetting of previous knowledge. Too much plasticity will result in previously encoded data being constantly forgotten, whereas too much stability will prevent the efficient encoding of new data into the system.

This dilemma occurs when classifiers are trained with a limited and changing training ensemble, causing the *catastrophic forgetting effect*. The problem of catastrophic forgetting has emerged as one of the main problems facing online learning systems. It happens when an online system, any biological or artificial memory, has to learn new inputs from the environment but without being disrupted by them. Catastrophic forgetting is defined as a complete forgetting of previously learned information by an online learning system exposed to new information [34]. Different types of online learning systems are affected with this effect, e. g., standard back-propagation neural networks, unsupervised neural networks, self-organizing maps [45].

3.2. Neural Networks for Vision

A general framework of neural networks is explained here. We put a particular focus on the convolutional neural networks. They are a special structure of neural networks and are primarily made to deal with visual tasks, and suitable for applications which deal with images, like the Tubby system for visual cooperation.

Neural Networks (NNs) and deep learning currently provide the best solutions to many problems in image recognition [50], speech recognition [42], and natural language processing [24]. A NN is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working together to solve specific problems. NNs, like people, learn by example. NNs with their amazing ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques [43].

Section 3.2.1 gives a brief introduction into neural network structures, basic principles and explains why they are so powerful. Section 3.2.2 presents deep neural networks and Section 3.2.3 provides an overview of convolutional neural networks. Feature extraction which is an important part of the thesis is explained in Subsection 3.2.3.

3.2.1. Neural Networks

Core building blocks of neural networks are processing elements and connections between them. The processing elements are called neurons (as shown in Figure 3.1), and the connections between the neurons are known as links which have a weight parameter (real numbers expressing the importance of the respective inputs to the output) associated with it. Each neuron receives stimulus from the neighboring neurons connected to it, processes the information, and produces an output. There are different ways in which information can be processed by a neuron, and different ways of connecting the neurons to one another which create specific neural network structures [43].

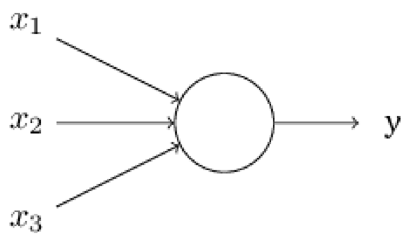

Figure 3.1: Neuron with connections [37]. In this particular case the neuron (circle) has 3 input connections (x_1, x_2, x_3) and 1 output connection $y(x_1, x_2, x_3)$. In general the input is defined as $\mathbf{x} = (x_1, \ldots, x_n)$ where n is the number of inputs, and $\mathbf{x} \in \mathbb{R}^n$. Each link from the input to the neuron is associated with a weight.

Neurons are arranged in a series of layers. The first layers are made of input neurons, and are designed to receive various forms of information from the outside world that the network will attempt to learn about, recognize, or otherwise process. The first layers are followed by one or more hidden layers of neurons which are processing the information. Hidden layers of neurons are followed by layers of neurons which signal how the network responds to the learned information, and are known as output neurons. Figure 3.2 illustrates how a neural network with one hidden layer looks like.

In NNs, the activation function of a neuron defines the output of that neuron given an input or set of inputs. Common activation functions in use are:

- *Step Function*: The output is a value y_1, if the input sum of the weighted inputs is above a defined threshold, and y_0 otherwise. The neuron is firing only in the first case. These step activation functions are useful when the network should classify an input pattern into one of two groups.

$$y(\mathbf{x}) = \begin{cases} y_1, & \sum_{i=1}^{n} w_i x_i \geq t \\ y_0, & \text{otherwise} \end{cases} \tag{3.1}$$

  where $\mathbf{x}$ is the input to the network, n is the number of inputs, t is a defined threshold and w_i are weights of the connections.

- *Linear combination*: The weighted sum input of the neuron plus a linearly dependant bias becomes the system output.

$$y(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b \tag{3.2}$$

where b is a bias.

- *Sigmoid Function*: The output is defined as

$$y(\mathbf{x}) = \frac{1}{1 + e^{-\sum_{i=1}^{n} w_i x_i + b}} \tag{3.3}$$

Sigmoid functions are prized because their derivatives are easy to calculate, which is helpful for calculating the weight updates in certain training algorithms.

- *Rectifier*:

$$y(\mathbf{x}) = \max(0, \sum_{i=1}^{n} w_i x_i + b) \tag{3.4}$$

The rectifier is the most popular activation function for deep neural networks [33], which are going to be explained in the next subsection. A unit employing the rectifier is called a rectified linear unit (ReLU). Rectified linear units find applications in computer vision [15]. A smooth approximation to the rectifier is the analytic function:

$$y(\mathbf{x}) = \ln(1 + e^{\sum_{i=1}^{n} w_i x_i + b}) \tag{3.5}$$

Parametric Rectified Linear Unit (PReLU) generalizes the traditional rectified unit:

$$y(\mathbf{x}) = \max(\sum_{i=1}^{n} w_i x_i + b, a(\sum_{i=1}^{n} w_i x_i + b)) \tag{3.6}$$

This modified rectified linear unit used in neural networks outperformed the best algorithms for image classification [18].

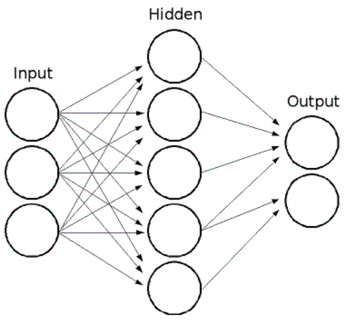

Figure 3.2: Neural network with 3 input neurons, 1 hidden layer with 5 neurons and 2 output neurons. [1]

More complicated structures of neural networks along with their power, advantages and disadvantages are presented next.

3.2.2. Deep Neural Networks

One of the most striking facts about neural networks is that they can compute any function [21]. No matter what the function $f(\mathbf{x}) \in \mathbb{R}^m$, there is guaranteed to be a network so that for every possible input $\mathbf{x} \in \mathbb{R}^n$ the value $y(\mathbf{x})$ which is the output of the network satisfies the following condition: $|f(\mathbf{x}) - y(\mathbf{x})| < \varepsilon$, where $\varepsilon > 0$. This result holds even if the network is restricted to have just a single hidden layer, where smaller ε requires bigger number of neurons. However, there are three conditions which the activation function $y(\mathbf{x})$ must fulfill. $\lim_{x\to\infty} y(\mathbf{x})$ and $\lim_{x\to-\infty} y(\mathbf{x})$ have to exist and be limited. Moreover, these two limits have to be different. If these conditions are satisfied we can state that NNs have *universality* [43].

[1]Source: http://docs.opencv.org/2.4/_images/mlp.png

Deep Neural Networks (DNNs) are distinguished from single hidden layer NNs by their depth, the number of hidden layers through which data passes in a multi-step process of pattern recognition (example of a DNN is shown in Figure 3.3). In DNNs, each layer of neurons trains on a distinct set of features based on the previous layer's output. Further it is advanced into the neural network, the more complex the features neurons can recognize, since they aggregate and recombine features from the previous layer. DNNs have a hierarchical structure which makes them particularly well adapted to learn the hierarchies of knowledge that are useful in solving real-world problems. When attacking problems such as image recognition, it helps to use a system that understands not just individual pixels, but also increasingly more complex concepts: from edges to simple geometric shapes, all the way up through complex, multi-object scenes. To conclude, DNNs are capable of discovering latent structures within unlabeled, unstructured data, which is the vast majority of data in the world [8].

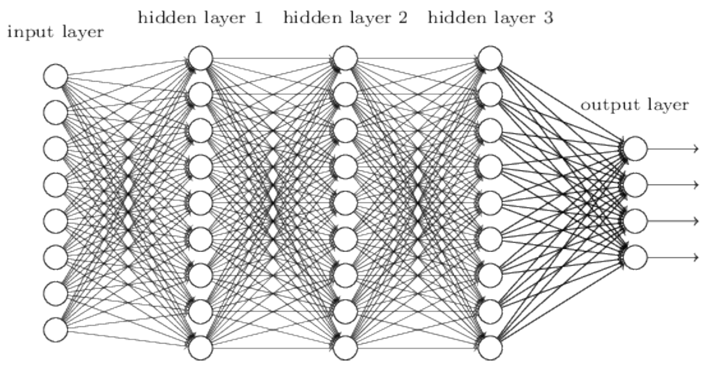

Figure 3.3: Example of Deep Neural Network architecture with 3 hidden layers of neurons [37]. The input layer consists of 8 neurons, each hidden layer contains 9 neurons and the output layer has 4 neurons.

However, even though the power of DNNs is great they are extremely computationally expensive. The more complicated the problem is, the higher the number of parameters which should be set by the network. This computation is requiring time and memory resources.

Once a NN has been structured for a particular application, the NN has to be trained. A training of NNs is a process of determining the weights values. To start this process the initial weights are chosen randomly. Then, the training, or learning, begins.

There are two approaches to training: supervised and unsupervised. Supervised training involves a mechanism of providing the NN with the desired output either by manually "grading" the network's performance or by providing the desired outputs with the inputs. The NN then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the NN. In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data.
The set of data which enables the training is the training set, as explained in Section 3.1. During the training of a NN the same set of data is processed many times as the connection weights are refined. If a network can not solve the problem, parameters of the NN have to reviewed, e. g., the number of layers, the number of neurons per layer, the connections between the layers, the activation functions and the initial weights.

There are many algorithms used to implement the adaptive feedback required to adjust the weights during training. The most common technique is backward-error propagation, known as backpropagation. The Stochastic Gradient Descent is a popular algorithm for training a wide range of models in machine learning, including (linear) support vector machines, logistic regression and graphical models. When combined with the backpropagation algorithm, it represents the standard algorithm for the training of artificial neural networks. Basics of stochastic gradient descent algorithm with backpropagation are presented next.

Stochastic Gradient Descent

Stochastic gradient descent is an efficient method for learning a linear, discriminative model by minimizing a convex loss function [4]. It was revived in the context of large-scale learning [5] and performs well for sparse and high-dimensional data. Firstly a simple supervised learning setup is considered. Each example z is a pair $(\mathbf{x}, y)$ composed of an arbitrary input $\mathbf{x} \in \mathbb{R}^n$ and a scalar output y. A loss function $l(\hat{y}, y)$ is considered and it measures the cost of predicting $\hat{y}$ when the actual answer is y. Then, a family F of functions $f_w(\mathbf{x})$ parameterized by a weight vector $\mathbf{w} \in \mathbb{R}^m$ is chosen. We seek the function $f \in F$ that minimizes the loss $Q(z, \mathbf{w}) = l(f_w(\mathbf{x}), y)$ averaged on the examples [43].

$$E(f) = \int l(f(\mathbf{x}), y) dP(z) \tag{3.7}$$

where $dP(z)$ represents the distribution of examples which is often unknown.

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} l(f(x_i, y_i)) \tag{3.8}$$

The empirical risk $E_n(f)$ measures the training set performance. The expected risk $E(f)$ measures the generalization performance, the expected performance on future examples. The statistical learning theory justifies minimizing the empirical risk instead of the expected risk (in case of unknown $dP(z)$) when the chosen family F is sufficiently restrictive [51] .

It has been proposed that minimization of empirical risk $E_n(f)$ is computed by *gradient descent*. Since this was too complicated a simplification was proposed. Instead of computing a gradient of $E_n(f)$ exactly, at each iteration estimates of this gradient are made on the basis of a single randomly picked example z_t. Each iteration updates the weights w:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \gamma_t \nabla_w Q(z_t, \mathbf{w}_t) \tag{3.9}$$

where γ_t is an adequately chosen learning rate.
However, we need an efficient way for calculation of $\nabla_w Q(z_t, \mathbf{w}_t)$ in Equation (3.9). In a general feed-forward network, each neuron computes a weighted sum of its inputs of the form:

$$a_j = \sum_{i=1}^{n} w_{ji} y_i \tag{3.10}$$

where y_i is the activation of a neuron, or input, that sends a connection to a neuron j , and w_{ji} is the weight associated with that connection. In Section 3.2.1 y_i was denoted as x_i since the meaning of this variable was considered only as an input to the neuron. In this section we considering it as an output of the previous neuron, so for generality we use y_i notation. It is defined that

$$y_j = f(a_j) \tag{3.11}$$

the output of a neuron is in general the value of a *nonlinear* activation function of its weighted inputs. For each pattern in the training set, we suppose that we have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output neurons in the network with Equations (3.10) and (3.11). This process is known as *forward propagation* because it can be regarded as a forward flow of information through the network.

We consider the evaluation of the derivative of $Q(z_t, \mathbf{w}_t)$ (for simplicity Q will be used instead of $Q(z_t, \mathbf{w}_t)$ in the following text) with respect to a weight w_{ji}. Note that Q depends on the weight w_{ji} only via the summed input a_j to neuron j. Therefore we can write [2]:

$$\frac{\partial Q}{\partial w_{ji}} = \frac{\partial Q}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} \tag{3.12}$$

A new notation is defined:

$$\delta_j := \frac{\partial Q}{\partial a_j} \tag{3.13}$$

where δ's are often referred to as *errors*. From Equation (3.10) we can derive:

$$\frac{\partial a_j}{\partial w_{ji}} = y_i \ . \tag{3.14}$$

We can obtain:

$$\frac{\partial Q}{\partial w_{ji}} = \delta_j \cdot y_i \ . \tag{3.15}$$

To evaluate the δ 's for hidden neurons, we make use of the chain rule for partial derivatives:

$$\delta_j \equiv \frac{\partial Q}{\partial a_j} = \sum_k \frac{\partial Q}{\partial a_k} \frac{\partial a_k}{\partial a_j} \tag{3.16}$$

where the sum runs over all neurons k to which neuron j sends connections. Finally, from previous equations a *backpropagation* formula is obtained [2]:

$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k \tag{3.17}$$

which tells that the value of δ for a particular hidden neuron can be obtained by propagating the δ's backwards from neurons higher up in the network [43].
In this section we explained the general framework of DNNs. In the next subsection a special type of deep neural networks suitable for image applications is explained.

3.2.3. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are biologically inspired variants of NNs. From Hubel and Wiesel's early work [23] on the animal's visual cortex , we know the visual cortex contains a complex arrangement of cells. These cells are sensitive to small subregions of the visual field, called a receptive field. The subregions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well suited to exploit the strong spatially local correlation present in natural images. Additionally, two basic cell types have been identified: Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern. Inspired by this discovery CNNs were developed and proposed as a structure to deal with images as inputs [43]. The example of the CNN is shown in Figure 3.4.

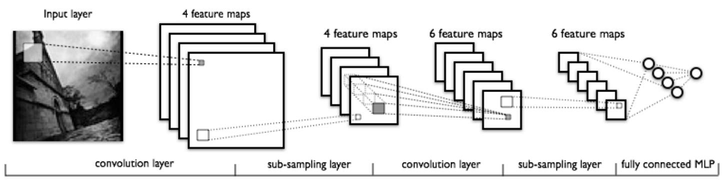

Figure 3.4: Example of Convolutional Neural Network. [2]

CNN architectures make the explicit assumption that the inputs are images, which allows encoding of certain properties into the architecture. These properties then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network [26]. CNNs are, like ordinary NNs, made up of neurons that have learnable weights and biases. Unlike the regular one dimensional layers of NN the layers of a CNN have neurons arranged in three dimensions: width, height and depth. A simple CNN is a sequence of layers. There are three main types of layers to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. Each of the layers is explained.

[2]Source: `deeplearning.net/tutorial/_images/mylenet.png`

Convolutional Layer

The Convolutional Layer (CL) is a core building block of a CNN that does most of the computational heavy lifting. The CL's parameters consist of a set of learnable filters. Filters are 2-dimensional (width and height of CL) arrangements of neurons. Every layer is made of multiple filters, and every filter is extracting a specific feature from an input. A CL is then 3-dimensional arrangement of neurons where 2-dimensional filters are arranged along a third dimension (which creates the depth of a CL). Every filter is connected to the small local receptive field, but it is connected to the full depth of the input, as shown in Figure 3.5. By a depth of the input is meant to 3 color channels of picture (1 picture has 3 color channels which are 2-dimensional). During the forward pass, each filter slides across the width and height of the input and a computation of the dot products between the entries of the filter and the input at any position is performed. As the filter slides over the width and height of the input, it produces a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network learns filters that are active when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer [26].

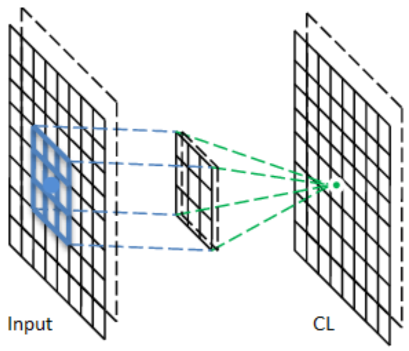

Figure 3.5: Representation how a single filter in CL works. It is shown how a single neuron (green point) in CL is connected (green connections) to a local receptive field (blue field) of the input.

When dealing with highly dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous layer. Each neuron is connected only to a local region of the input. The extent of the connectivity along the depth axis is always equal to the depth of the input. The connections are local in space (along width and height), but always full along the entire depth of the input [26].

Four hyperparameters control the size of the output volume:

- *Depth of CL*: The number **K** of filters we would like to use, each extracting a different feature of the input. For example, if the first CL takes as input the raw image, then different neurons along the depth dimension (different filters) may be activated in the presence of various oriented edges, or blobs of color.
- *Spatial extent*: The size of the receptive field **F** connected to a single neuron of a filter. This 2-dimensional field of the input which is locally connected. It is of $F \times F$ size.
- *Stride*: The sliding step size **S**. When the stride is 1 then receptive fields move (connected to the neurons of filters) move one pixel at a time. When the stride is 2 (or uncommonly 3 or more which is rare in practice) then the filters jump 2 pixels at a time. This will produce smaller output spatially.
- *Zero-padding*: Sometimes it will be convenient to pad the input with zeros around the border. The nice feature of zero padding (of size **P**) is that allows a control of the spatial size of the output volumes (most commonly to exactly preserve the spatial size of the input volume so the input and output width and height are the same).

Assuming an input of size $W_1 \times H_1 \times D_1$, where W_1 is width, H_1 height and D_1 depth of the input. In case of the image as an input $W_1 \times H_1$ are the dimensions of the image, and D_1 is 3 according to 3 color channels. Nevertheless, the other dimensions of input are possible as well. The CL will produce an output of size $W_2 \times H_2 \times D_2$ where:

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$
- $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- $D_2 = K$

Inspired by the complex cells found at animals which are locally invariant to the exact position of the pattern a reasonable assumption is made: If one feature is useful to compute at some spatial position (x_1, y_1), then it should also be useful to compute at a different position (x_2, y_2). The neurons in each filter are constrained to use the same weights and bias. Nevertheless, sometimes the parameter sharing assumption does not make sense. This is especially the case when the input images have some specific centered structure where completely different features should be learned on one side of the image than another. A practical example is when the input are faces that have been centered in the image. In that case it is typical to relax the parameter sharing scheme [43].

Pooling Layer

It is common to periodically insert a Pooling Layer in-between successive CLs in a CNN architecture. As in the case of CL, pooling layer is made of filters (2-dimensional arrangements of neurons) stacked along third axis. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The pooling layer operates independently on every filter (depth slice) of the input (1 color channel in the case of images as inputs) and resizes it spatially, using the $\max$ operation. Max-pooling can be seen as a way for the network to ask whether a given feature is found anywhere in a region of the image [37].

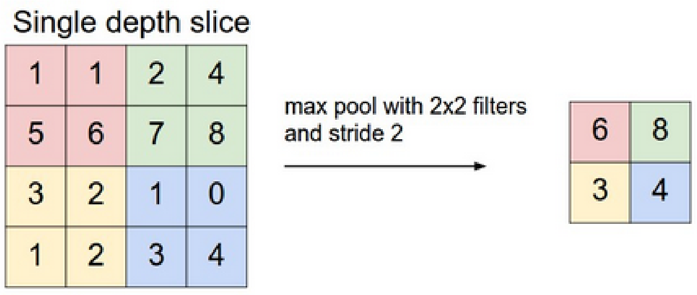

Figure 3.6: Representation how a single filter of size 2×2 with a *stride* of length 2 in MAX-pooling layer works [26].

The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 which downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations (an example is shown in Figure 3.6). In this case, every $\max$ operation takes maximum over 4 numbers. The depth dimension remains unchanged [43].

In addition to $\max$ pooling, the pooling units can also perform other functions, such as *average pooling* or *even* L_2*- norm pooling*. Average pooling was often used historically but has recently fallen out of favor compared to the $\max$ pooling operation, which has been shown to work better in practice [26].

Fully-Connected Layer

Neurons in a Fully-Connected Layer have full connections to all activations in the previous layer, as in regular NNs, and as it is shown in Figure 3.3. It is worth noting that the only difference between fully-connected and CLs is that the neurons in the CL are connected only to a local region of the input, and that many of the neurons in a CL share parameters.

Feature Extraction

Figure 2.4 shows that block 2 of Tubby system performs the feature extraction. The process which is going to be presented in this section is known as a deep neural networks visual feature generation process. Recently, it is shown in an experiment [14] that this process improves the performances of the system, compared with the feature extraction process already existing on Tubby [29]. The old feature generation process is replaced by the deep neural networks visual feature generation explained here. A process of data transformation in CNNs, which are used for classification of images, can be divided into two major phases: *feature extraction* and *classification*. Unlike many other learning algorithms, CNNs combine both in the learning process. Figure 3.7 shows which part of the network performs the feature extraction and which the classification.

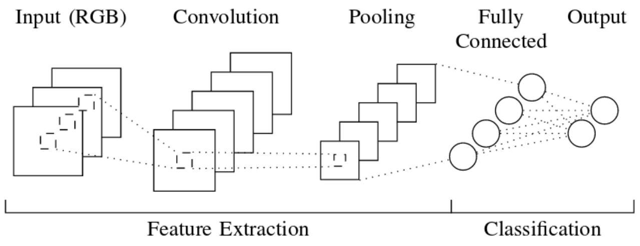

Figure 3.7: Schematic diagram of a CNN. The network comprises five different layers. Both feature extraction and classification are learned during training [19].

The activations which are the output of CLs can be interpreted as visual features (characteristics of an image). CNN models have been used as feature extractors by removing the output layer (which outputs class scores). In particular, a pre-trained CNN can be used as a generic feature extractor for other datasets [44]. Finally, the activations which are interpreted as visual features are given in a form of n-dimensional vector $\mathbf{x} \in \mathbb{R}^n$ which is called a feature vector. Feature vectors lie in n-dimensional feature space.

Application of CNNs

CNNs can be used for image [6], video [27] and speech analysis [35]. Potentially, they are suitable for other purposes when inputs are highly dimensional data. They can classify images, cluster them by similarity, and perform object recognition within scenes. CNNs can identify faces, individuals, street signs, eggplants, platypuses and many other aspects of visual data [8].
In summary, DNNs can deal with complicated multi-dimensional real-word problems. Downside of the full-connected DNN architectures is an enormous number of parameters. Convolutional neural networks resolve this problem. They are the most suitable solution for dealing with high-dimensional inputs as images, which is the case of interest for the thesis.

3.3. k Nearest Neighbour

This section explains k-Nearest Neighbors (kNN) classifier. Section 2.2 states that some changes are introduced to the original Tubby system. These changes are introduced in order to determine the setting which is going to be considered further. This new setting is considered due to easier assessment of the communication improvements. Section 2.2 states that a new process of learning objects is used, instead of the one already existing on Tubby system. The new process is kNN classification and it is explained next.

kNN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e. g., distance functions). kNN is a non parametric lazy learning algorithm and it has been used in statistical estimation and pattern recognition already in the beginning of 1970's [39]. A technique is said to be non parametric when it does not make any assumptions on the underlying data distribution. kNN is usually used in batch settings where all training data is available at once, but with small adaptations it can be used in an online setting as well.

kNN assumes that the data is in a feature space. More explicitly, the data points are in a metric space. The data can be scalars or multidimensional vectors. In this thesis the data is obtained from the feature extraction process (Subsection 3.2.3). The data are n-dimensional vectors. Each of the training data consists of a set of vectors $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^n, i = 1,..,m\}$ and a class label associated with each vector $L = \{l_1, \dots, l_m\}$.

kNN for Classification

A test instance $\mathbf{x} \in \mathbb{R}^n$ is classified by a majority vote of its neighbors. Nearest neighbours, k of them, are ordered by the distance and defined $\mathcal{X}^s = \{ \mathbf{x}_i^s \in \mathbb{R}^n, i = 1, \dots, k\}$ and their corresponding set of labels is $L^s = \{l_1^s, ..., l_k^s\}$. The test instance $\mathbf{x}$ is assigned to the class most common among its k nearest neighbors measured by a distance function. If $k = 1$, then the case is simply assigned to the class l_1^s of its nearest neighbor. Equation (3.18) denotes the distance functions between the test instance $\mathbf{x} = (x_1, \dots, x_n)$ and a neighbour $\mathbf{x}^s = (x_1^s, \dots, x_n^s)$ in the n-dimensional space.

$$\begin{array}{cc} \text{Euclidean} & \sqrt{\sum_{i=1}^{n}(x_i - x_i^s)^2} \\ \text{Manhattan} & \sum_{i=1}^{n} |x_i - x_i^s| \\ \text{Minkowski} & (\sum_{i=1}^{n}(|x_i - x_i^s|)^q)^{\frac{1}{q}} \end{array} \tag{3.18}$$

where q is the order of the Minkowski distance.

It should also be noted that all three distance measures are only valid for continuous variables. In the instance of categorical variables the Hamming distance [3] must be used.
In the batch setting, choosing the optimal value for k is best done by first inspecting the data. In general, a large k value is more precise as it reduces the overall noise but there is no guarantee. Cross-validation is another way to retrospectively determine a good k value by using an independent dataset to validate the k value. Historically, the optimal k for most datasets has been from $k = 3$ to $k = 10$ [11]. That generally produces better results than $k = 1$.

Although classification remains the primary application of kNN, it can be used for density estimation as well [11]. kNN is a versatile algorithm and is used in a big number of fields. Its applications vary from image [1] and text [48] classification, computational geometry [7] to graphs [38]. Some unusual applications cover content retrieval [9], gene expression [10], protein-protein interaction [30] and 3D structure prediction [28].

Section 3.1.1 presents the idea of online learning and Stability-Plasticity dilemma which affects the online learning systems. The kNN algorithm presented here is used in the online setting on Tubby system. An advantage of this algorithm in an online setting is that it is less sensitive to the effects of the dilemma than a lot of other online algorithms. The reason is that it preserves all training data.

3.4. Outlier Detection

In order to understand self-evaluation techniques which we propose in the next chapter an outlier detection is explained in this section.

Data analysis tries to extract simple representations of highly dimensional data which best depicts the complicated underlying structure. An outlier is a class sample which is distant from the other class samples, like shown in Figure 3.8. Most of the real world data contains outliers which are a consequence of different processes like measurement errors or miss-labeled samples. Therefore it is necessary to remove outliers from the data to avoid representation results which yield wrong conclusions. In the statistics literature, a stress is put on the problem of outlier detection in univariate data. In unvariate data objects are ordered using one-dimensional measure. In the case of multivariate data, ordering of the data is not that trivial, and more complicated models have to be applied.
From the ML field some heuristic methods originate, for instance, neural network models [25] or models which are inspired by the support vector classifiers [46]. They avoid very difficult computations of density estimation usually required in multivariate data, and directly fit a decision boundary around the data, but are often not simple to implement and optimize. The outputs of two-class classifiers can also be used for outlier detection [52], hence focusing on the outliers from the perspective of the classification problem.

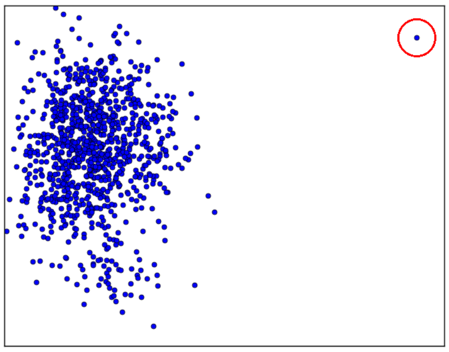

Figure 3.8: Outlier of the blue class (marked in red).

Notice that the methods mentioned above provide an ordering of samples in a data set, according to their typicality. Very untypical samples are candidates to be labeled as outliers. On the other hand, it is of similar importance to detect the prototypical samples in a data set. The prototypical samples are often useful to gain a better understanding of the data. Furthermore, simple indices are proposed based on nearest neighbors that allow an ordering of the data from outliers to prototypes [17]. This ordering provides a possibility for setting a threshold based on which is determined whether a data point is an outlier or not.

3.4.1. Indices for Ordering

A set of m data points from the n-dimensional Euclidean space, $\{\mathbf{x}_1, ..., \mathbf{x}_m\} \subset \mathbb{R}^n$, with the Euclidean norm, $||\mathbf{x}|| = \sqrt{\mathbf{x}^T\mathbf{x}}$ and the Euclidean metric is considered. Other metrics can be easily incorporated in this framework. For a test data point $\mathbf{x} \in \mathbb{R}^n$, let $\{\mathbf{x}_1^s(x), ..., \mathbf{x}_k^s(x)\} \in \{\mathbf{x}_1, ..., \mathbf{x}_m\} \subset \mathbb{R}^n$ be its k nearest neighbors among the given data points, ordered by their distance to the test point. Three indices for each point $\mathbf{x} \in \mathbb{R}^n$ dependant on these neighbors are defined. The ordering process uses these indices. The perception of the data is influenced by the choice of k: if k is chosen too small the focus is too local, if k is too large it is too global [17].

- Kappa $\kappa(\mathbf{x})$

  A kNN density estimator assesses the density at a particular point by calculating the volume of the smallest ball centered at that point which contains its k nearest neighbors. Unfortunately, the estimate is not always very accurate if the number of data points is small or the dimensionality is high. However, outlier detection does not require the actual density. In order to decide whether a data point is an outlier or not, an approximate estimate is a sufficient indicator. The first index therefore represents the k nearest neighbor density estimator: $\kappa(\mathbf{x})$ given in Equation (3.19) is the radius of the smallest ball centered at $\mathbf{x}$ containing its k nearest neighbors, i.e. the distance between $\mathbf{x}$ and its k-th nearest neighbor [17].

$$\kappa(\mathbf{x}) = ||\mathbf{x} - \mathbf{x}_k^s(x)|| \tag{3.19}$$

  In dense regions κ is small and in sparse regions κ is large, making it a good candidate for an outlier.

- Gamma $\gamma(\mathbf{x})$

  The index κ, however, considers the distance to the k-th nearest neighbor, but it ignores the distances to the closer neighbors. This suggests a refined index that takes the distances to all k nearest neighbors into account: $\gamma(\mathbf{x})$ given in Equation (3.20) is the average distance to the k nearest neighbors of $\mathbf{x}$ [17].

$$\gamma(\mathbf{x}) = \frac{1}{k}\sum_{i=1}^{k} ||\mathbf{x} - \mathbf{x}_i^s(x)|| \tag{3.20}$$

This index enables distinction of the two exemplary situations depicted in Figure 3.9, where the value of κ is the same in both situations, because the k-th ($k = 5$) nearest neighbor of a has both times the same distance to a, although the neighborhood on the right is denser. By exploiting all distances, $\gamma(a)$ can distinguish these situations.

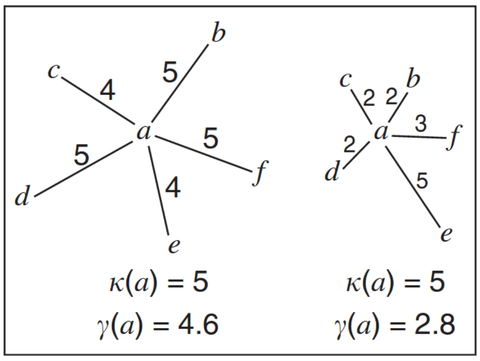

Figure 3.9: Distinction of the dense and sparse region based on γ and κ value [17]. The value of κ is the same in both situations presented on the image, because the k-th ($k = 5$) nearest neighbor of a has both times the same distance to a, although the neighborhood on the right is denser. $\gamma(a)$ is smaller for the case on the right and therefore able to distinguish these situations.

