Examination of Unsupervised Representation Learning by Predicting Image Rotations

Bachelorarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Eric Lagger 10-934-370

2020

Leiter der Arbeit:
Prof. Dr. Paolo Favaro
Institut für Informatik
## Contents

1 Introduction ........................................... 4
   1.1 Preface ........................................... 4
   1.1.1 Artificial Intelligence .......................... 4
   1.1.2 Machine Learning ............................... 4
   1.1.3 Idea of Supervised and Unsupervised Learning ... 5
   1.2 Motivation ........................................ 6
   1.3 Contributions .................................... 6

2 Background Knowledge ................................. 7
   2.1 Introduction Neural Networks ....................... 7
   2.1.1 Multilayer Perceptron ........................... 7
   2.1.2 Convolutional Neural Networks .................. 8
   2.1.3 Learning from Data .............................. 12
   2.2 Improving the Learning Algorithm .................. 16
   2.2.1 Overfitting and Underfitting .................... 17
   2.2.2 Normalization ................................... 17
   2.3 Transfer Learning ................................ 17

3 Prior Work ............................................ 18
   3.1 Supervised Algorithms ............................. 18
   3.1.1 ImageNet Classification with Deep Convolutional Neural Networks ............... 18
   3.1.2 Very Deep Convolutional Networks for Large-Scale Image Recognition ............... 18
   3.1.3 Network in Network ............................... 19
   3.2 Unsupervised Algorithms ............................ 20
   3.2.1 Unsupervised Representation Learning by Predicting Image Rotations .................. 20
   3.2.2 Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles .......... 21
   3.2.3 Unsupervised Learning of Visual Representations using Videos ....................... 21
   3.2.4 Unsupervised Visual Representation Learning by Context Prediction .................... 21
   3.2.5 Colorful Image Colorization ...................... 21
   3.2.6 Adversarial Feature Learning ..................... 21
   3.2.7 Unsupervised Learning by Predicting Noise ........................................ 22
Chapter 1

Introduction

1.1 Preface

To understand my thesis some back ground will be needed and elaborated. This was where I started as well some time ago and this was the first part I had to do as well for this thesis. It took a while to learn some of the components of machine learning and computer vision. It sure is a fascinating field and it was mostly a pleasure to get into it.

1.1.1 Artificial Intelligence

In the beginning of computers people knew how to use the intelligence of computers by solving for humans difficult mathematical problems and complicated equations in a much faster way than any human could do it, these were problems that had clear and specific mathematical rules. It was much more challenging for computers to solve tasks that are intuitive and easy for humans such as recognizing a face in a picture or understanding and processing spoken language.

The capability of artificial intelligence systems to acquire own knowledge by extracting patterns from raw data is known as machine learning.[1]

So while the term artificial intelligence covers all smart ways to solve any given problem and tasks this thesis will focus on the subfield of machine learning. This covers the idea of extracting information, especially patterns and correlations, in a given dataset. The used approach in this thesis will be neural networks.

1.1.2 Machine Learning

Computational methods using experience to improve performance or to make accurate prediction could broadly be a definition of machine learning.[2]

Experience is the past information available to the learner, typically electronic data collected and made available for analysis (human-labeled training sets or other information from interaction with environment e.g.). The size and quality of the information source is crucial for the performance of the learner.[2]

Predicting the label of a new unknown document given a finite set of labeled documents is an example of a learning problem. The larger the set is - the easier the task gets. If a large number of documents are labeled wrong the
task gets harder again, this is an example of the quality of the information. The complexity of the task is augmented with the number of labels. Machine learning is strongly related to data analysis and statistics as the success of the algorithms depend on the given data. Ideas from statistics, probability and optimization combined with computer science make up learning techniques. Machine learning has a wide field of application, this list has no entitlement to be complete:

- Text classification. As in the short example from before, this could be detecting spam.
- Natural language processing. Working with the natural (human) language and analyze it with computer language.
- Speech processing applications. Including speech recognition, speaker identification or acoustic modeling.
- Computer vision applications. In this thesis this field will be further analyzed. Face detection, object recognition are known applications.
- Computational biology applications. Predicting the function of a protein or gene analysis are included.
- Many problems are tackled with machine learning in some way the list above gives a more general overview. Here is a concrete list to get a better understanding of the challenges that can be faced.
  - Learning to play games such as chess or Go.
  - Unassisted control of vehicles such as cars.
  - Search engines.
  - Medical diagnosis.
  - More creative tasks can be approached with machine learning as well. Creating new images from given images is one that has been faced. Having datasets of drawings of a painter as Claude Monet and datasets of real photographs it is possible to try to make the drawings look like a real photograph with some astonishing results. Or the transition in the opposite direction.

A lot of points could be added and the field is still expanding.

