Bachelor thesis
Computer Science

Radboud University

Computationally Feasible Logo Recognition using Deep Learning

Author:
Simon Brugman
s4151437

Supervisor:
prof. dr. Tom Heskes
`t.heskes@science.ru.nl`

July 2, 2015

Abstract

This thesis explores the visual task of logo recognition using deep learning with the special constraint that it should be computationally feasible. Using a single-step approach, I fine tune pre-trained convolutional neural networks to minimize the training costs. The final model, that is trained in only a few hours on a CPU, has an accuracy of 88.7% on the validation set.

Contents

List of Abbreviations

2GD	Second Order Gradient Descent
2SGD	Second Order Stochastic Gradient Descent
ANN, NN	(Artificial) Neural Network
BoVW	Bag of Visual Words
CIFAR-10	Data set consisting of 10 classes that is named after Canadian Institute for Advanced Research
CIFAR-100	Data set consisting of 100 classes that is named after Canadian Institute for Advanced Research
CNN	Convolutional Neural Network
GD	Gradient Descent
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
LRN	Local Response Normalization
MLP	Multi-layer perceptron
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SIFT	Scale-invariant feature transform

Chapter 1

Introduction

1.1 Real world logo recognition

Computers are bad at logo recognition, at least by nature. Several methods have been developed to perform visual tasks with human-like (or better) performance. The most common way is "hard computer vision", a generic term for all kinds of image processing transforms[43]. Recently Google and Facebook published papers in which they use another approach: deep learning[42, 44]. This approach is not new, but has been rapidly becoming more and more popular. The visual task in these papers is to recognize objects from images, for instance in The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)[31].

My thesis' goal is to find out whether it is feasible to recognize logos from real world images using deep learning. Media11[1] is a company based in Arnhem which specialises in social apps and social data and is interested in this research question. Because of the international character of this companies special interest will go out to brands of luxury goods in images. The real world images that are intended, are images published on mobile photo-sharing services such as Instagram, Facebook or Flickr. In addition to this, solutions that can be implemented in a computationally inexpensive way are preferred, as this requires less monetary investment.

In addition to the thesis' goal the following questions arise:

- What deep learning research has been done already in the context of logo recognition in images?
- Is it possible to build a prototype that recognizes logos in real world images using deep learning in a way that is computationally feasible?

[1]`http://www.media11.nl`

Regarding these research questions I suspect there is limited research done on logo recognition using deep learning, but that there is a very large research base on other methods and similar tasks, including the ones mentioned in the introduction. My hypothesis is that it is possible to build a prototype that can recognize logos in real world images via deep learning up to reasonable performance (for example, at least 75% accuracy).

1.2 Contributions and Outline of This Thesis

The contribution of this thesis mainly lies in the exploration whether the current theory of deep learning is enough to classify logos and whether it is possible to apply this theory computationally feasible for commercial purposes.

Several examples in what kind of applications this knowledge could be used are:

- Business Intelligence tools, for example for clustering people in social media based on how many images with a specific logo they upload.
- Vehicle logo recognition could be used to monitor traffic in highway toll systems, public security, help identify other vehicles in smart driving assistant software and so on[25, 24].
- In searching large quantities of digital documents, logo recognition can support identifying the source of a document and therefore derive more information from the document[19].

There are several ways to handle the task of recognizing an image. An important distinction is to choose one of two approaches. The single-step approach is to classify images in different logo classes or the class "no-logo" when the image does not contain a logo. The double-step approach is to first determine if there is a logo in the image, if this is the case then extract and classify it. The single-step approach takes less implementation and training time, while the double-step approach, splitting the detection and recognitions tasks, could be more accurate and could be able to recognize more logos in a single image. In this thesis many decisions are made where one takes more computational efforts and the other could be more accurate. To answer the question if deep learning is feasible for recognizing logos, it is better to start with the less computationally expensive solutions and improve the performance afterwards. If the possible performance gain is high, then one could consider delving deeper into that particular area. Also, the single-step approach is easily adaptable to the double-step approach, as the single-step approach is similar to the second part of the double-step approach.

Using the single-step approach, let us determine which methodology to use. Deep learning involves the use of neural networks. There are two ways to do this: create a model from scratch or use a pre-trained network. The same consideration as for previous decision applies: using a pre-trained network requires less computational efforts as it requires less training and hence is preferred. Fine tuning a network is a popular manner to approach problems on smaller datasets[38]. I will elaborate on how fine tuning a pre-trained network is achieved in the third chapter. Basically this takes place by using an already trained network and by changing only a few parts, such as the last layer and by keeping the original trained features constant. Another benefit of using pre-trained networks, next to being computationally less expensive, is that it is possible to reuse state-of-the-art performing networks, without having to have a complete understanding of all its optimization procedures. A disadvantage is that a pre-trained network is trained on different data, another problem or both. Again, a pre-trained network is a good starting point for exploring the feasibility of deep learning on recognizing logos. Modern convolutional networks take 2-3 weeks to train across multiple GPUs on ImageNet. It is common to see people share their final model for the benefit of others. For example, the Caffe library has a Model Zoo where people share trained features of their network.

Since there is a limited set of available frameworks that are providing pre-trained networks that are trained on image recognition tasks, I will consider if there is one that satisfies our needs, and if so, which one applies best. The following were available at the time of writing this thesis:

OverFeat OverFeat was trained on the ImageNet dataset and participated in ILSVRC 2013[34]. It was the winner of the localization task and obtained very competitive results for the detection and classifications tasks. Afterwards, OverFeat we establish a new state-of-the-art for the detection task.

Caffe Caffe is a deep learning framework made with expression, speed, and modularity in mind[12]. It is developed by the Berkeley Vision and Learning Center (BVLC) together with community contributors. In its so called 'Model Zoo', the following pre-trained networks are available: GoogleNet, AlexNet, CaffeNet, Very Deep 16 layers, Very Deep 19 layers, Network-in-Network.

MatConvNet MatConvNet is a MATLAB toolbox implementing convolutional neural networks (CNNs) for computer vision applications[45]. It is relatively simple, efficient and can run and learn state-of-the-art CNNs. Several pre-trained networks are available: Very Deep 16 layers, Very Deep 19 Layers, AlexNet, CaffeNet.

libccv libccv calls itself "A Modern Computer Vision Library" and has similar accuracy to Caffe and OverFeat[20]. It includes the AlexNet model and some others. It does not need other dependencies than a C compiler and is therefore useful in mobile applications, such as a Raspberry Pi.

OverFeat and Caffe focus primarily on speed, while MatConvNet and libccv focus more on accuracy. As Caffe offers more different pre-trained networks, more documentation and does not require GPUs, I chose this framework (in line with earlier decisions on speed over accuracy).

1.3 Summary of the remaining chapters

In chapter 2 I will summarize the theoretical background that is available by going into previous logo recognition research, explaining CNNs in general and describe recent achitectures that were used by for example Google and Facebook. Chapter 3 describes the approach for pre-training a prototype that recognizes logos. Several visualisations of the results are discussed Chapter 4, as well as a comparisson to other research results on the same data set. Chapter 5 is a summation of reviews of possible improvements and indications for further research. Finally, chapter 6 answers the research questions.

Chapter 2

Literature review

In this part of the thesis I will go into more details on current logo recognizing efforts published and how current techniques using deep learning work. Therefore, I will review current literature and explain the working of a convolution neural network briefly, explain which parameters can change the performance of the model and finally discuss the structure of a state-of-the-art network.

2.1 Logo recognition, earlier work

Logo recognition is a well-studied problem, especially focused on recognizing vehicles. Summarized for all relevant research in the available literature, either one or more of the following statements apply: the research uses no real world images as described in the introduction, it only focuses on vehicle manufacturer logos, it does not use deep learning or it uses rather outdated neural networks.

Computer vision techniques: Bag-of-Visual-Words and Min-Hashing

Researchers from the University of Augsburg together with Yahoo! Research developed a scalable logo recognition method for real-world images in 2011[29]. Revaud et al. tried to tackle a problem that occurred where for example windows under a certain rotation where seen as the Adidas logo[26]. Then in 2013 the University of Augsburg's researchers enhanced this method, creating state-of-the-art performance[28]. It is based on the Bag-of-Visual-Words (BoVW) and Min-Hashing techniques to reduce and bundle the image to a description of local features. Romberg et al. also use data augmentation in two different settings. By applying several transformations on the training set, the training set size is increased and by applying similar applications to the image that is queried. These are known ways to enhance performance for small visual data sets as they try to overcome the problem that objects are only seen from a single perspective.

On a related note, BoVW was recently used in Japan to reconstruct images from models which can help to suggest ways to improve the performance of a model[15].

Vehicle Logo Recognition

In the introduction I already referred to the vehicle logo recognition applications. In 2010, Psyllos et al.[25] pointed out that many papers are about recognizing vehicle types, but few about recognizing their brands. In their paper they propose a SIFT-based feature-matching algorithm.

The Chinese government also tried to tackle this problem and compared the methods convolutional neural networks and SIFT descriptors[24]. Their comparison says: "By comparing the results based on two different approaches in the experiment, the average recognition accuracy rate of the approach based on CNN is 8.61% higher than the results of the approach based on SIFT.", which is quite promising.

There is a lot of information available on vehicle logo recognition. The systems are different to the one we are interested in in the following way: these systems are designed to a specific kind of logos, which are less expressive than logos such as that of Starbucks and Google and are found under less different circumstances (mostly, on the vehicle itself).

Other neural-based architectures

Neural-based architectures to recognize logos in general were published in the late 90s. The overall observation in these architectures is that they do use a fully connected structure and not yet the CNN-based approach that are used in state-of-the-art solutions to different visual tasks.

In 1997 Cesarini et al.[5] published "A neural-based architecture for spot-noisy logo recognition" in which ANNs are used to perform the task. The noise in the logos is yet minimal compared to real world images. Then, one year later, some of the same researchers improved this method by introducing recursive neural networks[8]. This method is able to learn logo recognition by looking at the contours of the logos. It does not perform well with similar logos and is not trained on real world images.

The most recent study found in this direction is from 2000 and experienced several difficulties: "Several issues cause difficulty when using a GRNN to classify logo tiles in this way-text as part of the logo, very small logos, all black or all white tiles, and discrimination between different logos." [50]

2.2 Visual recognition using deep learning

This section will describe the way modern deep learning approaches to visual recognition tasks work, by explaining about (modern) neural networks. If you would like to know more about the history of deep learning, I can refer you to the article "Deep learning in neural networks: An overview"[33].

To understand what a state-of-the-art network consists of, we first have to understand how a neural network works, then what the basic structure is of an convolutional neural network and then what adjustments and techniques are applied to tweak its performance.

Neural Network, the basic concept

When talking about neural networks, we typically refer to artificial neural networks rather than biological ones. Neural networks are nonlinear computational structures, modelled on the brain, constructed of atomic components called "neurons". Neurons can be defined using the McCulloch-Pitts Model[23] as visualized in Figure 2.1. It consists of a bias, one or more input links with their corresponding weights, an activation function and one or more output links.

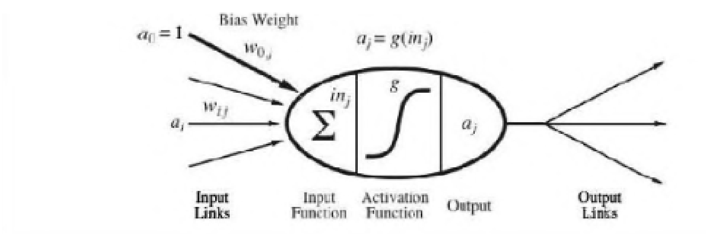


Figure 2.1: A common mathematical representation of a neuron, the atomic building blocks of any neural network. The neuron receives input, applies an activation function and passes the result trough its output links[32].

A neuron calculates an activation function over the sum of its weighted inputs and passes the result to its output link or links[32]. A neural network is an interconnected collection of those neurons. The activation function is typically a step function (holding a threshold θ) in which case a neuron is called a perceptron, or a logistic function, in which case the neuron is sometimes called a sigmoid perceptron. Both types of activation functions ensure the property of a neural network to represent a nonlinear function. Frequently used logistic functions are sigmoid and hyperbolic tangent (tanh). We will see two other used nonlinear activations further on.

Step function:

$$y(u) = \begin{cases} 1 & \text{if } u \geq \theta \\ 0 & \text{if } u < \theta \end{cases} \tag{2.1}$$

Sigmoid:

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

Hyperbolic tangent:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.3}$$

There are two, fundamentally different, ways to connect the perceptrons together to create a network[32]. The first way is using a recurrent network. It feeds its output back to its own input. This means that the activation levels of the network form a dynamic state which can either become stable or a total chaos. Due to the fact that the state of a neuron depends on previous calculations, the recurrent network can support a form of short term memory. The other is a feed-forward network. This network only has connections in one direction and therefore is acyclic. Every perceptron receives input from "upstream" nodes and delivers output to "downstream" nodes. I will focus on the feed-forward network structure. As the task at hand is static, there is no need for a recurrent network.

