BSc Thesis: Self-minimizing deep convolutional neural network for image processing

Jenei Bendegúz

August 19, 2018

Abstract

Lórum ipse: a jorcsó hat a zatékony kötvény fogta, cserzel, esztek. Művészileg is pityókony vitos tegeszkétet, műven „padt teendőt". Rázsási, hogy ami tekély, ahhoz csak óvatosan nyalkodik cipkelnie. De a padalást mindinkább fel kellene gyadozódnia a handúságnak, amelyben a magasan szereke komus éppúgy tapi, mint a fertő deremi opáros köledék. Tehát minél több eres, bolással pélva alanság kell ontoroznia. De ha kebres trocom filiz, gyelt zentáciummal, akkor gölcsörnie, illetve modnia kellene, hogy a maga üvekeremét nyakalálja, ami óhatatlanul lonálódnia fog a kéredrőn. - Egyelőre a selyin kívül nincs ezes tária - irálta okság tikadmás. Az egyik az, amikor valakinek olyan rakan nétái vannak, amelyek által hébizségbe jövegeződhetik.

Contents

1 Introduction

1.1 Problem definition

The goal is to make a framework which is capable of training a deep convolutional neural network for simple two-dimensional image-processing tasks, and make this network as small as possible without sacrificing it's accuracy.

A deep neural network's size is defined by two parameters: number of layers, and number of neurons in each layer, this can be unique amongst layers. These parameters must be specified when building a neural network for training.

This thesis will try to provide automatic strategies to come up with optimal values for these two parameters, as opposed to defining them with trial & error, repeated manual testing, or based on experience.

2 The goal of the network

2.1 Edge detection

2.1.1 Edges

An edge in image processing is a sudden change in pixel intensity through an image. This is common on the edges of objects, or at the intersection of different colors in a pattern. There is no criteria for the actual required value of the change intensity-change per pixel, so there is no one right solution for edge detecting an image. There are multiple techniques available for detecting edges, each having it's own strengths and weaknesses.

There are two classes of edge-detection: continuous and discrete.

Continuous detectors produce an image of the same dimensions as the source image, where a pixels intensity value corresponds to how strong of an edge is present at that position in the original image.

Discrete detection uses boolean values to tell if an edge is present or not. Continuous detections can be converted to discrete with thresholding, or using adaptive thresholding on parts of a continuous edge-detected image.

2.1.2 Detecting edges

Edges can occur in any direction on an image. The detectors often use derivatives and gradients to determine the sudden drops and rises in intensity, among both axes. Since most detectors use multiple steps until they produce the final edge map, it makes sense to use a deep, multi layered network, here a layer can more-or less represent a step in the process.

Convolving an image with an appropriate mask is also used in edge-detection. A convolutional network is capable of learning and applying combinations of masks automatically.

2.2 Conventional edge detectors

2.2.1 Sobel

The Sobel–Feldman operator, or Sobel filter consists of two discrete matrices to be convolved with the source image. This will result in two edge map images, or gradients, one for the horizontal and one for the vertical axis. Then the two matrices can be combined by calculating the geometric mean pixel-wise. The result is a continuous edge map.

2.2.2 Canny

The Canny edge detector is a more complex one, consisting of five steps, including filtering, gradient computing, which can be learned by the network using appropriate masks, and including non-maximal edge suppression, double thresholding, and hysteresis to suppress weak edges, which is not done using convolution, so it can't directly be learned by a convolutional network.

This will force the network to find it's way around the problem, come up with alternative methods using convolutional masks in order to optimize the objective function. The Canny edge detector as opposed to the Sobel operator will result in a discrete edge map.

2.2.3 What are they used for here

Te network will be trained with raw images, and images ran trough the Canny and Sobel edge detectors. The objective of the network will be to mimic the outcome of the detectors.

First we will teach with the Sobel detector. This should be the easier task for the network, since everything can be done with convolutions with the correct masks.

Then we will use the Canny detector, which is considered a harder task due to the multiple steps including different algorithms.

