

Derivation of a Convolutional Neural Network Based Dense Face Detector

February 16, 2018
Nikita Zozoulenko Na15b
nikita.zozoulenko@gmail.com
Katedralskolan Linköping
Supervisor: Rickard Engström
VT 2017 - HT 2018

Abstract

Abstract Here.....

In this paper the mathematical model behind the standard Artificial Neural Network and the Convolutional Neural Network is derived. A Convolutional Neural Network is then adapted to read handwritten digits and to make a cutting edge face detector, successfully detecting over 50 faces in a crowded scene.

Contents

1	Introduction	4
1.1	Background	4
1.2	Purpose	4
1.3	Problem statement	5
2	Metod	6
3	Resultat	7
3.1	Feed-forward neural networks	7
3.1.1	Tensors, indexing and notation	8
3.1.2	Forward propagation	8
3.1.3	Cost function	10
3.1.4	Gradient Descent	11
3.1.5	Backpropagation	12
3.1.6	Training neural networks	14
3.2	Konvolutionella neuala nätverk	15
3.2.1	Konvolutionslagret framåtpropagering	17
3.2.2	Konvolutionslagret bakåtpropagering	18
3.2.3	Aktiveringsfunktionslager framåtpropagering	20
3.2.4	Aktiveringsfunktionslager bakåtpropagering	20
3.2.5	Maxpoollagret framåtpropagation	21
3.2.6	Maxpoollagret bakåtpropagering	22
3.2.7	Batch Normalization framåtpropagering	23
3.2.8	Batch Normalization bakåtpropagering	24
3.2.9	Softmax framåtpropagering	27
3.2.10	Softmax bakåtpropagering	27
3.3	Problem Cases	28
3.3.1	Classification of handwritten digits	28
3.3.2	Dense Face Detection and Localization	30
4	Diskussion	32

1 Introduction

1.1 Background

Artificial neural networks, or more specifically convolutional neural networks, were popularized 2012 when the model was used to win the annual ImageNet Large-Scale Visual Recognition Challenge, beating all current machine learning models. Today they have achieved state of the art results in areas such as self-driving cars, image classification, object localization, automatic image annotation, semantic segmentation of objects in images and natural language processing. [1]

An Artificial neural network is a biologically inspired machine learning model that tries to replicate the way the brain in mammals functions. There are many different kinds of Artificial neural networks which vary in structure and architecture. The basic principle behind neural networks is that they are made out of a number of layers of neurons. The layers are built recursively such that the output of one layer is the input of the next layer. They are stacked on top of each other and the number of layers an Artificial neural network has is called its depth. How the neurons of the previous layer are connected to the neurons of the next layer depends on what type of Artificial neural network it is. [1]

1.2 Purpose

When I was first starting out in the field of deep learning I found there was a lack of fully derived explanations of the underlying mathematics behind artificial neural networks. Sources had either only explained one simple forward pass through the network, or had only derived the most simple case, ignoring the more complicated general case. The aim of this paper is to present a clear derivation of the general case for Feed-forward neural networks and Convolutional neural networks. Furthermore, the aim is to apply the derived model of the Convolutional neural network to two problems: Classifying handwritten digits and to detect and pinpoint the location of a variable number of human faces in an image.

1.3 Problem statement

What is a Feedforward neural network and Convolutional neural network? What is forward- and backpropagation and how is it derived in a Feedforward neural network and a Convolutional neural network? Can an Artificial neural network be trained to classify handwritten digits and detect a variable number of faces in images?

2 Metod

The majority of the paper and the derivations of the models are based on the course material from Stanford's "CS231n: Convolutional neural networks for Visual Recognition" course. Some advanced concepts were taken from the papers they were introduced in (e.g. Batch Normalization) and expanded to fit the general case of an arbitrary input to the model. The general case for the mathematical model of the Feedforward neural network and the Convolutional neural network were derived by hand and implemented in C++ with the linear algebra library Eigen and in Python in pure NumPy, a library for scientific computing. The derived partial derivatives required by the models were compared with their numerical approximations using the formal definition of a derivative.

When my own implementation of the Artificial neural networks became too computationally expensive and inefficient for the two problem cases classification of handwritten digits and dense face detection I implemented the models in Google's machine learning library TensorFlow and Facebook's GPU-accelerated tensor and dynamic neural network library PyTorch.

3 Resultat

3.1 Feed-forward neural networks

A feed-forward neural network is the most elementary version of an artificial neural network. As all varying kinds of neural networks, a feed-forward neural network is made out of a number of layers of neurons. The unique property of a feed-forward neural network is that all the neurons in a layer are connected to every neuron in the next layer (see figure 1). [1]

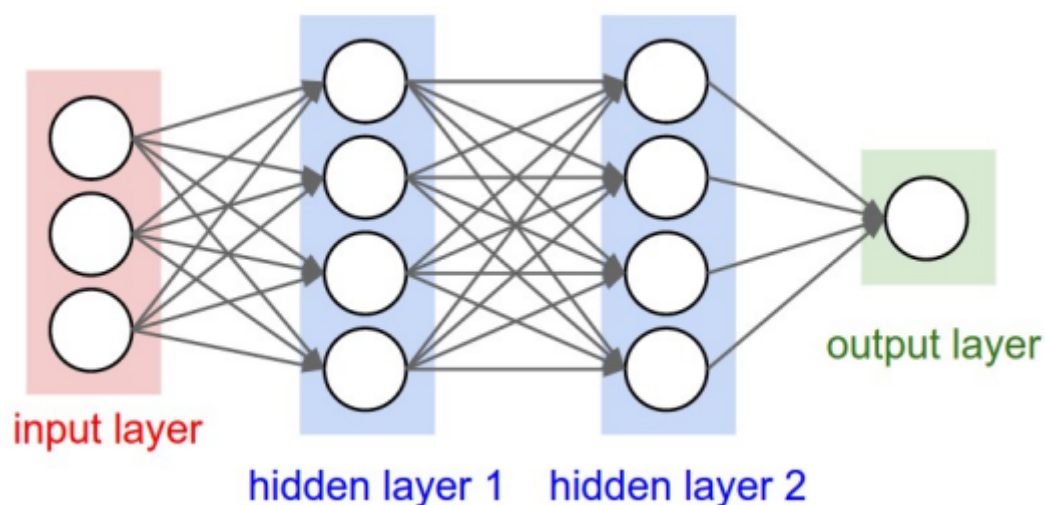


Figure 1: An illustration of a simple feed-forward neural network. It consists of 4 layers: 1 input layer (red), 2 hidden layers (blue) and 1 output layer (green). A circle represents a neuron. Every neuron in a layer is connected to all the neurons in the following layer, shown by the grey lines between the neurons. GLÖM INTE ATT REFERERA

A neuron is represented as a floating point decimal number. The value of a neuron is called its activation. Given an input of n data points the model wants to predict m different values where n and m are the number neurons in the input and output layers. Every data point or feature in the input are set as neurons in the input layer. For instance, if you want to predict a score given 12 different variables every single variable would correspond to one neuron in the input layer of size 12 and the score would correspond to a single neuron in the output layer of size 1. The value of the input neurons are transferred over to the next layer depending on the strength of the different connections every neuron has to the neurons in the next layer. This is repeated for every layer until the signal has reached the output layer. The process of propagating the value of the neurons from the input layer to the output layer is called forward propagation. [1]

The network is trained to predict correct values by optimizing the fixed

connections, also called weights, between all the layers in the neural network. [1]

3.1.1 Tensors, indexing and notation

A tensor is the generalization of vectors and matrices. Scalars are tensors of order 0. A tensor of order 1 is a vector $x \in \mathbb{R}^N$ and is a row vector with N elements. It can also be seen as a one-dimensional array. Matrices M are tensors of order 2 such that $M \in \mathbb{R}^{R \times N}$ and can be viewed as a vector of R elements where every element is another vector with N scalar elements. Matrices can also be seen as a two-dimensional array with RN elements. A tensor of order n is an n -dimensional array and is indexed by an n -tuple. For instance, a tensor $X \in \mathbb{R}^{R \times C \times H \times W}$ is indexed by the four-tuple (r, c, h, w) where $0 \leq r < R$, $0 \leq c < C$, $0 \leq h < H$ och $0 \leq w < W$. [1]

3.1.2 Forward propagation

In figure 1 only a single training example was worked on at a time. In practice a mini-batch of R training examples are propagated forward in a neural network at the same time. [1] [2]

Let M denote the number of layers in the neural network and l ; $0 \leq l < M$, one specific layer in the neural network. Let $N^{(l)}$ denote the number of neurons in layer l . The activation of layer l can be expressed as a tensor of order 2: $X^{(l)} \in \mathbb{R}^{R \times N^{(l)}}$ indexed by the two-tuple (r, i) where $0 \leq r < R$ and $0 \leq i < N$. In addition to the normal neurons a layer has, let $b^{(l)}$ be a bias neuron for the layer l (see figure 2). It is called a bias neuron because its value is independent to what input the neural network is given. [1] [2]

The weights expressing how strong the connection between the neurons of layer l and $l + 1$ are can also be expressed as a tensor of order 2. Let $W^{(l)} \in \mathbb{R}^{N^{(l+1)} \times N^{(l)}}$ such that the element $W_{j,i}^{(l)}$ is the strength of the connection between neuron $X_{ri}^{(l)}$ and $X_{rj}^{(l+1)}$ for arbitrary example r in the mini-batch. [1] [2]

