Synthesizing Images to Recognize Natural Images with Transfer Learning in Convolutional Neural Networks

Master's Thesis

Ernest Bofill Ylla

Aalborg University Copenhagen
MSc Medialogy

Department of Architecture, Design, and Media Technology
Aalborg University Copenhagen
www.en.cph.aau.dk

AALBORG UNIVERSITY
STUDENT REPORT

Title:
Synthesizing Images to Recognize Natural Images with Transfer Learning in Convolutional Neural Networks

Theme:
Scientific Theme

Project Period:
Fall Semester 2016

Participant(s):
Ernest Bofill Ylla

Supervisor(s):
Hendrik Purwins

Copies: 3

Page Numbers: 51

Date of Completion:
January 26, 2017

Abstract:

Convolutional neural networks are the state of the art in computer vision tasks thanks to breakthrough architecture innovations in the past few years such as the "Inception" architecture. Large datasets of annotated images, necessary to train CNNs are scarce. 3D models can be used to generate synthetic datasets of rendered images in a fast and automated way.
This thesis investigates how amplifying a small dataset of natural images with a much larger dataset of rendered images improves the classification accuracy of an Inception-V3 CNN retrained with transfer learning. Two image datasets of Lego bricks are generated for the experiment: a large synthetic dataset generated from a Lego 3D model and a small dataset of photos.
Results show that the amplified dataset produces a worse classification accuracy compared to no augmentation by 82% versus 68% after the augmentation. This observation cannot be extrapolated due to differences found between the natural and synthetic data that might have affected the recognisably. Despite of that, synthetic datasets still have a lot of potential in situations where image datasets are difficult to obtain. Further research should investigate how improvements in the rendering process influence image recognition.

Contents

Chapter 1

Introduction

Introduction to Domain and Concepts Artificial neural networks are a method of machine learning. A Convolutional Neural Network (CNN) is a specific type of artificial neural network where the activation of the neuron corresponds to a convolution operation [18].

In the field of Computer vision, image classification[1] is a computer vision task to recognise the object in an image from a list of object classes [16]. Image classification can be implemented with CNN, in fact, CNNs are the state of the art in many computer vision tasks including image classification [6, 23]. In the recent years, CNNs have been subject to significant architectural improvements and task-specific optimisations such as the GoogLeNet architecture or Inception-V3 [22, 23]. Figure 3.1.3 illustrates an example of image recognition and localisation using a CNN.

A CNN needs to be trained prior to performing a task. Training a CNN for image recognition requires a dataset of annotated images which needs to be large to prevent overfitting. A common training dataset is ImageNet which has 12 millions of images with 1000 classes, ImageNet is also the dataset used in ImageNet Large Scale Visual Recognition Challenge (ILSVRC) which is a reference in image recognition performance.

Transfer learning is a machine learning technique to apply knowledge learnt for a specific task and apply it to a different but related task. Transfer learning is used in image recognition tasks to reduce the required amount of training data by leveraging previously trained CNNs. The optimal size of the training dataset for CNN transfer learning depends on several factors, which are explained in section 3.1.3 but for a modern CNN implementation such as Inception-V3, a good approximation is one thousand images per class [15], whereas a regular training is optimal with a dataset size in the order of millions.

[1] Also referred to as Image recognition, where image recognition is more generic and may include other kinds of computer vision tasks such as detection [2] (RCNN).

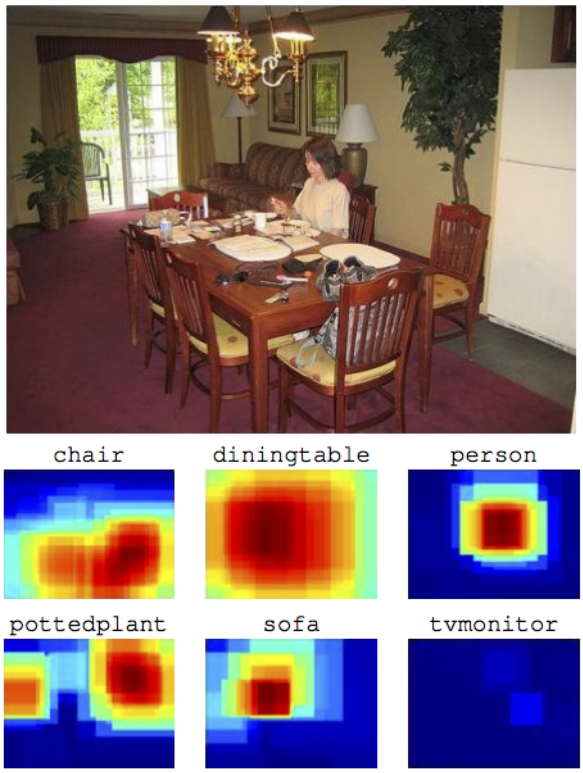


Figure 1.1: Copied from [12]: "Recognition and localization results of our method for a Pascal VOC test image. Output maps are shown for six object categories with the highest responses."

Introduction to Dataset Availability Limitation There is a limited availability of images datasets [12] and generating new ones is time consuming and can be economically expensive. Ignoring for now synthetic datasets – which are explained in chapter 2 – generating a dataset equals to photographing or scanning one or many objects or object categories (e.g. cars) with a large number of repetitions, where each repetition has some variation, for example different cars, from different angles, or a character handwritten by different people. Another possibility is to reuse existing images from, for example, an image search engine; in which case there is less control over the quality, the representativeness, and the proper annotation of the data without a thorough review. ImageNet uses flickr together with other search engines.

Sometimes, even when allocating sufficient time and economic resources, it may not be possible to generate a dataset because the object or objects to capture no longer exist or are not available. For example, a running criminal (from whom only a few or no images are known) will not willingly go to a photography studio to take photos for a dataset of his face.

Introduction to Solution, Value, and Application Examples Synthetic datasets can be generated from 3D models by means of rendering several 2D images with small variations of the 3D model in terms of, for example, position, lighting, background, among others. This method can drastically reduce the costs of generating datasets and even offer a solution to situations where generating a dataset was deemed impossible due to the lack of data.

In the same way, a small photographic dataset can be augmented with images synthetically generated from 3D models of the objects in the dataset. This augmentation, can provide the necessary volume of images to work with CNNs in optimal conditions.

Creating 3D models requires skills (e.g. 3D artists) and equipment (such a 3D scanners [4]) therefore it can be expensive and time consuming. Nevertheless, low-cost techniques are becoming more common, for example using a kinect camera [5].

After obtaining the 3D model, the rest of the process to generate a synthetic dataset consists of rendering the 3D model into 2D images several times with small variations from one rendered image to the next. This process can be automated and the type and degree of variation can be configured. Re-usability is a very important factor, which means that once effort has been put upon implementing an automated synthetic dataset generation process, reusing it with another 3D model comes at a very small effort.

These facts make the technique especially efficient at generating synthetic datasets from 3D models that already exist. For example, a furniture company that has digitised its catalogue into 3D models of the furniture and furniture parts, could implement machine learning applications at a relatively low cost. Potential applications include “augmented reality interactive assembly guides”, “industrial assembly line manufacturing”, “teaching musical instruments”, or “medical interventions”.

A major uncertainty regarding the use of synthetic datasets for training or applying transfer learning to a CNN is whether the CNN can recognise natural images from knowledge learnt from synthetic images.

Introduction to Research This thesis investigates how re-training a CNN with a relatively small photographic dataset by means of transfer learning compares in recognition accuracy to re-training the same CNN with the only difference of using a dataset amplified with synthetically rendered images.

There are two main steps required to carry on this research. First, generating a synthetic dataset for training, and a small photographic dataset for testing. The second step consists of applying transfer learning to a CNN with the synthetic dataset and perform an image classification test with the photographic dataset.

1.1 Structure of the Thesis

The next two chapters “2 - Generating Synthetic Datasets” and “3 - Image Classification with CNN and Transfer Learning” explores these two main topics of the thesis form a theoretical perspective. Especial emphasis is put on the transfer learning technique.

Chapter “4 - Implementation and Experiment” explains in detail the implementation work, design and execution of the experiment. It is based on theory explained in chapters 2 and 3. The chapter is divided into two sections: “4.1 - Generating the Lego dataset” and “4.2 - Image Classification Experiment”.

More detailed, in “4.1 - Generating the Lego dataset” I present the framework and the tool to generate synthetic datasets and its application to generate the Lego dataset from a 3D model. “4.2 - Image Classification Experiment” goes through from the initial steps of becoming acquainted with CNN and image recognition, the implementation with the machine learning framework TensorFlow, and the design of the experiment.

Chapter “5 - Results and Discussion” presents and analyses the results. And finally in chapter “6 - Conclusion and Future Work” I draw conclusions and suggest areas for further research that could be done in this subject.

