IKT411

Deep Learning on Smart Glasses

Authors:
Torry TUFTELAND
Guro ØDESNELTVEDT

Supervisor:
Ole-Christoffer GRANMO

Co-Supervisor:
Jaziar RADIANTI

The University of Agder

Grimstad, June 4, 2015

Abstract

Deep Learning is a sub-field of machine learning that involves learning through several levels of abstraction. It has been applied to fields such as computer vision, natural language processing and speech recognition, and has been shown to achieve state-of-the-art results in several of these fields. Smart glasses is another emerging and exciting field of technology. Smart glasses introduce capabilities to extend our perceived reality with information using advanced technology. This report is a summary of a project that uses computer vision algorithms based on deep learning and neural networks, and applies these algorithms to smart glasses. In this project we developed a proof-of-concept application for a set of smart glasses. The application recognizes objects through the camera of the smart glasses, and displays the results on the smart glass display. We also performed some experiments to benchmark the application and measure how it performs in a realistic environment. The experiments demonstrated that the application shows promise, but also have several areas where it may have room for improvement.

Keywords: Deep learning, computer vision, machine learning, neural networks, smart glasses, Android

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For the past few years, interest in Deep Learning have increased greatly. It has achieved state-of-the-art results within areas like computer vision [2], speech recognition[3], and natural language processing[4, 5]. In this project, we will specifically look at the possibilities for using advanced deep learning techniques for computer vision, and apply them to smart glasses. To achieve this, we developed a proof-of-concept application, and performed experiments relevant to the domain of smart glasses.

1.1 Project Background

This project is given by the Centre for Integrated Emergency Management (CIEM) [6]. It is a multidisciplinary research center which conducts research in fields like networks, mobile devices, human-centered sensing, social media, decision support and much more. CIEM was established in 2011 at the University of Agder, and is now a top research priority at the University. Their aim is to use technology to gather and distribute information in a crisis, which in turn can be used to manage the situation.

CIEM have recently acquired multiple sets of smart glasses for use in their lab. The motivation for this acquisition is to investigate the possibilities of using smart glasses in crisis situations. A use case may be to provide heads-up displays to firefighters that displays useful information about the environment and current situation. As the smart glasses has multiple sensors and a camera, these can be used to acquire information about the surroundings, and then send it back to the firefighters. Deep learning techniques have recently achieved state of the art performance on several image classification and object recognition tasks[7, 2], therefore it makes sense to investigate how these algorithms can be embedded into smart glasses and applied to crisis situations.

Smart glasses are wearable computers that can add information onto reality (Augmented Reality [8]). There are made many different models, with different designs and user interfaces. CIEM has the Epson Moverio BT-200.[9], so this is the model we are using in this project. It runs on Android, has multiple sensors and a front-facing camera. The BT-200 has a dual screen which gives a front facing display. Unlike for instance Google glasses, that only has a display on one eye.

1.2 Project Goal

The goal of this project is to create a proof-of-concept application for the BT-200 smart glasses that demonstrates how deep learning algorithms for computer vision can be applied to smart glasses. The app should take pictures with the camera mounted on the glasses, classify these images, and display the classification results on the smart glass display in real-time.

We will also investigate how various physical surroundings and lighting, may affect the classification performance with input from the smart glasses. This can give an impression on how a dataset from the smart glasses performs on the current state-of-the-art deep learning algorithms. Also, extensive testing identifies problems early in the project, which is useful for future work.

1.3 State-of-the-art

The motivation for this project is strongly related to the recent breakthroughs within deep learning for computer vision. In 2010, Ciresan et al.[10] broke the current state-of-the-art performance on the MNIST (Mixed National Institute of Standards and Technology database) handwritten digits dataset with GPU-implementation of a standard backpropagating neural network(see 2.1). However, the image resolution of the images in MNIST are relatively small(28×28), and many challenges arise when trying to apply such algorithms to larger images. But deep learning has proven very successful on larger images as well.

ImageNet is a popular dataset consisting of large images(256×256). Since 2010, the 'ImageNet Large Scale Visual Recognition Challenge'(ILSVRC) has been held annually[11]. In 2012, Krizhevsky et al.[2] used a deep neural network that introduced several new techniques and ideas. This neural network broke the previous record in the ILSVRC by far, and since then DL's have dominated the ILSVRC competition[1]. The current best result on the dataset from the ILSVRC competition is held by the Chinese search giant Baidu with a top-5 error rate of 4.58%. Top-5 error rate is the rate in which the denoted label of the image is not within the top-5 predicted results from the algorithm.

1.4 Contributions

From this project, we have created a novel prototype application for smart glasses based on object recognition. We have also built a dataset for evaluating object recognition capability in various physical and lighting settings. The dataset and source code is available on request.

[1]http://image-net.org/challenges/LSVRC/2014/results

1.5 Report Outline

The outline of this report is as follows: We start with a brief introduction to the theory of deep learning and neural networks in chapter 2. In chapter 3 we describe our proof-of-concept application, while in chapter 4, we present our experiments and experiment results. Lastly, we sum up our findings in chapter 6 and also discuss how the application can be further developed in the future in chapter 6.2.

Chapter 2

Theoretical Background

Deep Learning is a wide term used to describe a set of algorithms that uses multi-tiered architectures to model a learning problem. In order to understand deep learning, it is useful to understand neural networks. Many deep learning models today involve some kind of artificial neural networks, especially within computer vision. Deep learning in neural networks are simply neural networks with many ”hidden” layers, that is, layers that are not input or output layers. Since neural networks are a very wide and advanced topic, this chapter only provides a shallow summary of some relevant concepts. For a more in depth understanding, there are many books and resources available on the topic. This chapter is largely based on the book by Michael Nielsen, *’Neural Networks and Deep Learning’*[1].

2.1 Neural networks

The basic idea of a neural network is to represent the input as a vector, and then propagate this vector through a deep network of nodes, or neurons, in order to obtain a result vector. The input vector can for instance be pixel intensity values for an image, where each pixel is given a value in the vector. The size of the output vector will be equal to the number of possible classes or categories that the network are trained to learn. For a network learned to recognize handwritten digits, the output vector would consist of 10 neurons with values from 0 to 1 where the sum these of values are equal to 1. If the 4th value in the vector have the value 0.6, then we can interpret that the network believes there is a 0.6 certainty that the input image is the digit 4, assuming that the digits and output classes are mapped to their corresponding values.

For neural networks that takes images as input, the size of the input vector will be $m \times n \times r$, where m and n is the width and the height of the image, and r is the number of channels. For instance, if a network can take RGB images of size 28×28 as input, the input size would be $28 \times 28 \times 3$, with 3 being the red, green and blue channel.

Neural networks are trained through a set of training data. This is a set of labeled data samples, where the labels indicate the correct class for each given training sample. The samples are propagated through the network, and the results are compared to the label of the given sample. Small adjustments are then made in the network parameters to make the network give better results to similar training samples in the future. This will be explained further in section 2.1.2.

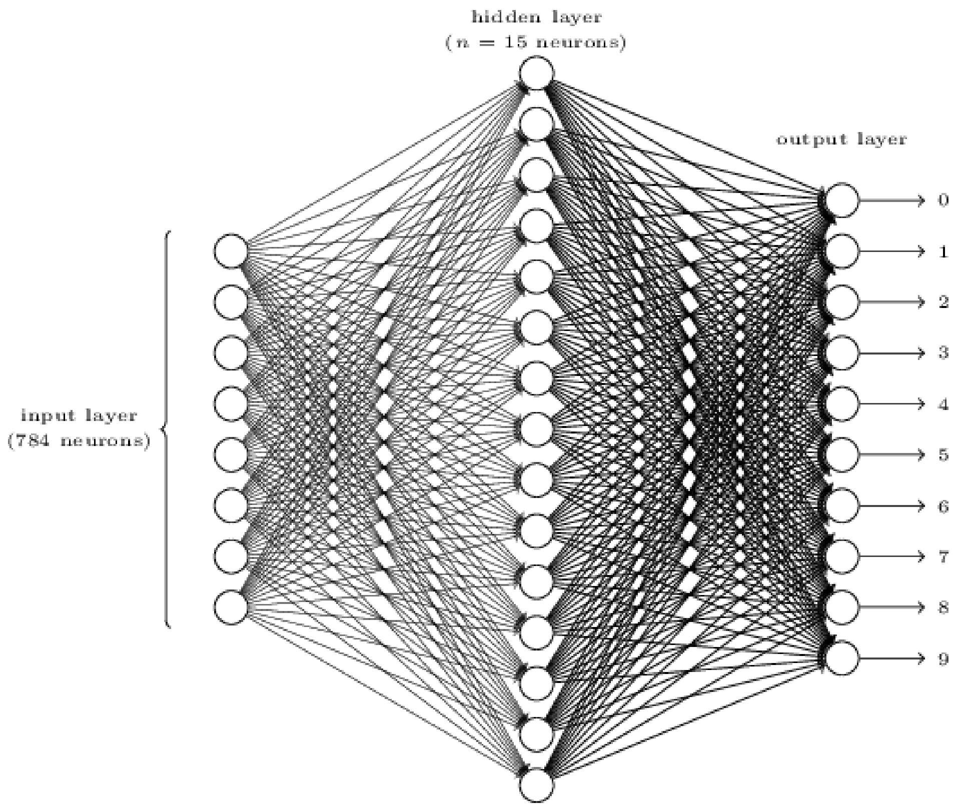


Figure 2.1: Neural network with one hidden layer[1]