To calculate the value of a neuron in layer $l + 1$ every neuron in layer l is multiplied by its corresponding weight in $W^{(l)}$ and summed together with the layers bias neuron. The sum is then put in a so called activation function f . The value of the activation function is the neurons activation in layer $l + 1$. [1] [2]

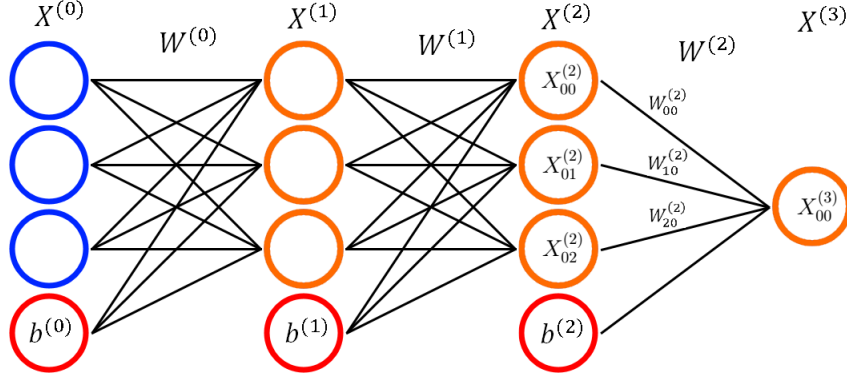


Figure 2: An example of a 4 layer feed-forward neural network. The input neurons are marked with blue. The bias neurons are marked with red. Black lines between neurons symbolize the weights between every pair of neurons between two layers.

Let $Z^{(l)} \in \mathbb{R}^{R \times N^{(l)}}$ be the value of each neuron in layer l before being put into the activation function. Given an input $X^{(0)}$ the forward propagation can be expressed recursively as: [1] [2]

$$Z_{rj}^{(l+1)} = b^{(l)} + \sum_{i=0}^{N^{(l)}-1} X_{r,i}^{(l)} W_{i,j}^{(l)} \quad (1)$$

$$X_{rj}^{(l+1)} = f(Z_{rj}^{(l+1)}) \quad (2)$$

The tensor of activations and weights are constructed in such as way that one forward pass through a single layer can be computed with a single dot product and having the bias term added to every element in the computed dot product. The activation function f is then applied element-wise on each neuron: [1] [2]

$$Z^{(l+1)} = X^{(l)} W^{(l)} + b^l \quad (3)$$

$$X^{(l+1)} = f(Z^{(l+1)}) \quad (4)$$

Common activation functions are Rectified Linear Units (ReLU), sigmoid (σ) and hyperbolic tangent (\tanh) and are defined by equation (5) - (7) (see figure 3). A non-linear activation function is chosen to enable the network to make use of non-linearities when learning to predict values. Without non-linear activation functions the whole model is equivalent to one large linear transformation of the input data. [1]

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (5)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (6)$$

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (7)$$

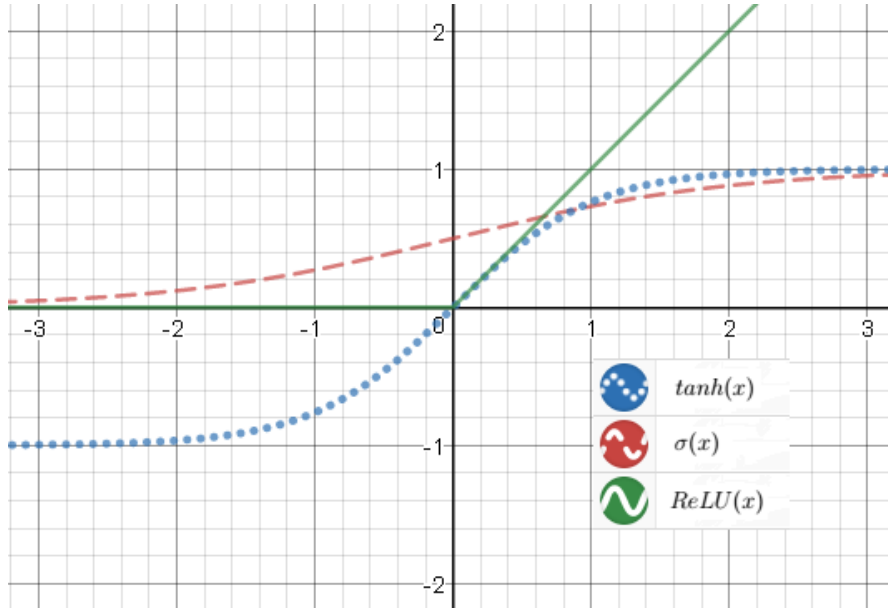


Figure 3: A graph of ReLU, σ och \tanh .

3.1.3 Cost function

Given an input X and a ground truth y the model wants to predict values \hat{y} which resembles the ground truth as closely as possible. It is achieved by defining a multivariate cost function $L(\theta; X, y)$ with the network's parameters θ (all the weights and biases) with respect to a single training example (X, y) . The cost function describes the quality of the prediction \hat{y} such that a lower cost represents a more accurate prediction. One way of defining the cost function is to use the mean squared error as in equation (8):

$$L(\theta) = \frac{1}{RN} \sum_{r=0}^{R-1} \sum_{i=0}^{N-1} (\hat{y}_{r,i} - y_{r,i})^2 \quad (8)$$

Where R is the batch size and N is the number of features in the last layer.

The neural network is trained to predict accurate results by iterating through the training data and minimizing the cost function. Since the given input X stays fixed, the network learns to optimize its weights $W^{(l)}$ and biases $b^{(l)}$ for every layer l .

3.1.4 Gradient Descent

The gradient $\nabla L(\theta)$ is a vector of partial derivatives with respect to the parameters θ of the function L defined by equation (11) and (10): [7] [4]

$$\nabla L(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (9)$$

$$\nabla L(\theta) = \left(\frac{\partial L(\theta)}{\partial \theta_0}, \frac{\partial L(\theta)}{\partial \theta_1}, \dots, \frac{\partial L(\theta)}{\partial \theta_{n-1}} \right) \quad (10)$$

The gradient $\nabla L(\theta)$ shows the direction of steepest ascent in the point $(\theta_0, \theta_0, \dots, \theta_0)$ in the n -dimensional vector space \mathbb{R}^n . For a function $f(x)$ of a single variable x , the gradient is simply the derivative of the function with respect to x and is the slope of the tangent line to f at x . For a multivariate function $f(x, y)$ of two variables x and y the gradient would be the a two-dimensional vector of the slope in the x dimension and y dimension respectively.

Gradient descent is the method of iteratively changing the values of θ proportionally to the negative gradient $-\nabla L(\theta)$ to minimize the function $L(\theta)$ (see figure 4). The most basic version of gradient descent is called Stochastic Gradient Descent (SGD) and uses the hyperparameter α , called learning rate, to control the magnitude of the gradient. Stochastic Gradient Descent is defined by equations (11) and (12).

$$\frac{\partial L(\theta)}{\partial \theta^{(l)}} = \nabla_{\theta^{(l)}} L(\theta) \quad (11)$$

$$\theta^{(l)} \rightarrow \theta^{(l)} - \alpha \frac{\partial L(\theta)}{\partial \theta^{(l)}} \quad (12)$$

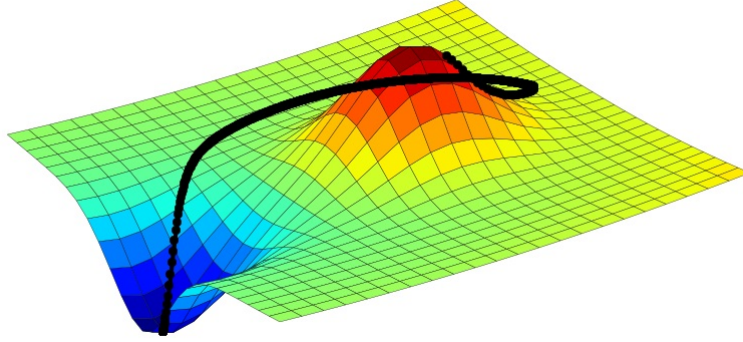


Figure 4: An illustration of Stochastic Gradient Descent on a function of two variables. Red symbolizes a high value of the function while blue symbolizes a low value. The parameters are initialized at the global maximum and their values are altered iteratively to move in the direction of the negative gradient: the direction of steepest descent. [17]

3.1.5 Backpropagation

Nätverket tränas genom att varje träningsexempel (X, y) framåtpropageras och insätts i kostnadsfunktionen L . Genom processen som kallas för bakåtpropagering beräknas alla partiella derivator med avseende på en viss vikt och används för att uppdatera vikterna tills nätverkets prognos liknar de sanna värdena. [7] [4] [2]

Backpropagation stands for "backwards propagation of errors" and is the process of calculating the partial derivatives of the loss function with respect to the model's parameters, or in other terms calculating the gradient. [2] [7]

The partial derivatives can be approximated numerically with the formal definition of a derivative defined by equation (13): [2] [7]

$$\frac{\partial L(\theta)}{\partial \theta_i} = \frac{L(\theta_0, \dots, \theta_i + h, \dots, \theta_{n-1}) - L(\theta)}{h} \quad (13)$$

This would not be a problem for the small neural network in figure 2 with a total of 24 parameters, but would be extremely inefficient for deep neural networks with millions of parameters. Instead, the chain rule is applied to calculate the precise value of the partial derivatives.