Chapter 2

Generating Synthetic Datasets

2.1 Related Work

As explained in chapter 1, a fundamental part of this thesis is to use a 3D model to generate an annotated dataset of rendered 2D images similar to figure 2.1. This section is a review of several academic research projects that generate synthetic datasets for a similar machine learning purpose. The table in Figure 2.2 compares the method and randomisation parameters used by other authors in similar projects.

My project has many similarities with [15]. They generate a dataset of rendered 2D images with orientation labels. The original size of the dataset is 80K rendered images which they further augmentate to 2 million images in order to retrain more network layers. This information helped decide the dataset size for this project. The goal of their experiment is to train a CNN model to estimate the orientation of the object (in this case chairs).

I use some of the procedures described in [13] to develop my theoretical framework for rendering the 2D images. They generate a dataset from rendered 3D models of cars and planes randomising: position, pose, lighting, and background (which they refer to as "real-world variation"). The dataset is used to compare the classification performance of a simple V1-like image recognition model between the synthetic dataset versus a "natural" dataset. In my case, the machine learning unit (consisting of a CNN) is more sophisticated and should have a much better prediction power.

[24] creates a dataset of synthetic face images by rendering 3D face models and presents a Support Vector Machine (SVM) face component-based face recognition system. The SVM is trained with the synthetic images and tested with real photographic images. The experiment in this thesis is similar in the sense that it also uses synthetic images for training and photographic images for testing, but I use a CNN instead of SVM, and the training dataset will be a synthetically augmented

dataset instead of a purely synthetic one.

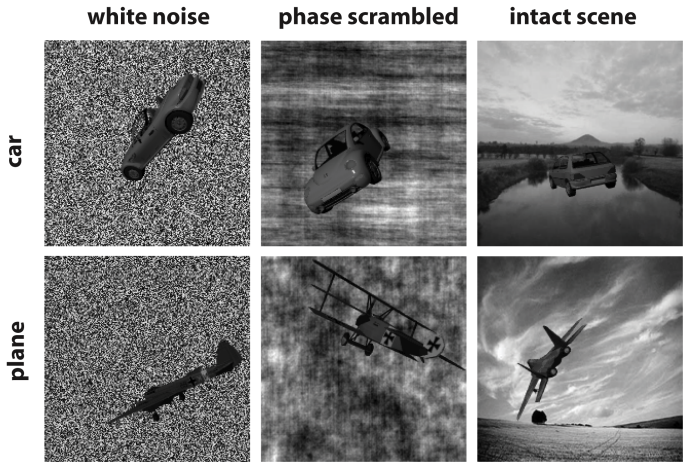


Figure 2.1: Copied from [13]: "3-D models rendered onto three types of backgrounds—white noise, phase-scrambled scene images (noise), and intact scene images."

2.2 Theoretical Framework for Synthetic Dataset Generation

As mentioned in the introduction, the image dataset is generated to apply transfer learning on a CNN model by re-training part of the neuron layers. I need to know how many images are needed, which parameters should be randomised, and what image resolution is best. This section explains the reasoning and references on deciding how to generate such dataset.

Size of the Dataset From [15] we know that a dataset size of 80K images would allow me to retrain 2 layers on the RCNN model from Caffe Model Zoo. A size of 2 Millions is enough to train the entire model from scratch.

[3] applies knowledge transfer on a LeNet5 CNN architecture [8]. They use 400 samples for each 20 classes for the initial training of the model. For the knowledge transfer they ran 5 trials "with training sets of 1, 5, 10, 20 and 40 samples/class". The dataset in this case consists of 62 classes of handwritten characters corresponding to '0'-'9', 'A'-'Z' and 'a'-'z'. They extract two subsets of 20: one for the initial training and another for knowledge transfer.

The dataset sizes of these two references are very different. Looking at the nature of the images, this thesis is more similar to [15] which uses images of chair in different orientations. In addition [3] explains that they intentionally use a smaller training dataset than usual. Therefore a size of 80K images is a good reference value for my experiment.

Randomisation parameters [13] defines real-world variation as variation in position, pose (or orientation), lighting, and background. Based on that definition, my dataset generation method will apply variation in those same attributes.

Image Resolution From the literature (summarised in Figure 3.13) the image resolution (in pixels) used to train the CNN, varies from 28x28 up to 299x299. Some experiments measured how the classification performance falls with a reduction of the initial resolution. [23], for instance, shows how a resolution of 299x299 obtains a classification accuracy of 76,7% while 151x151 reaches 76,4% and 79x79 a 75,2% using inception-v3 as the CNN model.

In this thesis research I will use an inception-like architecture, possibly "inception-bn", "googlenet" or "inception-v3". We know from the literature that experiments using these networks often choose a resolution of 224x224 or 299x299 for the case of [23].

Considering all the above, the dataset will be generated on a resolution of 448x448. This value is larger that any of the values used in previous research therefore I can scale the images down to match any those resolutions if needed.

Reference	Dataset name	Dataset type	Image resolution	Color	Light randomisation	Background	Orientation randomisation	Position randomisation	Scale randomisation
[9]	NORB with transformations	Photographic on a studio	96x96 (originally 640x480)	No	Yes	Highly cluttered images + distracting objects	Yes	Yes	Yes
[13]	Their own	Synthetic	150x150	No	Yes	Scene, noise or phase scrambled	Yes	Yes	Yes
[15]	Their own based on ShapeNet 3D models	Synthetic + Natural scene images for testing	Up to 255x255	No	Yes	Synthetic clutter background	Yes	No	No
Me	**Lego**	**Synthetic + fotographic for test**	**448x448**	**Yes**	**Yes**	**Natural scenes**	**Yes**	**Yes**	**Yes**

Figure 2.2: Dataset generation parameters comparison from references [9, 13, 15] and my own implementation.

Chapter 3

Image Classification with CNN and Transfer Learning

3.1 Introduction to CNN and Transfer Learning

As seen in chapter 1, image recognition can be implemented with CNNs. Thanks to important architecture optimizations introduced in the last few years, CNNs are today the state of the art in image classification tasks [22]. In this thesis I use a CNN to perform the image classification task. The way I use the CNN is different from a regular setting in two ways: 1) Transfer learning is used to partially retrain the network[1]. 2) The training dataset consists of synthetic images.

3.1.1 Introduction to Neural Networks

An Artificial Neural Network (ANN) is typically defined by three types of parameters:

- The interconnection pattern between the different layers of neurons.
- The learning process for updating the weights of the interconnections.
- The activation function that converts a neuron's weighted input to its output activation.

Several connection patterns are possible but a neural network is commonly represented as a forward fully connected network where each neuron in a layer is connected to all the neurons in the following layer, as shown in figure 3.1. The connections between neurons are weighted, the learning process of the network modifies the weights by iteratively feeding the training data into the network. Each

[1] Along the thesis, I often refer to the CNN as network to avoid filling up the text with the acronym CNN but unless otherwise specified, the network is always a convolutional, neural one.

training operation consists of calculating the output error and slightly adjusting the weights to reduce the error in the next iteration. Each element in the training dataset can be used several times for training to achieve convergence. Finally, the activation function[2] is a predefined function such as the "hyperbolic tangent", "sigmoid" or the "The Rectified Linear Unit" (ReLU) [19, 11].

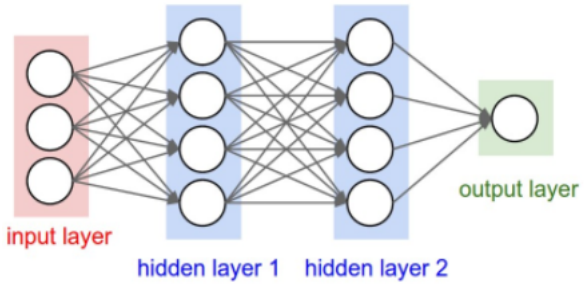


Figure 3.1: Copied from [19]: "A 3-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer. Notice that in both cases there are connections (synapses) between neurons across layers, but not within a layer."

3.1.2 Introduction to Convolutional Neural Networks

In image processing, a convolution operation consists of an input image processed with a convolution matrix that generates an output image. In this operation, each pixel of the input image plus its N surrounding pixels are multiplied with the convolution matrix to create a resulting convoluted image where result of the matrix multiplication determines the value of one pixel in the output image; figures 3.2 and 3.3. The value of N depends on the size of the convolution matrix, for example on 3x3 convolution $N = 1$ [17, 14].

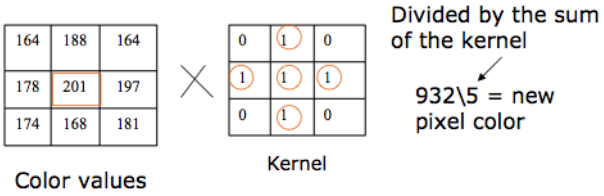


Figure 3.2: Calculating one pixel in a convolutional operation

A convolutional neural network is a type of neural network where the activation function in some of the neuron layers is a convolution: the "convolutional layers". A convolutional layer has several convolution matrices (also known as filters or kernels) [21]. Figure 3.4 is an example of a CNN layer.

