Assignment II
Neural Networks and Deep Learning

Adam Kosiorek

I. Objective

THE aim of the second assignment was to develop thorough understanding of deep neural networks. Firstly, through literature research and derivation of Backpropagation algorithm for arbitrary activation functions we gained insight into the inner workings of feed-forward neural networks. Secondly, we learned how to use a state-of-the-art open-source neural network package *caffe* [3] by classifying given images.

II. Neural Networks and Deep Learning

Neural networks are nothing more than a composition of series of non-linear functions and matrix-matrix products. Yet, due to clever learning algorithms, they can be trained to approximate any function. Before we can teach a network, however, we need a dataset of inputs and ground truth pairs as well as a cost function. The latter quantifies differences between network's output and the ground truth for a given input and is being minimized by the training procedure on the whole dataset. The term "deep learning" describes neural networks with hidden layers - the layers between input and output layers. It is believed that deep networks learn feature hierarchies, thus alleviating the need for feature engineering.

III. Deriving Backpropagation

In his book, M. Nielsen derived the Backpropgation algorithm for the sigmoid activation function σ [2]. Here, following the notation, I derive it for an arbitrary function f.

Let δ_j^L be the error of j^{th} output neuron, then:

$$\delta_j^L = \frac{\partial C(a^L)}{\partial z_j^L} = \frac{\partial C(a^L)}{\partial a_j^L}\frac{\partial a_j^L}{\partial z_j^L} \tag{1}$$

where:

$$\frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial f(z^L)}{\partial z_j^L} = f'(z_j^L) \tag{2}$$

and thus:

$$\delta^L = \nabla C(a^L) \odot f'(z^L) \tag{3}$$

For the previous layer, denoted l:

$$\begin{aligned} \delta^l = \frac{\partial C(a^L)}{\partial z^l} = \frac{\partial a^L}{\partial z^l}\nabla C(a^L) = \\ = \frac{\partial z^L}{\partial z^l}(f'(z^L) \odot \nabla C(a^L)) = \frac{\partial z^L}{\partial z^l}\delta^L = \\ = diag(f'(z^l))(w^L)^T\delta^L = f'(z^l) \odot ((w^L)^T\delta^L) \end{aligned} \tag{4}$$

Since relations between any two adjacent layers are the same it holds that:

$$\delta^l = f'(z^l) \odot ((w^{l+1})^T\delta^{l+1}) \tag{5}$$

The final equations for gradients w.r.t. weights and biases do not depend on the activation function and have the same form as in [2].

IV. Classification

caffe is easy to use. Written in C++ with CUDA acceleration, it is high performance, yet, due to Python and Matlab bindings and configuration-oriented architecture it is user-friendly. After completing *ImageNet classification* Notebook I have written a custom script for the assignment [1].

