Chapter 5

Experimental Results

This chapter describes the results gained during tests described in the previous chapter.

5.1 Which Criteria for Evaluation of the Approach Exists?

Machine learning algorithms have to be measured primarily according to their capability to generalize their training data. But beside the score they achieve on different competition datasets, it is important to evaluate the circumstances they were operated on. This includes secondary criteria such as:

- The variety and kind of information that were available for learning and how abstract is the accomplished knowledge retrieved by the algorithm?
- How much computational resources and computational time were necessary for the algorithm to achieve its score on a given dataset?
- Does the algorithm converge safe and sound, or is there a risk for the algorithm of getting stuck during the learning process?

In case of Neural Networks the secondary criteria for evaluation should not be undervalued, because on the one hand Neural Networks are extensive to train and every improvement that helps to train them less extensively improves the secondary criteria for them. On the other hand, Neural Networks have been shown to be the only method capable of performing fundamental tasks in pattern recognition to todays state of knowledge. Due to their ability to process given data on a higher degree of abstraction compared to other Machine learning methods, which I pointed out in the introduction. The described approach focuses more on the secondary criteria for evaluation, because my intention was to work around a new way to use the benefits of a ResNet for a deconvolution. This benefits are primary saving computational time and resources. For this matter the required computational times and resources for Residual and non-Residual Neural Networks are compared. For a real world application the primary criteria for evaluation of the approach used in this paper has to be improved. This step requires a lot of smaller optimizations and testing as explained in chapter [6.1].

5.2 Which Experiments were carried out?

The following 4 experiments were carried out:

1. A classical CNN approach with an alternative version of the original ResNet to classify objects in the CIFAR-10 dataset.

2. After the initial ResNet converged, it was linearized for the third experiment and was further trained on the CIFAR-10 dataset. At the beginning of the training process it received the pretrained weight parameters from the initial ResNet.

3. After the training of the linearized ResNet was finished the network was reversed to perform a deconvolution on the previous output of the convolution process. The deconvolutional version of the linearized ResNet was used to locate the recognized objects on the images.

4. For the fourth and last experiment the CIFAR-10 dataset was relabeled to a 2 class dataset with "cat" and "no-cat" images. After training the convolutional part of the network with the newly labeled "CIFAR-2" dataset, the network had to perform a deconvolution operation on 720p images, which were disassembled into 32×32 fragments to locate a cat.

On item 1.:

layer name	**output size**	**10-layer**	**18-layer**	**34-layer**	**58-layer**
conv1	32×32	filter 3×3×depth 3→(192,128,64), stride (s) 1, pad 1			
conv2_x	32×32	2×2 max pool, s 1 (s2 on 64×64 datasets)			
		3×3×192 3×3×192	3×3×192 3×3×192	3×3×128 3×3×128	3×3×64 3×3×64
conv2	16×16	3×3×192→384,s2 3×3×384	3×3×192→384,s2 3×3×384	3×3×128→256,s2 3×3×256	3×3×64→128,s2 3×3×128
conv3_x	16×16	3×3×384 3×3×384	3×3×384 3×3×384	3×3×256 3×3×256	3×3×128 3×3×128
conv3	8×8	3×3×384→768,s2 3×3×768	3×3×384→768,s2 3×3×768	3×3×256→512,s2 3×3×512	3×3×128→256,s2 3×3×256
conv4_x	8×8	3×3×768 3×3×768	3×3×768 3×3×768	3×3×512 3×3×512	3×3×256 3×3×256
conv4	4×4	3×3×768→1536,s2 3×3×1536	3×3×768→1536,s2 3×3×1536	3×3×512→1024,s2 3×3×512	3×3×256→512,s2 3×3×512
conv5_x	4×4	3×3×1536 3×3×1536	3×3×1536 3×3×1536	3×3×1024 3×3×1024	3×3×512 3×3×512
linear	3×3	average pool, s 2, pad 1; $((1536, 1024, 512) \cdot 9) \rightarrow 10$ fc; softmax			
GPU RAM		2060 MiB	4135 MiB	5249 MiB	4356 MiB
score, time		64,54%, 290 min	78,22%, 563 min	76,80%, 712 min	74,13%, 530 min

Table 5.1: (all convolutions use a stride of 1 and padding of 1, if not labeled otherwise)

On item 2.: The network of the second experiment has the same parameters for filter sizes, strides and padding as the network in the previous experiment but it has no layer shortcuts and the fully connected layer is converted to a equivalent two step convolution.