- Delta $\delta(\mathbf{x})$

Looking at the Figure 3.10, two situations that γ cannot distinguish (and κ as well) can be observed, because the distances from a to its neighbors $\{b, c, d, e, f\}$ are the same in both settings. The directions of neighbors b, c, d, e and f show the crucial difference: on the left, neighbors point approximately into the same direction. On the right, neighbors b, c, d, e and f are spread out in all directions. This information is captured by the Equation (3.21). Delta value $\delta(\mathbf{x})$ depicts the length of the mean of the vectors pointing from $\mathbf{x}$ to its k nearest neighbors [17].

$$\delta(\mathbf{x}) = \|\frac{1}{k}\sum_{i=1}^{k}(\mathbf{x} - \mathbf{x}_i^s(x))\| \tag{3.21}$$

$\delta(\mathbf{x})$ is large if the neighbors are located in the same direction. This happens especially for outliers. Value of $\delta(\mathbf{x})$ enables distinction of points in regularly filled sparse regions (all neighbors in different directions) from points which are outliers (all neighbors in the same direction).

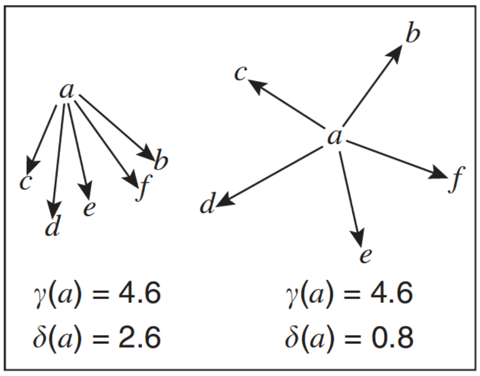

Figure 3.10: Distinction of the sparse and outlier region based on δ value [17]. The distances from a to its neighbors $\{b, c, d, e, f\}$ are the same in both settings. The directions of neighbors b, c, d, e and f show the difference: on the left, neighbors point approximately into the same direction. On the right, neighbors b, c, d, e and f are spread out in all directions. This information is captured by $\delta(a)$.

Please note that $\delta(\mathbf{x})$ is bounded by $\gamma(\mathbf{x})$ which itself is bounded by $\kappa(\mathbf{x})$, $\forall \mathbf{x} \in \mathbb{R}^n$ (as depicted in Equation (3.22)).

$$\kappa(\mathbf{x}) \geq \gamma(\mathbf{x}) \geq \delta(\mathbf{x}) \tag{3.22}$$

This means that if $\delta(\mathbf{x})$ is large (implying that $\mathbf{x}$ is probably an outlier) also $\gamma(\mathbf{x})$ is large. On the contrary, if $\delta(\mathbf{x})$ is small, then $\gamma(\mathbf{x})$ does not need to be small. Therefore, in contrast to $\delta(\mathbf{x})$, a point from a sparser region can be misjudged to be an outlier by $\gamma(\mathbf{x})$. The analog discussion holds for $\gamma(\mathbf{x})$ and $\kappa(\mathbf{x})$.

Based on these indices which we derive for every data point $\mathbf{x}$ we can infer general regions of data $\mathcal{X} = \mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, m$. The concept of regions is explained next.

Concept of regions

The two-dimensional data ($\mathbf{x} \in \mathbb{R}^2$) is presented on the left image of Figure 3.11. For every data point $\mathbf{x}$ shown in the image it is possible to determine in which region lies, the dense region of the data, the sparse region of the data or it is an outlier, using indices explained in Subsection 3.4.1. This is possible since the indices allow ordering of the data from prototypes (dense regions) to outliers. Based on these findings specific regions in the space can be inferred: dense, sparse and outlier region. Depending on the position, the data point $\mathbf{x} \in \mathbb{R}^2$ can be in one of these regions. The right image of Figure 3.11 shows regions of the two-dimensional space belonging to the data presented on the left.

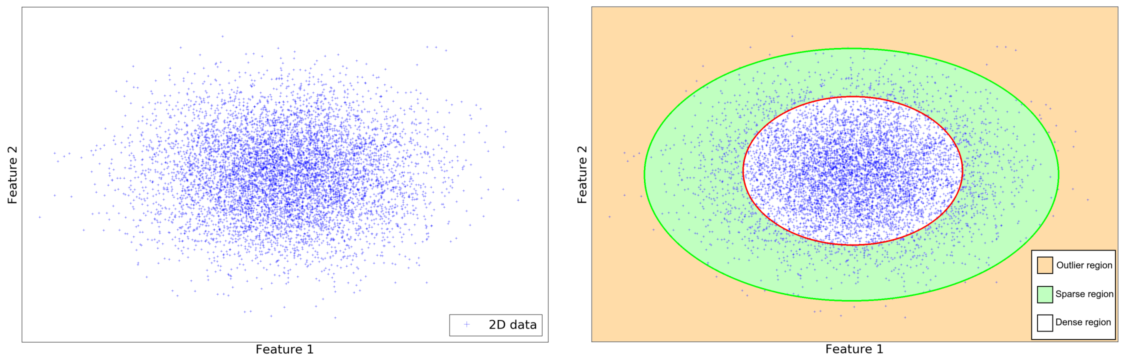

Figure 3.11: Image on the left represents 2D data. Image on the right captures the regions in the space, with orange denoted the outlier region, green the sparse region and with white denoting the dense region.

Note that this example refers to the two-dimensional space, since it is the most intuitive way to demonstrate the regions. However, these conclusions are equally applicable in the case of n dimensional space.

The subsequent section introduces the assessment of classifiers field.

3.5. Assessment of Classifiers

Section 2.2 lists the possible improvements of the Tubby system which are targeted for the thesis. The essential goal of the improvements is the assessment of classifiers. Therefore, this section explains the fundamentals on this topic.

There are two main approaches found in the literature to assess the performance of classifiers: the Confusion Matrix and the Receiver Operating Characteristic curve. Sections 3.5.1 and 3.5.2 give an overview of these methods, respectively, while Section 3.5.3 presents some simpler measures [43].

3.5.1. Confusion Matrix

A confusion matrix is a table that is used to describe the performance of a classifier on a set of test data for which the true values are known [49]. Each column of the matrix represents a predicted class while each row represents an actual class.
The classifier result can be a real value (continuous output), and in this case the classifier boundary between classes must be determined by a threshold value (for instance, to determine whether a person has hypertension based on a blood pressure measure). Or it can be a discrete class label, indicating one of the classes.
We consider a two-class prediction problem (binary classification), in which the outputs are labeled either as positive p or negative n. There are four possible outcomes from a binary classifier. If the

outcome from a prediction is p and the actual value is also p, then it is called a true positive (TP). However, if the actual value is n then it is said to be a false positive (FP). Conversely, a true negative (TN) has occurred when both the prediction output and the actual value are n, and a false negative FN is when the prediction output is n while the actual value is p. These four possible outputs constitute a confusion matrix shown in Table 3.1.

Data Class	Classified as p	Classified as n
p	TP	FN
n	FP	TN

Table 3.1: Confusion matrix for binary classification [47].

Extension to multi-class problems is simple and straightforward. If a classification system has been trained to distinguish between cars, airplanes and ships a confusion matrix will summarize the classification results. Assuming a sample of 27 vehicles: 8 cars, 6 airplanes, and 13 ships, the resulting confusion matrix is shown in the Table 3.2. In this confusion matrix, of the 8 actual cars, the system predicted that 3 are airplanes, and of 6 airplanes, it predicted that one is a ship and 2 are cars. It can be seen from the matrix that the system has trouble distinguishing between cars and airplanes, but can make the distinction between ships and other types of vehicles pretty well. All correct predictions are located in the diagonal of the table, so it is easy to visually inspect the table for errors, as they will be represented by values outside the diagonal [43].

Actual\Predicted	Cars	Airplanes	Ships
Cars	5	3	0
Airplanes	2	3	1
Ships	0	2	11

Table 3.2: Confusion Matrix Example. In this confusion matrix, of the 8 actual cars, the system predicted that 3 are airplanes, and of 6 airplanes, it predicted that one is a ship and 2 are cars. It can be seen from the matrix that the system has trouble distinguishing between cars and airplanes, but can make the distinction between ships and other types of vehicles pretty well. All correct predictions are located in the diagonal of the table, so it is easy to visually inspect the table for errors, as they will be represented by values outside the diagonal.

Given a distribution of real positive instances and real negative instances as given in Figure 3.12 we can derive a new analysis for the assessment of classification.

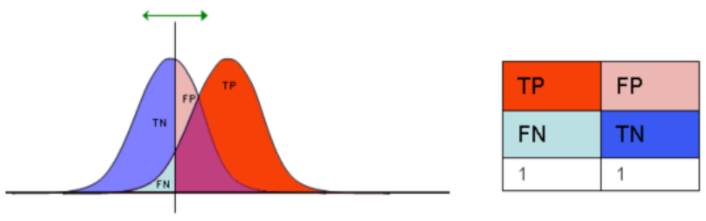

Figure 3.12: A distribution of positive and negative examples and a threshold are shown. The x-axis represents predicted probabilities, and the y-axis represents a count of observations. All of the examples right of the threshold are classified as positive and all of the examples left of the threshold are classified as negative.[3]

The following terms are defined as a measure for assessing the quality of a classifier. A schematic view of these measures is given in Table 3.3:

- *Sensitivity*: Probability that an instance will be classified as p when the instance is positive (true positive rate).
- *Specificity*: Probability that an instance will be classified as n when the instance is negative (true negative rate).

[3]Source: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

- *Positive Likelihood Ratio - Precision*: Ratio between the probability of a p classification result given the positive instance and the probability of the n classification given the negative instance.
- *Negative Likelihood Ratio*: Ratio between the probability of a n classification result given the positive instance and the probability of the p classification result given the negative instance.
- *Positive Predictive Value*: Probability that the instance is positive when the classification is p.
- *Negative Predictive Value*: Probability that the instance is negative when the classification is n.

Sensitivity	$\frac{TP}{TP+FN}$	Specificity	$\frac{TN}{TN+FP}$
Positive Likelihood Ratio	$\frac{Sensitivity}{1-Specificity}$	Negative Likelihood Ratio	$\frac{1-Sensitivity}{Specificity}$
Positive Predictive Value	$\frac{TP}{TP+FP}$	Negative Predictive Value	$\frac{TN}{TN+FN}$

Table 3.3: Measures for assessing the quality of classification.

3.5.2. Receiver Operating Characteristic

A receiver operating characteristic (ROC) curve [13] is a commonly used way to visualize the performance of a binary classifier, and an example of it can be seen in Figure 3.13. It is a plot of the True Positive Rate (on the y-axis) versus the False Positive Rate (on the x-axis). To generate the entire ROC curve, the True Positive Rate versus the False Positive Rate should be plotted for all possible classification thresholds [13]. The ROC illustrates the performance of a binary classifier as its discrimination threshold is varied.

A classifier that is good at separating the classes has a ROC curve that is close to the upper left corner of the plot [13]. Conversely, a classifier that is bad at separating the classes will have a ROC that is close to the black diagonal line shown in Figure 3.13. That line essentially represents a random guessing. Classifiers appearing on the left-hand side of the ROC graph, near the x axis, can be seen as *conservative*: they make positive classifications only with strong evidence so they make few false positive errors, but they often have low true positive rates as well. Classifiers on the upper right-hand side of the ROC graph can be seen as *liberal*: they make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates [43].

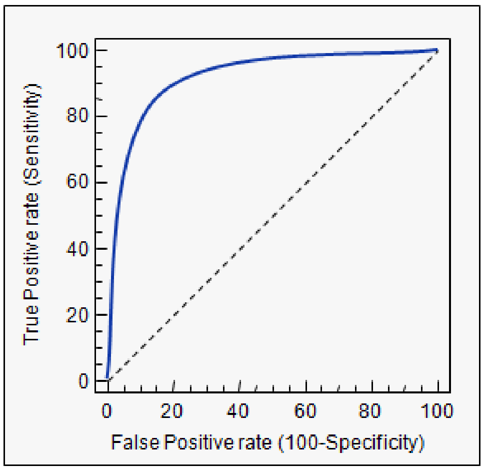

Figure 3.13: ROC graph example. [4]

The benefit of using a ROC curve to evaluate a classifier instead of a simpler metric is that the ROC curve visualizes all possible classification thresholds, whereas the simpler metric only gives a value for a single threshold.

Area under the curve (AUC)

The ROC is used for the quantification of the performance of a classifier. The area under the curve (AUC) [13] is the surface under the blue curve shown in Figure 3.13. The AUC is representing the probability that a classifier will rank a randomly chosen positive observation higher than a randomly chosen negative observation, and accordingly it is a useful metric even for datasets with highly unbalanced classes.

The ROC approach can be extended to classification problems with three or more classes using the *one versus all* approach. In the case of three classes, three ROCs will be created. In the first curve, the first class would be chosen as the positive class, and the other two classes grouped together as the negative class. In the second curve, the second class would be chosen as the positive class, and the other two classes grouped together as the negative class, etc.

A ROC curve can be used for setting the classification threshold. The threshold is set according to a goal which can be the minimization of the False Positive Rate or maximization of the True Positive Rate. What is preferred among these two cases depends on the given situation. As an example assume a classifier which is supposed to be used to predict whether a given credit card transaction might be fraudulent and thus should be reviewed by the credit card holder. For this purpose the threshold might be set to be very low. This results in a lot of false positives (i.e. false detection of the fraud), but it might be considered acceptable because it would maximize the true positive rate and thus minimize the number of cases in which a real instance of the fraud is not detected.

3.5.3. Simple Metrics

Binary Classification

Measures which are most often used for the assessment of binary classification are presented in Table 3.4. Variables used in the Formula field of the table are the values of a confusion matrix introduced in Subsection 3.5.1. It should be mentioned that there are other metrics used in the literature [16, 22, 41] but they are omitted here since they can be derived from the basic measures.

Measure	Formula	Evaluation focus
Accuracy	$\frac{TP+TN}{TP+FN+FP+TN}$	Overall effectiveness of classifier.
Precision	$\frac{TP}{TP+FP}$	Class agreement of the data labels with the positive labels given in the classifier.
Sensitivity	$\frac{TP}{TP+FN}$	Effectivness of a classifier to identify positive labels.
Specificity	$\frac{TN}{FP+TN}$	How effectively a classifier identifies negative labels.
Balanced Accuracy [5]	$\frac{1}{2}(\frac{TP}{TP+FN}+\frac{TN}{TN+FP})$	Classifier's ability to avoid false classification.

Table 3.4: Simple measures for assessing the quality of binary classification [47].

[4]Source: https://www.medcalc.org/manual/roc-curves.php

Multi-class Classification

In the case of a multi-class classification it is more difficult to establish a single measure for the quality of the classifier. The classification can be very accurate for a certain class, but very bad for other classes, so introducing *one versus all* approach (which was already mentioned in Subsection 3.5.2) would give different results for different classes. We need a unified measure(s) for the whole classification process which presents the ability of a classifier to distinguish among multiple classes.
There are two proposed ways for assessing the performance of the multi-class classifier. For an individual class C_i, the well known measures already mentioned are TP_i, FN_i, TN_i, FP_i, Accuracy_i, Precision_i, Sensitivity_i. Quality of the overall classification can be assessed as an average of the same measures calculated for $l_1, \dots, l_h$ (where h is the total number of classes) and this approach is referred as *macro-averaging* [47]. The other way for assessing the quality is *micro-averaging* and it represents the sum of counts to obtain cumulative TP, FN, TN, FP and then calculating a performance measures [47]. It should be noted that well-developed multi-class ROC analysis does not exist yet [32].

Measure	Formula	Evaluation focus
Average Accuracy	$\frac{\sum_{i=1}^{h} \frac{TP_i+TN_i}{TP_i+FN_i+FP_i+TN_i}}{h}$	The average per-class effectiveness of classifier.
Error Rate	$\frac{\sum_{i=1}^{h} \frac{FP_i+FN_i}{TP_i+FN_i+FP_i+TN_i}}{h}$	The average per-class classification error.
Precision_μ	$\frac{\sum_{i=1}^{h} TP_i}{\sum_{i=1}^{h} (TP_i+FP_i)}$	Agreement of the data class labels with those of a classifiers if calculated from sums of per-image decisions.
Sensitivity_μ	$\frac{\sum_{i=1}^{h} TP_i}{\sum_{i=1}^{h} (TP_i+FN_i)}$	Effectiveness of a classifier to identify class labels if calculated from sums of per-image decisions.
Precision_M	$\frac{\sum_{i=1}^{h} \frac{TP_i}{TP_i+FP_i}}{h}$	An average per-class agreement of the data class labels with those of a classifiers.
Sensitivity_M	$\frac{\sum_{i=1}^{h} \frac{TP_i}{TP_i+FN_i}}{h}$	An average per-class effectiveness of a classifier to identify class labels.

Table 3.5: Simple measures for assessing the quality of multi-class classification based on a generalization of the measures in Table 3.4 [47]. The M index stands for macro-averaging. The μ index stands for micro-averaging.

The chapter is concluded with the presented measures. The theoretical fundamentals are presented in order to set the background for methods which are presented next. Chapter 4 proposes methods for the communication improvement of the Tubby platform.

[5] It captures a *single point* on the ROC curve.

4 Proposed Methods

This chapter presents the main ideas for the improvement of the control loop between the user and the Tubby system. The chapter starts with two key questions that have to be answered in order to improve the communication strategy. The questions are outlined in Section 4.1. The following Sections 4.2 and 4.3 propose two measures which answer the questions. Furthermore, these two sections propose testing and training strategies, which use the proposed measures.

4.1. Self-evaluation

The literature dealing with the assessment of classifiers focuses mainly on statistical assessment of the overall performance of the classification. The measures presented in Section 3.5 evaluate the efficiency of classifiers after the prediction. On the other hand, it can be useful to know the confidence of the prediction, before the prediction is made, in online and interactive scenarios. Of special interest is the case when the probability of a mistake is large. The user can utilize the confidence of the prediction in multiple ways. Some examples are:

- The user avoids the use of the classifier in a specific situation if it is not safe
- The user lowers down expectations and disappointment in case of a wrong prediction
- The user provides additional information to the classifier if that can improve the performance
- The user changes the way he/she operates with the classifier
- The user is aware about the limitations of the classifier

The classifier can benefit from knowing the confidence of the prediction in the case where the user provides more information to the classifier, and where a change in the user's behaviour can lead to better predictions.

The exchange of valuable information, like the confidence of the prediction, between the classifier and the user improves the control loop and increases the understanding of the classifier's behaviour. However, determining in advance how large the probability of a mistake is can be a difficult task. In this project we aim for a self-assessment measure which answers the following question: *"How confident am I (the classifier) at making this prediction?"*. Note that it is stated *this prediction* meaning that we focus on a classification in an online scenario, described in Chapter 2.

On the other hand, as explained, classification is a process consisting of a training and testing phase. During the training phase we want to know the quality of the training data which the classifier is receiving, with respect to its generalization performance. Hence, the other direction in which the classification process can benefit from self-evaluation is answering the question *"How beneficial is the information I (the classifier) am getting?"*. In an online scenario a human teacher can affect the quality of training information which will be delivered to the system. The target here is to find an adequate measure of the quality of the received information.

4.2. Testing

The previous section outlines that the target is to find a measure which answers the question: *"How confident am I (the classifier) at making this prediction?"*. In order to provide an answer to this question a deeper analysis is necessary. The following paragraphs explain challenges in predictions, and what are the factors which affect the probability of an error. Our assumption is that objects are presented by multiple instances, i.e. multiple views.

Multiple views of one object translate to multiple feature vectors $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, p\}$. Each feature vector $\mathbf{x} \in \mathbb{R}^n$ corresponds to one image of p images of each object. The indices given in Subsection 3.4.1 provide a way to distinguish regions of the feature space of one object: dense, sparse and outlier region.

Figure 4.1 shows possible relations of two classes of feature vectors belonging to two different objects (Class $\mathcal{X}_1$ and Class $\mathcal{X}_2$) in the feature space. The image on the left shows two classes (objects) which do not share any part of the feature space. Therefore, they are well separated. The image in the middle shows classes which share a part of the feature space. However, both classes have instances in regions which are not shared by the other class. Therefore, they are partially overlapping. The image on the right shows two classes which share the same feature space and they do not have instances outside of the shared region. Therefore, they are totally overlapping. The range of the overlap between classes can vary from none to total overlap.

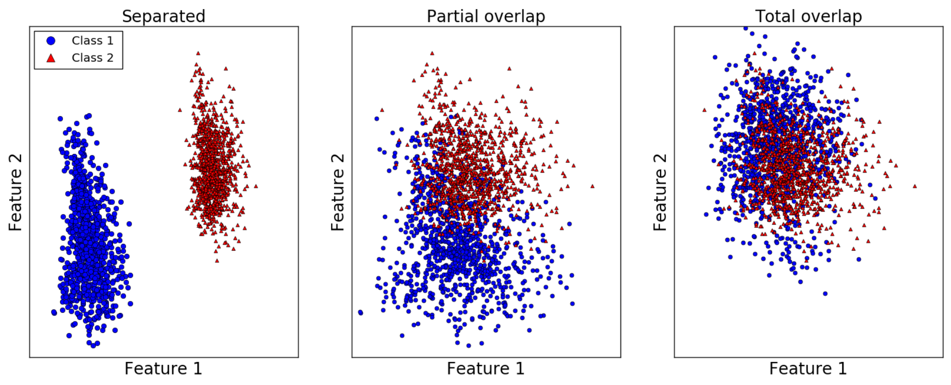

Figure 4.1: Possible relations of two classes in the two-dimensional feature space. The image on the left shows two classes (objects) which do not share any part of the feature space. Therefore, they are well separated. The image in the middle shows classes which share a part of the feature space. However, both classes have instances in regions which are not shared by the other class. Therefore, they are partially overlapping. The image on the right shows two classes which share the same feature space and they do not have instances outside of the shared region. Therefore, they are totally overlapping.

Using the indices κ (3.19), γ (3.20) and δ (3.21) we determine in which relation the two classes are. We take all feature vectors of the blue class (circles) and measure where feature vectors belonging to the other classes are, red class in this case. On the image on the left, all of the feature vectors in the blue class are outliers with respect to the red class. Therefore, they are separated. However, on the image in the middle, some of the blue class feature vectors are in the sparse and dense region of the red class. Accordingly, they are partially overlapping. Similarly, on the image on the right the overlap of the classes in the feature space is total. Note, however, that there exist cases where one class is totally overlapping with the other class while the other class has non-overlapping regions in the feature space.

Taking into account the previously described relations of classes, it can be inferred that the position of a test instance $\mathbf{x} \in \mathbb{R}^n$ in the n dimensional feature space and the relationship of neighboring classes, $\mathcal{X}_i, i \in 1, \dots, s$, determines the probability of an error of the prediction. Namely, lets consider the case where the test instance $\mathbf{x}$ is one of the instances of the red class (triangles) presented in Figure 4.1. On the image on the right, the possibility of an error during prediction is large. However, in the case of the left image wherever the testing instance is in the red class the probability of an erroneous prediction is very small.

In Section 2.3 we state that the goal of the thesis is to find methods which enable the system to provide information about its internal states and communicate that to the user. The main idea stated here is that the system can provide online information about its confidence of the prediction which allows the user to know how large the probability of an error is. The baseline assumption is the following: If users get a feedback of the system's confidence of the prediction they would try to rotate the object such that the confidence is maximized. We propose a graph of the confidence of the prediction running on an online system (like Tubby) while users present an object, as presented in Figure 4.2.

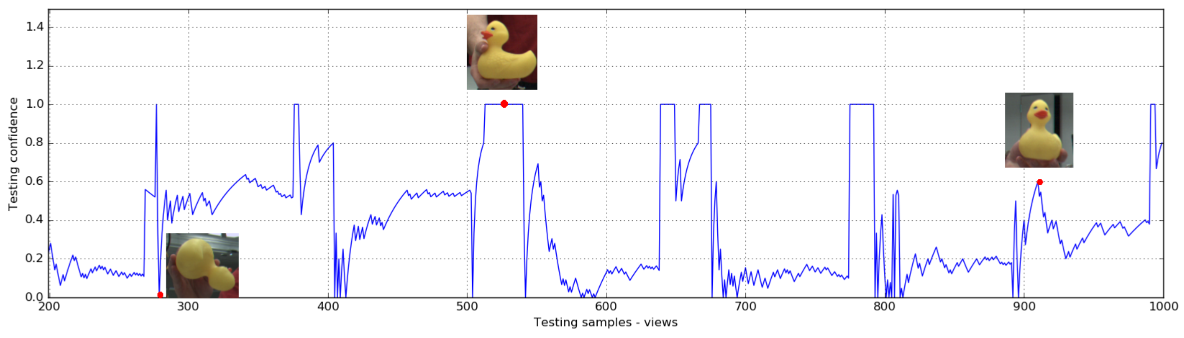

Figure 4.2: Online confidence graph during the testing phase. The user is rotating the yellow duck in front of the system. The system is providing the confidence of the prediction. The red dots capture a single measurement value. The confidence is low when some views of the ducks are seen, but higher when different, more recognizable, views are provided.

Furthermore, Figure 4.3 shows possible positions of a single testing instance **x** (green square) in the feature space of two classes. On the first image the classes are separated. We develop a measure which gives a high value in the case that the testing instance is in the region of only one class, like shown on the first image. However, if there is a partial overlap of the classes, like shown on images 2 and 3, two situations are possible. If the testing instance is in the overlapping region, like shown in image 2, the probability of an error of the prediction is large. Therefore, the confidence should be low. If the testing instance is in the region which is covered by only one class, like in image 3, the confidence should be high. The 4-th image shows total overlapping classes and the confidence should always be low.

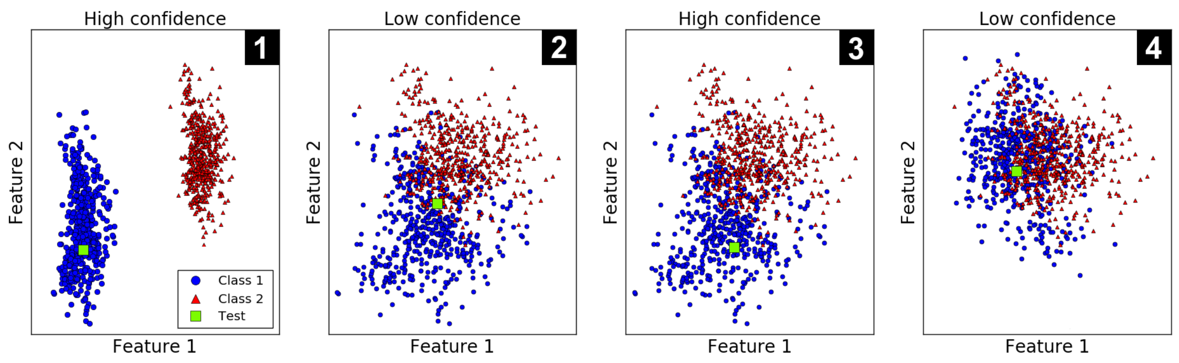

Figure 4.3: Possible positions of a single testing instance (green square) in the feature space of two classes. Image 1 shows separated classes and the testing instance in the feature space of one class. The confidence of the prediction is high. Image 2 shows partially overlapping classes and the test instance in the overlapping region. The confidence of the prediction is low. Image 3 shows the same partially overlapping classes. The test instance is in the region of the feature space of only one class. The confidence of the prediction is high. Image 4 shows totally overlapping classes. Wherever the testing instance is, the confidence is low.

The following Subsections 4.2.1 and 4.2.2 provide two measures which capture the probability of the correct prediction for an object in testing. These subsections propose two testing strategies which use the measures.

4.2.1. Testing Strategy 1

The conclusion drawn from the previous discussion is that we want two essential information. The first information is what classes are neighbours to each test sample **x**, i.e. to which classes neighbours of each test sample belong. The other information is in what region, outlier, dense, or sparse (concept of regions explained in Subsection 3.4.1) of those classes each test sample lies.

We derive a measure of the confidence of the prediction, while an object is in the online process of testing. We consider having a single test sample **x**, as well as a set of test samples $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, q\}$ where we assume that they belong to the same class. Namely, for every incoming test sample (view of an object) the corresponding feature vector **x** is determined. Based on the feature vector, a set of k neighboring feature vectors $\mathcal{X}^s = \{\mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, k\}$ of each test sample **x** and indices κ (3.19), γ (3.20) and δ (3.21) for each test sample **x** are determined. Indices κ, γ, δ are used for determining whether the test sample **x** is in the dense, sparse or outlier region of the feature space of the classifier. We determine the region using an index measure, which is given in Equation (4.1).

$$\text{index} = \kappa \cdot \frac{1}{\gamma} \cdot \frac{1}{\delta} \tag{4.1}$$

Equation (4.2) shows how we use index value to determine the region of the feature space.

$$\text{region} = \begin{cases} \text{index} < t_1, & \text{outlier} \\ t_1 \leq \text{index} < t_2, & \text{sparse} \\ \text{otherwise}, & \text{dense} \end{cases} \tag{4.2}$$

The idea is that the testing samples which are in the outlier region have approximately the same values of κ, γ, and δ. Furthermore, due to this reasoning ratio $\frac{\kappa}{\gamma} \approx 1$. For samples in the outlier region δ value is large (when compared with samples which are in the sparse and dense region). For samples which are in the dense region ratio $\frac{\kappa}{\gamma} \geq 1$ (this holds because of Equation (3.22)). For these samples δ is always small, since the characteristic of the dense region is that neighbours are spread around any sample in all directions. Therefore, index (4.2) should have higher value for samples in the dense region than for the outlier region. Samples which belong to the sparse region usually have values of δ which is between very large and very small. Accordingly, index has a value between the outlier samples and the dense samples values. The process of the region determination is shown in Algorithm 1.

Eventually, we determine the region for each test sample **x** of the testing class $\mathcal{X}$. Final prediction for $\mathcal{X}$ is made as an aggregation of all regions for separate testing samples. Next it is explained how. A stream of testing samples **x** of the testing object $\mathcal{X}$ is shown to the classifier over time. If the majority of the testing samples is within an outlier region of all remaining classes which the classifier learned, like shown on the left image in Figure 4.4, the probability is high that the testing class is new, i.e. that the classifier did not learn it before, and it is predicted as a new class (object). We determine the confidence (c) as the percentage of the testing samples which are in the outlier region of all other classes (blue (circles) and red (triangles) in the case shown). For example, if the classifier obtained q testing samples in total, and p_o of them are in the outlier region of all other classes, the confidence is $c = \frac{p_o}{q}$.

In the previous explanation we say *majority* of the testing class, but actually the testing class can be predicted as new if an arbitrary chosen percentage t_p of the testing class lies in the outlier region of all known classes. Higher values of t_p are more strict, which means that a classifier can make more mistakes in not recognizing a new object. On the other hand, smaller values of t_p can cause a classifier to recognize known objects as new objects. In the following explanations we use *majority* of the testing class to demonstrate the point. However, in these cases as well, the *majority* can be any arbitrary chosen percentage.

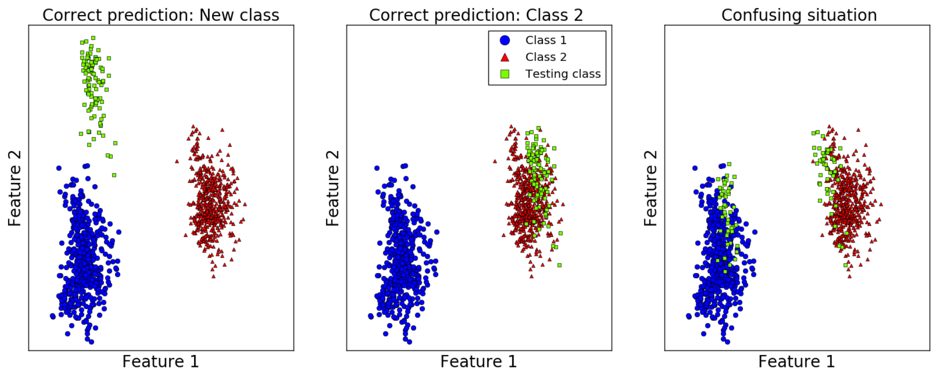

Figure 4.4: Possible positions of the testing class in the feature space which already contains two classes. Image on the left shows the testing class in the outlier region of both classes. The confidence of the prediction that the object is not known is high. Image in the middle shows the testing class totally overlapping with the red class (triangles). The confidence of the prediction, that the testing class is the red class, is high. Image on the right shows a part of the testing class overlapping with the blue class (circles) and another part of the testing class overlapping with the red class (triangles). The confidence of the prediction is low. The probability of a mistake is large.