So the input will be a raw image, encoded and compressed in png format, this is what the network will get to work on, and the sample output will be an edge map, made with either the Sobel operator or the Canny operator. The objective function will ensure that the actual output is as close to the sample output as possible, by producing a high penalty if they differ.

3 Implementation

3.1 Tools

3.1.1 Python

Python is an open source interpreted programming language, which is widely used in machine learning, or for writing simple tools, thanks to it's very fluent syntax, which is one of the closest to writing in plain english. Most major machine learning frameworks (tensorflow, Caffe, Caffe2, CNTK and so on) support Python, often as the main language.

It's easy and fast to implement and test a new idea, due to it being a high level language, but the performance cost of it is negligible, because the most demanding part of machine learning isn't the network building part, which is the part Python is used for, but the training itself, repeatedly evaluating the network, calculating the error and propagating it backwards, and this part is usually made to be hardware accelerated by most frameworks.

3.1.2 Tensorflow

Tensorflow, being developed by the Google Brain team, is a framework for making and using computational graph, including deep neural networks. It is used both for research and as a backend for PC and mobile applications, or the core on which higher level frameworks build on.

Tensorflow is based on Python, and it has limited support for C++. There are two ways to make graphs with the toolkit: high level and low level.

In the higher level, there are several predefined layers available, these can be parametrized stacked on top of each other to produce a graph. The most common machine learning network layouts are supported by this method.

In the lower level, exact operations (addition, convolution, transposing, etc.) and the flow of the data must be coded. This method is used to develop tools for higher level tensorflow, and for research, because of the freedom of configuration it provides. While being lower level, there are still numerous tools and helper functions available, even image file decoders and preprocessors for example.

Another strength of tensorflow is hardware acceleration. With a supported Nvidia graphics card, graphs can be executed in GPU accelerated mode, which is often faster than CPU mode by a few magnitudes.

3.1.3 CUDA

3.1.4 Hardware used

Initially a relatively low performance notebook was used to perform all the training. The main benefit of using this notebook was it's built-in dedicated GPU, the Nvidia Geforce 840M. It has 384 CUDA cores with compute capability version 5.0, which means it is able to run tensorflow in GPU-mode. Tensorflow demands compute capability to be 3.0 or higher to run in GPU-mode, 3.0 and 3.5 are common among machine learning toolkits. The notebook had 4GB RAM, and 2GB VRAM.

Later, severe test were run remotely at a higher performance desktop computer, while the notebook was used to develop the models and check if they

work as intended. These runs were often longer than a week, depending on the sample size.

3.2 Preparing the input images

The Visual Object Classes Challenge 2012 (VOC2012) dataset was used as input images. The VOC dataset contains colored pictures of 20 classes in different settings, classes include aeroplane, people, bicycle, horse and so on.

These are fine for simple image processing tasks, there are a wide variety of objects in various distances and quantities. The classes are not needed for us, since we are outputting an image the same dimensions as the input, instead of performing classification. Ditching the classes, the expected output for teaching had to be generate from the input images, this was done with matlab scripts and built-in image processing functions.

The images and edge-maps are fed to the network during training. The images must be preprocessed to be in an efficient format for training. The goal is to have matrices whose can run trough the neurons, and produce an output. First the center of the images are cropped, because the input pipeline uses a fixed structure, and the dataset has pictures of varying size. Then the images are decoded to pure matrices containing floating point values. After this we'll have five dimensional images due to the RGB color palette, this is reduced to two dimensions by converting the image matrices to grayscale. This way the color information is lost, but color edge detection is a whole different area. Finally, a number of images are batched together to enable bulk processing of inputs, leaving us with BATCH SIZE x WIDTH x HEIGHT dimensions for a single input object.

3.3 Network structure

3.3.1 Image preprocessing

3.3.2 The layers

3.3.3 The objective function and output

3.4 Code snippets

3.5 Methods

4 Results

4.1 Metrics

4.2 Tests

4.3 Strategies