1. INTRODUCTION

Most computer vision tasks are solved using machine learning. With machine learning, models are not directly constructed by experts, they are instead trained based on data. In the past few years deep learning algorithms have revolutionized machine learning. In many classification tasks deep learning has drastically surpassed previous state of the art results in classification accuracy [2].

In machine learning a linear increase in the amount of features (input parameters) exponentially increases the difficulty of training. Reducing the amount of features thus exponentially makes training easier. In the past, the solution was feature extraction; selecting features and constructing new features based on the existing features in the input. This leads to less features, and therefore is less difficult to train. Feature extraction is a special form of dimensionality reduction, which is the reduction of the amount of free variables in machine learning. Feature extraction is labor-intensive as the new features are constructed by hand. Constructed features essentially construct higher level representations of the lower level features [1].

The aim of deep learning is to create higher level representations of the data through the use of multiple layers of non linear operations [3]. These higher level representations are much more useful for classification than the basic features. The complete classifier, with all its components, form the machine learning architecture. In computer vision raw pixels form a poor feature for direct classification of data; pixels may be shifted, have noise or are displayed in different colors, but may still represent the same number. A deeper architecture such as a multi-layer neural network performs better in this task, as it is able to form a higher level representation of the data.

Initially deep learning research has been focused on unsupervised layerwise pretraining. Unsupervised layerwise pretraining essentially builds a neural network in layers. In each step a new layer is added to the neural network with less neurons than the previous layer. The layer is trained on unsupervised data, which means that the data is not annotated for a specific outcome. The goal of the layer is to perform a compression step with the lowest error rate. The error is calculated by performing the operation and reversing it and comparing the change in the data. Essentially through this layerwise training a higher level representation is build in the neural network. Subsequently unsupervised training is followed by a supervised training step to form the classifier. The advantage of pretraining is that it provides a good initialization of the neural network for supervised training, avoiding the supervised neural network from getting stuck in a bad local minima. When a neural network reaches a local minima the learning algorithm will not be able to improve the neural network.
beyond the state of the local minima [6].

Although direct supervised training of deep neural networks was deemed unfeasible due to the diminishing gradient flow as observed by Hochreiter et al. [12], it has been shown that with more powerful GPU implementations it is possible to directly train deep convolutional neural networks. GPU implementations of deep convolutional networks provide a forty times reduction in the training time of deep convolutional neural networks [5].

This paper focuses on deep convolutional neural networks trained using backpropagation. The key issue with training deep neural networks is learning a useful representation for the lower layers in the neural network, and letting the higher layers in the neural network do something useful with the output of the lower layers [6].

In recent years a lot of improvements have been made to the architectures of deep feedforward neural networks, drastically increasing the classification accuracy in many classification tasks. In this paper we investigate the architectures used in deep convolutional neural networks for computer vision. Analyze the role of the components in the neural network and what their effects are on training of the neural network. We also analyze the effects on neuron activations based on the selection of activation function. Finally we conclude how deep feedforward neural network training is affected by the architecture of the network.

2. METHODOLOGY

For the purpose of this work we look at applications of feedforward neural networks on the MNIST and Street View House Numbers (SVHN) dataset. A literature study has been conducted on the deep neural network architectures, specifically for the MNIST and SVHN dataset. Scopus and Google Scholar have been searched for papers on the MNIST and SVHN dataset. The top five papers achieving the highest classification accuracy on these datasets have been used to find previous state of the art results and related work on deep learning architectures. Any papers not using deep convolutional neural networks have been filtered.

Based on the architectures of the classifiers with the highest classification accuracy, an overview of the components used in deep convolutional neural networks architectures is presented. Through a literature study the overview of the architectural components was extended, as well as motivations for components have been added. The literature study is performed based on references found in the other literature study and by searching for related work on the components through Google Scholar.

Four neural networks with different activation functions have been trained using Pylearn2[10] to measure the activation values in the hidden layers of the neural networks. In the results the saturation levels for the different activation functions are compared.

