Pós-Graduação em Ciência da Computação

David Lopes de Macêdo

ENHANCING DEEP LEARNING PERFORMANCE USING DISPLACED RECTIFIER LINEAR UNIT

RECIFE
2017

David Lopes de Macêdo

ENHANCING DEEP LEARNING PERFORMANCE USING DISPLACED RECTIFIER LINEAR UNIT

A M.Sc. Dissertation presented to the Informatics Center of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Advisor: *Teresa Bernarda Ludermir*
Co-Advisor: *Cleber Zanchettin*

RECIFE
2017

Catalogação na fonte
Bibliotecária Elaine Cristina de Freitas CRB4-1790

M141e Macêdo, David Lopes de

Enhancing deep learning performance using displaced rectifier linear unit / David Lopes de Macêdo. – 2017.

90f.: fig., tab.

Orientadora: Teresa Bernarda Ludermir

Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, Recife, 2017.

Inclui referências e Apêndice.

1. Inteligência artificial 2. Redes Neurais Convolucionais 3. Funções de Ativação . I. Ludermir,Teresa Bernarda (Orientadora) II. Titulo.

006.3 CDD (22. ed.) UFPE-MEI 2017-280

Dissertação de mestrado apresentada por **David Lopes de Macêdo** ao programa de Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **Enhancing Deep Learning Performance using Displaced Rectifier Linear Unit**, orientada pela **Prof.ª Teresa Bernarda Ludermir**, coorientada pelo **Prof. Cleber Zanchettin**.

Aprovado em: 31/07/2017

____

Prof.ª Dr.ª Teresa Bernarda Ludermir
Orientadora, Centro de Informática/UFPE

RECIFE
2017

To my family.

Acknowledgements

This work would not have been possible without the support of many. I would like to thank and dedicate this dissertation to the following people:

To my advisor Teresa Ludermir. Teresa is an exceptional researcher and professor. Her guidance and support were fundamental to motivate me throughout this research.

To my co-advisor Cleber Zanchettin for his contributions to the work we have done.

To my family, especially my parents, José and Mary, my wife, Janaina, and my children, Jéssica and Daniel, for give me the love that I need through my whole life.

Things should be made as simple as possible, but no simpler.

—ALBERT EINSTEIN

Resumo

Recentemente, a aprendizagem profunda tem causado um impacto significante em visão computacional, reconhecimento de voz e compreensão de linguagem natural. Apesar de avanços significativos, recentemente os ganhos em desempenho em aprendizagem profunda tem sido modestos e usualmente dependem do incremento da profundidade dos modelos, o que normalmente requer mais recursos computacionais como tempo de processamento e uso de memória. Para abordar este problema, nós voltamos nossa atenção para o interfuncionamento entre as funções de ativações e a normalização em batch, o qual é praticamente obrigatório atualmente. Neste trabalho, nós propomos a função de ativação Displaced Rectifier Linear Unit (DReLU) a partir da conjectura que estender a função identidade da ReLU para o terceiro quadrante aprimora a compatibilidade com a normalização em batch. Ademais, nós usamos testes estatísticos para comparar o impacto de usar funções de ativação distintas (ReLU, LReLU, PReLU, ELU, and DReLU) na performance da velocidade de treinamento e na acurácia dos testes de modelos estado da arte VGG e Redes Residuais. Estas redes neurais convolucionais foram treinadas no CIFAR-10 e CIFAR-100, as base de dados mais comumente utilizadas em visão computacional para aprendizagem profunda. Os resultados mostraram que DReLU aumentou a velocidade de aprendizagem em todos os modelos e bases de dados. Ademais, avaliações de performance com o uso de testes estatíticos ($p < 0.05$) mostraram que DReLU melhorou a acurácia dos testes apresentados pela ReLU em todos os cenários. Além disso, DReLU apresentou melhor acurácia de testes que qualquer outra função de ativação testada em todos os cenários com uma exceção, no qual esta apresentou a segunda melhor performance. Desta forma, este trabalho mostra que é possível aumentar a performance substituindo a ReLU por uma função de ativação aprimorada.

Palavras-chave: Funções de Ativação. Normalização em Batch. ReLU. DReLU. Aprendizagem Profunda. Redes Neurais Convolucionais.

Abstract

Recently, deep learning has caused a significant impact on computer vision, speech recognition, and natural language understanding. In spite of the remarkable advances, deep learning recent performance gains have been modest and usually rely on increasing the depth of the models, which often requires more computational resources such as processing time and memory usage. To tackle this problem, we turned our attention to the interworking between the activation functions and the batch normalization, which is virtually mandatory currently. In this work, we propose the activation function Displaced Rectifier Linear Unit (DReLU) by conjecturing that extending the identity function of ReLU to the third quadrant enhances compatibility with batch normalization. Moreover, we used statistical tests to compare the impact of using distinct activation functions (ReLU, LReLU, PReLU, ELU, and DReLU) on the learning speed and test accuracy performance of VGG and Residual Networks state-of-the-art models. These convolutional neural networks were trained on CIFAR-10 and CIFAR-100, the most commonly used deep learning computer vision datasets. The results showed DReLU speeded up learning in all models and datasets. Besides, statistical significant performance assessments ($p < 0.05$) showed DReLU enhanced the test accuracy obtained by ReLU in all scenarios. Furthermore, DReLU showed better test accuracy than any other tested activation function in all experiments with one exception, in which case it presented the second best performance. Therefore, this work shows that it is possible to increase the performance replacing ReLU by an enhanced activation function.

Keywords: Activation Functions. Batch Normalization. ReLU. DReLU. Deep Learning. Convolutional Neural Networks.

List of Figures

List of Tables

List of Acronyms

ReLU	Rectifier Linear Unit	18
LReLU	Leaky Rectifier Linear Unit	22
PReLU	Parametric Rectifier Linear Unit	22
ELU	Exponential Linear Unit	22
DReLU	Displaced Rectifier Linear Unit	61
CNN	Convolutional Neural Network	24
VGG	Visual Geometry Group	25
ResNet	Residual Network	25
RBM	Restricted Boltzmann Machine	26
AE	Auto-Encoder	26
DBN	Deep Belief Network	26
SAE	Stacked Auto-Encoder	26
MLP	Multilayer Perceptron	26
FNN	Feedforward Neural Network	26
SAH	Single Algorithm Hypothesis	26
RNN	Recurrent Neural Network	29
GMM	Gaussian Markov Model	29
DNN	Deep Neural Network	29

Contents

1 INTRODUCTION

A journey of a thousand miles begins with a single step.

—LAO TZU

In this introductory chapter, we explain the context of this work, which is deep learning research. After that, we establish the problem of interest. Then we set the goals of this study and the contributions we achieved. Finally, we present an outline of the subject of the next chapters.

1.1 CONTEXT

The artificial neural networks research passed through three historical wave (Fig. 1.1) (GOODFELLOW; BENGIO; COURVILLE, 2016). The first one, known as cybernetics, started at the 1960s with the work of Rosenblatt and the definition of the Perceptron, which was showed to be useful in linear separable problems (ROSENBLATT, 1958). This initial excitement diminished in the 1970s by the work of Minsk and Papert (MINSKY; PAPERT, 1969), which demonstrated some limitations of this concept.

The second wave of artificial neural networks research, known as connectionism, began in the 1980s after the dissemination of the discovery of the so-called backpropagation algorithm (RUMELHART; HINTON; WILLIAMS, 1986), which allowed training neural networks with few hidden layers. Nevertheless, the Vanish Gradient Problem supported the idea that training neural networks with more than few layers was a hard challenge (HOCHREITER, 1991).

Therefore, this second wave was replaced by a huge interest in new statistical machine learning methods discovered or improved in the 1990s. Artificial neural networks research passed through another dismal period and fell out of favor again. Indeed, it was a time when the machine learning researchers largely forsook neural networks and backpropagation was ignored by the computer vision and natural language processing communities.

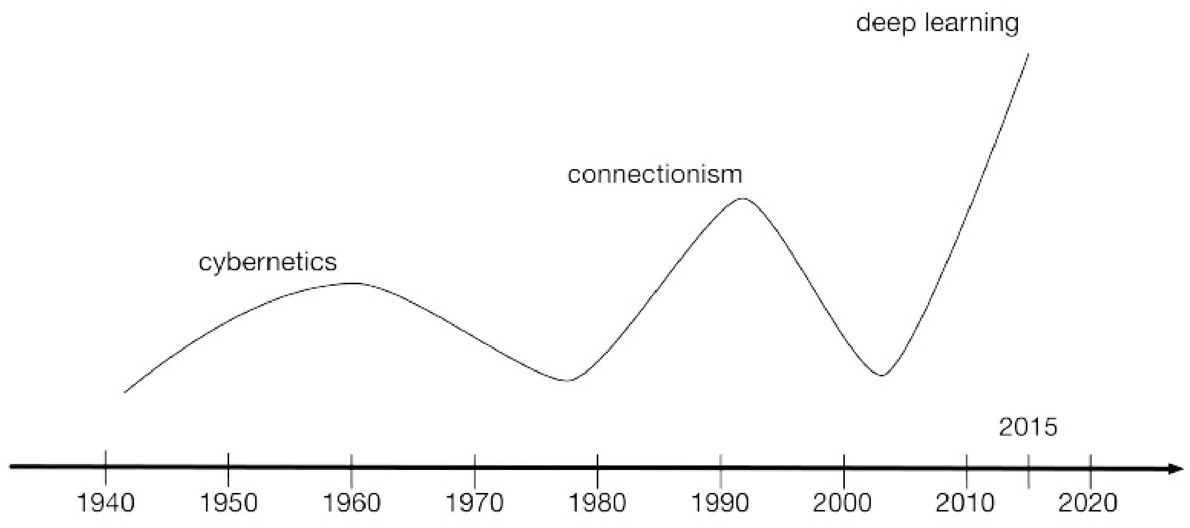


Figure 1.1: The three historical waves of artificial neural networks research (GOODFELLOW; BENGIO; COURVILLE, 2016).

The third and present wave of artificial neural networks research has been called deep learning, and it started at the late 2000s with some seminal works from Geoffrey Hinton, Yoshua Bengio, and Yann LeCun, which showed that it is possible to train artificial neural networks with many hidden layers. The recent advances in deep learning research have produced more accurate image, speech, and language recognition systems and have generated new state-of-the-art machine learning applications in a broad range of areas such as mathematics, physics, healthcare, genomics, financing, business, agriculture, etc.

Activation functions are the components of neural networks architectures responsible for adding nonlinearity capabilities to the models. In fact, considering Figure 1.2, the transformation performed by a generic shallow or deep neural network layer can be written by the equation bellow:

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{1.1}$$

As can be seen in Eq. 1.1, the activation function is the only component of a neural network, or a deep architecture, that incorporates nonlinearity capability. Indeed, if the activation function f is removed from the mentioned equation, a particular layer would be able only to perform affine transformations, which are composed of a linear plus a bias transformation. Since a composition of affine transformation is also an affine transformation, even very deep neural networks would only be able to perform affine transformations.

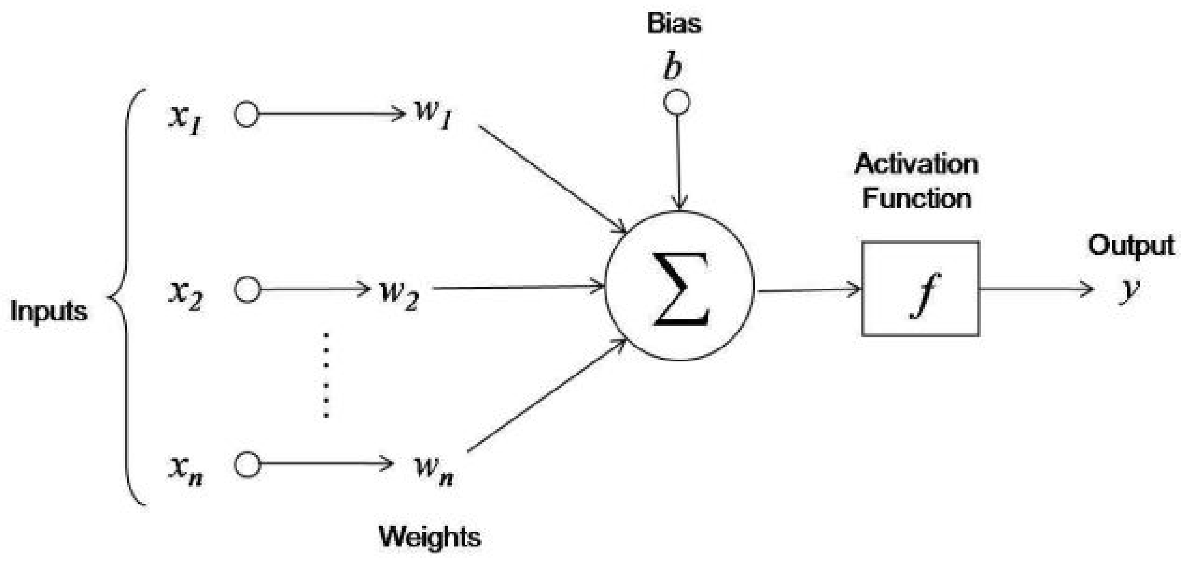


Figure 1.2: Activation functions introduces nonlinearity to shallow or deep neural networks (HAYKIN, 2008).

However, it is widespread knowledge that the vast majority of real world problems are nonlinear and therefore neural networks would be restricted to be useful in extremely few practical situations in case of absence of nonlinear activation functions. In this sense, it is remarkably important that activation functions enable neural networks to be able to perform complex nonlinear maps, mainly when several layers are hierarchically composed.

The demonstration of the fundamental role that activation functions present in neural networks performance does not rely only on theoretical arguments like the above mentioned but are in fact also endorsed by practical considerations.

Indeed, there is not doubt that one of the most significant contributions (or perhaps the major contribution) to make supervised deep learning a major success in practical tasks was the discovery that Rectifier Linear Unit (ReLU) (Fig. 1.3) outperforms traditionally used activation functions (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

The mentioned article showed that ReLU presents much better performance than traditionally used Sigmoid and Hyperbolic Tangent activation functions to train artificial neural networks, mainly when a deep architecture is being used (Fig. 1.4). The fundamental innovation of ReLU was showing that a linear identity to positive values is a clever mechanism to ensure the backpropagation of the gradient to very deep layers, which was believed to be impossible by the research community for decades (HOCHREITER, 1991).

Therefore, considering the theoretical and practical aspects above mentioned, the fundamental importance of activation functions for the design and performance of (deep) neural networks can not be overestimated.

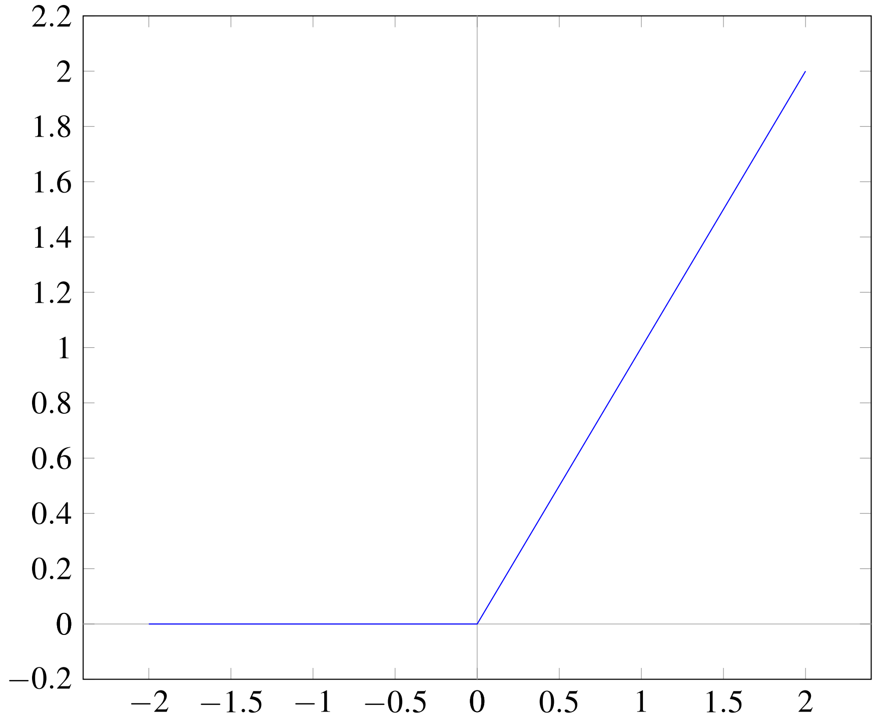


Figure 1.3: Plot of ReLU activation function.

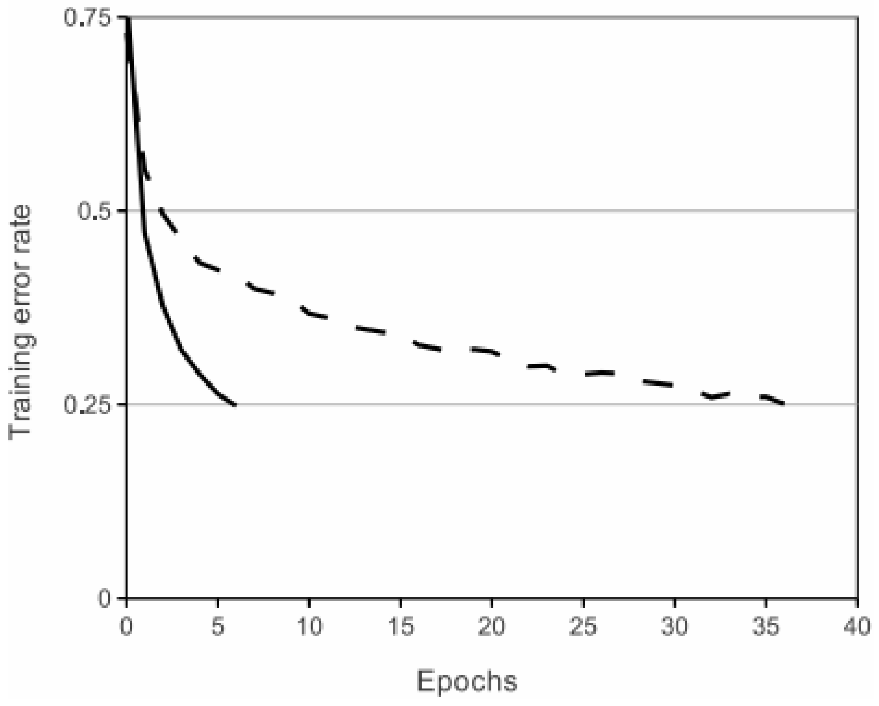


Figure 1.4: The ReLU activation function (solid line) presents much higher training speed and test accuracy performance than traditionally used activation functions like the Sigmoid and Hyperbolic Tangent (dashed line) (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Similar to activation functions, batch normalization (IOFFE; SZEGEDY, 2015) currently plays a fundamental role in training deep architectures. This technique normalizes the inputs of each layer, which is equivalent to normalizing the outputs of the deep model previous layer. Consequently, batch normalization allows the use of higher learning rates and also makes the weights initialization technique used almost irrelevant.

Batch normalization also works as an effective regularizer, virtually eliminating the need for another regularization technique called dropout (SRIVASTAVA et al., 2014). Therefore, batch normalization significantly contributes to improving the deep learning performance and currently represents a mandatory method in this research field.

1.2 PROBLEM

The Sigmoid activation function was one the most used during the past decades. The primary motivation for the use of Sigmoid is the fact that it has a natural inspiration and that the almost linear region near the y-axis can be utilized for training. The saturations are needed to provide nonlinearity to the activation function (Fig. 1.5).

However, the same saturations that are responsible for proving nonlinearity to Sigmoid, in practical terms, kill the gradients and therefore are extremely harmful to the backpropagation and consequently to the learning process. In fact, the extreme small gradients of saturation areas do not allow the update of weights in such cases. For this reason, for years, careful initialization was used to avoid the saturation region (LECUN et al., 2012). Nevertheless, during training, activations inevitably begin to fall in the saturation areas, which slow training.

The Hyperbolic Tangent was also commonly used during last decades in shallow neural networks. This activation function has essentially the same format of Sigmoid, but it presents zero mean output, which improves learning by working as a type of normalization procedure (LECUN et al., 2012) (Fig. 1.6).

Nevertheless, Hyperbolic Tangent still have saturation regions, and therefore all the above comments about the drawbacks of this aspect of the Sigmoid also applies. Therefore, despite representing an advance in comparison to Sigmoid, the Hyperbolic Tangent still presents severe limitations.

Currently, we know that neither Sigmoid nor Hyperbolic Tangent were able to train deep neural networks because of the absence of the identity function for positive input (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). The saturations presented by these activation functions produce zero slope and therefore the backpropagation procedure was unable to send gradient information to deeper layers. This phenomenon was studied in 1990's and become known as the Vanishing Gradient Problem (HOCHREITER, 1991).

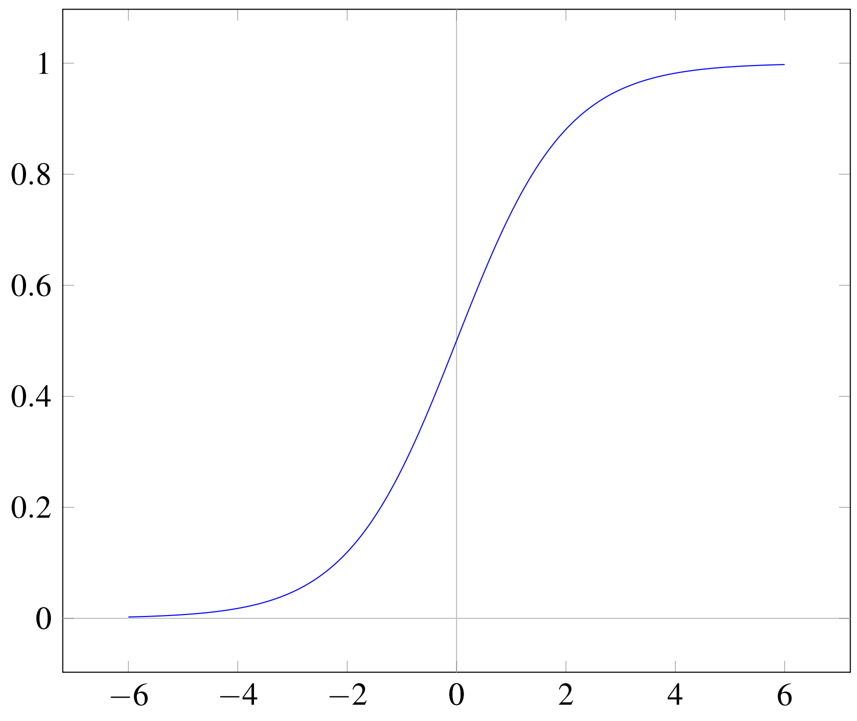


Figure 1.5: Plot of Sigmoid activation function.

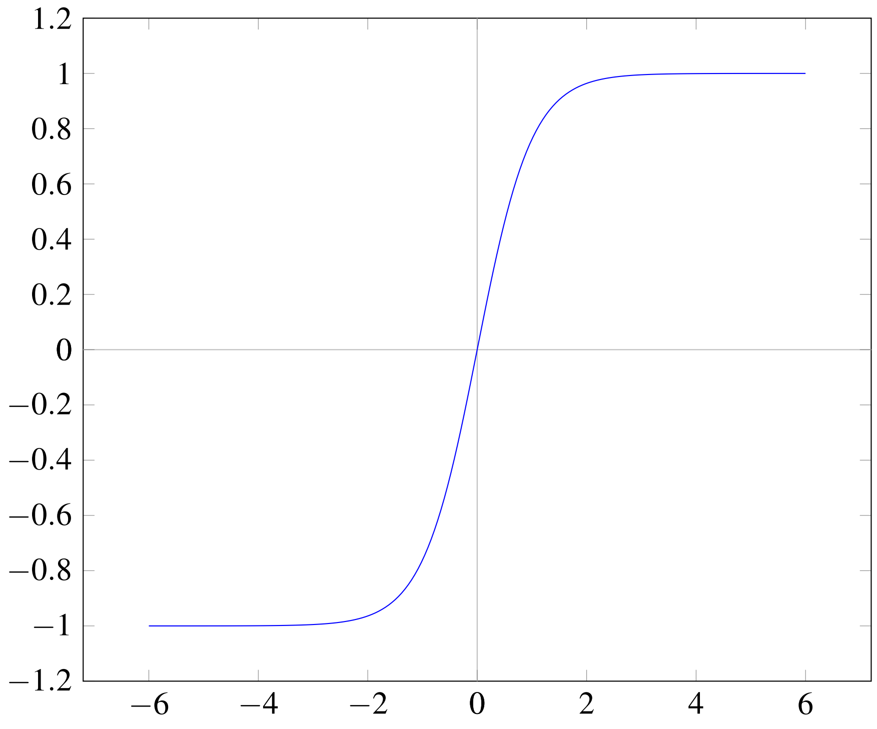


Figure 1.6: Plot of Hyperbolic Tangent activation function.

The discovery of ReLU allowed achieving higher accuracy in less time by avoiding this phenomenon. The ReLU avoided the mentioned problem by imposing an identity for the positive values, which allowed efficient and fast training of deeper architectures. In fact, by replacing the saturation in the first quadrant (present in either Sigmoid and Hyperbolic Tangent) by the linear function, the gradient can be backpropagated to deep layers without going to zero. Consequently, by allowing the training of deeper neural networks, the discovery of the ReLU (NAIR; HINTON, 2010; GLOROT; BORDES; BENGIO, 2011; KRIZHEVSKY; SUTSKEVER; HINTON, 2012) was one of the main factors that contributed to deep learning advent.

Nevertheless, even ReLU presents drawbacks. For example, some researchers argue that zero slope avoids learning for negative values (MAAS; HANNUN; NG, 2013; HE et al., 2016b). Therefore, other activation functions like Leaky Rectifier Linear Unit (LReLU) (MAAS; HANNUN; NG, 2013), Parametric Rectifier Linear Unit (PReLU) (HE et al., 2016b) and Exponential Linear Unit (ELU) (CLEVERT; UNTERTHINER; HOCHREITER, 2015) were proposed. Unfortunately, there is no consensus about how these proposals compare to ReLU, which therefore remains the most used activation function in deep learning research.