[2]http://www.cse.unsw.edu.au/ billw/mldict.html (The Machine Learning Dictionary)

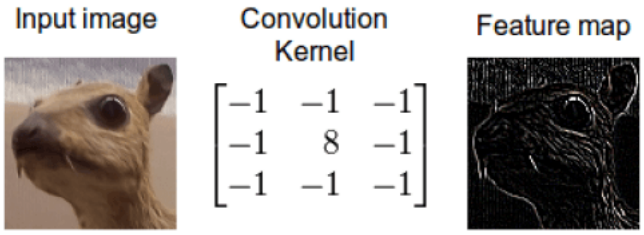


Figure 3.3: Example of a convolution filter applied to an image

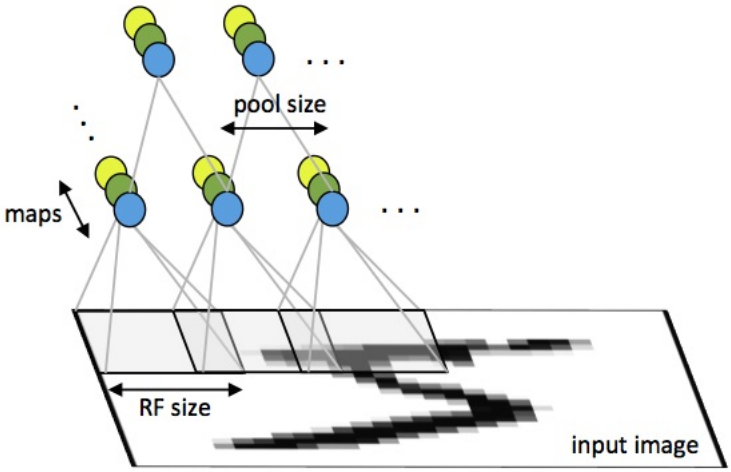


Figure 3.4: Copied from [21]: "First layer of a convolutional neural network with pooling. Units of the same colour have tied weights and units of different colour represent different filter maps"

3.1.3 Introduction to Transfer Learning

Transfer learning is a method to extend the domain of knowledge of a neural network. This means that a network can learn to recognise more classes [3] or it can be re-purposed for a task similar to what it was originally trained for [15]. For example, a CNN able to recognise cats and dogs could be re-trained to extend its knowledge and be able to recognise rabbits too. Or it could be re-trained to distinguish between lions and wolfs instead, but reusing some of the knowledge acquired in learning to recognise cats and dogs.

A prerequisite for transfer learning is a trained network. Applying transfer learning equals to selecting one or more neuron layers and re-training their connection weights with new data in the domain that the network should learn. The rest of the layers remain unaltered. Transfer learning requires less data in the training dataset compared to a regular training process.

Reasons for using transfer learning are primarily scarcity of data and time required to train a CNN from scratch [22].

- Retraining an entire CNN from scratch requires a lot of data. ImageNet database for instance has 1,2 Million images.
- A CNN is trained within a week using a few high end GPUs.

The choice of how many and which layers should be opened for re-training depends in large measure to the size of the new dataset and how similar it is to the original dataset [20]. Transfer learning in multiple layers with a relatively small dataset yields a high risk of overfitting; size references in relation to the number of layers can be found in Figure 3.13. The bottom layers[3] of a CNN contain more specific features of the data whereas the top layers are more generic, in practice this mean that when the new dataset is similar to the original one, best results are obtained re-training the bottom layers and locking the top ones since they represent generic features that would still apply.

Examples from Literature

Transfer learning was used to fine tune the weight of four layers of the CNN "Caffe Model Zoo". The CNN had 5 conv layers and 3 FC layers plus one Softmax loss layer and it was pre-trained with ImageNet [15]. 2 millions of images were used for the fine tuning of the top four neuron layers. The network learnt to recognise the orientation of chairs with 16-classes corresponding to to orientation ranges. Synthetic images were used for training.

Another example, [3] compares how the number of retrained layers affects the classification performance of CNN with transfer learning. They compare the accuracy with training sets of 1, 5, 10, 20 and 40 samples per class against retraining 1 to 5 layers (the network has 5 layers so 5 equals to training from scratch); figure 3.5 with accuracy results from that experiment. Their network has been pre-trained to recognise handwritten digits and it learns new digits with transfer learning; figure 3.6 shows the CNN architecture that they use.

3.2 Related Work

This section introduces breakthrough developments in CNNs; the table in Figure 3.13 compares the references reviewed for this thesis in terms of dataset features and the machine learning parameters that were used. CNN breakthrough developments in chronological order are:

[3]Some references in the literature present the network as a bottom-up flow in which case top layers correspond to bottom layers from this thesis and vice versa.

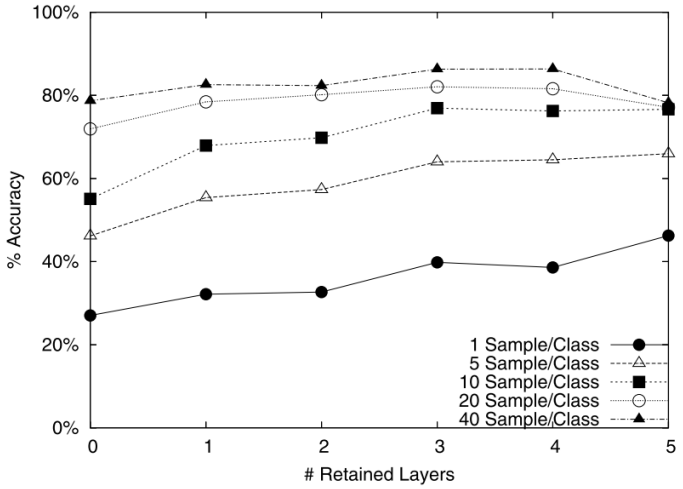


Figure 3.5: Copied from [3, p. 107]: "Comparison of learning curves showing accuracy vs. number of retained levels for various numbers of samples per class in the training set. Curves show, from top to bottom, results for 40, 20, 10, 5 and 1 sample per class. Each point represents the average of 5 trials on a testing set with 1,000 character samples."

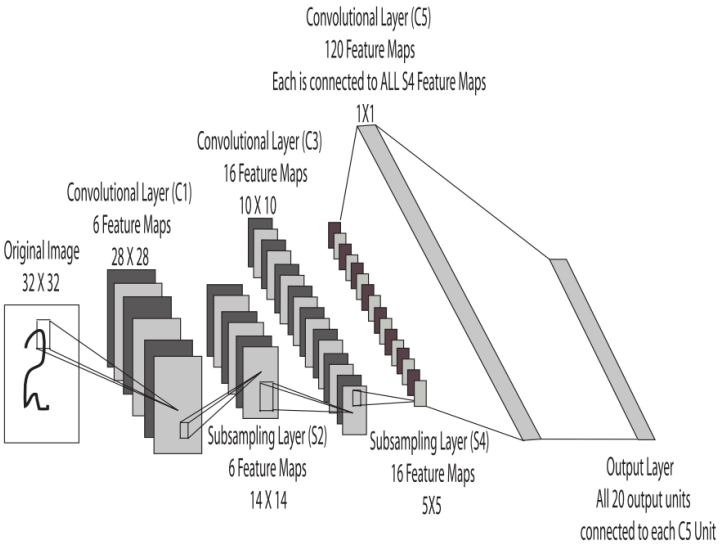
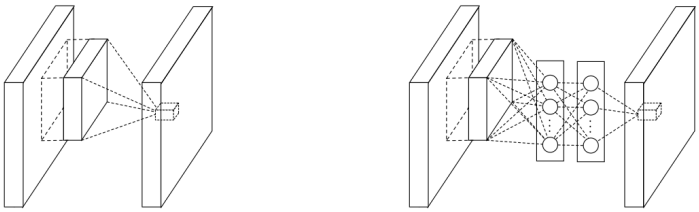


Figure 3.6: Copied from [3, p. 107]: "Architecture of our net, which is a slightly modified version of LeNet5."

1. LeNet-5 1989 [7]
2. NIN 2013 [10]
3. GoogLeNet 2014 [22]
4. Inception-V3 2015 [23]

LeNet 5 Significant architecture developments in chronological order start with LeNet-5 (1989) that introduces a standard structure of CNNs: "stacked convolutional layers (optionally followed by contrast normalisation and max-pooling) are followed by one or more fully-connected layers" [7].