1.1.3 Idea of Supervised and Unsupervised Learning

It is possible to divide algorithms in basically two different categories. One would be supervised algorithms and the other unsupervised. In supervised algorithms the Data consists of two parts. There is the input data for the algorithm and a label for each input. The label could also be referred at as the desired output of the algorithm. The goal of a supervised algorithm is to take the input and with some computations be able to predict the label. Once it performs well on the training data we hope that it will do so on new unlabeled data. An unsupervised algorithm will only receive input data and will try to learn something on the dataset without having a given label to begin with.
Self-supervised algorithms are a variation from the unsupervised algorithms. These algorithms create some kind of "free" label that is given within the data. Rotating an image could be such a creation of a label, a rotated image would have one label and a non rotated another label. The label is not given with each image but rather created with each image.

1.2 Motivation

In recent years deep convolutional neural networks achieved a lot of progress. To train such a network it a lot of data is required and in supervised learning algorithms it is necessary that the data is labeled. To label data there is a lot of human work needed and this takes a lot of time and money to be done. To avoid the inconveniences that come with this we would like to find systems that don’t need labeled data and therefore are unsupervised learning algorithms.

This is the importance of unsupervised algorithms, even though their outcome is not yet on the same qualitative level as supervised algorithms.

In this thesis we will discuss an approach of such a system and compare the results to other papers. A deep convolutional neural network is trained to learn the rotations that have been applied to a picture. So we take a large amount of images and apply some simple rotations and the task of the network is to discover in which direction the image has been rotated. The data doesn’t need to be labeled to any category or anything else. As long as all the pictures are upside down we hope to find some high dimensional patterns for the network to learn.

1.3 Contributions

Starting from a self supervised task described in the implementation has been adapted and further examined. A variety of alterations have been added and combined to try to improve the outcome.
Chapter 2

Background Knowledge

2.1 Introduction Neural Networks

Knowing the basic ideas of machine learning it is implied that these algorithms will analyze some data and try to find consistencies in the dataset and make some decisions based on some input. Further it will make some changes on how it decided to improve the outcome. Neural networks have been inspired by the way a brain (human or of another animal) functions. However they are not designed to be a realistic model of a brain.[1]

In practice variations of networks have been implemented. The most relevant for this thesis shall be briefly presented for the understanding. These networks are feedforward networks, meaning that the output is evaluated from the input \( x \) with some calculations to \( y \). There is no feedback information that is calculated and fed back as input.[1]

2.1.1 Multilayer Perceptron

The first network to be discussed in this thesis is the multilayer perceptron because it can be considered to be one of the more simple ones.

The calculations are split up into neurons. These neurons are organized in different layers. Neurons in a same layer are independent from each other but depend on the same neurons from the previous layer. A single neuron of an artificial network can take multiple inputs \( x_1, x_2, x_3, \ldots \) and multiplies each input with a specific weight \( w_1, w_2, w_3, \ldots \) These weighted inputs are summed up to calculate the logit of the neuron. In many cases a bias value is added to the logit.

\[
\text{logit} = \text{bias} + \sum_{i=0}^{n} w_i x_i
\]  

(2.1)

The logit is then fed into a function \( f \) to calculate the output of the neuron \( y \).
\[ f(\text{logit}) = y \] (2.2)

If the input and the weights are represented as a vector \( \mathbf{x} \) and \( \mathbf{w} \) we then can re-express the formulation of the output \( y \) with some simple vector algebra. The dot product is per definition the sum of all weighted inputs. With the bias added and calculated the output of the function it satisfies the definition from above.

\[ f(\mathbf{x} \cdot \mathbf{w} + \text{bias}) = y \] (2.3)

Formulating the calculations of a network as vector multiplications is crucial for the further calculations.\[7\] A schematically representation of a neuron and its inputs from the previous layer can be seen in figure 2.1.

Hooking up the input data with different neurons and these to the output nodes makes an artificial neural network. The network itself is considered to be in layers. So the input will be the first layer represented by a vector. The entries of the vector will be forwarded as inputs for the first layer of neurons, the second layer in the network. These neurons will calculate their output, each one as described above. Those outputs will be forwarded to the next layer of neurons and so to the last layer the output layer. The output, just as the input, is again a vectorized representation. In a fully connected network every input is taken as input for the second layer and every output of the second layer as input for each neuron in the third one and so on including the output layer. The layers between the input layer and the output layer are called hidden layers. The number of neurons in each layer can vary and it is usual that hidden layers have less neurons than the input layer.\[7\]

### 2.1.2 Convolutional Neural Networks

In the year 2001 Paul Viola and Michael Jones published a paper highlighting a computer vision problem. To tell if on a randomly selected picture was a human face.\[8\] For humans this task is very simple and intuitive but for computers this is a lot harder. Trying to do this with a network as described before is very challenging. If the raw data is given in form of a vector as input the problem is, that the position of the face can vary from picture to picture. In some cases the face will be in the upper left corner and on the image it might be in the lower right. The size of the face might change as well. If all that information of the
Figure 2.1: This is a short example of two layers in a multilayer perceptron. The outputs of the layer n-1 are processed with the corresponding weights, represented by the arrows and then fed as input with an optional bias to the new layer. For a fully connected network every neuron in layer n-1 has a weight for each neuron in layer n.

picture is forwarded to the network the noise with all the irrelevant information that is fed to it is too large for the network to actually be able to learn something useful.\cite{7}

In the algorithm of Viola and Jones they focused on the different light intensities in different regions of the face. With some combined intensity detectors and a machine learning algorithm (boosting) they actually achieved some very good results for the time.