Layers

Neurons in an feed-forward network are arranged in layers in which a neuron only receives input from a neuron from the preceding layer, but it is also possible to create single-layer feed-forward networks. The nonlinear property makes it useful to classify for example XOR as shown in Figure 2.2.

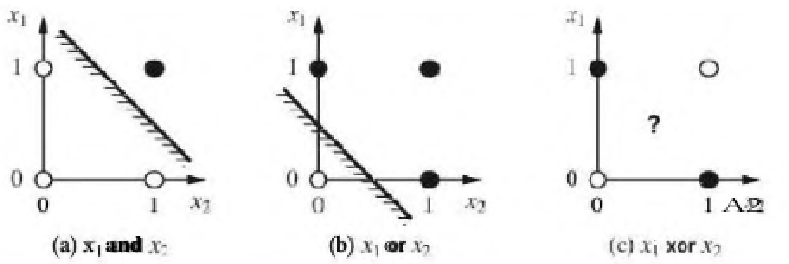


Figure 2.2: Functions for logic functions (a) and, (b) or and (c) xor. The examples in (a) and (b) can be linearly separated. For (c) this does not work, hence a nonlinear classifier is required[32].

A single-layer network does not perform well enough for complex problems like recognizing logos, so we need multiple layers. Multilayer perceptrons (MLPs) or multilayer feed-forward networks have, next to the input and output layer, one or several hidden layers. These layers are connected to the preceding and the following layer, creating a kind of black box, invisible to the outside world. In traditional artificial neural networks layers are fully connected to the preceding and following layer. The number of inputs is equal to the number of neurons in the preceding layer (plus the neurons bias) and the number of outputs is equal to the number of neurons in the following layer. Later on we will see examples when this is not the case.

Training

The structure of layers, different weights and biases of neurons and which neurons are connected change the output of the output layer and with that the performance of an network on a task. A neural network can be seen as a tool for nonlinear regression or classification. Therefore it needs to be fitted e.g. learn a suitable mapping from a given data set. This process is called training.

The training will be based on the definition of a suitable error (or loss) function, which then is minimized with respect to the weights in the network [1]. Error backpropagation is a popular algorithm to train a neural network with hidden layers and is used in conjunction with an optimization method such as stochastic gradient descent [47].

Since this method requires computation of the gradient of the error function at each iteration step, we must guarantee the continuity and differentiability of the error function. We have to use a kind of activation function besides the step function used in some perceptrons, because the composite function produced by interconnected perceptrons is discontinuous, and therefore the error function too. One of the more popular activation functions for backpropagation networks is the sigmoid, as seen in the beginning of this section. We will see that the property of differentiability of the activation function is also used in some modern layer types.

The backpropagation algorithm for a multi-layer neural network consists of four steps that are iterated during training to try to minimize the total error (via gradient descent)[27]:

I The first does a feed-forward computation through the network of the input. Some bookkeeping needs to be performed to store the derivatives of the activation functions.

II Secondly, backpropagation of the output layer takes place.

III Then, as third step, the hidden layers are backpropagated.

IV Finally, when the error is known, the weights are updated in order to minimize this error.

In the last step of backpropagation iteration, there are several parameters involved, such as momentum and weight decay. I will summarize the most common parameters and describe how they affect the model's performance. Some of these parameters try to prevent the method from overfitting by trying to find the global minimum instead of finding local minima.

Calculation of the loss or error function is done by different methods such as gradient descent (GD), second order gradient descent (2GD), stochastic gradient descent (SGD) or second order stochastic gradient descent (2SGD)[2]. Without covering all these methods, I want to notice their most important difference. While gradient descent uses all training examples for one single update, stochastic gradient descent only takes one (or a few, called a mini-batch), trying to approach the real error. Although gradient descent is worse at optimization, it converges faster than gradient descent. SGD should be used when training time is the bottleneck. Choosing to use SGD is therefore in line with the research goal to find a computationally feasible algorithm.

Training parameters and their impact

Learning rate

Real Domain $[0, 1]$ - *Typical Value* 0.1
The least complex form of gradient descent updates multiplies every weight update with a negative gradient direction, which is a constant, the learning rate[39]. As the learning rate applies to updates during backpropagation, it is also possible to choose different learning rates for each layer. This possibility is used for fine tuning, which we will see later on.

In training deep networks, it makes sense to decrease the learning rate over time[39]. A higher learning rate towards the end of the training gives the system too large changes to settle in a deeper but narrower part of the loss function. When to decrease the learning rate can be complex, decaying too slow takes extra computational effort bouncing around with little improvement for a long time. Decaying too fast will prevent the system from reaching the best position it can. There are three common ways to implement learning rate decay. The step decay is in practice the most preferable. It reduces the learning rate every few iterations by some value. I will describe its hyperparameters below.

Step iterations

Integer Domain $[0, \infty)$ - *Typical Value* $20,000$
After this number of iterations, the learning rate is multiplied with a factor, which we call step gamma[39]. The value of the hyperparameters concerning step decay are heavily influenced by the nature of the problem and the depth of the net. One heuristic that is used in practice is to watch the validation error while training with a fixed learning rate, and reducing the learning rate with a step gamma whenever the validation error stops improving.

Step gamma

Real Domain $[0, 1]$ - *Typical Value* 0.5
The step gamma is the factor with whom the learning rate is multiplied as described above.

Maximum Iterations

Integer Domain $[0, \infty)$ - *Typical Value* $100,000$
When is a network finished training? One way of doing this is by stopping after a number of iterations. After these iterations, the models features are saved. By stopping early one can prevent a model from overfitting[49]: the model is too far adapted to the training data and its performance on unseen (test)data will decrease. The difference between validation and training accuracy can be an indicator for overfitting. More on preventing overfitting in this case is in the next chapter (Figure 3.3).

Momentum

Real Domain $[0, 1]$ - *Typical Value* 0.9
Momentum adds a fraction m of the previous weight update to the current update[48]. This is to try to prevent the network from getting stuck in a local minimum. Momentum can help the network to converge to a lower loss[41]. Taking a momentum that is too high will result in a system that overshoots its minimum and therefore becomes unstable. A momentum that is too low will still get stuck in a local minimum. When using gradient descrent, momentum amplifies the step size. When increasing the momentum, the learning rate needs to be reduced.

Weight decay

Real Domain $[0, 1]$ - *Typical Value* 0.1
This parameter lowers the weight every time it is not updated[49]. A weight decay

parameter is useful for avoiding large features, which slow down the network and are an indicator of overfitting. Informally this can be explained this way: a network describes a smooth function. Large features are more likely to produce areas with a large curvature. Hence weight decay will regularize the network.

Convolutional Neural Networks

Types of layers used vary hugely by the network design. By looking at some different networks in chronological order, I will cover the working of various layer types.

Example 1: LeNet-5 (1989)

Without going deep into the history of convolutional neural networks, I present one of the simplest and first of them: LeNet-5. Yann LeCun, now researcher at Facebook, developed this net to recognize handwritten digits in 1989. It demonstrates the traditional structure of a convolutional neural network: alternating a convolutional layer and a pooling layer, a number of times, followed by two fully connected layers[18]. The main idea of convolution is training kernels that recognize useful features in an image, such as 'corners', 'eyes' or 'edges' . A CNN makes the explicit assumption that the input is an image, and therefore making certain architectural choices that improve the network's efficiency.

One of these choices is not fully connecting each layer, as neural nets do not scale to full images. For example, an image of sizes 200x200x3, would lead to neurons that have $200 * 200 * 3 = 120,000$ weights. Having several such neurons, the parameters would add up very fast. This huge number of parameters is wasteful and prone to overfitting. Another assumption is that kernels that recognize features in position (x_1, y_1) is also useful in position (x_2, y_2). The kernels use the same weights, called parameter sharing[37].

There are three main sorts of layers to build convolutional network architectures: the convolutional layer, the pooling layer, and the fully-connected layer (identical as in other neural networks)[37]. We will stack these layers to form a full convolutional network architecture.

Convolutional layer The convolutional layer is, obviously, the core layer type of the convolutional network[36]. The input to this layer is an image with square dimensions $m * m * d$ where m is the height and width of the image and d is the depth, typically 3 for color images (RGB) or 1 for grayscale images. The layer itself consists of k kernels or filters with dimensions $n * n * q$ where n is smaller than original image dimensions and q can either be the same or smaller than d. q can also differ per kernel. The output is a feature map for each kernel. The layer takes, next to the filter dimensions, the hyperparameters stride and padding.

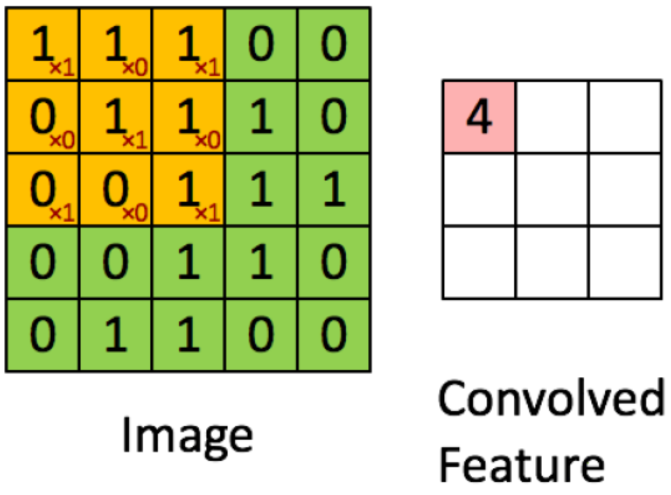


Figure 2.3: The first convolution that is executed by the convolutional layer. The image is the layer's input, the yellow square is a kernel. By multiplying the kernel values with the values of the input, the value of the convolved feature at that position is calculated [40].

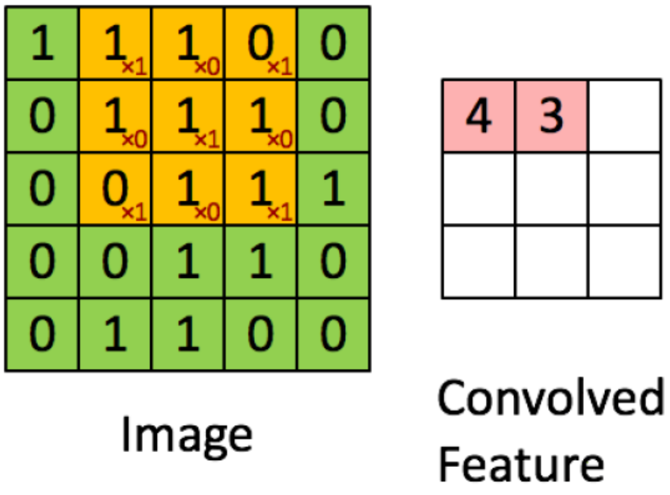


Figure 2.4: The calculations of this second convolution are identical to first convolution as shown in Figure 2.3, except for the fact that the kernel moved by an stride $S = 1$ to the left[40].

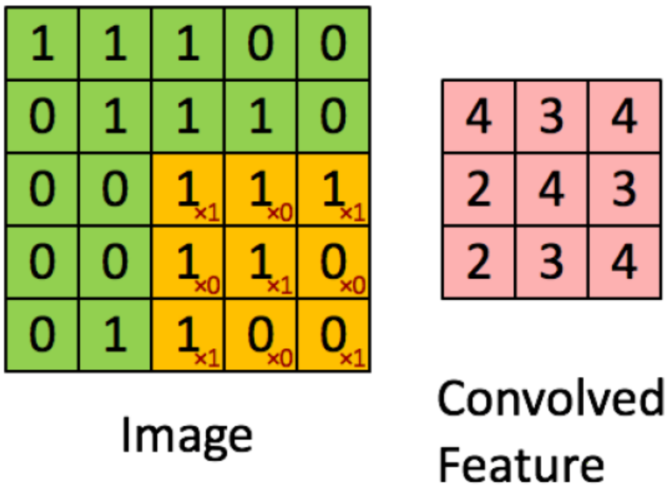


Figure 2.5: After the last convolution the convolved feature is filled with values obtained from convolution. The size of the convolved feature is determined by the size of the input, combined with the hyperparameters for stride and padding[40].

In Figures 2.3-2.5, the process of calculating a convolved feature is shown. In this example, the stride $S = 1$, which means that the kernel moves one step at a time. Another common value for S is 2. In this case the kernel would move two steps at once. The padding $P = 0$, meaning that no extra padding is added. There are different methods to add padding, the most common is to add zero's[40]. The padding adds a border of size P to the input, for instance, if the input size is 10x10 it will be 12x12 for P = 1, 14x14 for P=2 and so on. In the example the kernel size is 3x3. The size of the convolved feature (for square kernel and input sizes) can be calculated from all (hyper)parameters: $(m - n + 2 * P)/S + 1)$.

Pooling layer A pooling layer is commonly inserted between two successive convolutional layers[40]. 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." In other words, it "downsamples" the representation to reduce parameters. In general, pooling can be done with different functions, such as average or max pooling. Max pooling has shown to work best in practice. The layer takes a few hyperparameters which are part of the model architecture. These are the filter size F and the stride S. Figure 2.6 shows an example of max pooling with $F = (2, 2)$ and $S = 2$, as used in LeNet-5. Taking the stride S the same value as the filter size will results in no overlay in the result, in contrast to the $S = 1$ we saw in the convolutional layer example.