Network In Network Network in Network architecture (2013). Within the domain of Convolutional Neural Networks (CNN), Network In Network (NIN) consists of "micro neural networks with more complex structures to abstract the data within the receptive field". NIN architecture replaces CNN convolutional filters with micro neural networks, which are a "more potent nonlinear function approximator that can enhance the abstraction ability of the local model"; figure 3.7.



(a) Linear convolution layer (b) Mlpconv layer

Figure 3.7: Copied from [10, p. 2]: "Comparison of linear convolution layer and mlpconv layer. The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network. Both layers map the local receptive field to a confidence value of the latent concept."

[10] presents a NIN structure consisting of a stack of mlpconv layers, on top of which lie the global average pooling and the objective cost layer. Sub-sampling layers can be added in between the mlpconv layers as in CNN and maxout networks. Mlpconv layers model the local patches better, and global average pooling acts as a structural regulariser that prevents overfitting globally. NIN "demonstrated state-of-the-art performance on CIFAR-10, CIFAR-100 and SVHN datasets" at the time. Figure 3.8 is a representation of the overall structure of NIN.

GoogLeNet (Inception Architecture) [22] Introduces "a new level of organisation in the form of the Inception module" (2014), an evolution of the NIN architecture. At the same time they increased the depth of the network. Optimisations in the

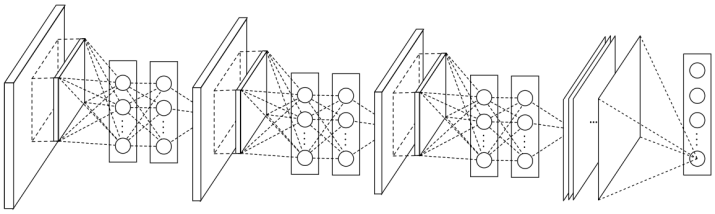


Figure 3.8: Copied from [10, p. 4]: "The overall structure of Network In Network. In this paper the NINs include the stacking of three mlpconv layers and one global average pooling layer."

network compensate the computational cost of a deeper network which allowed for a "significant quality gain at a modest increase of computational requirements". GoogLeNet is a particular implementation of the Inception architecture that was submitted and won ILSVRC 2014 image recognition competition.

The Inception architecture applies the principle of NIN: in [22] implementation NIN can be viewed as additional 1x1 convolutional layers followed typically by the rectified linear activation. The "1x1 convolutions have dual purpose: most critically, they are used mainly as dimension reduction modules to remove computational bottlenecks, that would otherwise limit the size of our networks" [22]. Figure 3.9 represents a convolutional layer with the additional 1x1 covolutions

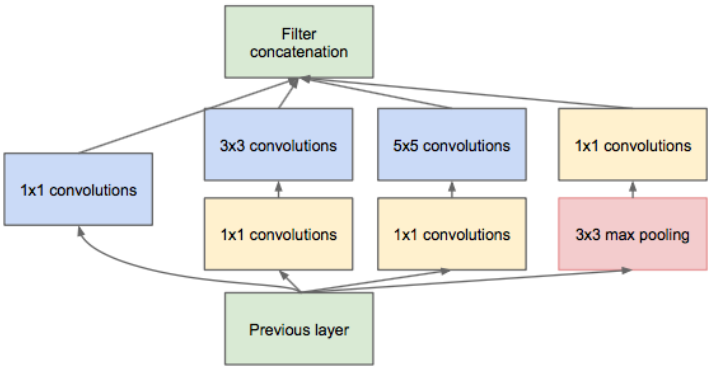


Figure 3.9: Copied from [22, p. 5]: "Inception module with dimension reductions."

They also reflect on the importance of additional 1x1 convolutional layers to obtain affordable computational costs: 1x1 convolutions perform a reduction in the filter dimension space prior to the expansive 3x3 or 5x5 convolutions. For example, consider having an input of size (F, W, H) feeded to the 1x1 convolution (C). F is the number of convolutional filters and W and H the spatial dimensions; C has F1 filters. The output size would be (F1, W, H) [1].

Inception-V3 The Inception architecture is hard to adapt to new use cases due to its complexity. Nonetheless, [23] unveils some design principles to guide the modification of Inception. They present Inception-V3 (2015), a continuation of the Inception family with some design improvements, better classification results and the ability to perform well with low-resolution images; figure 3.10 compares the recognition performance with the image resolution. I use Inception-V3 for the experiment presented in this report.

Receptive Field Size	**Top-1 Accuracy (single frame)**
79×79	75.2%
151×151	76.4%
299×299	76.6%

Figure 3.10: Copied from [23, p. 8]: "Comparison of recognition performance when the size of the receptive field varies, but the computational cost is constant."

An innovation that reduces the computational cost of Inception-V3 is the factorisation of convolutions with Large Filter Size: they replace 3x3 convolution with 3x1 convolution followed by a 1x3 convolution. This is 33% less expensive to compute; see figure 3.11 for an example of a 3x3 convolution factorisation.

Other architecture improvements are the use of "Auxiliary Classifiers" that "Improve the convergence during training by combating the vanishing gradient problem in very deep networks", "Efficient Grid Size Reduction" and "Label smoothing". Inception-V3 has better than state-of-the-art performance on the ILSVRC 2012 classification benchmark; figure 3.12.

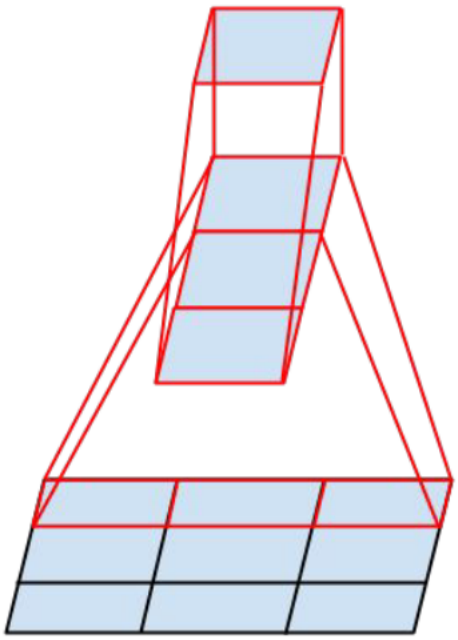


Figure 3.11: Copied from [23, p. 4]: "Mini-network replacing the 3x3 convolutions. The lower layer of this network consists of a 3x1 convolution with 3 output units."

Network	Models Evaluated	Crops Evaluated	Top-1 Error	Top-5 Error
VGGNet [18]	2	-	23.7%	6.8%
GoogLeNet [20]	7	144	-	6.67%
PReLU [6]	-	-	-	4.94%
BN-Inception [7]	6	144	20.1%	4.9%
Inception-v3	4	144	**17.2%**	**3.58%***

Figure 3.12: Copied from [23, p. 4]: "Single-model, multi-crop experimental results comparing the cumulative effects on the various contributing factors. We compare our numbers with the best published single-model inference results on the ILSVRC 2012 classification benchmark."

Reference	Dataset name	Dataset type	Dataset size	Image resolution	Color	# Train	# Transfer Learning	# Test	# Retrained Layers	# Classes	CNN Network	Top1 error	Top5 error	Accuracy
[7]		Handwritten zip code digits		256x256	No	7291		2007		10	LeNet5			91,90%
[8]	NIST Special Database 3	Handwritten digits	70K	32x32	No	60K		10K		10	LeNet5 (LeCun et al. 1999)			99,05%
[9]	NORB with transformations	Photographic on a studio	194K	96x96 (originally 640x480)	No	261K		58K		6	6 layer CNN			83,30%
[13]	Their own	Synthetic		150x150	No	100 per class		30 per class		2	V1-like model (not CNN)			from ~100% to < 50% (degrading with dataset difficulty)
[6]	ILSVRC-2010 (subset of ImageNet)	Photographic	1400K	224x224	Yes	1,2 million + 50K for validation		150K		1000	CNN with 60 million parameters, 650K neurons and five convolutional layers	37,50%	17%	
[10]	CIFAR-10, CIFAR-100, SVHN and MNIST	Photographic or scanning									Network In Network			from ~99,5% to ~65% (depending on dataset difficulty)
[21]	ILSVRC-2014 (subset of ImageNet)	Photographic	1400K	224x224	Yes	1,2 million + 50K for validation		150K		1000	GoogLeNet		6,67%	
[22]	ILSVRC-2012 (subset of ImageNet)	Photographic	1400K	299x299	Yes	1,2 million + 50K for validation		150K		1000	Inception-v3	21,20%	5,60%	
[3]	NIST Special Database 19	Handwritten Characters	800K	32x32	No	400 per class	1 to 40 per class	1K	0 to 5	20 (62 available)	LeNet5 with minor adjustments			~80%
[15]	Their own based on ShapeNet 3D models	Synthetic + Natural scene images for testing	2 million	Up to 255x255	No	~1M from ImageNet	2 million	2226	4	16	RCNN model from Caffe Model Zoo			93,20%
Me	**Lego**	**Synthetic + fotographic for test**	**60K**	**448x448**	**Yes**	**~1M from ImageNet**	**Independent experiments with 30, 1830 and 5400**	**30 per class**	**1**	**10**	**Inception-v3**	**57%**		**82%**

Figure 3.13: Literature comparison from references [7, 8, 9, 13, 6, 10, 22, 23, 3, 15] and my own implementation.