Moreover, the fact that ReLU only produces non-negative activations is particularly harmful to the batch normalization process. Indeed, before being used as inputs for the subsequent layer, batch normalized data are fed into activation functions (nonlinearities). After this, the outputs mean values after ReLU are no longer zero, but rather necessarily positives. Therefore, the ReLU skews the previous normalized distribution, undermining the batch normalization process efficacy.

1.3 GOAL

The goal of this work is to mitigate the problem mentioned above: the adverse effect the ReLU causes to the batch normalization procedure. In doing this, we aim to speed up learning and produces higher accuracy. To achieve this goal, we investigate the following fundamental research questions:

RQ1. Is it possible to enhance deep learning test accuracy performance using improved activation functions?

RQ2. Is it possible to enhance deep learning test accuracy performance without compromising the training speed?

RQ3. How much is the contribution of a given activation function independent of the model or dataset used?

RQ4. Does the choice of a particular activation function influences the overall performance of the deep models used?

RQ5. Can this enhanced performance be obtained without improving the computational resource requirements?

RQ6. Which of the proposed activation function has the best performance regarding test accuracy and training speed?

To answer **RQ1**, **RQ2**, **RQ3**, **RQ4** and **RQ5**, we intend to propose and investigate a new activation function and evaluate its training speed and test accuracy on significantly different models and datasets. The design of the activation function has the premise of not improving the computational resource requirements.

Regarding question **RQ6**, we aim to perform a comprehensive set of tests with high statistical significance ($p < 0.05$) to provide a systematic comparison of our proposal with the available commonly used activation functions.

To the best of our knowledge, this is the first study to perform a systematic comparison of the contribution of the several proposed activation function to the performance of very commonly used deep learning models and datasets using statistical tests.

1.4 OUTLINE

The remainder of this work is structured as follows:

- **Chapter 2** reviews essential concepts used throughout this work. First, we briefly introduce the deep learning recent advent as a new important research area inside machine learning and neural network community. After, we describe a particular type of deep model called Convolutional Neural Network (CNN) that has been shown to be very successfully dealing with image and other translational invariant data. Then we further describe two state-of-the-art CNN architectures largely currently used in application and research. Finally, we present the main regularization methods currently used in deep learning;

- **Chapter 3** shows the main proposed activation function used in deep learning systems. After that, we conclude the chapter with our proposed activation function called Displaced Rectifier Linear Unit;

- **Chapter 4** shows in details how the experiments were designed. Therefore, we describe the datasets, preprocessing and data augmentation methods employed. Moreover, we discuss the models, initialization, training, and regularization involved. We also describe the statistical tests we have used;

- **Chapter 5** exhibits for each dataset and model used the results of our experiments. We also discuss essential conclusions that can be derived by interpreting the experiment results;

- **Chapter 6** offers our concluding remarks of the present work. Therefore, we summarize the conclusions obtained. Nevertheless, we emphasize that more investigations are still necessary to generalize the results to other architectures and datasets. Moreover, we discuss where this work might lead and suggest other options of activation function that could be examined. Related researches are also briefly presented.

2 BACKGROUND

If I have seen further it is by standing on the shoulders of giants.

—ISAAC NEWTON

In this chapter, we review and introduce some essential concepts used in this work. First, we give an introduction and a general overview of the deep learning field of research. Later, we briefly describe the fundamentals of CNNs and present the Visual Geometry Group (VGG) and Residual Network (ResNet) model, which are two of the most commonly used type of CNNs architectures. Finally, we present the basic concepts of regularization techniques used for training deep architectures.

2.1 DEEP LEARNING

The seminal results that ignited the renaissance of interest in the artificial neural networks research were based on the usage of new unsupervised learning methods to allow the pre-training of the deepest hidden layers of a neural network (HINTON, 2005; HINTON; OSINDERO; TEH, 2006; BENGIO et al., 2006; RANZATO et al., 2007). After this procedure, known as unsupervised feature learning or representation learning, the whole network could be satisfactorily trained using the backpropagation algorithm. In other words, after an unsupervised pre-training, the deep neural network could be fine-tuned using conventional supervised learning techniques (BENGIO et al., 2006; RANZATO et al., 2007; HINTON; SALAKHUTDINOV, 2006).

In fact, the mentioned approach represented a significant theoretical advance considering that conventional shallow learning systems (artificial neural networks with few hidden layers and statistical machine learning methods of the 1990s) depend heavily on the design handcrafted pre-processing feature extractors such as SIFT (LOWE, 1999), HOG (DALAL; TRIGGS, 2005), SURF (BAY et al., 2008), etc., which is usually expensive and time-consuming. Therefore, avoiding human interference of handcrafted features is a welcome advance. Moreover, in many practical applications, much more time is spending making costly fine-tuned feature engineering than projecting the machine learning system itself. In the proposed approaches, instead of design features extractors, those features are learned directly from the data (Fig. 2.1).

Usually, despite presenting satisfactory performance in relatively small sets of labeled data, shallow learning systems tend to be unable to improve so much as the dataset gets bigger. This fact suggests that shallow architectures are less capable of capturing more subtitles characteristics of the dataset, which explain why features extractors are usually needed in a pre-processing phase.

In fact, the features learned in one layer using, for example, a Restricted Boltzmann Machine (RBM) (HINTON; OSINDERO; TEH, 2006) or an Auto-Encoder (AE) (BENGIO et al., 2006), can be hierarchically composed, layer by layer, to achieve more abstracted features from more concrete ones. The hierarchical composition of features is achieved, for example, using a Deep Belief Network (DBN) (HINTON; OSINDERO; TEH, 2006) or a Stacked Auto-Encoder (SAE) (BENGIO et al., 2006). In this sense, deep learning mimics the human way of acquiring knowledge, and therefore it is a nature inspired approach. In other words, deep learning is a way of proof that complexity (abstract concepts) can be created starting from simplicity (concrete concepts) (BENGIO, 2009; BENGIO; COURVILLE; VINCENT, 2013).

Besides, new theoretical results showed that early worries about being trapped in local minimums when training deep feedforward neural networks were overestimated (KAWAGUCHI, 2016). In fact, the reality is that the error surface of deep architectures seems to show many similar good local minimums instead of a few very high-quality global ones (DAUPHIN et al., 2014; CHOROMANSKA et al., 2014). Some others major theoretical results suggest that growing the number of layers is exponentially more efficient than increase the complexity of traditional shallow machine learning approaches or increase the number of nodes in the hidden layer of a shallow Multilayer Perceptron (MLP) or Feedforward Neural Network (FNN) (BENGIO; COURVILLE; VINCENT, 2013; BENGIO; DELALLEAU; ROUX, 2006; BENGIO, 2009; MONTUFAR; MORTON, 2012).

Another theoretical motivation to deep learning is based on the Single Algorithm Hypothesis (SAH) (DEAN; CORRADO; SHLENS, 2012), which is a claim that there is just one fundamental algorithm that probably underlies (almost) all cortical learning mechanisms. In fact, some researchers argue that the regular structure of the cortex implies the existence of a single learning algorithm all over the cortex. In this sense, some experiments show that different parts of the cortex can be able to learn functions regularly executed by others, which may be interpreted as another motivation to the mentioned hypotheses (ROE et al., 1992; MÉTIN; FROST, 1989).

Deep learning is based on the simple principle of hierarchical composition of trivial uniform procedures, the artificial neuron computation. This simplicity produces a considerable advantage over classical machine learning approaches. In fact, this characteristic is responsible by the extreme scalability that deep learning system are known to present. Hence, if enough training time and data are available, the massive scalability of deep learning approaches appears to achieve lower error rate performance than traditional machine learning alternatives.

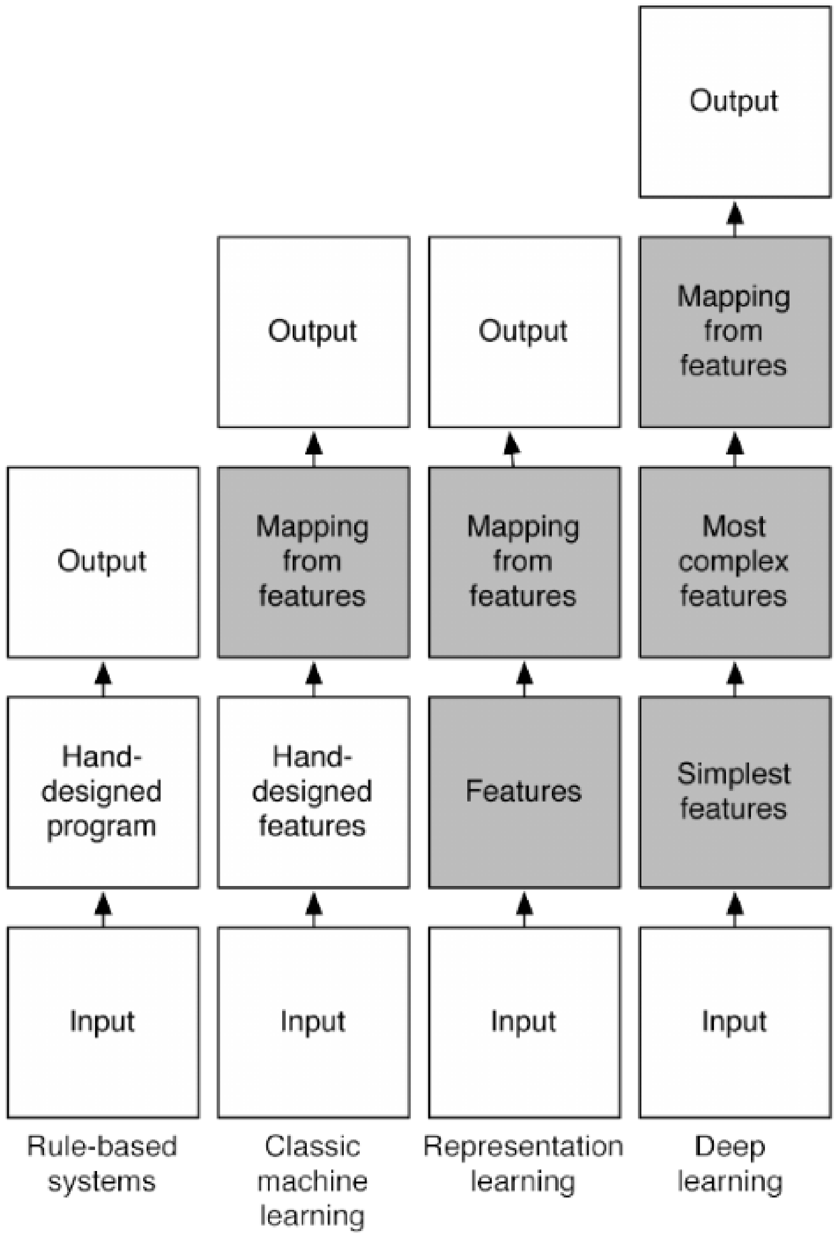


Figure 2.1: In deep learning approach, features are learned compositionally and hierarchically directly from data, avoiding the usage of previous designed handcrafted features extractors (GOODFELLOW; BENGIO; COURVILLE, 2016).

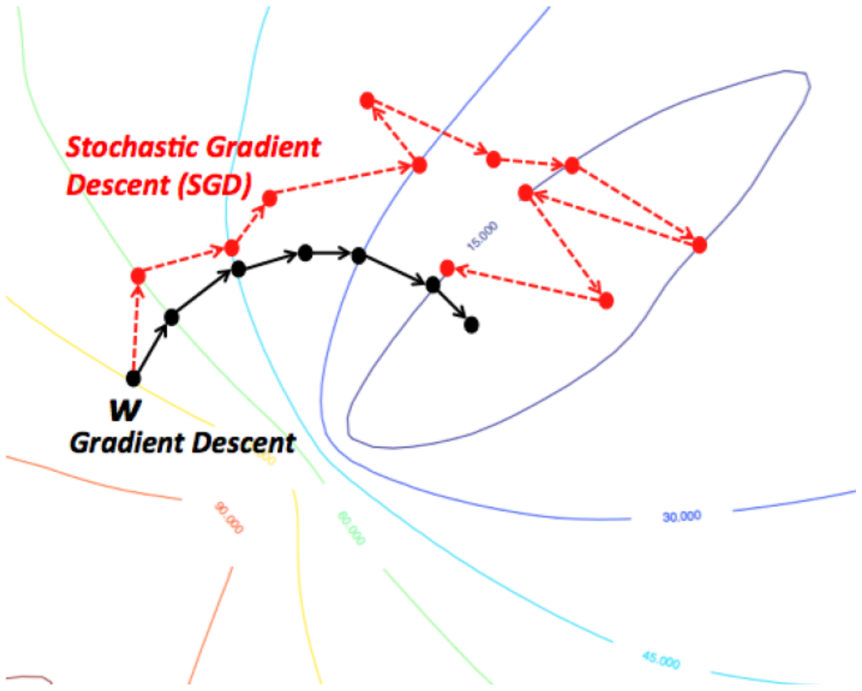


Figure 2.2: Visual comparison between deterministic gradient descent and stochastic gradient descent <https://wikidocs.net/3413>.

Another significant advantage of deep learning procedures is its incremental training characteristic. In fact, a solution based only on an analytic solution tends to be subjected to the curse of dimensionality (BENGIO; DELALLEAU; ROUX, 2006).

In addition to the discovery of the high performance of the ReLU, others techniques recently developed like dropout (SRIVASTAVA et al., 2014) and the massive use of data augmentation (KRIZHEVSKY; SUTSKEVER; HINTON, 2012) also contributed to the recent practical success of supervised trained of deep architectures.

Graphics Processing Unit (GPU) and larger datasets are other main factors that collaborate to these major achievements (KRIZHEVSKY; SUTSKEVER; HINTON, 2012). Taking into consideration that the prices of powerful parallel processing hardware are dropping fast, the training time is becoming a less important factor in the decision to adopt a particular machine learning solution when compared with its accuracy.

However, it was indeed the remarkable achievement provided by the discovery of ReLU that crucially contributed to the advent of deep learning. The other major factors were more relevant from an implementation (GPU parallel computation availability at lower prices) or circumstantial (availability of bigger datasets), but not theoretical, point of view.

The idea of training deep learning models in the so-called minibatches was also a significant milestone. Before the mentioned approach, the (shallow) neural networks were commonly trained using batch or online methods. In the first option, also called deterministic, the gradient descent and backpropagation are applied to all training examples, and then the weights are updated. This technique improves stability to training (Fig. 2.2) but makes it slow and requires high usage of memory, which makes it virtually prohibitive for deep learning applications.

The second option consists of updating the weights after the application of the gradient descended and backpropagation of each example of training. This mechanism is known as stochastic (or incremental) gradient descent, which is a stochastic approximation for the actual gradient descent calculate over the entire dataset. The main advantage of this approach is that it is incremental and the search for the minimum of the objective function can be persecuted interactively processing sequentially the examples of training, which speeds up training and dramatically lowers the required use of memory. However, this approach may lead to instability and convergence problems (Fig. 2.2).

The minibatch is a compromise achieved between this two extreme methods. The gradients are calculated over mini-batches of examples of training that typically is composed of a small fraction of the total that exists in the dataset. However, this amount is enough to avoid instabilities and very often converge smoothly. Despite being a simple trick, the minibatch allows an adequate training of massive datasets incrementally, which is one of the main factors that make deep learning very scalable when compared to alternative machine learning approaches.

Deep learning not only has changed the vision recognition field forever, but it also has become the mainstream solution for other machine learning fields, like speech recognition. In fact, the recent advances in speech recognition using Recurrent Neural Network (RNN) were possible due to an important paper that improved the performance of the mentioned approach when labeling unsegmented sequence data (GRAVES et al., 2006). Besides, in a seminal article from 2012 (HINTON et al., 2012), George Hinton, Microsoft and others research labs showed that deep learning techniques could reduce the error rate of current state-of-the-art speech recognition systems by about 25%, a remarkable result.

Currently, the replacement of Gaussian Markov Model (GMM) by Deep Neural Network (DNN) as the acoustic model in the early stages of a speech recognition system is a very established approach. In state-of-the-art speech recognition systems, the acoustic and language models are replaced altogether by a unified RNN (AMODEI et al., 2016).

2.2 ACTIVATION FUNCTIONS

Currently, the all major activation functions used in deep learning (ReLU, LReLU, PReLU and ELU) adopt the identity transformation to positive inputs, some particular function for negative inputs, and an inflection on the origin. In the following subsections, we briefly describe the nonlinearity layers compared in this work using statistical tests.

2.2.1 Rectifier Linear Unit

ReLU has become the standard activation function used in deep networks design. Its simplicity and high performance are the main factors behind this fact. The follow equation defines ReLU:

$$y = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.1}$$

The Eq. (2.1) implies ReLU has slope zero for negative inputs and slope one for positive values. It was first used to improve the performance of Restricted Boltzmann Machines (RBM) (NAIR; HINTON, 2010). After that, ReLU was used in other neural networks architectures (GLOROT; BORDES; BENGIO, 2011). Finally, ReLU has provided superior performance in the supervised training of convolutional neural network models (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

The identity for positive input values produced high performance by avoiding the vanishing gradient problem (HOCHREITER, 1991). An arguable drawback is the fact ReLU necessarily generates positive mean outputs or activations, which generate the bias shift effect (CLEVERT; UNTERTHINER; HOCHREITER, 2015). Consequently, while the identity for positive inputs is unanimously accepted as a reliable design option, there is no consensus on how to define promising approaches for negative values. The present work contributes to elucidate this important question.

2.2.2 Leaky Rectifier Linear Unit

LReLU was introduced during the study of neural network acoustic models (MAAS; HANNUN; NG, 2013) (Fig. 2.3). This activation function was proposed to avoid slope zero for negative inputs. The following equation defines it:

$$y = \begin{cases} x & \text{if } x \geq 0 \\ \beta x & \text{if } x < 0 \end{cases} \tag{2.2}$$

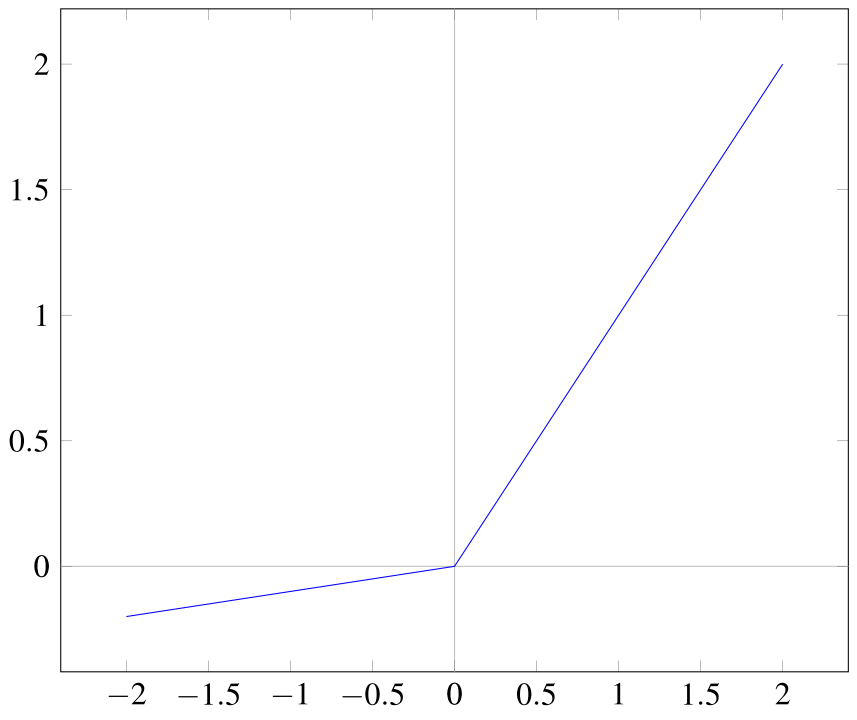


Figure 2.3: Plot of LReLU activation function.

LReLU has no zero slope if $\beta \neq 0$. In fact, it was designed to allow learning to happen even for negative inputs. Moreover, since LReLU does not necessarily produce positive activations, the bias shift effect may be reduced.

2.2.3 Parametric Rectifier Linear Unit

PReLU is also defined by the Eq. 2.2, but in this case β is a learnable rather than a fixed parameter (HE et al., 2016b) (Fig. 2.4). The idea behind the PReLU design is to learn the best slope for negative inputs. However, this approach may implicate in overfitting since the learnable parameters may adjust to specific characteristics of the training data.

2.2.4 Exponential Linear Unit

ELU has inspiration in the natural gradient (AMARI, 1998; CLEVERT; UNTERTHINER; HOCHREITER, 2015) (Fig. 2.5). Similarly to LReLU and PReLU, ELU avoids producing necessarily positive mean outputs by allowing negative activation for negative inputs. The Eq. 2.3 defines ELU:

$$y = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(exp(x) - 1) & \text{if } x < 0 \end{cases} \tag{2.3}$$

The main drawback of ELU is its higher computational complexity when compared to activation functions such as ReLU, LReLU, and DReLU.

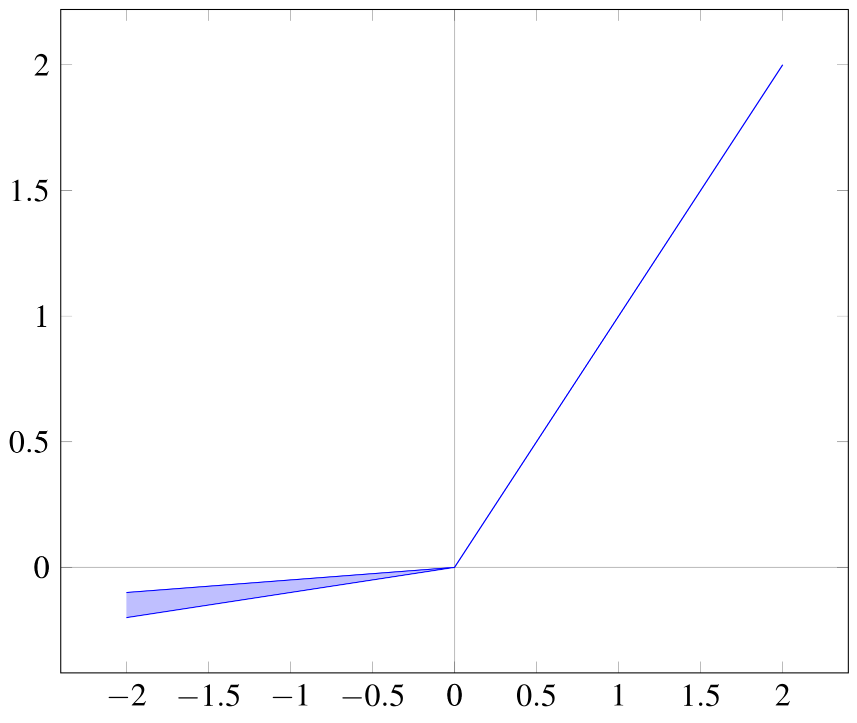


Figure 2.4: Plot of PReLU activation function.

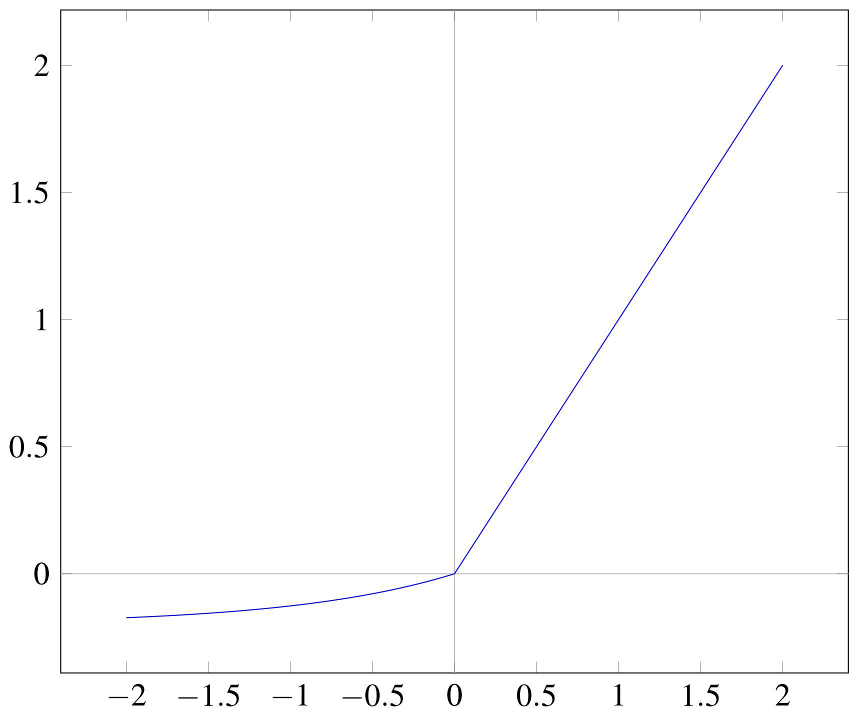


Figure 2.5: Plot of ELU activation function.

2.3 CONVOLUTIONAL NETWORKS

After some unsupervised theoretical results in representation learning, practical performance using purely supervised learning techniques to train deep architectures were observed. In fact, if the late 2000s showed some major unsupervised advances, the beginning of the 2010s presented the deep learning supervised remarkable results. Indeed, in the last few years, deep neural networks have won many prestigious contests in machine learning due to some recent supervised deep learning techniques discoveries (LECUN et al., 2012).

It was the supervised techniques advances that have made deep learning gain much more attention recently after deep neural architectures have been declared winners of a series of respected competition (CIRESAN et al., 2013; ROUX et al., 2013). In the most emblematic example, the ImageNet Large Scale Visual Recognition Challenge 2012 (ILSVRC 2012), the AlexNet architecture, which is a type deep architecture that uses convolutional layers and supervised learning to extract deep features of image data, transformed the computer recognition field (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

In fact, while the winner AlexNet solution presented error rate in the main classification task of 16.4%, the second best solution showed a performance of 26.2%, about 10% percentage points higher. It should be noticed that the competitors were allowed to use regular hand-designed feature extractors. Naturally, the AlexNet solution did not make use of this resources since the whole point of deep learning is avoid this approach and learning useful features directly from essentially raw data.

The ILSVRC 2012 used the ImageNet dataset in a composition of 1.2 million 256x256 color images to be classified in a recognition task of 1000 classes. In the subsequent years, practically all competitors adopted CNN architectures with distinct innovation aspects. Recently, CNN has become the mainstream approach to vision recognition related tasks.