If the majority of the testing samples is in the dense region of only one class, like shown in the middle image of Figure 4.4, the probability is high that the testing class is equal to the class in which dense region are the testing samples (red class). In the case shown in the middle image the testing class is predicted as the red class. We determine the confidence as the percentage of the testing samples which are inside the dense region. Similarly, if the classifier obtained q testing samples in total, $c = \frac{p_d}{q}$, where p_d is the number of the testing samples in the dense region of the class. However, if some of the testing samples are in the dense region of one class, and other testing samples are in the dense region of the other class, like shown on the right image of Figure 4.4, and their number is approximately equal, the probability of an error of the prediction is high. The testing class can be both red and blue class. The confidence of the prediction is equal to the difference of the percentages of the testing samples being in the dense region of these classes. If the prediction has to be made, the class with bigger percentage of the testing samples is predicted. Again, if the classifier obtains q testing samples, and p_1 of them are in the dense region of the blue class, and p_2 of them are in the region of the red class, the confidence is given in Equation (4.3).

$$c = \frac{\max(p_1, p_2) - \min(p_1, p_2)}{q} \tag{4.3}$$

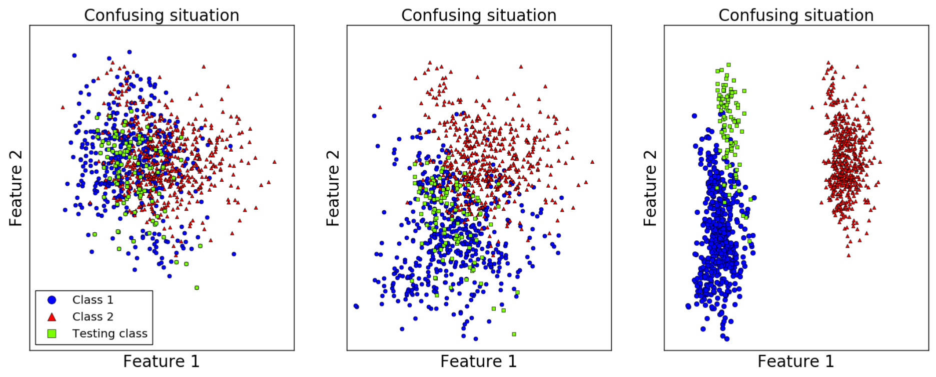

Figure 4.5: Possible positions of the test class feature vectors in the feature space which already contains two classes. Image on the left shows two classes in the memory totally overlapping, and the testing class totally overlapping with them. The confidence in prediction is low. The testing class can be both red (triangles) and blue (circles). Image in the middle shows two classes in the memory partially overlapping and samples of the testing class in the overlapping region and in the region which is covered only by the blue class. The confidence is not high. However, the probability is higher that the testing class is blue than red. Image on the right shows part of the testing class in the region covered by the blue class, and part of the testing class in the outlier region. The confidence is low. The testing class can be blue or a new never seen class.

Likewise, if all of the testing samples are in the overlapping region of two or more classes, like shown on the left image of Figure 4.5, meaning that neighbors of the test samples are in the dense or sparse regions of both or more classes, the probability of an error is high. Firstly, neighboring classes of all testing samples are determined (blue and red class on the left image). Next we determine a set of classes which are among k neighbours for every test sample. For example, if the test sample **x** has two blue class neighbours and three red class neighbours the corresponding set is $\{$blue class, red class$\}$. Further, for the whole testing class we determine the majority class, based on the sets which are determined before. The confidence is determined as the percentage of the testing samples which has majority class in their set of neighbouring classes. If the prediction has to be made, the majority neighbouring class is predicted. Let us assume the classifier obtained q testing samples in total, and p_1 of them have red class among their k neighbours, and p_2 of them have blue class among their k neighbours. Note that a single test sample can have both blue and red neighbours, if $k \geq 2$. This means that $p_1 + p_2 \geq q$ and $p_1 \leq q$, $p_2 \leq q$. Equation (4.4) denotes the confidence value in this case.

$$c = \frac{\max(p_1, p_2)}{q} \tag{4.4}$$

Another problematic situation is when some of the testing samples are in the overlapping region of the two or more classes, but some testing samples are in the region of only one class, like shown on the image in the middle in Figure 4.5. The confidence and the prediction are determined on the same principle as the previous situation. Finally, there is a case when some of the testing samples are in the outlier region of all classes, but some testing samples are in the dense or sparse region of one or more classes, like shown in the right image of Figure 4.5. An outlier region of all classes is in this case considered as the additional class. For every testing sample in this region we consider that k neighbours of the sample are from the additional outlier class. Further the principal from the previous cases can be applied. Algorithm 3 shows the confidence computation (when Algorithm 1 is used for the region computation).

What is not noted in the previous description is when the prediction is made. If it is clear that the confidence is above a specified threshold t during a short period of time the prediction is made. However, if the confidence of the prediction is not high enough the algorithm requires more information (more views of a testing object). Additional views are required until the confidence does not reach a desired value t or the number of views max_v which can be seen by the classifier does not reach a limitation. The number of views which can be seen by the classifier is limited in order to regulate the testing time. In a real online scenario the prediction has to be made after some time even if the confidence is not high enough. Lower values of t mean that the classifier will make predictions even if it is not sure in those, which can cause more errors in predictions. On the other hand, higher values of t will cause the classifier to require more testing views of a testing object more often. However, a number of wrong predictions will be reduced.
Algorithm 4 presents the working mechanism of Testing strategy 1.

Note that for some testing samples this method can wrongly identify the region (dense, sparse or outlier) of the feature space in which the test sample **x** lies. Some samples which are in the sparse region can be wrongly identified in the dense region. The reason is that δ value for these samples can be very small if k neighbours are spread in all directions (when compared to δ values of other points in the sparse region). Moreover, $\frac{\kappa}{\gamma}$ can be large as well (in comparison to other $\frac{\kappa}{\gamma}$ in the sparse region) if the difference between the k-th neighbour distance and the average distance of the test sample **x** is large. In this case, index (4.1) has a large value and the sample is identified in the dense region. Nonetheless, we consider that the number of these mistakes is insufficient to downgrade the outcome of the method. On the other hand, the advantage of this approach is that we do not need to change the thresholds (t_1 and t_2 in Equation (4.2)) when the feature space is changed. The thresholds are set only once. This characteristics makes the method easy to use and once the thresholds are set the accuracy of the method is stable. For example, if the dimensionality of the feature space is changed the method should stay effective as in the primary used feature space.

Algorithm 1 Determine region of the feature space (1): dense, sparse or outlier.

```
Require:
  Test feature vector $\mathbf{x} \in \mathbb{R}^n$
  Training feature vectors of all seen objects $\mathbf{X} = \{\mathcal{X}_i, i = 1, \dots, h\}$
  procedure Region 1
    Compute $\kappa$ (3.19), $\gamma$ (3.20), $\delta$ (3.21) for $\mathbf{x}$
    index = $\kappa \cdot \frac{1}{\gamma} \cdot \frac{1}{\delta}$
    # Determine the location of $\mathbf{x}$ in the feature space
    if index < $t_1$ then
      return $\mathbf{x}$ is in outlier region of $\mathbf{X}$
    else
      if index < $t_2$ then
        return $\mathbf{x}$ is in sparse region of $\mathbf{X}$
      else
        return $\mathbf{x}$ is in dense region of $\mathbf{X}$
      end if
    end if
  end procedure
```

4.2.2. Testing Strategy 2

Testing strategy 2 is very similar to the Testing strategy 1. It represents the same concept. However, the way of computing in which regions of other classes the testing feature vectors lie is different. In the previous case we use index which is captured in Equation (4.1). The disadvantages of the previous method are pointed out. Therefore, in the Testing strategy 2 we define index differently, given in Equation (4.5), and additionally use γ value for the determination of the feature space region of a test sample.

$$\text{index} = \delta \cdot \frac{1}{\gamma} \tag{4.5}$$

The determination of the region using γ (3.20) and index (4.5) is given in Equation (4.6).

$$\text{region} = \begin{cases} \gamma > t_1, \text{ index} > t_2, & \text{outlier} \\ \gamma > t_1, \text{ index} \leq t_2, & \text{sparse} \\ \text{otherwise}, & \text{dense} \end{cases} \tag{4.6}$$

The idea behind this is that samples which are in the outlier region have γ and index (4.5) larger than samples which are in the sparse or dense region. The reasoning is, again, outliers have distant neighbours and approximately the same δ and γ values (for an outlier which is infinitely far away from its class δ and γ are equal). For a sample in the sparse region γ is still large since k neighbours are more distant than in the dense region. However, δ value of samples in the sparse region are smaller than for the outliers because neighbours are spread in all directions and therefore index (4.5) is smaller. On the other hand, γ has to be small for samples in the dense region since neighbours are close. With this method we avoid the problem mentioned in the previous strategy, wrong classification of the samples in the sparse to the dense region. However, the disadvantage of this method is that it is dependant on the feature space, and the threshold t_1 in Equation (4.6) has to be changed for every change of a characteristics of the feature space, e. g., the dimensionality.

Note that besides index definition, the thresholds defined in Equation (4.5) are different than in Equation (4.2). Algorithm 2 captures the working mechanism of the region determination defined in Testing strategy 2. Algorithm 3 uses Algorithm 1 and Algorithm 2 for the computation of the regions in the feature space. Algorithm 4 shows Testing strategy 1 when Algorithm 1 is used and Testing strategy 2 when Algorithm 2 is used. Figure 4.6 shows the flowchart of the testing strategies.

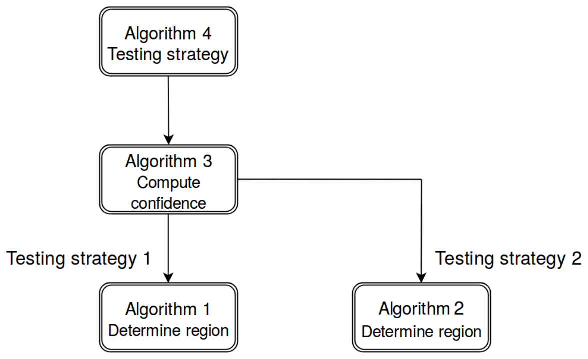

Figure 4.6: Flowchart of the testing strategies.

Algorithm 2 Determine region of the feature space (2): dense, sparse or outlier.

```
Require:
  Test feature vector **x** $\in \mathbb{R}^n$
  Training feature vectors of all seen objects **X** $= \{\mathcal{X}_i, i = 1, \ldots, h\}$
  procedure Region 2
     Compute $\kappa$ (3.19), $\gamma$ (3.20), $\delta$ (3.21) for x
     # Determine the location of x in the feature space
     index = $\delta \cdot \frac{1}{\gamma}$
     if $\gamma > t_1$ and index $> t_2$ then
        return x is in outlier region of X
     else
        if $\gamma > t_1$ and index $\leq t_2$ then
           return x is in sparse region of X
        else
           return x is in dense region of X
        end if
     end if
  end procedure
```

Algorithm 3 Compute confidence c

```
Require:
  Testing feature vectors $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, q\}$
  Training feature vectors of all seen objects $\mathbf{X} = \{\mathcal{X}_i, i = 1, \dots, h\}$
  Labels of all seen objects $L = \{l_i, i = 1, \dots, h\}$
  procedure Confidence
    outlier = 0
    for $\mathbf{x}_i \in \mathcal{X}$ do
      Determine region of the feature space (Algorithm 1 or 2) for $\mathbf{x}_i$
      if $\mathbf{x}$ is in outlier region of $\mathbf{X}$ then outlier++
      end if
    end for
    # Compute percent of the testing class in the outlier region
    $p_{\text{outlier}} = \frac{\text{outlier}}{q}$
    if $p_{\text{outlier}} > t$ then
      $c = p_{\text{outlier}}$
      # Test label prediction $l_{\text{test}}$
      $l_{\text{test}} = l_0$ # new object
    else
      # If only one class is known
      if $|\mathbf{X}| == 1$ then
        $c = 1 - p_{\text{outlier}}$
        $l_{\text{test}} = L$
      else
        for $\mathbf{x}_i \in \mathcal{X}$ do
          Determine the $k$ neighbours of $\mathbf{x}_i$ with their labels according to Euclidean distance
        end for
        Determine number $cn$ of different labels among all neighbours of all testing samples in $\mathcal{X}$
        Find the class $l_M$ which is among $k$ neighbours of a majority of all testing samples in $\mathcal{X}$
        if there is no majority neighbour class then
          $c = 0$
          $l_M = l_x$ #first label among all classes which are neighbours to the maximal number
          of testing samples
        end if
        Determine the percentage $p_{\max}$ of all testing class which have $l_M$ neighbours belonging to
        the class $l_M$
        if $p_{\text{outlier}} > p_{\max}$ then
          $c = \frac{p_{\max}}{cn+1}$
          $l_{\text{test}} = l_0$
        else if $p_{\text{outlier}} > 0$ then
          $c = \frac{p_{\max}}{cn+1}$
          $l_{\text{test}} = l_M$
        else
          $c = \frac{p_{\max}}{cn}$
          $l_{\text{test}} = l_M$
        end if
      end if
    end if
  end procedure
```

Algorithm 4 Testing strategy

```
Require:
  Maximum number of views of an object which can be seen $max_v$
  Minimum number of views of an object which have to be seen $min_v$
  Test feature vectors (stream) $\mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, v,\ v \leq max_v$
  Training feature vectors of all seen objects $\mathbf{X} = \{\mathcal{X}_i, i = 1, \dots, h\}$
  Labels of all objects seen $L = \{l_i, i = 1, \dots, h\}$
  procedure Test
      $q = 1$ # index of test view
      $\mathcal{X} = \{\}$
      while $c < t$ or $q < min_v$ do
          Require (additional) test view $\mathbf{x}_q$, $\mathcal{X} = \mathcal{X} \cup \mathbf{x}_q$
          Compute $c$ (Algorithm 3) for $\mathcal{X}$
          if $q \geq max_v$ then
              Break
          end if
          $q++$
      end while
      Predict test label $l_{\text{test}}$ (Algorithm 3)
  end procedure
```

4.3. Training

Another measure which the second question in Section 4.1 addresses is the answer to: *"How beneficial information am I (the classifier) getting?"*. Firstly, we provide an explanation what the beneficial information is, with respect to generalization of the classification performance. An example is given with the aim to demonstrate the difference between the beneficial and not beneficial information. Imagine you would have to see only three views of a new never seen object based on which you will have to recognize the object in the future. Figure 4.7 and 4.8 capture two extremes of views which could be seen:

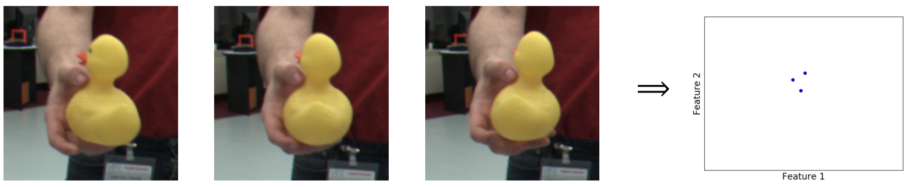

Figure 4.7: Three very similar views of the yellow duck → *non-beneficial* information. If the duck is seen from the back only, it would be unrecognizable from the side or the front.

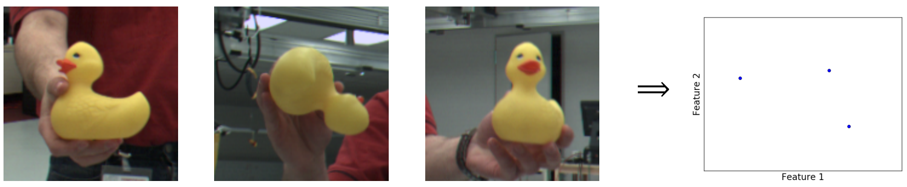

Figure 4.8: Three different views of the yellow duck → *beneficial* information. If these views of the duck are seen, the recognition of the duck in the future should be possible from the side, the front, or the back.

It can be concluded that a preferred scenario is shown in Figure 4.8. If the duck is seen from the back only, like shown in Figure 4.7, it would be unrecognizable from the side or the front. However, the images given in Figure 4.8 provide more information which increase the chance of recognizing the duck in the future.

Looking at the relation of the views presented in Figure 4.7 and 4.8 it is reasonable to assume that the views shown in Figure 4.8 are further away from each other than the views shown in Figure 4.7

in the feature space. We assume that these views cover a larger part of the feature space than views shown in Figure 4.7, and this is the reason why it is possible to recognize the yellow duck easier. The conclusion is that the set of views given in Figure 4.8 is more beneficial for the system than the set of views given in Figure 4.7. Our assumption is that a beneficial training sample is a training sample which expands the feature space covered by the remaining training feature vectors of the object.
Figure 4.9 illustrates how these findings generalize to other objects and more views.

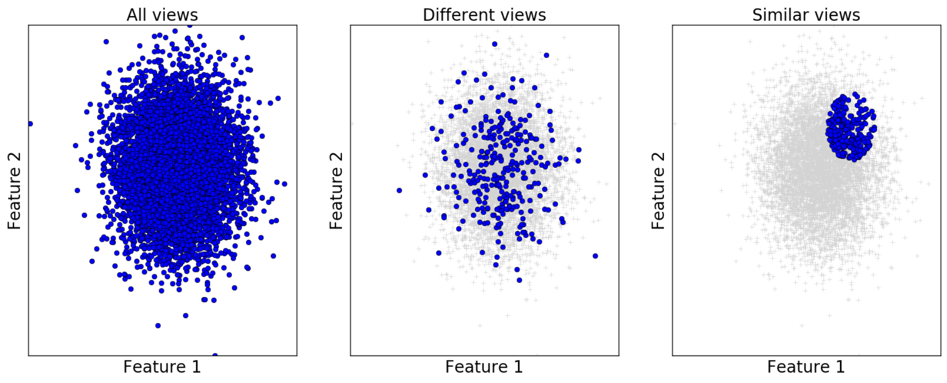

Figure 4.9: The feature space of similar and different views of one object. The left image shows all feature vectors of one object in the feature space. The middle image extracts a set of w different views of the object. The image on the right extracts a set of w similar views of the object.

The image on the left of Figure 4.9 shows all views of one object in the two-dimensional feature space. The middle image represents a set of w different views of the same object. The right image shows a set of w similar views of the same object. It can be noticed that the views shown on the right image are significantly closer to each other in the feature space than the views shown on the middle image. It can be noticed as well that the views in the middle image cover a larger part of the feature space (than the views on the right) even though the number w of views is the same.

Enabling users to know how beneficial information are providing to the system can increase the overall communication and performance of the system. The main idea stated here is that if the system provides online information about the quality of the received information that makes the user aware of the system's states, and affects the user's behaviour. The baseline assumption is the following: If users get a feedback (a score) of how beneficial the training information is, they would try to rotate the object such that the value (the score) is maximized. We propose a graph which shows how beneficial the received information is, running on an online system (like Tubby) while the user is presenting an object, as shown in Figure 4.10.

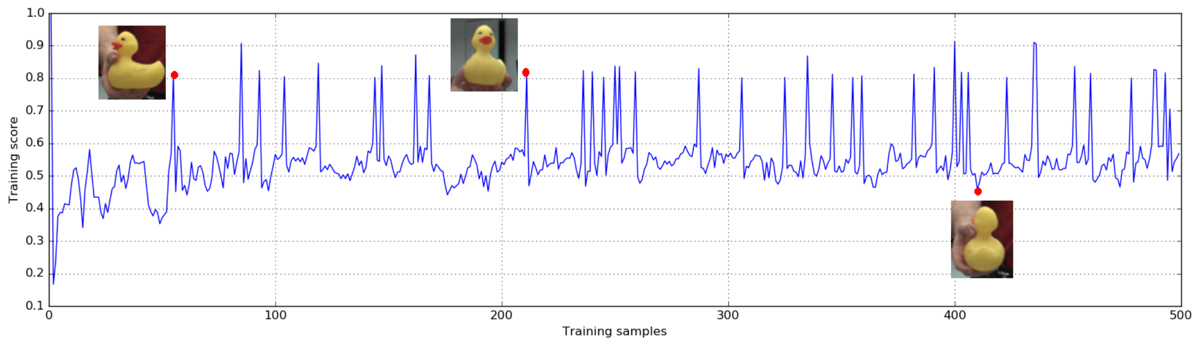

Figure 4.10: Online score measurement during training. The user is rotating an object in front of the system. The system is providing a score of how beneficial the currently shown view is. Red dots capture a single score value. The beneficiality of the view is low when some part of the ducks are shown to the system, which are similar to what is already acquired, but higher when different views are provided.

Figure 4.11 shows feature vectors of acquired training views of an object and a new training view of that object. The figure denotes possible positions of the new training view in the feature space. We develop a measure (a score) which has a low value if the new training view of the object provided to the system is in the dense region of the already acquired training views, like shown on the left image of Figure 4.11. The score has a high value for the case shown on the right image, where the new training view of the object is in the outlier region of the acquired training views. Finally, the score has a medium value for the case shown in the middle image.

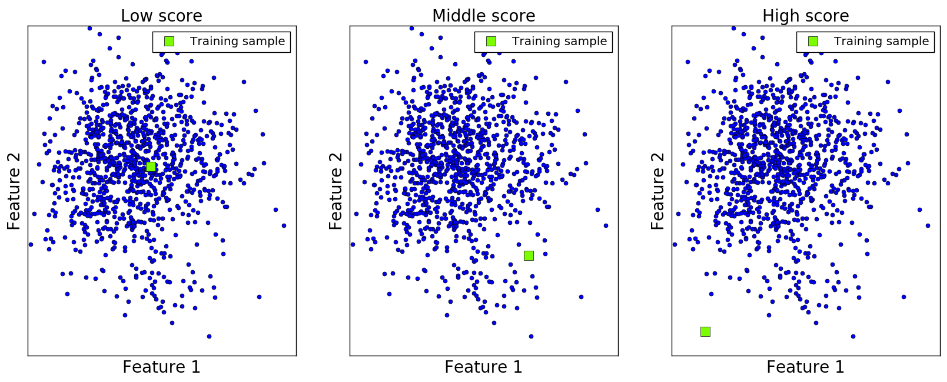

Figure 4.11: Possible positions of a new training view in the feature space of already acquired training views of the same object. The left image shows the training view which is in the dense region of the already acquired training views. The beneficiality score is low in this case. The middle image shows the new training sample in the sparse region. The score has a medium value. The new view of an object in the outlier region of the already acquired training views. The new training view expands the covered feature space of that object. The beneficiality score is high.

The following Subsections 4.3.1 and 4.3.2 provide two measures which capture the beneficiality of a new training view of an object shown to the system. Furthermore, these subsections propose two training strategies which use the measures.

4.3.1. Training Strategy 1

The conclusion from the previous discussion is that, ideally, the training samples cover the whole feature space of an object. However, in an online scenario the system does not know how big the feature space of any object is, or how many training samples will be shown to the system. The best which the system can do is to try to steer the user to show views of an object which cover the whole feature space of that object in the shortest possible time. The system can do this by giving a score to the user. The score denotes the beneficiality of the shown training view. A low score is given when the user shows similar views of an object and a high score is given when the user shows totally different views of that object, with respect to the views which are already shown to the system (of the same object). In this way the user is stimulated to show the system all different views possible. The system gets training views in regions of the feature space of an object which are overlapping with other classes, but also in regions which are not overlapping. It is meaningful to have the training samples in both overlapping and non overlapping regions. Overlapping regions are important because we want to know when the probability of an error during the prediction is high. On the other hand, non overlapping regions are important because these are the regions where the confidence value grows during testing (as explained in the previous section).

Training strategy uses the same underlying mechanism as the testing strategies. Namely, let us consider an online training scenario where a stream of training samples is shown to the system. We consider a single training sample $\mathbf{x}^t$, as well as a set of training samples $\mathcal{X}^t = \{\mathbf{x}_i^t \in \mathbb{R}^n, i = 1, \dots, p\}$ where we assume that they belong to the same class. The set of training samples $\mathcal{X}^t$ represents a set of previously acquired training samples of the training object. For the current training sample $\mathbf{x}^t$ shown to the system it is determined in which of the three regions (dense, sparse or outlier) of the feature space of the acquired training samples lies. The values of γ (3.20) and δ (3.20), for the current sample, are computed and directly used. Equation 4.7 shows the determination of the regions.

$$\text{region} = \begin{cases} \gamma > t_1,\ \delta > t_2, & \text{outlier} \\ \gamma > t_1,\ \delta \leq t_2, & \text{sparse} \\ \text{otherwise}, & \text{dense} \end{cases} \tag{4.7}$$

The idea behind this approach is very simple. Outliers have distant neighbours in approximately the same direction. Therefore they have large values of γ and δ (when compared with samples in the sparse or dense region). Samples in the sparse region have distant neighbours but spread in all directions which makes their γ values large but δ values small. Finally, samples in the dense region have close neighbours in all directions. Therefore values of γ and δ are small. This approach of determining regions of the feature space is expected to be the most precise and to make the least number of errors (when compared to the previously mentioned approaches given in Equation (4.2) and (4.6)). However, the accuracy of this region determination is very dependant on the choice of thresholds t_1 and t_2.

As stated already in the previous section the beneficial training sample expands the covered part of the feature space of the existing training samples, like shown on the right image in Figure 4.11. Looking back to the Section 3.4 which describes outlier detection it can be noticed that every outlier expands the covered feature space of the existing class. Therefore, a high score (meaning that the sample is beneficial) is given to training samples which are belonging to the outlier region of the training class $\mathcal{X}^t$. On the contrary, if the incoming training sample is detected to be in the dense region of the class, like shown on the left image in Figure 4.11, it is not considered to be beneficial and a low score is given. Equation (4.8) and Algorithm 5 show how scores are computed. The threshold t_3 limits the distance to the closest neighbour. When a view of an object is shown, the system checks whether a very similar view of that object was shown before. The threshold t_3 determines which training sample is considered to be very similar and discarded therefore.

$$\text{score} = \begin{cases} \mathbf{x}^t \text{ in outlier region,} & \gamma \cdot \frac{1}{t_1} \cdot c_1 \\ \mathbf{x}^t \text{ in sparse region with close neighbour,} & \gamma \cdot \frac{1}{t_1} \cdot c_2 + c_3 \\ \mathbf{x}^t \text{ in sparse region,} & \gamma \cdot \frac{1}{t_1} \cdot c_4 + c_5 \\ \mathbf{x}^t \text{ in dense region,} & \gamma \cdot \frac{1}{t_1} \cdot c_6 \end{cases} \tag{4.8}$$

where c_1, c_2, c_3, c_4, c_5 and c_6 are constants, and t_1 is a threshold. Algorithm 10 shows the working mechanism of the training strategy. The threshold t_3 in Algorithm 10 determines which training sample will be accepted into the training data. Setting this value high will make the classifier more selective, and it will accept only very different views of an object. Setting this value low will make the classifier less selective and it will accept views of an object which are similar to the views which are acquired already.

Initialization

The main disadvantage of the previously described method is that every time when the feature space is changed in some way (the feature extraction process or the number of dimensions of the feature vectors) the thresholds, t_1, t_2 from Equation (4.7), have to be set manually. Looking long term this is not the best solution. Ideally, the thresholds are set automatically. These thresholds determine whether the incoming training sample is considered to be in the region where neighbors are close, or far away. When they are close the training sample is considered to be in the dense region, whereas when they are far away it is considered to be an outlier. Close neighbors and the dense region of the feature space represent similar views of an object. Opposite of that, the outliers and the outlier region represent completely different views of the object, with respect to the views of the object which are already acquired. Essentially, we need information of the distance in the feature space of similar views of an object, in order to derive the values for the thresholds.
Therefore, we propose an improvement to the previously described method. We firstly assume that the feature space change will not happen often. Every time when this happens the system has to be initialized. The initialization is a presentation of the first object to the system. During initialization the system does not provide a score for the training views. The system is taking all views, and the user should try to show as much views as longer as possible. Based on the acquired views which are mapped into the feature space the thresholds are determined. The idea here is that a distribution of the acquired views of the initial object provides the information what *similar* or *different* view in the current feature space means. We assume that the most distant training samples of the initial object are represented by very different views, while the closest training samples are represented by very similar views. Therefore, the thresholds are derived as their distance in the feature space. The disadvantage of this approach is that its accuracy depends on the object chosen for the initialization. For example,

if a single color ball is used for the initialization, getting an idea of how far away are *different* views of any object in the feature space can be difficult. All views of an object like a single color ball look similar. It is expected that the most distant feature vectors still represent views which are in some way similar, rather than different. Therefore this method can identify thresholds which do not generalize well. Algorithm 7 shows the initialization.

Selectivity

Assuming that we can set the adequate thresholds, maintaining their value constant for a long period of time is not sustainable tactic either. When many different views of an object are acquired the system will provide only bad scores for any shown view because the view will be in some way similar to the views of the object which are acquired already. Setting more selective thresholds at the beginning, and than changing them in a way that selectivity is reduced when some views are acquired solves this problem. The system is more selective at the beginning trying to get as much totally different views of an object as possible, and to cover the largest possible part of the feature space of that object. It becomes less selective with acquired views and over time.
Therefore, we propose a further improvement of the Training strategy 1 which introduces the thresholds which become less selective with a number of acquired training samples. Algorithm 8 shows the procedure.

Algorithm 5 Compute score 1

```
Require:
  New training feature vector $\mathbf{x}^{tr}$
  $\kappa$ (3.19), $\gamma$ (3.20), $\delta$ (3.21) for $\mathbf{x}^{tr}$
  procedure Score 1
    if $\gamma > t_1$ and $\delta > t_2$ then
      return $\mathbf{x}_j^{tr}$ is in outlier region of $\mathcal{X}^t$
      # Beneficial view
      score = $\frac{\gamma}{t_1} \cdot c_1$ # $c_i \in \mathbb{R}, i = 1, \dots, 6$ are constants
    else
      if $\gamma > t_1$ and $\delta \leq t_2$ then
        return $\mathbf{x}_j^{tr}$ is in sparse region of $\mathcal{X}^t$
        # Can be beneficial view
        Determine distance $d$ to the closest neighbour $\mathbf{x}_1^s$ according to Euclidean distance
        if $d < t_3$ then
          score = $d \cdot \frac{1}{t_1} \cdot c_2 + c_3$
        else
          score = $\gamma \cdot \frac{1}{t_1} \cdot c_4 + c_5$
        end if
      else
        return $\mathbf{x}_j^{tr}$ is in dense region of $\mathcal{X}^t$
        # Not beneficial view
        Determine distance $d$ to the closest neighbour $\mathbf{x}_1^s$ according to Euclidean distance
        score = $\gamma \cdot \frac{1}{t_1} \cdot c_6$
      end if
    end if
  end procedure
```

4.3.2. Training Strategy 2

The second training strategy depicts the same concept as the Training strategy 1, with a slight change in the measures which are used for the score derivation. Namely, for the determination in which region of the feature space the training sample $\mathbf{x}^t$ is, instead of δ value, index $= \frac{\delta}{\gamma}$ (defined in Equation (4.5)) is used. This is the same region determination method as explained in Testing strategy 2 (Equation

(4.6)).

$$\text{region} = \begin{cases} \gamma > t_1, \text{ index } (4.5) > t_2, & \text{outlier} \\ \gamma > t_1, \text{ index } (4.5) \leq t_2, & \text{sparse} \\ \text{otherwise}, & \text{dense} \end{cases} \tag{4.9}$$

The thresholds t_1 and t_2 in Equation 4.7 (Training strategy 1) are set based on one randomly selected object from the database. In an online case this object is selected by the user of the system. These values depend to some degree on the chosen object. If the chosen object has a distribution of samples in the feature space which does not represent well the average representation of any object selected values for the thresholds can cause non optimal results of the strategy. Moreover, these values have to be adapted any time the feature space or some of its characteristics is changed (e. g., dimensionality). Therefore, in the Training strategy 1 there are two variables which are dependant on the object chosen for the initialization and which affect the accuracy of the strategy. On the contrary, t_2 in Equation (4.9) is not selected based on the choice of a random object, and its value does not depend on it. It is chosen by the system's designer and should be valid in the feature space of any dimensionality. Accordingly, Equation (4.9) has only one variable which is dependant on the object chosen for the initialization and affects the accuracy of the region determination. When compared to other approaches for the region determination mentioned earlier (Equation (4.2) and Equation (4.7)) this approach makes less errors than the approach mentioned in Equation (4.2) (Testing strategy 1) and it is less dependant on the object chosen for the initialization than the approach given in Equation (4.7) (Training strategy 1).

