Short Introduction to Deep Learning and its Math – Part I

Branislav Kisačanin
Nvidia Corporation and AwesomeMath Academy
b.kisacanin@ieee.org

Abstract

Since the "Deep Learning Big Bang" started in 2012, we are witnessing an unprecedented penetration of artificial intelligence into most, if not all, spheres of life, including science, engineering, medicine, business, politics, law, and entertainment. It is widely recognized that the main ingredients fueling this explosion of deep learning applications have been: advances in algorithms, availability of data, and developments in GPU computing. In this paper we briefly discuss these factors and slice through the algorithms to see the math at their foundations. We show that the key mathematical disciplines that empower deep learning are linear algebra, calculus, optimization, and probability and aim to explain that gaining a more fundamental understanding of why deep neural networks are so good at learning, will open the doors to even greater discoveries but also greater responsibilities.

Keywords: Artificial intelligence, machine learning, deep learning, neural networks.

1 Introduction

A cursory comparison of how we humans do things in 2020 versus how we did them just ten years ago, in 2010, shows that some of the biggest changes are due to the use of artificial intelligence in many aspects of our lives. To name just a few, in no particular order:

- New antibiotics can now be discovered much faster [1]
- Cars can now drive autonomously [2]
- Computers can now reliably translate between many languages [3]
- Everyone can make photo-realistic paintings [5] (try it here)

All of these and many other advances are based on the uncanny ability of deep neural networks to learn from data and allow us to solve problems that were considered too hard for earlier methods just a decade ago. The success of deep learning, currently the hottest area of machine learning and artificial intelligence, is so amazing, unexpected, and hard to understand, that it got its own "unreasonable effectiveness" paper [7], paralleling Wigner's influential paper on unreasonable effectiveness of mathematics in science [8].

Let us introduce the terminology we will use in this paper:

- *Artificial intelligence (AI)* is a broad area of computer science studying design and use of devices that can perceive their environments and utilize that information to maximize their chances of accomplishing their goals.

- *Machine learning (ML)* is a subset of artificial intelligence and is a study of algorithms for learning from data with the goal of being able to make decisions (predictions) without being explicitly programmed for the task.

- *Supervised learning* is a type of machine learning that involves a set of labeled training data (consisting of inputs to the learning function and corresponding labels) and the learning function (for example a neural network) which is being modified in order to minimize the difference between the desired and actual outputs of the learning function. At the same time, the goal is to train the learning function so it generalizes well, i.e., works well with test data – inputs it did not encounter during training. Two major classes of supervised learning are *regression*, in which the desired outputs are from a continuous set, and *classification* in which the desired outputs are from a discrete set. There are other types of learning (e.g., unsupervised, reinforcement), but here we limit the discussion to supervised learning.

- *Artificial neural networks (NNs)* are mathematical functions inspired by biological learning. Their building blocks are artificial neurons, which take a number of inputs and produce a single output, computed as a nonlinear function of a weighted sum of the neuron's inputs,

  $$f_k(x_{k1}, \ldots, x_{kn}) = \eta_k(b_k + w_{k1}x_{k1} + \ldots + w_{kn}x_{kn}),$$

  where $x_{k1}, \ldots, x_{kn}$ are the inputs to the artificial neuron labeled by integer k, $\eta_k(z)$ is a monotonic nonlinear function, while $w_{k1}, \ldots, w_{kn}$ and b_k are this artificial neuron's adjustable (learnable) parameters. This is called the McCulloch-Pitts model of a neuron [9]. Neural networks[1] differ by how many neurons they have, how their neurons are connected, what nonlinearities they use, how many layers they form, etc. Neural networks learn through iterative modifications of neuron parameters w and b.

- *Neural network layer* is a loosely defined subset of a neural network, a set of neurons which operate on hierarchically similar data. In the *input* layer, all neurons get their inputs from the outside world, perhaps as image pixels, audio signal samples, etc. Neurons in *hidden* layers and in the *output* layer get their inputs from the outputs of other neurons in the network. Neurons in the *output* layer project their outputs outside of the network.