On item 3.: At first this network uses the same convolutional part as experiment 2 and then appends the deconvolutional part, in addition to that all filter sizes are changed to 4×4 to combat checker board artifact effects in the deeper 18-layer version.

layer name	output size	10-layer	18-layer
conv1-5 part	3×3	⋮	⋮ (with 4×4 filter)
linear	4×4	filter 1×1×depth 10→512; average pool, stride 1, pad 1	
deconv1_x	4×4	3×3×512 3×3×512	4×4×512 4×4×512
deconv2	8×8	3×3×512→256,s2 3×3×256	4×4×512→256,p2 4×4×256
deconv2_x	8×8	3×3×256 3×3×256	4×4×256 4×4×256
deconv3	16×16	3×3×256→128,s2, 3×3×128	4×4×256→128,p2 4×4×128
deconv3_x	16×16	3×3×128 3×3×128	4×4×128 4×4×128
deconv4	32×32	3×3×128→64,s2 3×3×64	4×4×128→64,p2 4×4×64
deconv4_x	32×32	3×3×64 3×3×64	4×4×64 4×4×64
		28×28 max pool, stride 2	
deconv5	32×32	filter 5×5×64→3, stride 1, pad (1,2)	
GPU RAM		3571 MiB	6362 MiB
time		330 min	603 min

Table 5.2: (all convolutions use a stride of 1 and padding of 1, if not labeled otherwise)

On item 4.: The network in the fourth experiment has the same parameters for filter sizes, strides and padding as the network in the third experiment with the difference, that it uses a two step convolution with filter size 1×1×depth 512→2 and a binary relabeled CIFAR-10 dataset.

5.3 Which Data were Gained? Are those Data and Experiments Significant?

The best top-1 error score which was achieved as an result of the first experiment, was 78,22%. For that the 18-layer network from table 5.1 was used. The network needed GPU 5247 MiB RAM and needed 563 minutes of training. However, the original ResNet [HZRS15] has a top-1 error of 6.43% (6.61%±0.16%) which is equivalent to a top-1 error score of 93.57, for that they used a 110-layer network and unlike to the outlined approach they used data augmentation. This is a method to increase the training data by altering the training images during every epoch of training. Picture alterations like shifting, stretching and mask patches of the images. Modern ResNets achieve slightly better results, with top-1 error scores like 94.4% (Resnet-56: [HZRS16]). The best score on the CIFAR-10 dataset a network ever achieved is 96.53% by Fractional Max-Pooling [Gra14] according to the website: [Ben15]. However since 2015 other datasets where

used to test deep learning networks and therefore their might be networks, that would achieve higher scores on this dataset if tested.
But I think, that the score my network achieved in my first experiment is reasonable enough to work with the data collected from further experiments on this network. Including, but not limited to the object tracking in the fourth experiment.

The best top-1 error score which was achieved as an result of the second experiment, was 78.02%, which used the linearized version of the 18-layer network from experiment one. For a linear network without data augmentation this is a good top-1 error score and with a better optimized ResNet for linearization a top score for linear networks is expected to be possible

In case of the third experiment we have to look on the output images to determine how good the object localization works:

Figure 5.1: object localization with option 2

blue label = correct classfication, red label = wrong classification

I choose to overlay the test images with the output of the deconvolution and converted the test images to greyscale colored images to better evaluate the object recognized areas. In this output example only 78% of the images was classified correctly, but after all in 83.33% of the cases the marked object areas lay inside the correct object image space. This indicates, that roughly object localization is more simple for Neural Networks than object classification, though the network confuses objects with other object but detected nevertheless the correct image location for an confused object. Another interesting effect is, that the marked object areas get smaller as better the classification score becomes. I think this is because as better the object classification becomes as smaller the stimulation area of an object needs to be to trigger the classification. The results of this experiment cannot compete with deep learning segmentation networks, but they have the advantage that they don't rely on ground truth information. I didn't calculated the score for all correct marked pixels on all images of the cifar-10 dataset,

Figure 5.3: Cat or no-Cat Net

all size layers are merged together and added up the marked highlight area of the cat

because I don't posses the ground truth information to compare the marked pixels and the score wouldn't be as good anyway, but I counted the correctly marked objects, where most of the marked pixels lay inside the correct object image space on 1000 images. 82.5% images were correctly marked by the deconvolution network without any further training of the deconvolution part. This might be enough to try out a reasonable object tracking with this approach in experiment 4.

A big advantage of object detection vs object classification is, that we can train one specific network to scan an image for one specific object. So the network doesn't have to handle multiple recognitions at once and can concentrate on the localization of one object alone. In case of the network in experiment 4 the cat detection score is 95%. To visualize how object tracking on a normally sized image would with this approach look like I trained the network with the CIFAR-10 dataset and then let it work with 720p images, which are at first disassembled to 32×32 pixel images by the network and then scanned for cats separately. As you can see the cats are detected correctly by the network, but due to the fact, that on big not for Region of Interest (ROI) cropped images most of the space do not contain cats, false positives are a problem, even with a cat detection score of 95%. But I think with enough images of a video stream object tracking cloud work, because then the network could compare every cat detection with the previous images and just mark the space of a detected object if it is sure, that it detected a cat on this certain location of the images multiple times. The whole thing would actually function in real time with one processed 720p image per second, because the forwardpass of the network is capable of processing 1000 32×32 pixel images per second with the following GPU: NVIDIA TESLA K80 with 12 GiB RAM at 2.5 GHz memory clock. This might not be fast enough for real time tracking of fast moving objects.

Figure 5.5: Cat or no-Cat Net zoomed

The cat is higlighted in red

Figure 5.7: original cat image zoomed for comparison

5.4 Statistical Evaluation

Under normal circumstances it is mandatory to repeat an experiment for a statistical evaluation and calculate its standard deviation and the mean, but in case of deep learning I got every time I trained the same network with different random determined starting weights nearly the same results with a standard deviation under 0.02%, which is of course a non negligible deviation, but to evaluate this small deviation reasonable enough too much trainings cycles for every different network have to been absolved, which would require resources I don't have. The original ResNet has a standard deviation of 0.16 on the same dataset I used, which is most likely a result of their data augmentation I don't used.
In contrary to this very low deviation in the top 1 error score, there is of course the risk, that the network training parameters are chose badly and the network gets stuck during the training process, but if a network achieves a good score ones it won't get stuck with the same training parameters another time. This is a consequence of the million of weight parameters that are randomized at every beginning of the training, because deep learning networks own such a high amount of weight parameters affecting the network process, that the the impact on the outcome of each weight parameter add up to a regression to the mean effect. So Neural Networks aren't as chaotic and unpre-

dictable as their reputation, which is a good thing for real world technical applications. Because of that, I will focus on network internal statistics instead of evaluating the same network by its output statistically. Network internal statistics, are statistics like the progress of training or the confusion matrix (table 5.3), which shows which object class was confused with which other object class by the network.