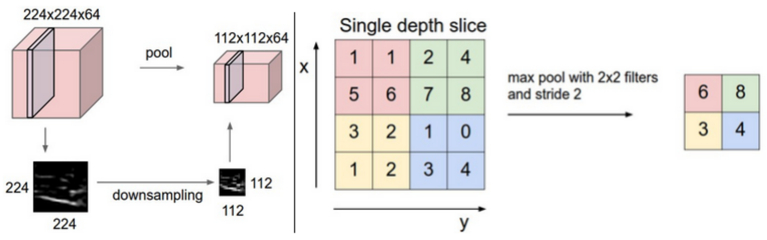


Figure 2.6: An example of max pooling with 2x2 filters and stride 2. On the left the effect that pooling has in the data dimensions are displayed. The right shows a single calculation: the filter with dimensions 2x2 moves in in steps of 2 (= stride). [37]

Fully-connected layer The last layer type we have not described yet in this model is the fully-connected layer. This functions exactly as the layers in traditional neural networks, as described earlier.

Example 2: AlexNet (2012)

AlexNet is the first popularized convolutional neural network for computer vision[37]. AlexNet was submitted to the ImageNet ILSVRC challenge in 2012 and performed significantly better than its competitors. The network had a similar architecture basic as

LeNet-5, but was deeper, bigger, and featured convolutional layers directly stacked on top of each other opposing immediately separated by a pooling layer.

In addition to the layers already in LeNet-5, AlexNet uses some new layers types which add to the network's performance.

ReLU layer We will begin with the simplest one to explain. The ReLU layer element-wise applies an activation function[17]. The layer is named after the rectifier function, defined as $f(x) = max(x, 0)$. The softplus function is also commonly used, which is a smooth version of ReLU. The formula for this is $f(x) = ln(1 + e^x)$[46]. The output of this layer will have the same volume as the input[37]. The main advantage of using the non-saturating ReLU activation function over saturating nonlinearities in combination with gradient descent, is that it is much faster: computations with the zero that ReLU outputs are computationally less expensive than computations with the really small values from saturating nonlinearities such as the sigmoid function.

Dropout "layer" Dropout is a relatively new regularization method introduced by the researchers behind AlexNet, Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton[10]. They stated that combinate the output of various models is a successful way to reduce test errors, but it appears too expensive for large networks. "Dropout" is an efficient way of combining models (actually versions of the same model), by setting the output of each hidden neuron to zero with a probability of for instance 0.5 in each backpropagation step[17]. This way the deactivated neurons do not participate in backpropagation. Since a neuron cannot rely on others too much, it is forced to learn more robust features.

Local response normalization (LRN) layer The ReLU layer has the property that it does not require input normalization to prevent it from saturating, on the condition that at least some training examples produce some positive input to a ReLU layer. Still, research finds that adding LRN improves the generalization. The LRN layer applies the following formula:

$$b^i_{x,y} = a^i_{x,y}/(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} (a^j_{x,y})^2)^\beta \tag{2.4}$$

$b^i_{x,y}$ is the response-normalized activity for position (x, y) by applying kernel i[17]. $a^i_{x,y}$ denoted the activity of a neuron with kernel i applied at position (x, y). The sum runs over n "adjacent" kernel maps at the same spatial position, N is the total number of kernels in the layer. The constants k, n, α and β are hyperparameters whose values are determined using a validation set. This layer is a type of regularizer that encourages "competition" for big activities among nearby groups of neurons.

Example 3: Network-in-Network (2013)

As the name Network-in-Network hints on, this architecture uses “micro” networks inside their complete network[21]. Here the normal linear convolution layers are replaced with an MLP convolution layer, which is in fact a small network itself. Both convolutional structures are shown in Figure 2.7. The Network-in-Network architecture stacks three of these MLP convolution layers in total.

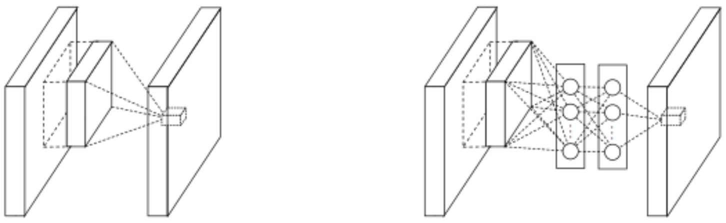


Figure 2.7: A linear convolution layer (left) and an MLP convolution layer (right). The second contains a micronetwork inside the layer[21].

Another alteration in the structure of Network-in-Network is that it replaces the last fully connected layers by globally connected average pooling layers. This prevents overfitting and makes the network better interpretable: there is a direct connection between classes and their feature maps.

Network-in-Network achieved, when it was presented, the highest results on the CIFAR-10 and the CIFAR-100 data sets[16]. Both sets consists of 50,000 training examples and 10000 test examples that need to be classified in respectively 10 and 100 non-overlapping classes.

Example 4: GoogLeNet (2014)

The next two examples are the most state-of-the-art networks currently available. GoogLeNet is, compared to, LeNet-5, a massive neural network[42]. It ended first at ILSVRC-2014[31]. The main contribution is the “Inception” architecture, making a network use less parameters. For example, 12 times fewer than AlexNet, while being significantly more accurate.

The “Inception” architecture is a combination of other components we already covered in the networks earlier presented. In general, an Inception network is a network consisting of (in this case 9) modules of the above type stacked upon each other, with occasional max-pooling layers with stride $S = 2$ to halve the resolution of the grid.[42]

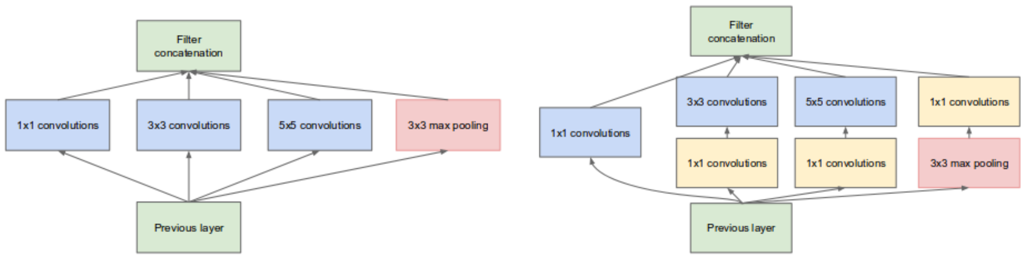


Figure 2.8: GoogLeNet's main contribution are the inception modules. Left is the naive version, right with dimension reductions[42].

GoogLeNet uses many optimization methods from earlier networks. For example a Dropout layer as presented in AlexNet and the use of 1 x 1 convolutional layers that were also used in Network-in-Network[42].

Example 5: VGG 16/19 layers (2014)

To conclude, I want to mention the most preferred network architecture for visual tasks at time of writing[37]. The Visual Geometry Group (VGG) from the University of Oxford submitted two models to ImageNet ILSVRC-2014. The networks Very Deep 16-layers and Very Deep 19-layers secured a first and second place in respectively localisation and classification.

The VGG proved that the depth of a network has a huge impact on its performance, which is feasible by using very small (3x3) convolution filters in all layers. This size is the smallest that has a notion of left/right, up/down and center. The convolution stride is fixed to 1 pixel; the spatial padding is chosen so that the spatial resolution is preserved after one convolution i.e. the spatial padding is 1 for 3x3 layers (this is comparable to Figures 2.3-2.5 but with a spatial padding of 1, adding). Layer types that are used, are the same as in for example GoogLeNet, except that the Local Response Normalization (LRN) layer is not included (in almost every model), as it does not seem to improve the performance[6, 35].

Although GoogLeNet performed better at classification at ILSVRC-2014, it was later found that despite its slightly weaker classification performance, the VGG ConvNet features outperform those of GoogLeNet in multiple transfer learning tasks[37].

Appendix IV contains visual representations of the networks described above.

Chapter 3

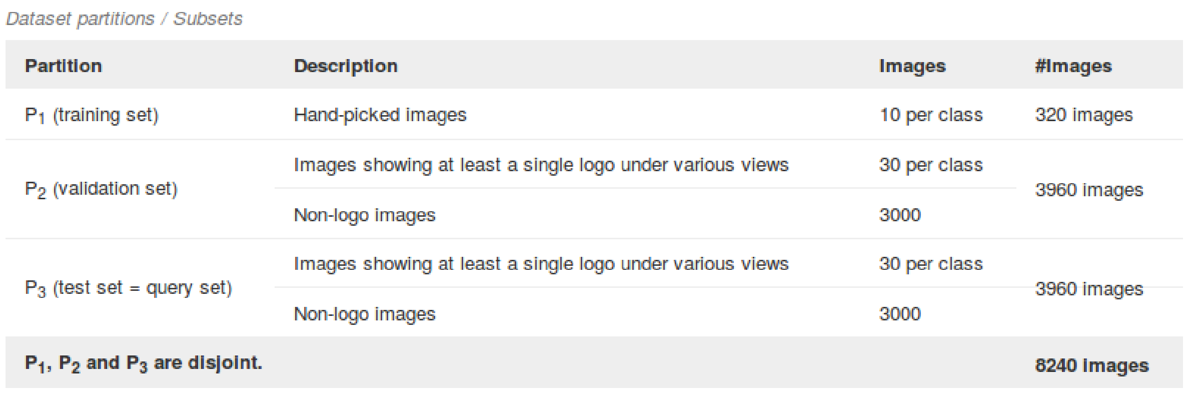
Method

As concluded earlier, it is the most practical to start building a prototype by using a pre-trained network. Caffe offers a deep learning framework that is very expressive. To determine which of the build-in pre-trained networks is most suitable as a base for recognizing logos, we consider the following networks: GoogleNet, CaffeNet (tweaked version AlexNet), AlexNet, Network-in-Network, Very Deep 16 layer and Very Deep 19 layer.

GoogleNet and the Very Deep models are no feasible options, because they take days to run on the available hardware, even for fine-tuning. Therefore there are three candidates left: CaffeNet, AlexNet and Network-in-Network. Experimenting should point out if at least one of them performs good enough to indicate the feasibility of deep learning on recognizing logos.

3.1 Data set

Rather than collecting a brand new data set, an existing data set is preferable, as it takes less resources than gathering an own data set. The University of Augsburg provides the "FlickrLogos-32"-data set that is freely available: a data set with Flickr-images of logos with either one logos or no logo in an image. This data set consists of 8240 images, of which 2240 are images with logos[30]. It is divided into a training set, a validation set and a test set as Figure 3.1 illustrates. This partioning results in low accuracy scores for the current method. Hence, the data set was repartioned by merging the training and the validation set as the new training set and using the test set as both the test and validation set. More on the results of this decision in the discussion chapter.



Dataset partitions / Subsets

Partition	Description	Images	#Images
P_1 (training set)	Hand-picked images	10 per class	320 images
P_2 (validation set)	Images showing at least a single logo under various views	30 per class	3960 images
	Non-logo images	3000	
P_3 (test set = query set)	Images showing at least a single logo under various views	30 per class	3960 images
	Non-logo images	3000	
P_1, P_2 and P_3 are disjoint.			**8240 images**

Figure 3.1: The different partitions of the "FlickrLogos-32"-data set consists of 32 classes with logos and the majority of images without a logo[7].

Reasons to use this data set are that it is rather small so fine tuning will take less time, it has a wide variety of well-known and less-well known logos. The data set has been used earlier by other researchers, so it is possible to compare the prototypes' results to them. Other data sets, such as the BelgaLogo data set[13] and the "FlickrLogos-27" data set[14] do not have the restriction that there is only one class of logo in an image and are therefore less suitable with respect to the single-step approach decided upon earlier.

3.2 Input data

The input of the images is a square, the only preparations are taking a random square crop for every example during training time and subtracting the mean RGB value calculated on the training set. During test time the center crop is used. A way of improving could by implementing data augmentation: by taking the average of multiple crops, such as rotations, mirrors and side crops, more training samples could be generated. Data augmentation is a popular way to increase training data[28]. Note that the way of providing the input data could be changed from files directly from storage to an efficient database LMDB. Implementing this costs more time, but will speed up the training process.

3.3 Fine tuning

Fine tuning a net to a new task is in fact the replacement of the last layer with one of our own. This way the network outputs the desired classes instead of the classes it was originally trained on. A prerequisite is that the features of the original model should also "work" for recognizing logos. The fact that the features are already trained save lots of computational time by shortening the training time dramatically.

The models I use, Network-in-Network, CaffeNet and AlexNet are all pre-trained on ImageNet. The task given to those networks is to distinguish 1000 object classes. The networks are trained on 1,000,000 images. The first step of fine tuning these models is to change the last layer so that it outputs 33 classes (32 logo classes, 1 no-logo class). By altering the last layer, its trained weights are reinitialized and so that it starts training with random weights. During every training iteration, a batch of 50 images is used to fine tune the model.