Algorithm 9 shows the process of updating the thresholds for the second training strategy. Algorithm 10 shows the working mechanism of Training strategy 2, when Algorithm 6 is used for the score computation, and Algorithm 9 for the thresholds update. Figure 4.12 shows the flowchart of the training strategies.

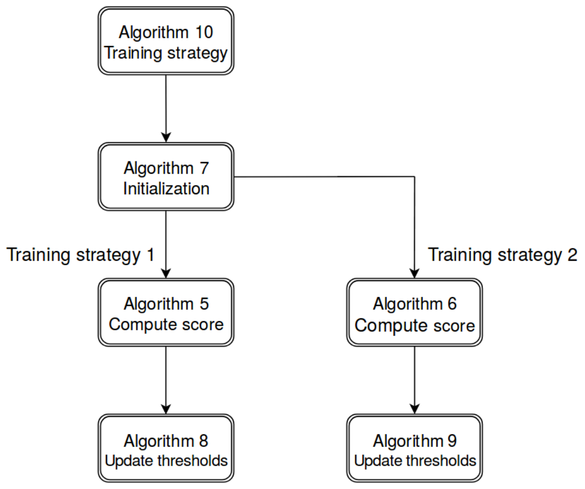

Figure 4.12: Flowchart of the training strategies.

Algorithm 6 Compute score 2

```
Require:
  New training feature vector $\mathbf{x}^{tr}$
  $\kappa$ (3.19), $\gamma$ (3.20), $\delta$ (3.21) for $\mathbf{x}^{tr}$
  procedure Score 2
    index = $\frac{\delta}{\gamma}$ (4.5)
    if $\gamma > t_1$ and index $> t_2$ then
      return $\mathbf{x}_j^{tr}$ is in outlier region of $\mathcal{X}^t$
      # Beneficial view
      score = $\gamma \cdot \frac{1}{t_1} \cdot c_1$          # $c_i \in \mathbb{R}, i = 1, \dots, 6$ are constants
    else
      if $\gamma > t_1$ and index $\leq t_2$ then
        return $\mathbf{x}_j^{tr}$ is in sparse region of $\mathcal{X}^t$
        # Can be beneficial view
        Determine distance $d$ to the closest neighbour $\mathbf{x}_1^s$ according to Euclidean distance
        if $d < t_3$ then
          score = $d \cdot \frac{1}{t_1} \cdot c_2 + c_3$
        else
          score = $\gamma \cdot \frac{1}{t_1} \cdot c_4 + c_5$
        end if
      else
        return $\mathbf{x}_j^{tr}$ is in dense region of $\mathcal{X}^t$
        # Not beneficial view
        score = $\gamma \cdot \frac{1}{t_1} \cdot c_6$
      end if
    end if
  end procedure
```

Algorithm 7 Initialization

```
Require:
  Initial feature vectors $\mathcal{X}^I = \{\mathbf{x}_i^I \in \mathbb{R}^n, i = 1, \dots, p\}$
  procedure Init
    for $\mathbf{x}_i^I \in \mathcal{X}^I$ do
      Determine $\gamma$ (3.20), and $\delta$ (3.21)
    end for
    Create $\gamma$ array: Order $\gamma$ values for all feature vectors $\mathbf{x}_i^I$ in $\mathcal{X}^I$ from the largest to the smallest
    Create $\delta$ array: Order $\delta$ values for all feature vectors $\mathbf{x}_i^I$ in $\mathcal{X}^I$ from the largest to the smallest
    Compute $\gamma_a$ # average $\gamma$ value in the $\gamma$ array
    Compute $\delta_a$ # average $\delta$ value in the $\delta$ array
    $t_1$ = $\gamma$ value which is smaller than $p_1$% of the members of $\gamma$ array
    $t_2$ = $\delta$ value which is smaller than $p_2$% of the members of $\delta$ array
    $t_3$ = $\gamma$ value which is smaller than $p_3$% of the members of $\gamma$ array
    return $t_1$, $t_2$, $t_3$, $\gamma_a$, $\delta_a$
  end procedure
```

Algorithm 8 Update thresholds t_1, t_2 and t_3 (1)

```
Require:
  $\gamma_a$, $\delta_a$ (Algorithm 7)
  Number of already acquired training views of an object $p$
  procedure Thresholds 1
    $t_1 = c_1 + c_2 \cdot p + c_3 \cdot \gamma_a + c_4 \cdot p \cdot \gamma_a$        # $c_i \in \mathbb{R}, i = 1, \dots, 6$ are constants
    $t_2 = c_5 \cdot p \cdot \delta_a$
    $t_3 = c_6 \cdot t_1$
    return $t_1$, $t_2$, $t_3$
  end procedure
```

Algorithm 9 Update thresholds t_1, t_2 and t_3 (2)

```
Require:
  $\gamma_a$ (Algorithm 7)
  Number of already acquired training views of an object $p$
  procedure Thresholds 2
    $t_1 = c_1 + c_2 \cdot p + c_3 \cdot \gamma_a + c_4 \cdot p \cdot \gamma_a$        # $c_i \in \mathbb{R}, i = 1, \dots, 6$ are constants
    $t_2 = c_5 \cdot t_2$
    $t_3 = c_6 \cdot t_1$
    return $t_1$, $t_2$, $t_3$
  end procedure
```

Algorithm 10 Training strategy

```
Require:
  Maximum number of training views per object $max_v$
  Training feature vectors (stream) $\mathbf{x}_i^{tr} \in \mathbb{R}^n, i = 1, \dots, v$, $v \leq max_v$
  Training feature vectors (of the same object) $\mathcal{X}^t = \{\mathbf{x}_i^t \in \mathbb{R}^n, i = 1, \dots, p\}$
  procedure Training
    Initialization (Algorithm 7)
    j=1 # index of training view
    while $j < max_v$ do
      Determine feature vector $\mathbf{x}_j^{tr}$ of the training view $j$
      Compute $\kappa$ (3.19), $\gamma$ (3.20), $\delta$ (3.21) for $\mathbf{x}_j^{tr}$
      Compute score (Algorithm 5 or 6)
      if score > $t$ then
        # Beneficial view, update the memory $\mathcal{X}^t$, include the view $\mathbf{x}_j^{tr}$
        $\mathcal{X}^t = \mathcal{X}^t \cup \mathbf{x}_j^{tr}$
        Update thresholds (Algorithm 8 or Algorithm 9)
        $j++$
      end if
    end while
  end procedure
```

Summary

This chapter presents the fundamental problem of self-evaluation and extracts two crucial questions which tackle the classification. We propose two testing and two training strategies as the answers to these questions. In the next chapter the strategies are evaluated.

5 Results and Discussion

The previous chapters provide the description of the Tubby platform, the theoretical foundation for the understanding of the project, and the proposed methods for improving the platform. This chapter provides the experiment description, results of the baseline performance and evaluations of the proposed methods and a discussion thereon.

5.1. Experiments Description

Details of the execution of the experiments are presented here. The targeted application for the proposed communication strategies is the online interactive scenario, as described in Chapter 2. However, the experiments are conducted in an offline setting, because the communication strategies can be assessed more accurate. The offline setting enables higher control over different parameters and their effect on the performance. Once it is shown that the strategies improve the offline performance, the strategies can be implemented and evaluated on the online platform. Following are details of the experiments: number of repetitions, the used database of images, testing and training data split, performance measure, the used feature space and the used number of objects in the training and testing phase.

Repetition

Every experiment is repeated 100 times where every repetition is called a run. The average results of the 100 runs are reported as the final measure.

Dataset

The data set which is used for the experiments described in this chapter is the HRI-EU database of images of 126 objects. Every object in the database has 1200 images of different views. Images of every object were taken consecutively such that views taken by two consecutive images are always shifted only for a small angle of an arbitrary axes, as depicted in Figure 5.1, which shows 20 consecutive views of the yellow duck. Figure 5.2 shows one image of every object in the database.

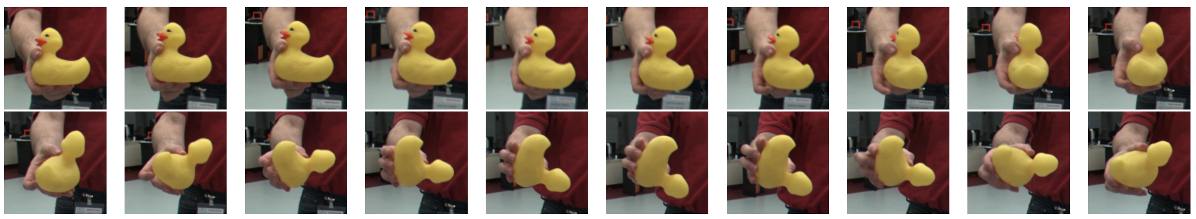

Figure 5.1: 20 consecutive views of the object yellow duck.

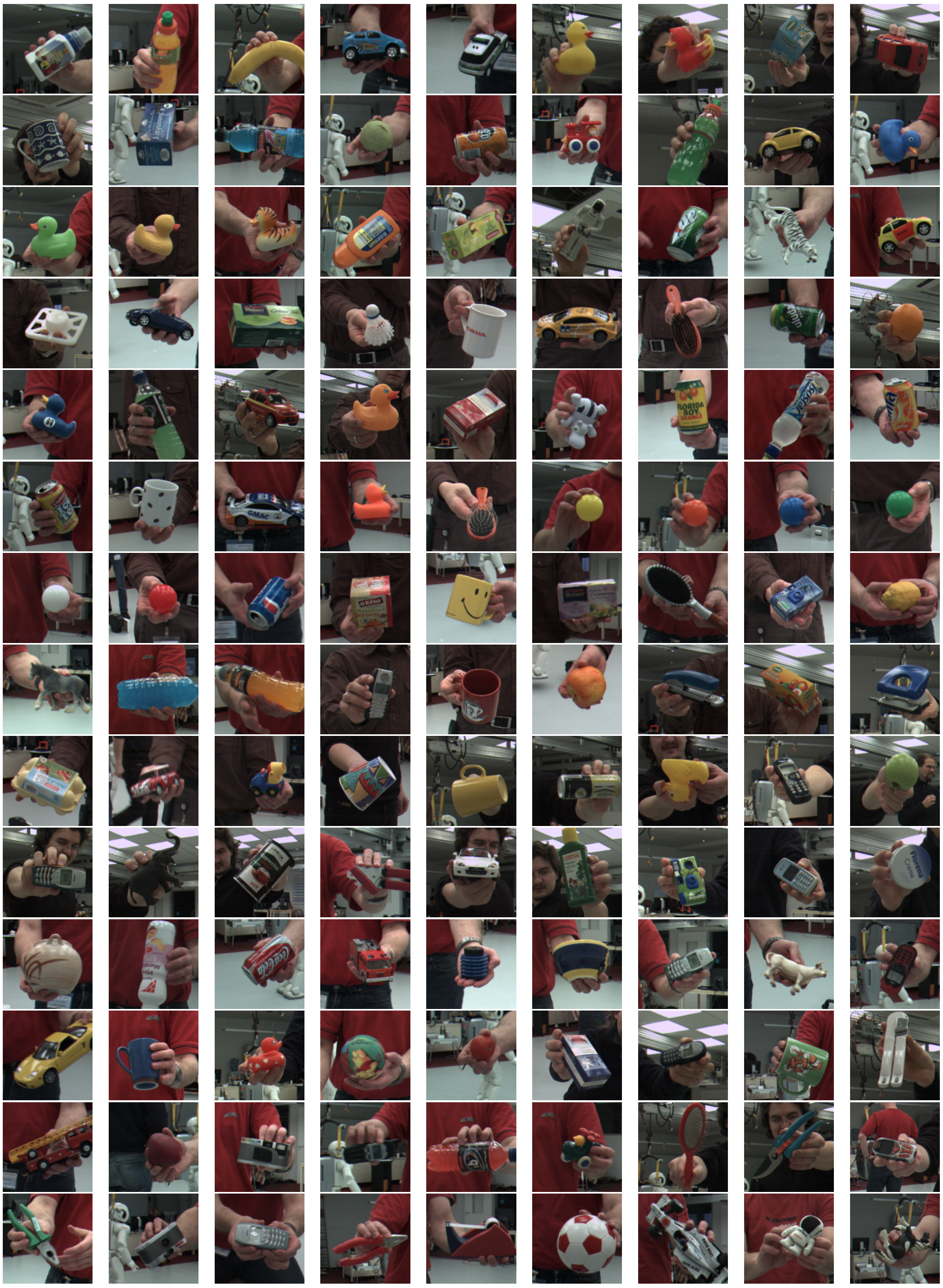
Figure 5.2: One view of every object of the 126 objects in the database used for the experiment.

Training - Testing data

For every execution of the evaluation experiment, data used for the training and testing phase are different. In every run a uniform-random number $g \sim u(1, 1200)$, $g \in \mathbb{N}$ is chosen for every object, which represents an image index. The overall image indexes of one object can be seen as a circle, like shown in Figure 5.3. The first and the last index of images are neighbours on the top. Index g is chosen, and following 200 views of the circle are taken as the testing data, whereas the remaining 1000 views are taken as the training data. The split of data is done for every object separately.

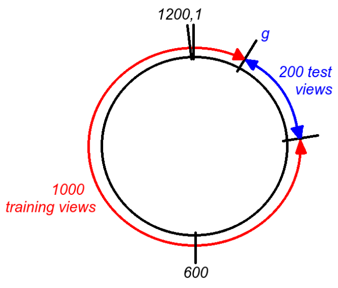

Figure 5.3: Training-testing data split.

Depending on the strategy, the number of testing views per object which are actually used in the experiment varies. All actually used testing views are obtained from the testing data described above. The minimal number of views per object used for the testing phase is set to be 10, whereas the maximal number is set to be 100. The strategies decide on a case to case basis how many views is used for the prediction of an object. The limitation in the number of testing views per object which the system can get is set to mimic the real online behaviour of the user engaged with the system. If the confidence of the prediction is low the system might require new views forever, which is not the optimal behaviour. Therefore, the limitation after which the system has to make the prediction is set.
In the training the system gets a specified number of views of each object which is used for this phase. Every experiment is run for 17 different values for the number of training samples of each object. The performance measure depends significantly on the number of training samples which the system gets. The chosen numbers are: 1, 5, 10, 15, 20, 30, 40, 50, 70, 80, 100, 120, 130, 150, 160, 180, 200. The variance of the performance measure is considerably small when the number of the training samples is larger than 200, which is the reason why higher values are not considered. The specified number of training views is taken from the training data for every object. In the proposed training strategies, the system takes a specified number of training views of each object which obtain a score larger than a threshold t. The baseline training strategy, and how a subset of specified number of views is chosen from the training data are explained in Subsection 5.2.1.
Therefore, the training and testing data actually used in the experiments are different.

Performance measure

The main performance measure used for the comparison in the experiments is the *accuracy* given in Equation 5.1.

$$accuracy = \frac{\sum_{i=1}^{h}\sum_{j=1}^{y_i} TP_{ij}}{\sum_{i=1}^{h} y_i} \tag{5.1}$$

h denotes the total number of classes (objects) used for the testing, TP_{ij} denotes the correctly predicted view of a certain class, y_i denotes the number of testing views of a certain class i.
The baseline performance is described in Section 5.2.1. The baseline strategy does not have the possibility of recognizing new objects as new. The testing strategies proposed in Section 4.2 have this possibility. If the object is predicted as new and never seen in the training and this is correct, it is excluded from the performance calculations. It is not considered as the correct prediction since the system does not know what the object is. But, it is not considered as the wrong prediction since the system knows that the object was not seen before. However, if the object is predicted as new and this is not correct, it is considered as the wrong prediction. If the object is new but predicted as some

of the training objects, this is considered equally wrong. One might argue that this treatment of the predictions related with new objects is unfair. The reward is not given to the performance in the case of the correctly predicted new object, but the penalty is given for the wrong prediction of a new object. However, even though the performance does not get a direct reward when the system recognizes a new object, the indirect reward is obtained by taking the sample out of computations (the number $\sum_{i=1}^{h} y_i$ in Equation (5.1) becomes smaller and therefore the accuracy increases). Furthermore, by computing the performance of the proposed strategies in this way we achieve the lowest possible value. If proven that this value still outperforms the baseline all other ways of computing the performance will only improve the difference between the proposed strategies and the baseline strategy.

Feature extraction

As described in Section 3.2, features of each image in the data set are extracted using the convolutional neural network *AlexNet* [31] which is trained on the *ImageNet* database[1]. Each feature vector $\mathbf{x}$ is the activation of the last neuron layer and it is 1000-dimensional vector ($\mathbf{x} \in \mathbb{R}^{1000}$).

Dimensionality of the feature space

The experiments are performed in four different dimensionalities of the feature space. All experiments are done in 2D ($\mathbf{x} \in \mathbb{R}^{2}$), 10D ($\mathbf{x} \in \mathbb{R}^{10}$), 100D ($\mathbf{x} \in \mathbb{R}^{100}$) and 1000D ($\mathbf{x} \in \mathbb{R}^{1000}$). For every run of 2D experiments 2 neuron outputs are taken randomly from the extracted feature vectors. The performance changes its value significantly depending on the chosen neurons. Moreover, it is not possible to extract a special set of neurons which represents good features for every object. Hence, for the sake of obtaining generality in the results and to reduce a dependency on the choice of neurons, a random selection is applied. As for 2D, 10 neuron outputs and 100 neuron outputs are selected randomly from the feature vectors in every run. In the case of 1000D the whole feature vector is used.

Number of training and testing objects

The performance is significantly dependant on the number of training and testing objects and whether all testing objects are used for training. Four representative cases are chosen for the experiments:

Setting 1 All 126 objects are in the testing, and all 126 objects are in the training.

Setting 2 All 126 objects are in the testing, and random 100 objects are in the training. In every run (out of 100) different sets of random 100 objects is selected, so that the effect of the objects chosen on the results is reduced to minimum.

Setting 3 Random 15 objects from the database are in the testing, and 10 out of those 15 objects are used for the training. In every run (out of 100) different sets of random 15 objects for the testing and 10 object out of those 15 objects for the training are selected, so that the effect of the objects chosen on the results is reduced to minimum.

Setting 4 Random 10 objects from the database are in the testing, and same 10 objects are used for the training. In every run (out of 100) different sets of random 10 objects is selected, so that the effect of the objects chosen on the results is reduced to minimum.

5.2. Results

The results of the experiments described above are presented in this section. The baseline performance is described in the following subsection.

5.2.1. Baseline Performance

The baseline performance is derived from the offline simulation of the online Tubby performance (Chapter 2). The modified system (change from category learning to object recognition and the classification process) is our baseline. In this changed setting Tubby takes views of a presented object and predicts what that the object is, based on a kNN classifier. In the offline setting this behaviour is simulated in a way that the system takes a certain number of consecutive views of the object. Consecutive views, as shown in Figure 5.1, are taken to correspond to the user's rotation of the object in front of Tubby.

[1] http://www.image-net.org/

In the best case, the user would rotate the object, and therefore increase the probability of a correct prediction. Whereas the worst case scenario is if the user holds the object static and thereby reduces the chances of a correct prediction.
In the baseline training the system gets a specified number of consecutive views of each object which is used for this phase. In the baseline testing phase a specified number of consecutive views of each testing object are taken. The prediction for every testing object is based on a majority vote for every view (image) of that testing object. Every view vote is the prediction of the kNN algorithm for that view.

In the previous section it is noted that the dimensionality of the operating feature space is chosen to be: 2D, 10D, 100D and 1000D. Figure 5.4 shows the comparison between the baseline performance of the system when all objects are used both for the training and testing phase. It is visible that the difference between performance in 100D and 1000D is not proportional to the difference between 2D and 100D performance. The performance value is not linearly dependant on the number of dimensions chosen.

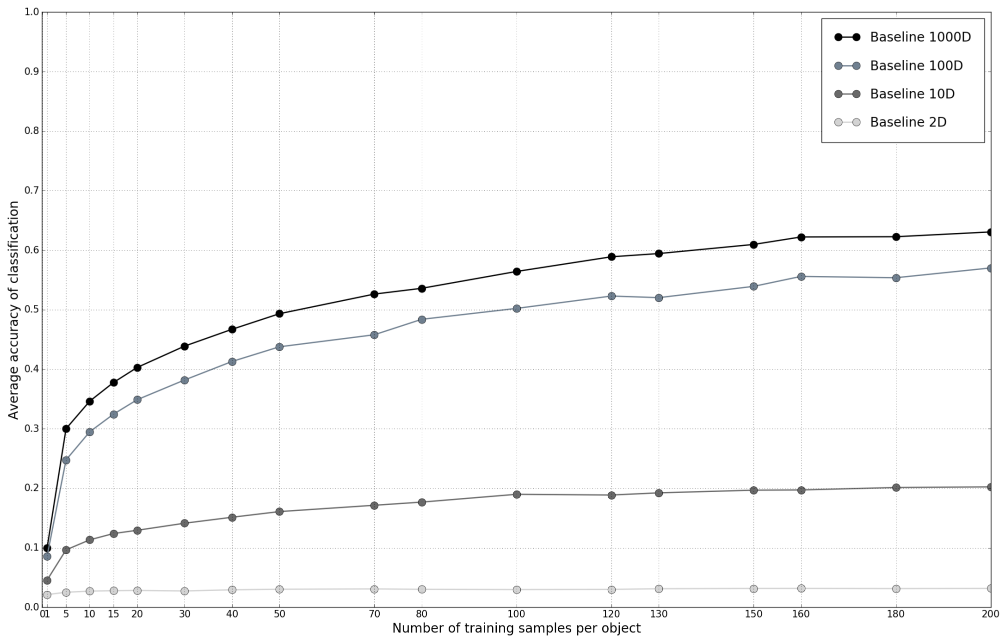

Figure 5.4: Baseline performance averaged across 100 runs for 2D, 10D, 100D, and 1000D. All 126 objects were used for the training and testing.

The following subsection presents results obtained in the described experiments with the baseline and the improved system. Chapter 4 proposes two training and two testing strategies. Every experiment is consisted of two phases, the training and testing phase. In order to assess the proposed methods the experimental settings are denoted in Table 5.1. Firstly the baseline performance is measured with the baseline training and baseline testing strategy. Afterwards experiments with the baseline training strategy and the proposed testing strategies are conducted, which are followed by the experiments with the proposed training strategies and the baseline testing strategies. Finally, a combination of the proposed training and testing strategies is executed.

Experimental settings	Training	Testing
Baseline	Baseline training	Baseline testing
Testing strategy 1	Baseline training	Testing strategy 1
Testing strategy 2	Baseline training	Testing strategy 2
Training strategy 1	Training strategy 1	Baseline testing
Training strategy 2	Training strategy 2	Baseline testing
Test 1 Training 1	Training strategy 1	Testing strategy 1
Test 1 Training 2	Training strategy 1	Testing strategy 2
Test 2 Training 1	Training strategy 2	Testing strategy 1
Test 2 Training 2	Training strategy 2	Testing strategy 2

Table 5.1: Experimental settings. The training column denotes which strategy is used for the training phase during the measure of the performance whereas the testing column denotes which testing strategy is used for the testing phase.

5.2.2. Resulting Performance

This subsection presents the most informative visualizations. All evaluations are included in the Appendix, along with tables containing the exact performance values. In the following figures it is shown how the performance varies depending on the used training and testing strategies. The description of the experiments is given in the previous section.

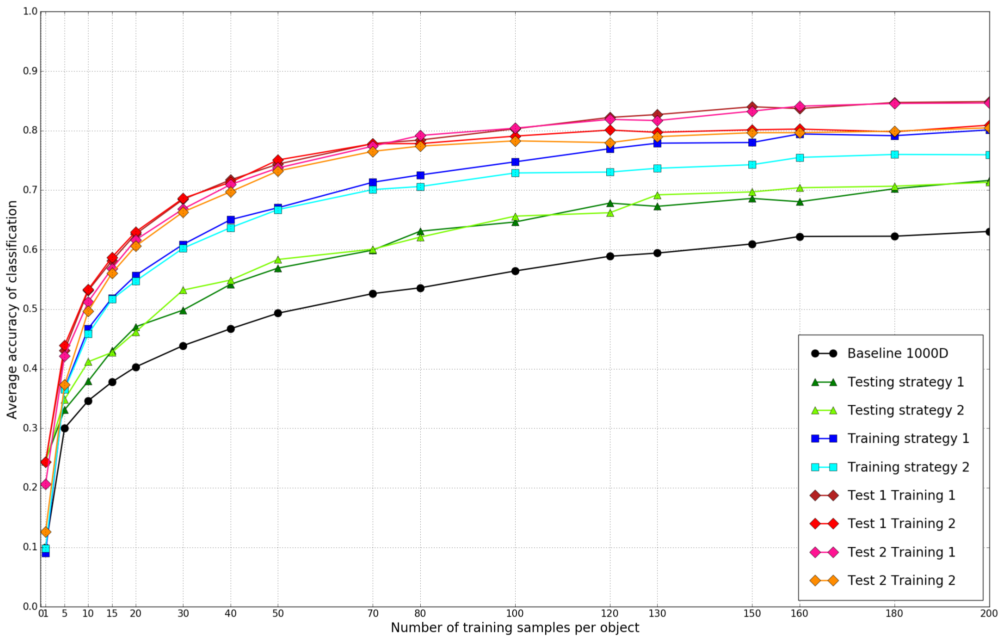

Figure 5.5: Setting 1 - 1000D. Performance of the proposed strategies compared with the baseline performance in 1000D feature space. All 126 objects are used in the testing, and all objects are used in the training.

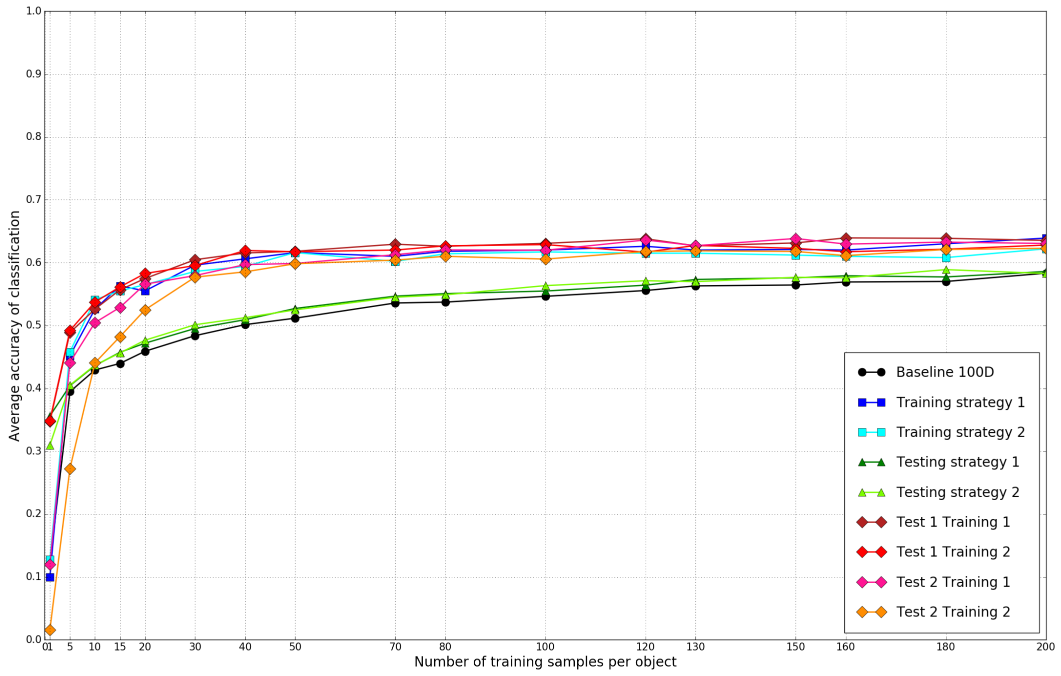

Figure 5.6: Setting 3 - 100D. Performance of the proposed strategies compared with the baseline performance in 100D feature space. Random 15 objects are used in the testing, 10 out of those 15 objects are used in the training.

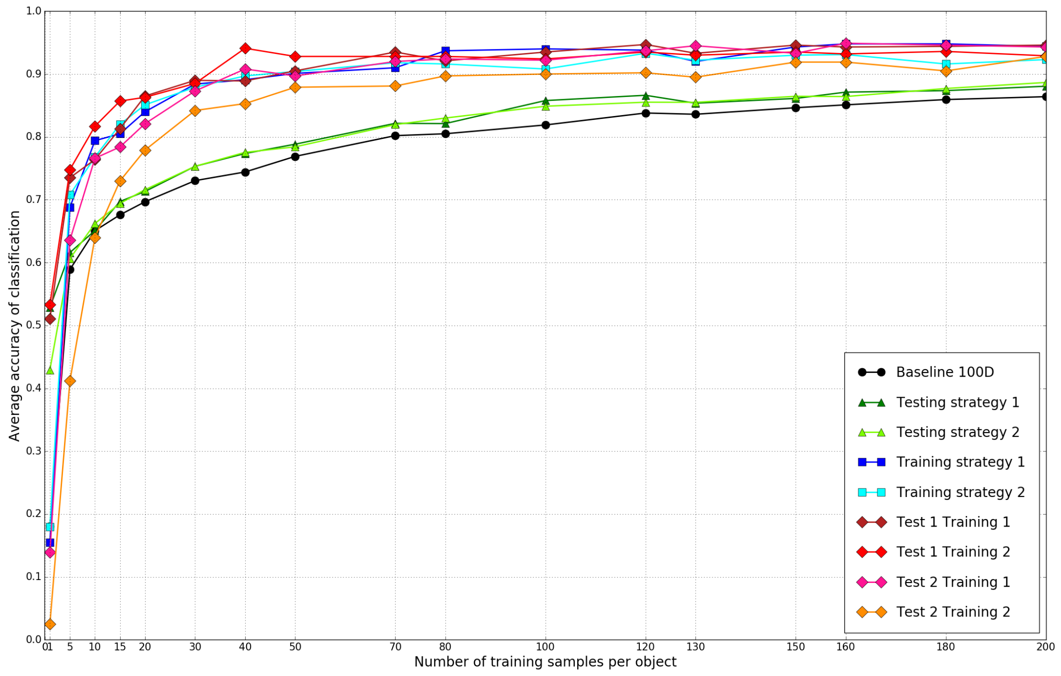

Figure 5.7: Setting 4 - 100D. Performance of the proposed strategies compared with the baseline performance in 100D feature space. Random 10 objects are used in the testing, and the same 10 objects are used in the training.

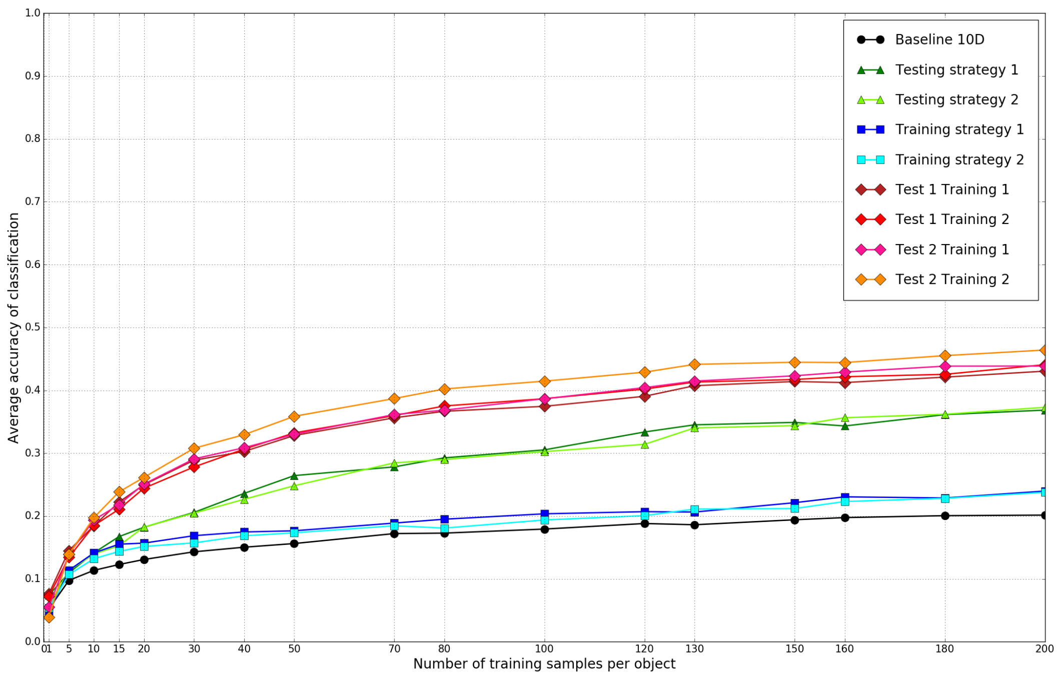

Figure 5.8: Setting 2 - 10D. Performance of the proposed strategies compared with the baseline performance in 10D feature space. All 126 objects are used in the testing, and random 100 out of 126 objects are used in the training.