Today, models such as VGG (SIMONYAN; ZISSERMAN, 2014) and Residual Networks (HE et al., 2016a; HE et al., 2016c) present much higher performance than the historic AlexNet. Nevertheless, some contribution of AlexNet persists to date. For example, AlexNet changed a common believe that supervised deep learning models needed unsupervised pre-training to achieve high performance. After ILSVRC 2012, it has become clear that unsupervised pre-training does not improve the performance of purely supervised training in these situations. It also clearly proved that learning compositional and hierarchical features (BENGIO, 2013) outperforms, in some circumstances, hand-crafted features (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

Historically, supervised training of CNNs makes regular use of data augmentation (CIREŞAN et al., 2011), the most common are the use of random crops and horizontal reflections. For example, AlexNet used five random crops of size 224x244 of each image for training. Moreover, horizontal reflections of each of those patches were added. Therefore, a total of more than 10 millions of image crops were used to train the model in ILSVRC 2012.

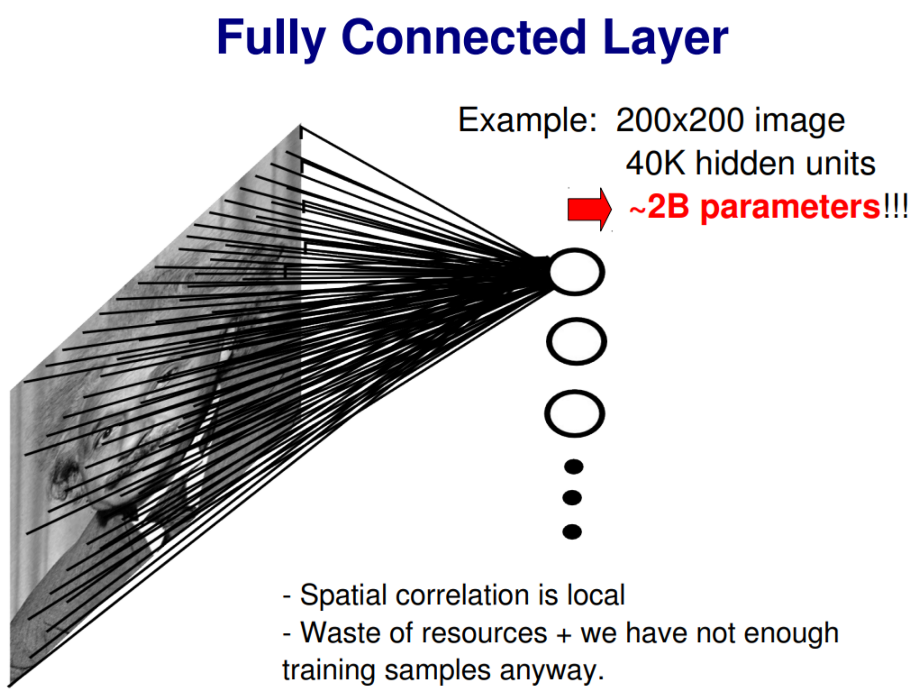

Fully Connected Layer
Example: 200x200 image
40K hidden units
~2B parameters!!!
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway.

Figure 2.6: A fully connected architecture produces a insurmountable number of parameters when dealing with image data <https://sites.google.com/site/deeplearningcvpr2014/>.

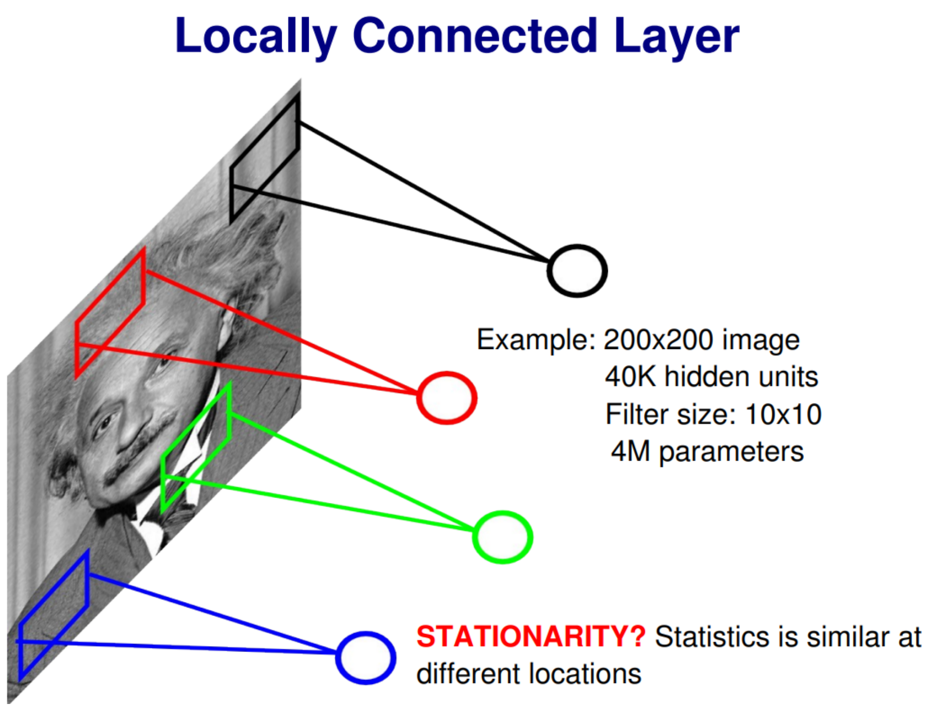


Figure 2.7: A locally connected architecture produces a high number of parameters when dealing with image data <https://sites.google.com/site/deeplearningcvpr2014/>.

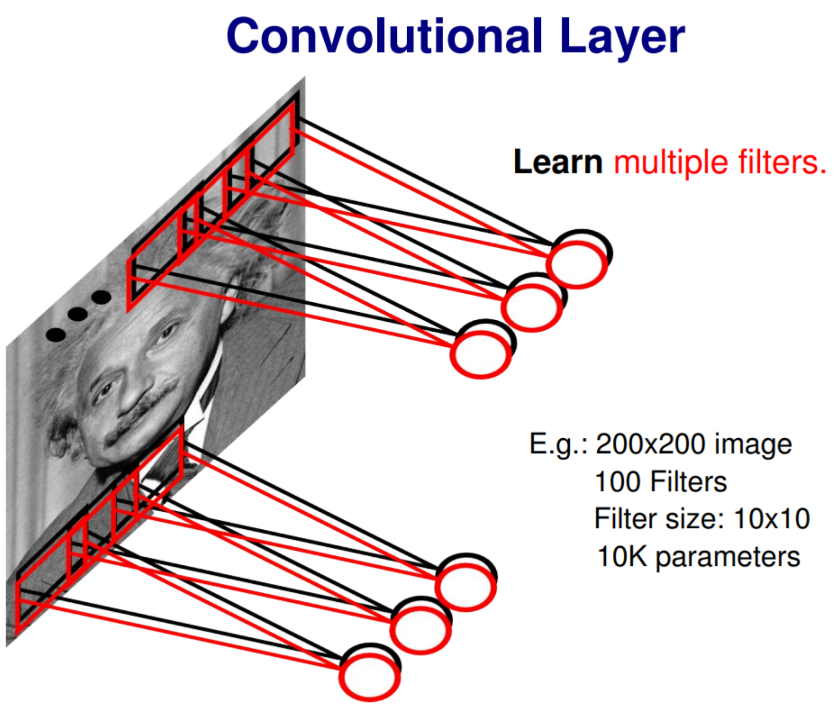


Figure 2.8: A convolutional architecture produces a reasonable number of parameters when dealing with image data <https://sites.google.com/site/deeplearningcvpr2014/>.

In fact, since the begging of the second cycle of intense neural networks research in the 80's, some tries of this paradigm to image recognition tasks were observed. The major obstacle was the fact that multilayer perceptrons or feedforward neural networks composed of nodes fully connected to the previous neurons produced an insurmountable number of parameters, which made the training of such models completely inviable. In fact, even currently much bigger image datasets are not enough to train this kind of fully connected architectures (Fig. 2.6).

The Neocognitron of Kunihiko Fukushima (FUKUSHIMA, 1980) was designed to overcome this problem. In reality, the Neocognitron is a multilayer feedforward neural network that was adapted with geometric a priori considerations to treat image data better. The main contribution of the Neocognitron was the introduction of the concept of translational invariance. Kunihiko Fukushima understood that a given pattern in a crop of an image (or any translational invariant datum) does not lose its essence by being translationally displaced. Therefore, he proposed models that do not need to be fully connected to all the nodes of the previous layer, but only to the ones located in a small local region called a receptive field (Fig. 2.7).

Moreover, he realized that to incorporate translational invariance to the model, it was also necessary that all the nodes that composed each receptive field in a given feature plane shared the small set of weights. This utmost propriety is called weight sharing. The term convolution is then applied to the composition of this two fundamental architectural design options (Fig. 2.8).

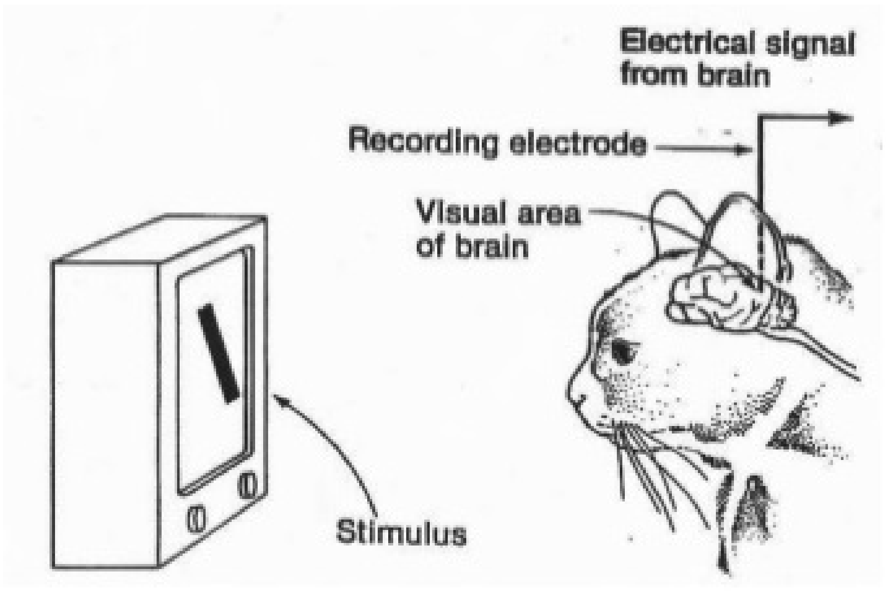


Figure 2.9: Experimental setup for study of the visual cortex of cats (HUBEL; WIESEL, 1959).

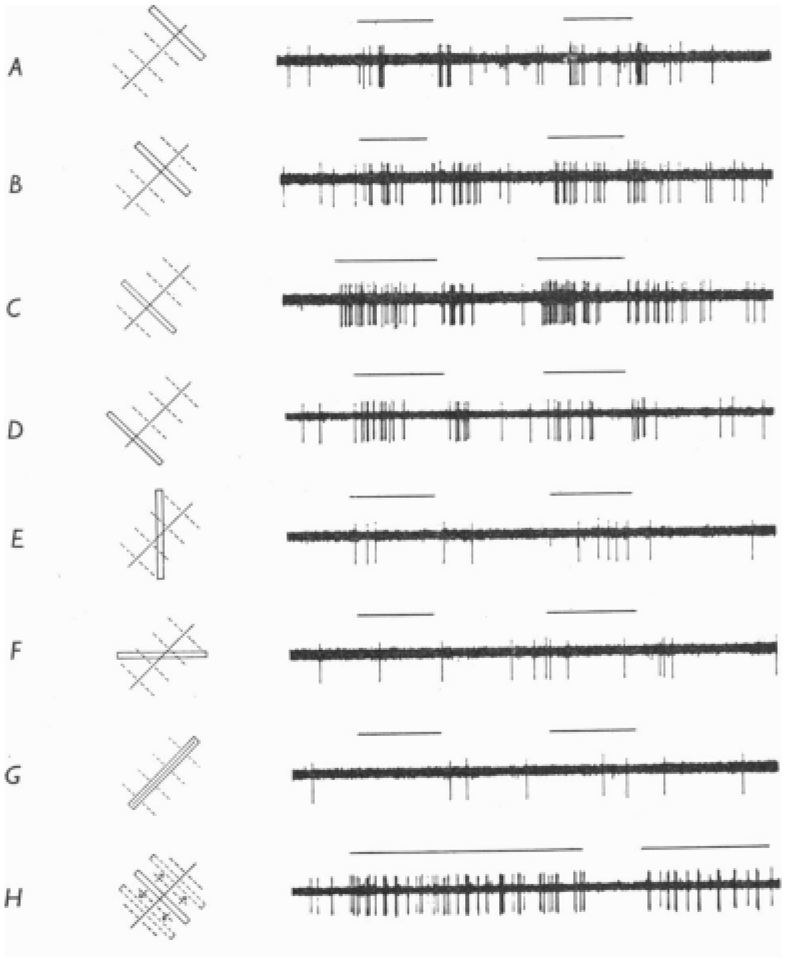


Figure 2.10: Demonstration of the existence of receptive fields in visual cortex of cats. Only specific neurons activate each time (right) while the bar moves (left) (HUBEL; WIESEL, 1962).

The design of the convolutional layers was natural inspired and clearly based on the seminal work of Hubel and Wiesel (HUBEL; WIESEL, 1959; HUBEL; WIESEL, 1962) that study the presence of receptive fields and functional architecture in the visual cortex of cats. In this work, electrical signals were captured from the brain of cats to study how their visual cortex respond to stimulus (Fig. 2.9). In fact, the study showed that the visual cortexes of cats are composed of areas specialized in responding to particular visual stimulus, the so-called biological receptive fields (Fig. 2.10). The mentioned work showed that each portion of the visual cortex is responsible for responding to specific local regions of the visual fields. In other words, a particular neuron is unaware of what happens to visual signals that are not is its local areas of particular interest.

The design of the Neocognitron based on receptive fields and weight sharing not only incorporated the translational invariance a priori to the architecture but also dramatically lower the number of parameters of the model when compared with approaches using fully connected neurons presented by classical multilayer perceptrons or feedforward neural networks. This fact enables modern CNNs to be trained and present high generality and low overfitting. Kunihiko Fukushima tried to train its Neocognitron using unsupervised learning.

Another significant contribution towards modern CNNs was made by Yann LeCun, which showed that similarly improved architectures could be trained using the backpropagation algorithm in the 1990's. Those new architectures were collectively called LeNet (LECUN et al., 1990; LECUN et al., 1998). Yann LeCun had success in tasks related to the classification of handwritten digits using the MNIST dataset (LECUN; CORTES; BURGES, 1998).

Therefore, the AlexNet was based on a previous architecture designed by Yann LeCun and called LeNet (LECUN et al., 1990; LECUN et al., 1998), which in turn was based on the works of Kunihiko Fukushima on the Neocognitron (FUKUSHIMA, 1980; FUKUSHIMA; MIYAKE, 1982; FUKUSHIMA; MIYAKE; ITO, 1983; FUKUSHIMA, 1988). Therefore, the Neocognitron was the architecture that essentially introduced the convolutional paradigm while the main contribution of Yann LeCun with his LeNet was to use the backpropagation algorithm to train Neocognitron inspired models.

The main building blocks are the convolutional, nonlinearity and pooling layers. A hierarchical composition of this type of layers is performed to construct a current CNN. The structure of a modern CNN is shown in the Figure 2.11. Typically, some fully connected layers are added at the top to work as classifiers of the features extracted in the previous layers. If not otherwise mentioned, the simple layer terminology was used in this work.

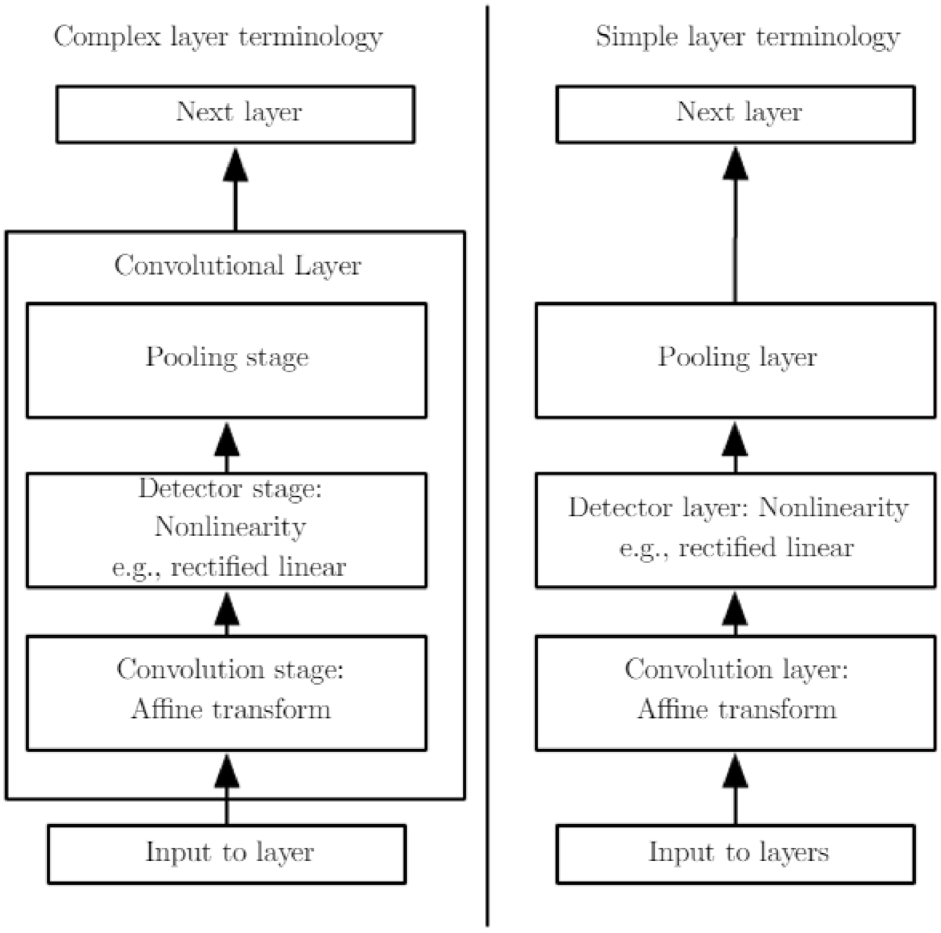

Complex layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Input to layer
Simple layer terminology
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform
Input to layers

Figure 2.11: Structure of a generic Convolutional Neural Network (GOODFELLOW; BENGIO; COURVILLE, 2016).

The fundamental of mathematical operation of a modern convolutional is presented in the Figure 2.12. The selective fields mentioned above give rise to the so-called kernels of the convolutional layers. Those kernels are composed of weights that are learned during the training of the model. The kernels (sometimes called filter banks) compose the learnable features of the convolutional layer. The layers are organized as squares of a given size. The inputs are the raw data pixels (in the case of the first layer) or the activations of the previous layer. The activations of the current layer in the forward pass are then calculated as shown in the mentioned figure.

The weight sharing characteristic of the convolutional layers can be understood as a way of passing the kernel horizontally and vertically over the input. The number of activations the kernel is displaced each time is called stride. In the case of the example, we have stride equals to 1x1. It is also possible to padding the input to allow a part of the kernel to move outside the input. The two-dimensional size of the output activations depends on the size of the input and the kernel, the padding, and the stride. In the Figure 2.12, the sizes of input and the kernel are, respectively, 3x4 and 2x2; the stride equals to 1x1 and there is no padding. In this case, the output has a height of 2 and a width of 3, which composes a 2x3 size output activation. The Figure 2.13 presents a numeric example of the calculation of one output activation.

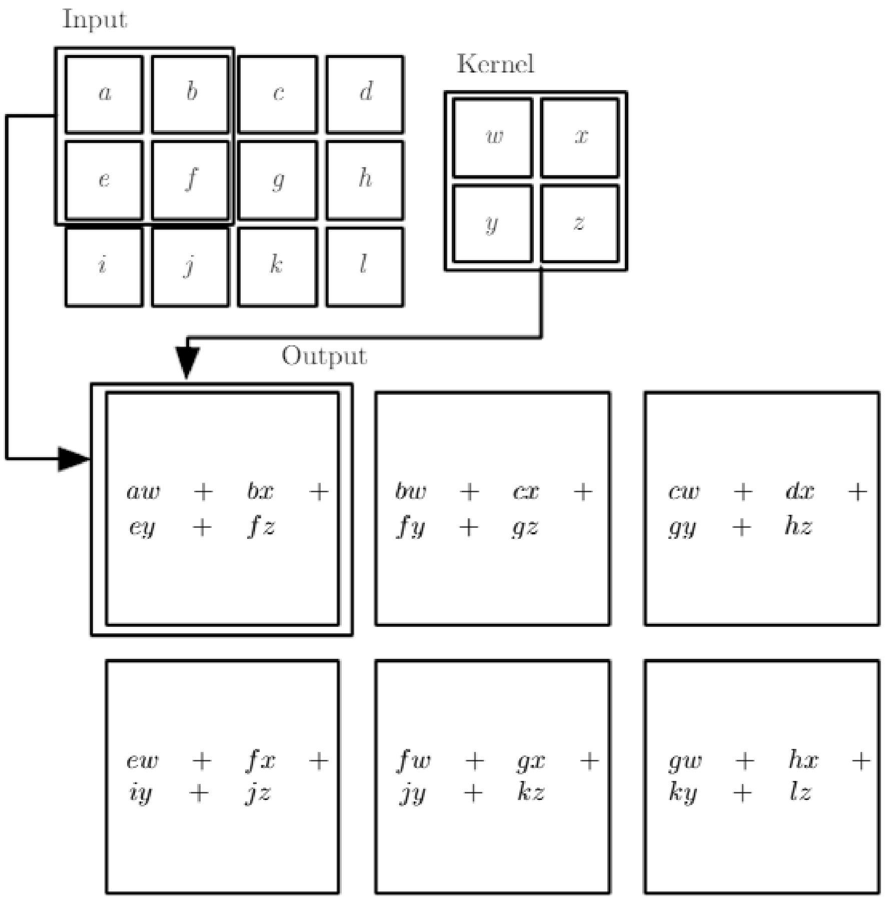


Figure 2.12: Fundamental of a convolutional kernel forward processing (GOODFELLOW; BENGIO; COURVILLE, 2016).

A kernel represents just a single two-dimensional feature map, which is defined as the set of all nodes or activations that belongs to the same convolutional layer and share the set of weights placed at distinct receptive fields. Naturally, many feature maps compose a given convolutional layer. Therefore the output of convolutional layer can be understood as a prism where each vertical plane is the output produced by a specific kernel (Fig 2.14). In this sense, in fact, convolutions are not a two-dimensional operation, but three-dimensional ones since the weights to a kernel come from a receptive field that has a depth in the output activations prism.

The Figure 2.15 presents a visualization of the kernels learned by the first layer of the AlexNet in the ILSVRC 2012. The 96 kernels are shown side by side in a disposition of 6 rows and sixteen columns. Since two GPUs were used, 48 kernels were learned by each GPU. Please, notice the depth of 48 planes of activations (feature maps) in the first layer of the Figure 2.15.

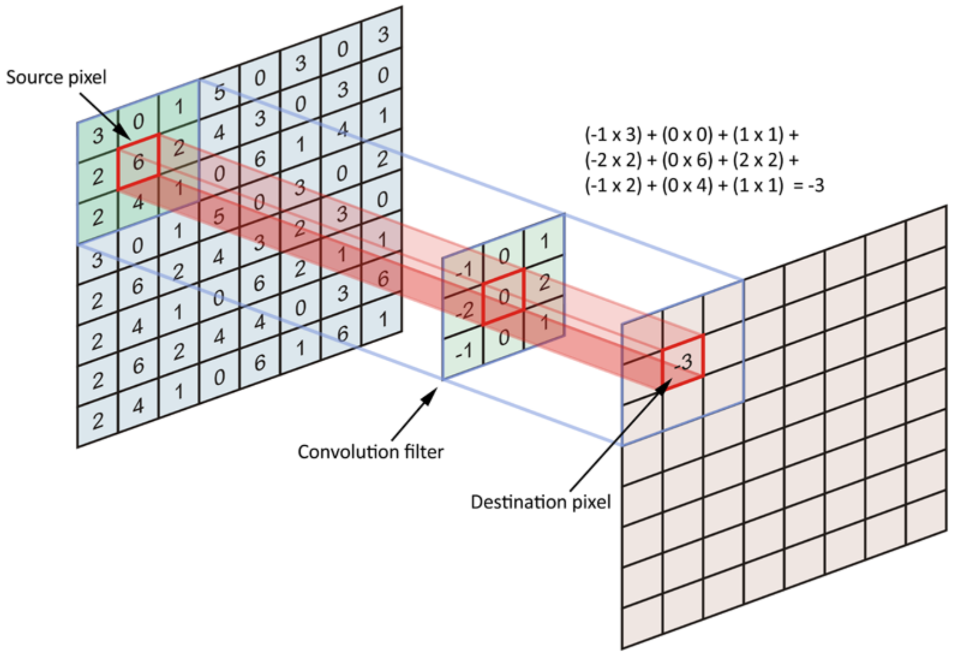

Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

Figure 2.13: Passing a convolutional kernel over a image <https://community.arm.com/>.

Therefore, different from classical fully connected neural networks (Fig. 2.16), CNNs arranges its neurons in three-dimensional prisms composed of width, height, and depth. In this sense, each convolutional layer of the models transforms a prism of input activations into another prism of output activation (Fig. 2.17). The width and height of a given activation (or neurons) prism are determined, as seen above, by the width and height of the input prism and by the sizes of the kernel, the stride the padding used. The depth of the output nodes (or activations) prism is a design option.

Besides the convolutional and nonlinearity layers, modern CNNs usually made use of max pooling layers, which function is just to capture the highest activation in a given square region. A polling layer allows capturing the information that a particular visual feature is present in a specific small region, which improves the robustnesses of the solution (Fig. 2.18).

The Figure 2.19 illustrates this operation to a kernel of size 2x2 and stride equals to two for a given plane of activations. Again, the actual operation is performed in a three-dimensional way as shown if the Figure 2.20. Unlike convolutional layers, max pooling layers can only change the width and the height of the input volume when generating the output volume, but the depth of the input prism is necessarily maintained in the output prism. Typically max pooling layers are placed after some of the convolutional layers presented in the model.

