Deep learning based head localization with Tensorflow

Udacity Machine Learning Nanodegree – Capstone Project – Christian Freischlag 21.12.2016

1 INTRODUCTION

In 2016 hardly a day goes by, without a news article about artificial intelligence or deep learning. One of the major reasons for this growth is the progress of deep convolutional neural networks in computer vision, which are currently outperforming all traditional methods of classification or regression on image data.

This work faces a typical application of deep learning in the context of computer vision, called object localization. Object localization describes the problem of finding the pixel position of an object in an image. To achieve this, the problem is defined a regression problem and implemented in Tensorflow.

2 DATA

Deep Learning is strongly restricted by the amount of existing data to train the covNet[1]. To enable our network to predict head positions in an image, we need a training dataset, which contains people and annotated head positions. Furthermore, it is very important to find a data set, which is already big enough and free to use.

2.1 THE MPII DATA SET

The MPII data set consists of more than 25,000 images, which contain over 40,000 annotated persons and is designed for 2D pose estimation. This data was selected based on a taxonomy of 800 activities and contains 410 different activities, which are also part of the annotation. This data set was extracted from youtube videos and contains many different perspectives and occlusions.

[M. Andriluka, L. Pishchulin, P. Gehler und B. Schiele, „2D Human Pose Estimation: New Benchmark and State of the Art Analysis," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 06 2014.]

2.2 EXPLORATION

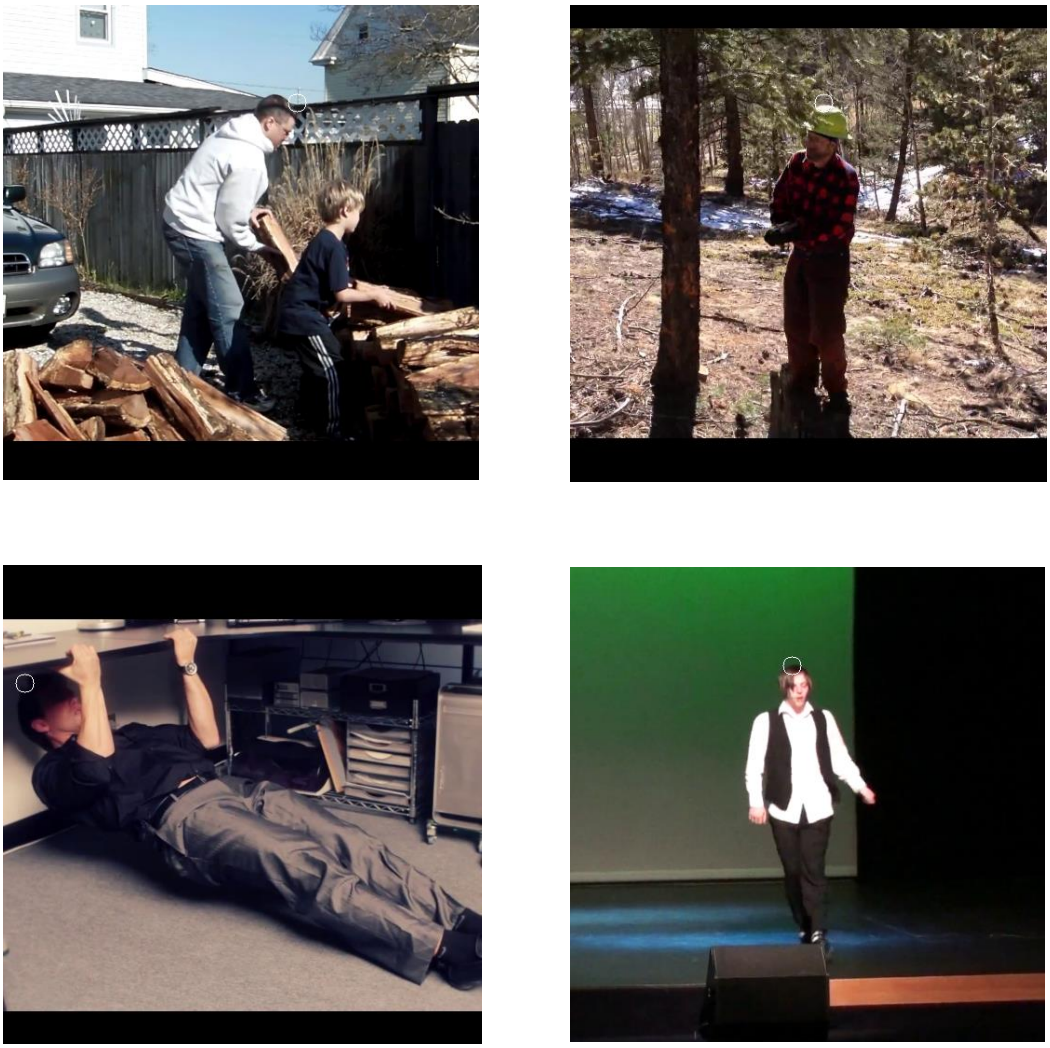
The dataset can be browsed on the creators website:

http://human-pose.mpi-inf.mpg.de/#dataset

[1] covNet, CNN or convolutional neural network

2.3 PREPROCESSING

The data is not preprocessed in any way, so aspect ratio and size is different for many of the images. To train a covNet, it is necessary to have equal sized images and it would also be beneficial crop and rescale the image and extract the annotations.



This preprocessed images where cropped and resized from the mpii dataset. The circle marks the head annotation.

To achieve such a preprocessing, the images are provided through a preprocessing pipeline in python. Basically, the images are cropped around the head position and surrounded by black borders, in the case the images would not be big enough to cover the cropped area.

3 TECHNIQUES

Deep convolutional neural nets consist of three core components, which are stacked to an architecture depending on the task. These three components are called convolutional layer, the pooling layer and the fully connected layer. While the first two layers are combined and are responsible for abstraction of features and generalization, the last layer is the classification layer.

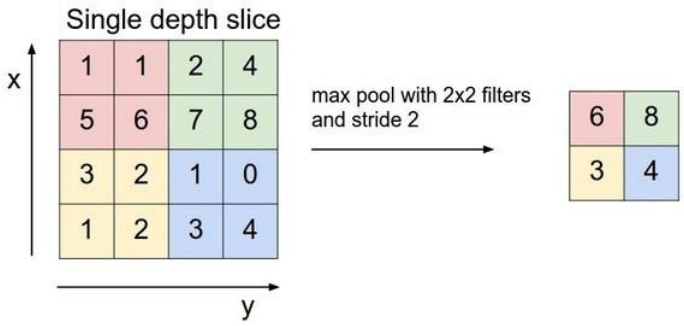
In this project, Tensorflow is used to define and implement such a deep neural network.

3.1 CONVOLUTIONAL LAYERS

The convolutional layer is the name giving layer, which applies a convolutional kernel to and images, extracting a feature. Several of these kernels are applied and the generated new "images" are called a feature map. The weights of the kernel are trained in the training phase and define the internal feature representation of an image.

3.2 POOLING

The pooling layer is the aggregating and generalizing layer. In the most cases a max pooling is proposed and this is what happens in this project too. In this picture shows, how max pooling works.



H. Lee, „Unsupervised Feature Learning Via Sparse Hierarchical Representations," Stanford University, 2010.

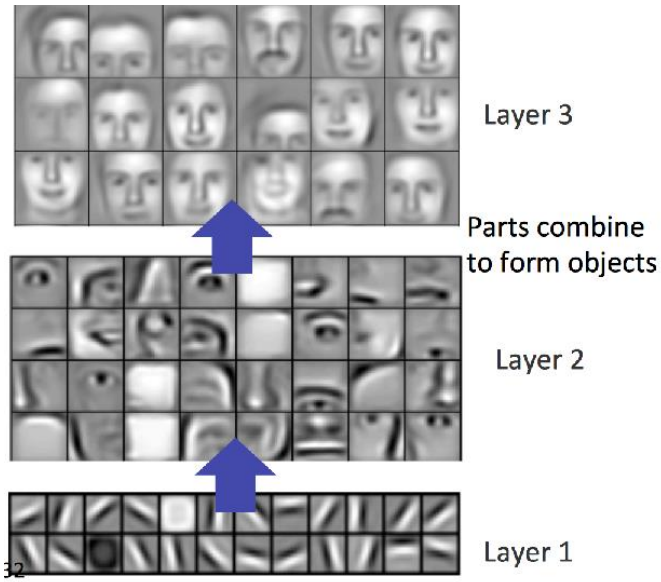
3.3 FULLY CONNECTED LAYER

This layer gathers all the features and does the regression/classification, based on the most abstract features. Sometimes there are several of these layers in succession, to make the classifier more complex. Because of the layer is fully connected and all these connections can be trained, this layer is usually responsible for the majority of free parameters and subsequently the training duration.

3.4 ABSTRACTION

The picture below shows how the abstraction process in convolutional neural networks work.