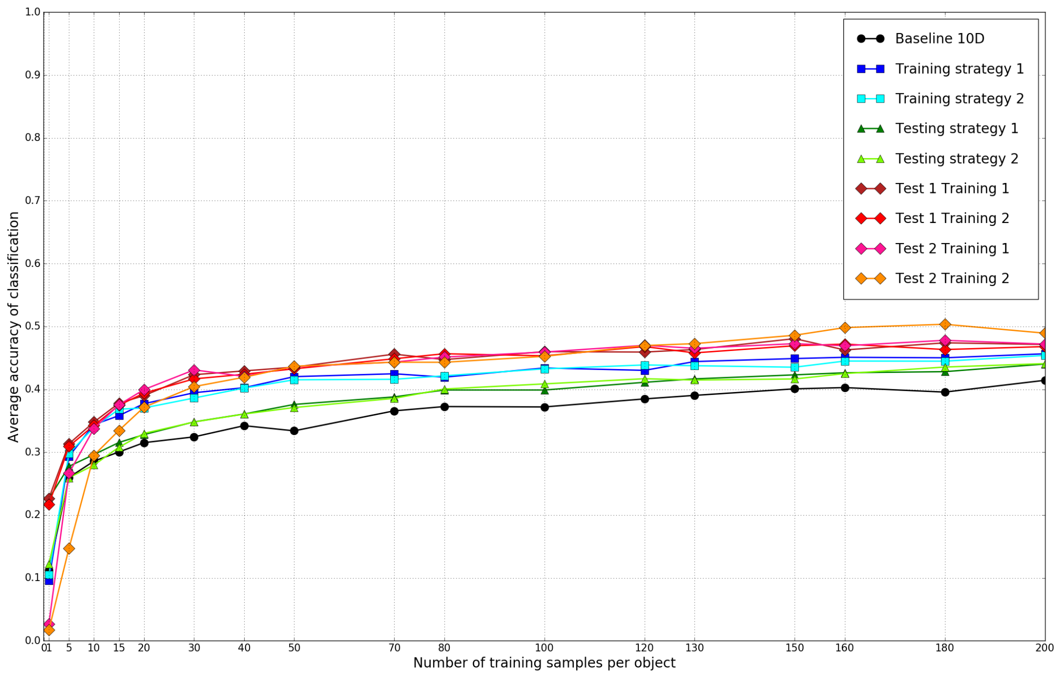

Figure 5.9: Setting 3 - 10D. Performance of the proposed strategies compared with the baseline performance in 10D feature space. Random 15 objects are used in the testing, 10 out of those 15 objects are used in the training.

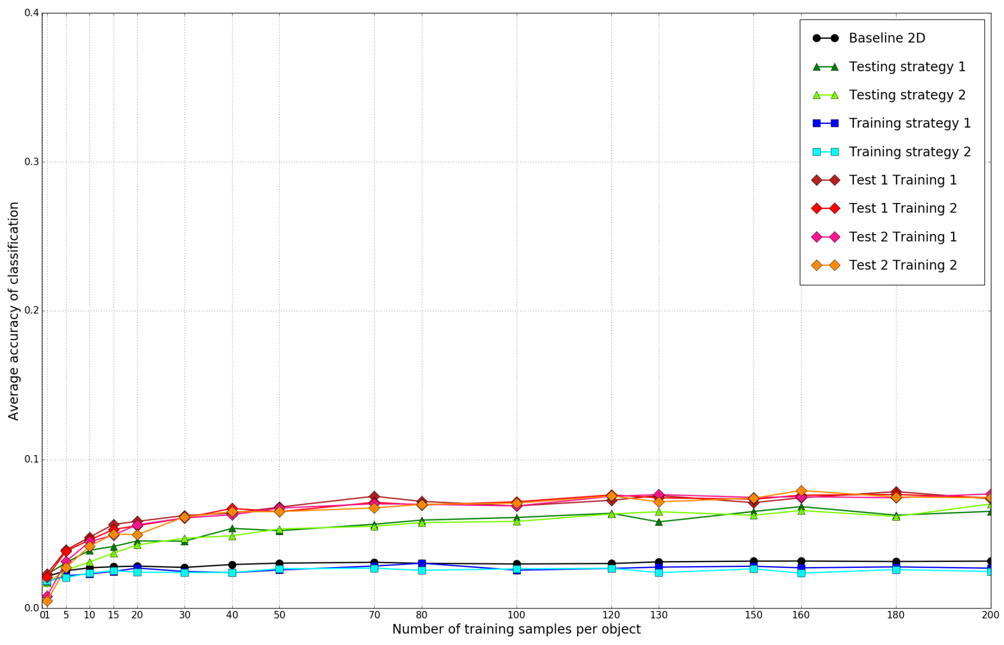

Figure 5.10: Setting 1 -2D. Performance of the proposed strategies compared with the baseline performance in 2D feature space. All 126 objects are used in the training, and all objects are used in the testing.

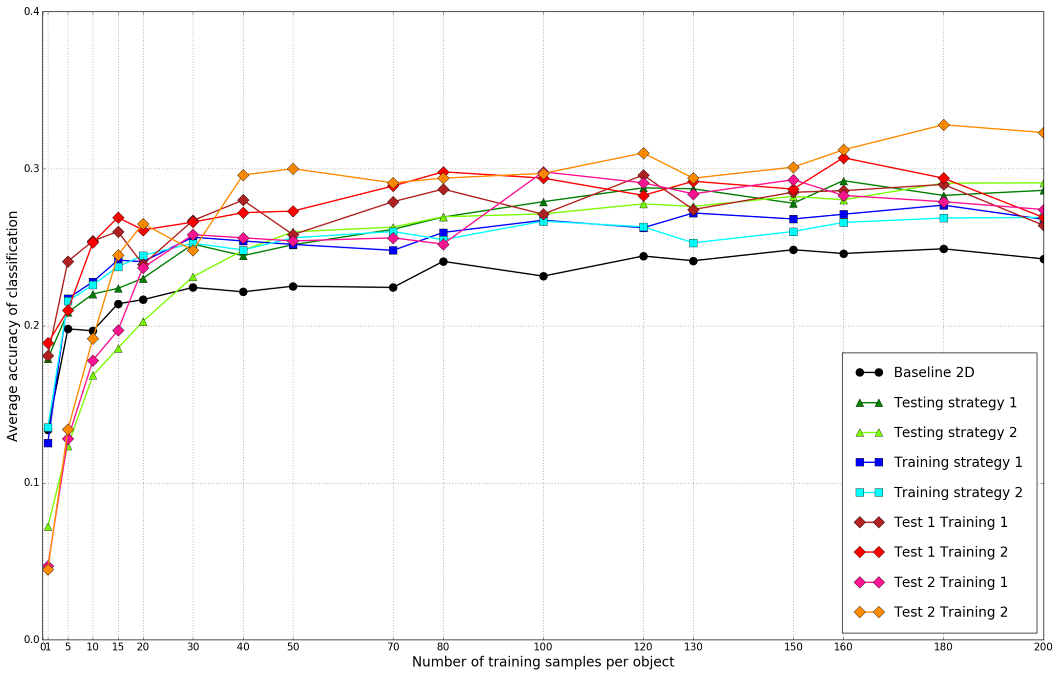

Figure 5.11: 2D - Setting 4. Performance of the proposed strategies compared with the baseline performance in 2D feature space. Random 10 objects are used in the testing phase, and the same 10 objects are used in the training phase.

5.3. Discussion

The discussion on the experiments from the previous section is divided into two parts. Firstly, Subsection 5.3.1 extracts important findings on the obtained results. Secondly, Subsection 5.3.2 presents an offline simulation of the interactive system which learns objects (like the Tubby system). This subsection provides additional insights into the working mechanisms of the proposed strategies and the overall system.

5.3.1. Discussion on the Evaluations

Table 5.1 lists the strategies which are evaluated. A discussion on the results for the testing, training and a combination of strategies is given in this section.

Testing strategies

The proposed testing strategies have an important characteristic. In the case of a low confidence of the prediction the system requires more views of a testing object to be shown. One might argue that this strategy has better performance, than the baseline, just because more views are provided to the system. Note, however, that the proposed testing strategies do not simply use more views of any object. The views are required when that is necessary. Moreover, the target of the thesis is the improvement of the communication and information exchange between the user and the system. Therefore, the proposed strategies are evaluated from the performance point of view, but their benefit is in the improved communication as well.

The observations related with the testing strategies are the following:

- Testing strategy 1 and Testing strategy 2 have approximately equal performance in all cases, and none of them is significantly better. Since Testing strategy 1 uses only one measure for the region determination (index (4.1)) this strategy is preferred for applications over Testing strategy 2 (which uses two measures for the region determination).
- The performance of Testing strategy 2 is lower than the performance of Testing strategy 1 when one training sample per object is shown to the system (as clearly visible in Figure 5.7). However, the proposed strategies are not supposed to work properly when the number of training samples per object is one. The confidence of the prediction is 0, in this case. Therefore, we do not consider this finding significant for the overall evaluation of Testing strategy 2. Nevertheless, the performance of Testing strategy 2 is lower than the performance of Testing strategy 1 and the baseline performance in two-dimensional feature space for a small number of training samples per object (< 25) and a small number of objects in the learning process (≤ 15, Setting 3 and 4). Figure 5.11 clearly shows this finding. Accordingly, we conclude that, overall, Testing strategy 1 performs slightly better.
- The largest absolute difference between the testing strategies' performance and the baseline is observed when the feature space is 10-dimensional and a lot of objects are in the learning process (like shown in Figure 5.8). This finding can be due to the fact that a lot of information is lost when 10 dimensions are extracted out of 1000 dimensional feature vector. Accordingly, it is reasonable to assume that the overlap of classes in the feature space is larger in 10D than in 1000D. Additionally, the large number of objects makes the overlap even more severe. Therefore, the baseline system makes a lot of mistakes. This means that the probability of making an error is high in many cases. Confusing situations trigger the need for more information. In these cases, the more shown views to the system make the largest difference in the performance. Regarding the two-dimensional feature space and a lot of objects in the learning process, the absolute difference of the performance of the testing strategies and the baseline is large as well, but smaller than in the 10-dimensional feature space. However, the relative improvement achieved by the implementation of the strategies has the largest value in 2D when a lot of objects are in the learning process.
- A large number of objects in the learning process in general yields a larger difference between the baseline and the testing strategies' performance. When variations in the same feature space are considered the conclusion is that more objects in the learning process causes the larger

improvement of the baseline. However, when the feature space is two-dimensional the larger number of objects causes just a moderate improvement in the difference between the testing strategies and the baseline, while at higher dimensions the improvement is larger. Figures 5.8 and 5.9 are both 10-dimensional evaluations, but the number of objects is different, and the difference in the improvement clearly shows that the larger number of objects brings larger improvement. When the system knows only a small number of objects the baseline strategy does not make so many errors. The number of errors increases when the number of objects is increased. Figure 5.16 illustrates what is happening in the feature space when more objects are shown to the system. More uncertain and confusing situations are present, as more overlapping regions of different classes are present. The testing strategies are designed to boost the performance in these situations.

- The testing strategies improve the baseline performance more when the testing and the training set of objects are the same, then when they are not. The baseline system cannot recognize new objects as new, so every prediction for any object which is not in the training set is by default wrong. The proposed strategies can predict a new object as new. However, the evaluation results do not show that the improvement of the baseline is better when the testing and training sets are different. Nevertheless, note that we exclude correct predictions of new objects from the performance calculations. On the other hand, we include wrong predictions of known objects as new objects which might be an explanation why the improvement is better in the case when the testing and training objects are the same.

Training strategies

- Training strategy 1 and Training strategy 2 have similar performance in most of the cases. It seems, however, that overall Training strategy 1 performs slightly better.

- The improvement of the baseline caused by the proposed training strategies is larger at higher dimensions ($\geq$ 100D). The difference in the improvement can be seen when Figures 5.10 and 5.5 are compared. In general, the higher the dimensionality of the operating feature space, the higher the difference between the performance of the training strategies and the baseline.

- The number of objects which is used in the training and testing has a significant impact. It turns out that in the highly dimensional feature space ($\geq$ 100D) the more objects in the learning process, the better the improvement of the baseline. Our explanation of this phenomena is that the overlap among different classes (objects) in the feature space of the system is much larger when there are approximately 100 objects in the learning process, than when there are only 10 objects. The larger overlap among object leads to a higher need for the high quality representation of objects. In the case of a small number of objects the baseline system performs well already. In the case of a larger number of objects, the effect of the fine extraction of the training data makes a difference in the situations which are confusing for the system. However, this finding does not hold for the low dimensional feature space ($\leq$ 10D). On contrary, in the lower dimensions the improvement is lower when the number of objects is large. The most extreme case is shown in Figure 5.10 where the performance of the training strategies is below the baseline performance.

- The training strategies do not show a consistent dependency on whether the testing object set contain objects which are not in the training object set. However, at higher dimensions ($\geq$ 100D) the training strategies in general improve the baseline more when the training and testing object set are the same. This might be justified with the fact that relative improvement is approximately the same. But, the baseline performance is worse for the case where the training and testing object set are not the same. Therefore, the absolute improvement is less in this case than in the case where the training and testing object set are the same. On the other hand, in two-dimensional feature space it seems that the training strategies bring the larger improvement when the testing and the training set are not the same. In the case of 10-dimensional feature space it turns out that the equality of the testing and training set does not have an impact on the performance improvement.

Combinations of the testing and training strategies

- All combinations of the testing and training strategies have similar performance in most of the cases. In general all performances of the combinations are better or equal to the training or the testing strategies' performance. The ordering of the performances of the combinations is not static for all cases. None of the combinations performs superior in all cases.

- The combinations with Testing strategy 2 have the performance which is lower than the baseline when a small number of objects is in the learning process (≤ 15, Setting 3 and 4) and for a small number of training views per object (≤ 15). This finding was established for Testing strategy 2 performance as well, for a slightly higher number of views per object (≤ 30). We conclude that due to the Testing strategy 2 performance the combinations with this testing strategy behave the same way. Figures 5.7 and 5.9 show this.

- The combination of Testing strategy 2 and Training strategy 2 in some cases outperforms all other strategies, like shown in Figure 5.8. Interesting is that there are cases when for a small number of training views per object (< 15), this combination has the performance which is under the baseline, but for the larger number of training views (> 20) it outperforms all other combinations and strategies (Figure 5.11). However sometimes this combination significantly underperforms all the other combinations, including the Training strategy 2 (Figure 5.7). Variations in the performance of this combination are large.

- The combinations of Testing strategy 1 with both of the training strategies provide the best results on average, among all the combination performances. These combination performances differ very little. However, the combination of Testing strategy 1 with Training strategy 1 seems to provide the best results overall.

- In higher dimensions of the operating feature space (≥ 100D) when a lot of objects is the learning process and in smaller dimensions of the operating feature space (≤ 10D) for any number of objects in the learning process, the combinations' improvement of the baseline is approximately equal to the sum of the corresponding testing and training strategies' improvements, as Figure 5.9 shows. For example, if the testing strategy improves the baseline for 3%, and the training strategy for 10%, the combination of these strategies improves the baseline for 13%. In some cases the combinations' improvement is even larger than the sum of the corresponding improvements of the testing and training strategy separately, which can be seen in Figure 5.8. Figure 5.10 shows the extreme case. The combinations' performance is better than the testing strategies' performance even if the performance of the training strategies is slightly under the baseline performance. However, this is not always the case. In general, the improvement achieved by any combination is better or equal to the improvements achieved by the corresponding testing or training strategy separately. In high dimensions of the operating feature space (≥ 100D) when a small number of training objects (≤ 15, Setting 3 and 4) is in the learning process the combinations' improvement is not equal to the sum of corresponding improvements. Figure 5.6 shows that, in this case, the combinations do not perform better than the training strategies separately. It is worth of noting that the testing strategies' performance is not significantly better than the baseline in this case.

Table 5.2 summarizes the effects of the important factors on the performance of the proposed strategies. The dimensionality column denotes how the performance varies if the dimensionality of the feature space is increased (e. g., from 2D to 100D). The number of objects column denotes how performance varies if the number of learning objects is increased (e. g., from 10 to 100). This column is split into 3 cases, corresponding to two-dimensional, 10-dimensional and a highly dimensional (≥ 100D) feature space, respectively. The equal sets column denotes how performance varies when compared the case where the training and testing object set are the not the same, and when they are the same.

Strategies	Dimensionality ↑	Number of objects ↑			Equal sets $no \rightarrow yes$		
		2D	10D	≥100D	2D	10D	≥100D
Testing strategies	↓	↑∼	↑	↑	↑∼	↑	↑
Training strategies	↑	↓	↓	↑	↓	∼	↑
Combinations	∼	∼	↑	↑∼	∼	↑	↑

Table 5.2: Summary of the effects of the important factors on the performance value. The dimensionality column denotes how performance varies if the dimensionality of the feature space is increased. The number of objects column denotes how performance varies if the number of learning objects is increased. The equal sets column denotes how performance varies when compared the case where the training and testing set are the not the same, and when they are the same. Arrow turned up ↑ means that the difference between the performance of the corresponding strategy with respect to the baseline performance increases when the variable in the column is changed in the noted direction. Likewise, the arrow turned down ↓ symbolizes decrease in the difference of the corresponding strategy and the baseline performance. The sign ∼ means that a dependency of the corresponding strategy on the change of the variable in the column is not visible, or the evaluations in this work are not enough for a clear conclusion about the dependency.

Finally, we conclude that overall, the combinations of the proposed strategies perform the best. The combinations outperform or are equal to the performance of the testing and training strategies. Testing strategy 1 in combination with Training strategy 1 provides an optimal solution for all dimensionalities of the feature space and any number of objects in the learning process.

5.3.2. Interactive System Simulation

The previous sections describe the conducted offline experiments and the evaluation of the proposed methods. However, these experiments do not capture completely the interaction which exists between the user and the Tubby platform in an online scenario. Namely, in the online scenario the testing and training phase are switching alternately. The presented object to the system is firstly in the testing phase, and the system predicts a label. If necessary or wanted by the user the system can switch to a training phase. The switch is necessary if the object was never seen by the system before, or the prediction is wrong. After the training of the presented object, another object is presented and the order of the testing and training phase is repeated.

This section presents a simulation which captures the described online behaviour more closely. A block diagram of the simulation of the *system-user* interaction during the presentation of one object is shown in Figure 5.12. The user firstly presents an object for the recognition to the system, which is block 1 of the diagram. The system uses the proposed testing strategy and assesses the confidence of the prediction. If the confidence is high enough the system predicts a label, which is depicted as block 2 of the diagram. If the prediction is correct, there is no necessity for further information (more views). The red circle represents the end of the interaction related with that object. This means that the presented object is already known (the system learned it before), and that the training data (views of the object) given to the system are beneficial enough that recognition is done fast and correct. Another object can be shown to the system, and the interaction starts from block 1 again. The prediction at block 2 can be wrong as well, or a new object (not known) can be predicted. This means that either the system does not have any information about the object (not known object) or the information (shown views) are not good enough for the correct prediction. The training phase is necessary in both situations, and this is depicted by block 3 of the diagram. In the case that the confidence of the prediction is low at block 1, the system requires more information (more views to be shown) which is represented by the block 4 of the block diagram. When the confidence is high enough or the limitation in the number of views is reached the system makes a prediction (block 5 of the block diagram). Again, the prediction at block 5 can be correct or wrong. In case of a correct prediction it can be concluded that the system has some small internal confusion which is the reason why more views were required, but the confusion was resolved by the system. Note that a case of a very low confidence at block 4 and a correct prediction at block 5 is possible. According to the low confidence value the system is still uncertain. The reason for the correct prediction can be that a specified set of views of the testing object is shown to the system. In some other case it might happen that the prediction is wrong if the other set of views of the same object is shown to the system. If the confidence is high enough, the training phase is optional, captured by block 6, depending on the user. If the confidence is low, the training phase is mandatory. On the other hand, if the prediction is wrong at block 5 this means that there might be a significant overlap of the feature spaces of the testing object and one or more known objects. If a new object is predicted here it means that there is some internal confusion among known objects and the samples of the testing object. Therefore, it is beneficial to provide more training data to the system. The training phase is denoted with block 7 in the diagram. Training phases depicted with blocks 3, 6 and 7 are terminated when the user stops presenting the object, which is simulated with a specified number of the beneficial training views which the system takes in the simulation.

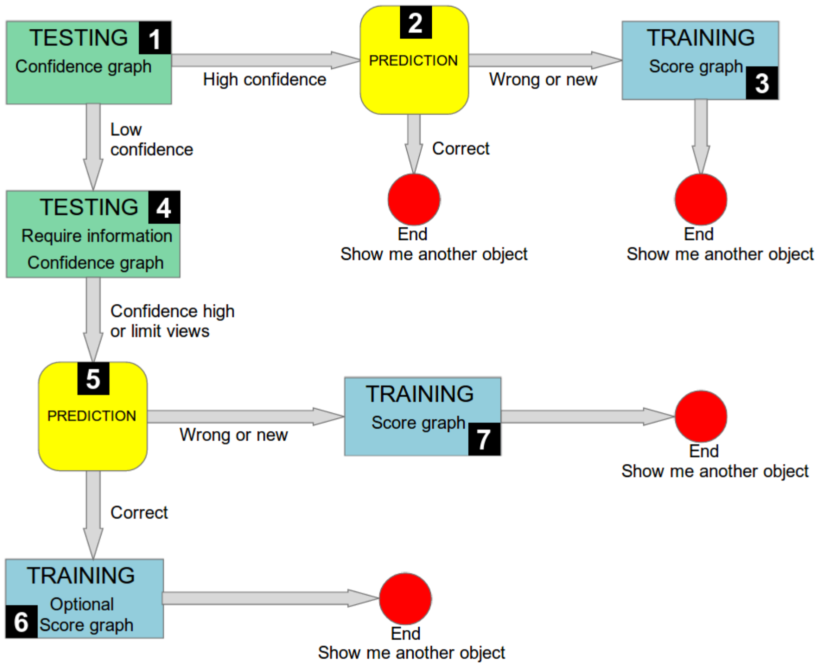

Figure 5.12: Functional block diagram of the simulation of the interaction between the user and the system. An object is presented to the system. It is firstly in the testing phase, and the confidence of the prediction is measured in block 1. If the confidence is high enough the prediction is made by block 2. The system can predict a correct known object, a wrong object or a new object. In the case of a wrong or a new object prediction, a training is necessary, performed by block 3. If block 1 estimates a low confidence, additional views are required by block 4. The prediction is made in block 5 when the confidence is high enough or the limitation in views is reached. Again in the case of a wrong or a new object prediction the training is necessary, performed by block 7. If the prediction is correct, the training done in block 6 is optional, depending on the user.

Testing strategy 1 and Training strategy 1 are implemented for the demonstration in 1000-dimensional operating feature space ($\mathbf{x} \in \mathbb{R}^{1000}$). Objects for the interaction are selected randomly from a pool of 20 different objects chosen from the database. The pool of 20 objects is chosen randomly from the database. We do not use all 126 objects from the database because it is more difficult to visualize and to make conclusions for a larger set of objects.

Online graphs

Figure 5.13 shows the online graph which the system displays to the user. The alternation between the testing (blue) and training phases (red) is captured by different colors. In the testing phase the confidence of predictions is shown to the user, while in the training phase the beneficiality of each training view is shown.

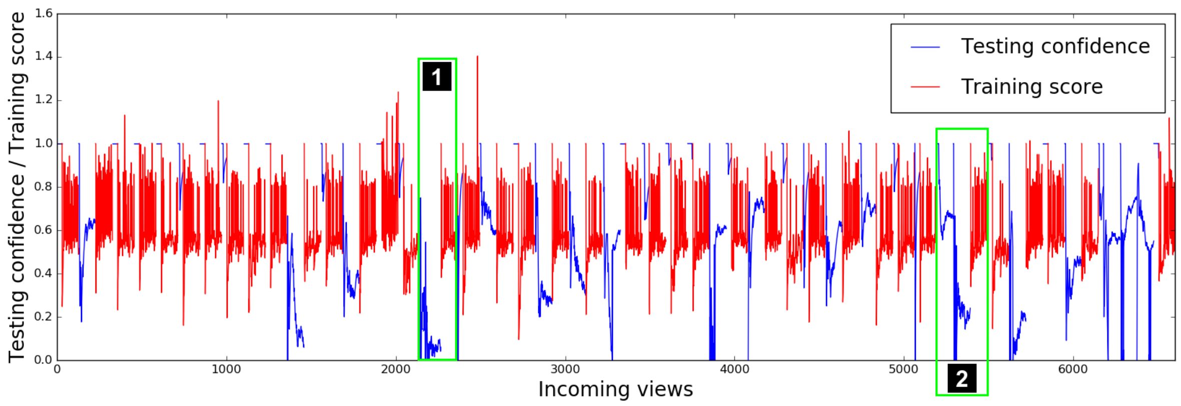

Figure 5.13: The graph shown to the user in the interactive learning scenario. Objects are presented to the system, which firstly performs testing, and provides the testing confidence to the user (blue) and then depending on the situation the object triggers a training phase (red). In the training phase the beneficiality graph is shown to the user. The green box 1 highlights an interaction during learning of one object. The green box 2 extracts a situation where two objects are in the interaction with the system.

Box 1 highlights the interaction of the system and the user during the learning process of one object. Figure 5.14 an example of the same information which box 1 provides. As explained already, the system is firstly in the testing phase, which is followed by a training phase. The blue square in Figure 5.14 represents the beginning of the testing phase and the blue circle represents the end of this phase. Likewise, the red square and the red circle represent the beginning and the end of the training phase, respectively.

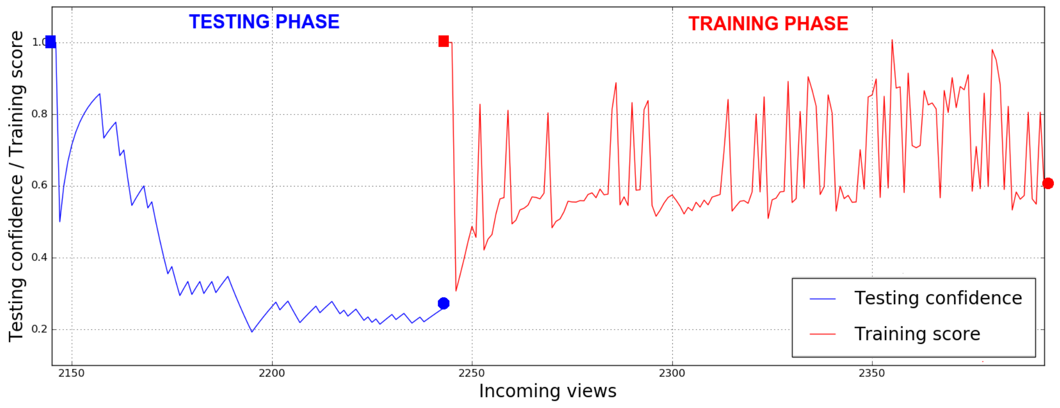

Figure 5.14: An example of one object during testing and triggered training phase. The lue square represents the beginning of the testing phase. The blue dot represents the end of the testing phase where a prediction is made. The testing confidence is shown during the testing phase. The red square is the beginning of the training phase, and the red dot is the end of the training phase. The training score is shown during the training phase.

Note this is just an example how a confidence and a training score graph might look like, and it is not a general rule. The confidence graph during testing varies in a specific way. It has a high value at the beginning. When only 1 or 2 views of the testing object are provided to the system (note that in the real-time scenario they are probably similar views), the confidence is usually very high. These testing samples usually have only one class among their neighbours, or they are in the outlier region. However, the confidence value drops significantly after a couple of more acquired views. When additional views are acquired by the system, some significant change in the feature space covered by the testing class happens. It might turned out that another class is a neighbour to some of the testing samples. The confidence value then starts to grow again as even more views are acquired. Now, the most recent samples are obtained in the direction which increases the system's confidence. For example, we may have two partially overlapping classes in the memory. Our testing samples are firstly in the region of only object 1 and then in the region of object 2 or in the overlapping region, and then

again in the region of the first object. This situation would develop the confidence graph of the previously described shape. Finally, from this point on, the confidence value is mainly dropping. This can be justified with more testing samples in the overlapping region, which are just making the decision harder for the system.
In general, a shape of the confidence graph can be: constant value, mainly increasing, mainly decreasing and a value which is varying a lot. The constant value is usually associated with new objects, where all samples of the testing object are in the outlier region. We assume that the other shapes are associated with the partial overlap among the testing object and known objects.

The desired shape of the testing confidence graph is given in Figure 5.15. Unlike the previous case, the confidence here is very low at the beginning, then varies a bit, and from a specific point on, primarily increases, making the probability of the correct prediction higher. In this case the decision is wrong with high probability if it is made at the beginning. However, if it is made after the acquisition of the additional views, it is correct with high probability.

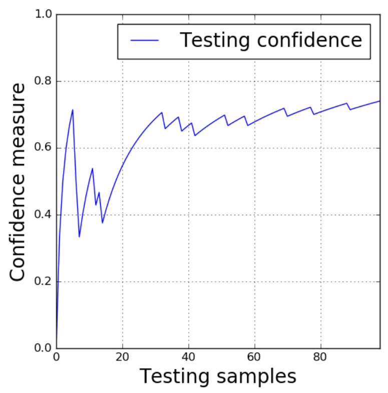

Figure 5.15: Confidence graph of one object during testing.

The training score graph in Figure 5.14 provides some meaningful insights, as well. The figure shows just an example of the graph, but, in general, the score graph has a shape shown in the figure. Firstly, it can be concluded that all the "spiky" values are the scores corresponding to the most valuable views, of the training object, for the system. Note that some of the scores given in the training phase in Figure 5.13 are higher than 1.0. These values correspond to the views which are of special value for the system. This can serve as a special reward, and as a warning, as well. If extremely high value is scored (e. g., 20), something must be either wrong or changed in the training setting. Because, it means that the training sample which scored the high value is extremely far away from the other training samples in the class. A reasonable assumption is that a significant change in the lightening of the room, during the training phase, could cause this. Additionally, in Figure 5.14 can be noticed that the score mean shifts slightly towards higher values over time. The thresholds which which are applied are changed during time as more views are acquired.

Figure 5.13 shows that the system has longer testing phases on the right side of the graph. More uncertain situations are present over time, as more objects are shown to the system. This finding is illustrated in Figure 5.16, which shows a two-dimensional feature space. The image on the right shows the memory of the system when 8 objects are shown to the system (indicated by the different colors). There are overlapping regions, classes which are relatively close in the feature space and more than two classes partially overlapping. The image on the left contains 2 out of 8 classes presented on the right. There are no overlapping regions, or relatively close classes. If a testing object is shown to the system, with only 2 objects previously seen, there is a smaller probability that the confidence of the prediction is low than if the system learned more objects. This is the reason why longer testing phases are visible at the right side of Figure 5.13 graph, then on the left side.

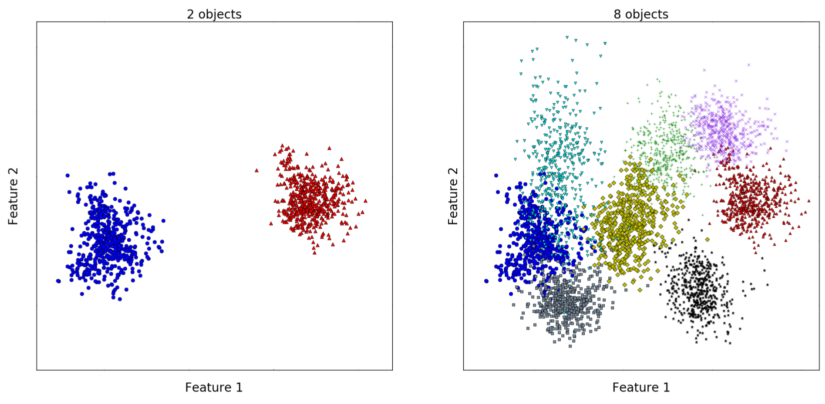

Figure 5.16: 2D feature space of the memory of the system. 2 objects are shown to the system on the image on the left. 8 objects are shown to the system on the image on the right. Each class is denoted with a different symbol, and it is corresponding to a different object shown to the system.