3. BACKGROUND

Artificial neural networks are brain inspired networks used in machine learning applications. Neural networks are composed of neurons. Each neuron is connected to other neurons through adaptive weights. In the simplest form of a neural network, the feedforward neural networks, the neurons are arranged in layers. There are special layers for the input and output of the neural network. The input layer basically is a set of features connected by weights to the neurons in the first layer, and the output layer contains the output activations for the network, which gives the classification. The layers between the input and output layer are called hidden layers. In a feedforward neural network, layers are only connected to adjacent layers, and only with weights going to higher layers. See Figure 1 for an image of a simple neural network.

Currently many state of the art results in computer vision have been achieved using deep convolutional neural networks [14]. Convolutional neural networks were inspired by the workings of the visual cortex. In which cells are only sensitive to small regions of the receptive field. A convolutional neural network similarly considers spatial locality of the input. Which is advantageous in computer vision since closely related pixels are also likely to be related. In higher layers of the neural network the same pattern of spatial locality is used, neurons in this way each higher layer of the network receive input from increasingly larger regions of the input image (see section 5.1).

As with any other machine learning technique, neural networks are trained based on training data. There are many different algorithms for training neural networks. However it turns out that backpropagation, one of the simplest algorithms of training neural networks, is one of the best.

3.1 Backpropagation

Backpropagation is a technique to propagates errors in the neural network back through the feedforward architecture and to adapt the weights. Training a neural network with backpropagation is composed of two simple steps the feedforward and the backpropagation step. In the feedforward step a training case is classified using the current neural network. In the backpropagation step a classification error is computed and propagated back through the neural network. The weights are updated based on the error, learning rate and gradient of the activation. Backpropagation is the method used to achieve gradient ascent in neural networks.

Output layer error:
\[ \delta_j^{(l)} = a_j^{(l)} - y_j \]

Hidden layer error signal:
\[ \delta_i^{(l)} = (\theta^{(l)})^T \delta^{(l+1)}. * \Delta a_i^{(l)} \]

\( \theta^{(l)} \) weights to layer i in the neural network.
\( \delta^{(l)} \) back-propagated error signal used to update the activation values in layer i.
\( \Delta a_i^{(l)} \) gradients of the activation function in layer i.

3.2 Diminishing gradient flow

The method of training a neural network is important, as a neural network is a complex system with many local minima. The training through backpropagation is largely affected by the complex interactions between the neurons of the neural network. A key issue with backpropagation is the diminishing gradient flow (also called the long time lag or vanishing gradient problem) [12], which essential makes it harder to train the lower layers of a multi layer neural network. Neurons in lower layers of a neural network get their errors passed down from neurons in the higher layers. In the highest layer weights get updated directly based on the error in the output layer and the gradient of the activation function. The weights in the next layer get updated based on the error in the higher level layer. Effectively lagging behind on the errors on the higher level layers. This causes either the error rate decays or explodes exponentially to the lower layers [12].
The effect of diminishing gradients can be illustrated using the sigmoid activation function, see Figure 2. A neuron with the sigmoid activation function may get saturated when nearing an activation value of zero or one. In a saturated neuron the gradient of the activation function $(\Delta a^{(i)})$ approximates zero, it diminishes the gradient flow to the lower layers in the neural network. Lowering the error signal $(\delta^{(i)})$ passed by gradient descent to layer $i - 1$. The problem of diminishing gradient flow has motivated many design decisions in deep feedforward neural network architectures.

4. RELATED WORK
The general structure of the deep convolutional neural network was introduced in 1998 by LeCun [13]. His deep convolutional neural network architecture called LeNet is basically what is still being used today. It is an architecture composed of convolution layers interchanged with pooling layers (see section 5.3). Since then however a lot of improvement have been made to the individual components within the architecture. This paper expands on the work by LeCun et al. by providing an overview of the current state of the art components.