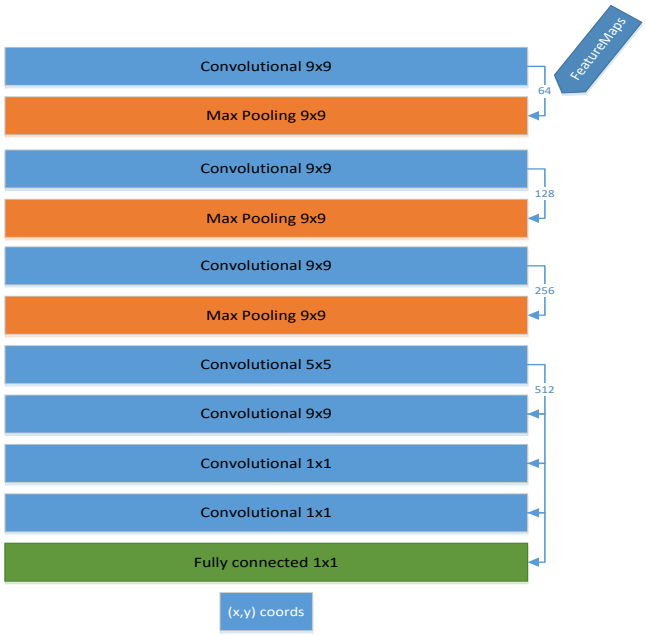
In Layer 1, there are features representing edges, therefore they are called low-level features, as they are very close to the raw data. The second Layer is constructed on these Layer 1 features and combines them to new, more abstract, features, called high-level features. Layer 3 then does the classification into different faces and outputs the class.



Compvisionlab, „Convolution – OpenCV,“ 07 04 2013. https://compvisionlab.wordpress.com/2013/04/07/convolution-opencv/.

3.5 ARCHITECTURE

In this work a simple network architecture is used, inspired by CIFAR-10.



4 METHOD

4.1 FRAMEWORK

Tensorflow is a framework for GPU accelerated deep learning applications. In this work a Geforce GTX 1080 is used to speed up the processing.

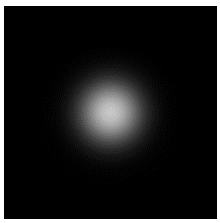
Not only Tensorflow is used for this project, there is also a extension toolbox called tensorpack[2]. Tensorpack provides some additional functions to make the development more comfortable. e.G. it provides an easy way to define network architectures like in this example:

```
.Conv2D('conv1.1', out_channel=64) \
.MaxPooling('pool1', 2, stride=2, padding='SAME') \
.Conv2D('conv2.1', out_channel=128) \
.MaxPooling('pool2', 2, stride=2, padding='SAME') \
.Conv2D('conv3.1', out_channel=256) \
.MaxPooling('pool3', 2, stride=2, padding='SAME') \
.Conv2D('conv4.1', out_channel=512, kernel_shape=5) \
.Conv2D('conv4.2', out_channel=512, kernel_shape=9) \
.Conv2D('conv4.3', out_channel=512, kernel_shape=1) \
.Conv2D('conv4.4', out_channel=512, kernel_shape=1) \
.FullyConnected('fc1', out_dim=2, nl=tf.identity)())
```

4.2 PROBLEM DEFINITION

In the Mpii data set there are several images, which include more than one human. If the network is trained to predict 2D coordinates of the head´s position, there is the problem of training with multiple labels. If an image shows more than one single person, but the label only marks one position of the existing heads, the network can´t determine which head is meant. This would lead to worse accuracy. There is also no easy way of making the labels dynamic, to detect multiple positions.

Thus, the common approach to solve this is to create an activation map. Activation maps are basically images of feature probabilities. To transform the position labels to these activation maps the Gaussian normal distribution around the center label position is generated as an image. The network is trained to find a prediction image, which fits the training image.



$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}$$

Training image (l) and Gaussian label image (r)

[2] https://github.com/ppwwyyxx/tensorpack

4.3 METRICS AND IMPLEMENTATION

4.3.1 Training error

In Tensorflow there are many predefined metrics, which can be used to measure the error, while training the covNet. The simplest one is the sum of squared error, which tends to explode, when costs are calculated, so the mean squared error is often used, which does roughly the same. Another more advanced cost metric, which is often used in covNets, is the cross-entropy loss.

4.3.1.1 Regularization

Deep neural networks suffer from the exploding or vanishing gradient error. To handle this problem, its common to use l2 regularization, which adds and penalty term to the cost function and basically penalizes big weights. To avoid the vanishing gradient a special non-linear activation function called rectified linear units (ReLU) is used.

