Image Classification with Convolutional Neural Networks

kwh131

30th March 2020

1 Abstract

bla bla bla

Contents

2 Presentation

bla

2.1 Introduction

- *'The first successful practical application of neural nets came in 1989 from Bell Labs, when Yann LeCun combined the earlier ideas of convolutional neural networks and backpropagation, and applied them to the problem of classifying handwritten digits. The resulting network, dubbed LeNet, was used by the United States Postal Ser- vice in the 1990s to automate the reading of ZIP codes on mail envelopes'* [3] p. 15.
- *'The ImageNet challenge was notoriously difficult at the time, consisting of classifying high- resolution color images into 1,000 different categories after training on 1.4 million images. In 2011, the top-five accuracy of the winning model, based on classical approaches to computer vision, was only 74.3%. Then, in 2012, a team led by Alex Krizhevsky and advised by Geoffrey Hinton was able to achieve a top-five accuracy of 83.6%—a significant breakthrough. The competition has been dominated by deep convolutional neural networks every year since. By 2015, the winner reached an accu- racy of 96.4%, and the classification task on ImageNet was considered to be a com- pletely solved problem.' [3] p. 17*

2.2 Problem Formulation

bla

2.3 Clarification of concepts

bla

3 Theory & Concepts

3.1 Pictures

3.1.1 Red-Green-Blue (RGB)

blabla

3.2 Deep Learning

- *'What is transforma- tive about deep learning is that it allows a model to learn all layers of representation jointly, at the same time, rather than in succession (greedily, as it's called). With joint feature learning, whenever the model adjusts one of its internal features, all other fea- tures that depend on it automatically adapt to the change, without requiring human intervention. Everything is supervised by a single feedback signal: every change in the model serves the end goal.'* [3] p. 18

3.3 Architecture of CNN

bla

3.3.1 Depth and width

3.3.2 Training- and Validation Set Error

bla

3.4 Layers

3.4.1 Weights

"The transformation implemented by a layer is parameterized by its weights" - weights are called the parameters of the layer. Learning in a deep network is adjusting the weights in each layer, such that they will map the given input to the correct target value [3] p. 10.

3.5 Tensors

Convolutional Neural Networks operate on *tensors.* Tensors are n-dimensional, often 3-dimensional matrices, that is they have a *height*, a *width*, and a *depth* axis. The dimension of the tensor is called its rank. In the field of image recognition, the height and width dimensions represents the pixel structure of the given images, while the the depth dimension represents the *channels* on which we operate, e.g. RGB-channels (red, green, and blue color intensity in a given image) which gives 3 channels initially. So if we have a small picture, say 32×32 pixels with 3 channels for RGB, then the input tensor would have have dimensions $32 \times 32 \times 3 = 3072$ input values.

'tensors are a generalization of matrices to an arbitrary number of dimensions (note that in the context of tensors, a dimension is often called an axis)'. The dimensionality of a tensor is called its *rank.* [3] p. 31.

3.5.1 Kernel Convolutions

Convolutions work on Have different properties:

1. *Sparce interactions* : Because the kernel is smaller then the input image only a subset of pixels around the pixel in question is interacted with when applying the kernel. This means that we need to store fewer parameters (less memory and more statistical efficiency). As we traverse the networks layers upwards, we get fewer and fewer parameters, but each parameter has more information about the original pixels then in the lower levels. The receptive field for higher layers is increasing.

2. *Parameter sharing* : When applying kernel convolutions, each member of the kernel is visited for every pixel in the input image (except boundaries depending on choice of design). That is we only try to learn one set of parameters for each location in the input image. Reducing the storage requirements and it does not affect the runtime of forward propagation.

3. *Equivariance* : This means that we can apply the convolution to the input image then shifting the kernel or the other way around and we would get the same result. This means that convolution produces a timeline that shows when different features appear in the input.