Glorot et al. studied the distribution of activation values in deep neural networks [7]. By monitoring the activation values of the neurons Glorot et al. found that less saturated neurons would allow neural networks to be trained deeper using supervised learning. Based on this research Glorot et al. later proposed the rectifier linear unit [8]. We also analyze the activation values in section 6.

5. COMPONENTS
A deep neural network using convolutional neural networks generally are applied with a specific set of components. Extensive research has been performed on many different kinds of components for feedforward neural networks. In this section the most important components will be summarized, their role in the neural network and the advantages and disadvantages of different components will be discussed.

5.1 Convolutional layer
Convolutional neural network models are used for supervised learning. In their ordinary form neural networks connect all neurons to all other neurons in the next layer of the neural network. Convolutional neural networks are different from ordinary neural networks in that neurons in each layer are only sparsely connected to the neurons in the next hidden layer. The neurons are connected to the next hidden layer based on their relative location in the neural network. In images this means that in the first hidden layer each neuron receives input vectors from small regions of the image. Additionally neurons are replicated such that the same features can be detected on different parts of the input image, see Figure 3. Reducing the amount of parameters to be learned by the neural network. Reducing the amount of parameters is important to reduce the training complexity [11].

5.2 Activation functions
The activation function in a neural network transforms the input value to the output value of the neuron. Two important factors determining the quality of an activation function: their ability to handle the diminishing gradient flow to the lower layers of the neural network (see section 3.2), and sparsity.

Most of the neurons in the human brain are not simultaneously activated, most of the time only a small portion of the neurons are active. Similarly neurons in a neural network should not always be activated. Glorot et al. [8] listed the advantages of a sparse activation function. A sparse activation function has the property that for most of the input the amount of activated neurons is a relatively small.

Originally sigmoidal activation functions were most often used as the activation function in neural networks. Sigmoidal activation functions include the sigmoid and hyperbolic tangent (tanh) functions. Sigmoidal functions are bounded by a minimum and maximum value. The bounds of sigmoidal activation functions cause saturated neurons in the higher layers of the neural network. These saturated neurons cause diminishing gradient flow to the lower layers of the neural network, limiting the depth of the neural network that can be trained.

$$\text{sig}(x) = \frac{1}{1 + e^{-x}}$$

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

Alternatively rectifier linear units have recently been proposed as an activation function. Even though the sigmoid activation function was inspired by biology, the rectifier linear unit more closely resembles the biological activation function. Rectifier linear units in contrast to sigmoidal functions are unbound and can represent any non negative real value. The function also has good sparsity properties due to having a real zero activation value. Rectifier linear units are only bound by their minimum value, which
Figure 4. Activation function values

is zero. Consequently due to this property rectifier linear units suffer less from the diminished gradient flow [8].

\[ \text{rect}(x) = \max(0, x) \]

More recently a special kind of activation function was introduced to complement the use of dropout. A dropout layer randomly omits half of the activations. Dropout is explained in more detail in section 5.4. Maxout is an activation function specifically created to take advantage of the specific properties of dropout. Maxout does not have a sparsity property, and does not include a constant zero such as the rectifier linear unit. One property of dropout however, is that it causes sparsity in the distribution of the activations, therefore maxout does not need to be sparse [9]. The maxout activation function is designed to never have a gradient of zero, allowing the network using maxout and dropout to achieve a good approximation to model averaging. The maxout activation function works by multiplying the input vector by a matrix, which gives a vector. From this vector the maximum value is selected. In backpropagation only the weight of the connection from the maximum value is affected [9].

\[ \text{maxout}(x) = \max_{j \in [1,A]} z_{ij} \]

\[ z_{ij} = x^T W_{\cdot, ij} + b_{ij} \]

\( W \in \mathbb{R}^{d \times m \times k} \) and \( b \in \mathbb{R}^{m \times k} \) are parameters.

It was found that the maxout activation function suffers less from diminishing gradient flow. Therefore is able to train deeper networks [9].

5.3 Pooling

Pooling in general is a form of dimensionality reduction used in convolutional neural networks. Its goal is to throw away unnecessary information and only preserve the most critical information [4].