The importance to see in this problem is that in some cases we need to focus on some patterns and information that is hidden somewhere in our data and that there is a lot of noise that might be of less interest to our goal.

In 2012 Alex Krizhevsky won the annual Image Net challenge with the AlexNet presented in his paper.\cite{9} The AlexNet was a pioneer with the convolutional neural networks. This paper achieved impressive results and as Nikhil Buduma wrote in his book: "(...) crushed 50 years of traditional computer vision research (...)."\cite{7} This approach seems to have changed the field of computer vision.

The first part to analyze is the filter. In their paper Viola and Jones they described how the shade on the face was important and how they focused on that when finding faces. The region under the eyes might be a little lighter as well as the whole nose. This is because of the shape of a regular face and the shades that are thrown. So a lighter vertical line, that might represent a nose, is called a feature and a filter is a feature detector over a given image.
Figure 2.2: Two filters searching for two features in a sample image. The grey pixels stand for a value of 0 and the black stand for a value of 1. In every possible position of the kernel it is evaluated if the values in the image match the values of the kernel. If this holds true the output value is 1 and 0 otherwise. This again is represented with black and grey Pixels. Adapted from:[7]
In the figure 2.2 the two small $2 \times 2$ tables are filters. The filter on the right searches for a vertical line in the top image. The vertical line is in this case the feature that is being searched for with the given filter. For every possible position of the filter in the image a box in the feature map is created. If the position matches the filter the box in the feature map is black and white otherwise. The feature map for both filters is given in the same figure. "This operation is called a convolution."[7]

Colored images do have three values representing the RGB scopes. These inputs are not restricted to two dimensions but do have three and for those kind of inputs the dimensions of the filters need to be changed as well. The images have now three slices, each one slice for red, green and blue values. The filters now do need three slices as well. This means that the shape of each filter is adapted such that the output still has the same same shape. Each filter creates a new slice of the output. Each slice is called a feature map. Reconsidering figure 2.2 as a color input the dimensions would change as follows.

Input: $8 \times 8 \times 3$

Filter: $2 \times 2 \times 3$

Output with two filters: $7 \times 7 \times 2$

The exact shape of the output may actually vary if some parameters are changed.

- The spatial extent, height and width, of a filter.
- The stride of the filter meaning how many lie between two evaluations. If this value is 1 it jumps into every possible position just as in the examples before.
- Zero padding is making the input larger by adding a circle around the input filled with 0. So if padding = 1 the $8 \times 8 \times 3$ will be $9 \times 9 \times 3$ input with 3 circles of 0s, one around each slice. The padding can be chosen in a way that the output size is equal to the input.
- The number of filters decides the size of the depth (third dimension) of the output.

And the exact formulas of the output are:

- The width of the output can be calculated with the following formula:
  \[ width_{out} = \frac{width_{in} - spacial\ extent_{width} + 2 \times padding}{stride} + 1 \] (2.4)

- The height of the output can be calculated with the following formula:
  \[ height_{out} = \frac{height_{in} - spacial\ extent_{height} + 2 \times padding}{stride} + 1 \] (2.5)

- The depth of the output simply corresponds to the number of filters as each filter generates one slice.\[2\]
  \[ depth_{out} = \text{number of filters} \] (2.6)
2.1.3 Learning from Data

Activation Function

All the vector multiplications are linear functions and are a strong limitation to a network. It can be shown that a network in one of the structures we’ve defined above can be expressed as a network with no hidden layers. To add additional complexity to a network different nonlinear activation functions have been implemented. This has been mentioned in equation \(2.2\), the activation function takes the logit as input and computes the output of the neuron. Activation functions are used in the same way in convolutional networks.\(^7\) The sigmoid function is one possibility of a nonlinear activation function.

\[
f(\text{logit}) = \frac{1}{1 + e^{-\text{logit}}}
\]  

When the logit is very small the output is close to zero and if the logit is large the output is close to one.

![Sigmoid Function](image)

Figure 2.3: Plot of the Sigmoid function

The ReLU, short for restricted linear unit is another example of nonlinear function.

\[
f(\text{logit}) = max(0, \text{logit})
\]  

However Alex Krizhevsky et al. described a normalization for the output on each position for each filter of the convolution. This was used in their paper on some layers instead of an activation function. For every filter \(i\) in the position
Figure 2.4: Plot of the ReLU function

$(x, y)$ the output $b^i_{x,y}$ is computed with the equation:

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

(2.9)

where $a^i_{x,y}$ is the logit of the filter $i$ at the position $(x, y)$. $k, n, \alpha$ and $\beta$ are constants. $N$ is the total number of filters. This is called the local response normalization.