- *Deep neural network (DNN) = deep learning network (DLN)* is a neural network with multiple hidden layers, i.e., with multiple layers between the input and the output layers. LeNet-5, an early example of a DLN, has five hidden layers [10]. One of the networks that marked the beginning of the "Deep Learning Big Bang," AlexNet, has seven hidden layers [11]. More recent DLNs boast many more hidden layers, for example, ResNet has 150 hidden layers [12].

- *Deep Learning* is a part of machine learning studying training of deep learning networks. As a process, deep learning commonly involves datasets with millions of labeled training images or other data and trillions of mathematical operations to adjust billions of parameters in a deep learning network. In this paper we will focus our attention on applications of deep learning in a supervised learning framework, but deep learning has applications in other types of learning (unsupervised, reinforcement, ...)

In the rest of this paper we will briefly describe the three developments that made the "Deep Learning Big Bang" possible (Section 2) and then look at the mathematical foundations of the algorithms used in deep learning networks (Section 3).

[1]From this point on we will follow the common terminology and will stop saying *artificial* neurons and *artificial* neural networks.

2 The Three Components of the Deep Learning Big Bang

It is generally acknowledged that in year 2012, three things converged that made the immense success of deep learning possible, thus starting the “Deep Learning Big Bang:”

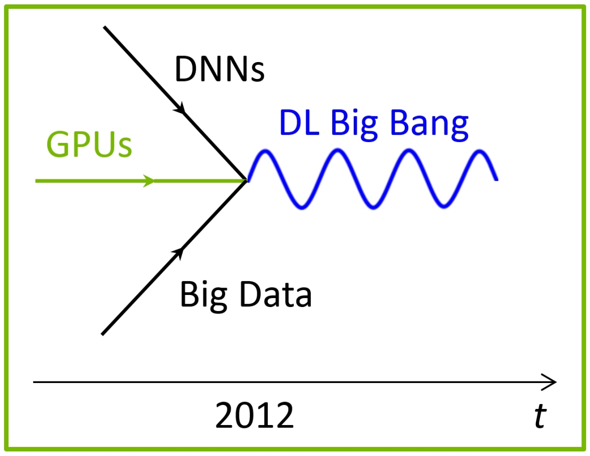

Figure 1: Illustration of the “Deep Learning Big Bang” as a Feynman diagram.

- *Data.* Over the years, labelled datasets used in machine learning have been steadily growing in size. A few years before 2012 they reached the sizes that could be used to demonstrate the learning power of deep learning networks. One dataset in particular had a huge impact on deep learning work: the 2009 ImageNet dataset with 14 million images hand-labelled into more than 20,000 categories (including 120 breeds of dogs). For comparison, the most utilized image dataset prior to 2009 was the 2005 PASCAL dataset, with 20,000 images representing 20 object classes. Since then many other datasets have been compiled and open-sourced by both academia and industry, further accelerating algorithm development for various applications: natural language processing, autonomous driving, medical diagnostics, etc. Availability of these datasets allows researchers interested in algorithm development to focus on algorithm development, because now they do not have to compile and label their own datasets, a very time-consuming and tedious task.

- *Digital computers.* The amazing growth of computational power of digital[2] computers has followed Moore’s Law since the late 1960’s, doubling about every two years. As we reach the limits of this growth due to the finite size of atoms, computational power will still continue to grow for highly parallel computers and algorithms that can use them. This is very favorable for deep learning networks, because unlike some other machine learning approaches, computations in DLNs are highly parallelizable, both during training and deployment. In fact, parallelizability of DLNs was used since the late 2000’s, most notably by AlexNet in 2012, when it was reported that the use of the GPU (graphics processing unit) processor for training and deployment had a 100-fold speed advantage over regular computer processor, the CPU (central processing unit). Since then, multiple new generations of GPU’s have been released with architectural improvements for computer graphics and gaming, their original application, as well as for deep learning.

