Bachelor Thesis

Semantic Segmentation of human knees using Convolutional Neural Networks

Paul-Louis Pröve

mail@plpp.de

supervised by

Prof. Dr. Dennis Säring

dsg@fh-wedel.de

Hamburg,

July 17, 2017

Contents

1 Abstract

Password attacks are at the edge of accessing someones secrets. By learning to judge the strength of a password and by understanding how hackers execute attacks, users can make better estimations on how safe they are.

The entropy is widely used to measure how safe a password is, but many sources draw inaccurate conclusions between the entropy of a random password and the strength of a password that was chosen by a person. It is important to understand how these two differ and why realistic password strength is often hard to determine.

Todays hardware gives hackers incredibly powerful machines to launch different types of password attacks. Common password patterns lower possible permutations by such a magnitude that even seemingly safe passwords can be successfully attacked. In combination with frequently used passwords and personal information, hackers can further increase the effectiveness of their attacks.

By explaining common terminologies and analysing different datasets we will look at password attacks from the perspective of users, system administrators and hackers. All three benefit by understanding how the others operate in practice.

2 Introduction

Motivation

Recent advances in Artificial Intelligence have led to workflows that not only run fully automated but also often exceed human performance. State of the art neural networks can classify images into thousands of categories more accurate and magnitudes faster than humans. They translate text between hundreds of languages, navigate cars autonomously through cities and detect intruders in computer networks. In most of these cases they have been trained on tens of thousands or even millions of data samples.

Neural networks have also found great success in the medical field,where new types of problems were introduced based on datasets that are much smaller. Medical imaging data includes MRI, Ultrasound and CT of which the latter is an invasive methods because it uses electromagnetic radiation.

3 Background

3.1 Medicine

write something

3.1.1 Magnetic Resonance Imaging

text

3.1.2 Epiphyseal Plates

text

3.2 Artificial Neural Networks

Neural Network

3.3 Terminologies

Explain popular vocabularies here

3.3.1 Convolutions

there shall be text

3.3.2 Segmentation

texty text

4 Methods

4.1 Setup*

The workstation for this study included an Intel i5 processor, 16GB of RAM and most importantly a NVidia Geforce GTX1060/6GB. Neural Networks can be trained more effienctly on GPUs than CPUs. This is because the simpler but highly parallized architecture of graphics chips plays in favor for the needed calculations in deep learning.

The workstation ran Ubuntu 16.04 with the CUDA and cuDNN libraries installed in order to take advantage of the GPU. As the main programming language Python 2 was chosen due to its simple syntax and popularity in the deep learning field. The code of this project is compatible with Python 3 as well. Keras was used as the framework for training models, because its top level syntax allows fast prototyping and testing. The development environment was a Jupyter Notebook, which allowed a flexible execution of code snippets instead of running the entire program for every single change.

The processing of medical image data needed a library that could handle these formats. SimpleITK is a Python binding of the ITK library written in C++. It includes many tools for image processing and is especially popular in the medical field. Other libraries were also used for smaller task. A complete listing can be found on the project's GitHub page, where the entire code is available.

4.2 Data Analysis

The dataset was a collection of three dimensional MR images showing human knees. The number of available samples grew during the project. For most of the development time it included 150 images that came from 3 different sources.

Source	Prospective	Perspective	Samples	Maps	Resolution
Epi	Yes	Coronal	80	40	800x800x41
Jopp	No	Coronal	65	36	512x512x24
Maas	No	Sagittal	5	0	multiple

Table 1: Details of available images sorted by their source

german males, 14-21, different resolutions, coronal/sagittal, segmentation maps for 76, mhd files, distribution of age.

4.3 Preprocessing*

The number of parameters in a Neural Network commonly range from hundreds of thousands to hundreds of millions. This complexity allows the model to learn on its own what features of an image are relevant for any given task. It works in conjunction with the fact that high volumes of data are available for the training.

Because of the small dataset that was available for this study, several types of data preprocessing were applied to the images. These techniques do one of three things:

- Decrease the amount of information per sample
- Decrease the variance between multiple samples
- Increase the total number of samples

Other preprocessing methods experimented with the difference between 2D and 3D data as well as the influence of separate segmentation channels on the output.

4.3.1 Cropping, Resizing and Resampling*

The framing of the raw images included large parts of the thigh and shin to be visible in the picture. Since these weren't relevant for the purpose of the study, they were cropped out. An algorithm was used to detect the center where Tibia and Femur meet and only use a square window around this point.