2.1.1 The Activation Function

The output of neurons travel through edges, represented as lines in figure 2.1. Each edge has a weight w_i, which is an adjustable parameter indicating the importance of the edge. Each neuron also has a bias value b, which is another adjustable parameter. The output of the neuron is determined by the inputs x_i, the weights w_i, and the bias values b through an activation function σ. The activation function can be any function that outputs a value between 0 and 1, but the most common activation function is the sigmoid function, which is denoted as follows:

$$\sigma = \frac{1}{1 + exp(-\Sigma w_i x_i - b)} \tag{2.1}$$

At first, this expression may look complicated, but all it really does is to output a smooth function between 0 and 1, in this way normalizing the output. If we substitute $(-\Sigma w_i x_i - b)$ with z, we get the following plot:

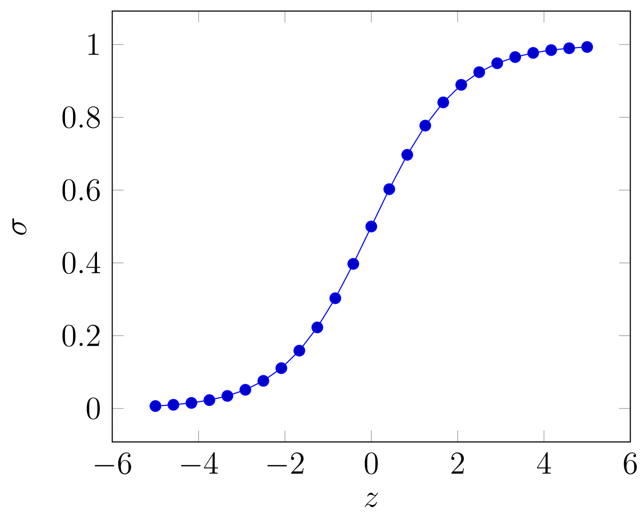


Figure 2.2: The Sigmoid function

The i'th input to a given neuron in layer j is equal to the output x from neuron i in layer $(j-1)$. This way, the values are propagated through the network using the sigmoid function with the network input, the weights and biases as parameters. Eventually, it will propagate up to the final layer where the computed output is the final result.

There are many alternatives to the sigmoid function, with pros and cons for every of them. Rectified linear units(ReLu) is an activation function that has gained a lot of traction within the machine learning community for the last few years. It is very simple, which makes it a faster and more efficient function than the sigmoid function[12]:

$$ReLu(x) = max(0, x) \tag{2.2}$$

2.1.2 Stochastic Gradient Descent (SGD)

Up until now, we have only talked about how a sample is classified by a neural network by propagating the input forward through the network. But how does the network learn these weights and biases in order to output realistic approximations? A standard approach to this problem is using the stochastic gradient descent algorithm.

A basic intuition for stochastic gradient descent is that it slightly modifies the network weights and biases on each training input in order to obtain better results for future inputs. To measure how well the weights and biases perform on the training output, we define a cost function $C(w, b)$:

$$C(w, b) = \frac{1}{2n}\Sigma||y(x) - a||^2 \tag{2.3}$$

In the cost function, w denotes all the weights in the network, while b denotes the biases. The notation with double vertical bars $||$ indicates the length of a vector. To sum up, the function returns the squared average distance from all the training examples a to their actual values $y(x)$

where n represents the number of training examples. The goal of SGD is to obtain a set of weights and biases such that the cost function returns a small value as possible, in other words, to minimize this cost function.

With two variables, the cost function can be depicted as a parabolic function where the goal is to find the value of x such that we find a global minimum of the function, this minimizing the cost.

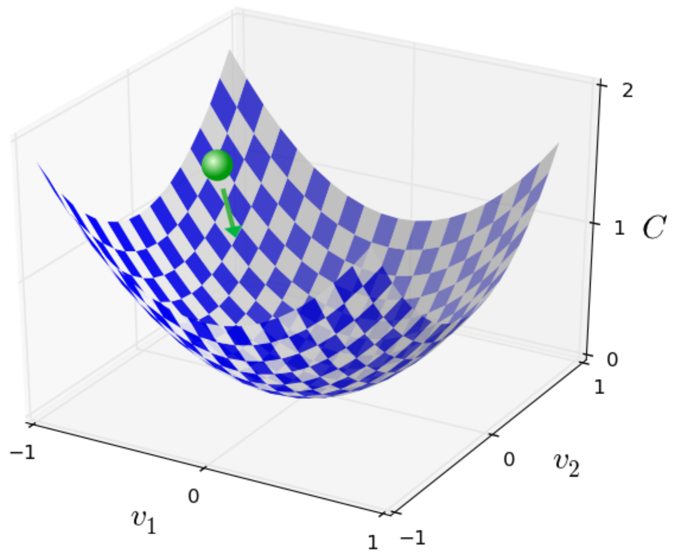


Figure 2.3: Minimizing the cost function[1]

As figure 2.3 illustrates, the goal of gradient descent is to move the green ball down the slope in order to find the global minimum of the cost function. The green ball is the starting point, which represents the random initialized weights and biases. In multidimensional space, gradient descent is not as easy to visualize, but the concept remains the same. The cost function is a multidimensional function in respects to the multivariate parameters w and b. To approach a global minimum for this function, we compute a gradient vector ∇C as follows(weights and biases are denoted as v_i):

$$\nabla C = \left(\frac{\delta C}{\delta v_1}, ..., \frac{\delta C}{\delta v_i}\right)^T \tag{2.4}$$

The gradient vector are the partial derivatives of the computed cost in respect to each of the variables v_i in the cost function. T is the transpose operation in order to turn the row vectors into column vectors. To compute the new weights and biases we multiply ∇C with a learning rate η, and subtract this from the old weights and biases, here denoted by w_k and b_l respectively:

$$w_k \rightarrow w'_k = w_k - \eta\frac{\delta C}{\delta w_k} \tag{2.5}$$

$$b_l \rightarrow b'_l = b_l - \eta \frac{\delta C}{\delta b_l} \tag{2.6}$$

The learning rate η is a value which indicates how fast the algorithm moves down the slope of the cost function. A learning rate too large may cause the cost function to never reach a global minimum. A value of η that is too low, and the computation time will increase, and the algorithm may use a very long time before it converges. If we think of the gradient vector as an arrow that points towards the global minimum, then the learning rate is a value that indicates how far we should travel in the direction of this arrow.

Gradient descent is quite computationally expensive, so there are a number of measures in order to make it more efficient. This where the stochastic part of the algorithm comes in. By estimating the gradient with only a small sample of randomly chosen training inputs, a so called mini-batch, one can quickly achieve a good estimate of the gradient, and thus speed up learning drastically.

2.1.3 Convolutional Neural Networks

Convolutional neural networks(CNN's) is a neural network variant that have proven to be very useful in a wide range of topics, especially computer vision. It is biologically inspired by simulating the cell arrangement of the visual cortex. The intuition behind it is that groups of neurons in the network only look at small overlapping portions of the image, so-called receptive fields. This way, the algorithm can tolerate translation and rotation of the input[13].

Chapter 3

Proposed Solution

This chapter describes the development of the proof-of-concept app, and explains the various challenges and decisions in regards to the app.