Due to the replication of weights in a convolutional neural network a feature may be detected across the input image. The detected features however are not invariant from the location of the input image. If a input image is shifted, the neuron detecting the feature is shifted as much. To make the detected features invariant from the location in the input image, pooling is used. It summarizes the output of multiple neurons in convolutional layers through a pooling function. It essentially takes nearby feature detectors and combines them into local or global 'bag of features' [4]. Pooling makes activations in a neural network less sensitive to the specific structure of the neural network [15]. It makes a network less sensitive to the exact location of the pixels.

Typical pooling functions are maximum and average, which are called max-pooling and average-pooling layers respectively when referred to as layers (subsampling layers are average-pooling layers). Additionally pooling layers may be spaced so that they take distinct separate inputs or may take in overlapping input which produces a more coarse coding [11].

Max-pooling as the name suggest basically returns the maximum value from the input, where average pooling takes the average. The main disadvantage of average pooling is that low or negative activations may downplay a higher activation value resulting in a near zero activation function even though the most important information was the high activation of one of the feature detectors. Especially when considering sparsity in the activation function, average pooling may be less effective and give low activation values. The main disadvantage of max-pooling is the whole disregard of the other values in the pooled layer. This is likely to over-fit the dataset rather quickly. When training over-fits the dataset training will stall, since the error signal passed through the neural network will diminish [15].

Zeiler et al. alternatively proposed stochastic max-pooling. A type of max-pooling specifically designed to work with the rectifier linear unit activation function. It randomly picks the activation value based on the normalized probability of the activation values. Each non zero activation value is assigned a probability, the activation value with the highest activation value having the highest probability. Then the activation value is picked at random from the non zero activation values based on the probability distribution. Experimental results has shown that it does not over-fit the dataset quickly and is able to continue training for a longer period [15].

5.4 Dropout/DropConnect

Neurons in a fixed architecture may come to rely on the output of specific neurons in the architecture. The neural network is trained to perform well on the training data. With test data however, the specific architecture may perform poorly as it has over-fit the dataset.

Dropout by Hinton et al. [11] has been shown to prevent over-fitting. It works by randomly omitting half of the neurons for each training case, see Figure 5. This omitting is done by setting the activation value to zero, neglecting the neuron temporarily. By doing this each neuron has to work with a random combination of its inputs instead of being trained on a fixed architecture.

Each neuron has to be able to adapt to the dropped activation values in the neural network. This makes the neurons more adaptive and less restricted to the existing architec-

Figure 5. Left: Dropout randomly sets activation values to 0. Right: DropConnect randomly sets weights to zero.
ture of the neural network. This approaches gives a more generalized results. Since dropout randomly omits neuron activations it is a form of ensemble learning: in a way it trains multiple random networks at a time.

DropConnect provides similar functionality to dropout: instead of randomly omitting half of the feature detectors it randomly omits the weights. The weights are omitted by setting the weights to zero, see Figure 5. Although it has been shown to further improve performance in many classification tasks, it has only been able to improve performance through model averaging of multiple DropConnect networks. Performance of a single DropConnect network has not been able to outperform a single Dropout model [14].

5.5 Output layer
The output layer of a classifier requires a specific property not required in other layers. It should form a probability distribution over the output classes, where the probabilities should sum to one.

Softmax classification layer
A softmax layer is used as the final classification layer of a neural network classifier. It is a linear classifier, as it uses the log probability distribution.

6. ANALYSIS
6.1 Neuron activations
Four different neural networks have been trained on the MNIST dataset, using the sigmoid, tanh, rectifier linear unit and the maxout activation functions. The neuron activations have been measured for each epoch and for each hidden layers in the neural networks.

The maxout neural network is composed of three maxout convolution layers interchanged with Max Pooling layers, the network is based on the network from the Maxout paper by Goodfellow et al. [9]. The other networks are composed of two convolution layers with one fully connected layer. Dropout is applied to the first hidden layer.