Finally, after a sequence of convolutional and max pooling layers that indeed perform the supervised feature learning, typically the CNN model is concluded by adding some traditional fully connected layers, which work as classifiers in a similar way they operate in traditional fully connected neural networks.

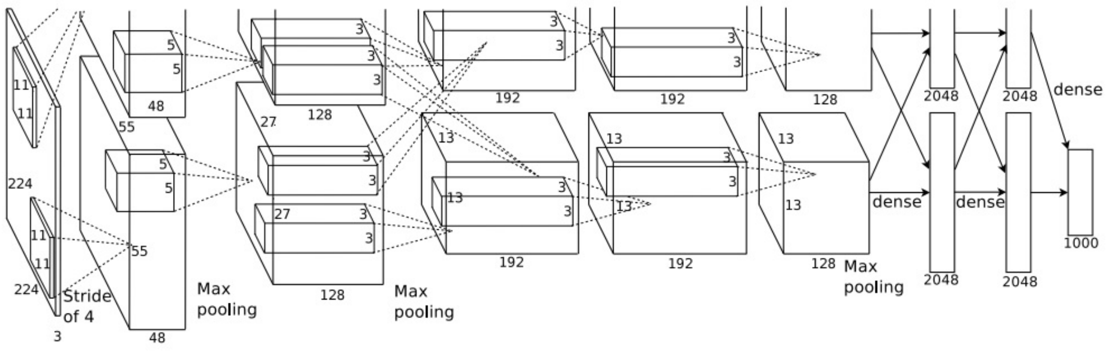


Figure 2.14: An illustration of the architecture AlexNet, the winner of the ImageNet Large Scale Visual Recognition Challenge 2012 (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

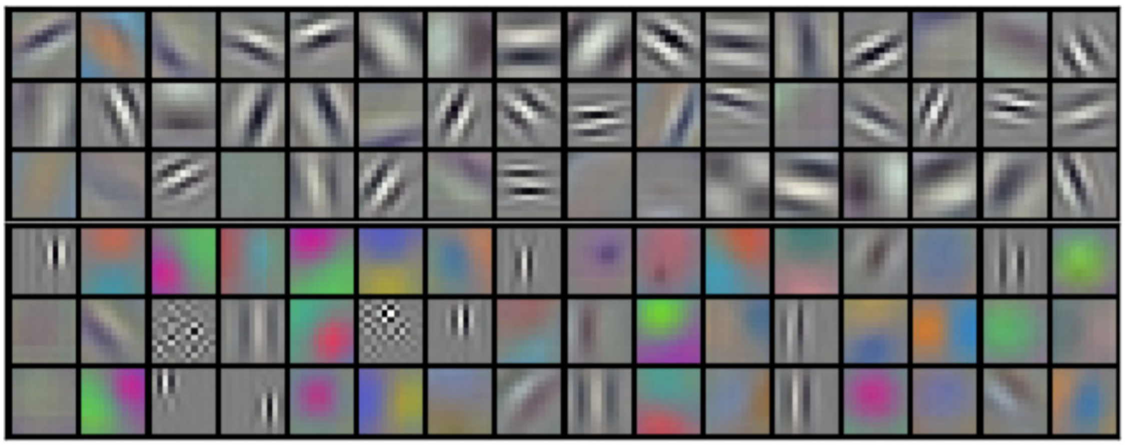


Figure 2.15: 96 convolutional kernels of size 11×11×3 learned by the first convolutional layer on the 224×224×3 input images using two parallel GPUs (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

For examples, the AlexNet (Fig. 2.14) aggregated five convolutional layers, some max-polling layers after specific convolutional layers, three fully-connected layers, from which the last one has 1000 nodes to final classification. The total model is composed of about 60 million parameters and incorporates 650,000 neurons.

It is a common practice to add the so-called Softmax layer after the last fully connected one. In this sense, consider the transformation performed by the last fully connected layer of a deep model given by:

$$\boldsymbol{y} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) \tag{2.4}$$

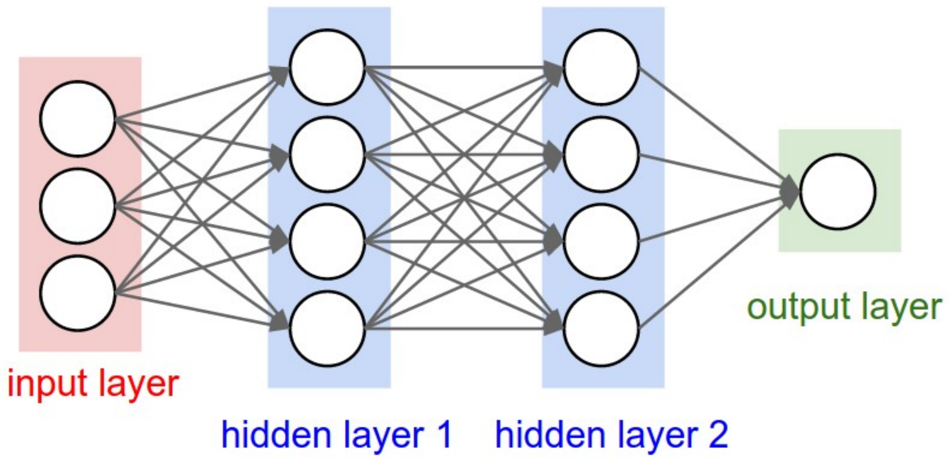


Figure 2.16: Visualization of a traditional fully connected neural network. Stanford University: Convolutional Neural Networks for Visual Recognition <http://cs231n.github.io/>.

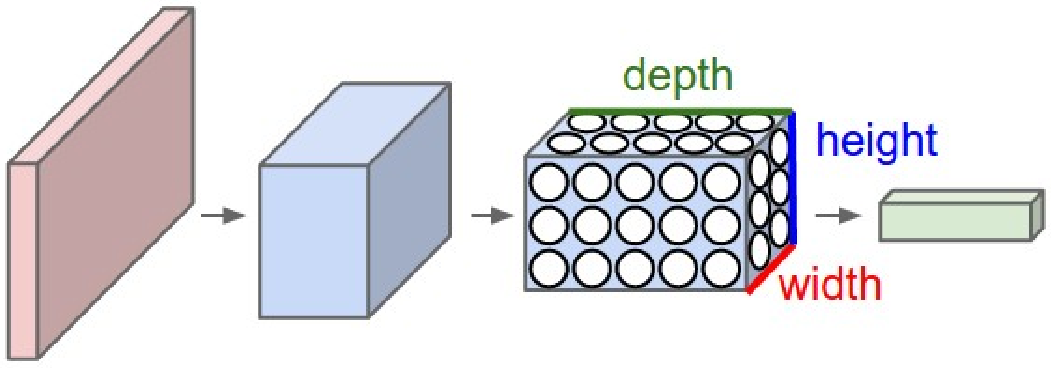


Figure 2.17: Visualization of a modern convolutional neural network. Stanford University: Convolutional Neural Networks for Visual Recognition <http://cs231n.github.io/>.

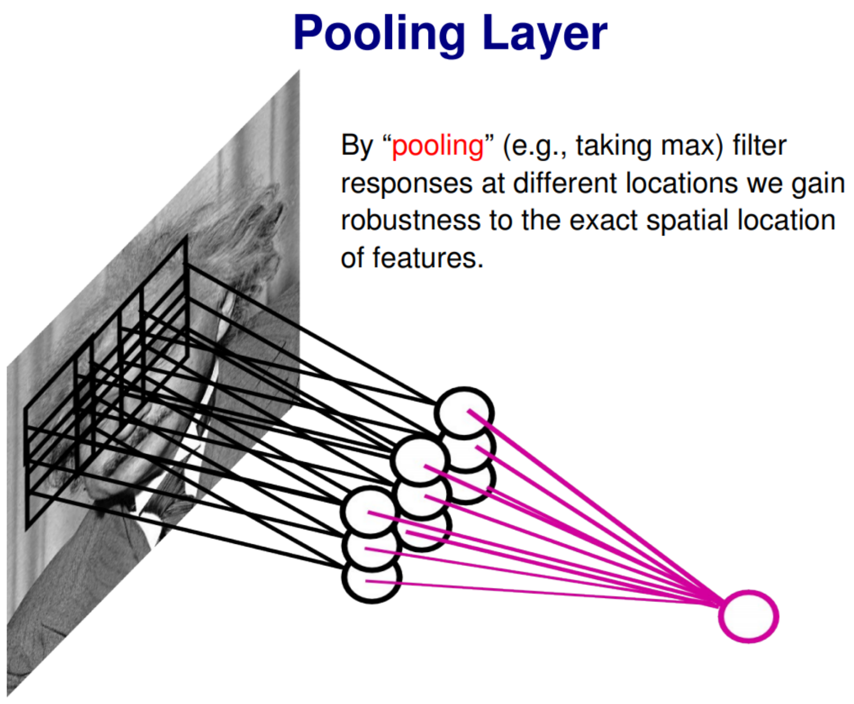

Pooling Layer
By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Figure 2.18: A pooling layer is a detector which perceives that visual features are present in a given small region <https://sites.google.com/site/deeplearningcvpr2014/>.

If we define $\boldsymbol{z} = softmax(\boldsymbol{y})$ as being the output of the application of a Softmax layer to $\boldsymbol{y}$, then we can write the equation to the component i of $\boldsymbol{z}$ as follows:

$$z_i = \frac{exp(y_i)}{\sum_j exp(y_i)} \tag{2.5}$$

The Softmax is applied to the scores values provided by the previous layer to allow a probability interpretation of the chance of each class (Fig. 2.21). Indeed, the original scores values of -2.85, 0.86 and 0.28 are almost meaningless from a human point of view. However, the output values of the softmax can be directly interpreted as the probabilities of classifying a given image in each class.

For example, from the mentioned figure we can obtain the information that the likelihood of the training sample belongs to the second class is 0.631. In this sense, the Eq. 2.6 assert that the probability of a example of training x_i be classified in the class j given the parameters $\boldsymbol{\theta}$ is equal to z_j. In mathematical terms:

$$P(y_j|x_i;\boldsymbol{\theta}) = z_j \tag{2.6}$$

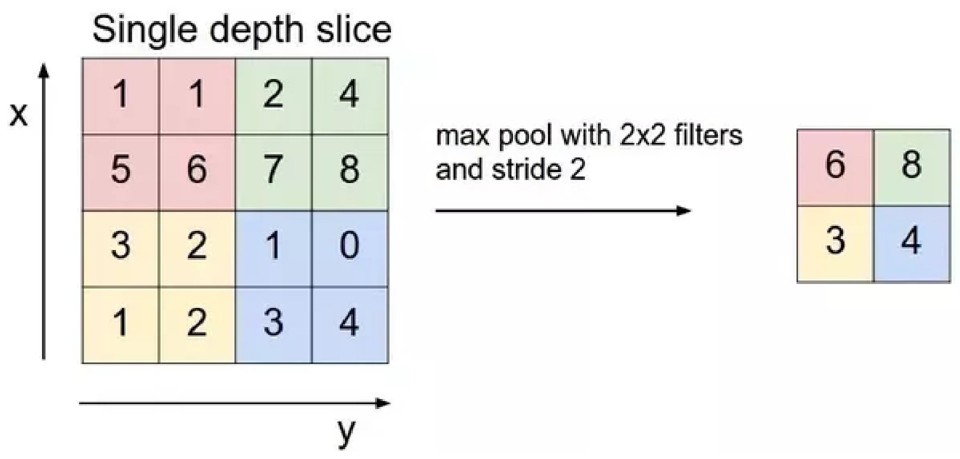


Figure 2.19: Passing a max pooling over a image (two-dimensional view). Stanford University: Convolutional Neural Networks for Visual Recognition <http://cs231n.github.io/>.

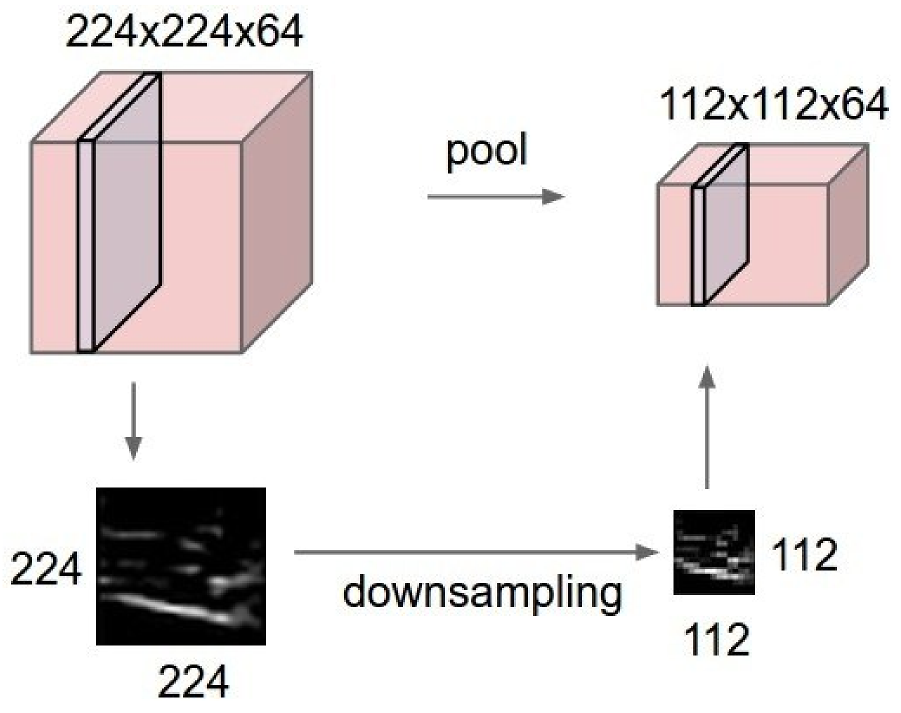


Figure 2.20: Passing a max pooling over an activation prism (three-dimensional view). Stanford University: Convolutional Neural Networks for Visual Recognition <http://cs231n.github.io/>.

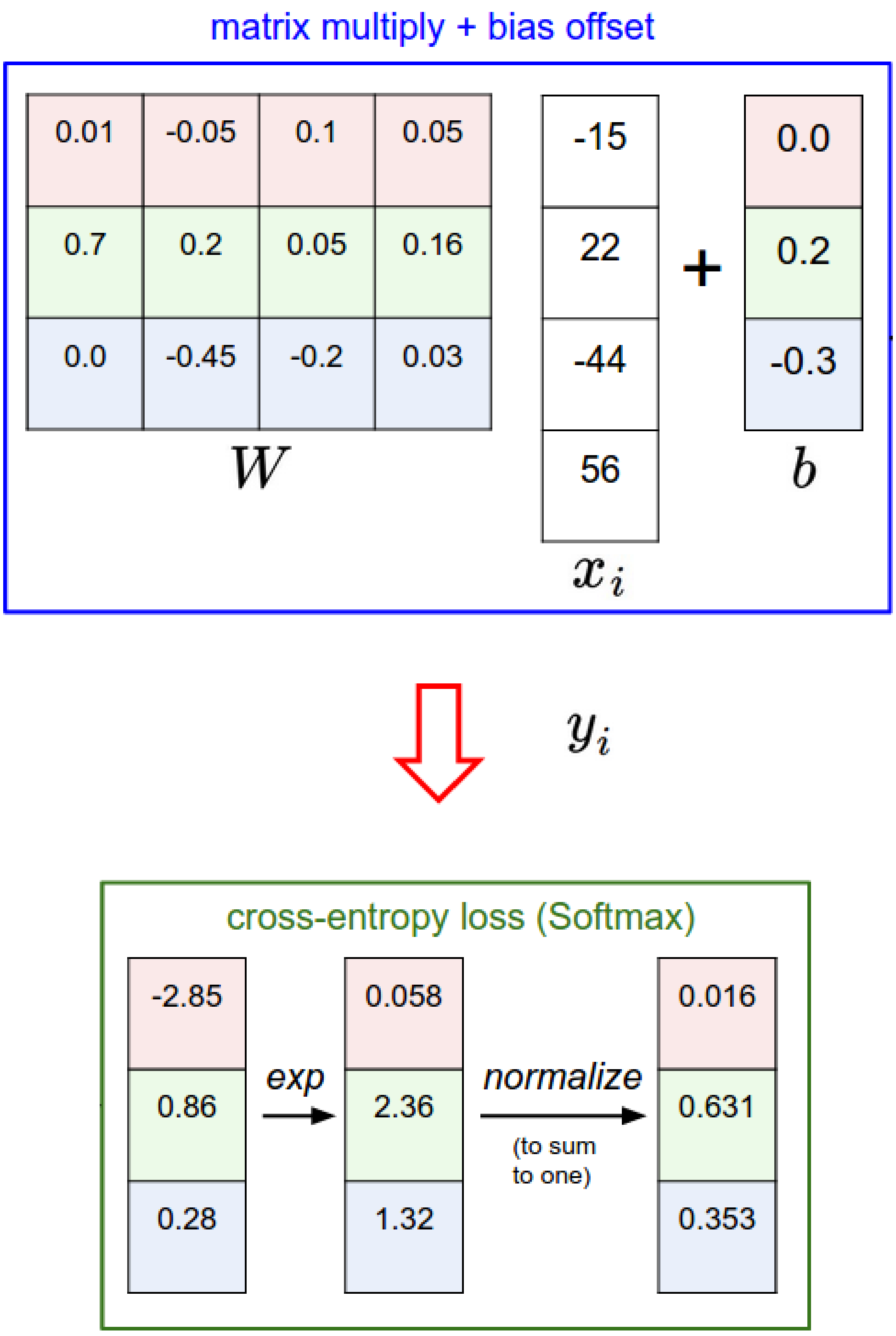


Figure 2.21: Example of mathematical operation performed by a Softmax layer Stanford University: Convolutional Neural Networks for Visual Recognition. <http://cs231n.github.io/>.

Associated with the Softmax layer is the cross-entropy loss. In fact, the information theory establishes that cross entropy between two probability distribution is a measure of how much they are similar. Therefore, given two probabilities distributions given by $p(x)$ and $q(x)$, the cross-entropy $H(p,q)$ between those is defined by the following equation:

$$H(p,q) = -\sum_i p_i \, log(q_i) \tag{2.7}$$

Therefore, if we want the distribution of probabilities for the classes given by the Softmax some how to represent the labeled data, we should then calculate the cross-entropy between them. Considering the models should present probability equals to one for the correct class and equals to zero for all others, the application of the Eq. 2.7 on our particular case implicates that:

$$L = -log(z_i) \tag{2.8}$$

In other words, the loss function is just the negative of the log of the Softmax value of the correct class (Eq. 2.8). Naturally, as expected, if the deep network predicts probability near zero to the correct class, the loss will be high. However, if the deep network predicts probability near one to the correct class, the loss will be low.

2.4 ARCHITECTURES

2.4.1 Visual Geometry Group

Despite its contributions and historical importance, AlexNets does not represent a state-of-art model today. Therefore, in this study, we opted to employ two state-of-the-art CNNs models that are commonly used to date.

The Visual Geometry Group architectures (SIMONYAN; ZISSERMAN, 2014) are composed of medium size models of up to 19 layers using small 3x3 convolutional kernels with stride equals to one and 2x2 pooling layers.

The VGG-19 models is of particular practical interest considering its large use (Fig. 2.22). Each prism represents a particular type of layer. Specifically to convolutional layers, the number of layers which correspond to each block is showed in figure. For other types of layers, each prism represents only one layer.

For example, the second, fourth, sixth, eighth, tenth prisms are composed of a single max pooling layer. However, while the first and the third prisms represent two convolutional layers each, the fifth, the seventh and the ninth prisms represent four convolutional layers each. Adding the three final fully connected layer, we obtain the whole nineteen layers.

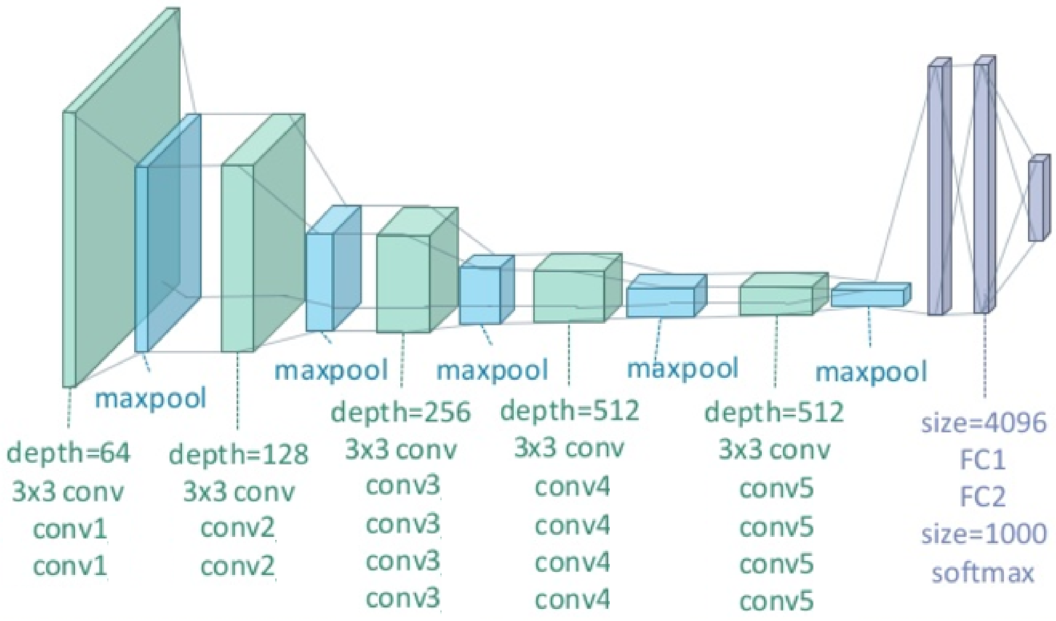


Figure 2.22: VGG 19 layers model visualization (SIMONYAN; ZISSERMAN, 2014).

The mentioned figure also informs the depth of each convolutional and max pooling prism. Therefore, we can observe that the third prism has a depth composed of 256 feature maps. Naturally, the depth of the max pooling layers equals the depth of the previous convolutional one. ReLU nonlinearity layers presented after each convolutional layer are not shown for brevity. The first two fully connected layers have 4096 nodes each, and the third has a thousand nodes.

The family of VGG configurations are presented in the Table 2.1. These are remarkably homogeneous architecture, mainly considering only the two options VGG-16 and VGG-19, which are in fact the two more commonly used.

The main contributions of this study were to show that increasing the depth of the models to more than ten layers as well as reducing the convolutional kernel size contributes to enhances the overall system performance (SIMONYAN; ZISSERMAN, 2014). The VGGs were the winner of the ILSVRC 2014.

2.4.2 Residual Networks

Residual Networks (HE et al., 2016a) introduced the skip connections (Fig. 2.23), which are currently the build blocks to construct very deep neural networks. In such case, instead of learning a complete transformation of activations from a layer to another, only a differential map is learned, which improves the backpropagation of the gradient allowing the training of deep networks with depths variating from a few tens to thousands of layers.

Subsequently, a variation of this architecture called pre-activation residual networks was proposed by placing the activation function before the weight layer in each residual building block (HE et al., 2016c), which improved the performance (Fig. 2.24). The Residual Networks were the winner of the ILSVRC 2015.

Table 2.1: VGG architecture configuration options.
The ReLU layers after each convolution are not shown
(SIMONYAN; ZISSERMAN, 2014).

ConvNet Configuration					
A	A-LNR	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	13 weight layers	16 weight layers	19 weight layers
input (224 x 224 RGB image)					
conv3-64	conv3-64 LNR	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
softmax					

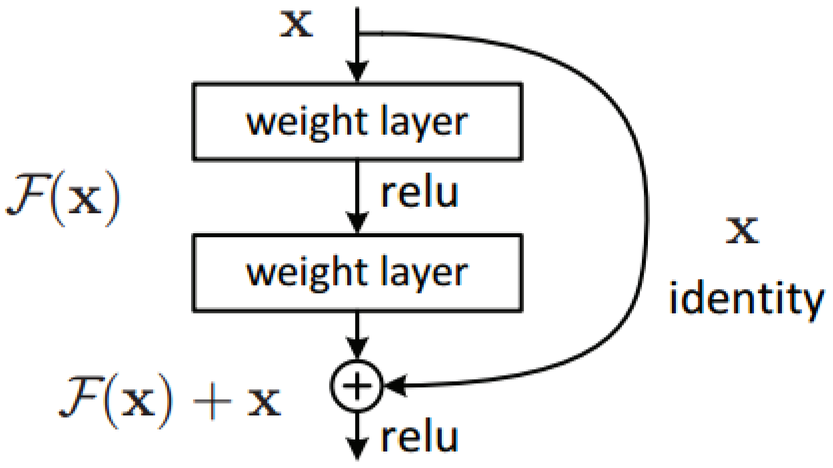


Figure 2.23: A residual learning building block (HE et al., 2016a).

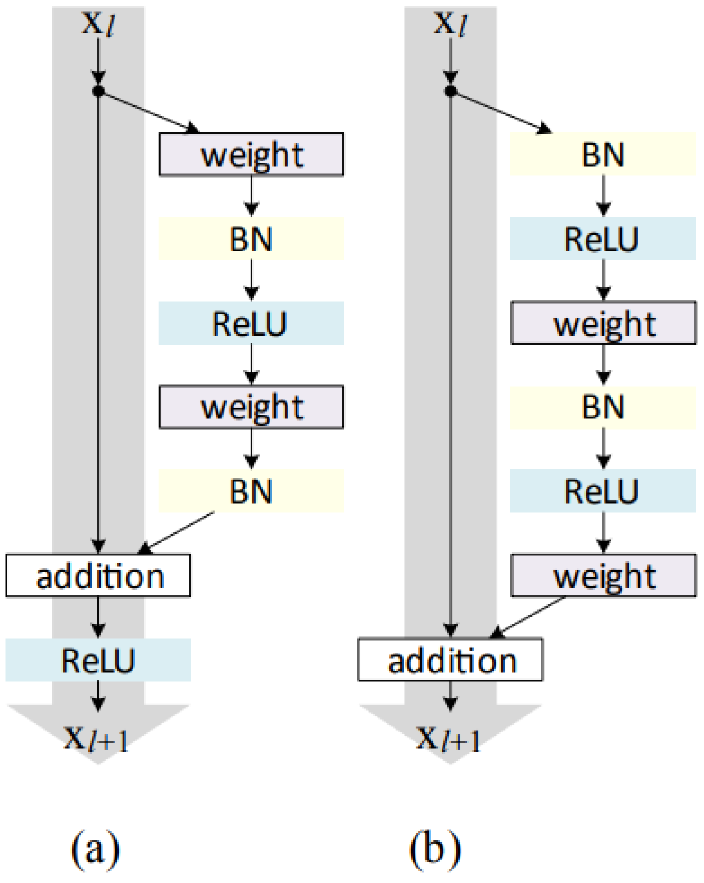


Figure 2.24: A residual learning building block with pre-activation. (a) Original design (b) Pre-activation design (HE et al., 2016c).