Although there is no theoretical size limitation to using convolutional neural networks, it is desirable to reduce the spatial resolution to decrease the amount of calculations. 224x224 pixels for width and height still gave enough detail to identify the shape of Femur, Tibia and Fibula.

Resizing the z-axis was problematic, because the resolution was roughly 20 times lower. When scaling along this dimension the segmentation maps of different slices started to blend together and create merged maps of multiple layers. In order to get the images from two different main sources on the same scale, the 41 slices per image of the Epi data were padded with empty pixels to 48 slices. Afterwards every second 2D image was taken to resample to the same 24 slices the data from Jopp et al. showed. It was not possible to do it the other way around and upscale 24 slices to 48, because the interpolated segmentation maps would have been misleading and false.

Throwing away perfectly good data is very uncommon in the machine learning field, especially if datasets are rather small. However, one slice shares

a lot of similar information with its neighboring slices and can be seen as a sort of image augmentation between the two. By using the same spatial resolution for both sources the balance between the amount of Epi and Jopp data was kept.

A total of 76 24x224x224 images were now available for training.

4.3.2 Normalization*

The normalization of images refers to the process of transforming all samples on the same scale of values. Two techniques for this are popular in the field of deep learning. The first one is called feature scaling, where every sample is normalized between 0 and 1.

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

The second technique calculates the standard score, where the mean is subtracted from the intensities and then divided by the standard deviation.

$$x' = \frac{x - \mu}{\sigma} \tag{2}$$

In this case the mean and standard deviation are not calculated for every image, but for the entire population of training samples. This centers the intensities around the average brightness. When normalizing validation and test sets, it is important to use the mean and standard deviation of the training data, instead of calculating them on their own values.

In theory the standard score is a more desirable approach, because it can help to fight exploding and disappearing gradients in a more effective way. In practice it led to the same results as using feature scaling,which was then chosen due to its easier normalization handling.

4.3.3 N4 Bias Field Correction*

A bias field is a low frequency nonuniform intensity that is an unwanted byproduct Magnetic Resonance imaging. Several methods have been developed in the past of which N4ITK [2] is the de facto standard in the field. This algorithm approximates the nonuniform field and balances intensities using B-Splines. As the name suggests it is included in the Insight Toolkit (ITK) for which SimpleITK delivered an available Python binding. Using the N4 Bias Field Correction with its default settings on a single 24x224x224 image took between 60 and 120 seconds.

4.3.4 Augmentation*

Image augmentation is a popular approach to virtually increase the size of the dataset. The general idea is that a neural network will overfit more when learning one sample n times, than learning n alternations of this sample just once. It helps to generalize on new images instead of memorizing patterns in the training data.

Lossless augmentation techniques are those that don't change the values and relative localities in a sample. In 2D these include vertical and horizontal flips, as well diagonal flips if width and height are equal. Note that any multiples of 90 degree rotations can also be created using a combination of these flips. Although the samples are changed as a whole, they do not vary in respect to their contained values. The term l̈osslessönly refers to the technical change and not necessarily to the semantic change. Flipping a slice of this dataset vertically switches the absolute positions of Femur and Tibia. This might make it harder for the network to distinguish between the two.

Lossy augmentation methods include a variety of image transformations. Common choices include:

- Horizontal shifts
- Vertical shifts
- Rotations
- Shear mapping
- Brightness adjustments
- Contrast adjustments
- Gamma adjustments

For the purpose of this study only horizontal flips were realized to give the impression that images from both knees were available. Other forms of image augmentation were not necessary because the use of Dropout was an extremely effective measure to prevent overfitting.

4.3.5 2D and 3D data*

Although convolutional neural networks are most commonly connected with 2D data such as photos or drawings, they can be applied in any dimensional space. In Natural Language Processing (NLP) 1D convolutions are often used on sentences and in finance they can be applied to time series forecasting. In the medical field where a variety of volumetric data exists, 3D convolutions have become a popular choice for building deep learning solutions.

In the context of neural networks one has to differentiate between volumetric data with one channel and 2D data that consists of multiple color channels. Both are example of 3D data, but the volumetric images features three spatial dimensions, whereas photos feature only two. Convolutions commonly traverse the spatial dimensions, which means photos with multiple channels are usually not subject to 3D convolutions. Instead multiple input channels are fed into a 2D CNN.