As mentioned in section 1.2, the goal was to develop a proof-of-concept app that takes pictures with the smart glass camera, classifies them, and sends the response to the user interface on the smart glasses. There are two main approaches to this problem: Either classify the images on the device itself, or set up a server and classify the images remotely. We chose the latter, and will discuss this decision in section 3.3.

Our solution consists of two main parts: The server application and the client-side application. The communication flow of these parts are illustrated in figure 3.1. We describe the two main parts in two different sections(section 3.1 and 3.2 respectively), and explain the frameworks we have used in each respective section. In section 3.3 we will discuss some of the architectural issues.

3.1 Server-side application

The task of the server is to classify images, and return the result from the classification to the client as JSON. This functionality is exposed through a REST-API that can take in HTTP requests from any type of client. This makes it flexible and reusable for other applications as well, not just Android applications.

Under the hood, the server application depends on Caffe - a deep learning framework based on C[14]. It also has wrappers for Python, which makes it ideal for rapid prototyping and experimentation. We also considered other frameworks such as Theano. However, since Caffe was the only framework that had a clear defined way to store and share trained models, we eventually opted for this.

3.1.1 Setting up Caffe

Caffe is a very efficient framework that utilizes the power of the GPU in its calculations. It has an expressive and simple syntax and is at the same time very powerful and fast. It also has a large community of users, and a website where users can post network architectures and model files. The wide community of Caffe was a crucial reason to why we chose the framework in the first place. This way we did not have to create our own implementation of the networks from

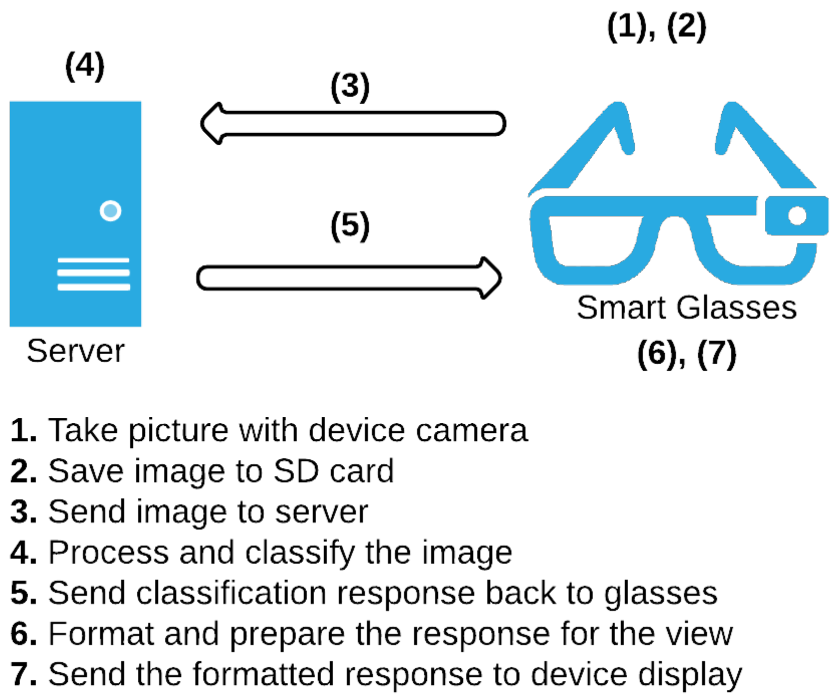

(1), (2)
(4)
(3)
(5)
Server
Smart Glasses
(6), (7)
1. Take picture with device camera
2. Save image to SD card
3. Send image to server
4. Process and classify the image
5. Send classification response back to glasses
6. Format and prepare the response for the view
7. Send the formatted response to device display

Figure 3.1: Solution network flow

scratch, but could rely on state-of-the-art networks trained by experts and scientists within the field. Models in Caffe are defined through standardized and easily readable file formats instead of code files, which makes model sharing easier. Also, the modularity of Caffe makes it easy to replace the underlying classification models without too much effort.

The highly abstracted syntax of Caffe comes at a cost - the ability to tweak and configure the networks. The layers in the model definition files relies on class files within the Caffe framework. This makes it hard to define custom layers or activation functions. But since this is primarily a proof-of-concept project, and not a deep dive into deep learning algorithms themselves, we traded off the ability to customize with the ease of use and simplicity of Caffe.

3.1.2 Algorithm description

Caffe has a predefined Classifier class which is initialized by providing the following arguments: the model definition file, the model itself(weights and biases) and the input dimension size. To classify a single image, we simply call the prediction method on the instantiated classifier class, with the image passed in as an argument.

The neural network we used in our experiments is the 22-layered GoogLeNet[7]. GoogLeNet won the classification task in the ILSVRC competition in 2014 where it achieved a 6.66% top-5 error rate[11]. The particular model in our project was trained by the Berkeley Vision and

Learning Center(BVLC), the team responsible for maintaining the Caffe framework. It was trained on the same dataset used in ILSVRC 2014 - a subset of ImageNet consisting of 1.2 million distributed over 1000 object categories. Even though this model is specifically trained to recognize objects, there is nothing in the way to replace it with a model that can recognize scenes and places[15], it all just depends on the dataset that it is trained on.

3.1.3 Image pre-processing

The GoogleNet network takes in images of size 256x256 pixels. Since the camera on the smart glasses captures images in 640x480 pixels, the images had to be preprocessed in some way in order to fit the network input. The Caffe classifier automatically resize images to fit to the network input. The resizing is done with interpolation and does not crop the image, but skewes it in order to fit it into a square shape. This transforms the objects in the image, but does not cut away anything from the image. The skew reduces the image height/width ratio from 4/3 to 1/1 and transforms the shapes and objects in the images to some degree.

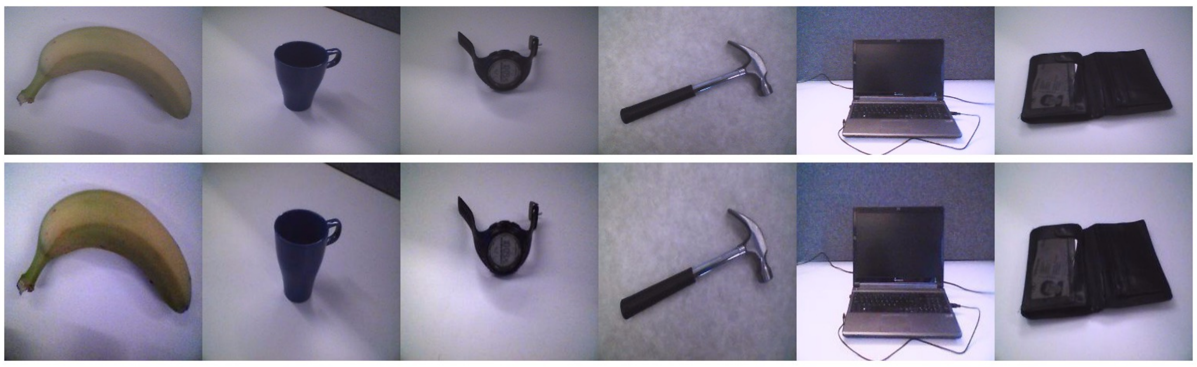


Figure 3.2: Sample images from smart glass camera. Top row is not transformed, while images in the bottom row is skewed to a square shape in order to fit to the network input

Since the skew did not transform the images dramatically, we decided to keep the skewing feature. Figure 3.2 shows some examples of skewed images captured with the camera on the smart glasses. A good alternative to avoid skewing the image could be to display a crop window on the app, and then center crop the image to a 1/1 ratio before resizing it.

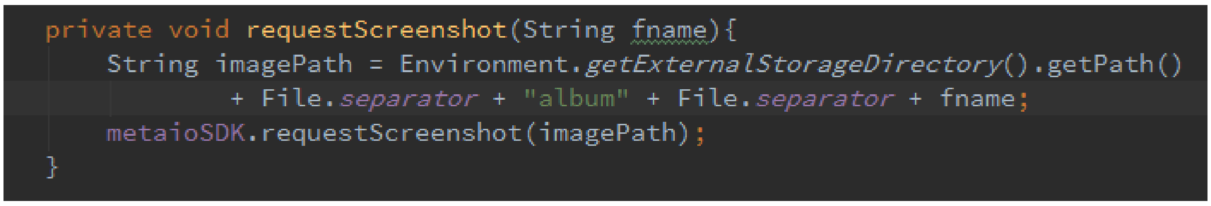
3.2 Client application

The goal of the proof-of-concept on the client-side is to develop an app that continuously takes pictures of the environment, sends these pictures to the server, and then displays the server response on the screen in real-time. The smart glasses run the Android operating system for smart phones and can in practice run any Android app. However, there are a few challenges related to developing Android applications on smart glasses. The screen of the BT-200 smart glasses are transparent, but in the Android operating system there are no obvious ways to implement see-through capabilities. Epson Moverio developer page [16] suggests two third-party

SDK's for development on the BT-200: Wikitude [17] and MetaioSDK[18]. Both frameworks have see-through capabilities, but MetaioSDK also provides some additional features including face detection and 3D object overlays[19]. These are features that may be useful in the future, therefore we found MetaioSDK to be the preferred framework for this project.