[2]If the reader is wondering if it is necessary to say *digital* for a computer, the answer is yes, because there are *analog* computers too, and they have also been tried in the context of neural networks. As of 2020, digital computers are still the best for neural networks, but if *quantum* computers realize their promise, we will likely use this new computing paradigm.

- *Algorithms.* As new types of artificial neural networks have been introduced since the 1950's, understanding of the impact of various types of layers on different applications has increased too. For example, in 1989 LeCun used convolutional layers in networks for reading handwritten characters, culminating in his 1998 LeNet-5 network, which was deployed in mail sorting by the US Postal Service. LeNet-5 is also significant as the first practical network to use *backpropagation* to learn the network parameters. Fast forward to 2012, when AlexNet, also a convolutional network, has won the ImageNet Challenge, a competition in image classification, by a large margin.

It would be of interest to present here the math behind the data and information aspects of Deep Learning, Shannon's Sampling Theorem in particular, as the critical link between the analog world around us and the discrete, digital representation of it used by computers. It would also be of interest to see the math used in processor design and manufacture, for example the design of photo masks which have to take into account significant light diffraction effects in order to accurately produce transistors and other components that are now two orders of magnitude smaller than the wavelength of light. However, in the rest of this paper we focus only on the math used to train and deploy deep learning networks, as there is still much in there to challenge even the best mathematicians to help us understand how DLNs learn as well as they do.

3 Math of Deep Learning

In this section we present the mathematics of *learning* in deep learning. We start with simpler tasks, steadily building towards deep learning networks.

3.1 Univariate Linear Regression

If we are given a set of data points, for example floor areas of homes recently sold in our neighborhood, and their prices, we may decide to use this data to help our neighbors predict the price they can get for their home. Assuming that the price of a home depends only on its floor area (which is questionable, because there are other factors influencing the price, such as how old is the roof, presence of water damage, size of the yard, to name a few), and assuming that this dependence is linear (which is also questionable and should be checked at least by "eyeballing" the data), this problem fits the description of a univariate linear regression formulated and solved as follows:

Problem statement: Given a set of m points $\left(x^{(i)}, y^{(i)}\right)$, $i = 1, \ldots, m$, learn the linear model (hypothesis function) $y = ax + b$ such that it is a "good" fit for the data.

Note: We usually define the goodness of fit using a cost function (also called the loss function, objective function, or the error function), which is to be minimized to achieve the best fit in that sense. There are many good choices for the cost function, but here we pick the one that is particularly easy to work with analytically, the L^2 or Euclidean norm of differences between model predictions $ax^{(i)} + b$ and true values $y^{(i)}$:

$$J(a,b) = \sum_{i=1}^{m} \left(ax^{(i)} + b - y^{(i)}\right)^2 .$$

With this choice of the cost function, the optimal parameters a and b are called the *least-squares solution.* A different choice of the cost function will likely result in a slightly different solution[3].

[3]The simplest illustration of this fact is with picking a statistic to represent the center of a set of numbers. If we pick it to minimize the L^2 distance from the data, we obtain the mean, while if we choose to minimize the L^1 distance (sum of absolute differences) from the data, we get the median. As we know, the mean and the median of the data are not necessarily equal.

Solution: We are looking for

$$(a,b)=\arg\min_{(a,b)} J(a,b)$$

and since in our case $J(a,b)$ is a convex function of parameters a and b, there exists a global minimum. We will first find it analytically and then we will formulate an iterative procedure to find it.

Analytic Solution: In order to find parameters (a,b) that minimize the cost function $J(a,b)$, we write

$$\frac{\partial J}{\partial a}=0 \quad \text{and} \quad \frac{\partial J}{\partial b}=0.$$

This results in the following system of two equations in two unknowns, a and b

$$\begin{aligned}\sum_{k=1}^{m} x^{(i)}(ax^{(i)}+b-y^{(i)}) &= 0\\ \sum_{k=1}^{m}(ax^{(i)}+b-y^{(i)}) &= 0\end{aligned}$$