Let $\delta^{(l)}$ denote the so called delta error at layer l , defined by equation (14):

$$\frac{\partial L(\theta)}{\partial X^{(l)}} = \delta^{(l)} \quad (14)$$

The delta error is the partial derivative of the loss with respect to a specific neuron in model and is needed to efficiently calculate the gradient. The delta error can be interpreted as how much a specific neuron affects the total loss of the cost function. A deviation in value of a neuron with a big delta error results in a greater change of the total loss compared to a neuron with a small delta error. Because the activation of layer $l + 1$ is a function of the previous layer l , the delta error can be computed recursively from the output layer and propagated backwards to the first layer in the network. By applying the chain rule the delta error $\frac{\partial L(\theta)}{\partial X^{(l)}}$ at layer l can be broken up into three partial derivatives $\frac{\partial L(\theta)}{\partial X_{r,j}^{(l+1)}}$, $\frac{\partial X_{r,j}^{(l+1)}}{\partial Z_{r,j}^{(l+1)}}$ and $\frac{\partial Z_{r,j}^{(l+1)}}{\partial X_{r,i}^{(l)}}$. Since a single neuron in layer l is connected to every neuron in layer $l + 1$ you have to sum over every neuron in layer $l + 1$. The first partial derivative is the delta error of the next layer and the other two partial derivatives can be derived from equations (1) and (2) and are easily differentiable since they are linear equations:

$$\begin{aligned} \delta_{r,i}^{(l)} &= \frac{\partial L(\theta)}{\partial X_{r,i}^{(l)}} \\ &= \sum_{j=0}^{N^{(l+1)}} \frac{\partial L(\theta)}{\partial X_{r,j}^{(l+1)}} \frac{\partial X_{r,j}^{(l+1)}}{\partial Z_{r,j}^{(l+1)}} \frac{\partial Z_{r,j}^{(l+1)}}{\partial X_{r,i}^{(l)}} \\ &= \sum_{j=0}^{N^{(l+1)}} \delta_{r,j}^{(l+1)} f'(Z_{r,i}^{(l+1)}) W_{j,i}^{(l)} \end{aligned} \quad (15)$$

The delta error of the last layer $L - 1$ depends on what loss function is used. For the mean squared error the delta error of the output layer is defined by equation (16):

$$\begin{aligned} \delta^{(l_{sista})} &= \frac{\partial L(\theta)}{\partial \hat{y}} \\ &= \frac{2}{RN} (\hat{y} - y) \end{aligned} \quad (16)$$

With the delta error defined at every layer in the network the partial derivative of the loss function with respect to the networks parameters can be calculated. Just like equation (15) $\frac{\partial L(\theta)}{\partial X^{(l)}}$ is broken up into three partial

derivatives $\frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}}$, $\frac{\partial X_{r,i}^{(l+1)}}{\partial Z_{r,i}^{(l+1)}}$ and $\frac{\partial Z_{r,i}^{(l+1)}}{\partial W_{i,j}^{(l)}}$. All the examples in the mini-batch are summed over since the weights affect every single example. $\frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}}$ is the delta error and the other two partial derivatives are derived from equations (1) and (2):

$$\begin{aligned} \frac{\partial L(\theta)}{\partial W_{j,i}^{(l)}} &= \sum_{r=0}^{R-1} \frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}} \frac{\partial X_{r,i}^{(l+1)}}{\partial Z_{r,i}^{(l+1)}} \frac{\partial Z_{r,i}^{(l+1)}}{\partial W_{i,j}^{(l)}} \\ &= \sum_{r=0}^{R-1} \delta_{r,i}^{(l+1)} f'(Z_{r,i}^{(l+1)}) X_{r,j}^{(l)} \end{aligned} \quad (17)$$

The partial derivative of the loss function with respect to the biases are found in a similar way to equation (15) and (17):

$$\begin{aligned} \frac{\partial L(\theta)}{\partial b^{(l)}} &= \sum_{r=0}^{R-1} \frac{\partial L(\theta)}{\partial X_{r,i}^{(l+1)}} \frac{\partial X_{r,i}^{(l+1)}}{\partial Z_{r,i}^{(l+1)}} \frac{\partial Z_{r,i}^{(l+1)}}{\partial b_{i,j}^{(l)}} \\ &= \sum_{r=0}^{R-1} \delta_{r,i}^{(l+1)} f'(Z_{r,i}^{(l+1)}) \end{aligned} \quad (18)$$

3.1.6 Training neural networks

Bakåtpropageringen sker lager till lager med start i outputlagret. Vid varje lager måste delta-felet $\delta^{(l)}$ bakåtpropageras och derivatan av det lagrets parametrar $W^{(l)}$ och $b^{(l)}$ beräknas. [1]

Neurala nätverk tränas genom att iterativt uppdatera nätverkets vikter med hjälp av *gradient descent*. Den hela träningsdatabasen delas in i mini-hopar och väljs ut slumpvist. En iteration av SGD appliceras genom att framåtpropagera ett träningsexempel och sedan bakåtpropagera felet för att beräkna gradienten av kostnadsfunktionen med avseende på alla vikter i närverket. Denna process upprepas tills nätverkets vikter konvergerar och nätverkets prognos efterliknar verkligheten. [1]

Två implementationer av ett feed-forward neuralt nätverk kan hittas på github i python och C++: <https://github.com/nikitazozoulenko>

3.2 Konvolutionella neurala nätverk

När människor vill identifiera någonting i en bild så letar vi efter vissa karakteristiska drag objektet har. En hund består exempelvis av en kropp, ett huvud och fyra ben. Kroppsdelarna består sedan själva av grundläggande geometriska former som i sig självt är kombinationer av kanter och linjer. Dessutom har hundar en viss textur, det som vi kännetecknar som något pälsliknande. Dessa karakteristiska drag är lokala inom bilden och kan extraheras av att endast se på en liten del av bilden i taget. Detta är principen bakom *Konvolutionella Neurala Nätverk (CNN)*: Genom så kallade *konvolutioner* kunna extrahera dessa karakteristiska drag. Nätverket lär sig ett antal små filter som den applicerar på en delmängd av bilden genom att filtret sammanrullar över hela bilden (se figur 4). Värdet av filtret över en delmängd av bilden blir aktiveringen av ett neuron i nästa lager. [1]



Figure 5: Resultatet av att ett filter för vertikal respektive horisontell kantdetektering har sammanrullat över en bild av en katt.

Till skillnad från *FCC* är neuronerna i ett *CNN* bara kopplade till närliggande neuroner i det föregående lagret. På detta sätt kan nätverket lära sig fler hög-nivåspecialartiklar ju djupare i nätverket signalen går. Exempelvis kan det hända att det första lagret lär sig att identifiera kanter och linjer medan de senare lagren lär sig att känna igen olika geometriska former och till sist känna igen ansikten eller object i det sista lagret. [1]

Modellen, precis som ett *feed-forward nätverket*, består av ett flertal lager neuroner sådant att resultatet av ett lager matas in till nästkommande lager. Det sista lagret benämns med $X^{(l_{sista})}$ eller \hat{y} . Vid varje lager finns dessutom vikter $W^{(l)}$ som beror på vad för slags lager det är. $W^{(l)}$ kan vara tom med inga vikter när lager inte bidrar till någon inlärning. Ekvation (19) illustrerar

strukturen av modellen.[1] [4]

$$X^{(0)} \xrightarrow{W^{(l)}} X^{(1)} \xrightarrow{W^{(l)}} \dots \xrightarrow{W^{(l_{sista}-2)}} X^{(l_{sista}-1)} \xrightarrow{W^{(l_{sista}-1)}} X^{(l_{sista})} = \hat{y} \quad (19)$$

Samtliga olika typer av lager i en konvolutionellt neuralt nätverk har två olika faser: framåtpropagering och bakåtpropagering. Vid framåtpropagering används aktiveringen av det föregående lagret för att beräkna aktiveringen vid nästintilliggande lager. Vid bakåtpropagering måste det rekursiva delta-felet propageras baklänges i nätverket med hjälp av kedjeregeln för att sedan beräkna den partilla derivatan av lagrets vikter med avseende på kostnadsfunktionen. [1]

För ett feed-forward neuralt nätverk används matriser för att representera neuronerna. I ett *CNN* är en tensor $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ med ordning 4 aktiveringen vid lager l och indexeras med fyr-tupeln (r, c, h, w) där $0 \leq r < R$, $0 \leq c < C$, $0 \leq h < H$ och $0 \leq w < W$. Aktiveringarna vid nästintilliggande lager $l + 1$ betecknas med $X^{(l+1)} \in \mathbb{R}^{R' \times C' \times H' \times W'}$ och indexeras med fyr-tupeln (r', c', h', w') där $0 \leq r' < R'$, $0 \leq c' < C'$, $0 \leq h' < H'$ och $0 \leq w' < W'$. [1] [4]

Aktiveringen brukar illustreras som en tredimensionell volym där W , H och C är bredden, höjden respektive djupet. En $H \times W$ skiva av volymen kallas för en *feature map* eller en *kanal*. Antalet kanaler betecknas med C . R står för hopstorlek då man bearbetar R exempel i taget i en så kallad mini-hop. [1] [4]