Figure 5.17 shows zoomed in view of the block 2 marked in green in Figure 5.13. This is the situation where the system is not sure at the beginning what the object is, so more views are required (testing phase 1). When more views are obtained the confidence increases and a correct prediction is made with high probability. Therefore, it is decided depending on the user that more training data is not necessary. The user can decide that the training is necessary in this case as well. Immediately after the end of the testing phase 1, another object is shown to the system. The testing phase of another object starts, followed by the training phase of that object (if needed).

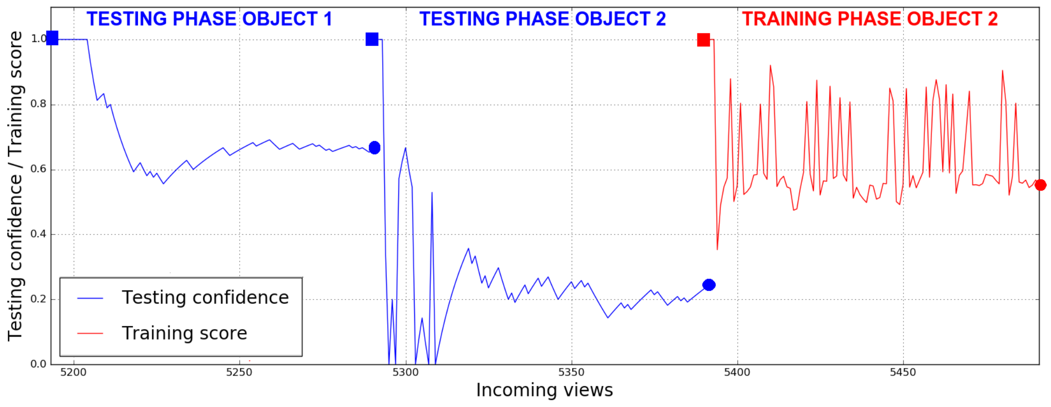

Figure 5.17: The testing phase of the first object and the testing and training phase of the second object. The first object does not need a training phase. The blue squares represent the beginning of the testing phase, whereas the blue circles represent the end of the corresponding phase. The red squares represent the beginning of the training phase, whereas the red circles represent the end of the phase. The testing confidence is shown during the testing phases, and the training score is shown during the training phase.

Figure 5.18 shows how many views per object the testing strategy requires in the interactive scenario to reach the confidence of 0.6 for each object, in this exemplary single run of the experiment. A minimum number of views per object is 10, and a maximum number of views is 100, even if the confidence is under 0.6. The thresholds and limits are adapted to the current experiments. For a new setting or data they might be set differently. Note that this is just one example, and that with some other order of objects or other objects in the learning process the graph would look differently. However, some general conclusions can be drawn.

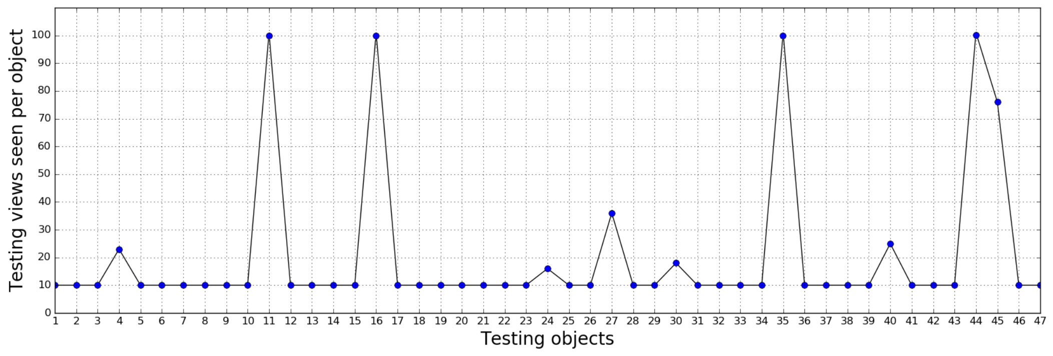

Figure 5.18: An example of how many views per object in the learning process the testing strategies require to reach the confidence level 0.6.

The testing strategy requires only a minimum number of views per object, for most of the objects. This is especially true at the beginning of the interaction when the system knows a small number of objects. In that specific situation, the minimum number of views is enough for the system to make a prediction. The requirement for additional views is more frequent at the right end of the graph, when the system learns more objects. Note as well, if the system knows a small number of objects and requires a maximum number of views of the current testing object this usually means that there is a significant confusion.
In the previous discussion we presented the important findings related with the online graphs and the behaviour of the confidence measure. An analysis of the behaviour of predicted labels is provided next. More meaningful insights into the learning process can be obtained from this analysis.

Predictions

Figure 5.19 shows an example of a change of the predicted label during the testing phase. In this case, the object is new for the system, and the system does not know the true label of the object (82 in this case). The label 0 indicates that the system predicts an object which is not known. As it can be seen the predicted label changes between 0 (new) and 106 (known). If the system was shown less than 30 views of the object it would make a wrong prediction. Since the system is alternating between a new and already known object this (most probably) means that some views of the testing object are in the outlier region of all known classes and some testing views are in the region of the object with label 106. Moreover, Figure 5.19 shows that the changes in the predicted label correspond to the sharp changes in the confidence. After approximately 35 testing views the system does not change the predicted label, and the confidence mainly increases. The object is correctly classified as new.

In general, in 1000-dimensional feature space with approximately 20 objects from the database in the interactive learning process, the predicted label of a testing object rarely alternates between more than two values. However, with an increase of objects in the learning process or a reduction of dimensions the overlap of objects in the feature space is expected to increase, and therefore the number of alternating values of a predicted label is expected to raise.

Figure 5.20 shows the case when the predicted label alternates between 4 values. When the system knows many objects and a new object is shown, there is a high probability that the new object will be partially overlapping in the feature space with some of the known objects. The more objects the system knows the probability is higher. In the case shown in the Figure 5.20 the system correctly predicts that the object is new. However, based on the change of the predicted label it is clear that the system has some internal overlap between the object 4 and objects 82, 88 and 99.

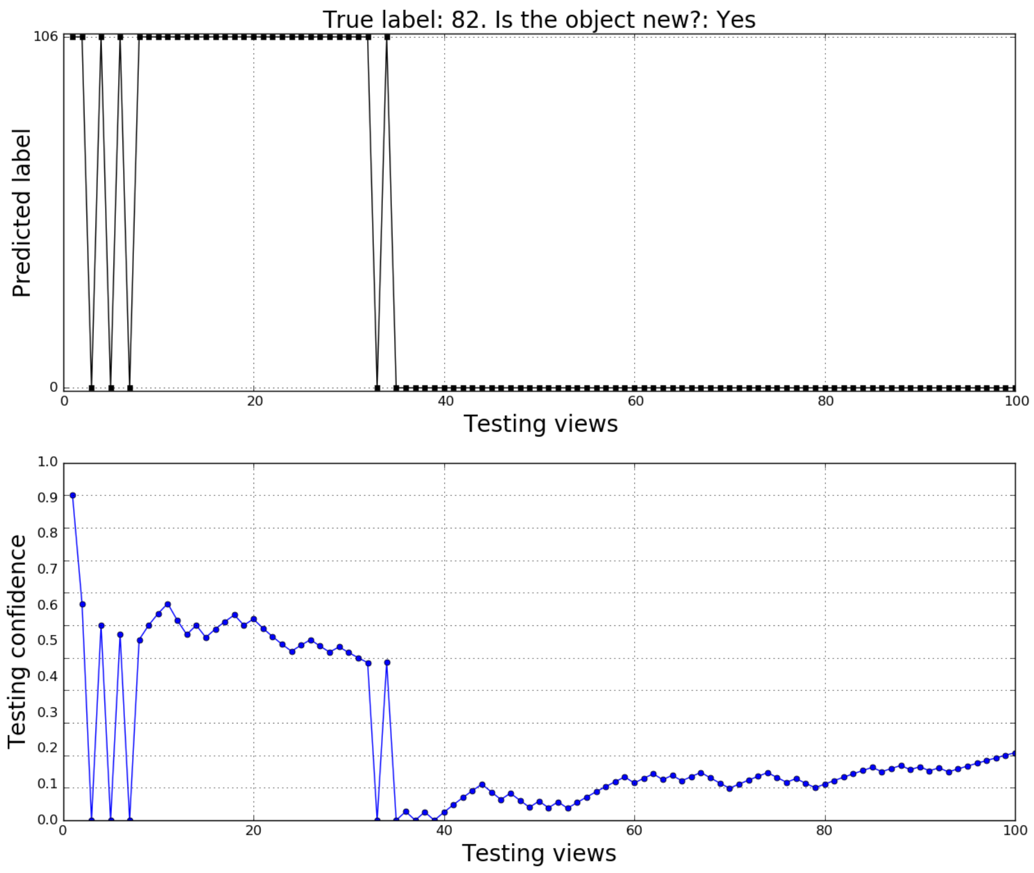

Figure 5.19: Upper graph: Change of the predicted label during the testing phase of one object. True label is 82, and the object is new for the system. The label value 0 denotes the prediction of a new object. Lower graph: Confidence of the prediction during the testing phase of that object.

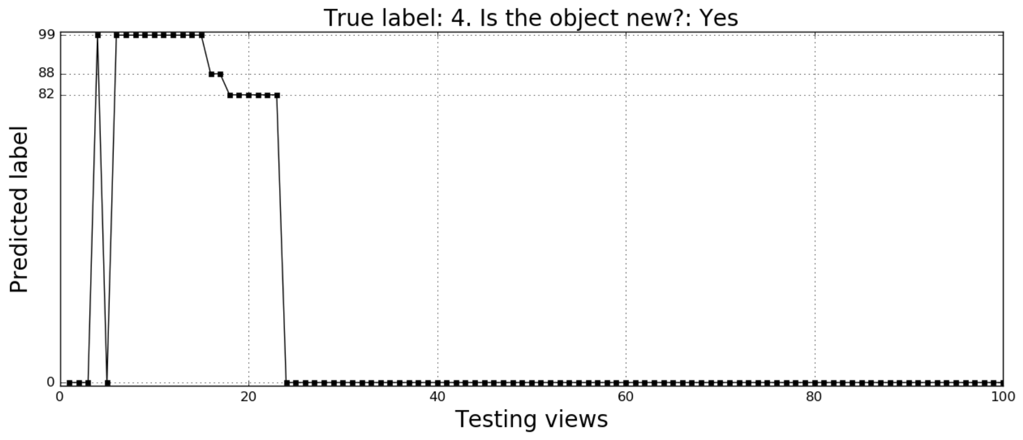

Figure 5.20: Maximal observed change of the predicted label during the testing phase of one object. True label is 4, and the object is new for the system. The predicted label alternates between 4 values. There are 20 objects in the learning process.

The user might benefit from getting the information about the change of the predicted label. For example, if the objects 82, 88 and 99 (Figure 5.20) are yellow color (or have some other common feature), but the object 4 has both yellow and a red side, in the future the user can show the red side to the system in order to increase the chance of the fast recognition. The baseline strategy would, for the case in Figure 5.20, predict a label 99, 88 or 82, depending on the number of the testing views which would be shown to the system.

Figure 5.21 presents the accuracy of the predictions. Correct predictions are marked as green circles. If a testing object is never seen, by the system, and predicted as new it is marked as a blue square. Wrong predictions are divided into three groups. If a known object is wrongly predicted as new it is marked with a yellow triangle. New objects which are recognized as known objects are marked as orange stars. Finally, if a known object is simply wrongly recognized among known objects it is marked as a red diamond.
Since the objects are taken randomly from a predefined pool of objects, it can be seen that at the beginning new objects are shown to the system more frequently than later. It is visible that the system recognizes mostly well new objects. However, it can be noted that the system makes mistakes in predicting a known object as new more often at the beginning. When only a couple of objects are shown to the system (like on the left image in Figure 5.16) there is a high probability that a small number of testing views of a known object is in the outlier region of all classes, and therefore predicted wrongly as a new object. On the other hand, confusions among known objects are more frequent when more objects are shown to the system. Figure 5.16 shows an example of the system's memory where more critical overlapping regions are present on the right image, than at the beginning when only a few objects are seen. When the already seen testing object is shown to the system, there is a higher probability that the testing samples are in the regions of one or more classes. Finally, at the beginning, it rarely happens that a new object is recognized as known object (mainly if the overlap of the testing object and the predicted known object is total in the feature space). However, when the system knows more objects the probability that a new object is recognized as a known object is higher. When a lot of objects is in the memory, it can be assumed that there must be some overlap of the testing object with some of the known objects. The more objects shown to the system, the smaller is the possible difference between the testing object and all known objects. The testing samples are closer to the regions of some objects than when fewer objects are shown to the system. Sometimes, this leads to the prediction of a new object as a known object.

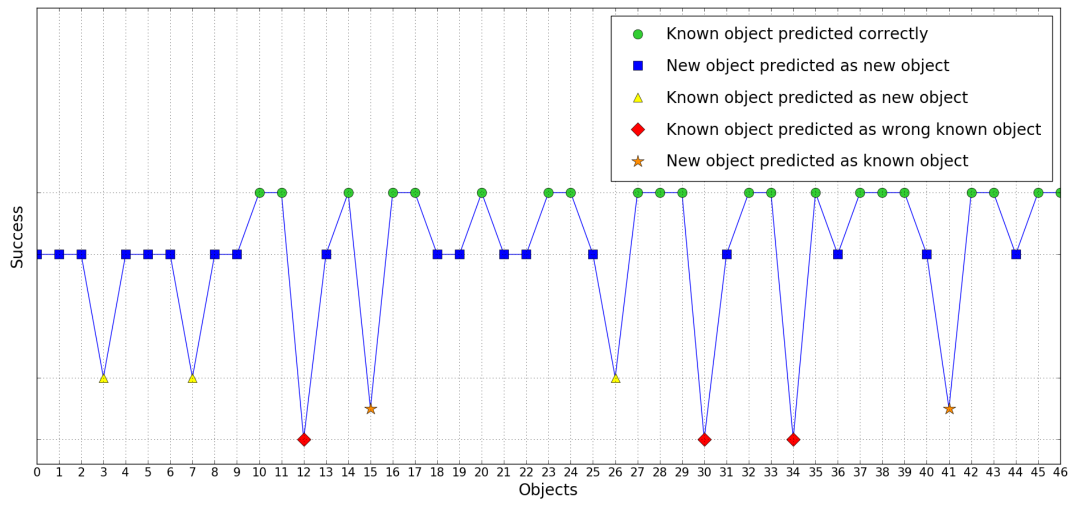

Figure 5.21: Success of the prediction for 46 objects shown to the system. The green circles represent correct predictions. The blue squares represent correct predictions of new objects. The yellow triangles represent erroneous predictions that a known object is predicted as a new object. The orange stars represent erroneous predictions that a new object is predicted as a known object, and the red diamonds that a known object is wrongly predicted as another known object.

To sum up, this subsection provides the discussion on the simulation of the interactive object learning system. We provided the insights into the working mechanism of the proposed testing and training strategy and the overall learning process. Some general conclusions are drawn, and some situation specific explanations are given. The next chapter concludes the thesis.

6 Conclusion

A collaboration between humans and machines enables the completion of complex tasks which cannot be performed individually by a human or a machine. In order to achieve effective collaboration a good communication is necessary. The main goal of this master thesis is the development and investigation of new communication strategies, which enrich the flow of information between the machine, Tubby platform, and the human, the user of the platform (Chapter 2). The crucial idea is the improvement of the communication between the user and the system by the implementation of new testing and training strategies. New strategies aim for the improvement through self-evaluation of classifiers. Firstly, a summary of the results obtained in the thesis is given. In Section 6.1 further research directions are pointed out.

At the beginning of the thesis we state a general motivation (Chapter 1) and a specific problem which is aimed to be solved (Chapter 2). We then introduce the theoretical background on the topics related with the problem (Chapter 3). Finally, as a result of the thesis we develop two measures of self-evaluation of classifiers (Chapter 4). The first developed measure captures the confidence of predictions. We propose the testing strategy which uses this measure. If the probability of making an error during the prediction is low, the confidence value is high. If the probability of making an error during the prediction is high, the confidence value is low. The second developed measure is a score which denotes how beneficial is the view of an object shown to the system. We propose a training strategy which uses this measure. The beneficiality of a training sample (view of an object) is essentially the similarity of the current training sample to the previously acquired training samples of the same object. If the current view of an object is different than all the previously shown views of the object the beneficiality value is high. On the contrary, if the current view is very similar to some of the acquired views the beneficiality value is low. Two ways of derivation are provided for each measure. Therefore, two testing and two training strategies are proposed. The aim of the testing strategies is to focus on cases when the confidence of predictions is low. In these situations the system requires more information (views of an object) to be shown. The aim of the training strategies is to focus on the extraction of the most beneficial information (views of an object) from the user, with respect to generalization performance of the classifier. All strategies use an online graph as a communication with the user. The proposed testing strategies show the confidence as graph, whereas the proposed training strategies show the score of beneficiality on a graph. The proposed strategies are evaluated against a baseline performance (Chapter 5). The evaluation is done across a number of dimensions of the feature space and for a different number of testing and training objects. The main findings are:

- The proposed strategies outperform the baseline performance, both separately the testing and training strategies and when combined. The baseline is outperformed from 1% to 25%.
- The combinations of the proposed testing and training strategies perform better or equal to the testing or training strategies separately. The performance of the Testing strategy 1 in combination with the Training strategy 1 is the best, compared to all other strategies.
- Three main factors which affect the performance of the proposed strategies are extracted:

   - Dimensionality of the operating feature space.
     The proposed training strategies improve the baseline more in the high dimensional space (≥100D). The proposed testing strategies improve the baseline more in the low dimensional space (≤10D). The improvement of the combinations of the proposed strategies with respect to the baseline performance does not show a visible dependency on the dimensionality.
   - Number of objects in the learning process.
     The proposed testing strategies improve the baseline more when a number of objects in the learning process is large (≥100, Setting 1 and 2). The training strategies improvement is dependent on the dimensionality of the feature space as well. If the dimensionality is lower or equal to 10, the training strategies improve the baseline less when the number of objects is increased. However, in the highly dimensional space (≥100D) the training strategies improve the baseline more for a larger number of objects. Regarding the combinations of the strategies, in two-dimensional feature space the improvement of the baseline does not show a dependency on the number of objects. In higher dimensions, however, the combinations improve the baseline more when the number of objects is higher.
   - Whether all testing objects are in the training object set, or not.
     The testing strategies improve the baseline more when the testing and training set are equal, in all dimensions. The training strategies' performance depends on the dimensionality of the feature space. In two-dimensional feature space the improvement of the baseline is larger when the object sets are different, i.e. the testing object set contains objects which are not used for training. In 10-dimensional feature space the improvement of the training strategies does not show a dependency on the equality of the object sets. In higher dimensions, however, the improvement is better when the object sets are equal. The improvement of the baseline caused by the implementation of the combinations of the proposed strategies is in general better when the training and testing object set are the same.

- The best improvement of the baseline is achieved when the dimensionality of the feature space is higher or equal to 10 and for a large number of objects in the learning process (≥100, Setting 1 and 2), which is the most realistic case when real applications are considered.

6.1. Future Research

The previous discussion provides directions for the future research.

The first direction is towards a practical implementation of the strategies which Chapter 4 proposes. The proposed testing and training strategies should be deployed on Tubby platform and their effect on the communication quality and the prediction performance should be assessed. In this regard the following questions are important. Are the proposed methods fast enough for the real-time computation? Do the conclusions from the offline evaluation hold in the online case? Are the assumptions about users' behaviour during testing and training of an object correct? Is the proposed way of communication with the system useful and intuitive for the user?

Another direction for the future research is an incorporation of the proposed methods to work with more powerful classifiers. kNN is effective, yet very simple algorithm. It is expected that further improvement in the classification accuracy can be achieved with advanced classification algorithms.

In Chapter 2 we state that Tubby was developed with the aim to learn visual categories. Due to complexity of the categorization task we used the simpler task of object recognition. Our focus was on the communication improvement. Since the communication improvement basis is set, the proposed strategies should be adapted to work with categorization task. Finally, the adapted strategies should be investigated and deployed on Tubby.

Bibliography

[1] G. Amato and F. Falchi. knn based image classification relying on local feature similarity. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, SISAP '10, pages 101–108, New York, NY, USA, 2010. ACM.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[3] A. Bookstein, V. A. Kulyukin, and T. Raita. Generalized hamming distance. *Information Retrieval*, 5(4):353–375, Oct 2002.

[4] L. Bottou. On-line learning in neural networks. chapter On-line Learning and Stochastic Approximations, pages 9–42. Cambridge University Press, New York, NY, USA, 1998.

[5] L. Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In Y. Lechevallier and G. Saporta, editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France, Aug. 2010. Springer.

[6] M. Browne, S. S. Ghidary, and N. M. Mayer. *Convolutional Neural Networks for Image Processing with Applications in Mobile Robotics*, pages 327–349. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[7] R. Cardenes, M. Sanchez, and J. Ruiz-Alzola. Computational geometry computation and knn segmentation in itk. 06 2006.

[8] Deeplearning4j Development Team. *Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0*.

[9] T. Dharani and I. L. Aroquiaraj. Content based image retrieval system using feature classification with modified KNN algorithm. *CoRR*, abs/1307.4717, 2013.

[10] M. Dhawan, S. Selvaraja, and Z. Duan. Application of committee knn classifiers for gene expression profile classification. *IJBRA*, 6(4):344–352, 2010.

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Ed)*. Wiley, 2001.

[12] J. Enns. *The Thinking Eye, the Seeing Brain: Explorations in Visual Cognition*. W.W. Norton, 2004.

[13] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[14] L. Fischer, S. Hasler, S. Schrom, and H. Wersing. Improving online learning of visual categories by deep features. submitted, NIPS Workshop: Future of Interactive Learning Machines, 2016.

[15] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. J. Gordon and D. B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[16] C. Goutte and É. Gaussier. A probabilistic interpretation of precision, recall and *F*-score, with implication for evaluation. In D. E. Losada and J. M. Fernández-Luna, editors, *Advances in Information Retrieval, 27th European Conference on IR Research, ECIR 2005, Santiago de Compostela, Spain, March 21-23, 2005, Proceedings*, volume 3408 of *Lecture Notes in Computer Science*, pages 345–359. Springer, 2005.

[17] S. Harmeling, G. Dornhege, D. Tax, F. Meinecke, and K.-R. Müller. From outliers to prototypes: Ordering data. *Neurocomputing*, 69(13-15):1608–1618, Aug. 2006.

[18] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society, 2015.

[19] L. Hertel, E. Barth, T. Käster, and T. Martinetz. Deep convolutional neural networks as generic feature extractors. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–4, 2015.

[20] S. C. H. Hoi, J. Wang, and P. Zhao. LIBOL: a library for online learning algorithms. *Journal of Machine Learning Research*, 15(1):495–499, 2014.

[21] K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[22] J. Huang and C. X. Ling. Constructing new and better evaluation measures for machine learning. In M. M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 859–864, 2007.

[23] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

[24] A. Jaech, L. Heck, and M. Ostendorf. Domain adaptation of recurrent neural networks for natural language understanding. *CoRR*, abs/1604.00117, 2016.

[25] N. Japkowicz. *Concept Learning in the Absence of Counter-Examples: An Autoassociation-Based Approach to Classification*. PhD thesis, The State University of New Jersey, 1999.

[26] A. Karpathy. Stanford University CS231n: Convolutional Neural Networks for Visual Recognition. 2015.

[27] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li. Large-scale video classification with convolutional neural networks. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pages 1725–1732. IEEE Computer Society, 2014.

[28] S. Kim, J. Sim, and J. Lee. Fuzzy *k*-nearest neighbor method for protein secondary structure prediction and its parallel implementation. In D. Huang, K. Li, and G. W. Irwin, editors, *Computational Intelligence and Bioinformatics, International Conference on Intelligent Computing, ICIC 2006, Kunming, China, August 16-19, 2006. Proceedings, Part III*, volume 4115 of *Lecture Notes in Computer Science*, pages 444–453. Springer, 2006.

[29] S. Kirstein, A. Denecke, S. Hasler, H. Wersing, H.-M. Gross, and E. Körner. A vision architecture for unconstrained and incremental learning of multiple categories. *Memetic Computing*, 1(4):291–304, 2009.

[30] A. Kolchinsky, A. Abi-Haidar, J. Kaur, A. A. Hamed, and L. M. Rocha. Classification of protein-protein interaction full-text documents using text and citation network features. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3):400–411, July 2010.

[31] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[32] N. Lachiche and P. A. Flach. Improving accuracy and cost of two-class and multi-class probabilistic classifiers using ROC curves. In T. Fawcett and N. Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 416–423. AAAI Press, 2003.

[33] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[34] M. McCloskey and N. J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pages 109 – 165. Academic Press, 1989.

[35] M. McLaren, Y. Lei, N. Scheffer, and L. Ferrer. Application of convolutional neural networks to speaker recognition in noisy conditions. In H. Li, H. M. Meng, B. Ma, E. Chng, and L. Xie, editors, *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 686–690. ISCA, 2014.

[36] M. Mermillod, A. Bugaiska, and P. Bonin. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*, 4:504–510, 2013.

[37] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[38] Y. Park, H. Hwang, and S. goo Lee. A novel algorithm for scalable k-nearest neighbour graph construction. *Journal of Information Science*, 42(2):274–288, 2016.

[39] E. Patrick and F. Fischer. A generalized k-nearest neighbor rule. *Information and Control*, 16(2):128 – 152, 1970.

[40] R. Polikar and C. Alippi. Guest editorial learning in nonstationary and evolving environments. *IEEE Trans. Neural Netw. Learning Syst.*, 25(1):9–11, 2014.

[41] D. M. W. Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[42] Y. Qian, M. Bi, T. Tan, and K. Yu. Very deep convolutional neural networks for noise robust speech recognition. *IEEE/ACM Trans. Audio, Speech & Language Processing*, 24(12):2263–2276, 2016.

[43] J. Radojevic. Cooperative visual category learning. Technical report, Delft University of Technology, Honda Research Institute - EU, March, 2017.

[44] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH, USA, June 23-28, 2014*, pages 512–519. IEEE Computer Society, 2014.

[45] F. M. Richardson and M. S. Thomas. Critical periods and catastrophic interference effects in the development of self-organizing feature maps. *Developmental Science*, 11(3):371–389, 2008.

[46] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Comput.*, 13(7):1443–1471, July 2001.

[47] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Inf. Process. Manage.*, 45(4):427–437, 2009.

[48] P. Soucy and G. W. Mineau. Beyond tfidf weighting for text categorization in the vector space model. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1130–1135, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc.

[49] S. V. Stehman. Selecting and Interpreting Measures of Thematic Classification Accuracy. *Remote Sensing of Environment*, 62(1):77–89, 1997.

[50] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, and S. Tubaro. Deep convolutional neural networks for pedestrian detection. *Sig. Proc.: Image Comm.*, 47:482–489, 2016.

[51] V. N. Vapnik and A. Y. Chervonenkis. *On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities*. Springer International Publishing, Cham, 2015.

[52] J. Weston, O. Chapelle, and I. Guyon. Data cleaning algorithms with applications to micro-array experiments. Technical report, Biowulf, 2001.

Appendix

Setting 1 - 1000D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0993	0.2456	0.2439	0.0906	0.0983	0.2431	0.2433	0.2063	0.1262
5	0.3000	0.3310	0.3484	0.3668	0.3651	0.4308	0.4391	0.4217	0.3729
10	0.3463	0.3792	0.4117	0.4674	0.4583	0.5314	0.5333	0.5125	0.4967
15	0.3777	0.4304	0.4274	0.5187	0.5167	0.5815	0.5871	0.5683	0.5606
20	0.4030	0.4705	0.4619	0.5567	0.5474	0.6262	0.6300	0.6164	0.6060
30	0.4388	0.4985	0.5321	0.6089	0.6025	0.6845	0.6863	0.6683	0.6633
40	0.4671	0.5418	0.5489	0.6504	0.6370	0.7173	0.7133	0.7094	0.6973
50	0.4933	0.5690	0.5836	0.6705	0.6673	0.7437	0.7507	0.7373	0.7321
70	0.5262	0.5993	0.6008	0.7133	0.7010	0.7783	0.7779	0.7738	0.7652
80	0.5359	0.6310	0.6211	0.7256	0.7061	0.7842	0.7783	0.7919	0.7740
100	0.5642	0.6465	0.6563	0.7475	0.7289	0.8031	0.7908	0.8041	0.7829
120	0.5889	0.6783	0.6621	0.7697	0.7304	0.8222	0.8010	0.8189	0.7798
130	0.5943	0.6728	0.6921	0.7789	0.7367	0.8270	0.7971	0.8169	0.7898
150	0.6096	0.6861	0.6971	0.7801	0.7428	0.8401	0.8014	0.8328	0.7965
160	0.6222	0.6806	0.7040	0.7944	0.7550	0.8371	0.8026	0.8411	0.7966
180	0.6226	0.7024	0.7067	0.7915	0.7600	0.8472	0.7981	0.8456	0.7991
200	0.6307	0.7163	0.7133	0.8013	0.7594	0.8488	0.8092	0.8464	0.8048

Table 6.1: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where all 126 objects are used for the testing and for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

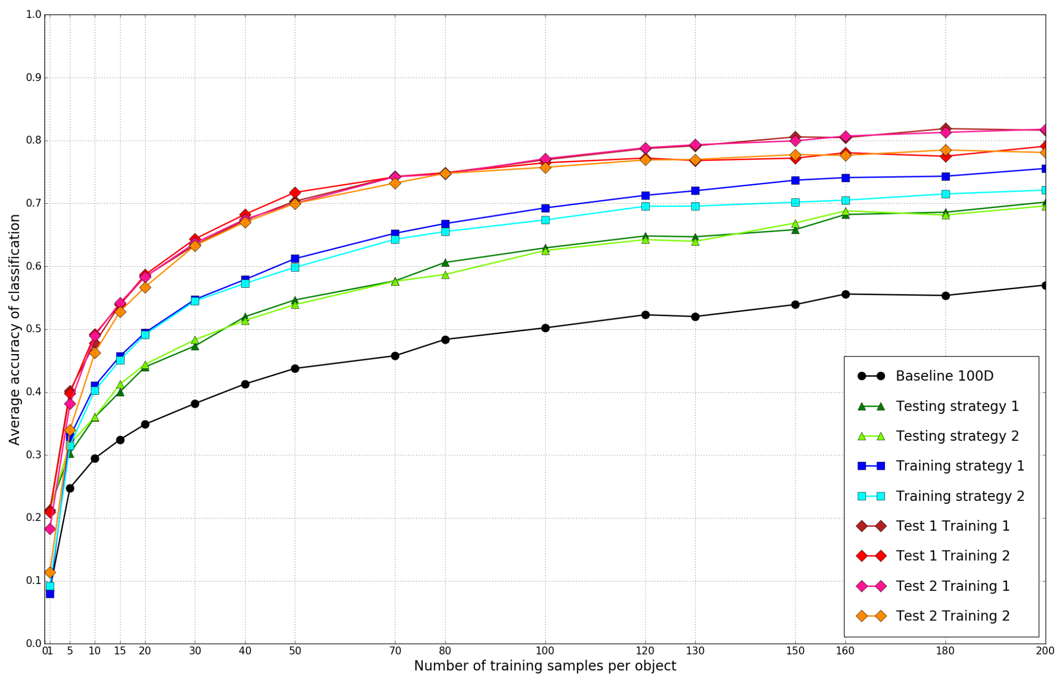

Figure 6.1: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where all 126 objects are used for the testing and for the training.