Although the dataset onsisted of volumetric MRI data, the z-axis showed a 20 times lower resolution than the other two. Not knowing the influence of this situation both 3D and 2D architectures were investigated for this project. The 3D model didn't use any MaxPooling on the already small z-axis. A kernel size of 3x3x3 was used similar to the 3D version of U-Net. The 2D data was created by using each of the 24 slices as a single input image.

While in theory the spatial information of the z-axis gave the 3D network more contextual information of every slice, the 2D model performed better at the end. The latter was also less computationally expensive and allowed working with data where only single slices where available per sample.

4.3.6 Separate Bone Maps

The initial segmentation maps included three separate channels for the differentiation between the Tibia, Femur and Fibula.

The initial segmentation maps came with three separate channels for the Femur, Tibia and Fibula. With this information it was possible to train a model that would segment the three bones while still differentiating between them. This helped the accuracy of the prediction opposed to using just a single channel for all of the bones. In places where the Femur and Tibia were very close to one another, the separate channels prevented the closing of this region by the network.

For another experiment the three channels were treated as one to create a network that would segment any type of bone in the image. This resulted in better performance when applied to sagittal images of the knee provided by Maas et al. The network was able to generalize on a situation it wasn't trained on.

4.4 Architecture

In search for a network that would perform well on the segmentation, different architectures were looked at and multiple settings were tried.

4.4.1 Pixel and Image Outputs

Early CNN architectures for segmentation would take small patches of images as an input to predict a single pixel through a classification pipeline. Afterwards a segmentation map was assembled using each of these predicted pixels. This process was very slow and it also prevented the network to have a field of view larger than the inserted patch.

As described in 3.2.2 the segmentation of an image can be seen as the classification of every pixel or voxel

4.4.2 Channels

growth and initial size

4.4.3 Dropout*

Dropout is a popular regularization technique that randomly zeros out a fraction of the weights during training [1]. It is understood that this helps the model to generalize better and reduce overfitting on a given dataset. Well known image classification architectures like VGG16, SqueezeNet or AlexNet use Dropout near the end of the network. Similarly U-Net uses Dropout at the end of the contracting path to prevent overfitting.

Since a single unit of dropout with a rate of 0.5 is common in other architectures, this was also chosen as a first candidate. Other tests included adding dropout between the convolutions on the contracting side,which led to slower training and lower scores. Adding dropout on the expanding side is rather unusual and also didn't perform well in the tests. In the end the initial candidate was chosen for future trainings.

4.4.4 Batch Size

Neural Networks use a process called stochastic gradient descent (SGD) or one of its variants to approximate the gradient on a small fraction of the data. The size of this fraction is called the batch size and describes how many samples are used for a single forward- and back-propagation step.

In the past it was believed that larger batches led to something called the generalization gap (1609.04836), where the accuracy of a model would drop if it was trained on particularly large batches. Recent work by Hoffer at al. (1705.08741) suggests other reasons for this drop in accuracy. While common

batch sizes range from 32 to 256, Goyal et al. showed accurate results using 8192 images per batch when training a model on imagenet (1706.02677).

Depending on the size of the input one may be restricted to smaller batches. In the field of 3D convolutions even one image can take up a majority of the RAM on a workstation.

4.5 Training

The training phase describes the actual learning process of the architecture. After initializing the neural network with random values, several parameters have an influence on the accuracy of the model and how quickly a possible optimum is reached.

4.5.1 Training, Validation and Testing

In order to measure how well a network generalizes on samples it hasn't seen before, the dataset was split up in a training and validation portion. While 80 percent of the data was reserved to make the model learn, the remaining 20 percent were used to measure the results.

In larger datasets it is common to use another fraction of the data as the Test Set, which is used as a second level validation method. By using the validation data multiple times throughout the training, the network may get an implicit view of its content. The test set can then be applied at the very end to measure the final accuracy of the model.

4.5.2 Metrics and Loss Functions*

Metrics are used in deep learning to measure the performance of a model. For example the accuracy is often chosen to describe how well a neural network is doing on a classification task. An accuracy of 0.9 indicates that 9 out of 10 samples are classified correctly.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3}$$

In the formula above T and F indicate whether a prediction was true or false. P and N stand for a positive or negative outcome.