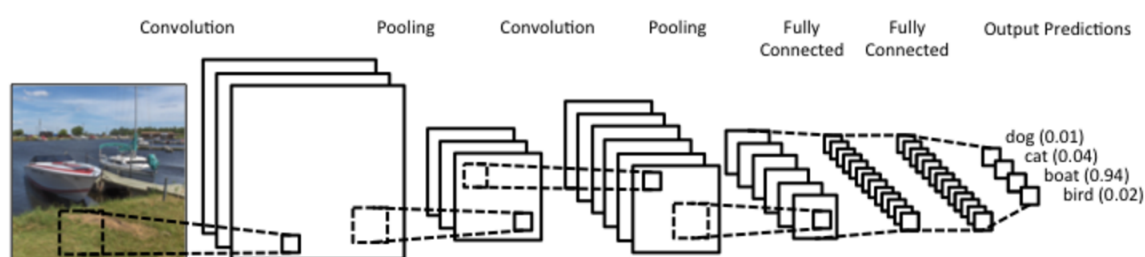


Figure 6: En illustration av ett konvolutionellt neuralt nätverk. Varje skiva är en egen *feature map*. [18]

På grund av att en konvolution är en lokal operator används CNN:s för data som innehåller lokalt sammanhängande samband, exempelvis bilder eller ljud. Om det är en bild som bearbetas har det första lagrets aktivering $C = 3$ kanaler, en för varje RGB-kanal, och en bredd och höjd lika med bildens bredd och höjd i pixlar. [1] [4]

3.2.1 Konvolutionslagret framåtpropagering

Ett konvolutionslager består av ett antal vikter kallade *kärnor* (*kernels*) eller *masker* (*masks*), representerade av en tensor med ordning fyra, $W^{(l)} \in \mathbb{R}^{C' \times C \times K_h \times K_w}$ för lager l . [1] [4]

När masken är över en godtycklig del av volymen multipliceras varje värde i delmängden av $W^{(l)}$ elementvis med respektive värde i masken vid samma position och summeras (se figur 6). Summan blir aktiveringen av ett neuron i nästa lager. Konvolutionsoperatoren betecknas med $*$. [1] [4]

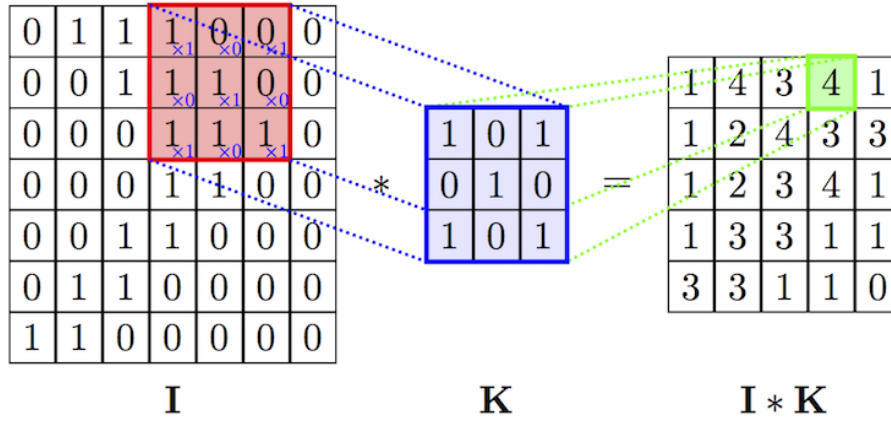


Figure 7: En kärna med storlek 3×3 sammanrullar över ett område med dimensioner 6×6 och bildar en aktivering med dimensionerna 4×4 . [19]

En feature map i lager $l + 1$ är resultatet av att en kärna med dimensioner $1 \times C \times K_h \times K_w$ har sammanrullat över aktiveringen av det föregående lagret. C' är antalet kärnor och blir dessutom antalet feature maps nästa lager har. [1] [4]

Kärnorna har två ytterligare egenskaper: ett kliv s och så kallad *zero-padding* p . s är hur stort kliv man tar efter varje gång filtret blir applicerat på tensor. Man ökar tensors höjd och bredd med $2p$ genom att fylla på med nollor vid tensors ändor (se figur 7). På grund av att aktiveringens höjd och bredd avtar ju djupare i nätverket de befinner sig på används zero-padding för att kontrollera storleken av tensor. [1] [4] [5]

Låt $W^{(l)} \in \mathbb{R}^{C' \times C \times K_h \times K_w}$, $X^{(l)} \in \mathbb{R}^{R \times C \times (H+2p) \times (W+2p)}$ och $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. Dimensionerna vid lager $l + 1$ beskrivs av: [1] [4] [5]

$$W' = \frac{W - K_w + 2p}{s} + 1 \quad (20)$$

$$H' = \frac{H - K_h + 2p}{s} + 1 \quad (21)$$

0	0	0	0	0
0	1	2	3	0
0	2	3	1	0
0	4	6	2	0
0	0	0	0	0

Figure 8: Ett område med dimensioner 3×3 zero-paddas med $p = 1$ och resulterande område får dimensioner 5×5 .

Då beskrivs en konvolution algebraiskt genom: [1] [4]

$$w = sw' \quad (22)$$

$$h = sh' \quad (23)$$

$$\begin{aligned} [X^{(l+1)}]_{r,c',h',w'} &= X_{r,c',h',w'}^{(l)} * W_{c'}^{(l)} \\ &= \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l)} \end{aligned} \quad (24)$$

Index på termen som ska sammanrullas i konvolutionen symboliserar vilka dimensioner som ska summeras. Exempelvis visar $W_{c'}^{(l)}$ att dimensionerna C , H och W (alla kanaler) ska summeras medan $W_{c',c}^{(l)}$ visar att endast H och W (en kanal) ska summeras.

I praktiken brukar konvolutioner implementeras med hjälp av funktionerna *row2im* och *im2row* vilka möjliggör att en konvolution att beräknas med en matrismultiplikation. [1] [4] [5]

3.2.2 Konvolutionslagret bakåtpropagering

Vid varje lager l bakåtpropageras det rekursiva delta-felet δ och den partiella derivatan av lagerts vikter med avseende på kostnadsfunktionen L beräknas.

Bakåtpropageringen av det rekursiva delta-felet $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ räkas ut med hjälp

av kedjeregeln. Derivatan kan inte endast delas upp i $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ och $\frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}}$, utan alla derivator måste summeras på grund av att det är mer än ett neuron som är ansvarig för framåtpropageringen likt ekvationer (17), (18) och (15).

$X_{r,c',h',w'}^{(l+1)}$ byts sedan ut mot dess definition enligt ekvation (24). [4] [10] [12] [14]

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}}
\end{aligned} \tag{25}$$

Varje produkt i den innersta summan kommer att vara lika med noll förutom om $X_{r,c,h'+j,w'+i}^{(l)} = X_{r,c,h,w}^{(l)}$. Förljaktligen insätter man $h'+j = h$ och $h'+j = h$. Summorna och derivatan förkortas: [10] [12] [14]

$$\begin{aligned}
&\sum_{c'}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} W_{c',c,j,i}^{(l+1)} \\
&= \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} W_{c',c,(h-h'),(w-w')}^{(l+1)} \delta_{r,c',h',w'}^{(l+1)}
\end{aligned} \tag{26}$$

Vilket man kan se är en summa av konvolutioner där en viss *feature map* av delta-felet sammanrullar över alla kärnor på en viss *feature map* med vikter som är roterade 180°. För att en konvolution ska kunna ske måste den roterade vikten zero-paddas på grund av att det glidande fönstret måste vara som mest lika stor som tensorn den sammanrullar över. Låt rotationen betecknas med funktionen $rot()$. [10] [12] [14]

$$\delta_{r,c,h,w}^{(l)} = \sum_{c'=0}^{C'-1} rot(W_{c',c,h,w}^{(l+1)}) * \delta_{r,c'}^{(l+1)} \tag{27}$$

En sundshetskontroll visar att detta är intuitivt då alla *feature maps* i $X^{(l)}$ används för att skapa en enstaka *feature map* i $X^{(l+1)}$. Det är därför man summerar över alla kärnor och endast konvolverar i en *feature map* i taget och summerar alltihop. [1]

Den partiella derivatan av kostandsfunktionen med avseende på vikterna hittas på ett liknande sätt. Här summeras dessutom R -dimensionen på grund av att alla exempel i mini-hopen har en påverkan på gradienten: [1] [10] [12] [14]

$$\begin{aligned}
\frac{\partial L(W)}{\partial W_{c',c,h,w}^{(l)}} &= \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial W_{r,c,h,w}^{(l)}} \\
&= \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial \sum_{c=0}^{C-1} \sum_{j=0}^{K_H-1} \sum_{i=0}^{K_W-1} X_{r,c,h'+j,w'+i}^{(l)} W_{c',c,j,i}^{(l)}}{\partial W_{c',c,h,w}^{(l)}} \\
&= \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} X_{r,c,h'+h,w'+w}^{(l)} \delta_{r,c',h',w'}^{(l+1)} \\
&= \sum_{r=0}^{R-1} \sum_{c'=0}^{C'-1} X_{r,c,h,w}^{(l)} * \delta_{r,c'}^{(l+1)}
\end{aligned} \tag{28}$$