Setting 2 - 1000D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1097	0.2185	0.1876	0.0930	0.1090	0.2071	0.2050	0.1768	0.0994
5	0.2955	0.2911	0.2698	0.3454	0.3457	0.3679	0.3756	0.3527	0.3017
10	0.3267	0.3359	0.3307	0.4308	0.4243	0.4408	0.4454	0.4354	0.4115
15	0.3543	0.3636	0.3586	0.4772	0.4718	0.4811	0.4956	0.4822	0.4617
20	0.3779	0.3921	0.4006	0.5012	0.4963	0.5240	0.5211	0.5145	0.4918
30	0.4053	0.4222	0.4228	0.5397	0.5430	0.5483	0.5658	0.5595	0.5389
40	0.4346	0.4411	0.4564	0.5674	0.5708	0.5818	0.5951	0.5835	0.5621
50	0.4527	0.4612	0.4633	0.5919	0.5951	0.6074	0.6106	0.5951	0.5908
70	0.4839	0.5099	0.5068	0.6117	0.6168	0.6252	0.6352	0.6363	0.6120
80	0.4869	0.5121	0.5192	0.6271	0.6199	0.6381	0.6361	0.6395	0.6160
100	0.5138	0.5289	0.5274	0.6430	0.6318	0.6542	0.6464	0.6568	0.6234
120	0.5315	0.5471	0.5263	0.6594	0.6317	0.6619	0.6479	0.6677	0.6383
130	0.5324	0.5520	0.5360	0.6573	0.6327	0.6627	0.6510	0.6616	0.6338
150	0.5446	0.5648	0.5647	0.6633	0.6373	0.6699	0.6520	0.6748	0.6427
160	0.5479	0.5693	0.5727	0.6695	0.6412	0.6785	0.6542	0.6738	0.6423
180	0.5542	0.5839	0.5729	0.6637	0.6447	0.6747	0.6560	0.6831	0.6510
200	0.5632	0.5781	0.5855	0.6752	0.6500	0.6829	0.6605	0.6882	0.6533

Table 6.2: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

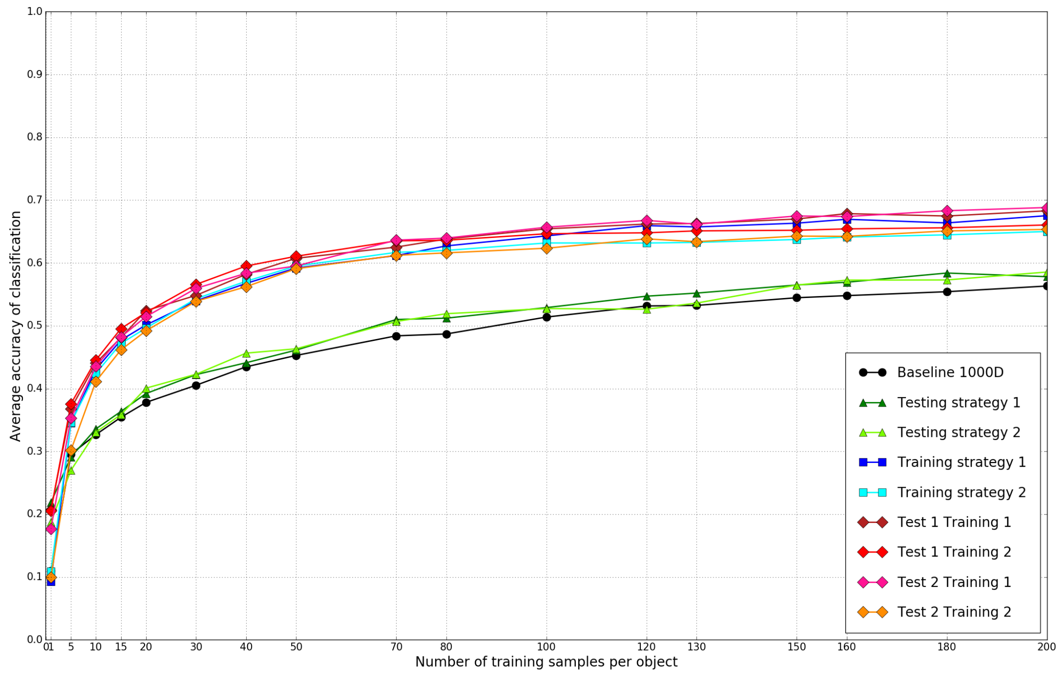

Figure 6.2: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training.

Setting 3 - 1000D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1256	0.3752	0.3285	0.110	0.133	0.3633	0.3833	0.1631	0.0082
5	0.4238	0.4328	0.4308	0.475	0.494	0.5200	0.5113	0.4980	0.2539
10	0.4548	0.4568	0.4631	0.541	0.533	0.5687	0.5560	0.5387	0.4291
15	0.4658	0.4843	0.4798	0.560	0.580	0.5693	0.5787	0.5718	0.4838
20	0.4858	0.4928	0.5010	0.577	0.592	0.6020	0.5873	0.5890	0.5410
30	0.5212	0.5243	0.5204	0.601	0.615	0.6020	0.6107	0.6092	0.5589
40	0.5236	0.5275	0.5304	0.605	0.632	0.6173	0.6213	0.6190	0.5965
50	0.5378	0.5452	0.5482	0.619	0.642	0.6287	0.6233	0.6231	0.5976
70	0.5588	0.5603	0.5594	0.623	0.637	0.6293	0.6320	0.6397	0.6163
80	0.5582	0.5625	0.5641	0.624	0.642	0.6373	0.6327	0.6272	0.6146
100	0.5616	0.5753	0.5745	0.646	0.649	0.6453	0.6340	0.6385	0.6239
120	0.5734	0.5808	0.5853	0.640	0.649	0.6413	0.6327	0.6427	0.6231
130	0.5806	0.5831	0.5942	0.635	0.643	0.6393	0.6367	0.6447	0.6274
150	0.5842	0.5932	0.5861	0.639	0.645	0.6473	0.6360	0.6451	0.6315
160	0.5890	0.5921	0.5923	0.632	0.650	0.6367	0.6347	0.6425	0.6232
180	0.5920	0.5916	0.5960	0.632	0.655	0.6433	0.6333	0.6451	0.6340
200	0.5942	0.5952	0.5983	0.633	0.646	0.6487	0.6340	0.6424	0.6336

Table 6.3: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

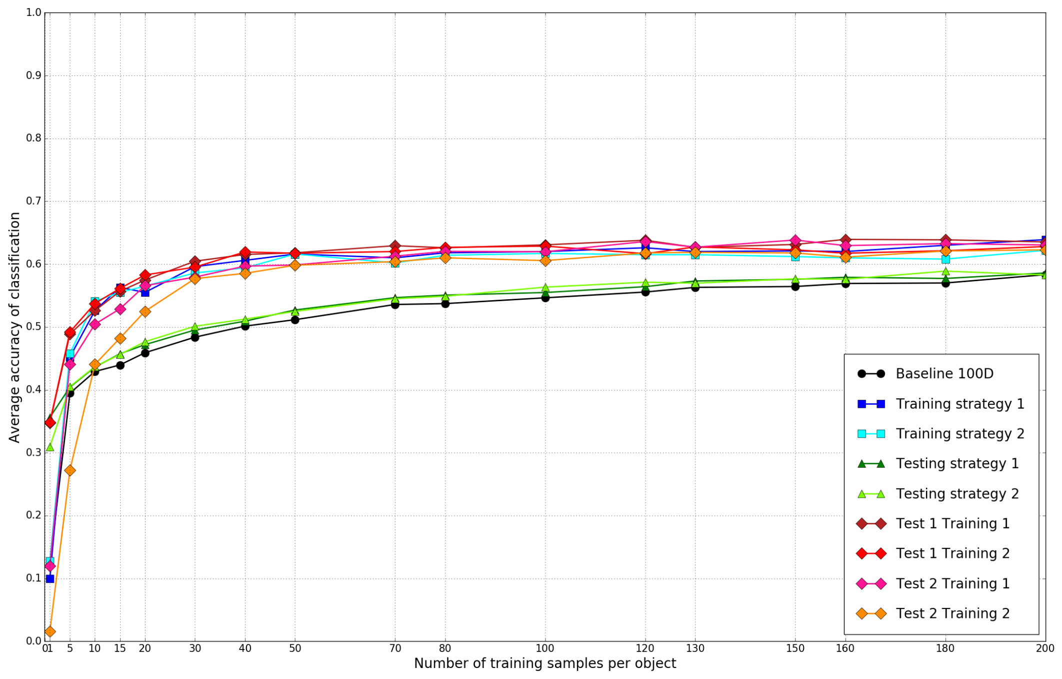

Figure 6.3: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training.

Setting 4 - 1000D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1838	0.5580	0.4714	0.156	0.186	0.574	0.574	0.171	0.019
5	0.6374	0.6414	0.6370	0.750	0.715	0.767	0.765	0.699	0.385
10	0.6784	0.6958	0.7008	0.848	0.824	0.831	0.844	0.787	0.607
15	0.7100	0.7294	0.7298	0.874	0.855	0.875	0.861	0.828	0.719
20	0.7400	0.7552	0.7484	0.901	0.890	0.900	0.889	0.845	0.761
30	0.7702	0.7888	0.7892	0.905	0.906	0.921	0.920	0.913	0.825
40	0.7804	0.8012	0.8142	0.935	0.913	0.929	0.923	0.918	0.858
50	0.8088	0.8046	0.8224	0.938	0.920	0.920	0.951	0.934	0.859
70	0.8328	0.8350	0.8372	0.948	0.931	0.945	0.955	0.940	0.895
80	0.8410	0.8592	0.8450	0.957	0.946	0.949	0.958	0.943	0.894
100	0.8370	0.8740	0.8686	0.948	0.955	0.953	0.960	0.945	0.911
120	0.8660	0.8694	0.8792	0.950	0.945	0.962	0.952	0.951	0.906
130	0.8708	0.8810	0.8750	0.951	0.964	0.967	0.952	0.960	0.921
150	0.8760	0.8864	0.8820	0.954	0.946	0.970	0.948	0.963	0.900
160	0.8866	0.8948	0.8924	0.949	0.954	0.955	0.946	0.957	0.906
180	0.8836	0.8926	0.8972	0.962	0.954	0.971	0.944	0.956	0.907
200	0.8890	0.8962	0.9002	0.961	0.951	0.963	0.945	0.966	0.920

Table 6.4: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where same 10 objects are used for the testing and training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

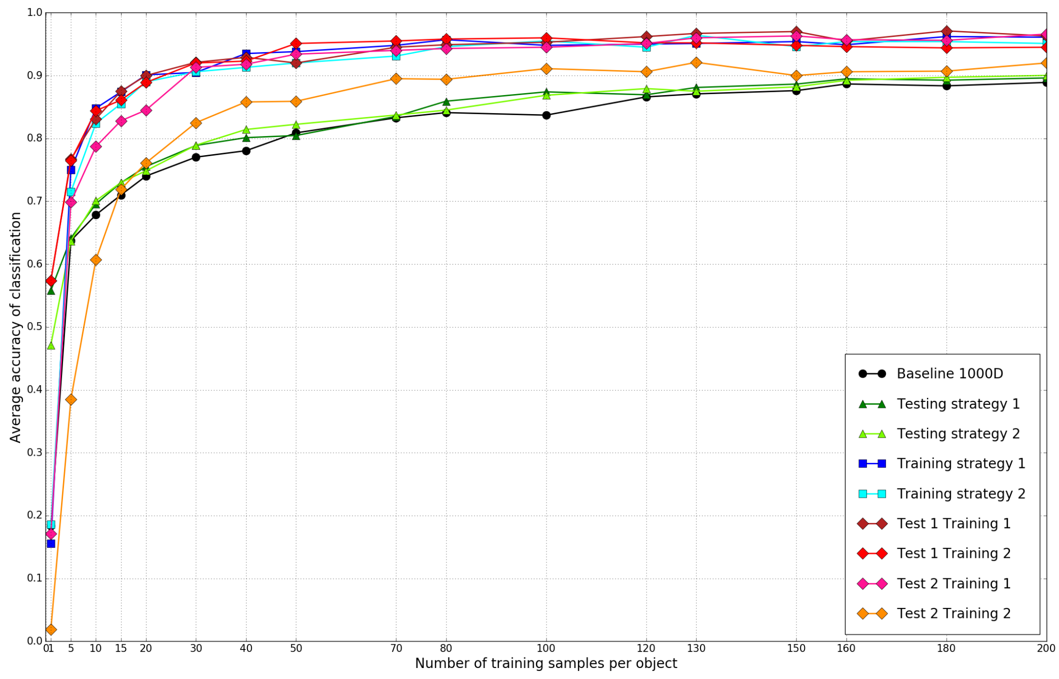

Figure 6.4: Classification accuracy of the proposed strategies in 1000-dimensional feature space for the case where same 10 objects are used for the testing and training.

Setting 1 - 100D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0854	0.2162	0.2136	0.0798	0.0922	0.2125	0.2093	0.1826	0.1138
5	0.2477	0.3028	0.3175	0.3290	0.3150	0.4021	0.3983	0.3814	0.3399
10	0.2949	0.3603	0.3606	0.4102	0.4027	0.4784	0.4924	0.4898	0.4629
15	0.3244	0.4005	0.4125	0.4570	0.4509	0.5392	0.5394	0.5421	0.5281
20	0.3488	0.4398	0.4441	0.4944	0.4915	0.5836	0.5865	0.5830	0.5670
30	0.3819	0.4734	0.4833	0.5474	0.5449	0.6340	0.6436	0.6371	0.6331
40	0.4130	0.5199	0.5140	0.5790	0.5729	0.6733	0.6827	0.6749	0.6702
50	0.4377	0.5466	0.5391	0.6122	0.5984	0.7035	0.7175	0.7000	0.6998
70	0.4579	0.5769	0.5763	0.6525	0.6429	0.7429	0.7421	0.7423	0.7324
80	0.4838	0.6062	0.5870	0.6679	0.6553	0.7484	0.7488	0.7467	0.7477
100	0.5022	0.6293	0.6252	0.6929	0.6737	0.7693	0.7646	0.7711	0.7574
120	0.5230	0.6483	0.6425	0.7129	0.6955	0.7872	0.7721	0.7884	0.7691
130	0.5203	0.6470	0.6397	0.7202	0.6957	0.7916	0.7682	0.7933	0.7698
150	0.5392	0.6583	0.6688	0.7369	0.7020	0.8057	0.7720	0.7994	0.7776
160	0.5559	0.6824	0.6883	0.7410	0.7052	0.8046	0.7806	0.8071	0.7764
180	0.5537	0.6860	0.6816	0.7433	0.7151	0.8189	0.7750	0.8129	0.7849
200	0.5702	0.7021	0.6960	0.7556	0.7212	0.8163	0.7910	0.8179	0.7810

Table 6.5: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where all 126 objects are used for the testing and for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

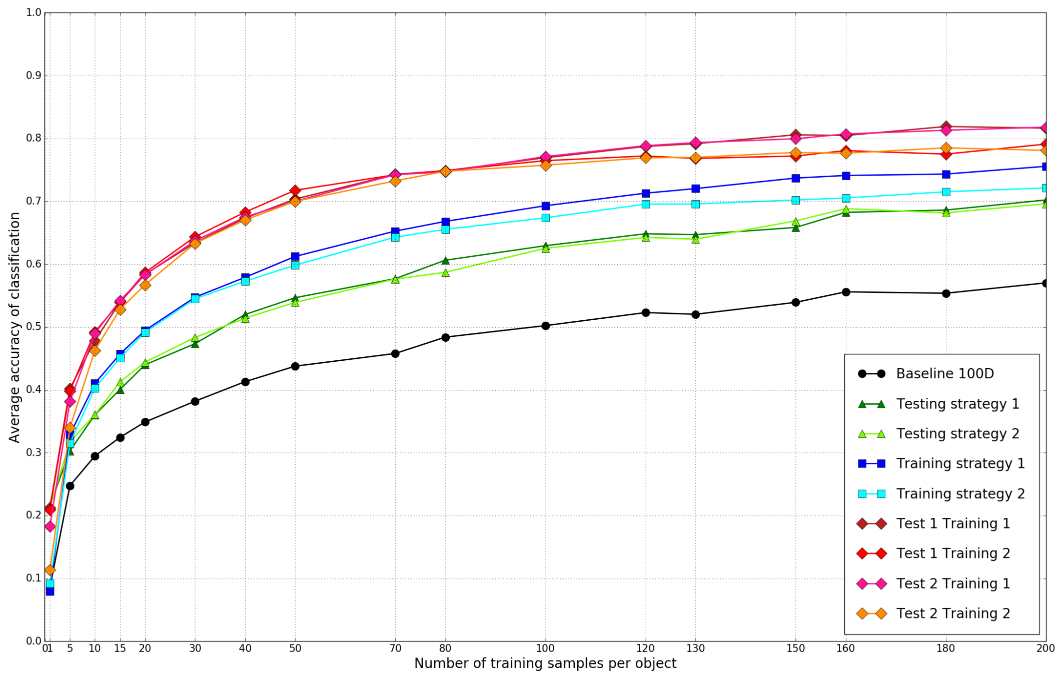

Figure 6.5: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where all 126 objects are used for the testing and for the training.

Setting 2 - 100D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1071	0.1809	0.1917	0.0894	0.0992	0.1787	0.1819	0.1580	0.0974
5	0.2642	0.2587	0.2610	0.3028	0.3031	0.3351	0.3334	0.3250	0.2939
10	0.3079	0.2988	0.3048	0.3889	0.3835	0.4042	0.4059	0.4036	0.3810
15	0.3312	0.3486	0.3469	0.4266	0.4227	0.4541	0.4522	0.4420	0.4353
20	0.3553	0.3628	0.3713	0.4564	0.4483	0.4800	0.4919	0.4805	0.4702
30	0.3838	0.4160	0.3882	0.4941	0.4906	0.5217	0.5325	0.5254	0.5186
40	0.4093	0.4232	0.4257	0.5260	0.5201	0.5529	0.5581	0.5436	0.5490
50	0.4242	0.4421	0.4509	0.5443	0.5387	0.5650	0.5805	0.5715	0.5722
70	0.4523	0.4785	0.4685	0.5770	0.5650	0.6028	0.6035	0.5996	0.5957
80	0.4689	0.4907	0.4966	0.5905	0.5713	0.6106	0.6129	0.6054	0.6054
100	0.4876	0.5152	0.5059	0.6049	0.5868	0.6213	0.6240	0.6191	0.6184
120	0.4994	0.5123	0.5144	0.6181	0.5916	0.6360	0.6263	0.6325	0.6287
130	0.5045	0.5294	0.5183	0.6207	0.5989	0.6410	0.6271	0.6316	0.6291
150	0.5152	0.5464	0.5410	0.6226	0.6055	0.6501	0.6291	0.6444	0.6315
160	0.5219	0.5335	0.5579	0.6274	0.6070	0.6547	0.6283	0.6457	0.6342
180	0.5309	0.5467	0.5483	0.6353	0.6089	0.6534	0.6355	0.6537	0.6439
200	0.5452	0.5556	0.5583	0.6438	0.6145	0.6617	0.6408	0.6628	0.6408

Table 6.6: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

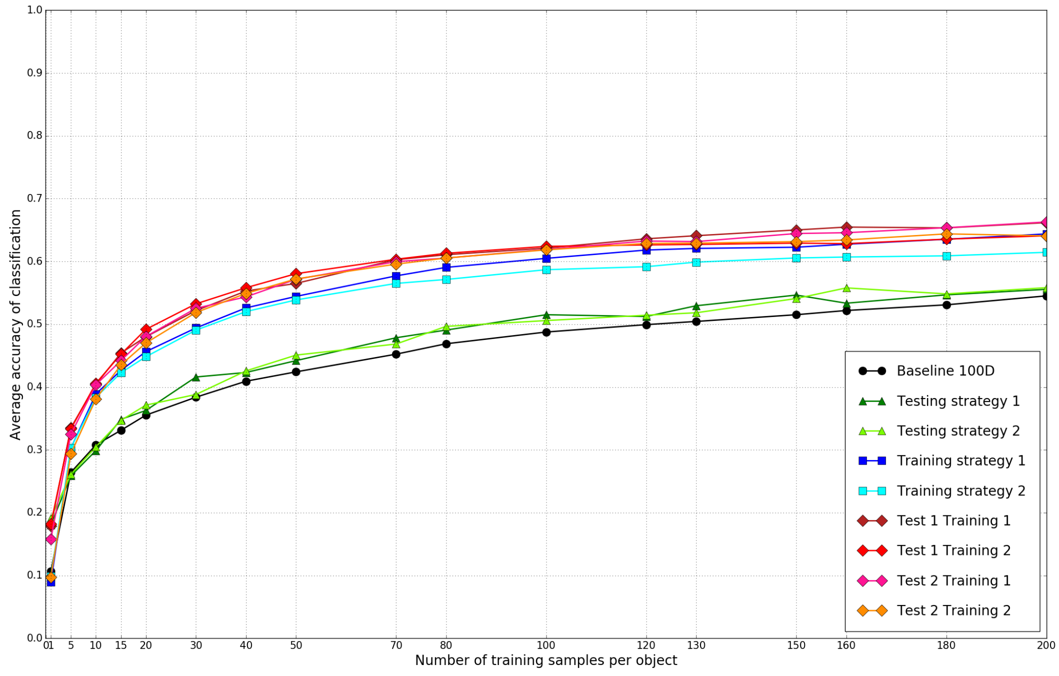

Figure 6.6: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training.

Setting 3 - 100D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1210	0.3561	0.3096	0.100	0.128	0.3473	0.3487	0.1201	0.0159
5	0.3946	0.4043	0.4051	0.452	0.458	0.4887	0.4920	0.4410	0.2722
10	0.4292	0.4361	0.4369	0.526	0.541	0.5260	0.5373	0.5045	0.4405
15	0.4396	0.4573	0.4562	0.563	0.555	0.5567	0.5613	0.5288	0.4822
20	0.4590	0.4719	0.4766	0.555	0.565	0.5747	0.5827	0.5657	0.5249
30	0.4838	0.4952	0.5012	0.596	0.586	0.6047	0.5953	0.5794	0.5769
40	0.5014	0.5092	0.5124	0.606	0.594	0.6153	0.6193	0.5968	0.5853
50	0.5116	0.5271	0.5248	0.616	0.616	0.6180	0.6173	0.5986	0.5982
70	0.5358	0.5467	0.5450	0.610	0.602	0.6293	0.6200	0.6126	0.6040
80	0.5372	0.5507	0.5488	0.618	0.614	0.6260	0.6267	0.6203	0.6100
100	0.5464	0.5548	0.5634	0.620	0.617	0.6307	0.6287	0.6197	0.6056
120	0.5556	0.5641	0.5712	0.626	0.615	0.6380	0.6167	0.6360	0.6177
130	0.5630	0.5732	0.5697	0.620	0.615	0.6267	0.6273	0.6270	0.6188
150	0.5644	0.5759	0.5767	0.621	0.612	0.6313	0.6227	0.6384	0.6178
160	0.5692	0.5791	0.5759	0.620	0.610	0.6393	0.6173	0.6295	0.6111
180	0.5700	0.5772	0.5888	0.630	0.608	0.6387	0.6213	0.6326	0.6209
200	0.5832	0.5859	0.5827	0.639	0.622	0.6353	0.6280	0.6305	0.6228

Table 6.7: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

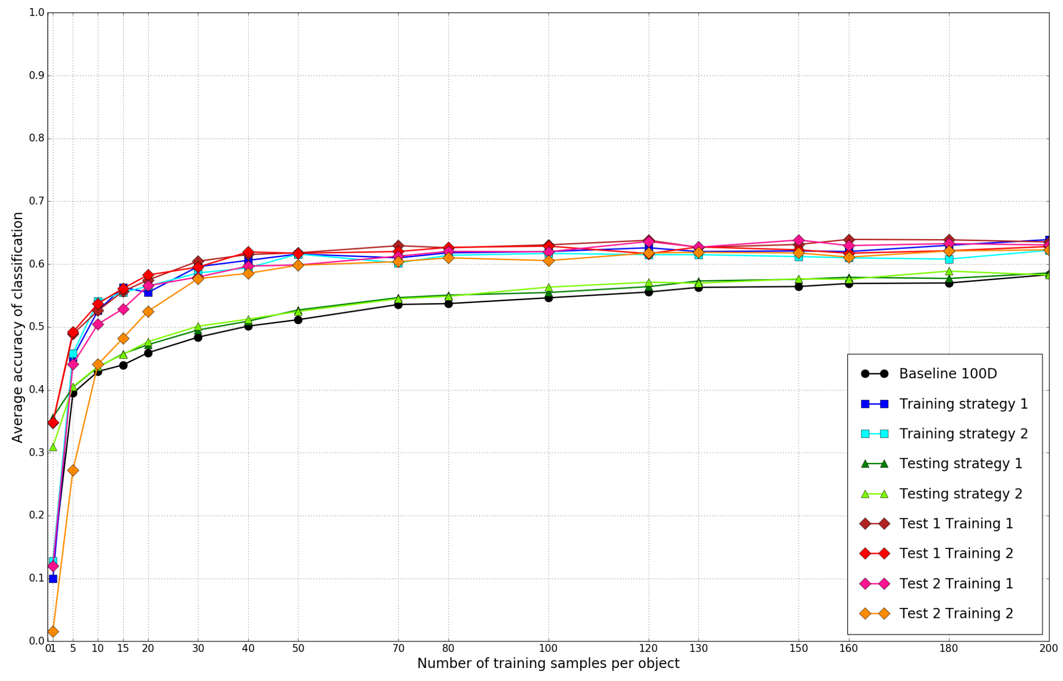

Figure 6.7: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training.

Setting 4 - 100D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1808	0.5284	0.4294	0.155	0.180	0.511	0.533	0.139	0.025
5	0.5890	0.6160	0.6064	0.688	0.708	0.735	0.748	0.636	0.412
10	0.6508	0.6510	0.6622	0.794	0.768	0.764	0.817	0.766	0.640
15	0.6760	0.6972	0.6942	0.805	0.820	0.813	0.857	0.784	0.730
20	0.6968	0.7132	0.7160	0.840	0.851	0.865	0.863	0.821	0.779
30	0.7304	0.7532	0.7532	0.884	0.879	0.890	0.885	0.873	0.842
40	0.7442	0.7736	0.7754	0.891	0.897	0.889	0.941	0.908	0.853
50	0.7690	0.7884	0.7842	0.901	0.904	0.905	0.928	0.897	0.879
70	0.8020	0.8216	0.8196	0.910	0.918	0.935	0.928	0.920	0.881
80	0.8050	0.8212	0.8300	0.937	0.916	0.921	0.928	0.924	0.897
100	0.8190	0.8578	0.8488	0.940	0.908	0.935	0.924	0.922	0.900
120	0.8380	0.8662	0.8552	0.938	0.933	0.947	0.935	0.937	0.902
130	0.8360	0.8536	0.8550	0.920	0.922	0.933	0.930	0.945	0.895
150	0.8464	0.8612	0.8646	0.943	0.930	0.946	0.935	0.933	0.919
160	0.8510	0.8712	0.8642	0.948	0.931	0.943	0.932	0.949	0.919
180	0.8594	0.8736	0.8770	0.948	0.916	0.944	0.936	0.946	0.905
200	0.8640	0.8806	0.8866	0.944	0.923	0.946	0.929	0.943	0.928

Table 6.8: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where same 10 objects are used for the testing and training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

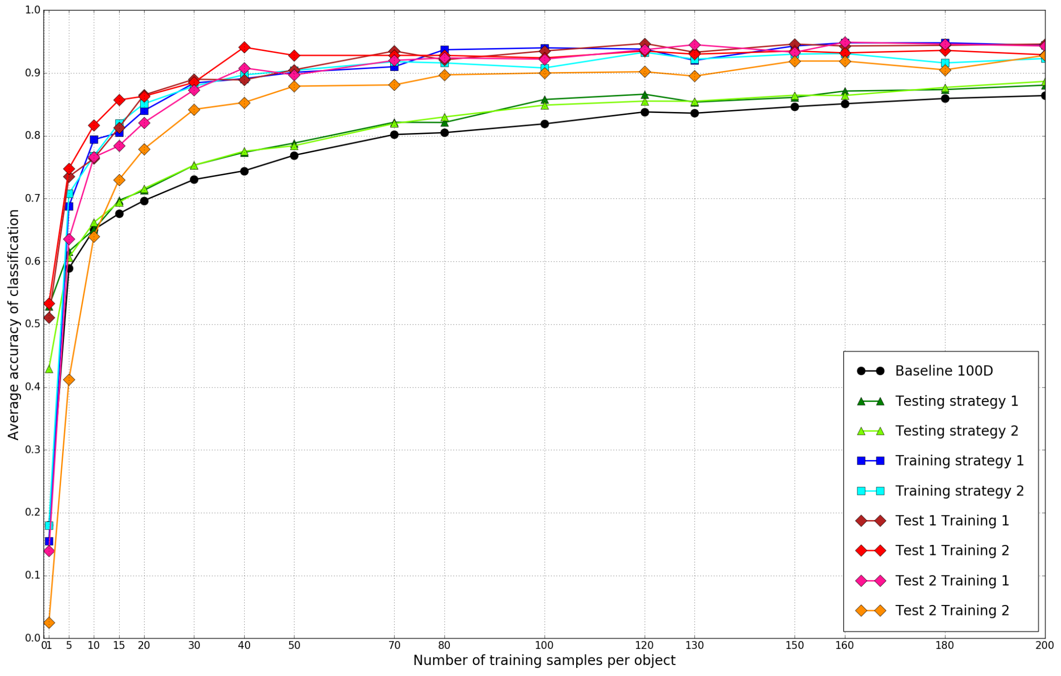

Figure 6.8: Classification accuracy of the proposed strategies in 100-dimensional feature space for the case where same 10 objects are used for the testing and training.

Setting 1 - 10D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0457	0.0817	0.0776	0.0433	0.0456	0.0808	0.0817	0.0616	0.0412
5	0.0965	0.1387	0.1352	0.1052	0.1065	0.1607	0.1686	0.1528	0.1508
10	0.1134	0.1657	0.1640	0.1313	0.1287	0.2197	0.2217	0.2129	0.2290
15	0.1239	0.1895	0.1889	0.1436	0.1378	0.2642	0.2646	0.2573	0.2756
20	0.1294	0.2075	0.2029	0.1557	0.1417	0.2946	0.2995	0.2908	0.3080
30	0.1413	0.2434	0.2468	0.1603	0.1472	0.3450	0.3365	0.3402	0.3592
40	0.1513	0.2679	0.2804	0.1713	0.1558	0.3778	0.3780	0.3647	0.4017
50	0.1608	0.2957	0.2990	0.1717	0.1629	0.4079	0.3938	0.3895	0.4338
70	0.1714	0.3488	0.3421	0.1780	0.1717	0.4473	0.4504	0.4310	0.4725
80	0.1768	0.3504	0.3510	0.1898	0.1769	0.4587	0.4586	0.4550	0.4842
100	0.1899	0.3648	0.3753	0.1939	0.1873	0.4706	0.4897	0.4767	0.5133
120	0.1887	0.3943	0.3917	0.2087	0.2003	0.5003	0.5036	0.5009	0.5319
130	0.1923	0.4120	0.3872	0.2140	0.1994	0.5029	0.5006	0.5013	0.5387
150	0.1968	0.4163	0.4155	0.2182	0.2139	0.5185	0.5233	0.5190	0.5485
160	0.1972	0.4202	0.4150	0.2371	0.2192	0.5272	0.5283	0.5244	0.5539
180	0.2013	0.4398	0.4320	0.2373	0.2213	0.5425	0.5345	0.5325	0.5655
200	0.2023	0.4444	0.4350	0.2429	0.2283	0.5467	0.5400	0.5454	0.5798

Table 6.9: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where all 126 objects are used for the testing and for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

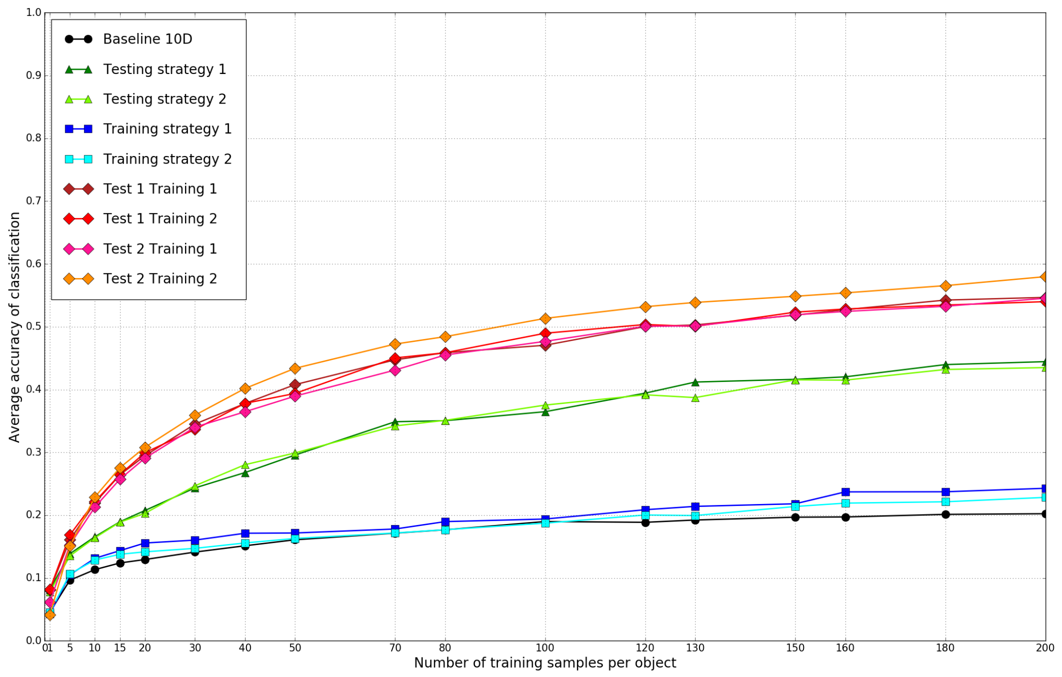

Figure 6.9: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where all 126 objects are used for the testing and for the training.