Chapter 4

Implementation and Experiment

This chapter explains the implementation work and the execution of image classification experiment. It is divided in two sections, first the method for the synthetic image dataset generation. Second the experimental work using that image dataset.

4.1 Generating the Lego dataset

I generate the synthetic image dataset based on a 3D model of a Lego fire truck. This method is based on the literature research from the section "2.2 - Theoretical Framework for Synthetic Dataset Generation".

The dataset and the Blender project to generate it are available[1] online on GitHub and "blender.stackexchange" community for other researchers who want to reuse the images for further experimentation or use the Blender generation script to create new datasets.

4.1.1 Software Tools

I compared 3D modeling and rendering software: Blender, Autodesk Maya and POV-Ray and ended deciding for Blender. POV-Ray was rejected for having worse rendering capabilities and because it uses a descriptive language instead of programming. The decision between Maya and Blender was made in favour of Blender based on having a popular online community where I would be able to obtain support. Blender includes an API in python programming language and a scripting mode which allows me to automate the rendering process with randomised parameters with a simple script that can be reused with different 3D models.

[1]The link to GitHub: https://github.com/ernestbofill/lego-image-dataset and to the Blender community: http://blend-exchange.giantcowfilms.com/b/2204/

4.1.2 Proof of Concept

In order to develop, test and tune the Blender generation script, I made several attempts to generate very small datasets. The proof of concept is a set of a 100 rendered images with automated changes in orientation and light (Appendix A with those 100 images). It uses a sample 3D model of a train from a website of free 3D models[2]. Figure 4.1 shows some rendered images from the proof of concept.

Figure 4.1: Images from proof of concept dataset with random lighting and orientation.

4.1.3 The 3D Model

The 3D model is a Lego "Fire Ladder Truck" downloaded from a website with free Lego 3D models[3]. I also bought the physical product to create the photographic dataset. The reason for choosing Lego is to recreate an assembly situation where the image recognition engine would assist the building process (as this was a suggested use case in the Introduction). Figures 4.2 and 4.3 show an image of the Lego product and a pre-selection of bricks (later referred to as parts or classes) for the dataset.

[2]http://www.3dxtras.com/ (Download page of the train 3D model
[3]http://www.eurobricks.com/ (Download page of the Lego 3D model)

10 bricks are selected for the dataset from the complete product. To export and use the 3D model in Blender I used the software tools "Lego Digital Designer" and "LegoDraw extention for Blender". The 3D model is a close representation of the real bricks but qualities such as colour or behaviour under certain light condition vary slightly. It is uncertain how these variations will influence the capacity of the CNN to recognise photographic images from knowledge learnt with synthetic images.

Figure 4.2: The selected Lego product.

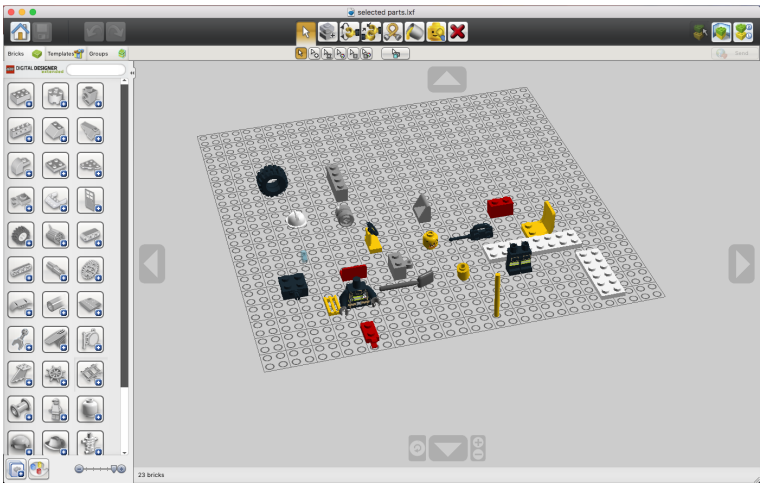


Figure 4.3: Some preselected Lego bricks from the 3D model laid out in the "LegoDraw" 3D modeling software. Ten of these were finally used for the dataset.

4.1.4 Synthetic Dataset Specifications

This section exposes the attributes of the synthetic dataset (specifications) in terms of name, size of the dataset, number of classes and variation between rendered images. The decision for these attributes is based on related work in the same research field, for example I incorporate the definition of "real world variation" from [13] and chose the size of the dataset from [15]. These are my synthetic dataset specifications:

- Name of the dataset: "Lego-10-6k"
- Method: Rendered images from a 3D model.
- Size: 60K images.
- Classes: 10 classes where each class is a different Lego brick.
- Light randomisation: The direction and intensity of light is randomised between a minimum and maximum intensity. There is a base lighting in all directions. Randomised lights are applied on top of that base light.
- Orientation randomisation: The object is randomly rotated up to 360 degrees in any direction.
- Position randomisation: The object varies the position slightly by randomly moving up, down, left, right, forward and backward.
- Distance randomisation: The object is slightly scaled up or down creating the effect of varying distance.

4.1.5 Implementation in Blender

The dataset generation implementation in Blender approximates real world variation in an automated manner using Blender's scripting functionality. This automated variation from one rendered image to the next makes it possible to generate a large dataset of synthetic images within an affordable time. It also meets the criteria from the dataset specifications in the previous section 4.1.4. This implementation results in a Dataset Generation script in Python which is published together with the dataset and included in Appendix B.

Randomising Light 6 light sources are positioned equidistantly around the object: front, back, right, left, above and below. These sources are Blender's light emitting panels. Initially each of these panels has an emission strength of 2. Figure 4.4 shows the light sources positioned around the object in Blender. Light randomisation is achieved in the following way:

- For each rendering iteration either 1 or 2 of the 6 light sources are selected randomly.
- The emission strength value of the selected lights is set to a random value ranging from a minimum of 1 up to 20 with a step of 1.
- The rest of the lights are left with an emission strength of 2 to create a basic lighting and avoid regions with complete absence of light.
- The camera rotates around the object in any direction up to 360 degrees with a step of 1 degree. This way the lights can illuminate from any direction in relation to the camera (although they preserve the distance from each other at all times).
- Each rendered image has an almost unique lighting setting.

This setup is intended to replicate a real situation where there is a general strong light reflected in all directions: which would make objects have at least a little bit of light in all their surfaces. This could emulate a daylight situation or an artificially well lit room. In addition there would be one or two sources of directional light that will disturb an otherwise smooth lighting. These could be the sun or other strong light sources.

This lighting setup is imperfect: it only covers a limited number of scenarios and it is not an accurate representation of the real world because it lacks surfaces for the light sources to reflect and the object does not cast shadows. It is uncertain how this imperfection will impact the CNN training and classification performance.

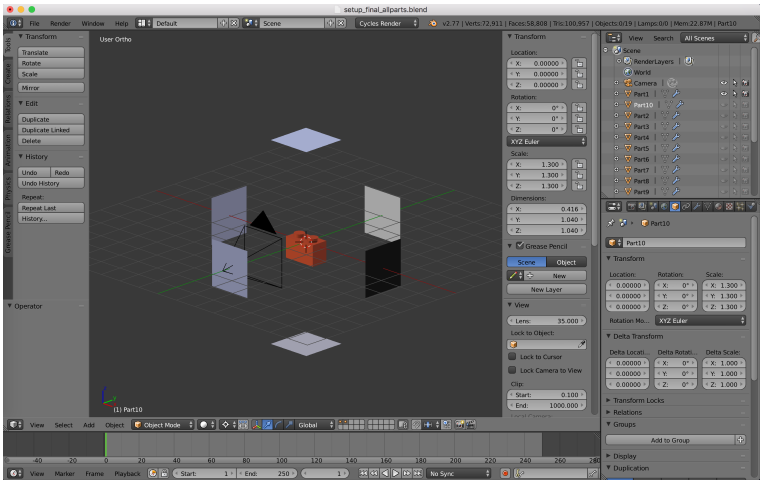


Figure 4.4: Blender project setup with the Lego brick, the light emitting panels and the camera; all prepared for rendering

Randomising Orientation, Position and Scale The object is randomly rotated up to 360 degrees in any direction with a step of 1 degree. It is also randomly scaled a maximum of +/- 20% to recreate the perception of different distances. In addition, the object is not completely centered; it is shifted from the center randomly according to a normal distribution with $mean = 0$ (the center of the image) and $sigma = 0,7$. With this I want to recreate a natural situation where a hypothetical photographer aims at having the object in the middle of the picture but naturally deviates slightly. Figure 4.5 shows a sample of two rendered images from the same object with observable differences in orientation, position and scale.