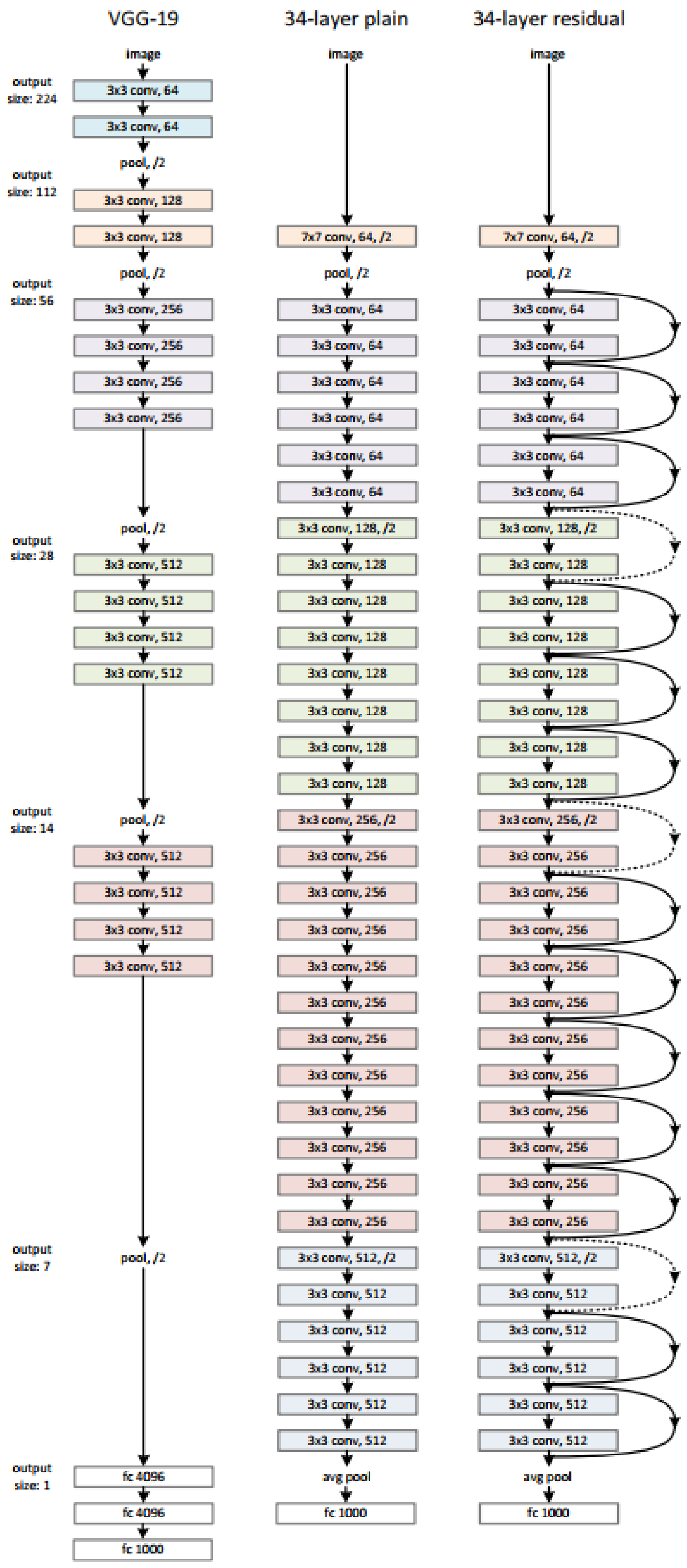


Figure 2.25: A comparison among a VGG-19, a 34-layers plain and a 34-layers residual architecture (HE et al., 2016a).

Despite ReLU allowed to train much more deeper models, after some dozens of layers, conventional deep networks deeper than this are not successfully trained (ZAGORUYKO; KOMODAKIS, 2017). In fact, this is the reason because we do not have a VGG-34, for example. The concept of skip connection introduced by the Residual Networks fixed this problem. In fact, this mechanism creates a bypass between the input and the output of each layer (or block of layers), which makes the backpropagation of the gradients much more effective thought the entire model allowing the training of networks much more deeps.

The skip connections developed in the context of Residual Networks inspired many other architectures such as Highway Networks (SRIVASTAVA; GREFF; SCHMIDHUBER, 2015) and DenseNets (HUANG et al., 2016) that explores this concept in different approaches. Nevertheless, some researchers argue that skip connections may not be the only or best approach to allow the training of deeper networks (ZAGORUYKO; KOMODAKIS, 2017).

The Figure 2.25 presents a visual comparison among a VGG-19, a 34-layers plain and 34-layer residual architecture. Currently, as of today, a 34-layers plain option produces low test accuracy because the gradient is not able to train the deepest layers. However, the skip connection of the residual equivalent model allows the training of the deepest layers and produces high accuracy. It should be noticed that two of the three last fully connected layers presented in VGG-19 are replaced by a one average pooling layer in the residual designs.

2.5 REGULARIZATION

Training differs from simple optimization in the sense that we are not searching for a minimum of some loss function specifically for the training data, but rather we are committed to generalize the performance of the model on unseen data. In this sense, regularization procedures are essential to optimize a given model to a particular data but simultaneously avoid overfitting, which is extremely important especially when dealing with deep models (BISHOP, 2006).

In fact, considering that essentially no handcrafted feature is used in deep learning, a huge number of parameters is needed to avoid bias. However, this fact makes deep models particularly vulnerable to variance problems. Therefore, regularization is a key point in training deep networks. In deep learning, the two most commonly used regularization methods are dropout and batch normalization.

2.5.1 Dropout

Dropout is a technique that avoids overfitting by randomly excluding some nodes from the deep models during training (Fig. 2.26). The main objective is to prevent co-adapting. Since during the test all nodes are used, this in some sense resembles an ensemble of classifiers (SRIVASTAVA et al., 2014).

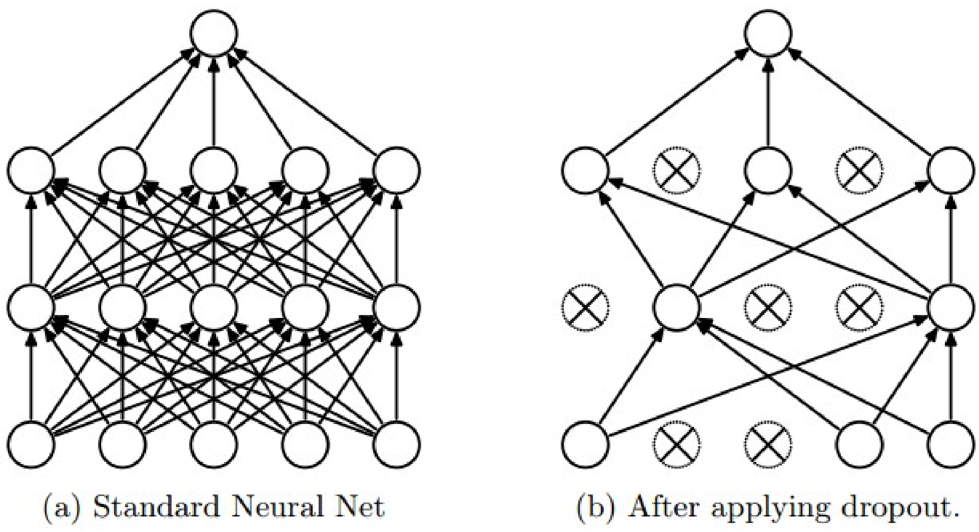


Figure 2.26: Example of dropout technique (SRIVASTAVA et al., 2014).

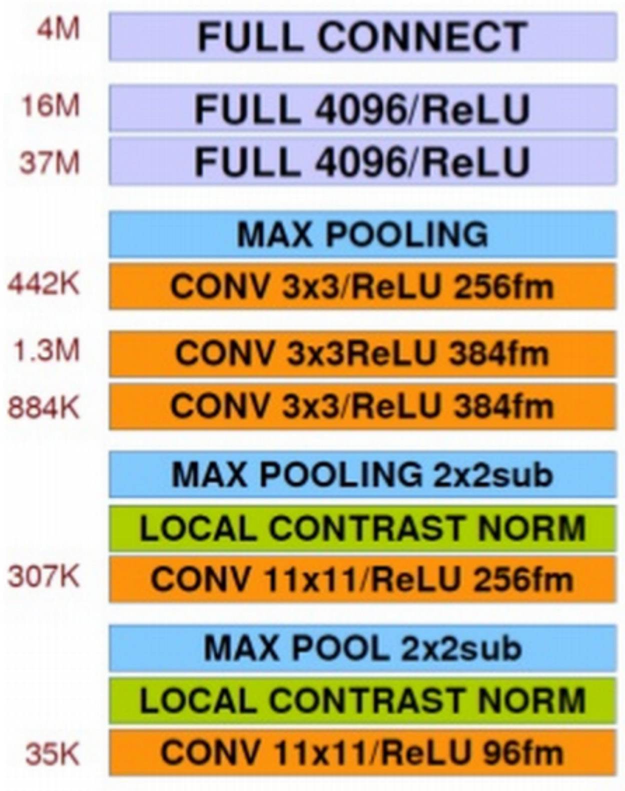


Figure 2.27: Number of parameters in each layer of AlexNet <https://sites.google.com/site/deeplearningcvpr2014/>.

Convolutional layers have typically much fewer parameters than fully connected ones (Fig. 2.27). Moreover, they are much less prone to overfitting due to the selective fields and weight sharing capability. Those are the reasons why dropout is essentially a technique restricted to use in fully connected layers and was very important to improve the AlexNet performance.

However, the fully connected layers are becoming increasingly less used. Moreover, batch normalization also works as an effective regularizer. For example, residual networks do not even use dropout anymore. Therefore, we believe dropout will be increasingly less used from now on (HE et al., 2016a; HE et al., 2016c; ZAGORUYKO; KOMODAKIS, 2017).

2.5.2 Batch Normalization

A major factor that hardens the training of deep network is the fact the activations of the previous layer are always changing. Moreover, in reality, even the probability distribution of activations of a given layer is continually modified during training.

To tackle the mentioned problem, the batch normalization normalizes the distribution of activation that is fed into the subsequent layer, which significantly improves training speed and test accuracy (IOFFE; SZEGEDY, 2015).

In machine learning, normalizing the distribution of the input data decreases the training time and improves test accuracy (TAX; DUIN, 2002). Consequently, normalization also improves neural networks performance (LECUN et al., 2012). A standard approach to normalizing input data distributions is the mean-standard technique. In this method, the input data is transformed to present zero mean and standard deviation of one.

However, if instead of working with shallow machine learning models, we are dealing with deep architectures; the problem becomes more sophisticated. Indeed, in a deep structure, the output of a layer works as input data to the next. Therefore, in this sense, each layer of a deep model has his own "input data" that is composed of the previous layer output or activations. The only exception is the first layer, for which the input is the original data.

In fact, consider that the stochastic gradient decent (SGD) is being used to optimize the parameters $\boldsymbol{\theta}$ of a deep model. Assume S a sample of m training examples in a mini-batch; then the SGD minimizes the loss given by the equation:

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}} \frac{1}{m} \sum_{S} L(S, \boldsymbol{\Theta}) \tag{2.9}$$

Furthermore, consider $\boldsymbol{x}$ the output of the layer $i-1$. These activations are fed into layer i as inputs. In turn, the outputs of layer i are fed into layer $i+1$ producing the overall loss given by the equation bellow:

$$L = G(F_{i+1}(F_i(\boldsymbol{x}, \boldsymbol{\Theta}_i), \boldsymbol{\Theta}_{i+1})) \tag{2.10}$$

In the previous equation, F_{i+1} and F_i are the transformation produced by the layers $i+1$ and i, respectively. The G function represents the mapping perpetrated by the above layers combined with the loss function adopted as the criterion. Considering that the output of layer i is given by $\boldsymbol{y} = F_i(\boldsymbol{x}, \boldsymbol{\Theta}_i)$, we can rewrite the above equation as follows:

$$L = G(F_{i+1}(\boldsymbol{y}, \boldsymbol{\Theta}_{i+1})) \tag{2.11}$$

Applying equation (2.9) to equation (2.11) and considering a learning rate λ, we can write the equation to update the parameters of the layer $i+1$ as the follows:

$$\boldsymbol{\Theta}_{i+1} \leftarrow \boldsymbol{\Theta}_{i+1} - \frac{\lambda}{m} \sum_{\boldsymbol{y}} \frac{\partial G(F_{i+1}(\boldsymbol{y}, \boldsymbol{\Theta}_{i+1}))}{\partial \boldsymbol{\Theta}_{i+1}} \tag{2.12}$$

In fact, the previous equation is mathematically equivalent to training the layer $i+1$ on the input data given by y, which in turn is the output of the previous layer. Therefore, indeed, we can see the output of the previous layer as "input data" for the effect of training the current layer using SGD optimization.

Considering each layer has its own "input data" (the output of the previous layer), normalizing only the actual input data of a deep neural network produces a limited effect in enhancing learning speed and test accuracy. Moreover, during the training process, the distribution of the input of each layer changes, which makes training even harder. Indeed, the parameters of a layer are updated while its input (the activations of the previous layer) is constantly modified.

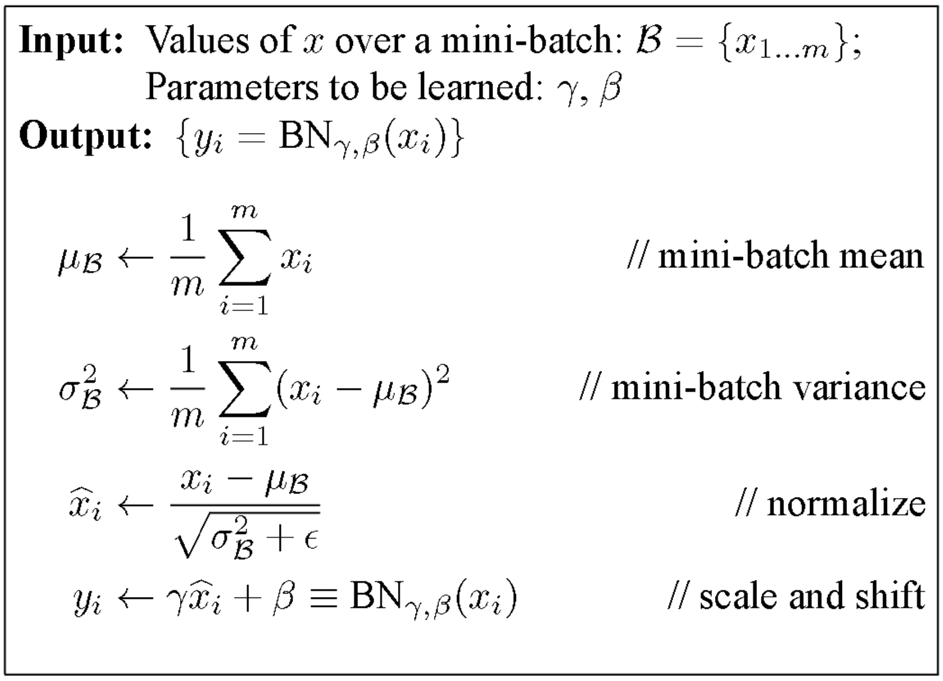
This phenomenon is called internal covariant shift, which is a major factor that hardens the training of deep architectures (IOFFE; SZEGEDY, 2015). In fact, while the data of shallow models is normalized and static during training, the input of a deep model layer, which is the output of the previous one, is neither a priori normalized nor static throughout training.

Batch normalization is an effective method to mitigate the internal covariant shift (IOFFE; SZEGEDY, 2015). This approach, which significantly improves training speed and test accuracy, proposes normalizing the inputs of the layers when training deep architectures.

The layers inputs normalization is performed after each mini-batch training to synchronizing with the deep network parameters update. Therefore, when using batch normalization, for a input $\boldsymbol{x} = (x^{(1)}, x^{(2)}, \ldots, x^{(d)})$, each individual dimension is transformed as follows:

$$\hat{x}^{(k)} = \frac{x^{(k)} - \hat{\mathbb{E}}[x^{(k)}]}{\sqrt{\widehat{\mathbb{V}\mathrm{ar}}[x^{(k)}]}} \tag{2.13}$$

The Figure 2.28 presents the batch normalization algorithm by showing the set of the mathematical operations performed on each feature dimensions over a given minibatch treatment.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.28: Batch normalization algorithm (IOFFE; SZEGEDY, 2015).

3 DISPLACED RECTIFIER LINEAR UNIT

All you have to do is decide what to do with the time that is given to you.
—GANDALF (The Lord of the Rings)

Put aside the Ranger. Become who you were born to be.
—ELROND (The Lord of the Rings: The Return of the King)

There can be no triumph without loss. No victory without suffering.
No freedom without sacrifice.
—OFFICIAL TRAILER (The Lord of the Rings: The Return of the King)

Aiming to the goals of this work, we concentrate our attention on the interaction between activation functions and batch normalization. We conjecture that nonlinearities that are more compatible with batch normalization present higher performance. Considering that an identity transformation preserves any statistical distribution, we assume that to extend the identity function from the first quadrant to the third implies less damage to the normalization procedure. Therefore, different from all previous proposed deep learning activation functions, DReLU does not present an inflection at the origin, but instead in the third quadrant.

In fact, there are two traditional design options to incorporating the batch normalization into a deep neural networks (Fig. 3.1). In the standard approach, the batch normalization layer is placed between a generic weights layer (fully-connected, convolutional, etc.) and the nonlinearity layer (Fig. 3.1a). Naturally, another similar block of layers follows, and the repetition of this procedure generates a stack of layers that characterizes a deep architecture.

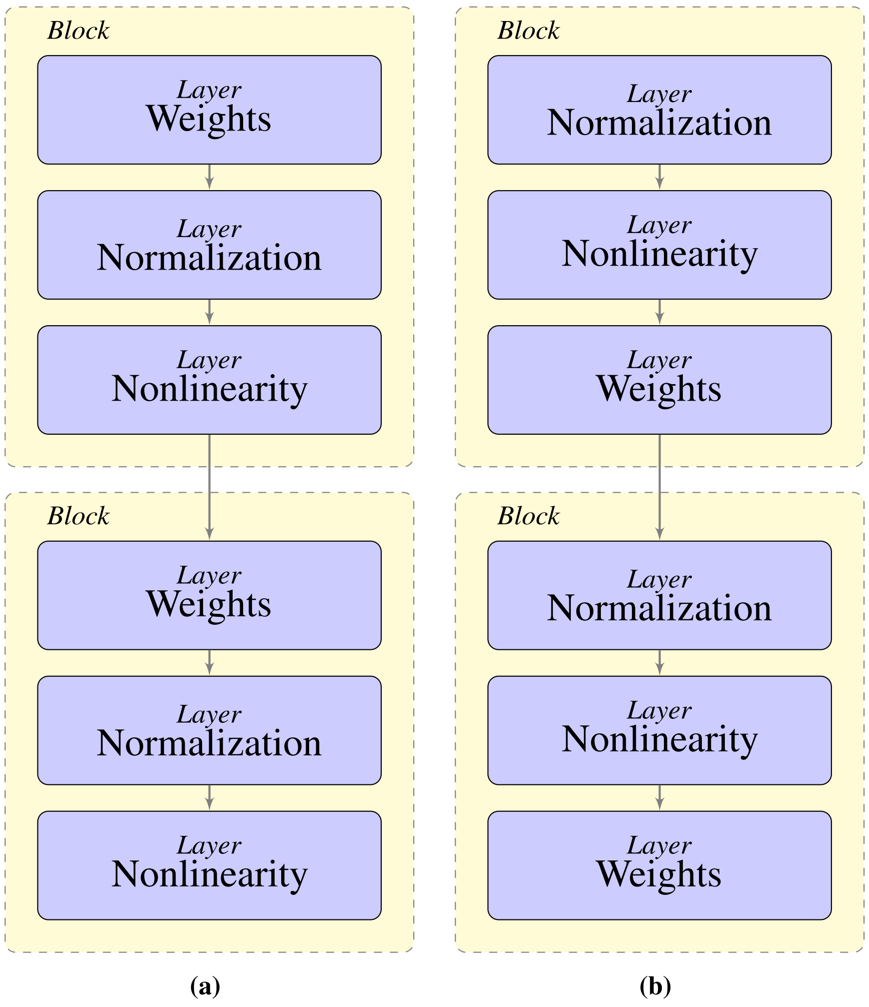


Figure 3.1: Building blocks of a deep neural network using batch normalization. (a) Standard design (IOFFE; SZEGEDY, 2015). (b) Pre-activation design (HE et al., 2016c).

In the pre-activation design, a generic weights layer (fully-connected, convolutional, etc.) is placed after a sequence of a normalization and a nonlinearity layers. This process is stacked to build a deep model (Fig. 3.1b). Regardless of the designed option chosen, after the batch normalization, the activations are fed into the nonlinearity (Fig. 3.1). Batch normalization is not usually performed after the nonlinearity (IOFFE; SZEGEDY, 2015).

In a standard design, consider $\boldsymbol{x}$ the input of a layer composed of a generic transformation $\boldsymbol{Wx}+\boldsymbol{b}$ followed by a nonlinearity, for instance, ReLU. After the addition of the batch normalization layer, the overall joint transformation performed by the block is given by:

$$\boldsymbol{z} = \mathrm{ReLU}(\mathrm{BN}(\boldsymbol{Wx}+\boldsymbol{b})) \tag{3.1}$$

For a moment, consider the intermediate activation y produced inside the block:

$$\boldsymbol{y} = \mathrm{BN}(\boldsymbol{Wx}+\boldsymbol{b}) \tag{3.2}$$

Without loss of generality, assume $\gamma = 1$ and $\beta = 0$ (IOFFE; SZEGEDY, 2015). Therefore, since $\boldsymbol{y}$ is the output of a batch normalization layer, we can write unbiased estimators for the expected value and variance of any given dimension k as follows:

$$\hat{\mathbb{E}}[\hat{y}^{(k)}] = 0, \widehat{\mathbb{V}\text{ar}}[\hat{y}^{(k)}] = 1 \tag{3.3}$$

Consequently, we investigate the expected value and variance of the activations distribution produced by the combined layer (Fig. 3.1a). Therefore, if $\boldsymbol{z}$ is the output of the block using a ReLU nonlinearity, it follows that:

$$\boldsymbol{z} = \text{ReLU}(\boldsymbol{y}) \tag{3.4}$$

The application of ReLU removes all negative values from a distribution. Hence, it necessarily produces positive values as output. Therefore, it can be written:

$$\hat{\mathbb{E}}[z^{(k)}] = \hat{\mathbb{E}}[\text{ReLU}(\hat{y}^{(k)})] > \hat{\mathbb{E}}[\hat{y}^{(k)}] = 0 \tag{3.5}$$

Considering that a distribution with $\hat{\mathbb{E}}[\hat{y}^{(k)}] = 0$ has to present negative values, replacing all negative activations with zeros makes the variance of $z^{(k)}$ necessarily lower than the variance of the original distribution $\hat{y}^{(k)}$. Consequently, we can write:

$$\widehat{\mathbb{V}\text{ar}}[z^{(k)}] = \widehat{\mathbb{V}\text{ar}}[\text{ReLU}(\hat{y}^{(k)})] < \widehat{\mathbb{V}\text{ar}}[\hat{y}^{(k)}] = 1 \tag{3.6}$$

Therefore, despite batch normalization, the activations after the whole block are not perfectly normalized since these outputs present neither zero mean nor unit variance. Consequently, regardless of the presence of a batch normalization layer, after the ReLU, the inputs passed to the next composed layer have neither mean of zero nor variance of one that was the objective in the first place (Fig. 3.1). The same conclusion can be deduced for the pre-activation design.

In this sense, ReLU skews an otherwise previous normalized output. In other words, ReLU reduces the correction of the internal covariance shift promoted by the batch normalization layer. Consequently, we conclude the ReLU bias shift effect (CLEVERT; UNTERTHINER; HOCHREITER, 2015) is related to the drawback ReLU generates to the batch normalization.

In this regard, we propose DReLU, which is essentially a diagonally displaced ReLU. It generalizes ReLU by allowing its inflection to move from happening exactly at the point $(0,0)$ to being placed at any point of the form $(-\delta, -\delta)$ (Fig. 3.2). Therefore, the slope zero component of the activation function provides negative activations, instead of null ones. Unlike ReLU, in DReLU learning can happen for negative inputs since gradient is not necessarily zero. The following equation defines DReLU:

$$y = \begin{cases} x & \text{if } x \geq -\delta \\ -\delta & \text{if } x < -\delta \end{cases} \tag{3.7}$$

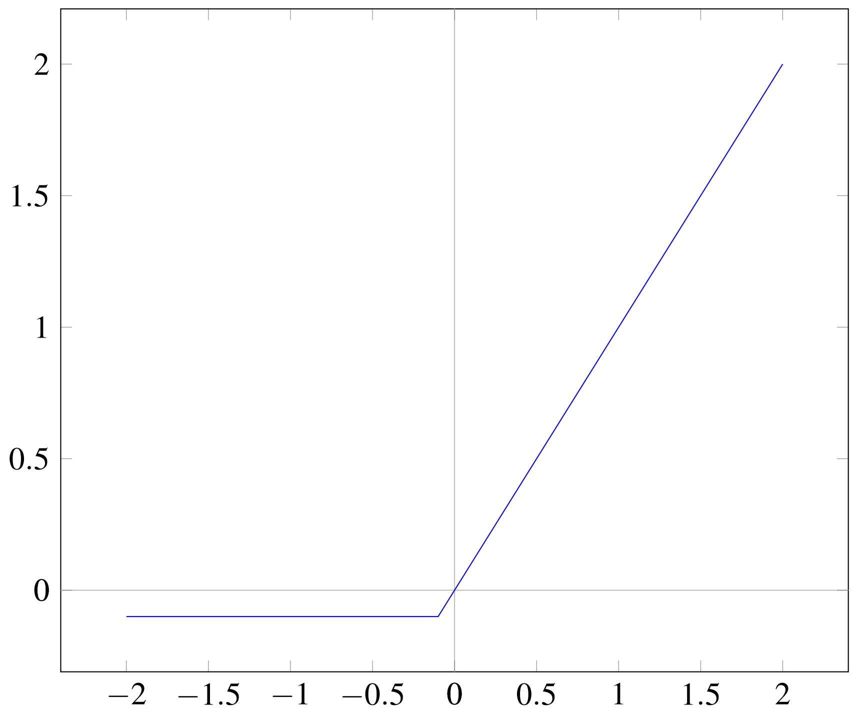


Figure 3.2: Plot of DReLU activation function.