Setting 2 - 10D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0525	0.0742	0.0574	0.0461	0.0537	0.0763	0.0725	0.0553	0.0392
5	0.0977	0.1096	0.1135	0.1138	0.1071	0.1452	0.1344	0.1393	0.1393
10	0.1137	0.1416	0.1395	0.1414	0.1322	0.1846	0.1851	0.1939	0.1976
15	0.1229	0.1673	0.1530	0.1553	0.1438	0.2223	0.2108	0.2182	0.2384
20	0.1309	0.1822	0.1825	0.1570	0.1517	0.2498	0.2444	0.2510	0.2616
30	0.1432	0.2059	0.2048	0.1688	0.1574	0.2888	0.2782	0.2910	0.3079
40	0.1505	0.2360	0.2267	0.1747	0.1686	0.3028	0.3065	0.3087	0.3295
50	0.1563	0.2644	0.2483	0.1766	0.1734	0.3281	0.3323	0.3301	0.3587
70	0.1721	0.2783	0.2847	0.1889	0.1846	0.3563	0.3598	0.3615	0.3870
80	0.1729	0.2926	0.2901	0.1950	0.1809	0.3667	0.3752	0.3685	0.4022
100	0.1793	0.3053	0.3026	0.2036	0.1938	0.3746	0.3867	0.3867	0.4147
120	0.1882	0.3337	0.3142	0.2070	0.2007	0.3906	0.4020	0.4043	0.4289
130	0.1862	0.3452	0.3402	0.2064	0.2112	0.4074	0.4132	0.4147	0.4413
150	0.1941	0.3490	0.3439	0.2213	0.2120	0.4140	0.4173	0.4232	0.4447
160	0.1976	0.3433	0.3565	0.2306	0.2232	0.4124	0.4217	0.4291	0.4442
180	0.2006	0.3613	0.3620	0.2288	0.2282	0.4210	0.4255	0.4385	0.4552
200	0.2015	0.3686	0.3730	0.2398	0.2378	0.4306	0.4409	0.4387	0.4642

Table 6.10: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

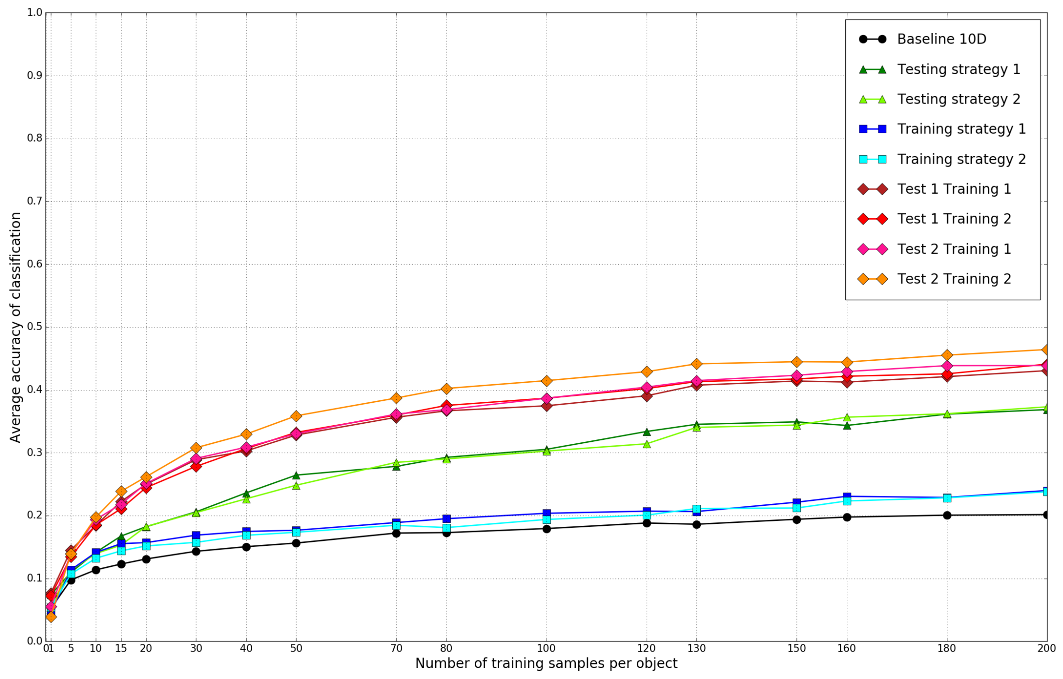

Figure 6.10: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training.

Setting 3 - 10D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1124	0.2260	0.1226	0.0960	0.1050	0.2267	0.2167	0.0269	0.0173
5	0.2598	0.2775	0.2593	0.2932	0.2980	0.3133	0.3093	0.2664	0.1468
10	0.2858	0.2963	0.2796	0.3438	0.3396	0.3480	0.3427	0.3378	0.2950
15	0.3004	0.3155	0.3080	0.3582	0.3684	0.3780	0.3747	0.3755	0.3347
20	0.3152	0.3280	0.3299	0.3772	0.3704	0.3900	0.3940	0.3996	0.3716
30	0.3244	0.3483	0.3480	0.3946	0.3862	0.4233	0.4167	0.4310	0.4043
40	0.3422	0.3608	0.3607	0.4026	0.4020	0.4293	0.4240	0.4207	0.4193
50	0.3340	0.3760	0.3711	0.4202	0.4152	0.4353	0.4327	0.4360	0.4367
70	0.3658	0.3880	0.3857	0.4248	0.4160	0.4560	0.4487	0.4433	0.4432
80	0.3726	0.3989	0.4005	0.4194	0.4216	0.4473	0.4567	0.4513	0.4430
100	0.3720	0.3989	0.4087	0.4340	0.4326	0.4600	0.4533	0.4593	0.4525
120	0.3848	0.4112	0.4169	0.4302	0.4388	0.4593	0.4680	0.4700	0.4697
130	0.3904	0.4167	0.4148	0.4442	0.4376	0.4633	0.4580	0.4653	0.4727
150	0.4008	0.4231	0.4166	0.4490	0.4352	0.4807	0.4693	0.4727	0.4860
160	0.4028	0.4264	0.4251	0.4510	0.4450	0.4627	0.4720	0.4693	0.4982
180	0.3956	0.4281	0.4355	0.4502	0.4448	0.4740	0.4633	0.4780	0.5035
200	0.4144	0.4403	0.4409	0.4566	0.4540	0.4713	0.4680	0.4720	0.4894

Table 6.11: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

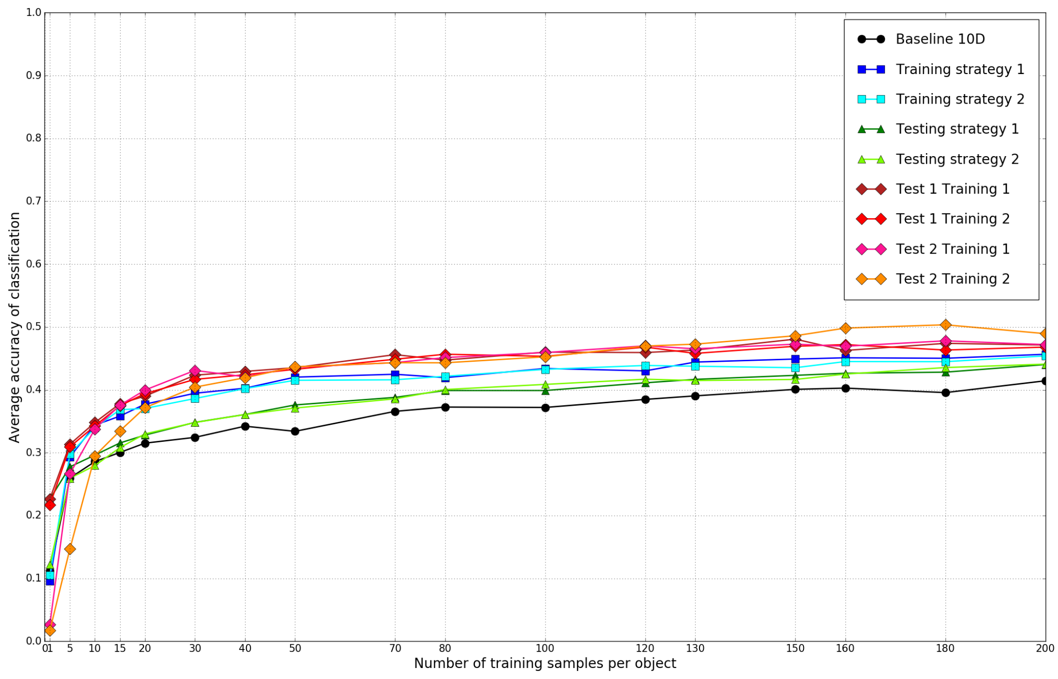

Figure 6.11: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training.

Setting 4 - 10D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1614	0.3318	0.1828	0.1440	0.1576	0.341	0.330	0.027	0.025
5	0.3888	0.3974	0.3806	0.4562	0.4478	0.500	0.461	0.310	0.187
10	0.4254	0.4386	0.4214	0.5108	0.5146	0.522	0.523	0.448	0.380
15	0.4464	0.4580	0.4616	0.5502	0.5382	0.553	0.560	0.532	0.482
20	0.4702	0.4852	0.5006	0.5796	0.5640	0.594	0.618	0.550	0.548
30	0.4942	0.5182	0.5224	0.5980	0.5926	0.626	0.637	0.624	0.626
40	0.5112	0.5374	0.5356	0.6052	0.6090	0.617	0.642	0.656	0.632
50	0.5230	0.5544	0.5474	0.6386	0.6236	0.671	0.654	0.656	0.646
70	0.5368	0.5810	0.5794	0.6364	0.6398	0.687	0.676	0.673	0.662
80	0.5494	0.5798	0.5936	0.6428	0.6526	0.667	0.685	0.663	0.666
100	0.5838	0.6132	0.6000	0.6544	0.6618	0.658	0.697	0.670	0.698
120	0.5866	0.6152	0.6144	0.6558	0.6692	0.701	0.720	0.716	0.711
130	0.6020	0.6468	0.6216	0.6806	0.6644	0.672	0.728	0.709	0.730
150	0.6004	0.6356	0.6284	0.6818	0.6714	0.697	0.714	0.707	0.701
160	0.6116	0.6386	0.6302	0.6844	0.6754	0.712	0.718	0.700	0.712
180	0.6096	0.6444	0.6414	0.6924	0.6806	0.725	0.729	0.707	0.708
200	0.6128	0.6454	0.6518	0.7042	0.6994	0.713	0.724	0.715	0.734

Table 6.12: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where same 10 objects are used for the testing and training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

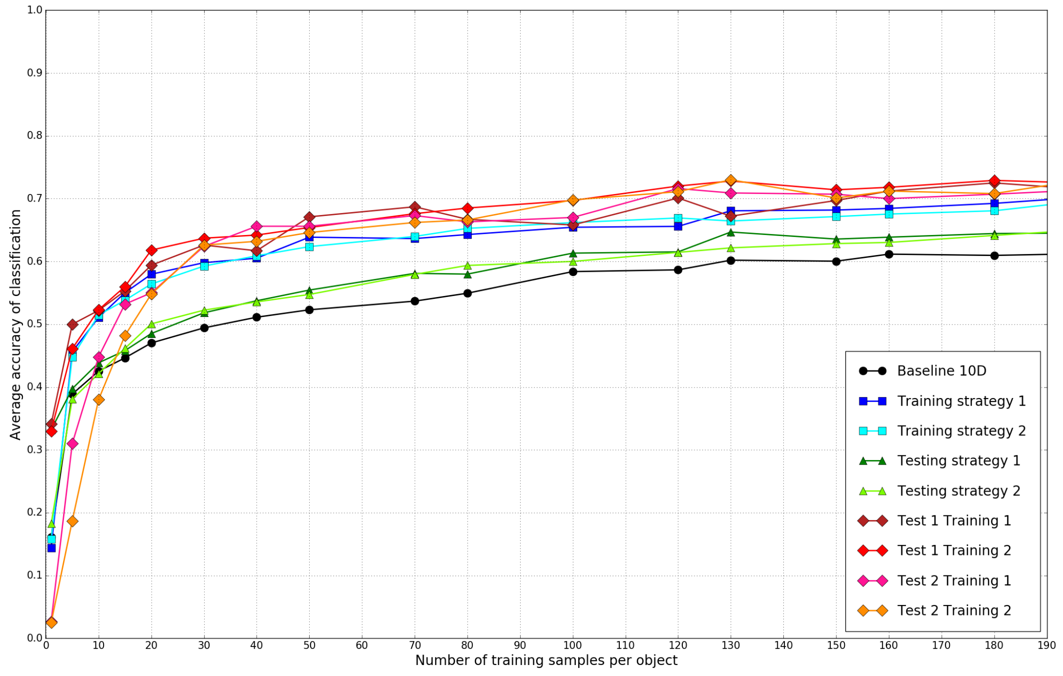

Figure 6.12: Classification accuracy of the proposed strategies in 10-dimensional feature space for the case where same 10 objects are used for the testing and training.

Setting 1 - 2D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0214	0.0227	0.0173	0.0180	0.0186	0.0233	0.0210	0.0082	0.0050
5	0.0253	0.0307	0.0257	0.0218	0.0206	0.0393	0.0384	0.0316	0.0275
10	0.0272	0.0390	0.0312	0.0232	0.0237	0.0478	0.0460	0.0448	0.0418
15	0.0280	0.0416	0.0371	0.0248	0.0252	0.0564	0.0529	0.0493	0.0499
20	0.0283	0.0452	0.0426	0.0271	0.0243	0.0584	0.0556	0.0561	0.0495
30	0.0275	0.0452	0.0471	0.0248	0.0240	0.0624	0.0607	0.0608	0.0614
40	0.0294	0.0537	0.0487	0.0240	0.0240	0.0640	0.0671	0.0629	0.0649
50	0.0304	0.0521	0.0533	0.0258	0.0266	0.0680	0.0649	0.0674	0.0650
70	0.0309	0.0565	0.0552	0.0284	0.0270	0.0753	0.0712	0.0702	0.0675
80	0.0302	0.0593	0.0576	0.0303	0.0256	0.0718	0.0694	0.0699	0.0698
100	0.0298	0.0610	0.0584	0.0256	0.0265	0.0688	0.0716	0.0688	0.0707
120	0.0301	0.0639	0.0634	0.0268	0.0269	0.0726	0.0762	0.0752	0.0755
130	0.0312	0.0581	0.0649	0.0277	0.0240	0.0758	0.0743	0.0764	0.0716
150	0.0317	0.0651	0.0625	0.0283	0.0265	0.0710	0.0733	0.0745	0.0740
160	0.0318	0.0683	0.0657	0.0272	0.0237	0.0743	0.0760	0.0750	0.0791
180	0.0315	0.0626	0.0617	0.0279	0.0260	0.0784	0.0765	0.0743	0.0748
200	0.0317	0.0652	0.0701	0.0270	0.0248	0.0736	0.0743	0.0769	0.0743

Table 6.13: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where all 126 objects are used for the testing and for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

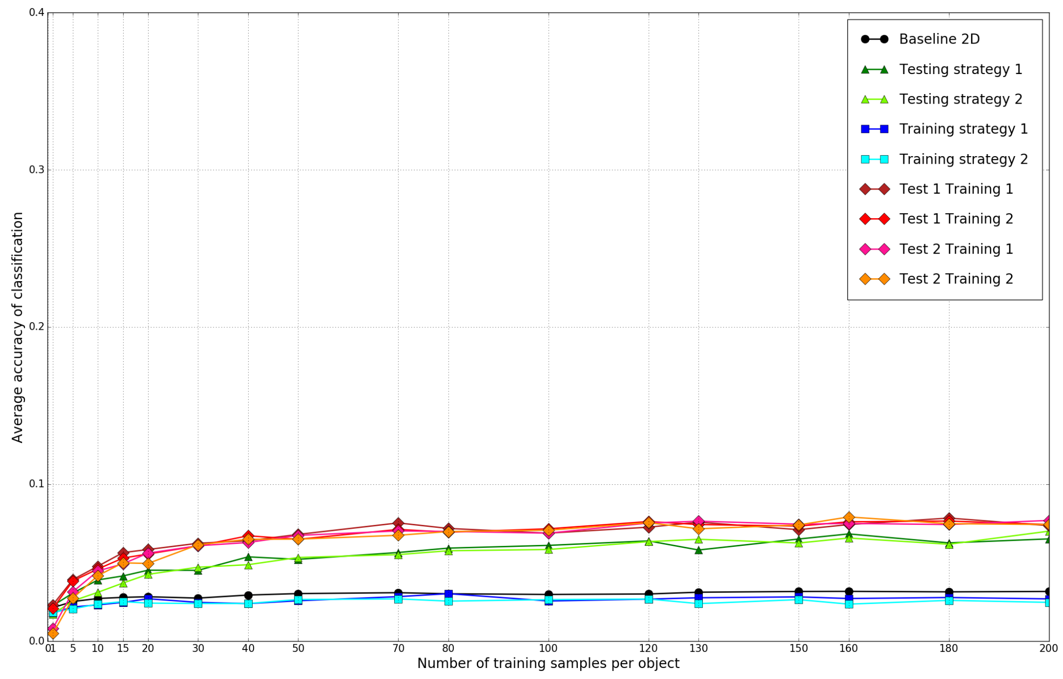

Figure 6.13: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where all 126 objects are used for the testing and for the training.

Setting 2 - 2D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0147	0.0223	0.0125	0.0206	0.0198	0.0219	0.0207	0.0094	0.0059
5	0.0154	0.0304	0.0242	0.0255	0.0282	0.0370	0.0360	0.0303	0.0296
10	0.0156	0.0359	0.0316	0.0296	0.0288	0.0478	0.0460	0.0412	0.0421
15	0.0170	0.0355	0.0356	0.0300	0.0303	0.0526	0.0507	0.0463	0.0485
20	0.0165	0.0417	0.0363	0.0301	0.0272	0.0538	0.0545	0.0521	0.0514
30	0.0168	0.0456	0.0443	0.0311	0.0278	0.0610	0.0586	0.0593	0.0570
40	0.0163	0.0475	0.0462	0.0308	0.0296	0.0625	0.0602	0.0619	0.0612
50	0.0169	0.0521	0.0495	0.0265	0.0295	0.0663	0.0663	0.0639	0.0631
70	0.0184	0.0530	0.0492	0.0323	0.0285	0.0667	0.0706	0.0662	0.0634
80	0.0164	0.0555	0.0529	0.0308	0.0298	0.0671	0.0668	0.0695	0.0627
100	0.0166	0.0603	0.0558	0.0293	0.0308	0.0729	0.0724	0.0699	0.0681
120	0.0165	0.0589	0.0575	0.0350	0.0273	0.0710	0.0687	0.0710	0.0703
130	0.0165	0.0590	0.0561	0.0318	0.0296	0.0759	0.0738	0.0738	0.0686
150	0.0174	0.0665	0.0619	0.0316	0.0287	0.0671	0.0690	0.0701	0.0676
160	0.0168	0.0632	0.0586	0.0315	0.0313	0.0727	0.0723	0.0729	0.0712
180	0.0169	0.0614	0.0574	0.0313	0.0300	0.0720	0.0755	0.0759	0.0737
200	0.0169	0.0635	0.0619	0.0337	0.0306	0.0737	0.0763	0.0705	0.0728

Table 6.14: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

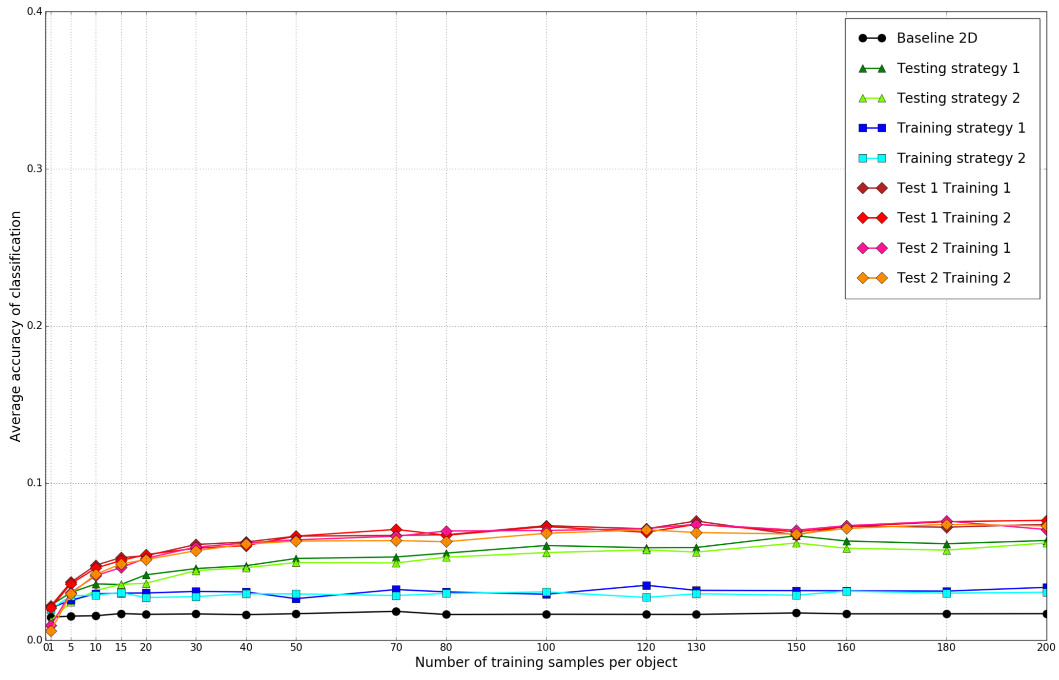

Figure 6.14: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where all 126 objects are used for the testing and random 100 for the training.

Setting 3 - 2D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.0916	0.1191	0.0483	0.0960	0.0896	0.1213	0.1160	0.0354	0.0298
5	0.1324	0.1348	0.0788	0.2932	0.1406	0.1360	0.1380	0.0854	0.0902
10	0.1336	0.1469	0.1042	0.3438	0.1558	0.1573	0.1440	0.1269	0.1315
15	0.1378	0.1501	0.1290	0.3582	0.1562	0.1580	0.1607	0.1598	0.1522
20	0.1464	0.1501	0.1389	0.3772	0.1590	0.1493	0.1713	0.1567	0.1755
30	0.1492	0.1640	0.1657	0.3946	0.1732	0.1500	0.1707	0.1681	0.1769
40	0.1488	0.1625	0.1628	0.4026	0.1658	0.1720	0.1800	0.1855	0.2013
50	0.1526	0.1647	0.1684	0.4202	0.1690	0.1873	0.1900	0.2000	0.2020
70	0.1582	0.1764	0.1712	0.4248	0.1746	0.1840	0.1880	0.1854	0.2100
80	0.1578	0.1763	0.1793	0.4194	0.1714	0.1920	0.1867	0.2021	0.2073
100	0.1662	0.1731	0.1764	0.4340	0.1724	0.1960	0.1840	0.2034	0.2100
120	0.1544	0.1843	0.1847	0.4302	0.1718	0.1967	0.1853	0.2073	0.2200
130	0.1576	0.1895	0.1872	0.4442	0.1728	0.1833	0.1907	0.2050	0.2047
150	0.1722	0.1917	0.1879	0.4490	0.1790	0.1787	0.1940	0.1933	0.1980
160	0.1706	0.1816	0.1848	0.4510	0.1820	0.1940	0.1840	0.1894	0.2053
180	0.1632	0.1923	0.1903	0.4502	0.1796	0.1807	0.1873	0.2040	0.2047
200	0.1656	0.1969	0.1879	0.4566	0.1734	0.1767	0.1960	0.1773	0.2007

Table 6.15: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

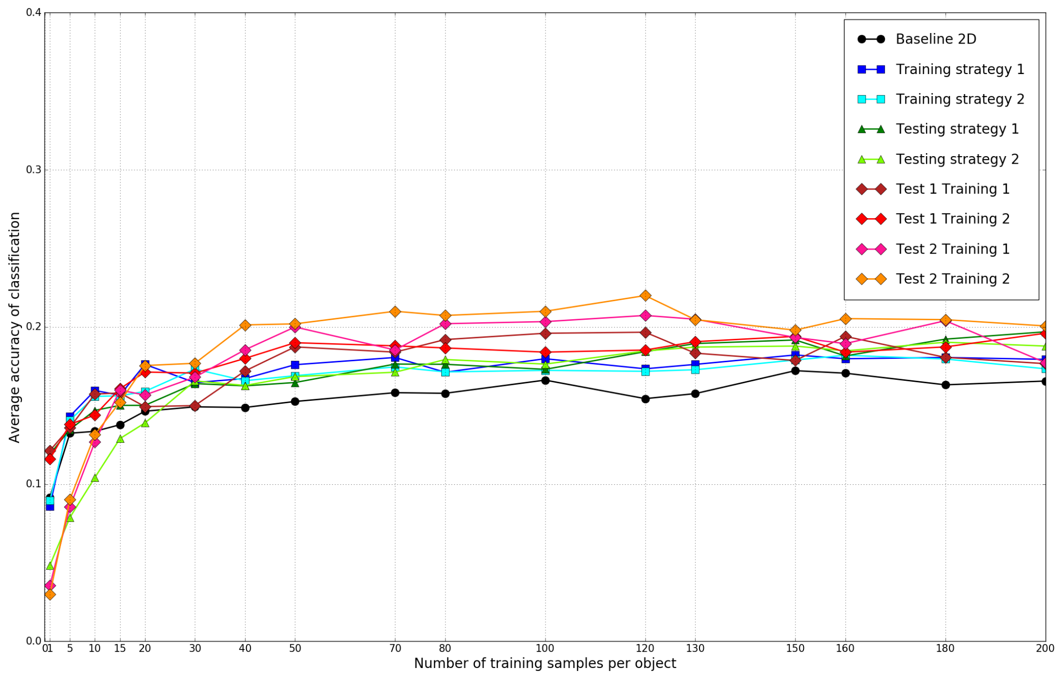

Figure 6.15: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where random 15 objects are used for the testing and 10 out of those 15 for the training.

Setting 4 - 2D

Nb	Baseline	Test 1	Test 2	Training 1	Training 2	Comb 1	Comb 2	Comb 3	Comb 4
1	0.1338	0.1792	0.0722	0.1254	0.1352	0.181	0.189	0.047	0.045
5	0.1980	0.2086	0.1236	0.2174	0.2158	0.241	0.210	0.128	0.134
10	0.1968	0.2202	0.1684	0.2280	0.2260	0.254	0.253	0.178	0.192
15	0.2140	0.2238	0.1856	0.2418	0.2376	0.260	0.269	0.197	0.245
20	0.2166	0.2302	0.2028	0.2408	0.2448	0.239	0.261	0.237	0.265
30	0.2244	0.2520	0.2312	0.2564	0.2526	0.267	0.266	0.258	0.248
40	0.2216	0.2446	0.2480	0.2540	0.2482	0.280	0.272	0.256	0.296
50	0.2252	0.2516	0.2598	0.2518	0.2560	0.258	0.273	0.254	0.300
70	0.2244	0.2614	0.2628	0.2480	0.2600	0.279	0.289	0.256	0.291
80	0.2410	0.2692	0.2694	0.2594	0.2548	0.287	0.298	0.252	0.294
100	0.2316	0.2790	0.2712	0.2672	0.2666	0.271	0.294	0.298	0.297
120	0.2444	0.2878	0.2776	0.2624	0.2630	0.296	0.283	0.291	0.310
130	0.2414	0.2872	0.2760	0.2718	0.2528	0.274	0.292	0.284	0.294
150	0.2484	0.2780	0.2824	0.2680	0.2600	0.285	0.287	0.293	0.301
160	0.2460	0.2924	0.2802	0.2710	0.2658	0.286	0.307	0.283	0.312
180	0.2490	0.2830	0.2908	0.2768	0.2686	0.290	0.294	0.279	0.328
200	0.2426	0.2862	0.2910	0.2678	0.2690	0.264	0.269	0.274	0.323

Table 6.16: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where same 10 objects are used for the testing and training. Comb 1 = Test 1 Training 1. Comb 2 = Test 1 Training 2. Comb 3 = Test 2 Training 1. Comb 4 = Test 2 Training 2.

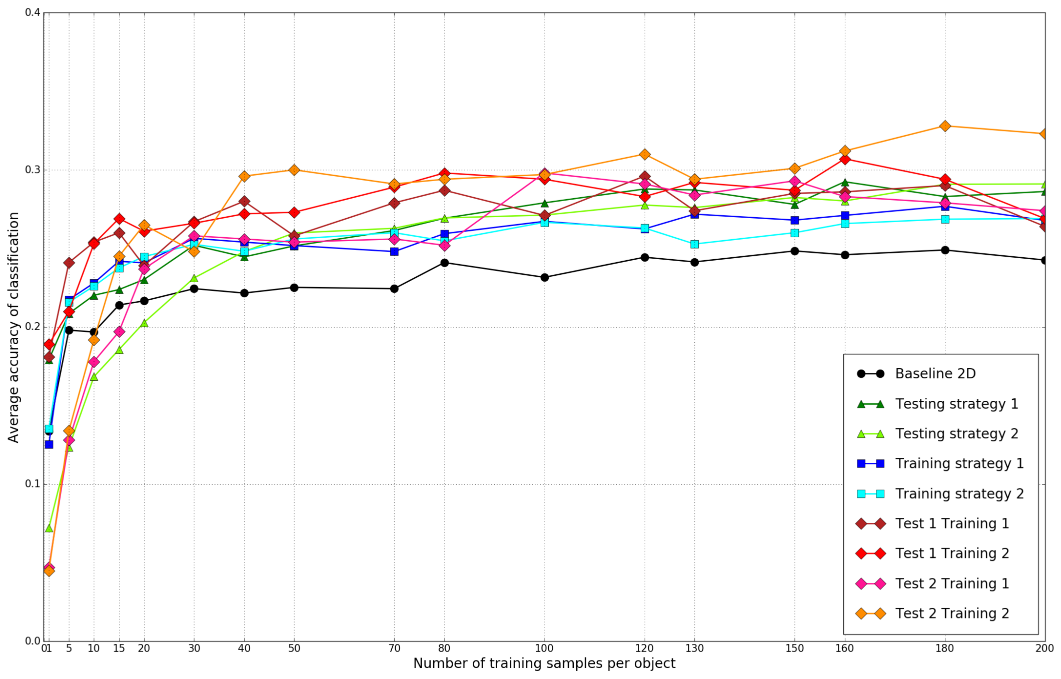

Figure 6.16: Classification accuracy of the proposed strategies in 2-dimensional feature space for the case where same 10 objects are used for the testing and training.

Glossary

kNN k-Nearest Neighbors. 18–20, 48, 49, 66

AUC Area Under the Receiver Operating Characteristic curve. 25

CL Convolutional Layer. 16–18

CNN Convolutional Neural Network. 15–18

DNN Deep Neural Network. 13, 15, 18

HRI-EU Honda Research Institute Europe. 3, 45

ML Machine Learning. 1, 10, 19

NN Neural Network. 10–13, 15, 17

ROC Receiver Operating Characteristic. 24–26

List of Figures

List of Tables

List of Algorithms