Loss Function

Now that the mechanics of a network are defined it is crucial to understand how these mechanics are used to solve machine learning problems. To this point it is known how a single example from a dataset is handled to compute an output. Each example from the dataset is a collection of features. These features are typically described in a vector in $\mathbb{R}^n$. For an image the feature values would be the values of each pixel. \[1\]

In a supervised learning algorithm the algorithm takes not only the examples from a dataset but each example comes with a label or target. The aim of a supervised algorithm is to predict the label of an example given its vector $x$. Usually this can be defined statistically by predicting the probability of a label $y$ given the random vector $x$, $p(x \mid y)$. \[1\]

In an unsupervised learning algorithm the algorithm tries to learn useful properties or structures in a given dataset. This usually is done by learning the probability distribution of the dataset. \[1\]

Defining a loss function is a very important part of the machine learning. There are several terms that actually refer to the same function, some of them would be cost function and objective function. In a supervised learning algorithm the data comes with a target. And the output of the network should be
the same as the target such that it is possible to subtract one from the other. This typically is a n-dimensional vector. On a classification task each entry of the vector would be a class or category. [10]

To define the loss of the algorithm a metrics is needed to define how far from the desired result the outcome actually is. To recap the nomenclature the outcome of the network is \( y(x) \), where \( x \) is the input, the desired output is \( a_x \). \( \theta \) stands for the learn able parameters of the network and \( n \) is the size of the dataset. In supervised tasks the desired output corresponds to the target and in unsupervised algorithms the desired output needs to be defined. The costs or loss \( C \) depending on the weights and biases of the network can look like: [10]

\[
C(\theta) = \frac{1}{2} \times \sum_{i=0}^{n} \| y(x_i) - a_{x_i} \|_2^2
\] (2.10)

The subtraction with the desired output \( a_x \) from the actual output \( y(x) \) with the euclidean norm, which makes that all errors added are non-negative, can be interpreted as the distance from both points in a n-dimensional space. Minimizing the loss function will bring the output \( y(x) \) closer to the desired output \( a_x \) which was the main goal of the algorithm.

This cost function is known as the \textit{quadratic cost function} or sometimes as well known as the \textit{mean squared error} or just \textit{MSE}.

**Softmax**

The softmax function may be seen as a generalization of the sigmoid function, which can be used for a probability of a single variable. Any time expecting to represent a probability distribution softmax may be an option. Therefore softmax is often used in a classification task as last activation function, to represent the probability over the given classes. For a distribution all values need to be between 0 and 1 and the sum of all values needs to be equal to 1. All of these values stored in a vector will be the input of the softmax activation, while other activation functions take one element as input the softmax needs to have the whole vector as input. For this vector \( z \) with entries \( z_i \) with a total of \( n \) vector entries the softmax is calculated with the formula:

\[
\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{n} e^{z_j}}
\] (2.11)

This gives a probability distribution over the vector \( z \).

**Gradient Descent**

To find weights and biases that have a high chance to compute a desired output for a given input the cost function needs to be minimized. Given the cost function \( C : \mathbb{R}^n \rightarrow \mathbb{R} \) the gradient \( \nabla C \) of this function is a vector in \( \mathbb{R}^n \) with the partial derivatives as values. The gradient relates the changes in the variables of a network to the output of \( C \). [10]

This relation is used to define an iterative algorithm to approach step by step at least a local minimum of a function. The update rule for this iteration is as follows:
\[ v_{m+1} = v_m - \lambda \times \nabla C \]  \hspace{1cm} (2.12)

\( v \) are the variables that were used to compute the costs for our learning algorithm, they are straight forward the weights and biases of the network, \( m \) is used as a variable for the current step and \( \nabla C \) is the gradient of the cost function. \( \lambda \) is a small positive number used to alter the size of a single step. This is also known as the learning rate. With this update rule a minimum for the cost function is approached. \( [10] \)

To calculate the loss there are three variants that differ from each other by how much data is taken into account. \( [11] \)

- **Batch Gradient Descent** computes the loss for the whole dataset. In this case the computed gradient is complete and will actually alter the variables towards the best possible direction for an improvement in the cost function. It takes a long time to compute the gradient and the network is not learning for a long time of computation, specially if the dataset doesn’t fit in the memory. \( [11] \)

- **Stochastic Gradient Descent** computes the loss after each sample from the dataset and updates the parameters. This computation is very light and the weights are constantly updated but the direction in which the updates happen fluctuates a lot and are not always an improvement for the costs of the dataset. \( [11] \)

- **Mini Batch Gradient Descent** takes the best part of both worlds. At random a little batch with samples from the dataset is extracted and the loss is calculated for this batch. The batch now represents the dataset and will compute a gradient that is closer to the actual gradient from Batch Gradient Descent. This is the typical choice for updating the weights and biases. \( [11] \)

**Back Propagation**

The process of a sample going through a network and therefore calculating the output is called forward propagation. After calculating the output and the cost this information is back propagated through the network for the calculation of the gradient. This algorithm is called back propagation. \( [1] \)

The back propagation algorithm takes benefit from the chain rule of calculus. This rule states that given

\[ y = g(x) \]

and

\[ z = f(y) = f(g(x)) \]  \hspace{1cm} (2.13)

for differentiable \( f \) and \( g \) the derivative for \( z \) with respect to \( x \) can be calculated as

\[ \frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx} \]  \hspace{1cm} (2.14)

this holds true for the partial derivatives as well
\[
\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}.
\]