	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck	score %
airplane	854	7	10	1	2	5	3	26	74	18	85.4
automobile	3	666	50	74	58	49	32	8	11	49	66.6
bird	3	59	623	47	163	50	21	10	13	11	62.3
cat	3	56	48	736	26	41	65	6	3	16	73.6
deer	3	28	126	32	737	10	42	6	6	10	73.7
dog	4	32	38	43	23	835	6	5	5	9	83.5
frog	2	25	22	36	63	1	824	0	13	14	82.4
horse	26	4	10	5	6	6	2	873	22	46	87.3
ship	54	3	10	5	3	7	12	21	859	26	85.9
truck	15	29	18	9	9	7	7	60	33	813	81.3
										total	78.2

Table 5.3: confusion matrix

That the network has problems to differentiate between ships and airplanes is no surprise, because most pixels are blue on this images, but there are some interesting confusions to look at. The pictures were trucks and horses were confused was often pictures with a similar greenish background and often with the object placed on a muddy pathway. The background often plays a greater role in classifying the image, therefore I recommend using images with randomly selected backgrounds for every object class on it to form a good classifying dataset. The biggest confusion happened between deers and birds and other animals found in woods. But this isn't just because of the wooden background, what the network often tries to find in animals to differentiate them from planes cars and trucks are the eyes of the animals and faces in general. This patterns are present more or less in all those animals and therefore the network has a hard time to classify them. However cars were also often confused with animals, but horses weren't confused with other animals most of the time. In case of the cars the explanation might be that the lights of the cars mimic certain features eyes owns, too. Horses are present in the CIFAR-10 dataset in two ways. There are closeup images of the heads, which can be differentiate easily from other animals, because the other animals don't have close up images of there heads and there are long shots with the whole horse inside the image. This images have a much smaller head on it than the images of the other animals and on 32×32 pixel sized images the main feature, the eyes, are barely visible at all and therefore easy to differentiate, too.
This result gives insight about the capability of Neural Networks. Neural Networks are able to recognize two totally different views of the same object, which means that two totally different input stimuli can trigger the same classification. This is also observable in case of the truck images. Because they are often from different view angels where some trucks used lorry mounted telescopic cranes and other assets on them, so that

they look totally different. Neural Networks are able to get an understanding of the 3 dimensional expansion of an certain object, so that they are able to recognize rotated objects on pictures, if they were trained for that before.

horisontal axis: epoch of training, vertical axis: score
red line: 18-layer ResNet test score, purple line: training score
blue line: 58-layer ResNet test score, green line: training score

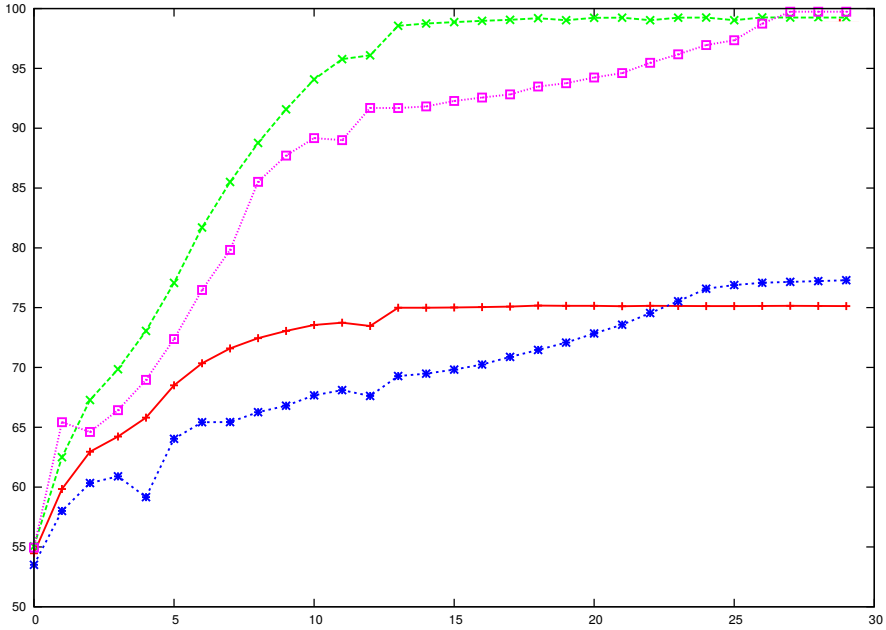

Figure 5.8: training curves wide vs deep

horisontal axis: epoch of training, vertical axis: score
red line: 18-layer ResNet test score, purple line: training score
blue line: 18-layer linearized ResNet test score, green line: training score