3.2.1 Creating the Application

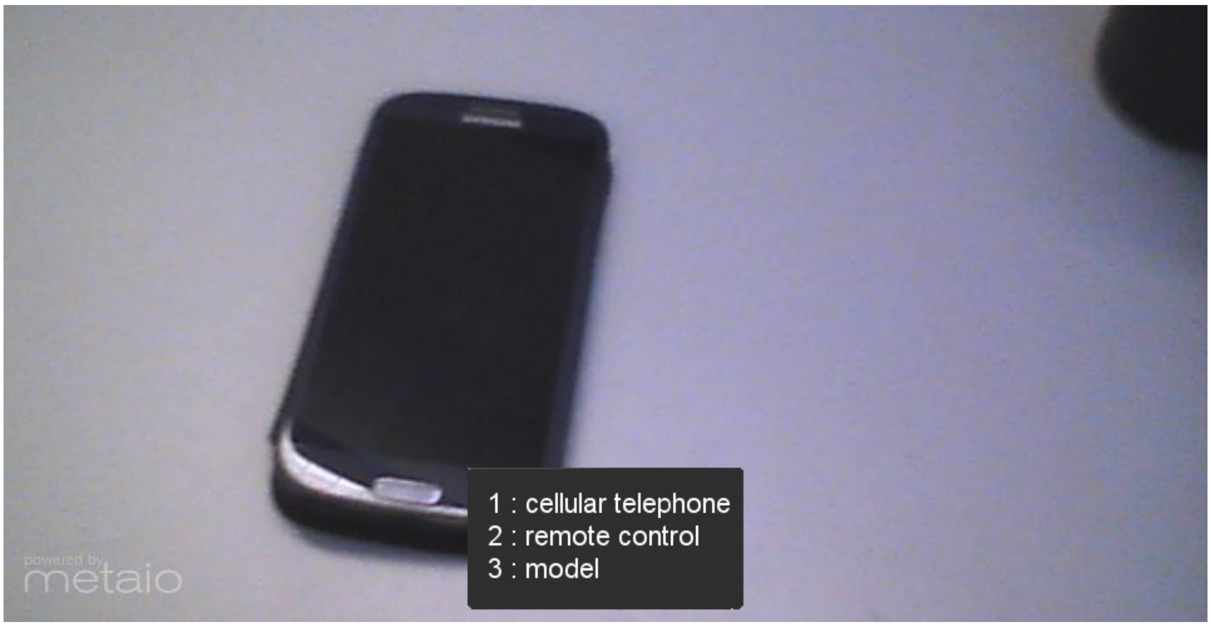
We had some issues making the application truly see-through, meaning, making the display transparent, and only draw UI elements on certain areas of the screen. Instead, we simulate see-through by displaying a live video-feed on the screen. This functionality is built-in within MetaioSDK, and drawing UI elements in top of the video feed was pretty straightforward. Pictures are taken every third second using the *metaioSDK.requestScreenshot()*, and then sent to the server API(see section 3.1) for classification.



```
private void requestScreenshot(String fname){
    String imagePath = Environment.getExternalStorageDirectory().getPath()
            + File.separator + "album" + File.separator + fname;
    metaioSDK.requestScreenshot(imagePath);
}
```

Figure 3.3: The method *screenshotrequest* from the application.

The JSON response from the server are then ordered and properly formatted for readability(see figure 3.4). The information displayed are the top three predicted categories, ordered from most the probable to the least probable object category.



```
1 : cellular telephone
2 : remote control
3 : model
```

Figure 3.4: Screenshot of the app running on the BT-200. The text window displays the result from the evaluation.

3.3 Discussion of Architecture Issues

In this chapter we discuss two important architecture decisions: The reason for why we chose a server-based implementation, and the use of frameworks on the client application.

3.3.1 Server-based- or embedded implementation

The goal of our app is to classify single images in real-time, and send responses back to the view continuously. The most important architecture consideration in regards to this was whether to embed the classification algorithm on the smart glasses, or to run the classification from a server. In deep learning, classifying single images on a pre-trained neural network model is a trivial operation compared to training the networks. Classifying a single image involves the following steps:

1. Resize the image to fit the neural network input dimensions.
2. Decompose the image into a matrix representation of values.
3. Unroll the matrix to a vector and use this as the first layer in the matrix.
4. Propagate the values through the pre-trained network(as described in section 2.1.1).

Embedded single-image classification is viable and also demonstrated on other projects such as in the open-source project DeepBeliefSDK[20]. However, an embedded implementation would not allow for the same flexibility as a server implementation. Caffe, the framework used on our server app, lets us easily replace models and networks through model files. To our knowledge, there are no corresponding frameworks available for Android devices. With our deep learning algorithms deployed on a server, this made it a lot easier to aggregate statistical data and conduct experiments. A server implementation also have more potential computing power, all depending on what server the app is deployed on. An embedded application on the other hand, is limited to the CPU power available on the device, but are not dependent on a reliable network connection.

Since a server application can be served from a REST API, one can connect multiple types of clients to the same API. There is no tight coupling between the type of client and the server app in a REST application. This makes it easier to create other types of clients applications later, such as an iOS app or web app.

The sum of arguments eventually led us to decide for the server version of the application. This way, we can more easily infer statistical data and conduct experiments, and there is still an opportunity to embed the algorithms on the client app itself at a later stage.

Server implementation	**Embedded implementation**
+ Ideal for experimentation and testing	+ Distributed classification
+ More computing power potential	+ Works when offline
+ Can serve multiple types of clients	- Harder to test and experiment with
- Single entry-point for classification (potential bottleneck)	- Limited computing power
- Network dependent	

Table 3.1: Server or embedded - pros and cons

3.3.2 Framework redundancy - MetaioSDK

The reason we chose to use MetaioSDK in the first place was because of its see-through capabilities [1]. But we did not manage to implement this in our app, so instead we simulated see-through by using a live video feed. This made MetaioSDK redundant, as this can be achieved through the Android operating system that runs on the smart glasses. However, we decided to keep the framework still, as its Augmented Reality features may be useful for further development.

[1]This feature is available in the Metaio 6.0.1 version and later.

Chapter 4

Testing

In this chapter we present the experiments conducted and the results. We specifically wanted to investigate how the classification algorithm performs on images taken with the smart glasses. The images taken with the BT-200 smart glasses are of relatively low quality with a resolution of only 640 by 480 pixels(0.3 mega pixels). We also found that it operates poorly when under suboptimal lighting conditions. The goal of these experiments are to assess the performance of the classification algorithm under various circumstances and conditions such as lighting, image noise and distance to objects.

In section 4.1 and 4.2 we explain how the tests were set up and an overview of the total results. In sections 4.3 through 4.7 we present and discuss each of the separate test cases.

4.1 Setting up the Experiment

To measure the performance of our experiments we constructed a base scene with good lighting and a neutral background. We selected a few objects and took pictures of them in this scene. Then we used the same objects and placed them in a number of other scenes, and compared the classification results in these scenes with the classification results in the base scene [1].

4.1.1 Selecting Objects and Scenes

There are mainly three metrics we investigated in our experiments: Distance to the object, scene lighting, and image noise. Each scene was a combination of these three metrics, where we had two settings for distance(far, close, cropped), three for lighting(bright indoor, daylight, dusk), two for image noise(noisy and clean), totaling in 18 different scenes.

