Classification of images with Convolutional Neural Networks

This scripture documents the work done by Aditya Raj () and Sören Schleibaum (474562) for the course Neural Networks with Statistical Learning at the Technical University of Clausthal. The date of committing is the 13th of March 2017.

Kommentiert [SS1]: Please add your imatriculation number here.

1 STRUCTURE

1 Structure 2
2 Introduction 3
2.1 Problem 3
2.2 Neural Networks 3
2.3 Convolutional Neural Networks 3
3 TensorFlow 6
4 Description of programming code 7
4.1 System 7
4.2 Graph 7
4.3 UML-Diagram 7
4.4 Description of modules 7
5 Evaluation 8
5.1 Environment 8
5.2 Accuracy 8
6 Summary 9
7 References 10
8 Appendix 11

2 INTRODUCTION

- Define Deep Learning
  - Training data
  - Test data
- Define Neural Network
  - Define Neuron
    - Each neuron performs a dot product and optionally follows it with a non-linearity
  - Consist of different layers
    - Neurons connected via edges
    - Each edge has a weight
    - Weights are initialized with small random values
  -
- Define Convolutional Neural Networks
  - Neural Networks
    - Convolutional Neural Networks are special part of that
    - CNN are feed forward networks
  -
- Define Classification (unequal to clustering)

2.1 PROBLEM

A common problem for CNN is the classification of images. We chose to classify between cat and dog images. The dataset used for this course was provided by Kaggle[1], a host for competitions in machine learning. The aim is to predict if an image shows one of the two classes cat and dog.

The training dataset consists of 25,000 images, 12,500 for each class.

- Number of images which are doubled from hand checking should be inserted here (can be found within the presentation)
- Varying size
- Not all images show only one cat nor dog
- Some images are only black

2.2 NEURAL NETWORKS

2.3 CONVOLUTIONAL NEURAL NETWORKS

Due to the lower number of parameters within the network CNNs are preferred above NN in case of image classification. This reduces the time used for training. The reason for the decreased number of parameters is the insertion of a convolutional layer. Beside this one, the following text describes briefly input, pooling, normalization, fully-connected, softmax linear, and output layer. Neither has every CNN all the listed ones nor are these all possible layers. We have chosen to describe these ones shortly because of their use within the network used to classify images of cats and dogs. The structure of the CNN shown in Figure 1: Structure of the implemented CNN has multiple appearances of convolutional, pooling, and normalization layer. Within neural networks several techniques are used two perform the

Kommentiert [SS2]: Fields should look like this.

[1] kaggle.com

task of classification. The concept of an activation function and of backpropagation are described within this work.

Activation functions are used after the dot product of a neuron is calculated and the bias added. They are used within convolutional and fully-connected layers. A common choice for constructors of NN was the sigmoid function. It maps a real number to the range between [0,1] and has the form

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

and fall out of favor because of different reasons. In case of an output close to 0 or 1 the gradient will become very small. This means only a small update of the parameters during backpropagation. Another activation function is the ReLU function. Its formula has the following form:

$$f(x) = \max(0, x)$$

This means that all outgoing values are at minimum 0. The function is continuous but not differentiable everywhere. In comparison to the sigmoid function it speeds up the gradient descent and is faster to compute due to its simple form.

Backpropagation is used to update the weights within the network. Based on the loss between prediction and actual value which the training data has, the local gradients are calculated. With an application of the chain rule by going recursively through the network all weights can be updated.

The **input** layer gets the input data and passes it forward to the next layer. Within this layer no computation is performed.

In traditional NN fully-connected layers are used over the whole range of the network. Within CNN the **convolutional** layer replaces fully-connected layer especially in the early layers. The main reason for is the reduced number of weights. Because of this reduction, the training via backpropagation is simplified and the overall training time is decreased (Karpathy).