Given a function \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) with \( f_1, f_2, ..., f_m \) differentiable component functions then the matrix of all \( m \) component functions derived with respect to all single variables \( x_1, x_2, ..., x_n \) is the Jacobian matrix.

\[
\text{Jacobian} = \begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n}
\end{bmatrix}
\]

With the Jacobian matrix \( J \) and gradients the equation (2.15) can be rewritten as

\[
\nabla_x z = J^T g \times \nabla_y z
\]

To calculate the gradient of a variable the back propagation algorithm uses the Jacobian matrix with a previous gradient and multiplies them.\(^1\)

It is crucial to note that the layers of neurons with their weights and biases are such encapsulated functions and therefore this is used to calculate the gradient which afterwards is used for stochastic gradient descent.

**Max Pooling**

After calculating the output of a convolution there might be a down sampling operation on the feature maps. The output number of these feature maps is not changed but the size of each feature map is reduced. A kernel is a small window that is moved over the feature map in the same way a filter was moved over the input. The stride with these kernels typically is the same as their width and height. For each position a kernel, that is moved over the feature map, takes, only the maximum value is taken into account and stored in the new feature map. A \( 2 \times 2 \) kernel will half the width and the height of each feature map. An alternative of taking the maximum from each kernel would be calculating the average. This would then be called average pooling.\(^{12}\)

**Overview of the Mechanics**

A network is a defined mathematical operation with different variables generating an output from a given input. These operations are a combination of linear vector multiplications and a nonlinear function also known as the activation function. For each input an output is generated and in some form. To measure the failure and success of a network a loss function is defined. The variables of a network are iteratively updated using a method minimizing the loss function.

**2.2 Improving the Learning Algorithm**

Some common problems and challenges should be considered when training a network. For some of them experiences have shown solutions.
2.2.1 Overfitting and Underfitting

A neural network with a large number of learnable weights and biases will be able to describe a large dataset nearly perfectly without capturing any underlying patterns. In this case the network will work for known data but predicting something from new unseen data will end in a rather random performance. The underlying patterns help a good network to be able to handle unknown data. Describing available data correctly doesn’t make it a good model yet. [10]

The true test of a model is its ability to make predictions in situations it hasn’t been exposed to before. [10]

Specially with small data sets the problem of overfitting will become urgent, specially when combined with large neural networks. Luckily when analyzing the learning of a neural network this effect becomes visible in different numbers. In addition to the dataset for the learning a small subset of similar data representing a probe of the dataset should be defined. This is called the test set. The network itself should never learn with this data. After some number of epochs the values for the dataset accuracy will still improve while the accuracy on the test set is stagnating and will eventually begin decreasing again. This is a strong sign for overfitting. Additionally after the same amount of epochs the costs for the dataset will still be decreasing in contrast to those of the test set where it will be increasing. From that moment on where the costs for the test set are increasing the learning effect is no longer on general data but already to specific on the given dataset and thus irrelevant for a general model. [10]

The opposite case would be stopping the learning process to early. This would be the case if the learning process is aborted while the costs for the test set are still sinking as well as the accuracy for the test improving.

2.2.2 Normalization

Data normalization is the procedure of giving the data zero mean and unit variance without losing any information as it is a bijective transformation. This helps the network to converge faster and has an impact on the overall accuracy. [12]

2.3 Transfer Learning

The main idea behind transfer learning is that a network learns its parameters from a first task and afterwards some of these parameters or all parameters are transferred to an other network for further use. Using pre-trained networks can be helpful when data is not complete. But also calculating all the weights and biases takes a lot of computation power and time. Depending on the task similarity and the test set similarity it may make sense to freeze all the parameters or it might be better to retrain them all but start from a pre-trained model. [12]

To show that a unsupervised pre-training was effective a lot of researchers do show that with frozen parameters in the first layers it was possible to achieve good results on a well known supervised task.
Chapter 3

Prior Work

3.1 Supervised Algorithms

3.1.1 ImageNet Classification with Deep Convolutional Neural Networks

In the year 2012 there was a paper\[9\] published by Alex Krizhevsky et al. which was a breakthrough for the state-of-the-art procedures in machine learning. The architecture won the ImageNet Large Scale Visual Recognition Challenge in the same year. The challenge consists of predicting the correct labels of some images given a labeled dataset with about 15 million images to train. This Dataset is called ImageNet\[13\]. This breakthrough augmented the interest in deep learning drastically.\[12\]

The architecture looks as figure 3.1. The first convolutional layer consists of 96 filters, $11 \times 11 \times 3$ in size. The max pooling kernels are $3 \times 3$ dimensional and are moved with a stride of 2. The height and width of the convolution filters is reduced in deeper layers, in the second layer the expansions among the first two dimensions are $5 \times 5$ and $3 \times 3$ in the third, fourth and fifth layer. The number of filters goes in the second layer to 256, 384 in the third and fourth and 256 in the fifth layer. The fully connected layers have 4096 neurons each.\[9\]

This Architecture is known as AlexNet.