The selection of objects in the experiments were varied, but still portable enough to move them between scenes. The objects we ended up with were the following:

- Banana
- Cup
- Digital watch
- Hammer
- Laptop
- Wallet

[1]See table A.3 for results

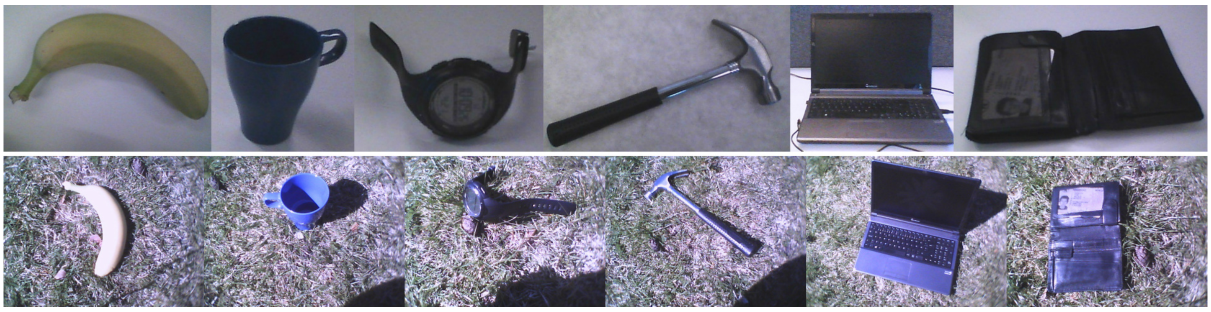


Figure 4.1: All objects in the base scene and in outdoor light with noisy background scene

The results from the main metrics experiments are presented in sections 4.3, 4.4 and 4.5. In addition to this, we performed classification tests on image quality and camera rotation in sections 4.6 and 4.7 respectively.

4.1.2 Measuring performance

When classifying an image, the classifier returns a list of all the objects that the algorithm is trained to recognize, ordered from the most to the least probable object for the specific image. The order is based on a probability score for each class where the sum of all the scores are equal to 1.

The most used evaluation metric used in computer vision competitions today are the top-5 and top-1 classification rates. Top-5 is the rate of classifications where the ground truth[2] is within the top 5 predicted results. Top-1 is the rate at which the ground truth is the top predicted result. E.g., if the algorithm is fed an image labeled as 'banana', and returns the result of 'apple', 'car', 'pencil', 'banana' and 'clock', ordered from most to least probable, then the result is counted as TRUE for the top-5 rate, and FALSE in the top-1 rate. Naturally, the total top-1 rate would in most cases be lower, and definitely not greater than, the total top-5 rate.

In addition to top-1 and top-5 rates, we also extracted statistical data such as mean, median and standard deviation based on the classification rankings. The ranks are values between 1 and 1000 for the 1000 possible classes. Generally, a rank not within the top 500 is statistically worse than guessing, while a rank within the top 5 is very good.

4.2 Overall results

The full result from the classifications are listed in table A.1 in the appendix. The values in the results denote the ranking of the ground truth within the list of the 1000 total categories. For instance, if an image of a 'cup' got a rank of 49, this means that the classification algorithm ranks 48 objects is more probably within the image compared to 'cup', and 951 objects that are less probable than cup.

[2]The ground truth is the actual class in which the image is labeled as. It is a term used in machine learning to denote the label of the training data.

Table 4.1 shows aggregated results for all data samples. The standard deviation is quite high, which indicates that there is much spread in the data samples; the index values range all from 1 to 717. The average value is quite high, but the median value is much lower. This means that there is a greater proportion of low valued samples and a few very high valued samples that contribute to the increased average value.

Average	**Median**	**Std. dev.**	**Top-1(%)**	**Top-5(%)**	**Top-100(%)**
113.49	17	181.56	13.89	37.04	71.30

Table 4.1: Statistics aggregated from all samples

The overall results from the classification on our dataset shows that the algorithm performs very well on images taken with the smart glasses. On all our images, it gave a top-1 rate of 13.89%. This may seem like a low value, but given that the algorithm is trained to recognize 1000 categories, then 13.89% is a 138 times better result than a random guess would give. Also, our dataset is built to test how different conditions affect the classification, so the scenes around the objects are constructed in order to make the classification challenging.

4.3 Distance

The goal of this experiment was to measure how the object distance from the camera would impact the classification result. We took pictures from two distances: close and far. In addition to this, we created a third category by cropping the close images to a bounding box around the objects themselves. The data from the cropped images are labeled 'Cropped' in table 4.2. The data is aggregated from classification of all images and grouped by the distance metric.

	Average	**Median**	**Std. dev.**	**Top-1**	**Top-5**	**Top-100**
Cropped	38.00	4.00	97.93	25.00	61.11	91.67
Close	67.81	10.50	143.89	16.67	41.67	83.33
Far	234.67	152.50	217.65	0.00	8.33	38.89

Table 4.2: Classification data grouped by distance

Overall, the 'cropped' images achieved far better results than the 'close' images with a 46% relative increase on the top-5 results, and with a 633% relative increase from the 'far' to the 'cropped' results[3]. This suggests that the application should definitely have some functionality to limit the bounds of objects before classification. This is further discussed in chapter 6.2.

[3]Calculated using relative change with absolute min function: $dr = \frac{|x-y|}{min(|x|,|y|)}$

4.4 Noise

Another test we did was background noise test. The goal for this test is to see how well the algorithm can recognize objects in a noisy environment. We define noise as something that might make it more difficult to separate objects from each other and/or a background.

	Average	**Median**	**Std. dev.**	**Top-1**	**Top-5**	**Top-100**
Clean	107.63	8.50	188.93	20.37	44.44	74.07
Noise	119.35	35.50	175.46	7.41	29.63	68.52

Table 4.3: Classification data grouped by noise

We started with the same setup as the control group: one picture close to the objects, and one picture with some distance to the objects. To apply noise in this environment we used other objects and placed them in the background. From the results we found that none of the objects passed the top-1 and only *cup* passed the top-5 when the pictures where taken up close. With distance none of the objects passed top-10. In the dusk scenes the noise also consisted of other objects. Here the *laptop* passes top-1 in both close and distance. In close up, *cup* and *wallet* passed top-5. When we did the daylight scenes we used grass instead of other objects as noise. Here *hammer* passed top-1 and the rest, except *banana* and *digital watch*, passed top-5. With distance, only the *laptop* passed the top-5.

From table 4.3 we find that there is a clear gap between the results of 'clean' and 'noise'. This was expected, since the noisy pictures also contains other objects. In e.g. the indoor scene with distance, the top results of the *banana* image were desktop computer, monitor and desk. If we look at the picture 4.2 (left) we see that there is a desk and a laptop there, with both of them dominating the picture a lot more that the banana. Another example, is from the dusk, noise, distance scene, where the object is *wallet*, but the top-1 result is folding chair. Looking at the picture (right part of 4.2), we can see a folding chair. This may indicate that the algorithm rates objects highly based on their size in the image. This is interesting when we compared to how humans finds objects in an image. This is discussed in the next section.

	Average	**Median**	**Std. dev.**	**Top-1**	**Top-5**	**Top-100**
Grass	229.92	140.00	250.38	0.00	8.33	41.67
Objects	125.17	50.50	158.38	4.17	25.00	66.67
White	38.25	7.00	47.58	33.33	41.67	83.33

Table 4.4: Classification data grouped by noise type. 'White' is the results from (indoor, clean) with close and far as distances. 'Grass' and 'Objects' consists of close and far images with grass and objects as backgrounds respectively.

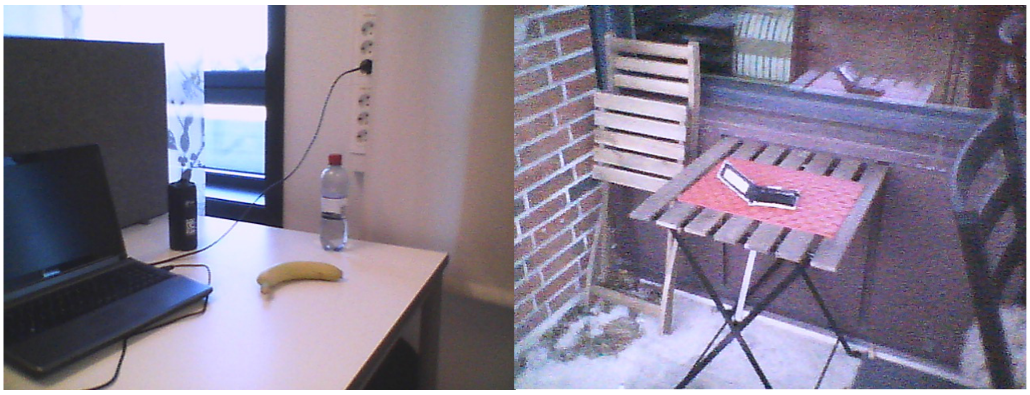


Figure 4.2: *Left*: Indoor, noise and distance picture. Object is banana. *Right*: dusk, noise, distance. Object is wallet.