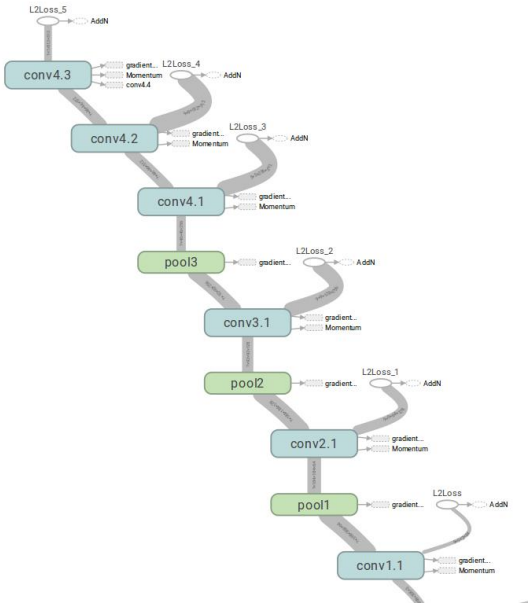
4.3.2 Hyperparameters

In CNNs hyperparameters are used to define parameters, which can also be adjusted to achieve a faster convergence and performance. (common defaults)

- learning rate (0.01)
- l2 penalty (0.001)
- batch-size (10)
- epochs (2)
- activation function (ReLU)
- dropout (active)

4.4 IMPLEMENTATION

The network was implemented in Tensorflow, which offers a visualization tool for networks. The following image shows the core of the visualization. It shows the data flow and where costs and gradients are calculated and which optimizer is used. In this case, it shows that the momentum optimizer is used and that gradients are calculated on every pooling and convolutional layer. Furthermore, the l2 loss is calculated only on the convolutional layers and summarized.



5 RESULTS

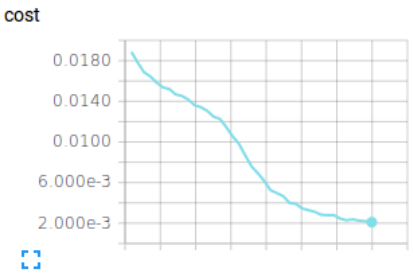
5.1 PROOF OF CONCEPT

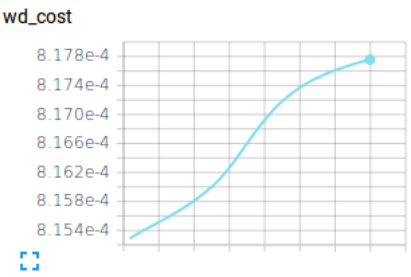
The first iteration only used one single head label to train the covNet. The learning rate was set to increase after several epochs, to prevent the network from oscillation and enable it to converge. No advanced optimizer was used (Adam). After 15 epochs, which took about 4 hours, the network was trained to find some reasonable predictions.

Results from the proof of concept: input images (upper), activation maps (lower), circle ground truth

5.2 RESULTS

In the final training, all heads were used to train the network. In the final training a l2 penalty of o.1 * 10^{-6} has been used to lower the importance of l2 regularization. In the first training the l2 error was dominating, so this was an adjustment, which made the results better. The final learning process worked very well, as the left image of the cost minimization shows. The other image, shows the l2 regularization costs. It is a typical behavior, that regularization cost increase, while the network is learning and minimizes the overall costs.





6 CONCLUSION

This works has shown, how a complex computer vision task is solved by a deep convolutional neural network with Tensorflow. The results are satisfying and show that deep learning provides the opportunity to detect and localize human heads, without putting very much effort in a custom model and feature engineering. It also should be remarked that this end-to-end approach is only applicable in cases, where enough data and the hardware is available.

6.1 IMPROVEMENTS

There is a lot of space for further improvements. The network architecture was not verified and hyperparameters were not optimized. There is also the possibility to extend the augmentation to enhance the variance of the training data.

6.2 REFLECTION

One thing, which is not covered in this documentation is the installation and correctly setup of Tensorflow with CUDA and CuDNN (NVIDIA CUDA® Deep Neural Network library), which takes a lot of initial time. Also, the extraction of the Mpii data set annotations and the preprocessing is very time consuming. The implementation of the covNet in Tensorflow is written very fast and works in the most cases after some tuning.

In the end the results are very astonishing for me, because aside from deep learning these results would take a huge effort to achieve. I would be interested in the visualization of the internal feature representations and may take a look into it.