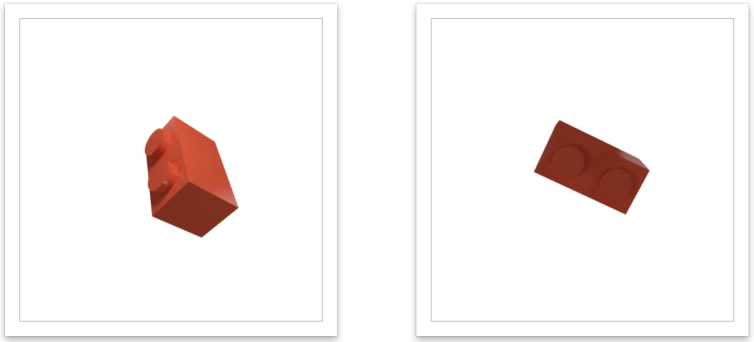


Figure 4.5: Comparison of two rendered images with different orientation, position, scale and lighting.

Randomizing background There are 6 background images with photos of nature; Figure 4.6. Unlike the other parameters, the number of backgrounds settings is limited to 6 options instead of thousands. The reason is that it is very time consuming with this process to have several backgrounds, therefore variation is compromised due to time constraints.

Each background is used the same number of times and it is uncertain how the repetition of the same backgrounds will impact the CNN training and classification performance.

4.1.6 The Photographic Dataset

The photographic dataset is a collection of photographic images of the same Lego bricks. It is fundamental for the image classification experiment (Section 4.2) as it will be used to train the network – amplified with synthetic images – and also used as the test dataset. These are the specifications of the photographic dataset in in terms of number of images, background, light conditions, position and orientation:

- Name of the dataset: “Lego-10-photo-60”.
- Method: photos taken with an iPhone 6 camera.

Figure 4.6: The same object rendered on top of each one of the 6 different background images used for the synthetic dataset

- Size: 600 images.
- Classes: 10 classes where each class is a different Lego brick.
- Location: 10 different indoors placements; Figure 4.7.
- Light randomisation: 10 corresponding to 10 locations where photos are taken. In addition each photo varies slightly despite being in the same location.
- Orientation randomisation: 6 orientations for each place: "front", "top", "back", both "sides" and "bottom". The orientation is never straight but a little bit from the side therefore showing multiple faces of the object.
- Position and distance randomisation: The object is quite centred and the shot is close so that object takes most of the camera visible range (similar to the synthetic images).

4.2 Image Classification Experiment

This section explains the image classification experiment and the process to prepare and execute it. As it was explained in the Introduction, I want to find out how amplifying the train dataset with the artificially rendered images compares with

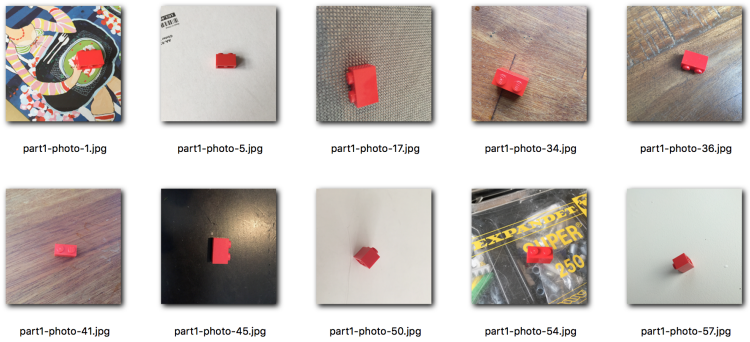


Figure 4.7: Samples from the photographic dataset, each one taken on a different location

a much smaller training dataset of photographic images in terms of recognition accuracy.

4.2.1 Initial Steps

I did some preliminary experiments in order test my machine learning environment. Next I will summarise these initial attempts and how they led to a working configuration.

Initially, I was using an AAU server with mxnet[4] deep learning framework and Jupyter's python interface. I followed an example to train a CNN with the mnist dataset. Then I modified the parameters to use a subset of Lego dataset with two classes instead of mnist but the training accuracy was around 50% which indicated there was an error. This problem added to some other difficulties that I was having to implement a proper training configuration with mxnet made me decide to try a different framework.

At this point I started using TensorFlow as the machine learning framework running on my private computer (hardware and system details in Section 4.2.4). I ran some preliminary experiments based on a tutorial by Google[5]. One of these preliminary experiments consisted on applying transfer learning on the Lego dataset reduced to only two classes and 1800 instances per class. Then, a prediction test with 10 photos of the Lego bricks on the re-trained network.

The results were promising. Trainings experiments reached 96% and 98,4% validation accuracy respectively. The prediction results were not as good, it classified 9/10 images correctly where 2 of the correct classifications had a low margin; Figure 4.8, Figure 4.9 and Figure 4.10

[4]https://github.com/dmlc/mxnet
[5]https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/

Image	Is part 1	Is part 2
Image 1-1	0,988	0,011
Image 1-2	0,990	0,009
Image 1-3	0,984	0,015
Image 1-4	0,977	0,022
Image 1-5	0,981	0,018
Image 2-1	0,035	0,964
Image 2-2	0,126	0,873
Image 2-3 (misclassified)	0,697	0,302
Image 2-4 (low margin)	0,435	0,564
Image 2-5 (low margin)	0,425	0,547

Figure 4.8: Preliminary classification test with 10 images and 2 classes, showing the predicted probability for each combination of image and class. Image m-n where m is the class and n the image number

True class / Predicted class	Class 1	Class 2
Class 1	5	1
Class 2	0	4

Figure 4.9: Confusion matrix of the preliminary classification test with 10 images and 2 classes

4.2.2 Determining the proportion of train and test images

After that initial experiment from the previous Section 4.2.1, I wanted to test my machine learning setup and the Lego dataset more before performing the final experiments. The main objective was to determine a good proportion of train and test images within the very limited amount of photographs (60 per class). These results would help optimise the test cases A, B and C presented in the next Section 4.2.3 cases later in the report. For now, Figure 4.11 shows the compared configurations and it's results.

4.2.3 Test Cases

Three test cases will be analysed and compared. The tests are designed to show the difference between training the network with a small photographic dataset, a dataset augmented with synthetic images and a dataset composed of synthetic images only. Figure 4.12 shows the distribution of photographic and synthetic data for each test case; the train data is different in every case but the test dataset is the same. The proportion of train and test data is based on the findings from the preliminary research explained in the previous Section 4.2.2.

Figure 4.10: Test images for the preliminary classification test with 10 images and 2 classes

#Train	#Test	Average acc.
200	400	0,706
300	300	0,746
400	200	0,756

Figure 4.11: Different proportions of train and test images and the classification accuracy calculated as the averaged probability of the correct classification

Train dataset

Case A

Photographic images per class: 30
Synthetic images per class: 0

Case B

Photographic images per class: 30
Synthetic images per class: 1800

Case C

Photographic images per class: 0
Synthetic images per class: 5400

Test dataset

Cases A, B, C

Photographic images per class: 30 (Different ones than the training dataset, divided randomly for train and test)

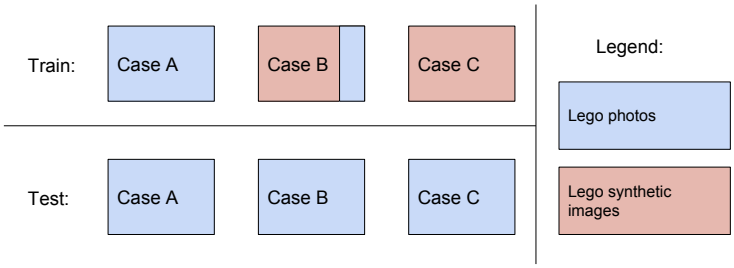


Figure 4.12: The dataset distribution of synthetic and photographic images for the test cases

4.2.4 Tools

The experiment runs on a Linux virtual machine by the vitalisation software VirtualBox. This virtual machine has allocated 5GB of RAM and 4 CPU cores of type "Intel Core i7" running at 2,4GHz. The machine learning framework is TensorFlow[6] by Google using Python as the programming language. The source CNN is Inception-V3 (see Section 3.2) trained with ImageNet [16]. The training dataset is the Lego synthetic dataset with 10 classes and the test dataset is the Lego photographic dataset with the same 10 classes.