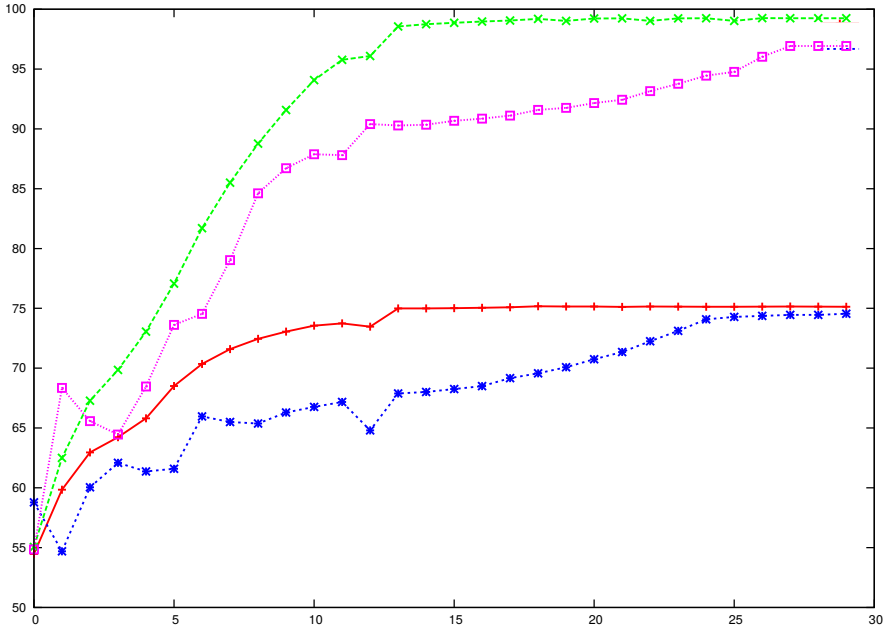

Figure 5.9: training curves linear vs residual

You can see in figure 5.8 that a deeper thinner network in opposition to a wider shorter network needs longer to converge, but is able to reach a higher classification score. The deeper network has more fluctuations in the training and testing score, because small changes in front layers can lead to big differences in the outcome. The leaps in epoch 13 of training are caused to a precision change of the training's rate to prevent the plateau effect.
In figure 5.9 is a comparison between a 18-layer ResNet and 18-layer linearized ResNet without a pretraining. The linearized ResNet reached a close score to the 18-layer not linearized ResNet but needs more epochs of training to reach that score.

5.5 Detail of Implementation

```
require 'nn'
require 'cunn';
local model = {}
local res = require 'residual'

function model.residual(N,L)
    local N = N or 15
    local L = L or 1
    local half = true

    local net = nn.Sequential()
    net:add(nn.Reshape(3,32,32))
    res.convunit(net,3,64*L)
    net:add(nn.SpatialMaxPooling(2,2, 1,1))
    res.convunitN(net,64*L,N)
    res.convunit2(net,64*L,half)
    res.convunitN(net,128*L,N)
    res.convunit2(net,128*L,half)
    res.convunitN(net,256*L,N)
    res.convunit2(net,256*L,half)
    net:add(nn.SpatialAveragePooling(2,2, 2,2))
    cls = nn.Sequential()
    local wid = 3
    cls:add(nn.Reshape(512*wid*wid*L))
    cls:add(nn.Linear(512*wid*wid*L,10))
    cls:add(nn.LogSoftMax())
    net:add(cls)
    local ct = nn.ClassNLLCriterion()

    net = net:cuda()
    ct = ct:cuda()
    return net,ct
end
return model
```

This Torch7 script (model.lua) forms the layout of the thesis ResNet. In Torch7 it is as easy a adding the different layers one after each other to the network.