3.1.2 Very Deep Convolutional Networks for Large-Scale Image Recognition

Over the past few years a lot of new architectures came to light and some improved the results from AlexNet. A lot of them were inspired by the AlexNet architecture. In 2015 Karen Simonyan et al. published a paper \[14\] showing that the depth had a great impact on the performance of a network. In general they said that adding layers and complexity would be beneficial for the performance and they showed that they could achieve better results with networks similar to AlexNet when they made them deeper. These performed better that the original.\[14\]
3.1.3 Network in Network

In the year 2014 Min Lin et al. published a paper [15] with a new deep network structure called Network in Network. They described classic convolutional neuronal networks consisting of alternately convolutional and pooling layers and a multilayer perceptron at the end of it. Network in Network changed the structure by replacing the classic convolution layers by mlpconv (mlp standing for multilayer perceptron and conv for convolution). In figure 3.2 an overview of such a mlpconv can be schematically seen or in figure 3.3 as it was described in the original paper.

Three of these blocks of convolutions would make a simple Network in Network structure. [15]
3.2 Unsupervised Algorithms

3.2.1 Unsupervised Representation Learning by Predicting Image Rotations

In 2018 a novel formulation for unsupervised learning technique was presented by Spiros Gidaris et al. and described in their paper. The algorithm takes images and rotates them before they are fed into the network. The task of the network is to predict the rotations of the given image. After training the network the first few layers are taken and given to a new network which afterwards tries to learn a supervised classification task without updating the first few layers. Their work came very close to an accuracy achieved with a supervised task without any fixed layers. This means that their algorithm learned to extract useful information from the dataset and that this information is useful for other tasks as well. This can be seen in their results table.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised NIN</td>
<td>92.80</td>
</tr>
<tr>
<td>Random Init. + conv</td>
<td>72.50</td>
</tr>
<tr>
<td>(Ours) RotNet + non-linear</td>
<td>89.06</td>
</tr>
<tr>
<td>(Ours) RotNet + conv</td>
<td>91.16</td>
</tr>
<tr>
<td>(Ours) RotNet + non-linear (fine-tuned)</td>
<td>91.73</td>
</tr>
<tr>
<td>(Ours) RotNet + conv (fine-tuned)</td>
<td>92.17</td>
</tr>
<tr>
<td>Roto-Scat + SVM Oyallon &amp; Mallat (2015)</td>
<td>82.3</td>
</tr>
<tr>
<td>ExemplarCNN Dosovitskiy et al. (2014)</td>
<td>84.3</td>
</tr>
<tr>
<td>DCGAN Radford et al. (2015)</td>
<td>82.8</td>
</tr>
<tr>
<td>Scattering Oyallon et al. (2017)</td>
<td>84.7</td>
</tr>
</tbody>
</table>

Figure 3.4: The results they show in their paper on the CIFAR10 dataset. Taken from: [6]
3.2.2 Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles

Mehdi Noroozi et al. presented a paper [5] in the year 2017 for another self-supervised algorithm. Their main idea is to cut an image into pieces and train a network to reconstruct the original positions of the cut pieces. Afterwards the first layers are again transferred to another network to evaluate the learned features. Their approach has given respectable results but was beaten by the idea of Spiros Gidaris et al. with their trained network predicting image rotations.

3.2.3 Unsupervised Learning of Visual Representations using Videos

A different source of information did Xiaolong Wang et al. take. In their paper [16] the input information is a video and the algorithm learns the changes within a patch of different images. This idea uses a cost function presented by J. Wang et al. [17]. Their idea takes no labeled data and is therefore as well unsupervised and was inspired by the evolution of a new born.

3.2.4 Unsupervised Visual Representation Learning by Context Prediction

A very similar idea to the jigsaw puzzles had Carl Doersch et al. in his publication [18]. Instead of solving the full exact puzzle the algorithm takes two input images and learns the position of the second one regarding the first one as center and eight possible positions aligned around for the second one. From this point a classification loss function is defined.

3.2.5 Colorful Image Colorization

An other very interesting idea was followed by Richard Zhang et al. [19]. They fed a network a grey scale image and let the network do the coloring. The difference from the ground truth to the predicted color is used to formulate a loss function. To ensure that the output images were colorful some measures were taken. The first one was not to simply formulate a euclidean loss function but make rather a multinomial distribution. The euclidean norm would only take the average of all possible outputs and specially colorful things would have stayed in a monotone rather boring color.

3.2.6 Adversarial Feature Learning

In a generative adversarial network typically two networks try to outsmart each other. While one is being trained to learn to differ original data from generated data the other network learns how to make better samples to fool the other network. The loss function comes from the classification into real data or generated data. Now one network minimizes the loss and the other maximizes the same loss. This idea was taken by Donahue et al. [20] to create a new form and learn some useful features. This new form takes three networks and while 2 want to minimize the loss one tries to maximize it.
3.2.7 Unsupervised Learning by Predicting Noise

Piotr Bojanowski et al. wrote a paper [21] on a model where they trained a network to predict noise as targets with their own elaborated system. The labels are generated and not given with the dataset. Because of the loss function this algorithm has some similarities with a clustering algorithm.
Chapter 4

The Fisheye RotNet

4.1 Method

4.1.1 First Thoughts

When visualizing the filters of the first layer in many trained networks it becomes obvious that these layers search for edges and other basic shapes in a given image. So it came to my mind that to enforce the learning of such filters it might be helpful to alter the edges in an image in a given way and let the network predict if the image has been altered or not. I had the images of a fisheye lens in mind, these do bend the straight lines into curves in an image. It should not be too challenging for a network to actually learn if a picture was taken with or without this effect.