4.2.5 Transfer Learning

The transfer learning configuration for this experiment keeps the weights of all layers except the second last one ("linear"); Figure 4.13. This layer is replaced with

[6]https://www.tensorflow.org/

a "softmax" layer with the same input size. The bottom layers have to do with more concrete features than the top ones, therefore one can assume that weights in the top layers that have been trained with different images will also work on the new set. The retraining of this new layer will estimate 20480 parameters corresponding to weights and biases.

The reason to train just one layer instead of 2 or more is the limited number of training images. According to [15] 80k training images are needed to retrain 2 layers in a similar CNN model. In my experiment, test case A has only 300 images, B has 18.300 and C 54000. Possibly C would benefit from training a second layer.

type	**patch size/stride** or remarks	**input size**
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 4.13: Copied from [23]: The Inception-V3 layers. Highlighted is the layer replaced and retrained in my transfer learning implementation

4.2.6 Analysis

I want to compare the accuracy with the different training sets in cases A, B and C. The accuracy is determined by the sum of correctly classified images divided by the total tested images: $acc = correct/total$. The best performing case will be analysed further to understand how factors such as shape, colour or transparency have an impact on the recognisably of the images.

The analysis is performed on Matlab and the raw data can be found in Appendix C. These are the analysis steps:

1. Evaluate the accuracy of test case separately with the formula $acc = correct/total$.

2. Continue with the best performing case which is analysed further.
3. Look at the confusion matrix and take the top 4 most confused classes.
4. A qualitative analysis of the most confused classes considering factors such as colour, shape or transparency and whether they are likely to be the reason for a higher confusion.

Chapter 5

Results and Discussion

5.1 Comparison of Test Cases

The accuracy for case A is 82%, case B is 68% and case C 60,67%; Figure 5.1. Case A, where the network was trained with 300 photographic images yields the best performance despite the relatively small amount of training data. In case B, with 18K synthetic images added to the 300 photos, the performance drops $14,00pp$ (percentage points). Finally, in Case C, trained exclusively with a much larger number of synthetic images, has the worst result with $7,33pp$ less than case B.

	Case A	Case B	Case C
Train set	300 photos	18k synthetic + 300 photos	54k synthetic
Accuracy	82,00%	68,00%	60,67%

Figure 5.1: Test results.

These results suggest that the CNN is not as good at recognising the object from knowledge learnt from synthetic representations compared to learning from photographic data. In case B, despite the photos being part of the training dataset, a much larger number of synthetic images would dilute the effect of the photos instead of adding value to the training, but it still performs better than C with no photos at all.

Another plausible explanation for these results are the differences in the randomisation of the background, lighting, orientation, position and distance between the photos and rendered images; Figure 5.2. The synthetic images have a perfect randomisation of orientation whereas the photos have less variation. Position and distance variations appear to be similar in the two datasets. In the rendering process, lights are randomised in terms of intensity and direction, nevertheless a qualitative difference compared to a real world image is noticeable: primarily, the reflections on the object tend to be more white compared to the photos and there

is a lack of walls and other elements around the object for the light to bounce back (which is partially compensated with the inclusion of several light sources around the object). In addition, the synthetic images do not cast any shadows. On the other hand the photos have 10 light conditions with small variations in each single take. The 10 light conditions correspond to the 10 different locations chosen to take the photographs, in some there is only natural light while in others artificial light from a bulb is dominant. Finally, comparing the backgrounds: in the synthetic dataset there are 6 random backgrounds placed behind the image, those backgrounds are images of landscapes and they have no connection with the object; it is as though the object was floating in the middle of those landscapes. In the photographic dataset there are 10 different backgrounds corresponding to the 10 places where the photos were taken, most of them are more uniform compared to the synthetic dataset and there is always a connection with the object since the object rests on top the surface that is on the background.

Feature	**Synthetic dataset**	**Photographic dataset**
Number of lighting conditions	Almost infinite combinations of direction and intensity	10 with small variations
Light reflection	Only on the object, turning white under intense light	Natural
Shadows	No shadows	Natural
Color of the Lego brick	Slightly biased, especially the red color	Photographic resemblance
Number of different backgrounds	6	10 with small variations
Type of background	Landscape image placed behind the object	The surface where the lego brick rests, e.g. desk, cardboard, etc.
Orientation	Perfectly randomised	Manually randomised
Position	Random within a range	Approximately centered (manually)
Distance	Random within a range	Manual variation within a range

Figure 5.2: Differences in the randomisation parameters between the synthetic dataset and the photographic dataset.

Based on those differences between the synthetic dataset and the photographic dataset, it could be that the CNN can actually recognise the features of the object, but is confused with variations in the context. Backgrounds could be particularly responsible for this problem since the variation is smaller than the other parameters (6 variations for the synthetic images and 10 for the photos). Because of that, the network might have learnt features from the background images.

A final consideration, the test accuracy in case C is still 6 times better than that of a random classifier (accuracy: 10%) and may be a suitable image recognition method in absence of photos or more realistic representations to train the CNN.

Figure 5.3: Comparison of backgrounds. On top, the 6 backgrounds found in the synthetic dataset. Below, a sample of 6 backgrounds found in the photographic dataset (the photographic dataset has 10 backgrounds in total)

5.2 Confusion Results in Case A

This section takes a closer look at the best performing experiment, which is Case A. I want to analyse which classes are confused with what others and try to establish patterns.

True Labels	Estimated Labels 1	2	3	4	5	6	7	8	9	10	Totals
1	29	1	0	0	0	0	0	0	0	0	30
2	0	22	2	1	0	0	0	5	0	0	30
3	0	0	29	0	0	0	0	1	0	0	30
4	0	0	0	16	14	0	0	0	0	0	30
5	0	2	0	12	13	0	0	1	0	2	30
6	0	0	0	0	0	30	0	0	0	0	30
7	0	0	0	0	0	3	26	1	0	0	30
8	0	4	0	0	0	0	0	26	0	0	30
9	0	0	0	1	0	0	2	0	26	1	30
10	0	0	0	0	0	0	0	0	1	29	30
Totals	29	29	31	30	27	33	28	34	27	32	300

Figure 5.4: Confusion Matrix for Case A

There are 2 clear patterns based on the confusion matrix; Figure 5.4

- Confusion between Lego brick No. 4 and Lego brick No. 5 (classes 4 and 5)
- Confusion between Lego brick No. 2 and Lego brick No. 8 (classes 2 and 8)

The confusion between classes 4 and 5 is likely to be the result of strong similarities in shape and colour; Figure 5.5. The only difference between the two is that class 4 is 1/3 longer than 5 and they are by far the most confused Lego bricks with a 46% of wrongly classified photos in class 4 and a 57% miss-classification in class 5. When observed from a perfectly frontal perspective the two objects are completely indistinguishable, nevertheless the CNN also confused many instances where the shape of the object could be clearly recognised.

Those images confused with class 8 tend have similar backgrounds with abundant elements in it. It it possible that the background is a major factor in this

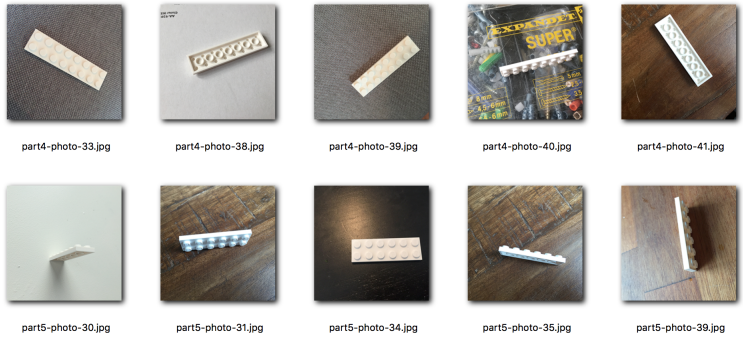


Figure 5.5: Examples of confusion between classes 4 and 5. On top, instances of 4 classified as 5. Below, instances of 5 classified as 4.

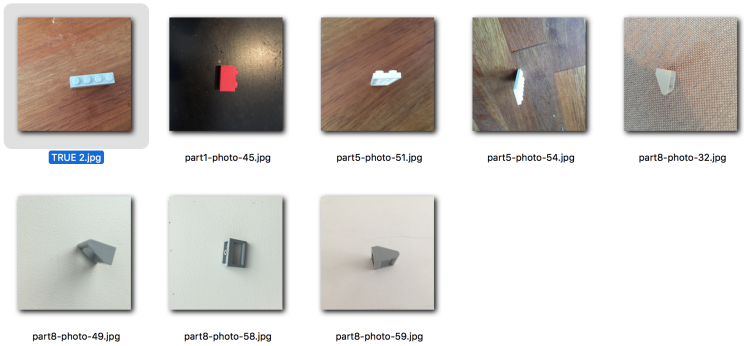


Figure 5.6: Examples of images miss-classified as class 2 and a true class 2 highlighted on the top left corner.

confusion, but it is unclear how; Figure 5.7. Finally for the images confused with class 2, in most cases it is a class 8 miss-classified as 2. Class 8 has the same colour as 2 and from certain perspectives a similar shape too; Figure 5.6. The incidence of images confused with class 2 is 1,85% (5 out of 270) and for class 8 is 2,96% (8 out of 270).