[4] Convolutions operate over 3D tensors called *feature maps*, with two spatial axis (*height* and *width*) as well as a *depth axis* (also called the *channels* axis). [3]

3.5.2 Pooling layer

Pooling layers help to make the representation approximately invariant to small changes to the input. That is, if we change the input by a small amount we do not get changes in most of the output. Practically you take a subgrid in your feature map and apply the pooling operation to the given grid. The operation could be returning the maximum, taking a weighted average, finding the L^2-norm of the grid etc. Pooling is needed when we need to handle images of the same size, but the training data has images of varying sizes.
One convolution extracts one kind of features at specified locations in the input image. We want to extract many features in the entire input image, why oftenly different set of convolutions is applied to the input image in the same layer of the network. The generates more channels (wider layers). [4]

3.5.3 Rectified Linear Unit

ReLU - Activation functions, such as the sigmoidal function that outputs values between -1 and 1. Harder to train with gradient descent based learning than the ReLU function, $g(\mathbf{z}) = max(0, \mathbf{z})$. bla

3.5.4 Hidden Layers

bla

3.5.5 Softmax Output Layer

bla

3.6 Optimization and Regularization

3.6.1 Stochastic Gradient Descent or RMSProp

Minibatch SDG, where you take a batch of training examples to train the model to take bigger and more inaccurate steps towards a (hopefully) global minimum, but converge faster to that minimum than if you had to compute the gradient for each training example. If convergence is not achieved a possible solution is to use optimization algorithm RMSProp that stores at exponentially decreasing average of previous gradient values to start with large steps and end with smaller, hopefully ending in a convex "bowl" to find local (possible global) minimum.

- *Loss function/objective function*: takes the predictions of the network and the true target and computes a distance score capturing how well the network has done on this specific example. Use the score as a feedback signal to adjust the values of the weights through the network to optimize it [3] p. 10-11.

3.6.2 Back-Propagation in Fully Connected Multi-Layer Perceptron

Explore dynamic programming with back-propagation to avoid computationally costly algorithms. And it takes up less space. On page 214 in Deep Learning by Goodfellow et al. Talk about chain rule for functions and how it is applied to attain the gradient of different parameters with respect to the loss function.

3.6.3 L^2 Parameter Regularization

L^2 parameter regularization: commonly known as wight decay. Add a regularization term, $\Omega(\theta) = 1/2||w||_2^2$.

3.6.4 Dataset Augmentation

Transform training data to generate more ”fake” but relevant data for the CNN. Cropping, resizing, flipping around x-axis or y-axis, stretching, rotating etc. on the pictures can help generate augmented data to train the model.

3.6.5 Early stopping

Keeping track of the validation set error and keeping track of the hyperparameters at each epoch, such that we return the hyper parameters for the model at the point in time where the validation error was the smallest. Reduces overfitting to training data.

3.6.6 Pretraining and parameter sharing

3.6.7 Parameter Sharing between Model Architectures

bla

4 Method

4.1 Empirical Gathering

Finding, reading and listing scientific articles, text books, assignment etc. for use in my Bachelor Thesis

4.2 GPU training

- CUDA - NVIDIA launched

4.3 Tools and packages

4.3.1 TensorFlow and Keras

4.3.2 PyTorch

4.3.3 Theanos

- PyTorch

- TensorFlow

- Keras
- Googles 'Collaboration'

5 Analysis

bla

5.1 Different architectures

bla

5.1.1 Theory Behind Architecture Choices

bla

5.1.2 Manual for architectures

bla

6 Discussion

bla

6.1 Adversarial Training and correction from Adversarial Attacks

bla

6.2 Comparison between architectures

bla

7 Conclusion

bla

8 Bibliography

References

[1] Naveed Akhtar and Ajmal Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. *IEEE Access*, 6(August):14410–14430, 2018.

[2] Léon Bottou. Online Learning and Stochastic Approximations (revised 5/2018). *Online Learning in Neural Networks*, pages 1–35, 2018.

[3] Francois Chollet. *Deep Learning with Python*. Manning Publication Co., 2018.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[5] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, pages 1–13, 2014.

[6] Antonio Gulli. *TensorFlow 1.x deep learning cookbook : over 90 unique recipes to solve artificial-intelligence driven problems with Python.*

[7] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*, volume 8. 2018.

[8] Andrew Norton and Yanjun Qi. Adversarial-Playground: A Visualization Suite for Adversarial Sample Generation. 2017.

[9] Moacir Antonelli Ponti, Leonardo Sampaio Ferraz Ribeiro, Tiago Santana Nazare, Tu Bui, and John Collomosse. Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask. *Proceedings - 2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials SIBGRAPI-T 2017*, 2018-January:17–41, 2017.

[10] Richard S. Sutton and Andrew G. Barto. *Reinforcement Leaning.* 2018.

9 hallooo

10 Appendix

bla