We can rewrite these as

$$\begin{aligned}S_{xx}a+S_xb &= S_{xy}\\ S_xa+mb &= S_y\end{aligned}$$

where

$$S_x=\sum_{k=1}^{m} x^{(i)}, \qquad S_{xx}=\sum_{k=1}^{m}\left(x^{(i)}\right)^2,$$

$$S_y=\sum_{k=1}^{m} y^{(i)}, \qquad S_{xy}=\sum_{k=1}^{m} x^{(i)}y^{(i)}.$$

Finally, the least-squares solution is given by

$$a=\frac{mS_{xy}-S_xS_y}{mS_{xx}-S_xS_x} \qquad \text{and} \qquad b=\frac{S_{xx}S_y-S_{xy}S_x}{mS_{xx}-S_xS_x} \tag{1}$$

To show that his choice of a and b really minimizes $J(a,b)$, we need to show that the Hessian matrix of $J(a,b)$ is positive definite at that point. The Hessian is

$$H=\begin{bmatrix}\dfrac{\partial^2 J}{\partial a^2} & \dfrac{\partial^2 J}{\partial a\partial b}\\ \dfrac{\partial^2 J}{\partial b\partial a} & \dfrac{\partial^2 J}{\partial b^2}\end{bmatrix} = 2\begin{bmatrix} S_{xx} & S_x\\ S_x & m\end{bmatrix}$$

Sylvester's criterion tells us that in general H is non-negative definite because

1. $S_{xx}=\sum\left(x^{(i)}\right)^2\ge 0$ and
2. $mS_{xx}-S_x^2\ge 0$ (due to Cauchy-Schwartz inequality).

We also see that unless $x^{(1)}=\ldots=x^{(m)}$, H is indeed positive-definite and so parameters (a,b) given by Equation (1) minimize $J(a,b)$.

Notes: The special case when $x^{(1)} = \ldots = x^{(m)}$, which would result in a vertical line, cannot be handled directly by this approach because the assumed model, $y = ax + b$, does not include vertical lines. Besides, due to the way we defined the cost function J, as the sum of squared *vertical* distances between data points and the fitted line, the results of such line-fitting are not always intuitive, especially for steep lines. A different type of least-squares cost function handles these situations better, by using the *perpendicular* distances between the points and the fitted line. This approach is called the *total least-squares* and relies on the singular value decomposition of the data covariance matrix.

Iterative Solution: We look at the iterative solution of the univariate linear regression because it will help us develop the intuition for the later, more complex problems, which do not have analytical solutions. The iterative solution is based on the idea of *gradient descent.* Imagine hiking in foggy conditions in the mountains and trying to find the bottom of the valley with limited visibility around you. The best you can do is to look around you, determine the local gradient of the terrain (i.e., the direction in which the terrain goes up fastest), and make a step in the opposite direction. With $k \geq 0$ denoting the iteration index, we can write this as follows:

$$\begin{aligned} a_{k+1} &= a_k - s\,\frac{\partial}{\partial a} J(a_k, b_k) \\ b_{k+1} &= b_k - s\,\frac{\partial}{\partial b} J(a_k, b_k), \end{aligned}$$

where $s > 0$ is the learning step size, probably the most important learning hyper-parameter. When dealing with a new problem, we need to empirically determine it so that the learning converges (i.e., s should not be too big), yet it doesn't converge too slowly (i.e., s should not be too small).

Another important learning hyper-parameter is the initial point (a_0, b_0), which we pick randomly in the absence of prior knowledge about optimal values.

When $J(a, b)$ is convex, this iterative procedure is guaranteed to converge to the optimal values of a and b.

3.2 Multivariate Regression

On our way towards understanding of how deep learning networks are trained, we next need to generalize what we did so far to inputs represented by arrays of n numbers and to models (hypotheses functions) that are possibly nonlinear.