The **pooling** layer reduces the size of the data by applying a function like max- or average-pooling is computed on the given data. Beside the function to hyperparameters can be chosen: the spatial extend and the stride. When stride and spatial extend are two and the given input is a 4x4, the output data is of dimension 2x2. This means a reduction by factor two for the outgoing data.

Within the **normalization** layer the values are normalized or inhibited. The layer helps to reduce the size of parameters and the excited neurons can fire relatively more. Within the current development of CNN, the normalization layer fall out of favor, because there is little to no documented use.

Fully-connected or **local** layers are used to combine the results of all filters especially in the end. Each neuron in this layer is connected to all outgoing edges of the previous node. It is possible to convert a fully-connected layer into a convolutional layer by increasing the local region of the convolutional layer.

- The **softmax-linear** layer

The **output** layer computes the predicted value for the input data for each class. No activation function is applied here.

With the described layers a CNN can be constructed. Passing through all layers the input data is reduced through several computations until only the prediction is left. In the example of cat and dog classification for each image a possibility for both classes are the output. The initially random weights are updated during the training process. Within each training step calculations on the current input

are performed including dot products, activation functions and for instance max-pooling. After the loss is computed the weights are updated through backpropagation or the learned concepts are applied to the network. The trained model can afterwards be applied to new and never seen data to predict.

3 TensorFlow

To solve the image classification problem described above we have chosen to use the programming language Python[2] version 3 and TensorFlow[3] version 1.0. Within the area of Machine Learning multiple software libraries are available. We have decided to prefer TensorFlow above others because of its multi-language support. Moreover, many functions for creation of neural networks and graphs are already accessible through the package and this library provides documented tutorials. Even if we have not used the support of a Graphic Processing Units (GPU) this might be helpful to scale up the project later.

The software library TensorFlow supports operation used in machine learning. It was developed by the Google Brain Team to support Deep Learning Neural Networks and released in November 2015. Multiple programming languages for instance Java and Python are supported. The library provides a support for computation on both devices, Central Processing Unit (CPU) and GPU (TensorFlow).

The basic concept behind TensorFlow is the usage of graphs which provide the computations inside their nodes and tensors which hold the data. The nodes are connected through edges and the data flows through the graph via Tensors (TensorFlow).

Tensors are a class provided by the TensorFlow library, which can store multi-dimensional arrays. For instance, an RGB image of height and width of 60 pixels is represented as a tensor of shape 60x60x3. The three stands for the three colors values red, green, and blue. The tensors are stored in a format provided by the Python library NumPy[4] (TensorFlow).

Nodes represent computations within the graph model. It can have multiple inputs and outputs. The incoming data from an edge is processed within the node and passed to all its outgoing edges. One example for a node is a pooling layer. When data comes through an incoming edge the pooling function of the corresponding layer is applied and the modified tensors are passed to the outgoing edges (TensorFlow).

The training or later usage of a model works in two phases. Firstly, the graph is constructed. After the graph, has been constructed, the input data can be inserted in the graph. In the second step the graph is placed on a device, for example the CPU. The graph can only perform computations on input data when a session is running. This works by placing the nodes and their operations on a device Since neural networks are trained over time and multiple inputs, the same operations run on the same graph (TensorFlow).

Kommentiert [SS3]: Update link

Kommentiert [SS4]: Line break is bad

Kommentiert [SS5]: Add link

Kommentiert [SS6]: Source is missing

Kommentiert [SS7]: Set up the right link within the footnote

Kommentiert [SS8]: Set source to tensorflow

[2] https://www.python.org/
[3] https://www.tensorflow.org/
[4] http://www.numpy.org/

4 DESCRIPTION OF PROGRAMMING CODE

Code is available at gitlab

4.1 SYSTEM

- TensorFlow
  - Only CPU computation is used due to the lack of
  - During this project version 1.0 together with the Python version 3 language was used.

4.2 GRAPH

The design of the CNN used within this work is shown in Figure 1: *Structure of the implemented CNN*.