Replacing ReLU by DReLU in Eq. 3.1, the activations of the composed layer become:

$$\boldsymbol{z} = \mathrm{DReLU}(\mathrm{BN}(\boldsymbol{Wx} + \boldsymbol{b})) \tag{3.8}$$

Since DReLU extends the identity function into the third quadrant, it is no longer possible to conclude Eq. 3.5 is valid. Therefore, the consequence presented in the mentioned equation is probably at least minimized. In this case, the $\hat{\mathbb{E}}[z^{(k)}]_{DReLU}$ is much probably near to zero than $\hat{\mathbb{E}}[z^{(k)}]_{ReLU}$. Hence, we can write:

$$\hat{\mathbb{E}}[z^{(k)}]_{DReLU} < \hat{\mathbb{E}}[z^{(k)}]_{ReLU} \tag{3.9}$$

Indeed, all of our experiments clearly showed that the identity mapping extension produced less damage to the normalization performed by the previous layer and in fact mitigated the bias shift effect when compared to ReLU. Moreover, in all experiment scenarios DReLU in fact outperformed ReLU in both training speed and test accuracy.

Furthermore, DReLU exhibits a noise-robust deactivation state for very negative inputs, a feature not granted by LReLU and PReLU. A noise-robust deactivation state is achieved by setting the slope zero for highly negative values of input (CLEVERT; UNTERTHINER; HOCHREITER, 2015). Some authors argument that activation functions with this propriety improve learning (CLEVERT; UNTERTHINER; HOCHREITER, 2015).

Finally, DReLU is less computationally complex than LReLU, PReLU, and ELU. In fact, since DReLU has the same shape of ReLU, it essentially has the same computational complexity. However, the LReLU and PReLU involves an additional multiplication while the ELU implies an exponential calculation for each activation of the deep network.

4 EXPERIMENTS

What is real? How do you define "real"? If you're talking about what you can feel, what you can smell, what you can taste and see, then "real" is simply electrical signals interpreted by your brain.

—MORPHEUS (The Matrix)

In our experiments, we investigate the activation function Displaced Rectifier Linear Unit (DReLU), which partially prolongs the identity function beyond origin. Hence, DReLU is essentially a ReLU diagonally displaced into the third quadrant. Therefore, different from all other previously proposed activation functions, the inflection of DReLU does not happen at the origin, but in the third quadrant. In this regard, we evaluated how replacing the activation function impacts the learning speed and test accuracy performance of well established and widely used standard state-of-the-art models. Considering the widespread adoption and practical importance, we adopted CNN (LECUN et al., 1998; KRIZHEVSKY; SUTSKEVER; HINTON, 2012) in our experiments.

Moreover, as particular examples of CNN architectures, we used the previous ImageNet Large Scale Visual Recognition Competition (ILSVRC) winners Visual Geometry Group (VGG) (SIMONYAN; ZISSERMAN, 2014) and Residual Networks (ResNets) (HE et al., 2016a; HE et al., 2016c). These architectures have distinctive designs and depth to promote generality to the conclusions of this work. Finally, we decided to employ the two most broadly used computer vision datasets by deep learning research community: CIFAR-10 (KRIZHEVSKY, 2009) and CIFAR-100 (KRIZHEVSKY, 2009).

At least ten executions of each of experiment were executed. However, in cases a significant level was not achieved in such a case, ten additional runs were performed. In this systematic comparative study, performance assessments were carried out using statistical tests with a significance level of 5%. The experiments were conducted without using dropout (SRIVASTAVA et al., 2014) since recent studies show that, despite improving the training time, dropout provides unclear contributions to the overall deep model performance (IOFFE; SZEGEDY, 2015). Moreover, dropout has recently become a technique restricted to fully-connected layers, which in turn are being less used and replaced by spatial average layers in modern architectures (HE et al., 2016c). Therefore, considering fully connected layers, which are the most propense to overfit, are currently rarely used in modern CNN, the usage of dropout is accordingly becoming unusual.

The experiments were executed on a machine configured with an Intel(R) Core(TM) i7-4790K CPU, 16 GB RAM, 2 TB HD and a GeForce GTX 980Ti card. The operational system was Ubuntu 14.04 LTS with CUDA 7.5, cuDNN 5.0, and Torch 7 deep learning library.

4.1 DATASETS, PREPROCESSING AND DATA AUGMENTATION

We trained the models on the CIFAR-10 and CIFAR-100 datasets. The CIFAR-10 dataset (Fig. 4.1) possesses ten classes containing 6000 images, from which 5000 are used for training, and 1000 are left for testing. Each training example is an RGB image of size 32x32. The CIFAR-100 (Fig. 4.2) image dataset aggregates 100 classes, which in turn contain 600 example images each. From these, 500 are used for training, and 100 are employed for testing. Again, each example is a 32x32 RGB image.

The experiments used regular mean-standard preprocessing. Therefore, each feature was normalized to present zero mean and unit standard deviation throughout that training data. The features were also redimensioned using the same parameters before performing the inference during test.

The data augmentation performed was randomized horizontal flips and random crops. Therefore, before training, each image was flipped horizontally with a 0.5 probability. Moreover, four pixels reflected from the picture opposite sides were added to expand it vertically and horizontally. Finally, a 32x32 random crop was taken from the enlarged image. The random crop was then used to train.

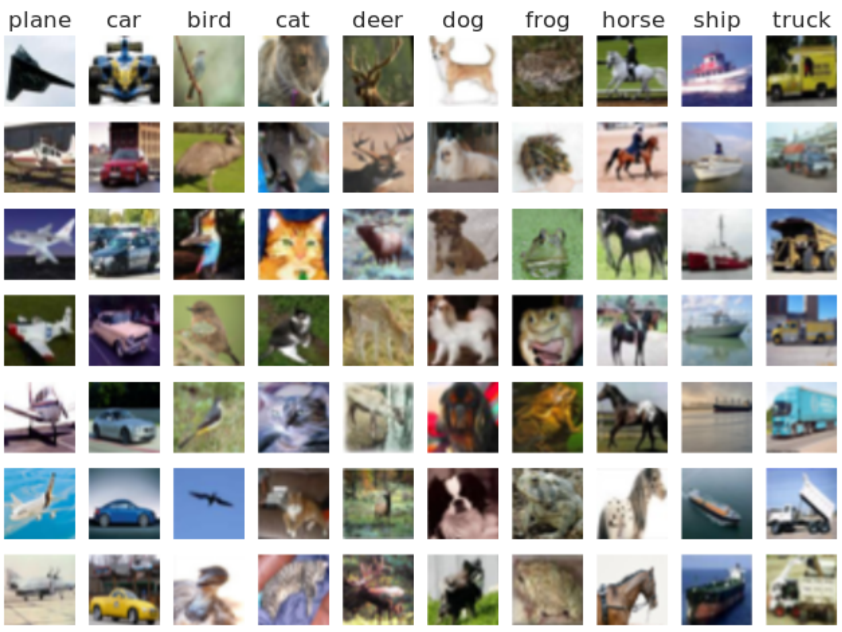


Figure 4.1: Random sampling of CIFAR-10 color images.

Figure 4.2: Random sampling of CIFAR-100 color images.

4.2 ACTIVATION FUNCTIONS PARAMETRIZATION

In this work, we used the parameters originally proposed by the activation function designers (HE et al., 2016b; CLEVERT; UNTERTHINER; HOCHREITER, 2015) since these have been kept in subsequent papers (SHAH et al., 2016; CLEVERT; UNTERTHINER; HOCHREITER, 2015). Therefore, we kept $\beta = 0.25$ for both LReLU and PReLU, and for ELU we maintained $\alpha = 1.0$. We decided to keep those parameters because we consider the original authors and the follower papers, which respectively proposed and kept the original parameterizations, performed hyperparameter search and validation procedures to estimate parameter values that provide high performance for their proposed or used work.

In case of DReLU, δ must not go to infinity because we would lose the nonlinearity in such a case. Therefore, a compromise that causes less damage to the normalization while keeps the activation function nonlinearity has to be achieved. To define the parameter value of DReLU, we performed hyperparameter validation experiments (Table 4.1).

The models were trained with deferent δ values to choose its value. The results are the mean values of each experiment five executions. Based on these experimental results, we decided to set $\delta = 0.05$ since it presented the best mean of means.

Table 4.1: Hyperparameter Validation Experiments
Displaced Rectifier Linear Unit CIFAR-10 Test Accuracy

δ	VGG-19 (%)	ResNet-56 (%)	ResNet-110 (%)	Mean (%)
0.01	93.89±0.20	93.45±0.13	93.98±0.19	93.77
0.05	93.91±0.18	**93.50±0.07**	**94.10±0.10**	**93.83**
0.10	**94.00±0.13**	93.43±0.25	93.93±0.14	93.78
0.25	93.67±0.19	92.93±0.19	93.31±0.12	93.30
0.50	92.70±0.08	91.62±0.37	92.04±0.17	92.12

4.3 MODELS AND INITIALIZATION

As a particular instance of a VGG model, we used the VGG variant with nineteen layers (VGG-19). To train models with a considerably different number of layers, we chose the pre-activation ResNet with fifty-six layers (ResNet-56) and also the pre-activation ResNet with one hundred ten layers (ResNet-110). We employed pre-activation ResNets because they present better performance when compared to the original aproach (HE et al., 2016a). The experiments used the Kaiming initialization (HE et al., 2016b).

4.4 TRAINING AND REGULARIZATION

The experiments used an initial learning rate of 0.1, and a learning rate decay of 0.2 with steps in the epochs 60, 80 and 90 for both CIFAR-10 and CIFAR-100 datasets. At epochs 40 and 70, we evaluated the test accuracy of the partially trained models. After the final epoch 100, we performed the last test accuracy. Therefore, we were able to assess how fast each model was learning based on the activation function used by the model.

The experiments employed mini-batches of size 128 and stochastic gradient descent with Nesterov acceleration technique as the optimization method. The moment was set to 0.9, and the weight decay was equal to 0.0005. Batch normalization was used, but not dropout.

4.5 PERFORMANCE ASSESSMENT

We define an experiment as the training of a deep model using a distinct activation function on a given dataset. If not otherwise mentioned, we conducted ten executions of each experiment. We define a scenario as the set of experiments regarding all activation functions on a specific dataset using a particular model.

In this regard, this work presents the consolidated results of six scenarios (two datasets versus three models) that correspond to 30 experiments, which in turn represents a total of 320 executions (training of deep neural networks). In two cases, we executed 20 instead of 10 runs of a given experiment of ReLU and DReLU.

To make assessments about the performance of activation functions, we chose the Kruskal-Wallis one-way analysis of variance statistical tests (KRUSKAL; WALLIS, 1952) because the weight initialization was different for each experiment execution. Consequently, we had independent and also possibly different size samples for the test accuracy distributions.

Moreover, the Kruskal-Wallis tests can also be used to confront samples obtained from more than two sources at once, which is appropriated in our study since, for a given scenario, we are comparing five activation functions simultaneously. Besides, it does not assume a normal distribution of the residuals, which makes it more general than the parametric equivalent one-way analysis of variance (ANOVA). We used the Conover-Iman post-hoc tests (CONOVER; IMAN, 1979) for pairwise multiple comparisons.

5 RESULTS

I've seen things you people wouldn't believe. Attack ships on fire off the shoulder of Orion. I watched C-beams glitter in the dark near the Tannhäuser Gate. All those moments will be lost in time, like tears in rain. Time to die.

—ROY BATTY (Blade Runner)

Never send a human to do a machine's job.

—AGENT SMITH (The Matrix)

In the following subsections, we analyze the tested scenarios. In each case, we first discuss the activation functions learning speed based on test accuracy obtained for the partially trained models. We consider that an activation function presented better learning speed if the corresponding model achieved higher test accuracy on the partially trained models since the training time of an epoch shows no significant difference among the activation functions.

Subsequently, we comment about the test accuracy performances of the activation functions, which corresponds to the respective model test accuracy evaluated after 100 epochs. Naturally, we consider that an activation function presents better test accuracy if it showed the higher test accuracy for the final trained models on a particular dataset.

In all scenarios, the null hypotheses were the test accuracy samples taken from different activation functions originated from the same distribution. In other works, all the compared activation functions have the same test accuracy performance in the particular scenario.

Table 5.1: Kruskal-Wallis Test Results

	CIFAR-10			CIFAR-100		
Score	VGG-19	ResNet-56	ResNet-110	VGG-19	ResNet-56	ResNet-110
$\tilde{\chi}^2(4)$	56.087	40.115	49.451	44.918	41.253	41.169
p-value	1.922×10^{-11}	4.097×10^{-8}	4.7×10^{-10}	4.135×10^{-9}	2.383×10^{-8}	2.48×10^{-8}

The null hypotheses were rejected for all scenarios (Table 5.1), which means that with statistical significance ($p < 0.05$) at least one of the activation functions presents a test accuracy performance that is different from the others activation functions. Therefore, we used the Conover-Iman post-hoc tests for pairwise multiple comparisons for all combination of datasets and models (Tables 5.4, 5.5, 5.6, 5.7, 5.8, 5.9). In these tables, the best results and p-values of the comparison of DReLU to other activation functions are in bold format.

5.1 BIAS SHIFT EFFECT

The Table 5.2 presents the averaged nonlinearities layers mean activations performed in the CIFAR-10 training dataset. It shows again that DReLU is more efficient to reduce the bias shift effect during training than ReLU.

The Table 5.3 presents the mean of the nonlinearities layers mean activations performed in the CIFAR-100 training dataset. It shows that DReLU is more capable of reducing the bias shift effect during training than ReLU.

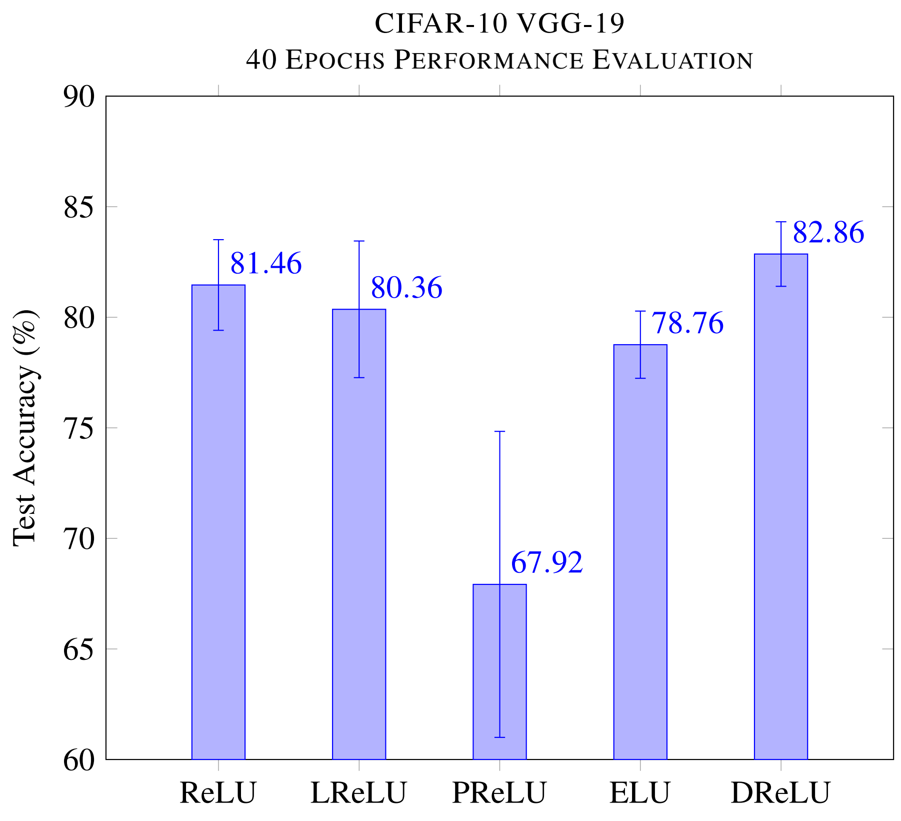
5.2 CIFAR-10 DATASET

5.2.1 VGG-19 Model

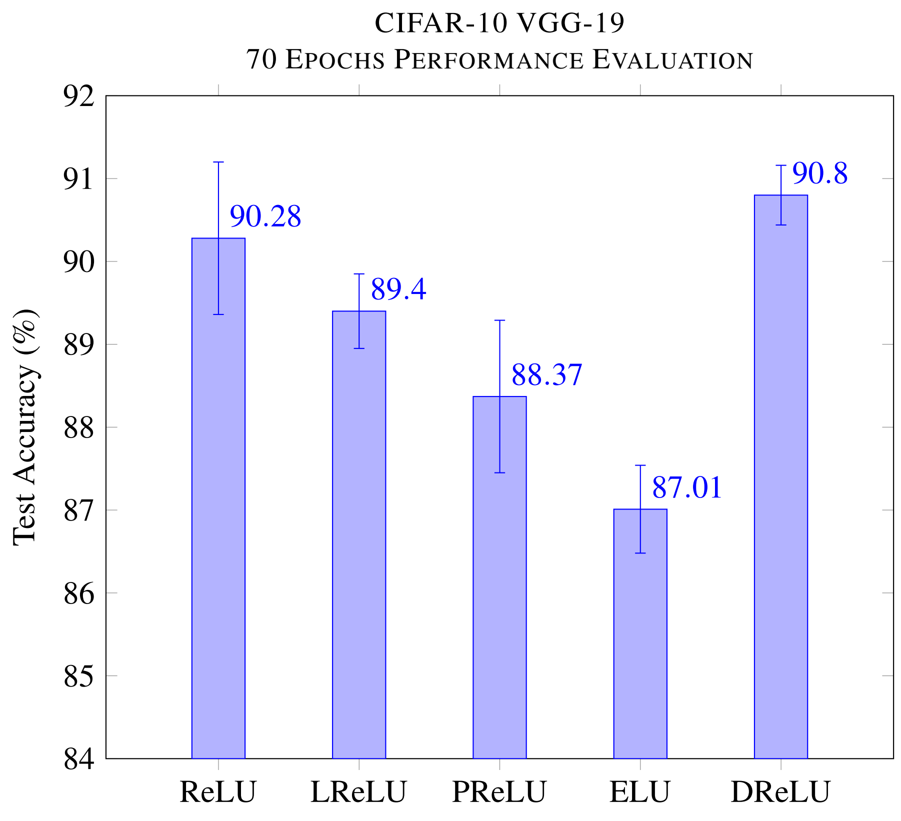
In this scenarios, DReLU provided faster learning than any other activation function for either 40 and 70 epochs (Fig. 5.1). The second fastest activation function was the ReLU in either 40 and 70 epochs evaluated.

The results also showed that LReLU performed better than PReLU in both cases. The slowest activation function for 40 epochs was PReLU, while ELU was the slowest after 70 training epochs case.

Moreover, DReLU presented the best test accuracy performance (Table 5.4). We performed 20 executions of either DReLU and ReLU experiment. ReLU provided the second best test accuracy. LReLU performed slightly better than PReLU in test accuracy. The activation function ELU presented the worst performance almost two percentage points behind PReLU.



(a)



(b)

Figure 5.1: VGG-19 model test accuracy means and standard deviations on the CIFAR-10 dataset. (a) 40 trained epochs. (b) 70 trained epochs.

Table 5.2: CIFAR-10 Averaged Nonlinearities Layers Mean Activations

	40 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0630 ± 0.0009	0.1006 ± 0.0027	0.0669 ± 0.0009
DReLU	0.0293 ± 0.0010	0.0671 ± 0.0016	0.0376 ± 0.0011

	70 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0449 ± 0.0006	0.0783 ± 0.0014	0.0515 ± 0.0006
DReLU	0.0139 ± 0.0003	0.0471 ± 0.0011	0.0244 ± 0.0009

	100 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0364 ± 0.0004	0.0673 ± 0.0011	0.0435 ± 0.0005
DReLU	0.0066 ± 0.0002	0.0351 ± 0.0008	0.0161 ± 0.0007

5.2.2 ResNet-56 Model

DReLU also provided faster learning than any other activation function on either 40 and 70 epochs in this scenario (Fig. 5.2). Again, ReLU was the second fastest activation function considering the 40 and 70 epochs experiments performed.

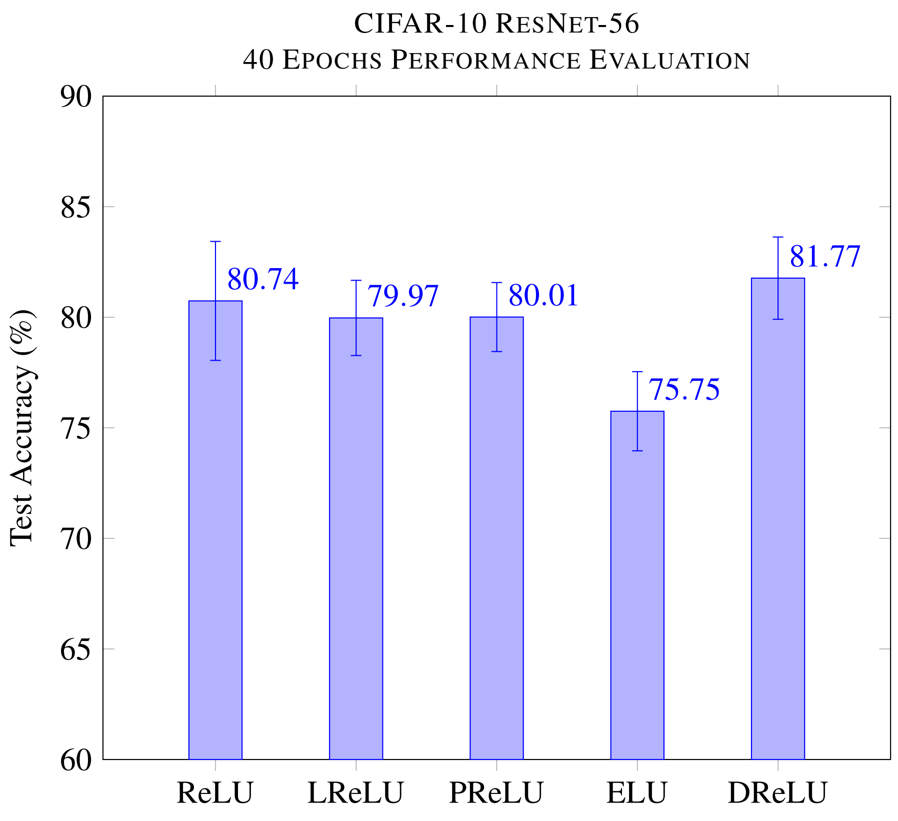
The training speed presented by LReLU and PReLU was remarkably similar, mainly in the 40 epochs case. ELU presented the worst training speed results on both 40 and 70 epochs evaluated situations.

Furthermore, DReLU was again the most accurate option followed by ReLU (Table 5.5). Once more, ReLU presented the second best test accuracy performance among all the analyzed activation functions.

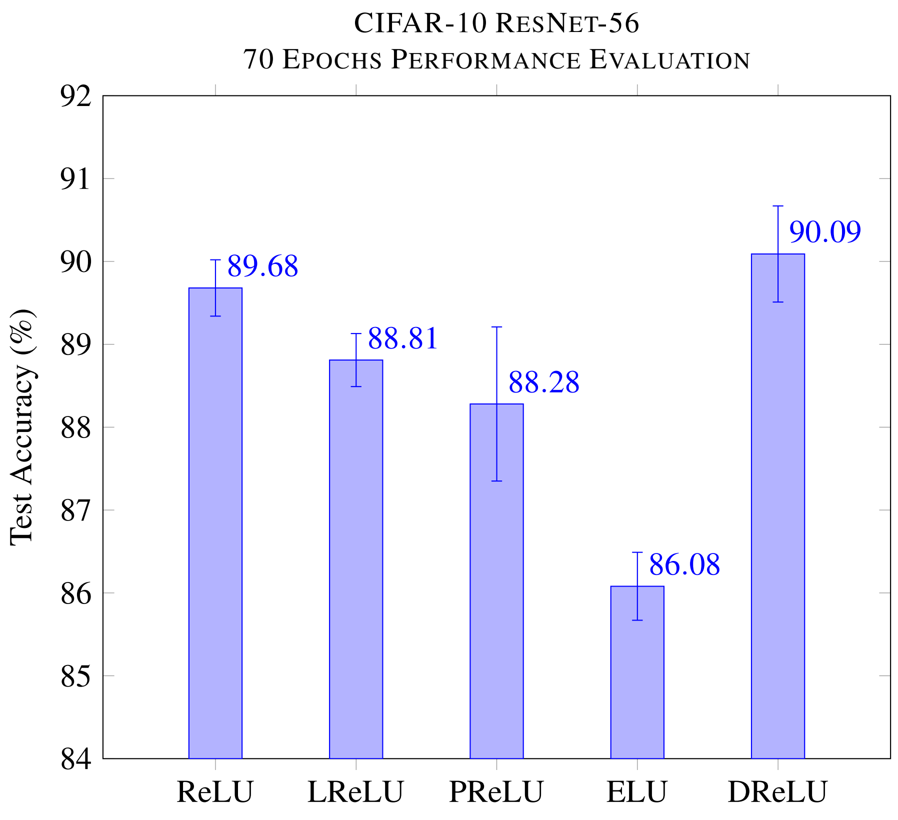
For this case, the final test accuracy performances of LReLU and PReLU were equal while ELU clearly presented the worst performance by a considerable margin of about three percentage points.

5.2.3 ResNet-110 Model

DReLU provided the fastest leaning on either 40 and 70 epochs once more (Fig. 5.3). The second fastest place was essentially a tie between the ReLU, LReLU, and PReLU in the 40 epochs, but ReLU was the clear second fastest after 70 epochs.