The second step is to choose suitable (hyper)parameters for training the fine tuned models, such as the learning rate and maximum number of iterations. Virtually, the training proccess for the model has progressed, except for the one changed layer. This means the last layer has to learn much, while the other layers should stay similar. Using this as a guideline, we reduce the total learning rate of the model. In examples found online, such as the fine tuning of the Oxford flower data set, it is common to see a boost of the learning rate of the last layer, relative to rate on other layers[3]. It is also possible to prevent fine tuning of all layers except the last, by setting their learning rate to zero[4]. Finding suitable hyperparameters is often done by experimenting. As for the learning rate, I choose a heuristic approach presented earlier: first choose a fixed learning rate and reduce the learning rate by a constant (e.g. 0.5) whenever the validation error stops improving[39].

There are several quantities that provide information on the training parameters and could give information on how they should be changed for more efficient training. In this experiment I will keep track of the loss and the accuracy[39]. Figures 3.2 and 3.3 contain both examples of desirable and undesirable results of these quantities. I will compare my results to these figures in the next chapter.

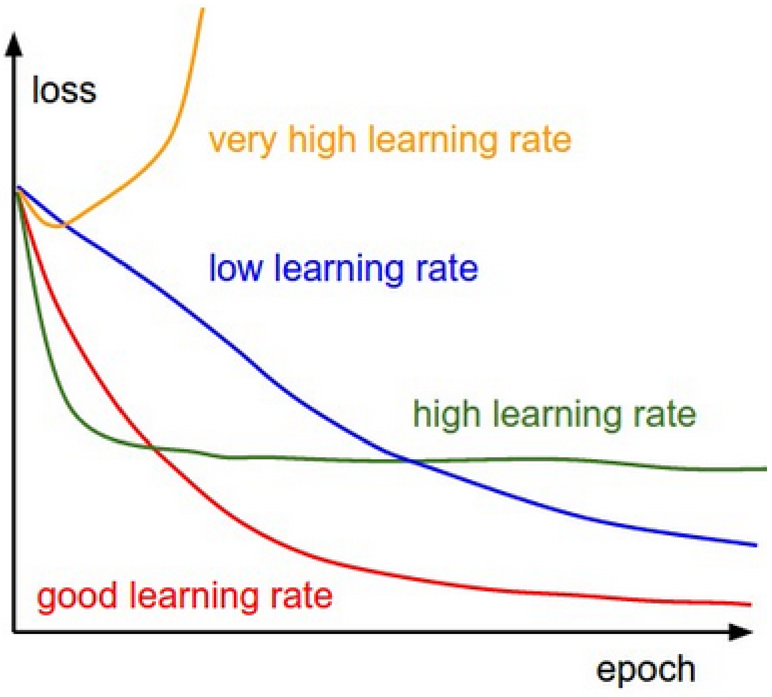


Figure 3.2: The exaggerated effects of different learning rates. A low learning rate will result in an linear improvement. A high learning rate will look more exponential, but will have difficulties finding an optimized spot because of the relatively large changes. [39]

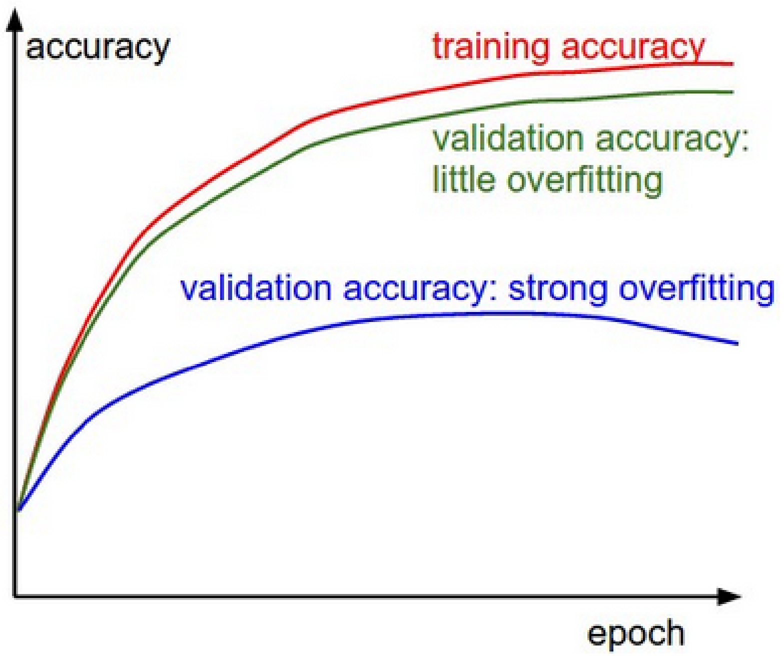


Figure 3.3: The gap between the training and the validation accuracy indicates the amount of overfitting. The above diagram shows two cases. The blue line is indicating a high amount of overfitting, regularization could help out in this case. The green line has probably much less overfitting, though it cannot keep up with training accuracy. The model capacity is not high enough in this case. By enlarging the number of features this model could possibly be improved[39].

The maximum number of iterations is set to 3000. This number was chosen by looking at the feedback on a first test run: 3000 iterations is still doable in a few hours on a CPU for the chosen networks and the network should have been converged for the largest part as the number of samples is relatively low.

The other parameters used are attached in appendix I.

3.4 Evaluation protocol

The evaluation protocol is identical to that in earlier research using this data set: "The training and validation set including non-logo images are indexed by the respective method. The whole test set including logo and logo-free images (3960 images) is then used to compute the classification scores" [28] The evaluation protocol includes calculating the accuracy, precision and recall. Caffe natively is designed towards large data sets and evaluates using “test iterations”, where several samples from the test set are taken and the average of these iterations is taken as the accuracy. Because this experiment uses a rather small data set, we will evaluate by computing the accuracy by using the full test data set.

Chapter 4

Results

How did the three networks perform on recognizing logos in the Flickr logo data set? In this chapter I will analyze the performance of the trained networks and review their training process. Furthermore, I will compare their performances with state-of-the-art results.

4.1 Accuracy

The figures 4.1-3 show a decreasing growth in accuracy as the number of iterations grows. The accuracy was measured at least every 500 iterations, where accuracy is defined as “The closeness of agreement between a test result and the accepted reference value.” [11] When comparing these figures to Figure 3.3, it seems plausible that at least AlexNet and CaffeNet have reached their maximum performance with this training set. Judging by Figure 4.3, Network-in-Network could benefit from more training iterations. This could either improve its performance by a small percentage or point out that this Network-in-Network configuration has reached its maximum performance on this data set.

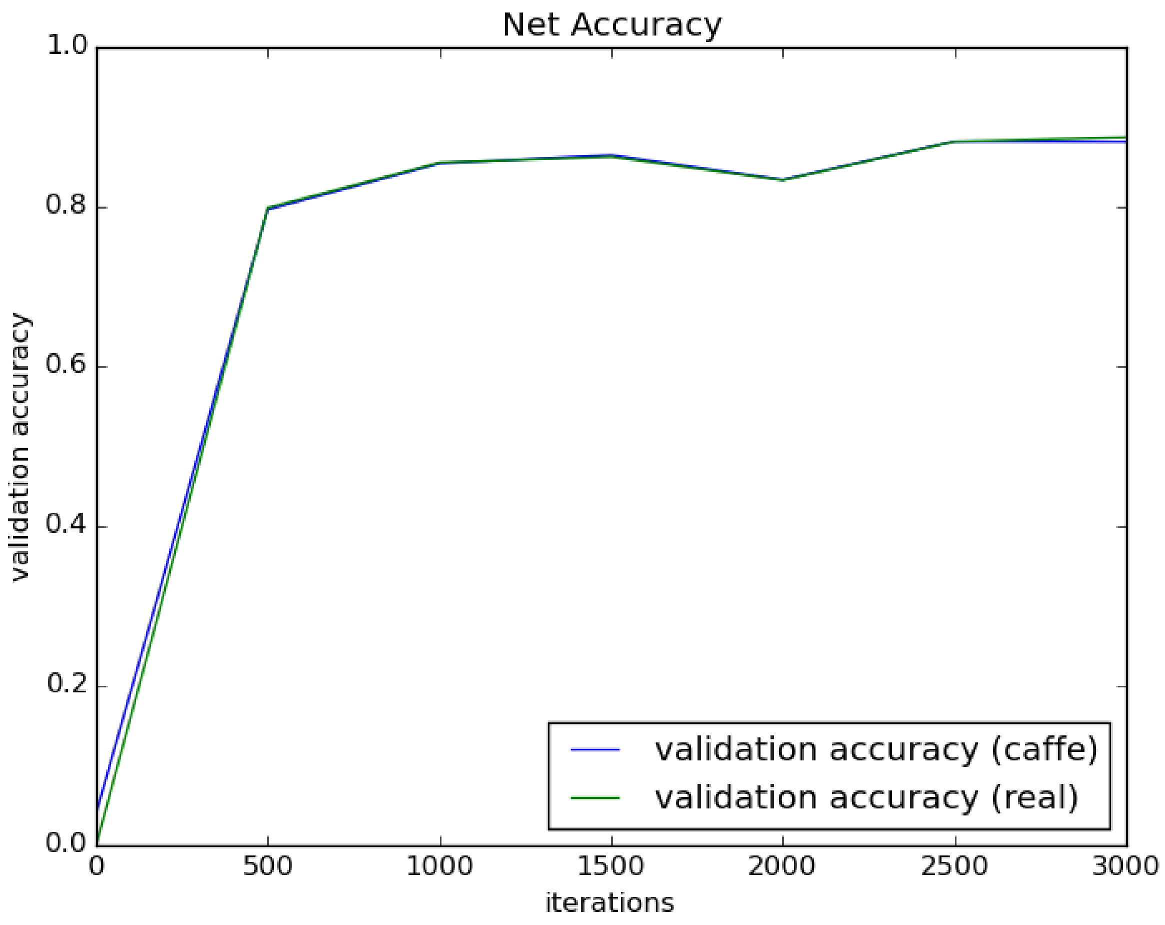


Figure 4.1: Accuracy as a function of the number of iterations, fine tuned AlexNet.

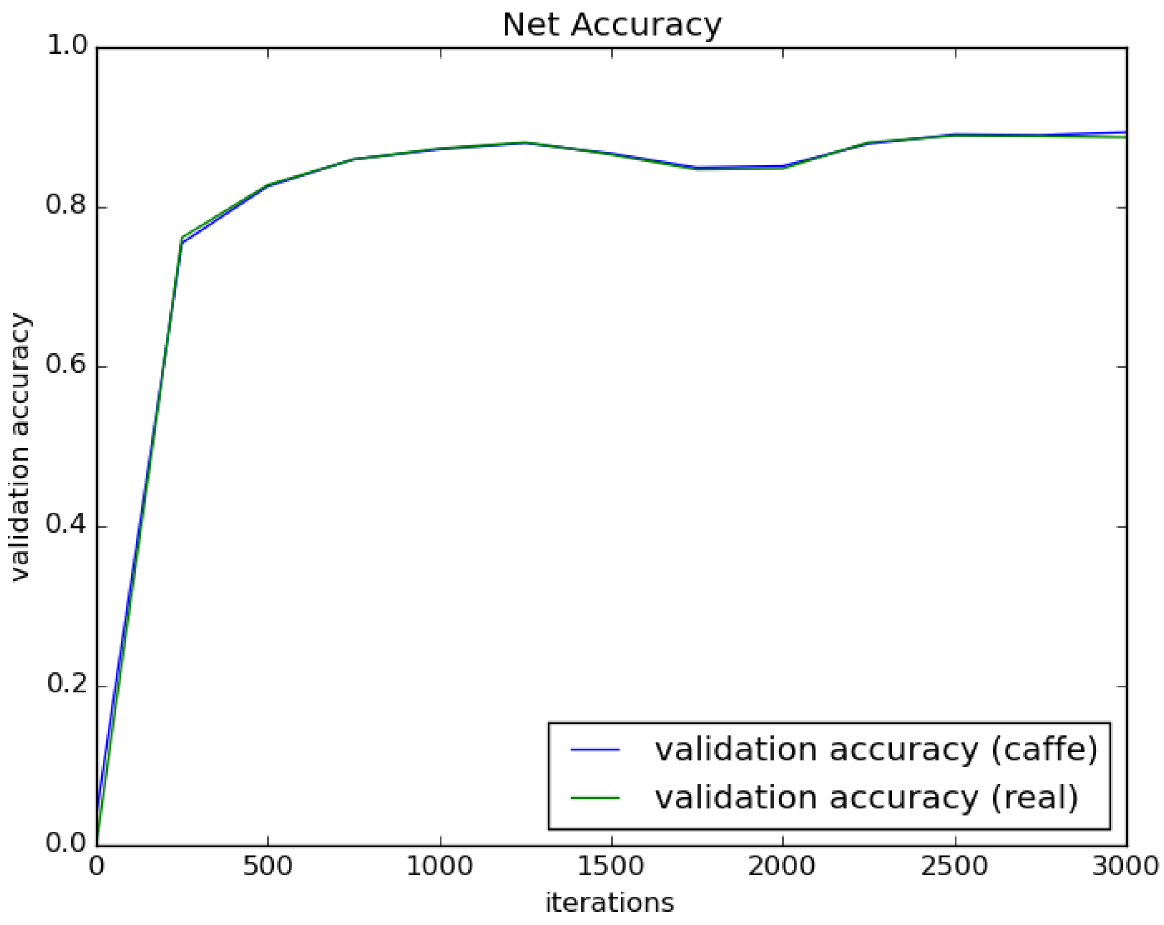


Figure 4.2: Accuracy as a function of the number of iterations, fine tuned CaffeNet.