To motivate this formulation, think again of predicting the cost of a home, but this time we have more information about the recently sold homes in its neighborhood. Let

- m be the number of data points in the training set
- n be the number of features, values that describe each object (home, for example)
- $x^{(i)}$ be the column vector of features describing the i^{th} training example
- $x_j^{(i)}$ be the j^{th} feature of the i^{th} training example

To simplify the notation, we will homogenize our feature vectors by appending ones as the 0-index values, so

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1},$$

where $i = 1, \ldots, m$ and $x_0^{(i)} \equiv 1$. We will find it useful to arrange these feature vectors as rows of an $m \times (n+1)$ matrix X (the so called design matrix):

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \vdots \\ \left(x^{(m)}\right)^T \end{bmatrix}_{m \times (n+1)}$$

Also, let

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

be the outputs of our training data. Let us use a linear hypothesis function

$$h_\theta(X) = X\theta,$$

where

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

are the parameters we can optimize in order to minimize the cost function, for example the L^2 error

$$J(\theta) = \|h_\theta(X) - y\|^2.$$

Analytic Solution: With the quadratic cost function and a linear hypothesis function, we want to pick parameters θ so that the cost function $J(\theta) = \|X\theta - y\|^2$ is minimized. Since

$$\|X\theta - y\|^2 = (X\theta - y)^T(X\theta - y) = \theta^T X^T X\theta - 2y^T X\theta + y^T y$$

and derivatives of quadratic and linear forms are given by

$$\frac{\partial(u^T Q u)}{\partial u} = 2Qu \quad \text{and} \quad \frac{\partial(Qu)}{\partial u} = Q^T,$$

we can write

$$0 = \frac{\partial J(\theta)}{\partial \theta} = 2X^T X\theta - 2(y^T X)^T = 2X^T(X\theta - y).$$

This equation is called the *normal equation* (because it indicates that the vector $X\theta - y$ is normal (perpendicular) to the range of matrix X). It gives us the following analytic solution to our problem

$$\theta = (X^T X)^{-1} X^T y.$$

Note that $X^+ = (X^T X)^{-1} X^T$ is known as the left Moore-Penrose pseudoinverse and generalizes the matrix inverse. Also, it is not difficult to see that this formula generalizes the analytic solution of Equation (1) we derived for the univariate linear regression. In this case the Hessian matrix of $J(\theta)$ is

$$H = \frac{\partial^2 J(\theta)}{\partial \theta^2} = 2(X^T X)^T = 2X^T X \geq 0.$$

Note that $X^T X$ is the Gramian matrix of X and it is positive definite if and only if rows of X, which are our feature vectors, are linearly independent. Indeed, that is a desirable property of feature vectors, otherwise we have degenerate training data.

Iterative Solution: With

$$\operatorname{grad} J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) \end{bmatrix}$$

we can write the iterative solution as follows,

$$\theta(k+1) = \theta(k) - s \operatorname{grad} J(\theta).$$

Note that k is the iteration index, s and $J(\theta)$ are still scalars, while $\theta(k+1)$, $\theta(k)$, and $\operatorname{grad} J(\theta)$ are $(n+1) \times 1$ column vectors. When $h_\theta(X) = X\theta$ and $J(\theta) = \|h_\theta(X) - y\|^2$, we can write

$$\operatorname{grad} J(\theta) = 2X^T(X\theta - y).$$

3.3 Logistic Regression: Regression or Classification?

Do not let the unfortunate choice of words confuse you, in machine learning we use logistic regression for classification. Recall that in regression, the model outputs are continuous values. On the other hand, in classification we want the model output to have discrete values. For example, most of us rely on our e-mail apps to do a binary classification of e-mails as "spam" or "not spam." We might be tempted to do classification by learning a linear regression and then applying a threshold to its results. The issue with that is how outliers in training data influence the learned model.

Here we present an approach based on learning a logistic regression first, and thresholding it to obtain the classification. To further answer our subsection title question, logistic regression without thresholding is indeed a regression, but with thresholding it becomes a classifier.

We start from the logistic function (also called the sigmoid function)

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

which maps real numbers to interval $(0,1)$. We define our hypothesis function[4] as

$$h_\theta(x) = \sigma(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}.$$

We will make a classification decision $y = 1$ when $h_\theta(x) \geq 0.5$ and $y = 0$ otherwise. The decision boundary is $\theta^T x = 0$.