- Images are represented as 24x24x3
  - 3 for RGB
- Running of TensorFlow
  - 1. Create the graph
  - 2. Place the graph in a session and run it (it runs multiple times doing the same operation on different data)
- Tensorflow
  - Only CPU computation is used due to the lack of
  - During this project version 1.0 together with the Python version 3 language was used.
-

4.3 UML-DIAGRAM

4.4 DESCRIPTION OF MODULES

- Cifar10.py
- Cifar10_eval.py
- Cifar10_eval_single_directory.py
- Cifar10_input.py
- Cifar10train.py
  - The checkpoints of the trained model are saved as pickled objects
- CreateTestBatch.py
- Helper.py
- ImageInformation.py
- PredictImageInDirectory.py
- Resizer.py
- Settings.py
  - This module stores all global variables to paths, especially to the data passes

Questions

- Do we use dropout? (Reducing the number of parameters)
- We somehow should mention hyperparameters

5 EVALUATION

5.1 ENVIRONMENT

The machine used to train and evaluate the network is 64-bit Linux system with 7.7 Gigabyte of memory. The computations are performed on one core of an Intel Core i7-4510U CPU with 2.00 GHz x 4. Due to the high time used to train the network the test run through the night without performing other heavy computations on the machine.

5.2 ACCURACY

- If the learning rate is too high due to the ReLU activation function parts of the used network can be dead (never activated during training)
- The training data is not perfect

6 SUMMARY

7 REFERENCES

Karpathy, Andrej. *CS231n Concolutional Neural Networks for Visual Recognition*. kein Datum. 07. 03 2017. <http://cs231n.github.io/convolutional-networks/>.

TensorFlow. *Basic Usage*. n.d. 07 03 2017. <https://www.tensorflow.org/versions/r0.10/get_started/basic_usage>.

- TensorFlow documentation
- Stanford course for neural networks
  - Quelle: http://cs231n.github.io/convolutional-networks/
- Python 3 documentation
- Alex Krizhevsky and his description of the net used
- ImageNet paper should be used
- The literature Feng send us should be used

8 APPENDIX

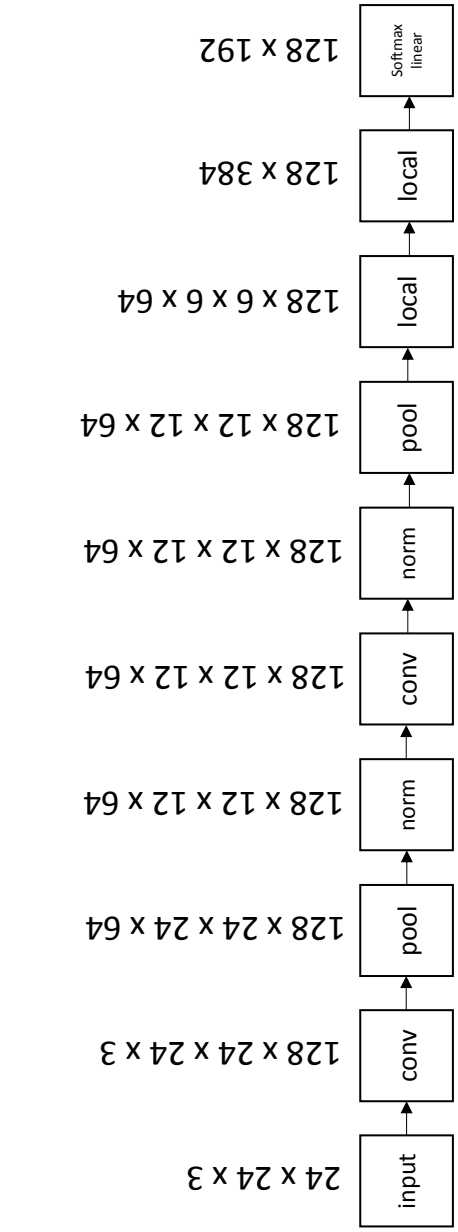

Figure 1: Structure of the implemented CNN