(a)



(b)

Figure 5.2: ResNet-56 model test accuracy means and standard deviations on the CIFAR-10 dataset. (a) 40 trained epochs. (b) 70 trained epochs.

Table 5.3: CIFAR-100 Averaged Nonlinearities Layers Mean Activations

	40 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0950 ± 0.0016	0.1288 ± 0.0016	0.0852 ± 0.0020
DReLU	0.0578 ± 0.0010	0.0972 ± 0.0030	0.0551 ± 0.0012

	70 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0744 ± 0.0008	0.1058 ± 0.0015	0.0680 ± 0.0015
DReLU	0.0362 ± 0.0009	0.0743 ± 0.0024	0.0390 ± 0.0008

	100 Epochs Performance Evaluation		
Nonlinearity	VGG-19	ResNet-56	ResNet-110
ReLU	0.0640 ± 0.0006	0.0943 ± 0.0013	0.0594 ± 0.0013
DReLU	0.0261 ± 0.0007	0.0618 ± 0.0023	0.0300 ± 0.0007

ELU presented the worst training speed for the 40 epochs. However, for 70 epochs, the training performances of LReLU, PReLU, and ELU were similar.

DReLU was the most accurate solution also for this scenario (Table 5.6). In this case, we also performed 20 runs of DReLU and ReLU, which presented the second highest test accuracy performance yet again. The results also show that the test accuracy performances of LReLU and PReLU were very similar and once more the ELU performed worse than any other activation function analyzed.

5.3 CIFAR-100 DATASET

5.3.1 VGG-19 Model

The experiments showed DReLU outperformed the test accuracy results of ReLU and all other assessed activation functions on either 40 and 70 epochs. In this sense, the results demonstrated that DReLU produced the fastest training (Fig. 5.4).

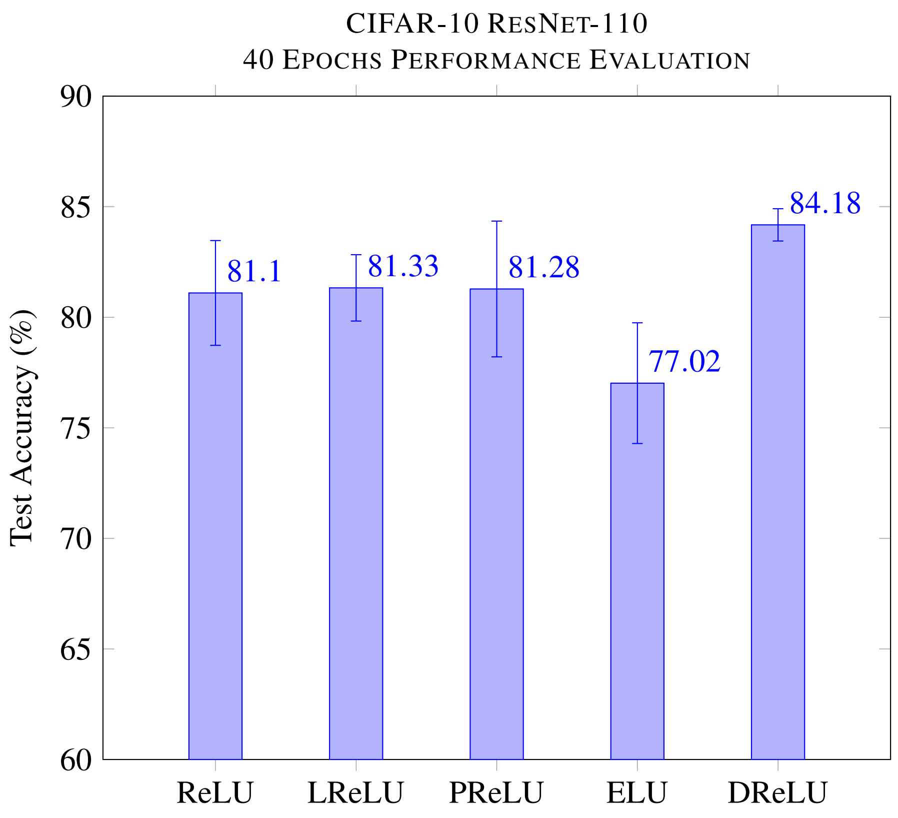
The 40 epochs results showed LReLU as the second fastest training activation function with ReLU in third place. In the 70 epochs case, ReLU performed the second best result while LReLU obtained the third place.

Table 5.4: CIFAR-10 VGG-19 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests *p*-values

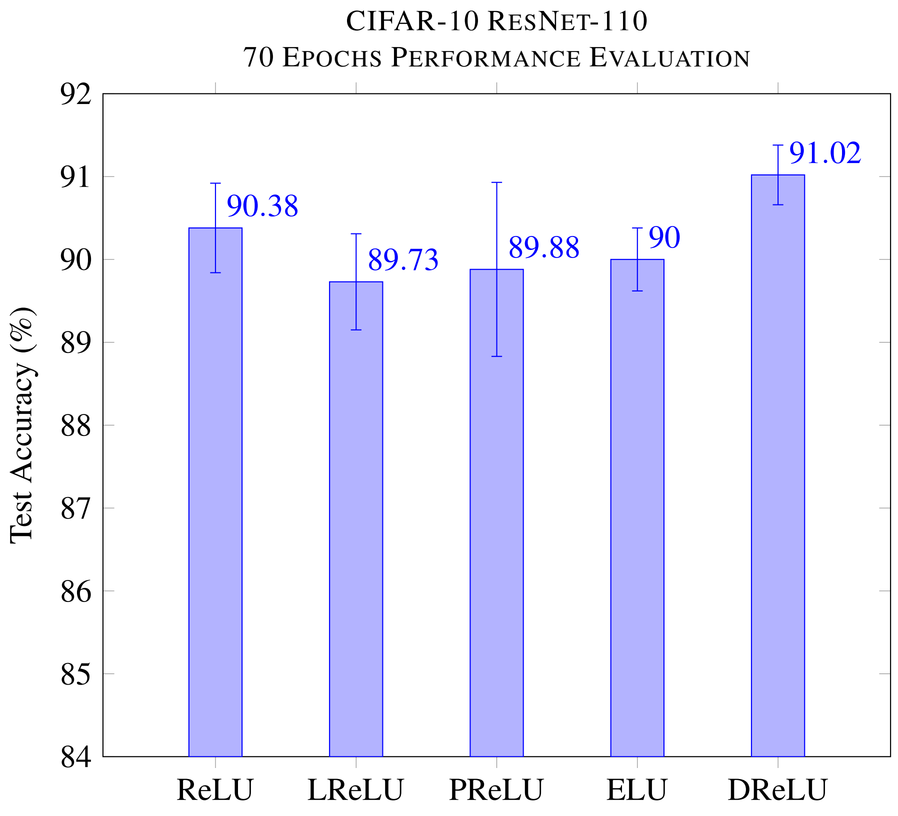
Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	93.82 ± 0.13	-	-	-	-
LReLU	93.35 ± 0.18	$p < 1.4 \times 10^{-9}$	-	-	-
PReLU	93.27 ± 0.22	$p < 2.2 \times 10^{-10}$	$p < 0.69449$	-	-
ELU	91.40 ± 0.15	$p < 2 \times 10^{-16}$	$p < 0.00022$	$p < 0.00081$	-
DReLU	$\mathbf{93.92 \pm 0.11}$	$\boldsymbol{p < 0.00330}$	$\boldsymbol{p < 5.6 \times 10^{-14}}$	$\boldsymbol{p < 9.1 \times 10^{-15}}$	$\boldsymbol{p < 2 \times 10^{-16}}$

Table 5.5: CIFAR-10 ResNet-56 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests *p*-values

Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	93.29 ± 0.21	-	-	-	-
LReLU	93.01 ± 0.21	$p < 1.2 \times 10^{-6}$	-	-	-
PReLU	93.01 ± 0.10	$p < 4.7 \times 10^{-5}$	$p < 0.9726$	-	-
ELU	90.21 ± 0.11	$p < 4.0 \times 10^{-13}$	$p < 1.2 \times 10^{-6}$	$p < 1.3 \times 10^{-6}$	-
DReLU	$\mathbf{93.52 \pm 0.14}$	$\boldsymbol{p < 0.0027}$	$\boldsymbol{p < 1.1 \times 10^{-9}}$	$\boldsymbol{p < 1.0 \times 10^{-9}}$	$\boldsymbol{p < 2 \times 10^{-16}}$



(a)



(b)

Figure 5.3: ResNet-110 model test accuracy means and standard deviations on the CIFAR-10 dataset. (a) 40 trained epochs. (b) 70 trained epochs.

Table 5.6: CIFAR-10 ResNet-110 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests p-values

Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	94.00 ± 0.18	-	-	-	-
LReLU	93.56 ± 0.26	$p < 1.6 \times 10^{-7}$	-	-	-
PReLU	93.74 ± 0.09	$p < 2.3 \times 10^{-5}$	$p < 0.25754$	-	-
ELU	90.76 ± 0.29	$p < 1.8 \times 10^{-13}$	$p < 0.00473$	$p < 0.00013$	-
DReLU	$\mathbf{94.11 \pm 0.18}$	$\boldsymbol{p < 0.03614}$	$\boldsymbol{p < 1.6 \times 10^{-7}}$	$\boldsymbol{p < 2.3 \times 10^{-5}}$	$\boldsymbol{p < 1.8 \times 10^{-13}}$

Table 5.7: CIFAR-100 VGG-19 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests p-values

Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	73.40 ± 0.22	-	-	-	-
LReLU	73.01 ± 0.22	$p < 2.2 \times 10^{-5}$	-	-	-
PReLU	71.55 ± 0.24	$p < 1.7 \times 10^{-13}$	$p < 1.1 \times 10^{-6}$	-	-
ELU	67.57 ± 0.17	$p < 2 \times 10^{-16}$	$p < 5.6 \times 10^{-14}$	$p < 6.7 \times 10^{-6}$	-
DReLU	$\mathbf{73.69 \pm 0.21}$	$\boldsymbol{p < 0.00012}$	$\boldsymbol{p < 1.5 \times 10^{-11}}$	$\boldsymbol{p < 2 \times 10^{-16}}$	$\boldsymbol{p < 2 \times 10^{-16}}$

The PReLU and ELU alternate the worst training speed results in this scenario. Therefore, while PReLU presented the slowest training speed for 40 epochs, ELU present the worst result after 70 training epochs.

Moreover, DReLU presented the best test performance (Table 5.7). ReLU presented the second best test accuracy performance. LReLU produced the third best result, followed by PReLU. The worst performance was shown yet again by ELU.

5.3.2 ResNet-56 Model

DReLU overcame the test accuracy results of the other activation functions on either 40 and 70 epochs once again. Therefore, we concluded DReLU generated the fastest learning (Fig. 5.5). The ReLU and LReLU alternate the second and third places in the 40 and 70 epochs.

PReLU and ELU presented similar low training speed results for the 40 epochs case. Regarding the 70 epochs case, ELU was the slowest activation function.

Regarding test accuracy, DReLU outperformed ReLU and all other options, with exception to LReLU. Although, no statistical significance was achieved in this particular pairwise comparison (Table 5.8). ReLU presented the third best test accuracy, followed by PReLU and ELU, which again presented a low accuracy result.

5.3.3 ResNet-110 Model

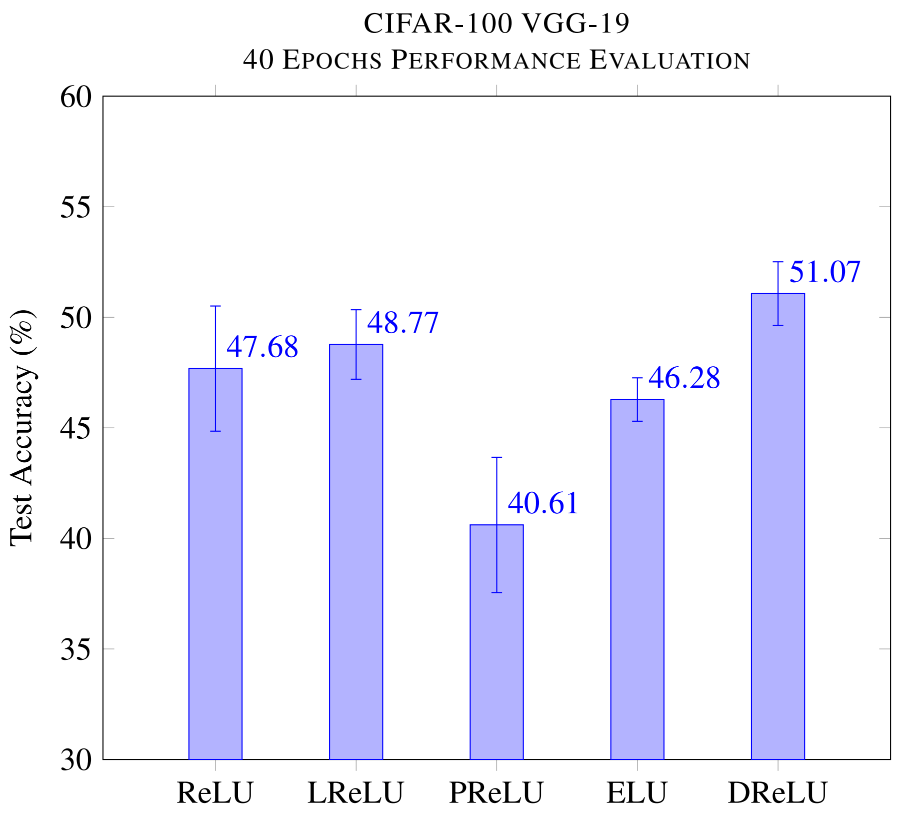
DReLU also provided the fastest learning in both 40 and 70 cases (Fig. 5.6). The second place for the fastest activation function in the 40 epochs was essentially a tie between ReLU and LReLU. However, ReLU was the clear second place for the 70 epochs case.

PReLu was the slowest activation function in the 40 epochs case. However, after 70 epochs, the ELU presented the worst result, which was approximately four percent behind the second worst presented by PReLU.

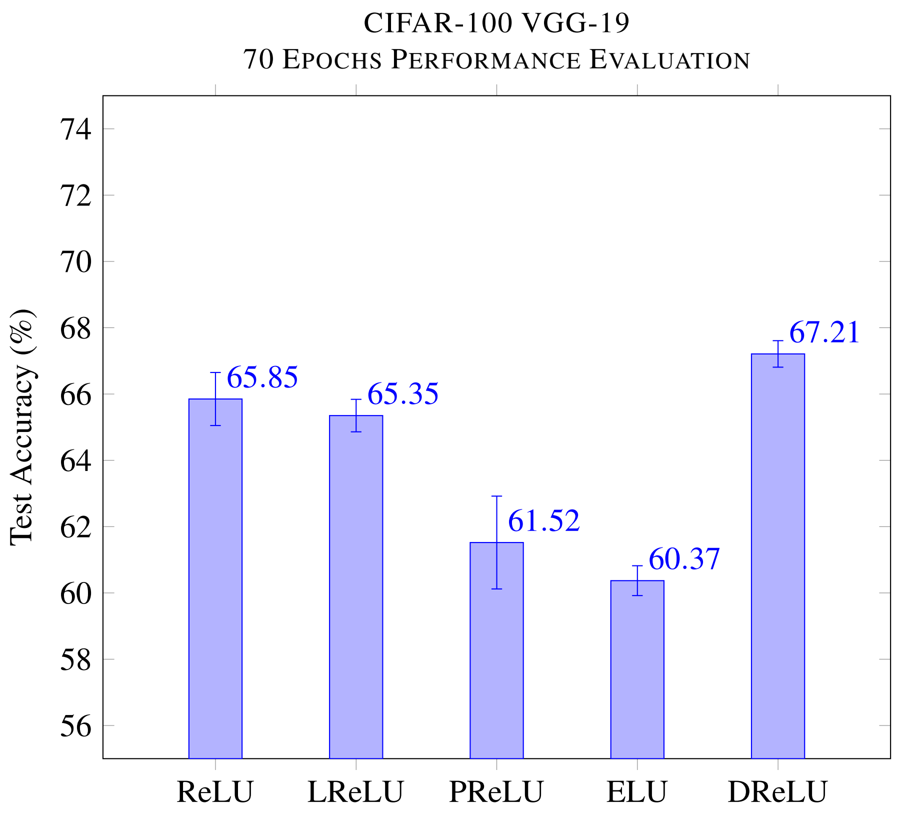
Finally, DReLU test accuracy outperformed ReLU and all others activation functions once again (Table 5.9). LReLU presented the second best test accuracy performance followed by ReLU in third place. ELU yet again presented the worst test accuracy result, far behind the fourth place obtained by the activation function PReLU.

5.4 DISCUSSION

In the CIFAR-10 dataset, DReLU presented the best learning speed for all considered models. Moreover, the results showed DReLU surpassed ReLU test accuracy in all analyzed scenarios. Actually, DReLU was the most accurate activation function in all evaluated models.

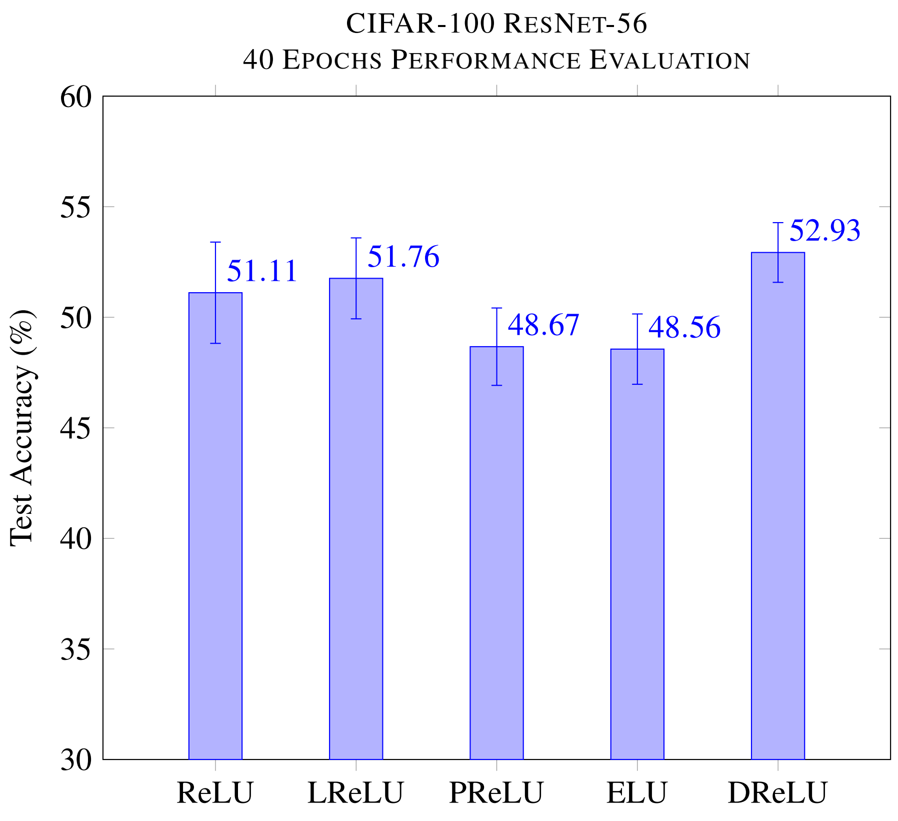


(a)

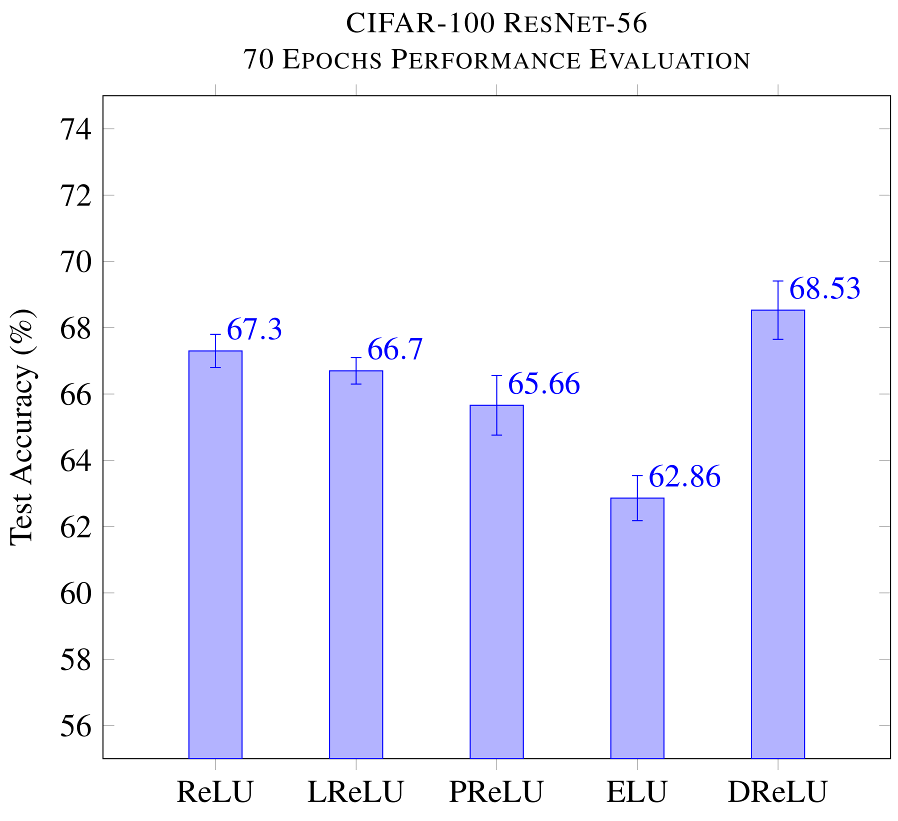


(b)

Figure 5.4: VGG-19 model test accuracy means and standard deviations on the CIFAR-100 dataset. (a) 40 trained epochs. (b) 70 trained epochs.



(a)



(b)

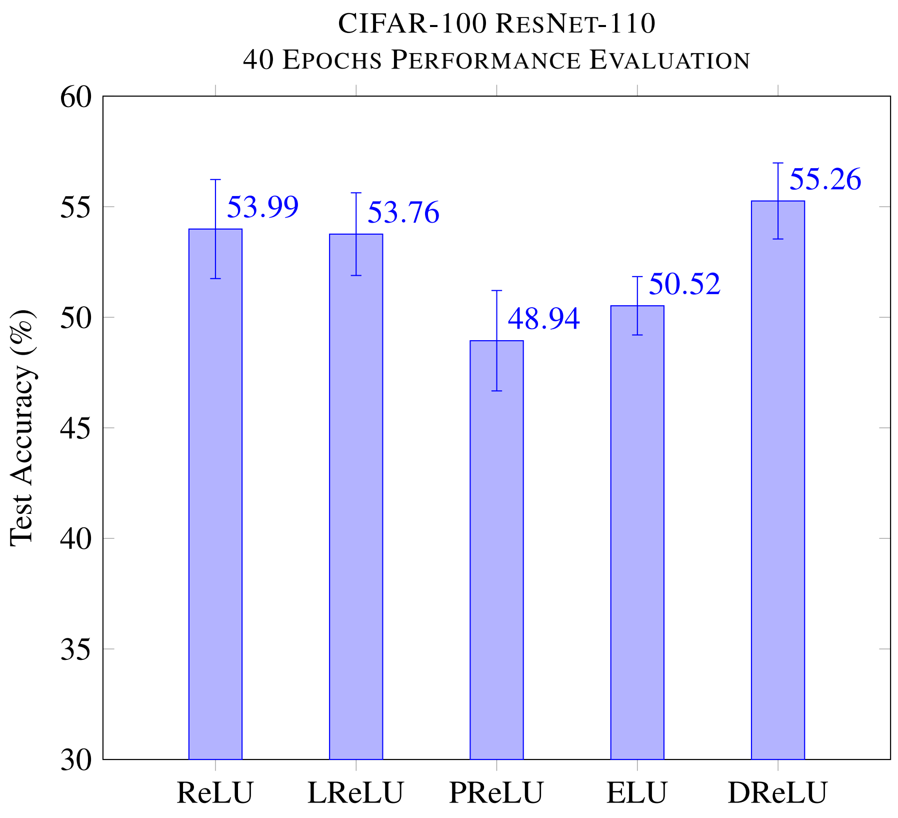
Figure 5.5: ResNet-56 model test accuracy means and standard deviations on the CIFAR-100 dataset. (a) 40 trained epochs. (b) 70 trained epochs.