`res.convunitN ( net ,64* L , N )`
defines one convolutional layer, where 64×L defines the number of mini batch layers. L and N are the parameters for the width and depth of the network, which means that the convunit with 64×L mini batches is added N times to the sequential network `net`.
`res.convunit2 ( net ,64* L , half )`
This command adds the mini batch transition from 64×L layers to 128×L layers to the network. This operation only adds one convunit to the network.
`cls : add ( nn.Linear (512* wid * wid *L ,10))`
This command adds the fully connected layer to the network with 512×wid×wid×L×10 weight parameters, where "wid×wid" is the spatial dimension of the network before it is passed to the fully connected layer.
One convunit of the network is defined in residual.lua and looks like this:

```
local function convunit(net,fin,fout,fsize,str,pad)
    local pad = pad or 1
    local str = str or 1
    local fsize = fsize or 3
    --layer one of building block
    net:add(nn.SpatialConvolution(fin,fout,fsize,fsize,str,str,pad,pad))
    net:add(nn.SpatialBatchNormalization(fout))
    net:add(nn.ReLU(true))
    --layer two of building block
    local str = 1
    local fin = fout
    net:add(nn.SpatialConvolution(fin,fout,fsize,fsize,str,str,pad,pad))
    net:add(nn.SpatialBatchNormalization(fout))
    --no second relu?
end
```

A convunit (convolutional unit) denotes one residual building block of the ResNet illustrated in figure 4.2. The shown program code of the convunit adds two convolutional layers to the network with a following batch normalization. In additon to that follows after the first convolution a ReLU layer. However the shown code just applies to the first convunit of the network, which does not contain the shortcut connection illustrated in the figure of the residual building block. To add the shortcut we need the following program code:

```
require 'nn'
require 'cunn'

local function convunit(net,fin,fout,fsize,str,pad)
[...]
end
--linear part of one resiudal building block
local function convunit2(net,fin,half)
    local half = half or false
    if(half) then
        --doubling the number of mini-batches
        convunit1(net,fin,2*fout,3,2,1)
    else
        convunit1(net,fin,fout,3,1,1)
    end

end
local function resUnit(net, unit, fin, half)
    local half = half or false
    local net = net or nn.Sequential()
    --split in two parallel paths
    local cat = nn.ConcatTable()
    cat:add(unit)
    if(half==false) then
        --shortcut
        cat:add(nn.Identity())
    else
        --shortcut with mini-batch number adaption
        cat:add(nn.SpatialConvolution(fin,2*fin,1,1,2,2))
    end
    net:add(cat)
    --element wise addition of the two paths
    net:add(nn.CAddTable())
    net:add(nn.ReLU(true))
    return net
end
--one residual buidling block
local function convunit2(net,fin,half)
    local unit = nn.Sequential()
    convunit2(unit,fin,half)
    resUnit(net,unit,fin,half)
end
```

this line:
`convunit2 ( net , fin , half )`
defines a function that calls
`convunit2 ( unit , fin , half )` and
`resUnit ( net , unit , fin , half )`,
where “convunit2” forms the convunit shown above and “resUnit” is adding a parallel

network path to the convunit to create the shortcut connection. “convunit2” does also check if it is called with the half parameter, which means it has to increase the mini-batch size of the convolution.
To add N number of convunits to the network the following code is added to the file above:

```
local function convunitN(net,fin,N)
    local N = N or 0
    --n residual building blocks
    for i=1,N do
        convunit2(net,fin)
    end
end

local res = {}
res.convunit = convunit
res.convunit2 = convunit2
res.convunitN = convunitN
return res
```

the for loop adds N times of residual convunits (with shortcuts) to the network. To path the Lua tables with the whole convunits back to the model.lua program code we have to add them into a table `res` and return the table, so that the command
`local res = require 'residual'`
in the model.lua file can acquire the convunits.
If we want to build a deconvolutional network we can use the Torch7 function
`nn.SpatialFullConvolution(...)` instead of our
`nn.SpatialConvolution(...)` function.
In addition to that we have to invert the order of the different convunits.

The main function with some comfort features, the training's function and the deconvolutional network implementation of the thesis approach can be accessed at the public GitHub with the url:

`https://github.com/Arcademiker/deconvResNet`