6.1.1 Environment
The experiments have been conducted using Pylearn2 [10]. It provides implementations for many of the state of the art feedforward neural network components, such as dropout and maxout. It also includes a fairly robust monitoring system, and out of the box support for the MNIST and SVHN dataset.

6.1.2 Results
See Figure 6 for the activation values of the hidden layers in the neural networks with the sigmoid, tanh and maxout activation function. The activation values are normalized to the same total number of activations per layer. See Figure 7 for the distribution of activations for the rectifier activation function.

Sigmoid.
From the graph can be seen that hidden layer 1 has many saturated neurons at activation value 0. Similarly the activation values in hidden layer 2 also peak at activation value 0. Hidden layer 0 barely has any saturated neurons, but has a lot of neurons near to their initial value.

Tanh.
Hidden layer 1 has a broad distribution of the activation values, where hidden layer 0 has a lot of values near their initial value. Hidden layer 2 has small peaks at the saturation values (-1, 1).

Rectifier.
The rectifier linear unit is designed to have many saturated neurons. A graph similar to Figure 6 would only show a peak at activation value zero. A graph is shown in Figure 7 with the percentage of saturated neurons in each hidden layer for each epoch.

Maxout.
In the maxout network the activation values are distributed in a gradually broader range in the higher layers. Hidden layers 0 only has a small range of activation values, where hidden layer 1 and 2 have an increasing range of activation values. As the maxout activation function does not have a clear saturation point it is impossible to distinguish the saturation in the hidden layers using this graph.

7. DISCUSSION
The saturated neurons in the higher layers of the neural network for the sigmoid and tanh activation function confirm the results found by Glorot et al. [7]. For the rectifier
linear unit function the distribution of the activation values also confirms earlier research[8], additionally from the graph can be seen that the rectified linear unit is very sparse. More noteworthy for the sigmoid, tanh and maxout activation functions are the activation values in hidden layer 0, which all peak around their initial value. The maxout activation function however has a much wider range of values which could provide an indication that maxout network is able to effectively learn a good representation for the lower layers in the neural network.

The maxout activation function and the rectifier linear unit have been fairly effective in handling the diminishing gradient flow in convolutional neural networks. Dropout has also made a significant contribution to the improvement of deep convolution neural network performance.

To improve on the state of the art, new components may be developed which better solve the problem of diminishing gradient flow. Intuitively one may also consider the problem of diminishing gradient flow to be a problem with gradient descent, which can be solved by designing a new algorithm which better trains the lower layers of the neural network.

8. CONCLUSION

Achieving state of the art performance in deep convolutional neural networks relies heavily on the application of specific techniques that make backpropagation work with deep neural networks. An overview of the state of the art components of convolutional neural networks has been given. The choice of activation function is important for the neural network to handle the diminishing gradient problem. Key to a good activation function is that it should cause sparsity and limit the diminishing gradient flow. Rectifier linear units are designed to cause sparsity, as well as limit the saturation of the neurons in the neural network. The maxout activation function was specifically designed to work with dropout, and to let dropout better approximate model averaging. Dropout is a model averaging technique which works by randomly omitting half of the feature detector. It is an effective regularizer and causes sparsity in the neural network. Pooling is used for dimensionality reduction, max-pooling is most often applied in state of the art classifiers. Other pooling functions may improve performance.

An analysis has been made of activation values using different neural network architectures. The analysis confirms earlier results on neuron saturation, no new observations could be made from the analysis.

9. FURTHER WORK

These results do not provide an answer on how to improve deep convolutional neural networks. The results further confirm that the maxout activation function finds a better representation for the lower layers of the neural network.

Further work could analyze the effects on activation values of a maxout network with too many layers, as well as analyze the gradient flow in these networks. This could provide clues to what issues still exist, and what limits there are to learning deep networks.

Interestingly both stochastic pooling and dropout contain a random component. It is unclear if stochastic pooling and dropout work together or that their function interferes. Further research could analyze the effects of combining multiple random components into deep convolutional neural networks.

10. REFERENCES