4.4.1 Algorithm performance versus human performance

In order to measure how the algorithm performed against humans on images with many objects, we performed an experiment: A person were given three images with multiple objects. The subject were asked to identify the five most dominating objects in each image, and rank the objects from most dominating to least dominating. This is very similar to the task that the algorithm performs when classifying images.

We performed this test with three subjects. All subjects gave, independently of each other, very similar predictions, while the algorithms predictions greatly differed form these. All subjects pointed out 5 correct objects that actually was in the images, except for one case where a subject simply wrote 'thing'. The algorithm predicted many objects that was not in the images, but on the other hand managed to be more specific than the subjects in one case: Where 2 out of 3 subjects wrote chair as an object in the image, the algorithm wrote folding chair, which was more specific and also correct. The third subject agreed with the algorithm and wrote folding chair.

Another interesting case was with the 'banana' image(see left image of figure 4.2). All subjects wrote banana as their top prediction in this image, while the algorithm gave the banana a rank of 419. In the image, the banana is relatively small compared to the total image area, but is very easy to spot as it is placed in the center of the image and has a very bright and distinct color. These may be factors that the algorithm do not consider when classifying the image, while they appear to be important for the subjects in this test.

For further reading, the full results of this experiment can be found in the appendix B.

4.5 Light

As mentioned in previous sections, we tested the algorithm in different light settings; indoor, daylight and dusk. The goal is to test how different lighting conditions affects the quality of the pictures, and thus the classification.

	Average	Median	Std. dev.	Top-1	Top-5	Top-100
Indoor	54.89	5.50	100.70	22.22	50.00	83.33
Daylight	154.75	38.00	219.94	5.56	30.56	61.11
Dusk	130.83	15.00	191.49	13.89	30.56	69.44

Table 4.5: Result grouped by light

For the control group we used indoor light. This gave a clean and constant light that did not change over time, and affected the pictures as little as possible. By calculating the average of the different scenes by light (table 4.5) we see that indoor has the highest top-5 and top-1 values. Thus this light is the easiest of the three light types for our algorithm to classify objects in.

The daylight scenes are taken outside on a sunny day. This light is much brighter than the indoor light and is different because it only has one source of light(the sun). This resulted in distinct shadows and a lot of reflection on smooth surfaces, which may explain why daylight had slightly lower classification success rates than dusk light(see 4.5). Figure 4.3 shows an example of an image that was misclassified in outdoor light. The top-1 result of this picture was *goblet*, while the correct class, *cup*, was ranked 32. This particular image has a very distinct shadow, which may have been the cause of the misclassification. The algorithm has no way to tell the difference between an object and a shadow unless it has been specifically trained for this.

The images taken at dusk are darker and more grainy than the others. And in general, we found that the camera performed worse with just slightly reduced lighting. We also took some pictures at night time and tried to classify these, but the classifier were not able to classify any of these at all. When we inspected the images manually, we were not able to see anything but weak contours at best. Therefore, we did not include these results in the main results, as they would cause a lot of noise in the data. The results from the classification of the nighttime images can be found separately in table A.2.

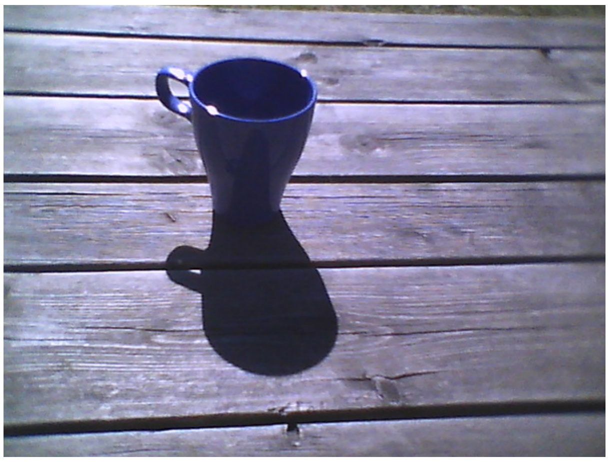


Figure 4.3: The cup, taken outside in sunlight.

4.6 Image Quality

Another important aspect to consider is the resolution and general image quality for the camera on the BT-200. To test this we compared some images taken with smart glasses with images taken with a smart phone camera. The camera resolution on the BT-200 is VGA(0.3 megapixels), while the smart phone has 8 megapixels. We took pictures of three different objects: hammer, water bottle and a backpack. They are all taken in the same room, in the same light setting and as close as possible. As table 4.6 shows, the classification results improved on all objects with the smart phone compared to the smart glasses. For the water bottle and hammer, the top-1 results are correct with both the smart glasses and smart phone, but the smart phone images have higher certainty for the correct class.

	Hammer		**Water Bottle**	
Rank	*Glasses*	*Smart Phone*	*Glasses*	*Smart Phone*
1	Hammer: 0.38	Hammer: 0.61	Water Bottle: 0.50	Water Bottle: 0.93
2	Ballpoint: 0.16	Ballpoint: 0.04	Nipple: 0.17	Pop Bottle: 0.03
3	Nail: 0.08	Pole: 0.02	Pop Bottle: 0.16	Plastic Bag: 0.01

	Backpack	
Rank	*Glasses*	*Smart Phone*
1	Ski mask: 0.32	Backpack: 0.80
2	Backpack: 0.08	Abaya: 0.04
3	Cloak: 0.03	Bulletproof vest: 0.04

Table 4.6: Results of hammer, water bottle and backpack taken on a clean background. Using glasses versus smart phone.

The classification of the backpack images was a lot better on the smart phone compared to the glasses. Backpack was ranked 2nd with a certainty of 0.08 on the smart glasses, while it was ranked 1st and with a certainty of 0.80 on the smart phone. This is a dramatic increase, and is a good example for how much the image quality can impact the classification. Although this may not hold true for all cases, our experiments suggests that an increase of camera quality can definitely improve the classification results.

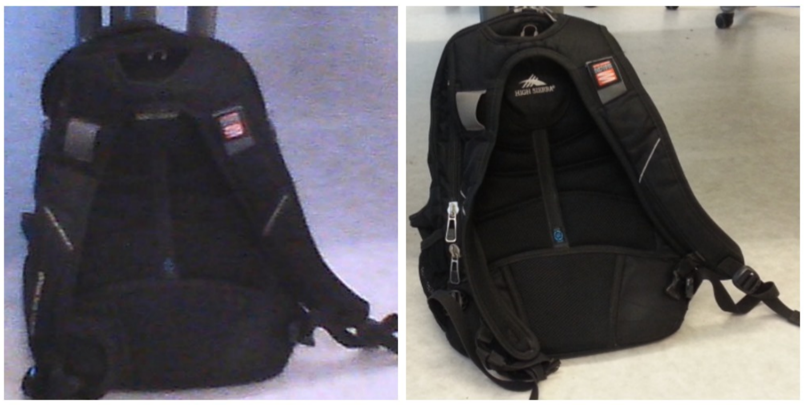


Figure 4.4: Camera quality - smart glasses(left) versus 8mp smart phone(right)

4.7 Orientation

The smart glasses takes pictures in landscape orientation, which results in more space on the sides if the object is vertically oriented. This extra space may contain more noise and make the classification process more difficult. With the smart phone we could flip the orientation to portrait instead, thereby removing potential noise. To examine to what degree camera orientation may impact classification accuracy, we took two new sets of pictures using the smart phone. One of the backpack with background noise and one of the water bottle with no background noises. This experiment is very similar to the experiment performed in chapter 4.3, but is different as it does not bound the objects completely.