3.2.3 Aktiveringsfunktionslager framåtpropagering

Aktiveringsfunktionen f appliceras elementvis på alla neuroner i $X^{(l)}$. Följaktligen har $X^{(l)}$ och $X^{(l+1)}$ samma dimensioner. Låt aktiveringsfunktionen betecknas med f . Nervsignalen framåtpropageras genom: [4]

$$X_{r,c,h,w}^{(l+1)} = f(X_{r,c,h,w}^{(l)}) \tag{29}$$

Aktiveringsfunktioner ökar nätverks precision och får dem att konvergera snabbare, vilket leder till att mindre datakraft krävs för att träna nätverket. [1]

3.2.4 Aktiveringsfunktionslager bakåtpropagering

Aktiveringsfunktioner har inga parametrar som ska optimeras och således är $W^{(l)}$ och $\frac{\partial L(W)}{\partial W^{(l)}}$ tomma. Bakåtpropageringen av det rekursiva delta-felet härleds med hjälp av kedjeregeln: [1] [4]

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\
&= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l+1)}} \frac{\partial X_{r,c,h,w}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \delta_{r,c,h,w}^{(l+1)} f'(X_{r,c,h,w}^{(l)})
\end{aligned} \tag{30}$$

3.2.5 Maxpoolingret framåtpropagation

Här är igen inputneuronerna representerade av $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ och skapar output $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. Lagret saknar vikter men har däremot hyperparametrarna k (kärnstorlek) och s (stride eller kliv). *Maxpooling* delar in varje *feature map* i $X^{(l)}$ i ett antal sektioner med dimensioner $k \times k$ genom att ett glidande fönster med samma dimensioner sammanrullar över alla lagrets *feature maps* (se figur 8). Aktiveringen vid ett neuron i lager $l + 1$ blir lika med det största värdet i korresponderande $k \times k$ sektion. [1] [4] [5]

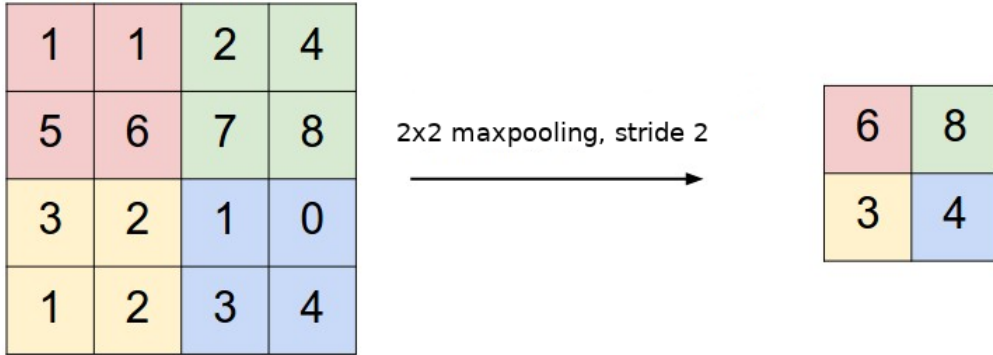


Figure 9: Maxpooling med $k = 2$ och $s = 2$ av ett område med dimensioner 4×4 där resultatet bildar ett område med dimensionerna 2×2 .

Liknande konvolutionslagret utan zero-padding blir det nästintillkommande lagrets dimensioner: [1] [4] [5]

$$W' = \frac{W - k}{s} + 1 \tag{31}$$

$$H' = \frac{H - k}{s} + 1 \tag{32}$$

$$C' = C \tag{33}$$

Antal kanaler förblir konstant. [1] [4]

Matematiskt beskrivs maxpoolagret genom: [1] [4]

$$X_{r,c',h',w'}^{(l+1)} = \max_{0 \leq j < k, 0 \leq i < k} X_{r,c',(h's+j),(w's+i)}^{(l)} \quad (34)$$

3.2.6 Maxpoolagret bakåtpropagering

Maxpooling saknar vikter och därmed är $\frac{\partial L(W)}{\partial W^{(l)}}$ tom. Det som återstår är bakåtpropageringen av delta-felet. Med hjälp av kedjeregeln kan man dela upp derivatan i två bråk, $\frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ och $\frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \cdot \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}}$ är den rekursiva delta-delet. $X_{r,c',h',w'}^{(l+1)}$ byts sedan ut mot dess definition enligt ekvation (34): [1] [4] [14]

$$\begin{aligned} \delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\ &= \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial X_{r,c,h,w}^{(l)}} \\ &= \delta_{r,c',h',w'} \frac{\partial \max_{0 \leq j < k, 0 \leq i < k} X_{r,c',(h's+j),(w's+i)}^{(l)}}{\partial X_{r,c,h,w}^{(l)}} \end{aligned} \quad (35)$$

Den partiella derivatan i den sista ekvationen kommer vara lika med 1 om $X_{r,c',(h's+j),(w's+i)}^{(l)} = X_{r,c,h,w}^{(l)}$. I annat fall kommer $X_{r,c,h,w}^{(l)}$ inte ha någon påverkan på neuron index (r, c, h, w) i lager $l + 1$ och den partiella derivatan blir lika med 0: [1] [4] [14]

$$\delta_{r,c,h,w}^{(l)} = \begin{cases} \delta_{r,c',h',w'} & \text{om } \begin{matrix} h = h's + j, \\ w = w's + i \end{matrix} \\ 0 & \text{i annat fall} \end{cases} \quad (36)$$

Alltså omdiregeras delta-felet till det ansvariga neuronet vars index kommer att behöva hållas i minnet. Om det finns två eller fler sektioner med samma neuron som är ansvarig för framåtpropageringen så kommer delta-felen summeras från samtliga korresponderande sektioners delta-fel. [1] [4] [14]

3.2.7 Batch Normalization framåtpropagering

Utan Batch Normalization (BN) är det svårt att få djupa nätverk att konvergera. Detta är till följd av att en liten ändring till det första lagret kan leda till en kaskad av förändringar i de senare lagren. I litteraturen kallas detta för *internal covariate shift*. BN försöker att minimera denna *internal covariate shift* genom att med avseende på alla exempel i mini-hopen normalisera varje *feature map* till varje lager. Resultatet är snabbare konvergens och att det tillåter större träningshastigheter. Följaktligen har bearbetningen av flera exempel i taget i en mini-hop en annan praktiska tillämpning än att enbart beräkna konvolutionerna mer tidseffektivt. [1] [9]

Igen är aktiveringen vid lager l och $l + 1$ $X^{(l)} \in \mathbb{R}^{R \times C \times H \times W}$ respektive $X^{(l+1)} \in \mathbb{R}^{R \times C' \times H' \times W'}$. BN har ingen påverkan på dimensionerna av aktiveringen. [1] [9]

Först beräknas medelvärdena μ_c och varianserna σ_c^2 till varje *feature map* c : [1] [9]

$$\mu_c = \frac{1}{RHW} \sum_{r=0}^{R-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} X_{r,c,h,w}^{(l)} \quad (37)$$

$$\sigma_c^2 = \frac{1}{RHW} \sum_{r=0}^{R-1} \sum_{h=0}^{H-1} \sum_{w=0}^{W-1} (X_{r,c,h,w}^{(l)} - \mu_c)^2 \quad (38)$$

Sedan beräknas den normaliserade aktiveringen \hat{X} . Epsilon används för numerisk stabilitet. [1] [9]

$$\hat{X}_{r,c,h,w} = (X_{r,c,h,w}^{(l)} - \mu_c)(\sigma_c^2)^{-\frac{1}{2}} \quad (39)$$

Sist introduceras 2 vikter, $\gamma_c^{(l)}$ och $\beta_c^{(l)}$, vilka tillåter nätverket att upphäva normaliseringen om nätverket dömmar det att vara användbart. [1] [9]

$$X_{r,c,h,w}^{(l+1)} = \gamma_c^{(l)} \hat{X}_{r,c,h,w} + \beta_c^{(l)} \quad (40)$$

När nätverket ska köras utanför träning, också kallat *runtime* är det dock inte alltid möjligt att beräkna medelvärdet och variansen av mini-hopen på grund av att man oftast enbart vill testa ett exempel i taget. Medelvärdet och variansen för hela populationen måste då räknas ut och användas i stället för de beräknade värdena. Detta kan göras för små databaser, men om man arbetar med data som innehåller miljontals exempel är det enklare att uppskatta populationens statistik med hjälp av att updatera ett exponensiellt

glidande medelvärde (EWMA) vid varje framåtpropagering: [1] [9]

$$\mu_{EWMA_c} \rightarrow \lambda\mu_c + (1 - \lambda)\mu_{EWMA_c} \quad (41)$$

$$\sigma_{EWMA_c}^2 \rightarrow \lambda\sigma_c^2 + (1 - \lambda)\sigma_{EWMA_c}^2 \quad (42)$$

Där μ_{EWMA_c} och $\sigma_{EWMA_c}^2$ betecknar de exponensiella glidande medelvärdena och λ betecknar dämpfaktorn.