The result of a loss function is a metric that will be minimized during the backpropagation process. In order to be used with gradient descent it needs to be differentiable. That is why the accuracy cannot be used as a loss

function. It is a binary metric that works with true or false values and not with probabilities.

In situations like these a surrogate function is used that has a high correlation with the target metric. For classification problems this is often the cross entropy. Because a segmentation can be seen as a classification for every output pixel, it was also chosen as a candidate for this study.

$$CrossEntropy = Inserthere \tag{4}$$

Another option was the F1 score, which is a specific implementation of the F-Measure when beta is 1.

$$F_\beta score = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} \tag{5}$$

Although the F1 score is commonly applied as a binary measure and therefore not differentiable, a s̈oftv̈ersion can be used that accounts for continuous probabilities. To use it as a loss function where 0 describes the best possible outcome the F1 loss was defined as 1 - F1 score.

$$F_\beta loss = 1 - F_\beta score \tag{6}$$

The value of beta can be adjusted to change the emphasize between precision and recall. Precision describes how much of the predicted area was actually true, whereas recall describes how much of the ground truth was recognized by the model. This can be useful for datasets with high imbalances between classes, like this study showed between Fibula, Femur and Tibia. However, as it turned out the predictions were very well balanced with the F1 score alone.

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

To test whether the F1 loss or the cross entropy showed better results, both loss functions were used in separate runs to compare their segmentations. The F1 trained model showed better results regarding the F1 score and the cross entropy model showed better results on the cross entropy loss. In regards to Precision, Recall, Accuracy and Intersection-over-Union (IoU) the performance of F1 trained network showed higher scores.

Another test run used both loss function at the same time. This was done by feeding the output of the second to last layer into two separate segmentation outputs. Each of these last layers were trained with the F1 loss and cross entropy loss respectively. All previous layers were therefore trained by the sum of both loss functions. This architecture combined the best results of the previous runs in a single model, but also increased the training time per epoch by 30%.

Evaluating these two outputs on the other metrics played in favor for the F1 loss output. Combining both maps didn't improve the scores any further. Since the cross entropy was chosen as a surrogate function and not as a performance metric, only the F1 loss was kept as a loss function for all future tests.

4.5.3 Optimizer

The optimizer

4.5.4 Learning Rate Policy*

The learning rate policy describes how the learning rate is changed throughout the training. With the introduction of adaptive optimizers like Adam or RMSProp there has been a lower emphasize on this topic. Learning rates that were set too high or too low, will be adjusted by the optimizer after a few iterations. Even though this reduces the number of possible defects, a lot of training time can be saved with the right policy.

10 epochs were run at different learning rates to compare initial results and to examine the point at which the model wouldn't converge at all. 0.002 was the highest rate at which the model started training, but 0.001 resulted in the best score.

After the model stopped to improve at epoch 65 the learning rate was changed to 0.0001. This gave a small boost of accuracy. In order to have a smooth transition between learning rates, the decay was set to 0.001. This meant that the initial learning rate of 0.001 would reach 0.0001 after 68 epochs and then continue to decrease even further.

4.5.5 Early Stopping*

Neural networks will continuously minimise the loss on the training set. This result needs to be validated on data the network hasn't seen before. At a certain point during training the performance on the validation set will start

to decrease, because the model is overfitting on the training data. The number of iterations to reach this point is dependent on many hyperparameters, as well as the random values the network has been initialized with. As such it's difficult to calculate how many epochs the training will need to reach its peak.

Early stopping is a simple technique that will end the training process as soon as the model stops improving on the validation data. In order to do this, a patience is defined how long the network should continue training after the score has stopped increasing. This is important because not every epoch will lead to a new best score on the validation data.

For test runs in this study a patience of 9 was selected, which meant the training would stop after 10 epochs without improvement. Depending on the architecture and other hyperparameters this point was reached after 30 to 60 epochs. For the last training with the final set of hyperparameters the patience was increased to 19, which didn't improve the accuracy. This was also a verification that the initial value of 9 was a good fit for this problem.

5 Results

best model performance, parameter count, size

6 Discussion

age prediction, size of data, augmentation

References

[1] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[2] Nicholas J. Tustison, Brian B. Avants, Philip A. Cook, Yuanjie Zheng, Alexander Egan, Paul A. Yushkevich, and James C. Gee. N4ITK: Improved N3 bias correction. *IEEE Transactions on Medical Imaging*, 29(6):1310–1320, 2010.