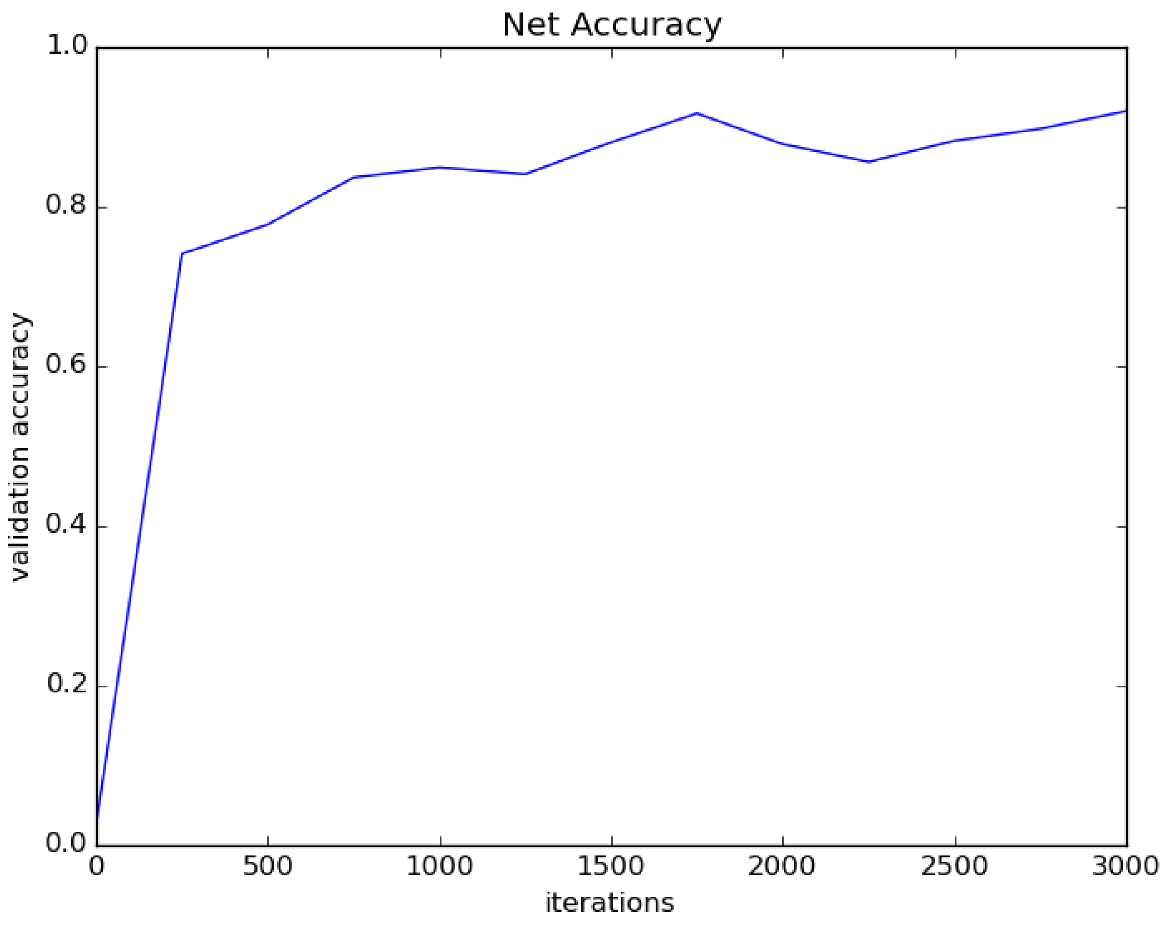


Figure 4.3: Accuracy as a function of the number of iterations, fine tuned Network-In-Network.

To compare the trained nets, a new net is introduced. This 'PredictNoLogoNet' simply predicts every image as the "no-logo" class. Due to the fact that the "no-logo" class is present in much larger quantities than every other class, this will result in an accuracy of 0.758. The final accuracy of every model is displayed in Table 4.1. The top-1 error rate for CaffeNet is 11.3%. It performs slightly better than Network-in-Network, which has a top-1-error rate of 11.7%.

Method	**Accuracy**
PredictNoLogoNet	0.758
AlexNet-logos-3000	0.877
CaffeNet-logos-3000	0.887
Network-In-Network-logos-3000	0.883

Table 4.1: The accuracy of different nets, where the PredictNoLogoNet is a virtual net classifying every image as no-logo.

4.2 Confusion matrices

A confusion matrix or an error matrix is a tool that allows visualization of the performance of an machine learning algorithm. All of the examples from the test set are displayed. One axis shows the true label of the test set while the other axis displays the predicted label of the test set. Obviously, many examples that have the same true class as predicted class will indicate that the model is recognizing the logos. Figures 4.4-6 show the confusion matrices for respectively AlexNet, CaffeNet and Network-in-Network. In ideal circumstances, the diagonal line from the left top to the right bottom corner would be colored dark red, while all other squares will be dark blue.

From Figure 4.4 and 4.5 we see that AlexNet and CaffeNet models predict more logo classes as the non-logo class, while Network-in-Network relatively predicts more from the non-logo class as logo classes. CaffeNet and Network-in-Network have almost the same accuracy. These confusion matrices implies that all nets have difficulties distinguishing similar types of logos, such as that from beer brands and gas companies. The highest numbers of examples found outside the diagonal (without looking at the no-logo class) are for example: Erdinger classified as Paulaner (both German beers), Heineken classified as Carlsberg (also both beer brands) and Esso classified as Texaco (gas companies). This can be explained by the original task the nets were trained on, namely recognizing objects. When a net for instance detects a green beer bottle, it is safe to say this will be a logo of a beer brand. Distinguishing between those brands is then harder: in the task of classifying objects, it is not important what is on the object. For example, when classifying a van, it is trained to detect the wheels, the shape of the cabin and the shape of the cargo space, but not the logo on the side. In the discussion chapter, I will elaborate on how these cases could be predicted better by combining model ensembles.

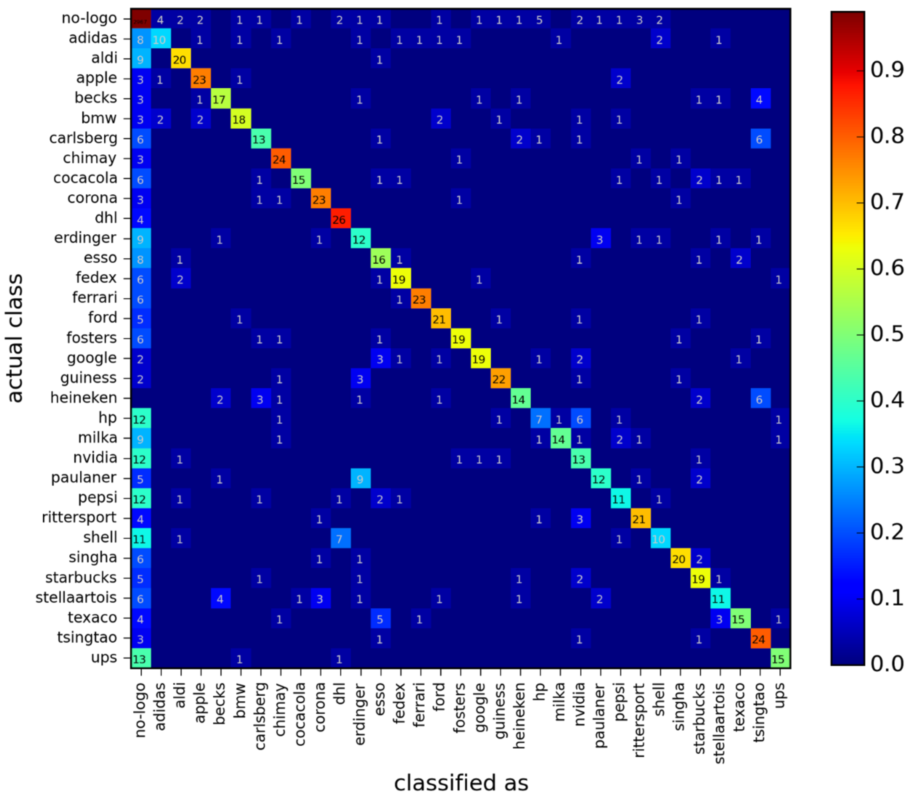


Figure 4.4: A confusion matrix for the fine tuned AlexNet with 3000 iterations. Each cell contains a the number of examples that had the class on the row they are in, but are classified as the column they are in.

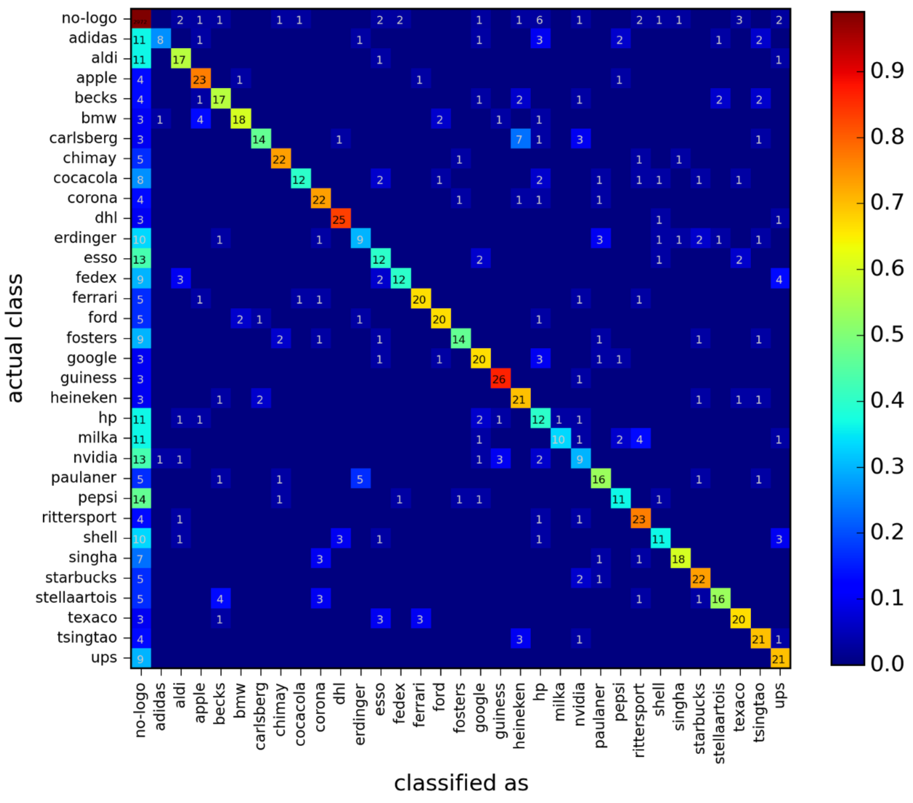


Figure 4.5: A confusion matrix for the fine tuned CaffeNet with 3000 iterations.

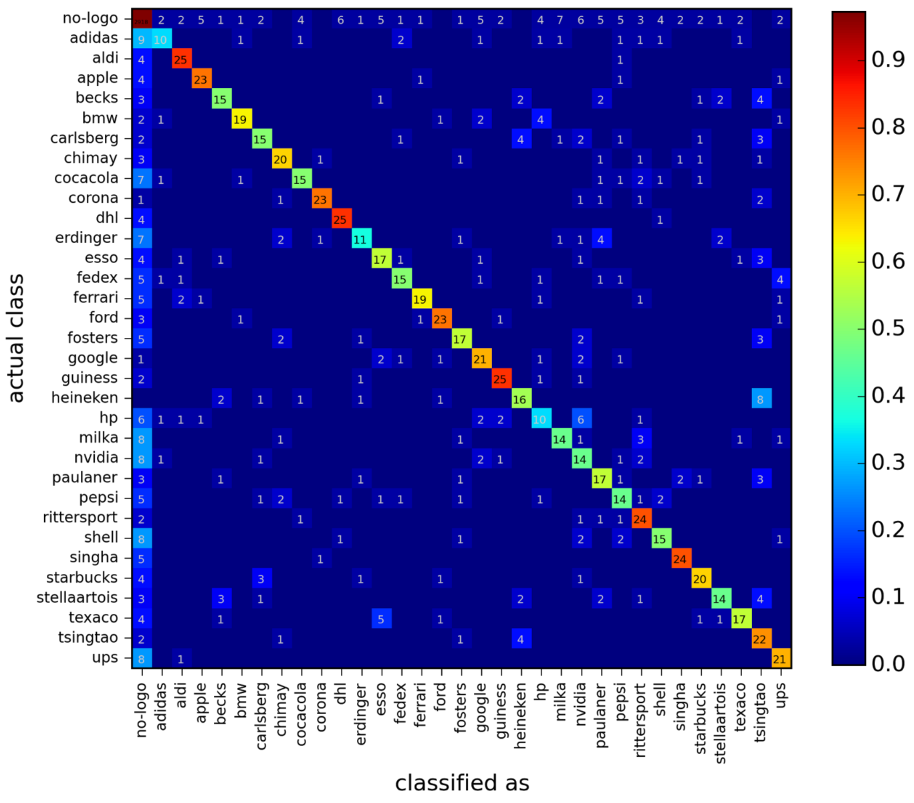


Figure 4.6: A confusion matrix for the fine tuned Network-In-Network with 3000 iterations.