3.2.8 Batch Normalization bakåtpropagering

För BN behöver det rekursiva delta-felet $\delta^{(l)}$, derivatan av kostandsfunktionen med avseende på $\gamma_{c'}^{(l)}$ och derivatan av kostandsfunktionen med avseende på $\beta_{c'}^{(l)}$ beräknas. För att beräkna detta krävs något som heter kronecker-deltat, oftast betecknat med $\delta_{i,j}$ men kommer vara betecknat med $I_{i,j}$ i denna rapport på grund av δ används för en annan term. Kronecker-deltat har följande egenskaper: [11] [13]

$$I_{i,j} = \begin{cases} 1 & \text{om } i = j \\ 0 & \text{om } i \neq j \end{cases} \quad (43)$$

$$\frac{\partial a_j}{\partial a_i} = I_{i,j} \quad (44)$$

$$\sum_j a_i I_{i,j} = a_j \quad (45)$$

Först bryts $\frac{\partial L(W)}{\partial X^{(l)}}$ upp i tre bråk och sedan summeras alla partiella derivator likt ekvation (25). [11] [13] Här summeras dessutom R-dimensionen på grund av att aktiveringar från hela mini-hopen har en påverkan på $\delta^{(l)}$.

$$\begin{aligned} \delta_{r,c,h,w}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l)}} \\ &= \sum_{r'=0}^{R'-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r',c',h',w'}^{(l+1)}} \frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} \frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} \end{aligned} \quad (46)$$

$\frac{\partial L(W)}{\partial X_{r,c,h,w}^{(l+1)}}$ är det föregående rekursiva delta-felet. $\frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}}$ hittas enkelt på grund av att den är en linjär funktion. [11] [13]

$$\begin{aligned}\frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} &= \frac{\partial(\gamma_{c'}^{(l)} \hat{X}_{r',c',h',w'} + \beta_{c'}^{(l)})}{\partial \hat{X}_{r',c',h',w'}} \\ &= \gamma_{c'}^{(l)}\end{aligned}\quad (47)$$

För derivatan av den centrerade aktiveringen med avseende på den originella aktiveringen tillämpas produktregeln: [11] [13]

$$\begin{aligned}\frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial(X_{r',c',h',w'}^{(l)} - \mu_{c'}) (\sigma_{c'}^2)^{-\frac{1}{2}}}{\partial X_{r,c,h,w}^{(l)}} \\ &= (\sigma_{c'}^2)^{-\frac{1}{2}} \frac{\partial(X_{r',c',h',w'}^{(l)} - \mu_{c'})}{\partial X_{r,c,h,w}^{(l)}} - \frac{1}{2} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) (\sigma_{c'}^2)^{-\frac{3}{2}} \frac{\partial \sigma_{c'}^2}{\partial X_{r,c,h,w}^{(l)}}\end{aligned}\quad (48)$$

Derivatan av den första faktorn med avseende på aktiveringen beräknas med hjälp av ekvationer (43), (44) och (45). [11] [13]

$$\begin{aligned}\frac{\partial(X_{r',c',h',w'}^{(l)} - \mu_{c'})}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial(X_{r',c',h',w'}^{(l)} - \frac{1}{RHW} \sum_{r''=0}^{R-1} \sum_{h''=0}^{H-1} \sum_{w''=0}^{W-1} X_{r'',c',h'',w''}^{(l)})}{\partial X_{r,c,h,w}^{(l)}} \\ &= I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}\end{aligned}\quad (49)$$

Derivatan av den andra faktorn med avseende på aktiveringen beräknas på ett liknande sätt med hjälp av kedjeregeln och ekvationer (43), (44) och (45). [11] [13]

$$\begin{aligned}\frac{\partial \sigma_{c'}^2}{\partial X_{r,c,h,w}^{(l)}} &= \frac{\partial \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X_{r',c',h',w'}^{(l)} - \mu_{c'})^2}{\partial X_{r,c,h,w}^{(l)}} \\ &= \frac{1}{RHW} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} 2(X_{r',c',h',w'}^{(l)} - \mu_{c'}) (I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}) \\ &= \frac{2}{RHW} (X_{r,c',h,w}^{(l)} - \mu_{c'}) I_{c',c} - \frac{2}{(RHW)^2} \sum_{r'=0}^{R-1} \sum_{h'=0}^{H-1} \sum_{w'=0}^{W-1} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) \\ &= \frac{2}{RHW} (X_{r,c',h,w}^{(l)} - \mu_{c'}) I_{c',c}\end{aligned}\quad (50)$$

Den sista summan blir lika med noll på grund av att termerna summeras ihop till medelvärde minus medelvärde.

När alla komponenter till bakåtpropageringen av delta-felet är beräknade är insättning av ekvation (48), (49) och (50) i ekvation (46) det enda som kvarstår:

$$\begin{aligned}
\delta_{r,c,h,w}^{(l)} &= \sum_{r'=0}^{R-1} \sum_{c'=0}^{C'-1} \sum_{h'=0}^{H'-1} \sum_{w'=0}^{W'-1} \frac{\partial L(W)}{\partial X_{r',c',h',w'}^{(l+1)}} \frac{\partial X_{r',c',h',w'}^{(l+1)}}{\partial \hat{X}_{r',c',h',w'}} \frac{\partial \hat{X}_{r',c',h',w'}}{\partial X_{r,c,h,w}^{(l)}} \\
&= \sum_{r',c',h',w'} \delta_{r',c',h',w'}^{(l+1)} \gamma_{c'}^{(l)} (\sigma_{c'}^2)^{-\frac{1}{2}} (I_{r',r} I_{c',c} I_{h',h} I_{w',w} - \frac{1}{RHW} I_{c',c}) \\
&\quad - \sum_{r',c',h',w'} \delta_{r',c',h',w'}^{(l+1)} \gamma_{c'}^{(l)} \frac{1}{RHW} (X_{r',c',h',w'}^{(l)} - \mu_{c'}) (X_{r,c,h,w}^{(l)} - \mu_{c'}) (\sigma_{c'}^2)^{-\frac{3}{2}} I_{c',c} \\
&= \delta_{r,c,h,w}^{(l+1)} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} - \frac{1}{RHW} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} \\
&\quad - \frac{1}{RHW} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \gamma_c^{(l)} (X_{r',c,h',w'}^{(l)} - \mu_{c'}) (X_{r,c,h,w}^{(l)} - \mu_c) (\sigma_c^2)^{-\frac{3}{2}} \\
&= \frac{1}{RHW} \gamma_c^{(l)} (\sigma_c^2)^{-\frac{1}{2}} \left(RHW \delta_{r,c,h,w}^{(l+1)} - \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} \right. \\
&\quad \left. - (X_{r,c,h,w}^{(l)} - \mu_c) (\sigma_c^2)^{-\frac{3}{2}} \sum_{r',h',w'} \delta_{r',c,h',w'}^{(l+1)} (X_{r',c,h',w'}^{(l)} - \mu_{c'}) \right)
\end{aligned} \tag{51}$$

Derivatn av vikterna hittas på ett liknande sätt som ekvation (46) till (51).
[11] [13]

$$\begin{aligned}
\frac{\partial L(W)}{\partial \gamma_c^{(l)}} &= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial \gamma_c^{(l)}} \\
&= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial (\gamma_{c'}^{(l)} \hat{X}_{r,c',h',w'} + \beta_{c'}^{(l)})}{\partial \gamma_c^{(l)}} \\
&= \sum_r \sum_{c'} \sum_{h'} \sum_{w'} \delta_{r,c',h',w'}^{(l+1)} \hat{X}_{r,c,h',w'} I_{c',c} \\
&= \sum_r \sum_{h'} \sum_{w'} \delta_{r,c,h',w'}^{(l+1)} \hat{X}_{r,c,h',w'}
\end{aligned} \tag{52}$$

$$\begin{aligned}
\frac{\partial L(W)}{\partial \beta_c^{(l)}} &= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \frac{\partial L(W)}{\partial X_{r,c',h',w'}^{(l+1)}} \frac{\partial X_{r,c',h',w'}^{(l+1)}}{\partial \beta_c^{(l)}} \\
&= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c',h',w'}^{(l+1)} \frac{\partial(\gamma_{c'}^{(l)} \hat{X}_{r,c',h',w'} + \beta_{c'}^{(l)})}{\partial \beta_c^{(l)}} \\
&= \sum_r^{R-1} \sum_{c'}^{C'-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c,h',w'}^{(l+1)} I_{c',c} \\
&= \sum_r^{R-1} \sum_{h'}^{H'-1} \sum_{w'}^{W'-1} \delta_{r,c,h',w'}^{(l+1)}
\end{aligned} \tag{53}$$

3.2.9 Softmax framåtpropagering

Funktionen softmax används i det sista lagret för att gränsa aktiveringen till värden i intervallet $[0, 1]$ och har egenskapen att alla prognostiserade värdena i ett exempel från mini-hopen summeras till 1. Om modellen ska klassificera ett objekt som kan vara av C olika klasser kan \hat{y} tolkas som sannolikheten att objektet är av varje klass c ; $0 \leq c < C$. [1]

Inmatningsvärde för softmaxlagret är en matris $X^{(l)} \in \mathbb{R}^{R \times C}$ och producerar en matris $X^{(l+1)} \in \mathbb{R}^{R \times C}$ med samma dimensioner. Softmaxlagret definieras enligt:

$$X_{r,c}^{(l+1)} = \frac{e^{X_{r,c}^{(l)}}}{\sum_{c'=0}^{C-1} e^{X_{r,c'}^{(l)}}} \tag{54}$$

3.2.10 Softmax bakåtpropagering

Softmaxlagret saknar vikter och endast det rekursiva delta-felet beräknas vid bakåtpropagering. Kvotregeln, kedjeregeln och kronecker-deltat I används för att härleda den partiella derivatan. Likt de andra lagren summeras de partiella derivatorna på grund av att aktiveringen för ett neuron i lager $l+1$