A. Configuration

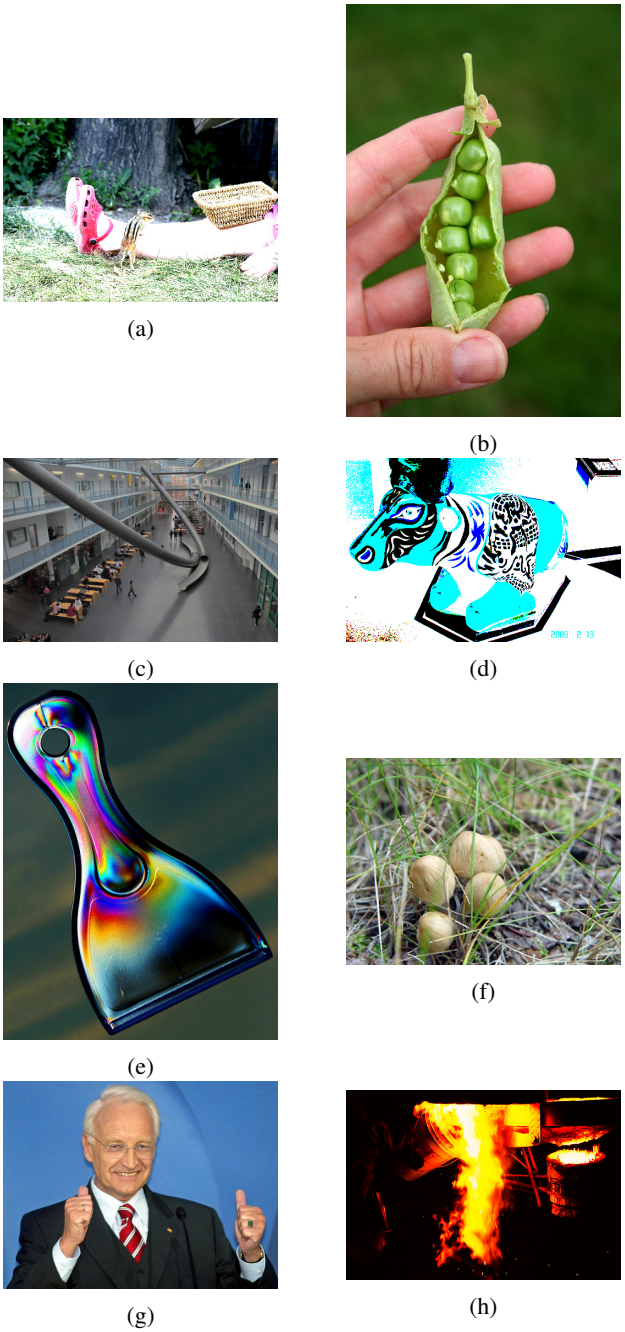
Every neural network in *caffe* needs to be configured before being operational. It needs a deployment configuration file - Google Protocol Buffer file defining the network with free (e.g. non-database) inputs and a trained model. Since we wanted the output class name after classification we also need a synset file with class names and identifiers from ILSVRC2012 [4]

B. Data Preparation

Provided images are 3-channel with various sizes and aspect ratios, while the reference *caffe* model operates on $256 \times 256 \times 3$ data. Convolutional neural networks are scale invariant but are sensitive to changes in aspect ratio. I have therefore resized each input image to 256 px on the longer size and then padded it with zeros to form a square matrix. Padding with zeros should not affect classification, since zeros do not contribute to convolution results. Moreover, the training data for the ImageNet model had intensity values between 0 and 255 and was preprocessed by subtracting mean value computed on the whole dataset and I preprocessed input images accordingly.

C. Results and Discussion

Figure 1 depicts images given for classification and table I shows results. Only images (f) and (g) were classified correctly, as might be suggested by high classification probability and low entropy. Images (e) and (h) can be seen as similar to their predicted categories with lower classification probability and higher entropy. All other images have very low probability and high entropy, suggesting that the classifier is wrong. It seems that classification probability and entropy can be good measures of prediction credibility. According to expectations, the lower the probability (the wider it is spread), the higher the entropy.



(a) (b) (c) (d) (e) (f) (g) (h)

Fig. 1: Images for Classification.

TABLE I: Classification Results

#	#class	class name	probability	entropy
a	428	barrow, garden cart, lawn cart, ...	0.07	5.22
b	311	grasshopper, hopper	0.14	3.72
c	403	aircraft carrier, carrier, flattop, ...	0.13	4.67
d	620	laptop, laptop computer	0.12	4.63
e	673	mouse, computer mouse	0.41	2.75
f	947	mushroom	0.76	1.17
g	834	suit, suit of clothes	0.62	1.09
h	980	volcano	0.36	3.32

REFERENCES

[1] Assignment Code: https://github.com/akosiorek/CSE/tree/master/MLCV/ex2/

[2] M. A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015 http://neuralnetworksanddeeplearning.com.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama and T. Darrell, "Caffe: Convolutional Architecture for Fast Feature Embedding." arXiv preprint arXiv:1408.5093, 2014.

[4] http://image-net.org/challenges/LSVRC/2012/index