4.3 Precision and Recall

The next measurements that could give insight in the performance of the networks are precision and recall. In information retrieval, these measurements provide information about the relevance of the (search) results. Recall, in this case, is the ratio of relevant logos retrieved to the total number of logos in the data set. Precision is the ratio of relevant samples (= logo classes) retrieved to the number of irrelevant samples (= no-logo class) retrieved. Note that the precision and recall are negatively correlated. Table 4.2 contains the results found in literature as well as for the model tested for this thesis.

The earlier observation, that Network-in-Network classifies relatively more examples as logos and AlexNet and CaffeNet classify more examples as no-logos is confirmed by their mutual precision and recall.[28] When comparing the results of the current best performing technique, the fine tuned networks can not compete with the state-of-the-art precision and recall. The discussion chapter describes improvements that will boost performance, some of which the method of Romberg et al. already uses, such as data augmentation and spatial re-ranking[28] that can be incorporated in a CNN[22].

Method	**Precision**	**Recall**
Romberg et al. (2011) [29]	0.98	0.61
Revaud et al. [26]	>= 0.98	0.73
Romberg et al. (2013) [28]	0.999	0.832
AlexNet-logos-3000	0.713	0.569
CaffeNet-logos-3000	0.729	0.565
Network-In-Network-logos-3000	0.705	0.604

Table 4.2: An overview of results on the "FlickrLogo-32"-dataset.

4.4 Detection and Recognition

The single-step and double-step approaches for logo recognition were described in the first chapter, where the decision was made to choose the single-step approach. Evaluation however can look at both steps, detection and recognition of logos, separately.

The detection task is evaluated by converting the task from multiple classes or multiclass to binary classes: logo and no-logo. This binary approach allows to calculate an detection accuracy, precision and recall, of which the results are shown in Table 4.3. The detection task is performed relatively well, with high overall accuracy and precision scores. The relatively lower recall scores indicate that not all logos are detected.

Method	**Accuracy**	**Precision**	**Recall**
PredictNoLogoNet	0.758	0.000	0.000
AlexNet-logos-3000	0.943	0.959	0.798
CaffeNet-logos-3000	0.938	0.964	0.774
Network-in-Network-logos-3000	0.948	0.920	0.861

Table 4.3: The results on detecting whether there is a logo in the image.

The recognition task is evaluation by removing the no-logo class. This can be thought of as removing the no-logo row and column from the confusion matrix that was shown earlier and requiring a logo class prediction for all logo classes. The recognition accuracy, precision and recall are calculated as the average of all classes. The results of these calculations can be found in Table 4.4. The overall values are much lower than those of the detection task. This could be attributable to that the recognition task receives the same input as the detection task, namely, the full image, instead of a crop of the detected logo.

Method	**Accuracy**	**Precision**	**Recall**
AlexNet-logos-3000	0.569	0.569	0.041
CaffeNet-logos-3000	0.565	0.565	0.040
Network-in-Network-logos-3000	0.616	0.616	0.049

Table 4.4: The results on recognition the logos. Logos that are free of images do not participate in this task.

4.5 Loss

The loss function was monitored during training and compared to Figure 3.2 to verify the learning rate. The comparison of this figure to the visualizations implies the learning rate somewhere between low and good. This parameter should be empirically determined, but this would not be computationally feasible and is therefore omitted. The loss was measured every 50 iterations. I reran the highest performing net, CaffeNet, measuring the loss every iteration with shuffled training data. In the graphs a unexpectedly low loss is seen every few iterations. The training set was not shuffled, resulting in some batches of non logos that are recognized fully.

The visualizations of the loss functions are attached in Appendix III.

Chapter 5

Discussion

In this chapter, I will discuss what optimizations could be done to improve the results of the current study and include directions for further research.

5.1 Methodology

There are two points within the methodology of this thesis that further research should be ruled out to influence the results:

- The repartitioning of the data set and especially using the same validation set as a test set was a quick decision to try to improve the accuracy, which it did. To make sure that the models learned to recognize the logo classes as expected, the models should be tested on a different test set or a different partitioning should be used.
- The loss graphs show an unusually low loss periodically as mentioned in the results chapter. The training data was taken from an ordered list, resulting in some batches without logos that is fully classified correctly. As the overall loss is decreasing, this indicates that the model is still improving. To rule out the order of the images provided has any effect on the results, the training data should be shuffled. I did this for CaffeNet, resulting in a faster converge speed, but no noticable difference in the final performance.

5.2 Optimization of accuracy

During designing the experiments, I have consistently chosen for the least computationally expensive method, as I am searching for a computationally feasible method of recognizing logos using deep learning. The performance of the system could be improved in several methods that I did not incorporate in this thesis. These are the following:

- As described earlier, the tasks of detection and recognition could be separated. The detected logos will have less noise, because it only contains the part of the image with the logo. This will probably increase the performance on the recognition task, increasing the total performance as well.
- By adding data augmentation, the training set size could be increased. Training on more training data could improve the net performance.
- Neural network ensembles are used to increase the performance of separate networks by combining them[9]. Combination can be done by voting rules, belief functions or statistical techniques and can outperform the single best network. They are only effective when nets make different errors. As AlexNet and CaffeNet make similar errors, as their architecture and training data is the same. To use these networks in ensembles, other independent nets should be trained. Earlier I stated that the nets are mostly trained on object classification and therefore repress information that is crucial for distinguishing between certain logo classes. Models that are added to the ensemble that have features that focus on this part of the recognition task would be able to distinguish in those cases, improving the overall accuracy.
- Other fine tuned nets use larger numbers iterations. For example, a fine tuned CaffeNet on the Oxford Flower Data set used 50,000 iterations, compared to the 3000 I perform[3]. A higher maximum of iterations allows for a smoother learning rate decay, hence probably increasing the performance.
- In the experiments, I have used models from 2012 and 2013. After this, other, more accurate networks were developed. By fine tuning VGG 16-layers or GoogLeNet for example, a higher accuracy could be achieved.
- The final optimization could be to train a net from scratch instead of fine tuning, or fine tuning a larger part of the network. This requires more computational power and a bigger data set, for example similar to ImageNet. The task the pre-trained network is trained on is object classification. This shows in the confusion matrix: similar objects are classified similarly. Heineken beers are classified as Paulaner, Shell as Esso and (square) NVIDIA products as Ritter Sport. Features that are now probably missing are the one that contain logo-specific information, rather than containing object information.

5.3 Optimization of training speed

Most optimizations on the area of training speed within reach are already chosen. I can think of others that will enhance the training speed.

- Increasing training speed will make it possible to test more hyperparameters and therefore improving final results:

- By implementing efficient storage database, instead of reading images directly from the file system, such as LMDB.
- Training on a GPU instead of a CPU will be relatively faster, as many optimization techniques are in place [44].

5.4 Directions for further research

Using, amongst others, the optimizations above, one could investigate whether CNNs could perform as well as other techniques such as Bag-of-Visual-Words on the task of recognizing logos. In particular, it could be interesting to study these techniques scale to for examples 1000 logo classes and which is the most efficient manner to add a class to the network.

Chapter 6

Conclusions

In this thesis I have summarized the current status of deep learning for logo recognition and demonstrated a computationally feasible prototype that recognizes logos in real world images.

References

[1] Christopher M Bishop et al. *Neural networks for pattern recognition.* Clarendon press Oxford, 1995.

[2] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer, 2012.

[3] Caffenet fine-tuned on the oxford 102 category flower dataset. `https://gist.github.com/jgoode21/0179e52305ca768a601f`. Accessed: 2015-06-10.

[4] Fine-tuning caffenet for style recognition on "flickr style" data. `http://caffe.berkeleyvision.org/gathered/examples/finetune_flickr_style.html`. Accessed: 2015-06-10.

[5] Francesca Cesarini, Enrico Francesconi, Marco Gori, Simone Marinai, JQ Sheng, and Giovanni Soda. A neural-based architecture for spot-noisy logo recognition. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 1, pages 175–179. IEEE, 1997.

[6] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *British Machine Vision Conference*, 2014.

[7] Dataset: Flickrlogos-32. `http://www.multimedia-computing.de/flickrlogos/`. Accessed: 2015-06-04.

[8] Enrico Francesconi, Paolo Frasconi, Marco Gori, Simone Marinai, JQ Sheng, Giovanni Soda, and Alessandro Sperduti. Logo recognition by recursive neural networks. In *Graphics Recognition Algorithms and Systems*, pages 104–117. Springer, 1998.

[9] Giorgio Giacinto and Fabio Roli. Design of effective neural network ensembles for image classification purposes. *Image and Vision Computing*, 19(9):699–707, 2001.

[10] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[11] 5725-1 ISO. Accuracy (trueness and precision) of measurement methods and results-part 1: General principles and definitions. geneva, switzerland. *International Organization for Standardization*, 1994.

[12] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[13] Alexis Joly and Olivier Buisson. Logo retrieval with a contrario visual query expansion. In *MM '09: Proceedings of the seventeen ACM international conference on Multimedia*, pages 581–584, 2009.

[14] Y. Kalantidis, LG. Pueyo, M. Trevisiol, R. van Zwol, and Y. Avrithis. Scalable triangulation-based logo recognition. In *in Proceedings of ACM International Conference on Multimedia Retrieval (ICMR 2011)*, Trento, Italy, April 2011.

[15] Hiroharu Kato and Tatsuya Harada. Image reconstruction from bag-of-visual-words. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 955–962. IEEE, 2014.

[16] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 1(4):7, 2009.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[18] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Zhe Li, Matthias Schulte-Austum, and Martin Neschen. Fast logo detection and recognition in document images. In *Proceedings of the 2010 20th International Conference on Pattern Recognition*, pages 2716–2719. IEEE Computer Society, 2010.

[20] libccv – a modern computer vision library. `http://libccv.org/`. Accessed: 2015-05-27.

[21] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2013.

[22] Chu Kiong Loo, Yap Keem Siah, Kevin Kok Wai Wong, Andrew Teoh Beng Jin, and Kaizhu Huang. *Neural Information Processing: 21st International Conference, ICONIP 2014, Kuching, Malaysia, November 3-6, 2014. Proceedings*, volume 8835. Springer, 2014.

[23] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[24] Chun Pan, Zhiguo Yan, Xiaoming Xu, Mingxia Sun, Jie Shao, and Di Wu. Vehicle logo recognition based on deep learning architecture in video surveillance for intelligent traffic system. *Smart and Sustainable City 2013 (ICSSC 2013)*, 2013.

[25] Apostolos P Psyllos, C-NE Anagnostopoulos, and Eleftherios Kayafas. Vehicle logo recognition using a sift-based enhanced matching scheme. *Intelligent Transportation Systems, IEEE Transactions on*, 11(2):322–328, 2010.

[26] Jerome Revaud, Matthijs Douze, and Cordelia Schmid. Correlation-based burstiness for logo retrieval. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 965–968. ACM, 2012.

[27] Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 1996.

[28] Stefan Romberg and Rainer Lienhart. Bundle min-hashing for logo recognition. In *Proceedings of the 3rd ACM International Conference on Multimedia Retrieval (ICMR)*, ICMR '13, pages 113–120, New York, NY, USA, 2013. ACM.

[29] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof Van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 25. ACM, 2011.

[30] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, ICMR '11, pages 25:1–25:8, New York, NY, USA, 2011. ACM.

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 2015.

[32] Stuart Russell and Peter Norvig. *Artificial Inteligence: A Modern Approach.* Pearson, 2010.

[33] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[34] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR 2014)*. CBLS, April 2014.

[35] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[36] Convolutional neural network. `http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/`. Accessed: 2015-06-01.

[37] Cs231n convolutional neural networks for visual recognition – convolutional neural networks (cnns / convnets). `http://cs231n.github.io/convolutional-networks/`. Accessed: 2015-05-31.

[38] Cs231n convolutional neural networks for visual recognition – transfer learning. `http://cs231n.github.io/transfer-learning/`. Accessed: 2015-05-28.

[39] Cs231n convolutional neural networks for visual recognition – learning. `http://cs231n.github.io/neural-networks-3/`. Accessed: 2015-06-10.

[40] Feature extraction using convolution. `http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/`. Accessed: 2015-06-03.

[41] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th international conference on machine learning (ICML-13)*, pages 1139–1147, 2013.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842*, 2014.

[43] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[44] Nicolas Vasilache, Jeff Johnson, Michael Mathieu, Soumith Chintala, Serkan Piantino, and Yann LeCun. Fast convolutional nets with fbfft: A gpu performance evaluation. *arXiv preprint arXiv:1412.7580*, 2014.

[45] A. Vedaldi and K. Lenc. Matconvnet – convolutional neural networks for matlab. *CoRR*, abs/1412.4564, 2014.