4.1.2 The Fish-eye Lens

The original image is positioned in a 3 dimensional space parallel to the $x$ and $y$ plane. From there it is projected onto the unit sphere w.r.t. the Euclidean norm. This projection is then again projected onto the $x$ and $y$ plane. This creates an effect which is actually similar to a fisheye lens.

The projection to the unit sphere is a projection towards the center of the sphere. This causes that the borders of the image are no longer straight lines when they are down projected to the final plane.

Images with a larger distance to the $x$ and $y$ plane have a smaller effect and are smaller on the sphere (if they had the same size as original images). To avoid black or white semicircles on the image the corners are cropped.

4.1.3 Mathematics of fisheye

As can be seen in the figure 4.1 this algorithm can be divided into two simple calculations.

Projection onto unit sphere

The image is always centered on top of the unit sphere and parallel to the $x$ and $y$ plane. This means that the $z$ axis is its normal vector and goes through
the middle of the image. The size of the image was fixed to 2 so it would range from \(-1\) to 1 for \(x\) and \(y\) values. Therefore a Pixel \(p\) of an image with \(n \times n\) pixels has following position in \(\mathbb{R}^3\):

\[
x_{\text{original}} = 2 \ast \left(\frac{p_x}{n-1} - 0.5\right) \tag{4.1}
\]

\[
y_{\text{original}} = 2 \ast \left(\frac{p_y}{n-1} - 0.5\right) \tag{4.2}
\]

\[
z_{\text{original}} = z \tag{4.3}
\]

The values \(p_x\) and \(p_y\) represent the entry of the pixel in the image. So Pixel \((5,3)\) will have \(p_x = 5\) and \(p_y = 3\). The range of values for both is \(\in \mathbb{N}\) and in \([0,1,...,n-1]\). The value \(z \in \mathbb{R}\) represents the height of the image and decides how strong the effect will be.

To compute the positions on the sphere the Euclidean distance \(d\) to the center of the sphere is calculated and the original position is divided by the distance.

\[
d = \sqrt{x_{\text{original}}^2 + y_{\text{original}}^2 + z_{\text{original}}^2} \tag{4.4}
\]

\[
x_{\text{sphere}} = \frac{x_{\text{original}}}{d} \tag{4.5}
\]

\[
y_{\text{sphere}} = \frac{y_{\text{original}}}{d} \tag{4.6}
\]

\[
z_{\text{sphere}} = \frac{z_{\text{original}}}{d} \tag{4.7}
\]

Like this a Position on the sphere for a given pixel in an image is calculated.

**Projection to the \(x\) and \(y\) plane**

For the relation \(\mathbb{R}^3 \rightarrow \mathbb{R}^2\) the \(z\text{\text{sphere}}\) value is simply ignored.

\[
x_{\text{final plane}} = x_{\text{sphere}} \tag{4.8}
\]
This outputs an image which no longer has straight borders. To get an image with straight borders the corners are cropped off. Figure (a) from 4.2 shows an original image from imagenet while (b) from 4.2 shows a processed image.

**Interpolation**

After processing an image to have a fisheye like effect the positions of the original image can be calculated but these most likely won’t line up with the position of the pixels in the new image. To calculate the values of the new pixels an interpolation is needed. Three different types were taken into account and one is chosen at random.

- **Nearest Neighbor**, the value of a pixel takes the value of the original pixel that is projected nearest to it.
- **Bilinear**, from the surrounding pixels a weighted mean is calculated where the weights correspond to the distance from the pixel.
- **Trilinear**, same as bilinear but in three dimensional space \[22\]
4.2 From four to eight classes

The original Rotnet worked with a variable amount of classes. This paper [6] is the base for my thesis. They trained a network to predict the image rotations of an image and achieved respectable results. Square images can be rotated 4 times such that no more computation and cropping is necessary to get the same dimensions as the original image. These rotations come with the angles $0\degree$, $90\degree$, $180\degree$ and $270\degree$. Applying a fisheye effect on each one of these doubles the number of classes. The network now learns to predict if a rotation has been done and if the rotation was combined with a fisheye effect.

4.3 Building Blocks

4.3.1 Unsupervised Algorithm

For the CIFAR 10 Dataset the architecture was a Network in Network model. This was previously done already by the RotNet paper [6]. To keep the numbers in a state of maximum comparability between the architectures only few things were changed. The training was done with a number of 4 mlpconv layers and a
linear classifier. The results from the RotNet paper have been reproduced with this configuration.