är en funktion av alla neuron i lager l : [1] [3] [15]

$$\begin{aligned}
\delta_{r,c}^{(l)} &= \frac{\partial L(W)}{\partial X_{r,c}^{(l)}} \\
&= \sum_{c'=0}^{C-1} \frac{\partial L(W)}{\partial X_{r,c'}^{(l+1)}} \frac{\partial X_{r,c'}^{(l+1)}}{\partial X_{r,c}^{(l)}} \\
&= \sum_{c'=0}^{C-1} \delta_{r,c'}^{(l+1)} \left(\frac{(e^{X_{r,c'}^{(l+1)}}) I_{c',c}}{\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}}} - \frac{(e^{X_{r,c'}^{(l+1)}})(e^{X_{r,c}^{(l+1)}})}{(\sum_{c''=0}^{C-1} e^{X_{r,c''}^{(l+1)}})^2} \right) \\
&= \sum_{c'=0}^{C-1} \delta_{r,c'}^{(l+1)} X_{r,c'}^{(l+1)} (I_{c',c} - X_{r,c}^{(l+1)}) \\
&= \delta_{r,c}^{(l+1)} X_{r,c}^{(l+1)} \left(1 - \sum_{c'=0}^{C-1} X_{r,c'}^{(l+1)} \right)
\end{aligned} \tag{55}$$

3.3 Problem Cases

The code for All kod till de exempel på praktiska tillämpningar kan hittas på github: <https://github.com/nikitazozoulenko>

3.3.1 Classification of handwritten digits

En enkel CNN-modell kan användas för att klassificera handskrivna siffror. För att uppnå detta har MNISTdatabasen för handskrivna siffror används. Den består av 60 000 unika exempel av handskrivna siffror. [16]

Låt modellens prognos betecknas med \hat{y} . Aktiveringsfunktionen softmax används i det sista lagret för att gränsa värdena till intervallet $[0, 1]$ och har egenskapen att alla prognostiserade värdena i ett exempel summeras till 1. Följaktligen kan varje värde i \hat{y} tolkas som sannolikheten att bilden är av varje klass. [1]

Input för modellen är en $R \times 1 \times 28 \times 28$ tensor där R står för minihopstorleken. Modellen prognostiserar tio värden per bild i form av en tensor $\hat{y} \in \mathbb{R}^{R \times C}$, ett värde för varje klass $C = 10$ av siffror. Konstansdfunktionen som har minimerats under tränings tiden är funktionen *cross-entropy* betecknat med L . Den verkar på två sannolikhetsfördelningar: De verkliga



Figure 10: Tio bilder av handskrivna siffror från MNISTdatabasen. [16]

sannolikheterna y och de prognostiserade sannolikheterna \hat{y} : [1] [3]

$$L(W) = - \sum_{r=0}^{R-1} \sum_{c=0}^{C-1} y_{r,c} \log \hat{y}_{r,c} \quad (56)$$

$$\frac{\partial L(W)}{\partial \hat{y}_{r',c'}} = - \frac{y_{r',c'}}{\hat{y}_{r',c'}} \quad (57)$$

Endast en modell tränades på grund av begränsningar i datakraft. Den bestod av tre stycken konvolutionsblock, följt med ett softmaxlager (se figur 10). En mini-hopstorlek av 50 användes och 5000 iterationer av framåt- och bakåtpropagering kördes på min egenimplementerade modell skriven i python. Totalt tog det 4 timmar att träna nätverket på min CPU.

Den slutgiltiga precisionen blev 99.2% på 10 000 nya handskrivna siffror modellen aldrig hade sett tidigare. Av 10 000 handskrivna siffror lyckades modellen klassificera 9 919 siffror rätt.

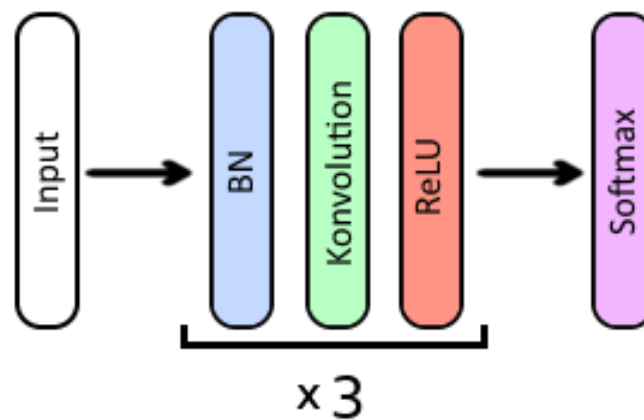


Figure 11: Modellen som användes för att läsa av handskrivna siffror. Batch Normalization benämns med BN, konvolution menas en konvolution med kärnstorlek $k = 5$ och stride $s = 2$. ReLU är aktiveringsfunktionen Rectified Linear Units och softmax är softmaxlagret.

3.3.2 Dense Face Detection and Localization

Ett konvolutionellt neuralt nätverk är anpassat för att detektera upp till flera tusen ansikten åt gången för realtidsvideo. Modellen producerar ett bestämt antal *boudning boxes*: koordinater som ska detektera alla olika objekt i bilden. Jag utgick från arkitekturen från RetinaNet från *Focal Loss for Dense Object Detection* (Lin et al.) och anpassade den till $K = 1$ klasser. I varje position i outputlagret föreslår modellen en *boudning box* med fyra koordinater: två punkter för det övre vänstra hörnet respektive nedre hörna hörnet av den positionens *bounding box*. Utöver det förutsägs det $K + 1$ sannolikheter att *bounding boxen* innehåller ett objekt av alla K förgrundklasser och 1 bakgrundklass (lådan innehåller inga objekt).[1][23]

Modellen för objekt-detektering går ut på att man utgår från ett antal så kallade anchor boxes vid varje rumslig position i det sista konvolutionella lagret. Om lagret har bredden och höjden W respektive H och A olika storlekar på anchor boxes har lagret totalt WHA olika anchor boxes. Vid träning tilldelas en anchor box ett objekt om dens *intersection over union* (IoU) med den verkliga bounding boxen är större än 0.5 (se figur 11 IoU används för att beräkna hur lika två olika mängder är. I detta fall är mängderna areorna av en anchor box och ett objekts verkliga bounding box. IoU definieras som storleken av snittet av två mängder A och B dividerat med storleken av unionen av A och B : [1] [22]

$$IoU(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (58)$$

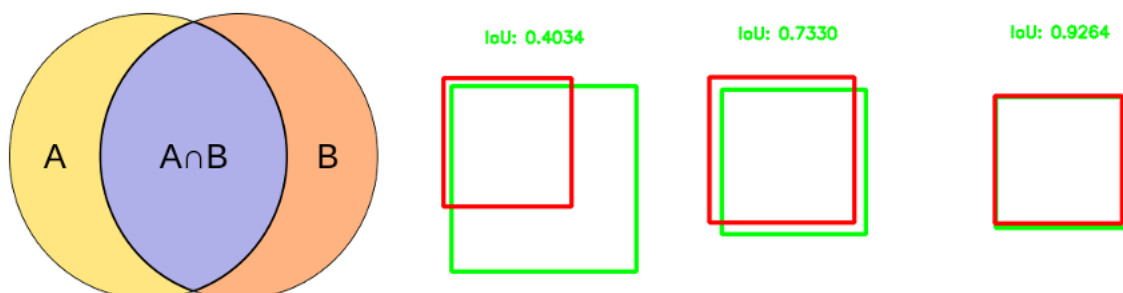


Figure 12: IoU definieras som storleken av snittet av två mängder A och B dividerat med storleken av unionen av A och B . En större IoU medför att prognosen är närmare sanningen. [22]

RetinaNet använder sig av en *feature pyramid*-arkitektur beskriven i "*Feature Pyramid Networks for Object Detection*" (Lin et al.) och en ResNet101 (He et al.) som *backbone*. Featurepyramiden används för att få ut *feature maps* från olika delar av det konvolutionella neurala nätverket för att kunna

detektera objekt av olika storlekar. Varje pyramidnivå matas in i ett klassifikationshuvud respektive regressionshuvud (se figur 12) för att räkna ut sannolikheterna att varje anchor box innehåller de $K + 1$ olika klasserna, samt att förfina varje anchor box koordinater genom att prognostisera 4 olika offsets för varje sida av anchor boxens bounding box. [21] [23] [24]

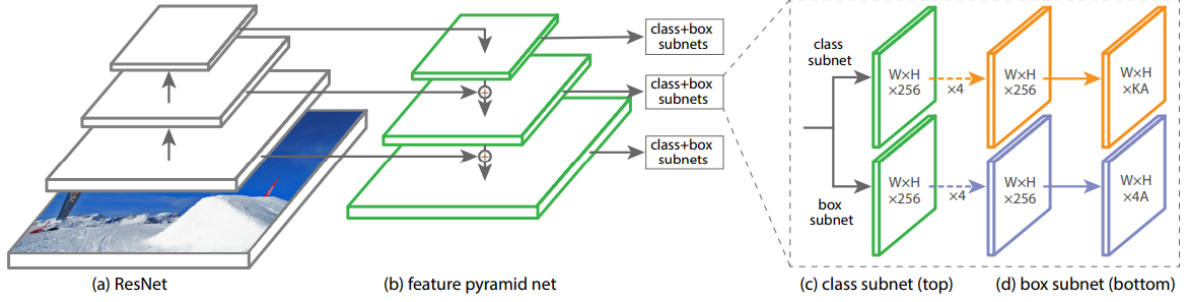


Figure 13: RetinaNets arkitektur från *Focal Loss for Dense Object Detection* (Lin et al.). Varje pyramidnivå matas in i ett klassifikationshuvud och ett regressionshuvud. [23]