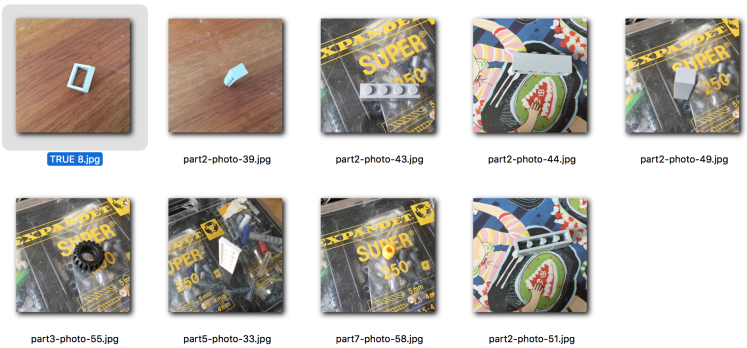


Figure 5.7: Examples of images miss-classified as class 8 and a true class 8 highlighted on the top left corner.

Chapter 6

Conclusion and Future Work

This thesis aims to find out whether re-training a CNN with a small dataset that has been synthetically augmented – thus becoming a much larger dataset – increases the classification performance compared to no augmentation. This performance has been measured in terms of classification accuracy. The results of the experiment are contrary to that hypothesis since the synthetically augmented dataset registered a worse classification performance.

This observation cannot be extrapolated because it could be the product of specific conditions in the experiment. For the CNN re-trained with a large synthetic dataset to outperform one re-trained with a small number of photographs the synthetic dataset should have a higher level of realism in terms of, e.g., light, colour or background. It is yet to determine which features have more influence in the results but a more realistic representation of the object could improve the performance of the synthetic dataset.

The preparation of this experiment required the generation of a vast amount of annotated synthetic images. Rendering synthetic images proved to be fast, automatable, versatile and adjustable. In practice this means that a high-end computer could render $\sim$100.000 images within a week without human intervention. Because of that, this dataset generation technique can be very useful either to amplify existing data or to create new datasets. ‘

In future work, it would be useful to determine a threshold where the number of photographic images is so small that amplifying with the synthetic data improves the classification accuracy. In the same direction, experimenting with retraining a second or even more layers of the network and whether that improves the result. A CNN should perform better retraining more than one layer, but as more layers are opened for transfer learning, more data is needed to achieve optimal results.

Another element for further research is the influence of the different features that make the synthetic images realistic. Some features could be more important

than others, and more effort should be put upon them. For example, in this research I suspect the diversity of background images is too small and it might have had a negative impact on performance. Additional experiments with a dataset modified to include more background settings would clarify the relevance of this matter and offer guidance on generating image datasets in the future.

Assuming that both the rendering realism and the CNN transfer learning process can be optimised: the augmentation of training datasets by synthetically rendered images can provide a cheap and effective way to introduce machine learning in several domains, especially when 3D models are already available. This could be the case of for example, vehicles, furniture, tools, electronics or architecture among many others.

Bibliography

[1] Indie Al. *deep learning - What does 1x1 convolution mean in a neural network? - Cross Validated*. 2016. URL: http://stats.stackexchange.com/questions/194142/what-does-1x1-convolution-mean-in-a-neural-network (visited on 03/28/2016).

[2] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: (2013). arXiv: 1311.2524. URL: http://arxiv.org/abs/1311.2524.

[3] Steven Gutstein, Olac Fuentes, and Eric Freudenthal. "Knowledge Transfer in Deep Convolutional Neural Nets". In: *International Journal on Artificial Intelligence Tools* 17.03 (2008), p. 555. ISSN: 0218-2130. DOI: 10.1142/S0218213008004059. URL: http://www.worldscinet.com/ijait/17/1703/S0218213008004059.html.

[4] Bernd Heisele, Gunhee Kim, and Andrew J. Meyer. "Object Recognition with 3D Models". In: *Procedings of the British Machine Vision Conference 2009* (2009), pp. 29.1–29.11. DOI: 10.5244/C.23.29. URL: http://www.bmva.org/bmvc/2009/Papers/Paper081/Paper081.html.

[5] Shahram Izadi et al. "Kinect Fusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology - UIST '11* (2011), p. 559. ISSN: 9781450307161. DOI: 10.1145/2047196.2047270. arXiv: 1409.1556. URL: http://dl.acm.org/citation.cfm?id=2047270{\%}5Cnhttp://dl.acm.org/citation.cfm?doid=2047196.2047270.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances In Neural Information Processing Systems*. 2012, pp. 1–9. ISBN: 9781627480031. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[7] Y. LeCun et al. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541. URL: http://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.4.541.

[8] Y LeCun et al. "Object Recognition with Gradient-Based Learning". In: *Feature Grouping*. 1999, pp. 319–345.

[9] Bottou LeCun, Y. Huang, F.J. "Learning Methods for Generic Object Recognition with Invariance to Pose and Lighting. I". In: (2004).

[10] Min Lin, Qiang Chen, and Shuicheng Yan. "Network In Network". In: (2013), p. 10. arXiv: 1312.4400. URL: http://arxiv.org/abs/1312.4400.

[11] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015. URL: http://neuralnetworksanddeeplearning.com/chap1.html (visited on 10/10/2016).

[12] Maxime Oquab et al. "Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks". In: *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (2014), pp. 1717–1724. ISSN: 10636919. DOI: 10.1109/CVPR.2014.222.

[13] Nicolas Pinto, David D. Cox, and James J. DiCarlo. "Why is real-world visual object recognition hard?" In: *PLoS Computational Biology* 4.1 (2008), pp. 0151–0156. ISSN: 1553734X. DOI: 10.1371/journal.pcbi.0040027. arXiv: arXiv:1202.2745v1. URL: http://dx.plos.org/10.1371/journal.pcbi.0040027.

[14] Victor Powell. *Image Kernels explained visually*. URL: http://setosa.io/ev/image-kernels/ (visited on 12/20/2016).

[15] Ruizhongtai Charles Qi. "Learning 3D Object Orientations From Synthetic Images". In: (2015), pp. 1–7. URL: http://cs231n.stanford.edu/reports/rqi{_}final{_}report.pdf.

[16] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. ISSN: 15731405. DOI: 10.1007/s11263-015-0816-y. arXiv: 1409.0575. URL: http://link.springer.com/10.1007/s11263-015-0816-y.

[17] Steven W Smith. "Linear Image Processing". In: *The Scientist and Engineer's Guide to Digital Signal Processing*. 1997. Chap. 24, pp. 397–422. ISBN: 9780750674447. DOI: 10.1016/B978-0-7506-7444-7/50061-3. URL: http://www.dspguide.com/ch24/1.htm.

[18] Stanford University. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/ (visited on 01/12/2017).

[19] Stanford University. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-1/ (visited on 12/20/2016).

[20] Stanford University. *CS231n Convolutional Neural Networks for Visual Recognition - Transfer Learning*. URL: http://cs231n.github.io/transfer-learning/.

[21] Stanford University. *Unsupervised Feature Learning and Deep Learning Tutorial*. URL: http://ufldl.stanford.edu/tutorial/supervised/FeatureExtractionUsingConvolution/ (visited on 12/20/2016).

[22] Christian Szegedy et al. "Going deeper with convolutions". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 07-12-June (2014), pp. 1–9. ISSN: 10636919. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[23] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *arXiv preprint* (2015). ISSN: 08866236. DOI: 10.1002/2014GB005021. arXiv: 1512.00567. URL: http://arxiv.org/abs/1512.00567.

[24] B. Weyrauch et al. "Component-Based Face Recognition with 3D Morphable Models". In: *2004 Conference on Computer Vision and Pattern Recognition Workshop* (2004), pp. 1–5. ISSN: 03029743. DOI: 10.1109/CVPR.2004.41. URL: http://link.springer.com/10.1007/3-540-44887-X{_}4http://link.springer.com.zorac.aub.aau.dk/chapter/10.1007/3-540-44887-X{_}4.

Appendix A

Proof of Concept Dataset

Proof of concept synthetic dataset of a 100 images:
`https://www.dropbox.com/s/5zffzh6m60q3dw0/sample-dataset.zip`

Appendix B

Python Generation Scripts and Lego Datasets

The python generation scripts to generate a synthetic image dataset in Blender and the Lego datasets:

`https://github.com/ernestbofill/lego-image-dataset`

Appendix C

Raw Test Data

The test results as a csv files:

`https://www.dropbox.com/s/zvde03p3502yowj/raw_data.zip`