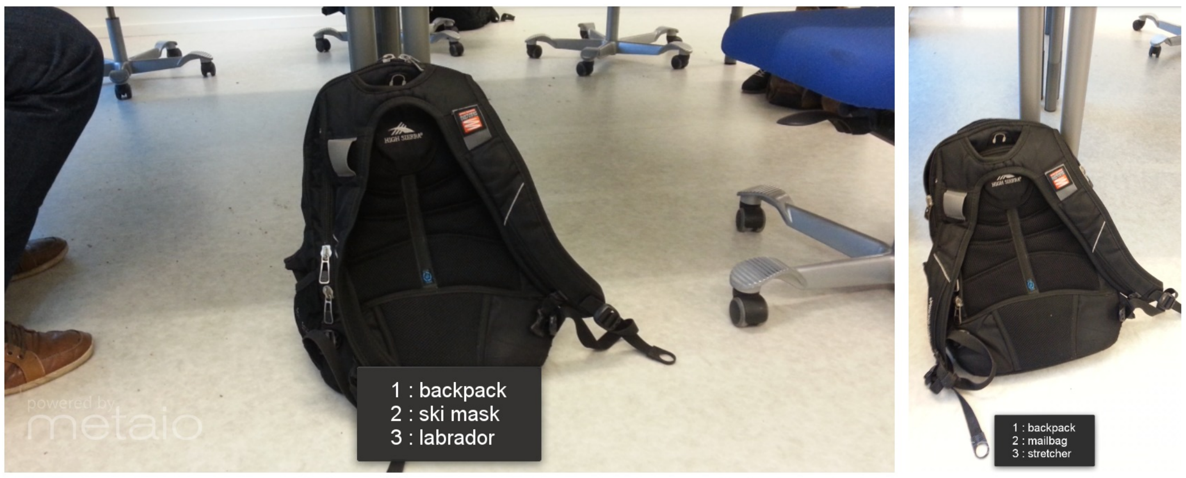


Figure 4.5: App screenshot of backpack. Landscape and portrait orientation from left to right respectively.

Based on these results we can see a tendency that by bounding the interesting object, and thereby removing potential noise, the results will for most cases improve. This is similar to the result we achieved in chapter 4.3.

Water Bottle		
Rank	*Portrait orientation*	*Landscape orientation*
1	Water Bottle: 0.99	Water Bottle: 0.93
2	Pop Bottle: 0.01	Pop Bottle: 0.02
3	Perfume: 0.0	Ballpoint: 0.01

Backpack		
Rank	*Portrait orientation*	*Landscape orientation*
1	Backpack: 0.98	Backpack: 0.19
2	Mailbag: 0.01	Ski mask: 0.10
3	Stretcher: 0.0	Labrador: 0.04

Table 4.7: Table comparing the result of a water bottle and backpack as an object in portrait and landscape orientation.

Chapter 5

Discussion

In this chapter we discuss what aspects of the application that could have been done differently, and further discuss our test methods and results.

5.1 Embed classifying algorithm on smart glasses

In chapter 3.3.1 we reasoned for why we chose to implement our classification algorithm on a server instead of embedding it on the device itself. The reasons for this were largely based on convenience; it is a lot easier to test and experiment through a server API than an Android application. However, for production use, embedding the classification algorithm on the device itself is definitely viable and should be considered for future development. Being dependent on an Internet connection and a running server, may prove problematic in a crisis situation.

5.2 Testing

Looking back at the experiments we conducted, we see that some parameters could have been better defined. If the set up had been consistent for all images, the results could have been deemed more trustworthy. On the other hand, since the scenes were not completely consequent, we were able to find other interesting trends, such as how a background of multiple objects compared to generally noisy background(see section 4.4) in the classification.

Since our dataset is relatively small, this may question the validity of our results. Therefore one should be careful of concluding on absolutes when using this dataset, but it may be very useful in order to identify trends and test hypotheses. The concept of having each image as a combination of metrics was very useful in order to compare many hypotheses quickly.

The noise test we conducted came up with some interesting results, but it also has its flaws. We should have been more consistent in our use of noise, and used the same noise type for all tests. So that the results would be more comparable. Instead we ended up with two different noise tests: one with other objects as noise and one with grass as noise. The results from these tests diverged from our initial hypotheses; our first guess was that the algorithm would give better result with grass as background than it would with objects as background. We believed this since the grass is uniform and similar over the entire area. However, the results suggests that this may not be the case.

When testing various camera distances and orientations, we found that by cropping the image to contain only the object, this improved the classification results greatly. Thus automatic cropping of objects in the images before classification is something that is definitely worth to look into. This is further discussed in chapter 6.2.

Chapter 6

Conclusion and Further Work

6.1 Summary of Results

The goal for this project was to create an application that runs on the smart glasses and use deep learning algorithms. Our solution is a running application that runs on the BT-200 smart glasses and all other android devices as well. It takes pictures with the built-in camera and sends them to a server. The server classifies the pictures and sends back a list of classes that are ranked. The application displays the top-3 results on the smart glass display, ranked from 1-3. This solution accomplishes the initial project goals for a proof-of-concept application.

We also performed experiments to test how well the algorithm worked with the glasses. From this we learned that there are many attributes that might affect the results. Distance, noise and light all affected in some way, but also various combinations of the three gave different results. From the available time and resources we had, including a small sized database of test images, we can not conclude on what degree the results are affected. This is therefore something that should be further investigated in the future.

Image quality testing came up with some interesting results as well. We found that a high resolution on the pictures are easier for the algorithm to classify. We also saw that the orientation of the pictures and how much the object dominated, affected the results. We have mentioned that a possible approach to this problem be by locating the objects first, and then use bounding boxes to classify each object in the image separately. This is further discussed in chapter 6.2.

6.2 Further Work

As this project only aimed to be a proof-of-concept that demonstrated the possibilities of applying deep learning, a lot of work remains before the application can be useful in any real-world domain. This section sums up some important known issues, and suggests areas in need of improvement.

6.2.1 Object localization

The application makes no effort to locate objects in the images before classifying them. That is, providing bounding boxes of objects in the image, and classify each of the objects individually. We found in chapter 4.3 and 4.7 that by cropping the objects in bounding boxes, we can greatly improve the classification performance, so object localization is definitely a feature that should be emphasized in further development.

Object localization is a task that can very well be solved with deep learning algorithms. GoogleNet, the deep neural network used in our classification application, achieved state-of-the-art results[7] on object localization in addition to classification in the 2014 ILSVRC competition.

6.2.2 Train classifier on domain dataset

The classifier in this project were trained on a subset of the ImageNet dataset. This subset contains 2.1 million images of 1000 different objects. Use of this subset was very useful for demonstrating the proof-of-concept application, but in order to use image recognition and classification in a custom domain such as emergency management, a custom dataset should be built. This dataset could contain scenes and objects that are useful to recognize or identify in an emergency situation. A classifier trained on a specialized dataset would probably produce much more relevant and useful feedback than the general ImageNet dataset.

6.2.3 Real see-through in client application

As mentioned in chapter 3.2 we used MetaioSDK as a foundation of our client application, but we do not actually use much of its functionality. It has many useful features, e.g. object detection algorithms and true see-through functionality. With true see-through functionality, this opens up for the possibility to identifying bounding boxes around object of interest and possibly also faces.

Bibliography

[1] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2014.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[3] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech Recognition with Deep Recurrent Neural Networks,” *arXiv:1303.5778 [cs]*, Mar. 2013. arXiv: 1303.5778.

[4] D. Chen and C. Manning, “A fast and accurate dependency parser using neural networks,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 740–750, Association for Computational Linguistics, 2014.

[5] R. Socher, A. Perelygin, J. Y. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, “Recursive deep models for semantic compositionality over a sentiment treebank.”

[6] “About CIEM - CIEM - Centre for Integrated Emergency Management.” `http://ciem.uia.no/about-ciem`. Accessed: 2015-04-09.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going Deeper with Convolutions,” *arXiv:1409.4842 [cs]*, Sept. 2014. arXiv: 1409.4842.

[8] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, “Recent advances in augmented reality,” vol. 21, no. 6, pp. 34–47.

[9] “Epson Moverio Smart Glasses.” `http://www.epson.com/cgi-bin/Store/jsp/Landing/moverio-bt-200-smart-glasses.do`. Accessed: 2015-04-10.

[10] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, “Deep, Big, Simple Neural Nets for Handwritten Digit Recognition,” *Neural Computation*, vol. 22, pp. 3207–3220, Sept. 2010.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, 2015.

[12] X. Glorot, A. Bordes, and Y. Bengio, “Deep Sparse Rectifier Neural Networks,” pp. 315–323, 2011.

[13] DeepLearning 0.1. LISA Lab, “Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation.” `http://deeplearning.net/tutorial/lenet.html`. Accessed: 2015-04-29.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional architecture for fast feature embedding,” *arXiv preprint arXiv:1408.5093*, 2014.