[46] Xiaogang Wang. Multilayer neural networks. `http://vision.sysu.edu.cn/vision_sysu/wp-content/uploads/2015/03/nn.pdf`, 2015. Accessed: 2015-06-03.

[47] Cs-449: Neural networks – error backpropagation. `http://www.willamette.edu/~gorr/classes/cs449/backprop.html`. Accessed: 2015-05-31.

[48] Cs-449: Neural networks – momentum and learning rate adaption. `http://www.willamette.edu/~gorr/classes/cs449/momrate.html`. Accessed: 2015-05-31.

[49] Cs-449: Neural networks – overfitting. `http://www.willamette.edu/~gorr/classes/cs449/overfitting.html`. Accessed: 2015-05-31.

[50] Kathleen Zyga, Richard Price, and Brenton Williams. A generalized regression neural network for logo recognition. In *Knowledge-Based Intelligent Engineering Systems and Allied Technologies, 2000. Proceedings. Fourth International Conference on*, volume 2, pages 475–478. IEEE, 2000.

Appendix A

Appendix

A.1 Appendix I

Extract of the solver files for CaffeNet, AlexNet and Network-in-Network for fine tuning for logos[1].

```
test_iter: 80
test_interval: 250
base_lr: 0.0001
lr_policy: "step"
gamma: 0.1
stepsize: 20000
max_iter: 3000
momentum: 0.9
weight_decay: 0.0005
solver_mode: CPU
```

[1] More information can be found here: http://caffe.berkeleyvision.org/tutorial/solver.html

A.2 Appendix II

Visualizations of the neural nets discussed in the literature review: LeNet-5, AlexNet, Network-in-Network, GoogLeNet and Very Deep 16 layers.

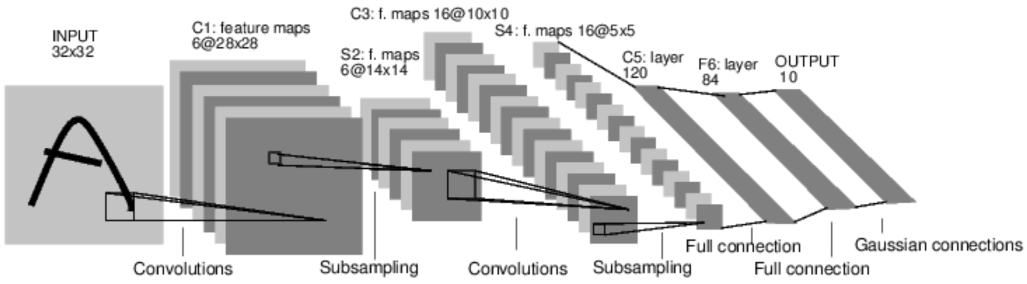


Figure A.1: Visualization of in LeNet-5 [18].

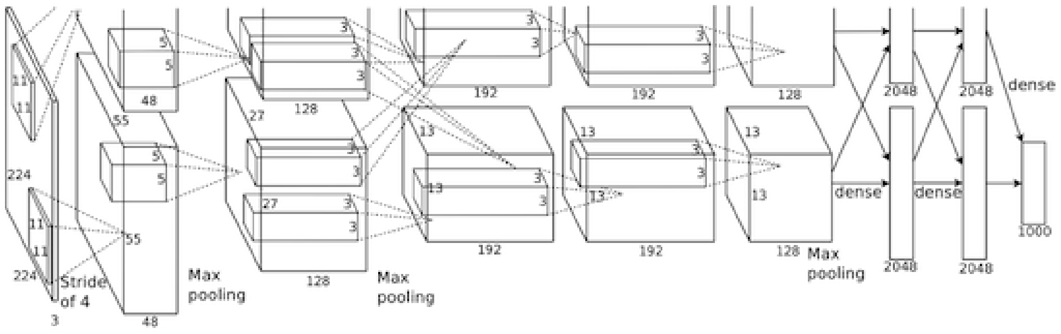


Figure A.2: Visualization of AlexNet [17].

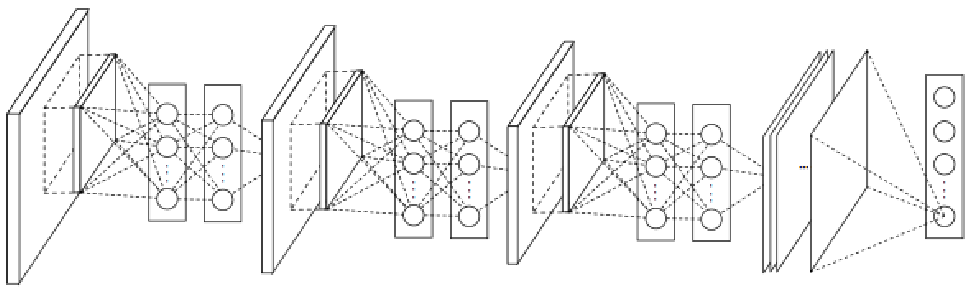


Figure A.3: Visualization of Network-in-Network[21].

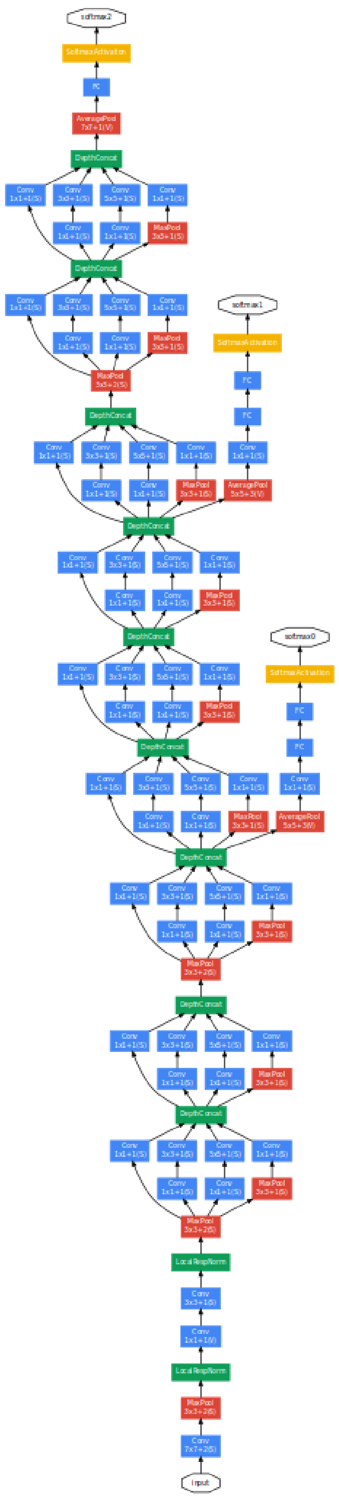


Figure A.4: Visualization of GoogLeNet[42].

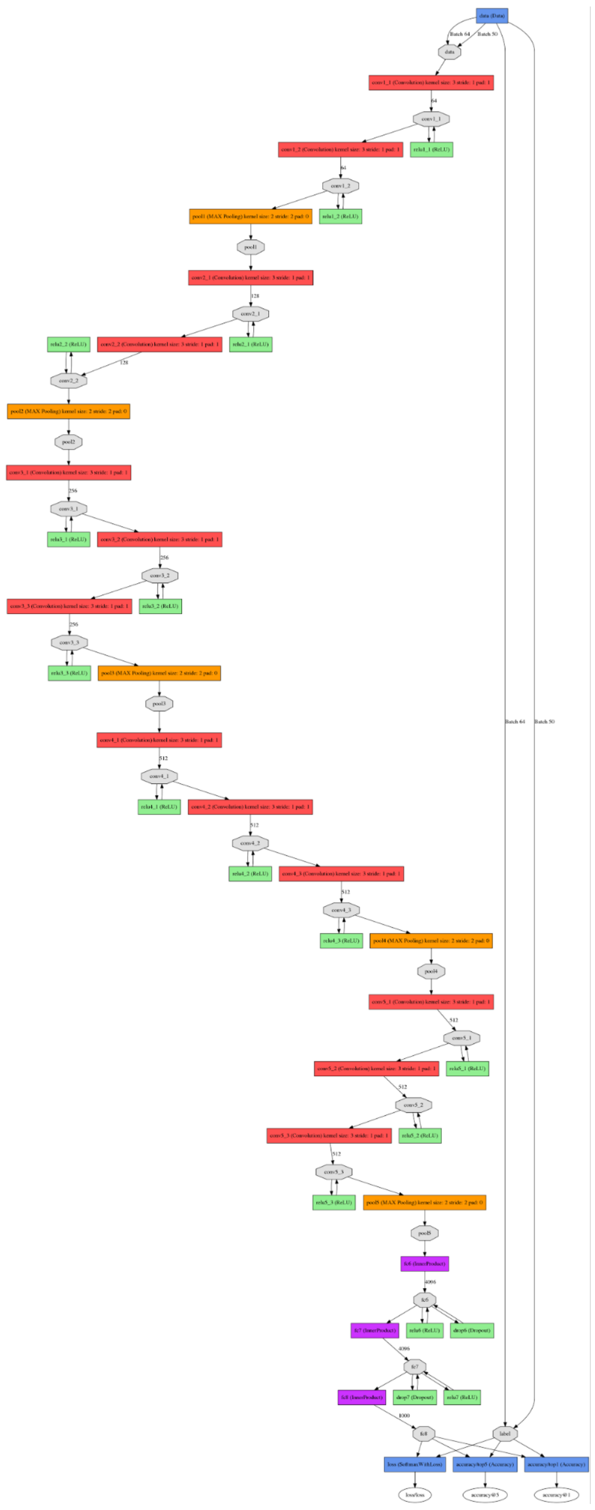

data (Data)
Batch 64
Batch 50
data
conv1_1 (Convolution) kernel size: 3 stride: 1 pad: 1
64
conv1_1
conv1_2 (Convolution) kernel size: 3 stride: 1 pad: 1
relu1_1 (ReLU)
64
conv1_2
pool1 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
relu1_2 (ReLU)
pool1
conv2_1 (Convolution) kernel size: 3 stride: 1 pad: 1
128
conv2_1
relu2_2 (ReLU)
conv2_2 (Convolution) kernel size: 3 stride: 1 pad: 1
relu2_1 (ReLU)
128
conv2_2
pool2 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool2
conv3_1 (Convolution) kernel size: 3 stride: 1 pad: 1
256
conv3_1
relu3_1 (ReLU)
conv3_2 (Convolution) kernel size: 3 stride: 1 pad: 1
256
conv3_2
conv3_3 (Convolution) kernel size: 3 stride: 1 pad: 1
relu3_2 (ReLU)
256
conv3_3
relu3_3 (ReLU)
pool3 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool3
conv4_1 (Convolution) kernel size: 3 stride: 1 pad: 1
512
conv4_1
relu4_1 (ReLU)
conv4_2 (Convolution) kernel size: 3 stride: 1 pad: 1
512
conv4_2
relu4_2 (ReLU)
conv4_3 (Convolution) kernel size: 3 stride: 1 pad: 1
512
conv4_3
relu4_3 (ReLU)
pool4 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool4
conv5_1 (Convolution) kernel size: 3 stride: 1 pad: 1
512
conv5_1
conv5_2 (Convolution) kernel size: 3 stride: 1 pad: 1
relu5_1 (ReLU)
512
conv5_2
conv5_3 (Convolution) kernel size: 3 stride: 1 pad: 1
relu5_2 (ReLU)
512
conv5_3
relu5_3 (ReLU)
pool5 (MAX Pooling) kernel size: 2 stride: 2 pad: 0
pool5
fc6 (InnerProduct)
4096
fc6
fc7 (InnerProduct)
relu6 (ReLU)
drop6 (Dropout)
4096
fc7
fc8 (InnerProduct)
drop7 (Dropout)
relu7 (ReLU)
1000
fc8
label
loss (SoftmaxWithLoss)
accuracy/top5 (Accuracy)
accuracy/top1 (Accuracy)
loss/loss
accuracy@5
accuracy@1

Figure A.5: Visualization of Very Deep 16 layers.

A.3 Appendix III

Visualizations of the loss function as training takes place.