Till skillnad från RetinaNet har min modell, FaceNet, ett mindre antal pyramidnivåer för att lätta på beräkningskraften och för att detektera mindre objekt. RetinaNets anchors har areor från 32^2 till 512^2 pixlar spridda över pyramidnivåer P3 till P7. FaceNet använder sig av areor 14^2 till 220^2 pixlar spridda över pyramidnivåer P3 till P6. Regressionshuvudet och klassifikationshuvudet i RetinaNet består av 5 konvolutionslager med kärnstorlek 3. FaceNet använder istället 2 och 3 konvolutionslager för regressionshuvudet respektive klassifikationshuvudet. I det originala Feature Pyramidnätverket användes linjär interpolation för att göra storleken av högre feature maps större. I FaceNet används k-nearest neighbors (k-NN) istället. ResNet50 användes istället för ett ResNet101 som backbone för att lätta på beräkningskostnaderna. [21] [23] [24]

Kostnadsfunktionen som användes var en kombination av Focal Loss (Lin et al.) och Smooth L1 Loss. FaceNet använder samma hyperparametrar som uppnådde bästa resultat för RetinaNet: $\alpha = 3$ för förgrundklassen och $\gamma = 2$. [23]

$$L_r(x) = \begin{cases} 0.5x^2 & \text{om } |x| < 1 \\ |x| - 0.5 & \text{i annat fall} \end{cases} \quad (59)$$

$$L_c(p, \hat{p}) = -\alpha(1 - p)^\gamma p \log \hat{p} \quad (60)$$

$$\begin{aligned}
L(W) = & \sum_{k \in \text{pyramid}} \frac{1}{N_c^k} \sum_{a \in \text{anchors}} L_r(r_a - \hat{r}_a) \\
& + \sum_{k \in \text{pyramid}} \frac{1}{N_c^r} \sum_{a \in \text{anchors}} L_c(p_a, \hat{p}_a)
\end{aligned} \tag{61}$$

N_c^k och N_r^k betecknar alla anchors som deltar i klassifikationskostnaden respektive regressionskostnaden i pyramidnivå k . För klassifikationskostnaden är \hat{p} 1 för alla anchors som har blivit tillgivna ett objekt och 0 för alla som bara innehåller bakgrundsklassen. Endast anchors som innehåller ett objekt deltar i regressionskostnaden. r_a innehåller de 4 koordinaterna den slutgiltiga boudning boxen har.

Modellen tränades i 350 000 iterationer med en mini-hopsstorlek av 3. Träningshastigheten började på 0.005 och dividerades med 10 vid 200 000 iterationer. Varje träningssexempels storlek gjordes slumpmässigt om till 512^2 , 576^2 eller 640^2 pixlar och speglades horisontellt med sannolikhet 0,5 för att artificiellt utöka mängden träningsdata. Den totala tränings tiden var 18 timmar på en NVIDIA GTX 1080ti. Figur 13 visar kvantitativa resultat från WIDERFace valideringsdata.

En iteration av framåtpropageringen tar 20 ms vilket möjliggör att modellen kan köras i realtid, 50 gånger i sekunden.

4 Diskussion

Vad är ett konvolutionellt neuralt nätverk? Hur härleds framåt- och bakåtpropagering i konvolutionella neurala nätverk? Hur kan modellen tillämpas för att implementera sifferavläsning, ansiktsgenkänning och objekt-detektering i bilder?

Den matematiska modellen av artificiella neurala nätverk och konvolutionella neurala nätverk, samt bakåt- och framåtpropagering förklaras och härleds i sektion 3.2 och 3.3. Ett exempel på hur modellen kan tillämpas är att klassificera handskrivna siffror, som i sektion 3.4.1. En annan möjlighet är att anpassa ett konvolutionellt neuralt nätverk för att detektera alla ansikten i en digital bild. På grund av att en iteration av algoritmen tar 20 ms är det möjligt att köra modellen i realtid. Det kan exempelvis användas i övervakningskameror eller inom robotik för att känna igen människoansikten i realtid.



Figure 14: Kvantitativa resultat från valideringsdaten från WIDERFace: Bilder som modellen aldrig har sett tidigare.

Algoritmen är dock inte perfekt och har flera brister. Modellen prognostiserar att det finns ansikten på flera ställen där det inte finns några i verkligheten. För att undankomma detta låter jag endast modellen visa upp bounding boxes med sannolikhet större än 0.5 att det finns ett ansikte där. Resultatet blir att modellen inte hittar 100% av all ansikten i bilden (se figur 13). Ytterligare har modellen problem med små ansikten. Detta är dels på grund av att modellen inte kan urskilja ett ansikte med så lite information (ca minimum 10^2 pixlar per ansikte), och dels för att FaceNet prognostiserar ca 30 000 olika anchors för de minsta storlekarna. Detta leder till att klassifikationskostnaden består av till mestadels negativa exempel. Delta-felet av bakgrundsklasserna dominerar och gör att delta-felet av förgrundsklassen (ansikten) inte har en stor påverkan på den slutgiltiga gradienten.

References

- [1] *CS231n: Convolutional Neural Networks for Visual Recognition*. F. Li, A. Karpathy och J. Johnson. Stanford University, föreläsning, vinter 2016.
- [2] *Unsupervised Feature Learning and Deep Learning*. Stanford University, Department of Computer Science. URL <http://ufdl.stanford.edu/wiki/>. Senast uppdaterad 31 mars 2013.
- [3] *Notes on Backpropagation*. P. Sadowski. University of California Irvine Department of Computer Science.
- [4] *Introduction to Convolutional Neural Networks*. J. Wu. National Key Lab for Novel Software Technology, Nanjing University, Kina. 1 maj, 2017.
- [5] *A guide to convolution arithmetic for deep learning*. V. Dumoulin och F. Visin. FMILA, Université de Montréal. AIRLab, Politecnico di Milano. 24 mars, 2016.
- [6] *High Performance Convolutional Neural Networks for Document Processing*. K. Chellapilla, S. Puri, P. Simard. Tenth International Workshop on Frontiers in Handwriting Recognition. La Baule, Frankrike, Suvisoft. Oktober 2006.
- [7] *Scientific Computing 2013, Worksheet 6: Optimization: Gradient and steepest descent*. University of Tartu, Estland. 2013.
- [8] *You only look once: Unified, real-time object detection*. J. Redmon, S. Divvala, R. Girshick, och A. Farhadi. arXiv preprint arXiv:1506.02640, 2015.
- [9] *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. S. Ioffe och C. Szegedy. arXiv preprint arXiv:1502.03167, 2015.
- [10] *Backpropagation In Convolutional Neural Networks*. J. Kafunah. DeepGrid, Organic Deep Learning. URL <http://www.jefkine.com/>. 5 september 2016.
- [11] *What does the gradient flowing through batch normalization looks like?* C. Thorey. Machine Learning Blog. URL <http://cthorey.github.io/>. 28 januari 2016.

- [12] *Note on the implementation of a convolutional neural networks.* C. Thorey. Machine Learning Blog. URL <http://cthorey.github.io/>. 2 februari 2016.
- [13] *Understanding the backward pass through Batch Normalization Layer.* Flaire of Machine Learning URL <https://kratzert.github.io>. 5 september 2016.
- [14] *Convolutional Neural Networks.* A. Gibiansky. URL <http://andrew.gibiansky.com>. 24 februari 2014.
- [15] *Classification and Loss Evaluation - Softmax and Cross Entropy Loss.* P. Dahal. DeepNotes. URL <https://deepnotes.io/softmax-crossentropy>. 24 februari 2014.
- [16] *The MNIST database of handwritten digits* Y. LeCun, C. Cortes och C. Burges. Courant Institute, NYU. Google Labs, New York. Microsoft Research, Redmond. URL <http://yann.lecun.com/exdb/mnist/>. Hämtad 3 november 2017.
- [17] *Pygradsc.* J. Komoroske. URL <https://github.com/joshdk/pygradesc>. 12 oktober 2012.
- [18] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.
- [19] *Understanding Convolutional Neural Networks for NLP.* D. Britz. WildML, Artificial Intelligence, Deep Learning, and NLP. 7 november 2015.
- [20] *Deep learning for complete beginners: convolutional neural networks with keras.* P. Veličković. Camebridge Spark. URL <https://cambridgespark.com/content>. Senast uppdaterad 20 mars 2017.
- [21] *Deep residual learning for image recognition.* K. He, X. Zhang, S. Ren, och J. Sun. arXiv preprint arXiv:1512.03385, 2015.
- [22] Jaccard Index. Wikipedia. URL https://en.wikipedia.org/wiki/Jaccard_index. Hämtad 20 januari 2018
- [23] *Focal Loss for Dense Object Detection.* Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He and Piotr Dollár. arXiv preprint arXiv:1708.02002, 2017

- [24] *Feature Pyramid Networks for Object Detection*. Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan and Serge J. Belongie. arXiv preprint arXiv:1612.03144, 2016
- [25] *WIDER FACE: A Face Detection Benchmark*. Yang, Shuo and Luo, Ping and Loy, Chen Change and Tang, Xiaoou IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016