[15] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, “Learning Deep Features for Scene Recognition using Places Database,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 487–495, Curran Associates, Inc., 2014.

[16] “Epson Moverio Developer Program - Epson America, Inc..” `http://www.epson.com/cgi-bin/Store/jsp/Landing/moverio_developer-program.do`. Accessed: 2015-04-20.

[17] “Augmented Reality SDK for Epson Moverio BT-200.” `http://www.wikitude.com/products/eyewear/epson-augmented-reality-sdk`. Accessed: 2015-04-20.

[18] “Metaio releases first true “see-through” wearable Augmented Reality, now in Metaio SDK.” `http://www.metaio.com/press/press-release/2014/metaio-releases-first-true-see-through-wearable-augmented-reality-now-in-metaio-sdk`. Accessed: 2015-04-20.

[19] D. Amin and S. Govilkar, “Comparative Study of Augmented Reality SDK’s,” vol. 5, no. 1, pp. 11–26.

[20] P. Warden, “DeepBeliefSDK: The SDK for Jetpac’s iOS Deep Belief image recognition framework.” `https://github.com/jetpacapp/DeepBeliefSDK`. Accessed: 2015-05-04.

Appendices

Appendix A

Classification results

	cup	**laptop**	**wallet**	**digital watch**	**hammer**	**banana**
day_clean_close	32	1	2	161	10	5
day_clean_cropped	3	2	1	88	9	2
day_clean_dist	677	2	180	389	118	656
day_noise_close	58	2	9	717	13	181
day_noise_cropped	4	2	44	319	32	73
day_noise_dist	433	29	130	582	150	455
dusk_clean_close	3	8	55	489	2	50
dusk_clean_cropped	4	1	1	503	1	10
dusk_clean_dist	283	14	437	521	15	557
dusk_noise_close	11	1	5	190	43	85
dusk_noise_cropped	10	1	15	38	2	19
dusk_noise_dist	502	2	9	303	155	365
in_clean_close	1	1	2	117	1	1
in_clean_cropped	3	1	2	52	2	1
in_clean_dist	109	6	8	91	38	84
in_noise_close	2	4	28	63	55	33
in_noise_cropped	1	4	5	4	108	1
in_noise_dist	423	4	46	192	64	419

Table A.1: Statistics aggregated from all samples. Values are ranks from 1 to 1000 for the 1000 object categories.

	cup	**hammer**	**banana**
night_noise_close	488	58	413
night_noise_cropped	600	73	653
night_noise_dist	617	314	526

Table A.2: Classification results from nighttime scenes. Values indicate ranks from 1 to 1000.

Rank	**Cup**		**Banana**		**Laptop**	
1	cup	0.453	banana	0,490	laptop	0,517
2	coffee mug	0.278	butternut squash	0,081	notebook	0,464
3	pitcher	0.074	slug	0,068	desktop computer	0,006
4	bucket	0.071	hook	0,029	monitor	0,003
5	water jug	0.054	spaghetti squash	0,026	screen	0,002
6	measuring cup	0.025	toilet seat	0,022	mouse	0,001
7	whiskey jug	0.015	bolete	0,020	computer keyboard	0,001
8	coffeepot	0.011	ladle	0,020	printer	0,001
9	vase	0.003	lampshade	0,013	space bar	0,000
10	teapot	0.003	nematode	0,013	chest	0,000
Rank	**Digital watch**		**Hammer**		**Wallet**	
1	teapot	0,323	hammer	0,922	wallet	0,651
2	puck	0,106	shovel	0,016	purse	0,147
3	brassiere	0,063	hatchet	0,014	buckle	0,029
4	hook	0,049	ladle	0,011	binder	0,025
5	bottlecap	0,031	screwdriver	0,006	mailbag	0,018
6	hair slide	0,026	nail	0,005	pencil box	0,014
7	magnetic compass	0,022	carpenter's kit	0,004	lighter	0,012
8	loupe	0,021	scabbard	0,003	loupe	0,005
9	buckle	0,021	can opener	0,003	digital watch	0,005
10	mask	0,016	hook	0,003	rubber eraser	0,005

Table A.3: Top-10 classification results from objects in base scene(indoor_clean_close)

Appendix B

Algorithm performance versus human performance

Task

Look at the following images and identify the 5 most dominating objects within each image. Rank the objects from most dominating to least dominating.

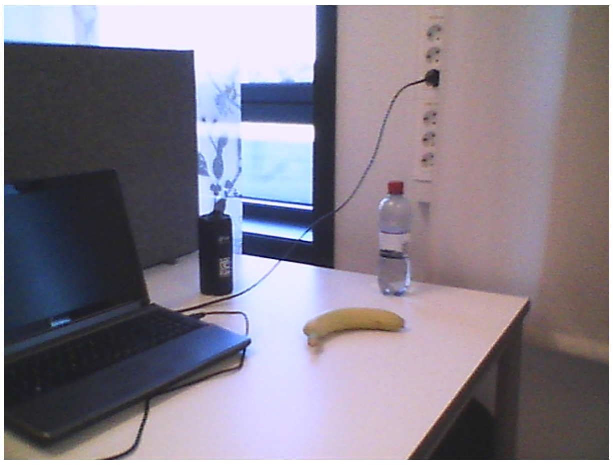


Figure B.1: Test image 1

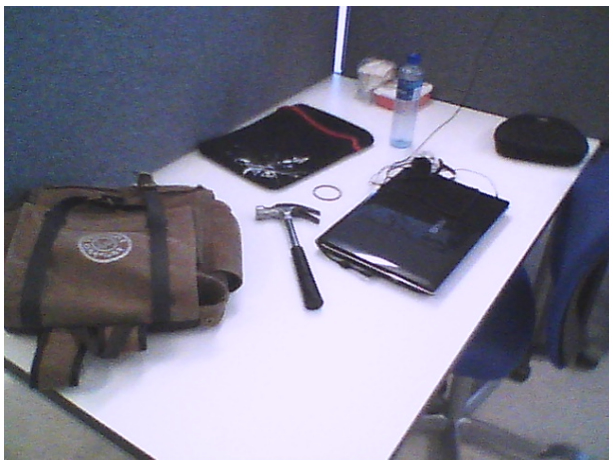


Figure B.2: Test image 2

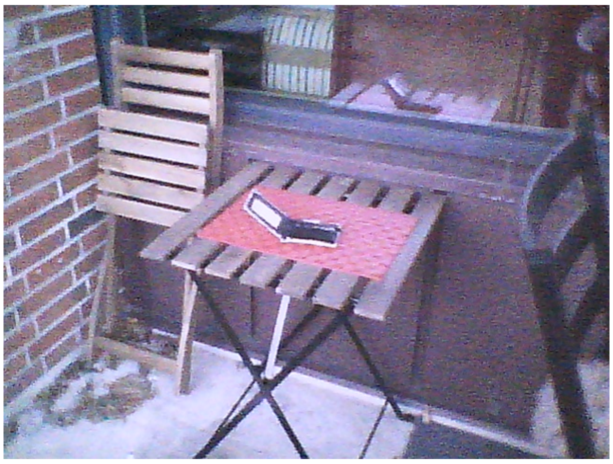


Figure B.3: Test image 3

Results

	Image 1			
	Subject 1	*Subject 2*	*Subject 3*	*Algorithm*
1	Banana	Banana	Banana	Washbasin
2	PC	PC	Water bottle	Desk
3	Water bottle	Dark wall	Coffeemug	Desktop computer
4	Desk	Bottle	PC	Screen
5	Window	White table	Desk	Monitor

Table B.1: Top-5 results on image 1

	Image 2			
	Subject 1	*Subject 2*	*Subject 3*	*Algorithm*
1	Hammer	Hammer	PC	Switch
2	PC	Brown bag	Hammer	Hard disc
3	Bag	Black PC	Computer-bag	Printer
4	Desk	Black PC folder	Backpack	Polaroid camera
5	Elastic band	Bottle	Desk	Desk

Table B.2: Top-5 results on image 2

	Image 3			
	Subject 1	*Subject 2*	*Subject 3*	*Algorithm*
1	Table	Cellphone	Wallet	Folding chair
2	Table cloth	Red tablecloth	Table	Shopping cart
3	Thing	Table	Table-cloth	Doormat
4	Window	Chair	Folding chair	Carpenter’s kit
5	Chair	Brick wall	Wall	Park bench

Table B.3: Top-5 results on image 3