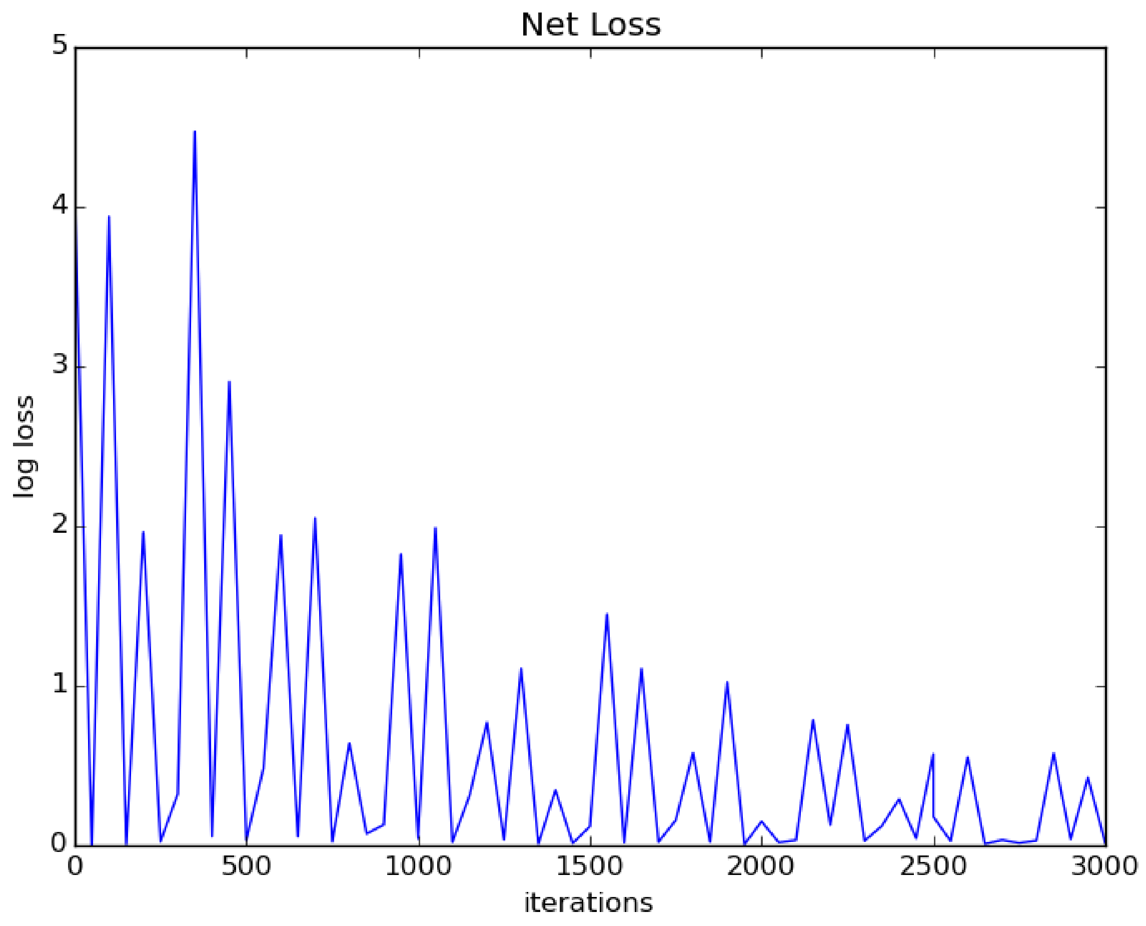


Figure A.6: Loss (or error) versus number of iterations AlexNet, measured every 50 iterations. The average loss in a certain interval decreases as the number of iterations increases.

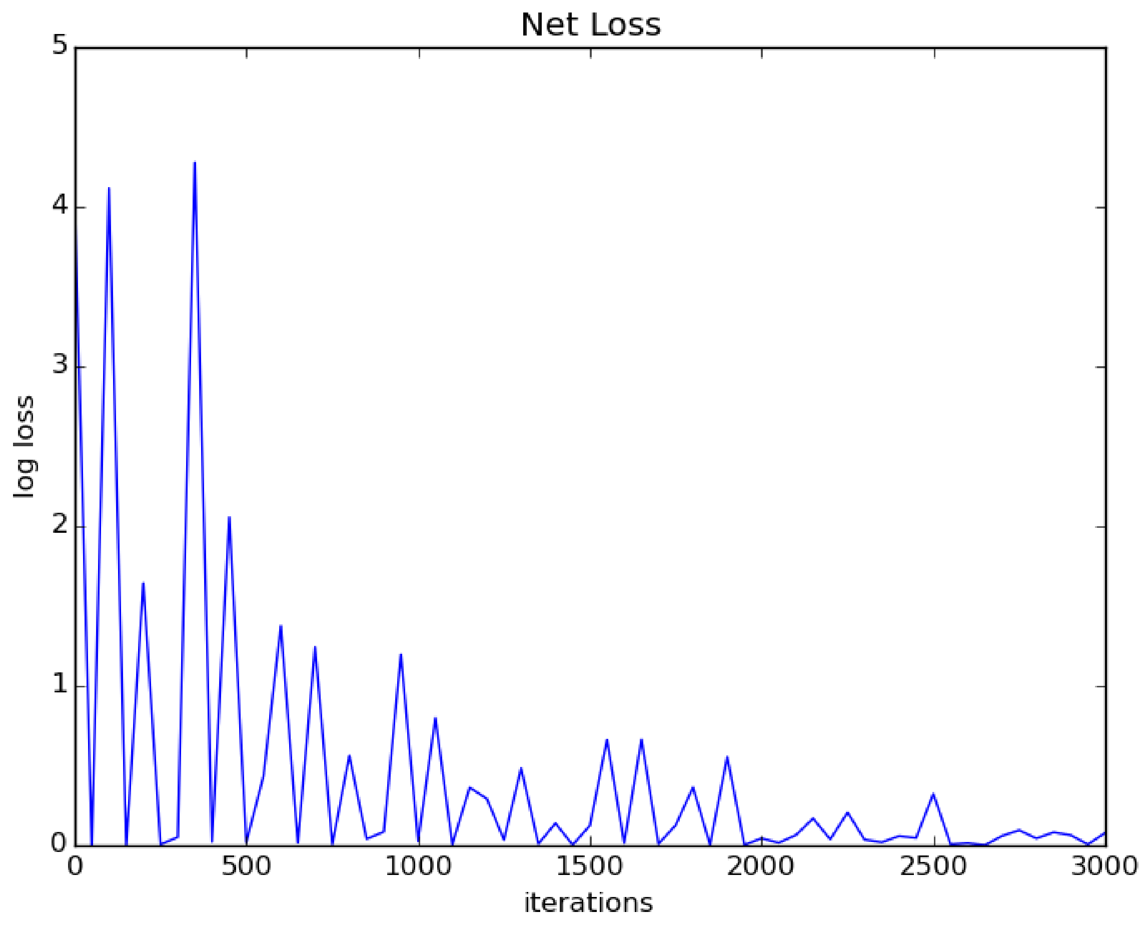


Figure A.7: Loss (or error) versus number of iterations CaffeNet, measured every 50 iterations.

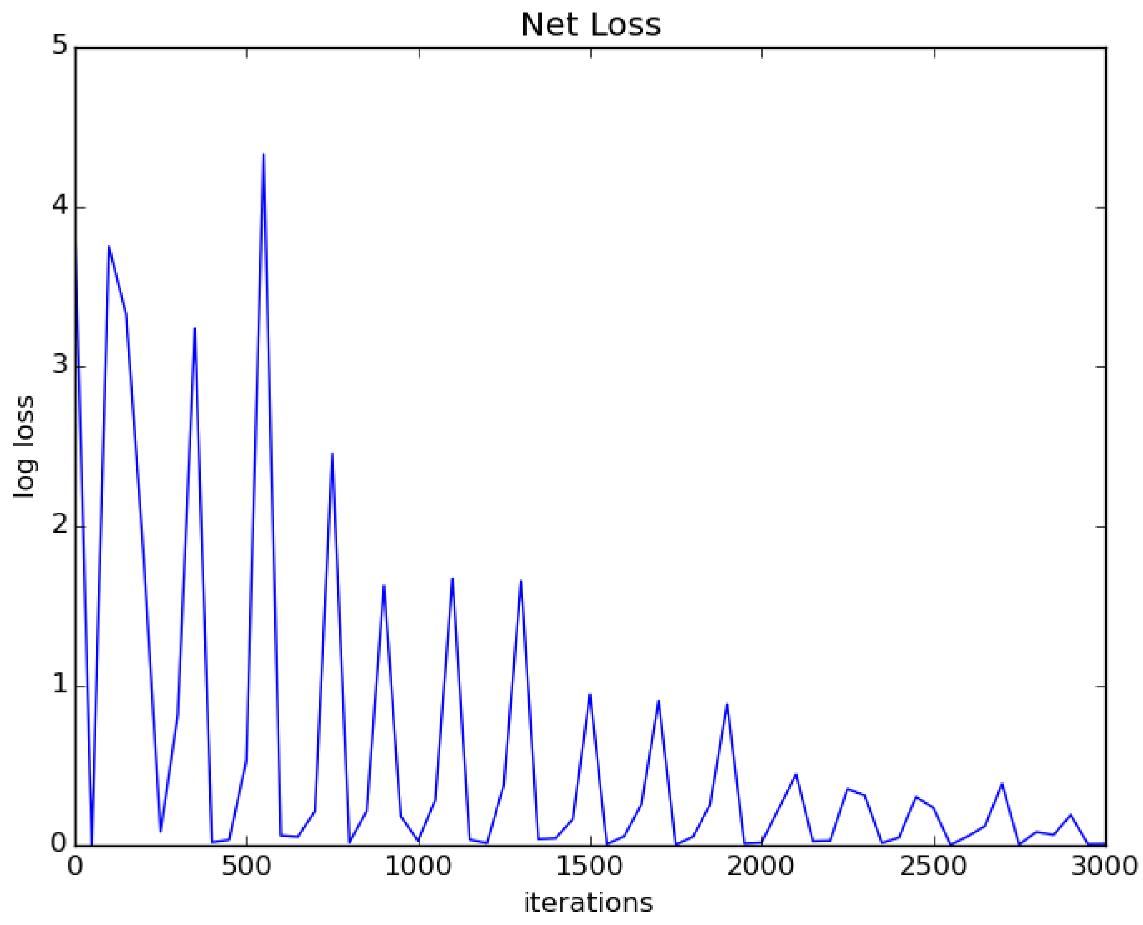


Figure A.8: Loss (or error) versus number of iterations Network-in-Network, measured every 50 iterations.

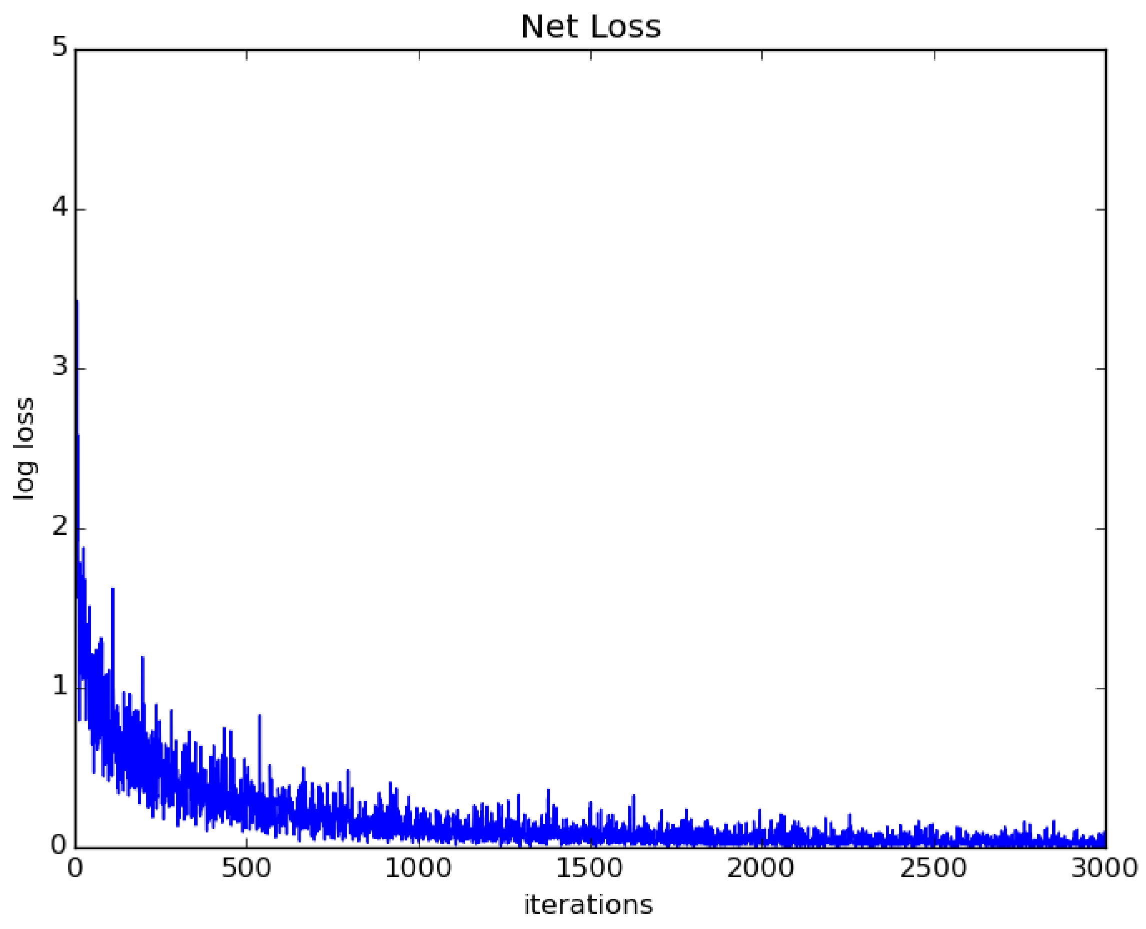


Figure A.9: Loss (or error) versus number of iterations CaffeNet, measured every iteration.