Table 5.8: CIFAR-100 ResNet-56 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests *p*-values

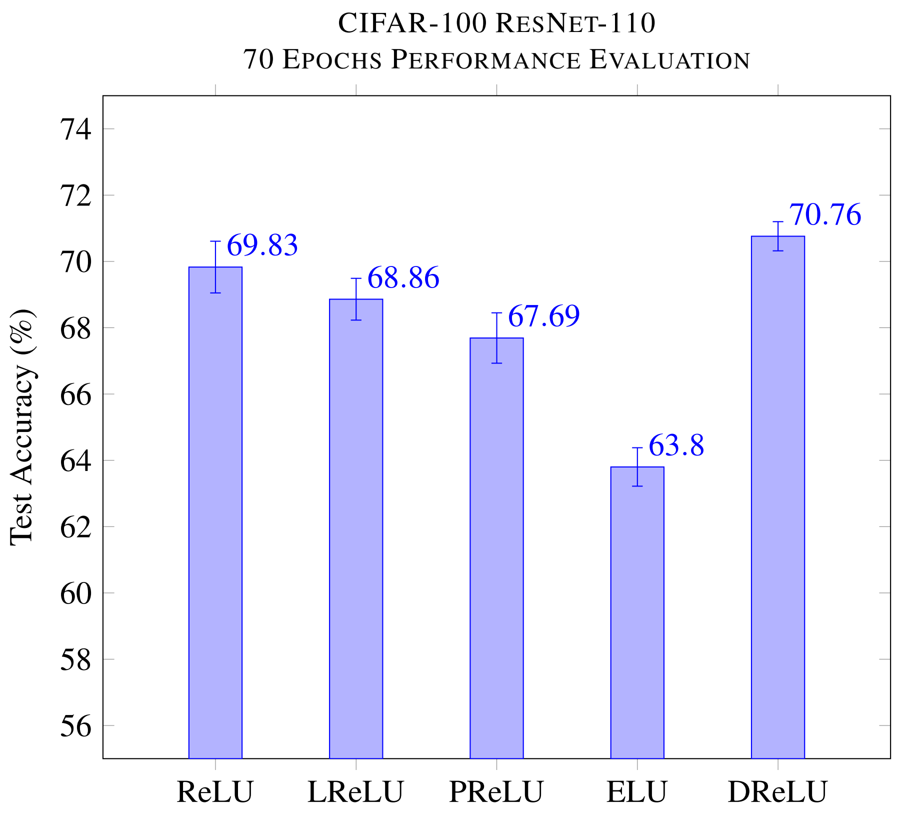
Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	73.70 ± 0.22	-	-	-	-
LReLU	$\mathbf{74.20 \pm 0.31}$	$p < 6.2 \times 10^{-5}$	-	-	-
PReLU	72.53 ± 0.29	$p < 1.5 \times 10^{-5}$	$p < 5.4 \times 10^{-12}$	-	-
ELU	70.13 ± 0.41	$p < 5.7 \times 10^{-11}$	$p < 2 \times 10^{-16}$	$p < 0.00059$	-
DReLU	74.12 ± 0.38	$\boldsymbol{p < 0.00228}$	$\boldsymbol{p < 0.24300}$	$\boldsymbol{p < 2.7 \times 10^{-10}}$	$\boldsymbol{p < 2.4 \times 10^{-15}}$

Table 5.9: CIFAR-100 ResNet-110 100 Epochs Performance Evaluation
Test Accuracy Means, Standard Deviations and Post Hoc Tests *p*-values

Unit	Accuracy (%)	ReLU	LReLU	PReLU	ELU
ReLU	75.70 ± 0.28	-	-	-	-
LReLU	75.94 ± 0.29	$p < 0.0286$	-	-	-
PReLU	74.86 ± 0.37	$p < 1.2 \times 10^{-5}$	$p < 5.3 \times 10^{-9}$	-	-
ELU	70.90 ± 0.42	$p < 3.6 \times 10^{-11}$	$p < 2.9 \times 10^{-14}$	$p < 0.0005$	-
DReLU	$\mathbf{76.20 \pm 0.31}$	$\boldsymbol{p < 2.3 \times 10^{-5}}$	$\boldsymbol{p < 0.0286}$	$\boldsymbol{p < 1.2 \times 10^{-5}}$	$\boldsymbol{p < 3.6 \times 10^{-11}}$



(a)



(b)

Figure 5.6: ResNet-110 model test accuracy means and standard deviations on the CIFAR-100 dataset. (a) 40 trained epochs. (b) 70 trained epochs.

Regarding the CIFAR-100 dataset, DReLU also presented the fastest learning for all three models considered. Moreover, the results showed DReLU always outperformed ReLU test accuracy in all studied models. Besides, DReLU was the most accurate in two evaluated models and the second in the other one.

In this study, we commonly found ReLU as producing the second best training speed and test accuracy performance. This apparent surprise result may be explained by the usage of batch normalization. Indeed, the correction of the internal covariate shift problem enabled by the batch normalization technique acted in benefit of ReLU and detriment of the other previously proposed units. This fact can explain why ReLU normally outperforms LReLU and PReLU in this situations but apparently did not when these activation functions were proposed a few years ago before the advent of batch normalization.

We also observe that the use of batch normalization without dropout is especially harmful to ELU, which commonly present the worst results in our tests. In this point, we emphasize that experiments using both batch normalization and dropout may indeed perform worse than ones entirely batch normalization optimized (IOFFE; SZEGEDY, 2015). In this sense, the fact that ELU performed better than ReLU in an environment using batch normalization and dropout (CLEVERT; UNTERTHINER; HOCHREITER, 2015) does not necessarily implicate the same conclusions would happen in an environment with training parameters fully optimized for batch normalization but without dropout, as we have in our experiments.

In the ELU original paper, the massive use of dropout also in convolutional layers may have helped more ELU than ReLU. However, in current networks, dropout is not habitually used in convolutional layers, but only sometimes in fully connected ones. In the mentioned papers, the dropout was increased in usage during the training from some to almost all layers in the model, a very unusual approach.

Moreover, not only the usage of dropout may affect the conclusion, but also the learning parameters can influence the results. The ELU has less strong no linearity and may be improved with the slow training produced by dropout. Since we are not using dropout and therefore we have a very fast environment, the hard nonlinearity of ReLU and its fast learning speed may have helped ReLU to prevail.

Another explanation is that the customized models used worked better to ELU than to ReLU. At this point, we remember that the best results in the original paper were obtained in models without batch normalization that made exclusive use of dropout regularization. However, we have to emphasize that the ELU paper is very clear in demonstrating that ELU performs much better without batch normalization with dropout.

Therefore, batch normalization enhanced ReLU performance, mainly considering an environment without dropout that undoubtedly tends to slow the overall training speed by increasing the number of epochs needed to achieve good test accuracy. Wherever happened, the fact is that our studies did not show ELU beating ReLU in the experiments realized.

6 CONCLUSION

Become who you are!
—FRIEDRICH NIETZSCHE

Seize the present, trust tomorrow e'en as little as you may.
—HORACE (Odes)

You still have time. Do great things with it.
—DAVID MACÊDO

In this final chapter, we present a brief review of the achievements and main contributions of this study in Section 6.1. Moreover, we propose future works in Section 6.2, which contemplates some suggestions of new activation functions that can be investigated. Finally, we briefly discuss related deep learning topics we are researching.

6.1 CONTRIBUTIONS

This work sheds light on the impact of activation functions on the performance of a deep neural network. To the best of our knowledge, this work is the first systematic comparative study of the performance of the main proposed activation functions designed specifically for use in deep learning to use statistical tests.

In fact, the use of the statistical tests has been shown to be of fundamental importance as the experiments showed a substantial overlap of the test accuracy performance of the compared activation functions. Therefore, this work showed that make conclusions with few repetitions is inappropriate. Moreover, we made a comprehensive systematic study, testing simultaneously the main activation functions currently in use.

In this sense, this work makes the following contributions:

- *Proposes the activation function DReLU.* We generalize ReLU with the proposal of DReLU, which is essentially a ReLU diagonally displaced into the third quadrant;
- *Shows that DReLU provides better training speed than all other evaluated activation functions, including ReLU, in all scenarios.* The statistical tests showed with significance of 5% that DReLU provides the best training speed in all evaluated models and datasets;
- *Shows that DReLU enhances the ReLU performance in all scenarios.* We show using statistical tests with significance of 5% that DReLU enhances the ReLU training speed and test accuracy in all evaluated models and datasets;
- *Shows that DReLU provides better test accuracy than all other evaluated activation functions in all scenarios, with one exception that presented no statistical significance.* The statistical tests showed with significance of 5% that DReLU provides the best test accuracy in almost all evaluated models and datasets with on doubtful exception in which it tied with LReLU in first place;
- *Shows that the design of more efficient activation function is an effective way to enhance deep learning performance saving computational resources.* We show that the performance (training speed and test accuracy) of deep neural networks can be enhanced saving computational resources such as processing power and memory usage by the design of enhanced activation functions.

The experiments used batch normalization but avoided dropout. Furthermore, they were designed to cover standard and commonly used datasets (CIFAR-10 and CIFAR-100) and CNN models (VGG and Residual Networks) of significantly distinct depths and architectures.

In fact, in addition to enhancing deep learning performance (learning speed and test accuracy), DReLU is less computationally expensive than LReLU, PReLU, and ELU. Moreover, the mentioned gains were obtained by just replacing the activation function of the model, without any increment in depth or architecture complexity, which usually increases the computational resource requirements as processing time and memory usage.

Furthermore, considering some evaluated models included skip connections, which are a tendency in the design of deep architectures like ResNets, we conjecture the results may generalize to other architectures such as DenseNets (HUANG et al., 2016) that also make use this structures.

The questions raised at the beginning of this research were satisfactorily answered using the statistical test performed. Regarding **RQ1**, the mentioned statistical tests showed that indeed is possible to enhance deep learning test accuracy by design improved activation functions such as DReLU. Since the DReLU is in fact faster then ReLU and presents better performance, we can also answer **RQ2** positively.

The results showed that the quality of the evaluated activation functions is essentially independent of the model or dataset used. Therefore, we can answer **RQ3** stating that the overall performance of a given activation function is relatively similar regardless of the model or dataset used. Moreover, the experiments clearly showed that the choice of the activation function clearly affects the overall performance of the deep model. Hence, **RQ4** can also be answered positively.

Despite enhancing the performance of the networks, DReLU did not improved the computational resources requirements. This also answers the question **RQ5** positively. Finally, DReLU clearly presented the highest overall test performance and the faster training speedy. Therefore the answer to **RQ6** is DReLU.

6.2 FUTURE WORK

More studies are necessary to verify if the results of this work extend to other model types, datasets and, more generally, other classes of architectures such as Recurrent Neural Networks (RNN). Besides, more investigations need to be performed to evaluate if these conclusions apply to unsupervised and reinforcement deep learning.

Future studies can be made to verify if other activation functions can overcome the performance of DReLU and be even more compatible with batch normalization. In fact, since DReLU is essentially a displaced ReLU, it can be investigated if this transformation of other activation functions also improves performance.

Therefore, if we apply similar displacements to LReLU, PReLU, and ELU, we would obtain DLReLU (Fig. 6.1), DPReLU (Fig. 6.2) and DELU (Fig. 6.3), respectively. The correction of the mean of outputs may help to improve the performance of those transformed nonlinear activation functions.

An alternative approach is to make δ a learnable parameter by incorporating it into the backpropagation procedure. In this sense, the gradients of the loss functions with respect to it would be used to update its values. δ could also be made a layer or even a feature specific, which could improve performance.

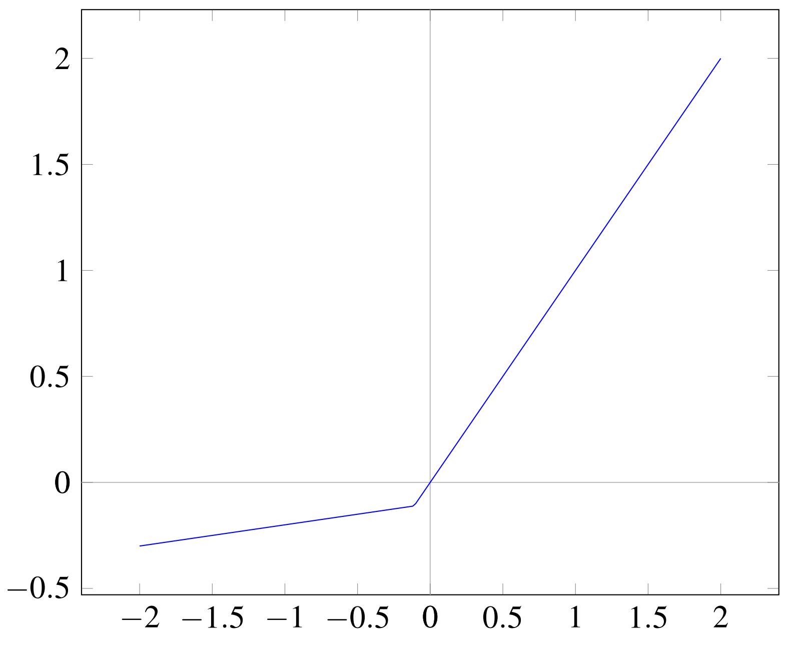


Figure 6.1: Plot of DLReLU activation function.

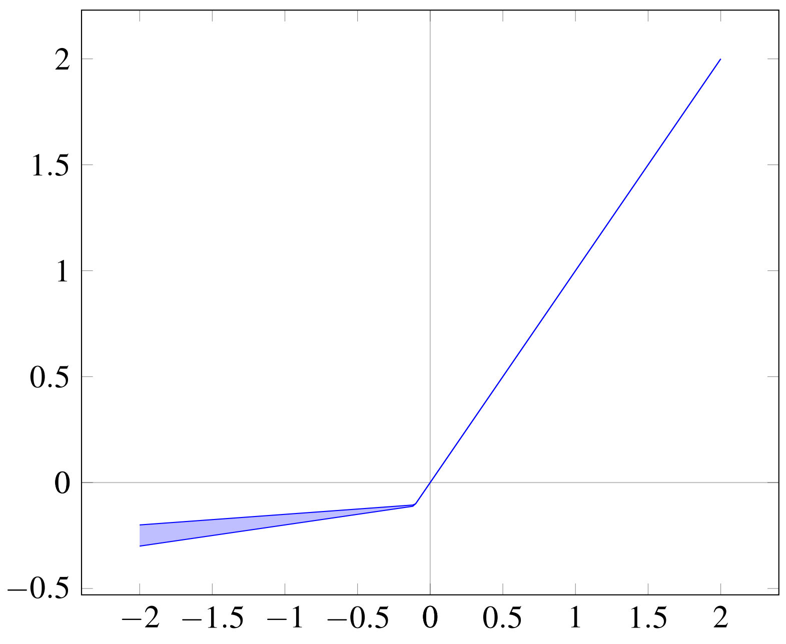


Figure 6.2: Plot of DPReLU activation function.

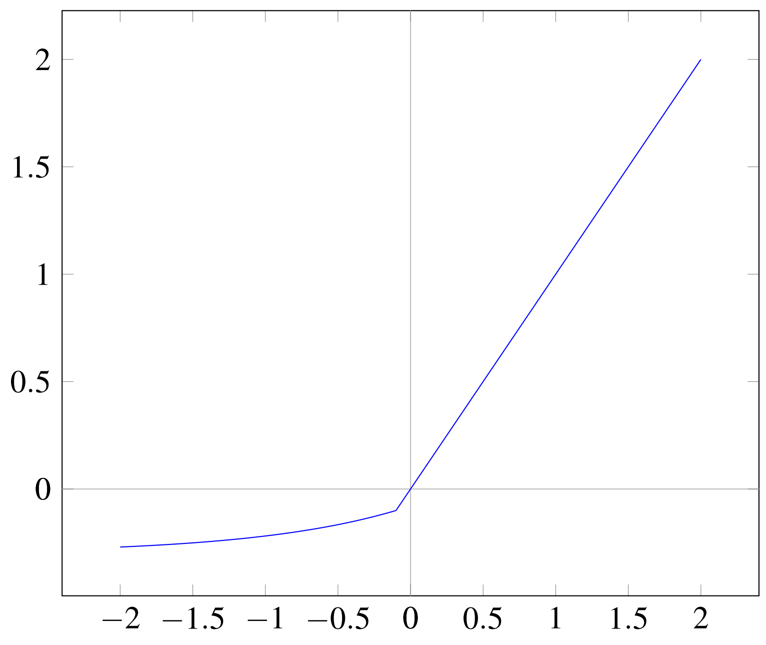


Figure 6.3: Plot of DELU activation function.

AMARI, S.-i. Natural Gradient Works Efficiently in Learning. *Neural Computation*, v. 10, n. 2, p. 251–276, 1998.

AMODEI, D.; ANANTHANARAYANAN, S.; ANUBHAI, R.; BAI, J.; BATTENBERG, E.; CASE, C.; CASPER, J.; CATANZARO, B.; CHENG, Q.; CHEN, G.; CHEN, J.; CHEN, J.; CHEN, Z.; CHRZANOWSKI, M.; COATES, A.; DIAMOS, G.; DING, K.; DU, N.; ELSEN, E.; ENGEL, J.; FANG, W.; FAN, L.; FOUGNER, C.; GAO, L.; GONG, C.; HANNUN, A.; HAN, T.; JOHANNES, L. V.; JIANG, B.; JU, C.; JUN, B.; LEGRESLEY, P.; LIN, L.; LIU, J.; LIU, Y.; LI, W.; LI, X.; MA, D.; NARANG, S.; NG, A.; OZAIR, S.; PENG, Y.; PRENGER, R.; QIAN, S.; QUAN, Z.; RAIMAN, J.; RAO, V.; SATHEESH, S.; SEETAPUN, D.; SENGUPTA, S.; SRINET, K.; SRIRAM, A.; TANG, H.; TANG, L.; WANG, C.; WANG, J.; WANG, K.; WANG, Y.; WANG, Z.; WANG, Z.; WU, S.; WEI, L.; XIAO, B.; XIE, W.; XIE, Y.; YOGATAMA, D.; YUAN, B.; ZHAN, J.; ZHU, Z. Deep speech 2: end-to-end speech recognition in English and mandarin. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. JMLR, 2016. p. 173–182. Available at: <http://dl.acm.org/citation.cfm?id=3045410>.

BAY, H.; ESS, A.; TUYTELAARS, T.; GOOL, L. V. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding*, 2008. ISSN 10773142.

BENGIO, Y. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning*, v. 2, n. 1, p. 1–127, 2009.

BENGIO, Y. Deep learning of representations: Looking forward. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. [S.l.: s.n.], 2013. v. 7978 LNAI, p. 1–37. ISBN 9783642395925. ISSN 03029743.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation Learning: A Review and New Perspectives. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, v. 35, n. 8, p. 1798–1828, 2013. ISSN 1939-3539.

BENGIO, Y.; DELALLEAU, O.; ROUX, N. L. The Curse of Highly Variable Functions for Local Kernel Machines. *Advances in Neural Information Processing Systems 18*, 2006. ISSN 10495258.

BENGIO, Y.; LAMBLIN, P.; POPOVICI, D.; LAROCHELLE, H. Greedy layer-wise training of deep networks. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. MIT Press, 2006. p. 153–160. Available at: <http://dl.acm.org/citation.cfm?id=2976476>.

BISHOP, C. M. C. C. M. *Pattern Recognition and Machine Learning*. [S.l.: s.n.], 2006. ISSN 10179909. ISBN 978-0387310732.

CHOROMANSKA, A.; HENAFF, M.; MATHIEU, M.; AROUS, G.; LECUN, Y. The Loss Surface of Multilayer Networks. In: *arXiv.org*. [S.l.: s.n.], 2014.

CIRESAN, D. C.; GIUSTI, A.; GAMBARDELLA, L. M.; SCHMIDHUBER, J. Mitosis Detection in Breast Cancer Histology Images using Deep Neural Networks. *Proc Medical Image Computing Computer Assisted Intervenction (MICCAI)*, 2013. ISSN 03029743.

CIREŞAN, D. C.; MEIER, U.; MASCI, J.; GAMBARDELLA, L. M.; SCHMIDHUBER, J. High-Performance Neural Networks for Visual Object Classification. *Advances In Neural Information Processing Systems*, 2011. ISSN 10495258.

CLEVERT, D.-A.; UNTERTHINER, T.; HOCHREITER, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). 11 2015. Available at: <http://arxiv.org/abs/1511.07289>.

CONOVER, W. J.; IMAN, R. L. On Multiple-Comparisons Procedures. *Proceedings of the Joint Statistical Meetings, Houston Texas, August*, 1979.

DALAL, N.; TRIGGS, B. Histograms of oriented gradients for human detection. In: *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*. [S.l.: s.n.], 2005. ISBN 0769523722. ISSN 1063-6919.

DAUPHIN, Y.; PASCANU, R.; GULCEHRE, C.; CHO, K.; GANGULI, S.; BENGIO, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. p. 1–14, 2014. Available at: <http://arxiv.org/abs/1406.2572>.

DEAN, T.; CORRADO, G.; SHLENS, J. Three Controversial Hypotheses Concerning Computation in the Primate Cortex. *Twenty-Sixth AAAI Conference on Artificial Intelligence*, p. 1543–1549, 2012. Available at: <http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/viewPDFInterstitial/5093/5299>.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 1980. ISSN 03401200.

FUKUSHIMA, K. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1988. ISSN 08936080.

FUKUSHIMA, K.; MIYAKE, S. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern Recognition*, 1982. ISSN 00313203.

FUKUSHIMA, K.; MIYAKE, S.; ITO, T. Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 1983. ISSN 21682909.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep sparse rectifier neural networks. *AISTATS '11: Proceedings of the 14th International Conference on Artificial Intelligence and Statistics*, v. 15, p. 315–323, 2011. ISSN 15324435.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.: s.n.], 2016. ISBN 3540620583, 9783540620587.

GRAVES, A.; FERNANDEZ, S.; GOMEZ, F.; SCHMIDHUBER, J. Connectionist Temporal Classification : Labelling Unsegmented Sequence Data with Recurrent Neural Networks. *Proceedings of the 23rd international conference on Machine Learning*, 2006. ISSN 10987576.

HAYKIN, S. *Neural Networks and Learning Machines*. [s.n.], 2008. v. 3. 906 p. ISSN 14337851. ISBN 9780131471399. Available at: <http://www.amazon.com/dp/0131471392>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. p. 770–778. ISBN 978-1-4673-8851-1. Available at: <http://ieeexplore.ieee.org/document/7780459/>.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE International Conference on Computer Vision*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2016. v. 11-18-Dece, p. 1026–1034.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Identity Mappings in Deep Residual Networks. 3 2016. Available at: <http://arxiv.org/abs/1603.05027>.

HINTON, G. What kind of a graphical model is the brain? In: *Proceedings of the 19th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 2005. p. 1765–1775. Available at: <http://dl.acm.org/citation.cfm?id=1642293.1642643>.

HINTON, G.; DENG, L.; YU, D.; DAHL, G. E.; MOHAMED, A.-r.; JAITLY, N.; SENIOR, A.; VANHOUCKE, V.; NGUYEN, P.; SAINATH, T. N.; KINGSBURY, B. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *IEEE Signal Processing Magazine*, v. 29, n. November, p. 82–97, 2012. ISSN 1053-5888. Available at: <http://psych.stanford.edu/~jlm/pdfs/Hinton12IEEE_SignalProcessingMagazine.pdf>.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, v. 18, p. 1527–1554, 2006.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the Dimensionality of Data with Neural Networks. *Science*, 2006.

HOCHREITER, S. *Untersuchungen zu dynamischen neuronalen Netzen*. Thesis (Doctorate) — Institut f. Informatik, Technische Univ. Munich., 1991.

HUANG, G.; LIU, Z.; WEINBERGER, K. Q.; MAATEN, L. van der. Densely Connected Convolutional Networks. 8 2016. Available at: <http://arxiv.org/abs/1608.06993>.

HUBEL, D. H.; WIESEL, T. N. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 1959. ISSN 00223751.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 1962. ISSN 00223751.

IOFFE, S.; SZEGEDY, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv*, 2015. Available at: <http://arxiv.org/abs/1502.03167>.

KAWAGUCHI, K. Deep Learning without Poor Local Minima. *NIPS*, n. Nips, 5 2016. Available at: <http://arxiv.org/abs/1605.07110>.

KRIZHEVSKY, A. Learning Multiple Layers of Features from Tiny Images. *Science Department, University of Toronto, Tech*, 2009. Available at: <https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, p. 1097–1105, 2012. ISSN 10495258.

KRUSKAL, W. H.; WALLIS, W. A. Use of Ranks in One-Criterion Variance Analysis. *Journal of the American Statistical Association*, 1952. ISSN 01621459.

LECUN, Y.; BOSER, B. E.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. E.; JACKEL, L. D. Handwritten Digit Recognition with a Back-Propagation Network. *Advances in Neural Information Processing Systems*, 1990.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2323, 1998. ISSN 00189219.

LECUN, Y.; BOTTOU, L.; ORR, G. B.; MÜLLER, K. R. Efficient backprop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 7700 LECTU, p. 9–48, 2012.

LECUN, Y.; CORTES, C.; BURGES, C. THE MNIST DATABASE of handwritten digits. *The Courant Institute of Mathematical Sciences*, 1998.

LOWE, D. G. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999. ISSN 0-7695-0164-8.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In: *Proceedings of the 30 th International Conference on Machine Learning*. [s.n.], 2013. p. 6. Available at: <https://web.stanford.edu/~awni/papers/relu_hybrid_icml2013_final.pdf>.

MÉTIN, C.; FROST, D. O. Visual responses of neurons in somatosensory cortex of hamsters with experimentally induced retinal projections to somatosensory thalamus. *Proceedings of the National Academy of Sciences of the United States of America*, v. 86, n. 1, p. 357–361, 1989. ISSN 0027-8424.

MINSKY, M.; PAPERT, S. *Perceptrons: An Introduction to Computational Geometry*. [S.l.: s.n.], 1969. ISSN 00189340. ISBN 0262631113.

MONTUFAR, G. F.; MORTON, J. When Does a Mixture of Products Contain a Product of Mixtures? 6 2012. Available at: <http://arxiv.org/abs/1206.0387>.

NAIR, V.; HINTON, G. E. Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of the 27th International Conference on Machine Learning*, n. 3, p. 807–814, 2010.

RANZATO, M. A.; POULTNEY, C.; CHOPRA, S.; LECUN, Y. Efficient Learning of Sparse Representations with an Energy-Based Model. *Advances In Neural Information Processing Systems*, 2007. ISSN 10495258.

ROE, a. W.; PALLAS, S. L.; KWON, Y. H.; SUR, M. Visual projections routed to the auditory pathway in ferrets: receptive fields of visual neurons in primary auditory cortex. *The Journal of neuroscience : the official journal of the Society for Neuroscience*, v. 12, n. 9, p. 3651–3664, 1992. ISSN 0270-6474.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, v. 65, n. 6, p. 386–408, 1958.

ROUX, L.; RACOCEANU, D.; LOMÉNIE, N.; KULIKOVA, M.; IRSHAD, H.; KLOSSA, J.; CAPRON, F.; GENESTIE, C.; NAOUR, G.; GURCAN, M. Mitosis detection in breast cancer histological images An ICPR 2012 contest. *Journal of Pathology Informatics*, 2013. ISSN 2153-3539.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. *Learning internal representations by error propagation*. 1986. 318–362 p.

SHAH, A.; KADAM, E.; SHAH, H.; SHINDE, S. Deep Residual Networks with Exponential Linear Unit. *Arxiv:1604.04112*, p. 7, 4 2016. Available at: <http://arxiv.org/abs/1604.04112>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR, abs/1409.1556*, 2014. ISSN 09505849.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 2014.

SRIVASTAVA, R. K.; GREFF, K.; SCHMIDHUBER, J. Highway Networks. 5 2015. Available at: <http://arxiv.org/abs/1505.00387>.

TAX, D. M. J.; DUIN, R. P. W. Feature scaling in support vector data descriptions. In: *Conference of the Advanced School for Computing and Imaging*. [S.l.: s.n.], 2002. ISBN 0199206139.

ZAGORUYKO, S.; KOMODAKIS, N. DiracNets: Training Very Deep Neural Networks Without Skip-Connections. 6 2017. Available at: <http://arxiv.org/abs/1706.00388>.