The cost function commonly used in this context is

$$J(\theta) = \sum_{i=1}^{m} H(y^{(i)}, h_\theta(x^{(i)})),$$

where $H(p,q)$ is the cross-entropy[5] of distribution q relative to distribution p and is given by

$$H(p,q) = -\sum_i p_i \log q_i$$

In our case $y^{(i)} \in \{0,1\}$ so $p_1 = 1$ and $p_2 = 0$, therefore

$$H(y^{(i)}, h_\theta(x^{(i)})) = -y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})).$$

[4]This definition of $h_\theta(x)$ is a special case of the McCulloch-Pits model of a neuron with the nonlinearity chosen to be the sigmoid function.

[5]The cross-entropy is related to the Kullback–Leibler divergence by an additive constant. Both are used to measure dissimilarity of two probability distributions.

Just like the choice of a quadratic cost function in linear regression, the choice of this cost function is largely driven by the fact that it works well in practice and is easy to handle analytically, as demonstrated by the fact that

$$\operatorname{grad} J(\theta) = \frac{\partial J(\theta)}{\partial \theta} = \begin{bmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) \end{bmatrix} = X^T (h_\theta(X) - y),$$

so gradient descent for logistic regression can be written as

$$\theta(k+1) = \theta(k) - s \operatorname{grad} J(\theta).$$

Notes: Before writing your own program to do gradient descent, consider using well tested library functions that do the same, and in fact offer many other minimization algorithms, such as `fminunc` in MATLAB and OCTAVE or `scipy.optimize` in PYTHON.

Logistic regression gives us a binary classifier. To build a classifier with L classes, we can use L binary (one-vs-rest) classifiers. To pick the final answer, we take the class whose one-vs-rest classifier produced the highest score.

4 Conclusions

In Part I of this paper we discussed the constituent parts of the “Deep Learning Big Bang” and took a look at the math behind basic regression and classification learning approaches. They are a great place to look at basic blocks of much more complex deep neural networks.

In Part II we will discuss issues encountered during training, even with simple regression and classifier models presented in this part, in particular overfitting. We will also cover regularization of the cost function, which helps us alleviate overfitting. After that we will finally get to talk about deep neural networks, computational graphs used to represent them and compute with them during training and deployment. In that context we will also talk about the key part of the DLN training process, the backpropagation. Finally we will provide pointers for further reading and learning and introduce some of the biggest open problems facing deep learning: conditions for regularization to work in general, why deep neural networks learn as well as they do, how can we explain their “logic” to users, and what may be ethical issues facing the use of deep learning networks.

References

[1] J. M. Stokes et al., A deep learning approach to antibiotic discovery, *Cell.* Volume 180, Issue 4, pp. 688-702, 2020.

[2] M. Daily et al., Editors, Self-driving cars, Special issue of *Computer*, Volume 50, Issue 12, 2017.

[3] E. de Vries, et al., No Longer Lost in Translation: Evidence that Google Translate Works for Comparative Bag-of-Words Text Applications, *Political Analysis*, Volume 26, Issue 4, pp. 417-430, 2018.

[4] J. Sommerlad, Google Translate, *Independent*, 19 June 2018. (link)

[5] T. Park et al., Semantic image synthesis with spatially-adaptive normalization, *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* 2019. (link)

[6] M. Burns, Nvidia AI turns sketches into photo-realistic landscapes in seconds, *The Crunch*, 18 March 2019. (link)

[7] T. J. Sejnowski, The unreasonable effectiveness of deep learning in artificial intelligence, *Proc. of the National Academy of Sciences*, 2020.

[8] E. P. Wigner, The unreasonable effectiveness of mathematics in the natural sciences, *Comm. on Pure and Applied Mathematics*, Volume 13, pp. 1-14, 1960.

[9] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 1943.

[10] Y. Lecun et al., Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 1998.

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet classification with deep convolutional neural networks, *NIPS*, 2012.

[12] K. He et al., Deep Residual Learning for Image Recognition, *CVPR*, 2016.