The loss function used to calculate the error is the so-called cross entropy loss

$$loss(X_i, \theta) = -\frac{1}{K} \sum_{y=1}^{K} \log(F^y(g(X_i|y)|\theta))$$  \hspace{1cm} (4.10)

$X_i$ stands for the sample and $\theta$ are the learnable parameters of the network. $K$ are the number of transformations that are computed (for four rotations this means $K = 4$ e.g.). $g(X_i|\theta)$ is the transformation $y$ applied to the sample $X_i$ and $F^y$ is the output of the model.

### 4.3.2 Supervised

After learning the features from a self-supervised training, the first layers of the trained network are going to be transferred to a new network which then will be trained with a supervised classification task without updating the migrated layers. After doing this it is important to show that after learning the network performs better than with random initialized layers that are not updated. If this condition holds true the patterns learned from the unsupervised algorithm are useful. If the networks are alike we can further compare the usability of learned features from different unsupervised tasks.

The best results were achieved when having the two first layers of the network for CIFAR 10 supervision tasks and are those presented in this paper. On top of the pretrained layers comes a mlpconv layer and a linear layer which is used to learn the categorizations of CIFAR 10.
Chapter 5

Results

5.1 Performances

To show the relevance and efficiency of this implementation different experiments were performed. Most of the experiments were taken on the CIFAR10 Dataset, the results and the efficiency on this dataset made it very suitable for a variety of experiments.

5.1.1 Height of the Image

When projecting the image on to the unit sphere the height defines how strong the distortion is and therefore has a great impact on the performance of the features and the quality of them. When a height of 0.1 is chosen the results were not useful. When choosing a height of about 2 the best results were achieved. Afterwards the distortions were so small that the network began to focus on details that weren’t useful for later use in classification tasks.

5.1.2 Pure Fisheye

When comparing only fisheye effects it can be seen that the effect of the strength of the fisheye effect has a much stronger effect than it has when combined with the fisheye. In figure 5.1 this effect can be easily seen. It needs to be mentioned that this algorithm alone performs pretty badly and a random initialization according to [6] has a higher performance.

5.1.3 Fisheye combined with RotNet

For improvement of the performance both algorithms combined during pre-training get the following results for the transferred classification task. Figure 5.2 shows the validation curve for differently pretrained models. The strongest deformation got the worst results and a projection to the unit sphere from a height of about 2 gave the best results, afterwards the results would get worse again. While a lot of configurations would be detaining the learning algorithm the best results were in fact a small improvement to the original algorithm[6] for the CIFAR 10 Dataset. The clear bumps that can be seen in the graph are
5.1.4 Flip combined with RotNet

Another implementation was to combine a Flip transformation instead of a fisheye effect. The results from this implementation were comparable with the results from RotNet and did not show an improvement.
5.1.5 Store as .jpg and Reload Files

To make sure that the network doesn’t realize if an image was loaded from a .jpg file or if it was computed with the interpolation there were some experiments run where all the computed images were stored again as .jpg when computed and reloaded afterwards. This didn’t show any effects on the learning procedure so this was again disregarded.

5.2 Output

When on top of the pretrained network with both rotation and fisheye effect a simple classification task with the CIFAR10 is tried to be learned the task performs slightly better than the results that are presented in the paper. The validation percentage without fine tuning is about 0.5% better than what they described.

5.2.1 Plots

Loss

The loss during the self supervised task shows an extremely high curve for the strongest deformation while all of the others show similar results. This means that the deformation of the image is so strong that the network has a lot more of trouble when trying to predict the rotations.

Figure 5.3: The train loss during fisheye and rotation task. The highest loss is achieved with the strongest deformation.
Figure 5.4: Some visualized kernels from an AlexNet like structure.

Kernels

Kernels from the first layer of an AlexaNet like structure with a learning on the Imagenet Dataset. The kernels have some clear similarity with the kernels presented in the original paper. This shows that this algorithm not only works on the CIFAR10 dataset but it might be a good option for other datasets as well.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis the RotNet algorithm \cite{6} has been analyzed and the respectable results they presented have been confirmed. From different sources some inspirations for additional research have been acquired and with them new calculations were elaborated and implemented. In combination of the original idea and these new implementations a novel unsupervised task was created. The results were compared to the results in the original paper. The stand alone new implementations weren’t that efficient and couldn’t catch up to the original work. But in combination of the original with the fish eye effect it was possible to achieve a marginal amelioration of the results. Some other implementations were rather not so successful and might not be worthy to continue combining in this setting.

6.2 Future Work

It is a long way to go and a lot of ideas and experiments are still to come. RotNet has shown to be a very good performing task that could only be marginally improved by adding a fisheye effect to the algorithm. The fisheye effect alone did not have slightly as much impact on the transferred learning algorithm as the RotNet.

A lot of further experiments could be done of which some may really improve the algorithm. In order of transformations some spatial transformations as described in the paper of Søren Kaae Sønderby et al. \cite{23} could actually bring further improvement or maybe some color jittering, described in \cite{24} for the input images might do the trick. There are endless possibilities of how the algorithms might be improved and how they could be combined. It is important to keep up the work for future progress.
Bibliography


