Universitat de València
Escola Tècnica Superior d'Enginyeria

PhD Thesis. Doctorat en Enginyeria Electrònica

Contributions and applications around low resource deep learning modeling

by Iván Vallés-Pérez

Supervisor: Emilio Soria-Olivas, PhD
València - September, 2022

Acknowledgements

I have always had in mind the idea of pursuing a PhD program. However, after finishing my bachelor degree, I found great professional opportunities that I could not overlook. Nonetheless, I never lost the esire to continue my educational journey. Thus, a few years later, I found the fantastic opportunity to start a part-time PhD at Universitat de València. Being a part-time PhD student at the *Intelligent Data Analysis Laboratory* of the *Universitat de València* was lots of fun, and one of the best decisions I could have made.

I would like to start by sincerely expressing my gratitude to Emilio Soria-Olivas, PhD my advisor. First of all, thank you for instilling your passion in artificial intelligence with me. Your encouragement and guidance has been a key factor in the development of this thesis, without which this work could never have been possible. I would also like to thank all the members of the Intelligent Data Analysis Laboratory of the University of Valencia, many of whom directly collaborated and reviewed my work: Antonio-José Serrano-López, Ph.D., Marcelino Martínez-Sober, Ph.D., Juan Gómez-Sanchis, Ph.D., Fernando Mateo-Jiménez, Ph.D and Joan Vila-Francés, Ph.D.

On the professional side, I would like to thank all the *Alexa TTS* research group, for accepting me into their family. Thanks to all the people who directly collaborated to my contributions to the speech generation field: Dr. Roberto Barra-Chicote, Ph.D., Dr. Jasha Droppo, Ph.D., Mr. Julian Roth, M.Sc. and Mr. Grzegorz Beringer, M.Sc. Special mention to Dr. Roberto Barra-Chicote, Ph.D., my mentor, for all his guidance and support, and for continuously boosting my motivation and helping me be a better scientist.

Finally, and more importantly, I would like to acknowledge my family, especially my partner María, my brother David and my parents Reme and Ignacio, for all the unconditional support during these years, in all its modalities. This work would not have been possible without your love.

Abstract

Deep learning is the current state of the art for several machine learning tasks. Many of these tasks require large amount of computational resources, which limits their adoption in embedded devices. The main goal of this dissertation is to study methods and algorithms that allow to approach deep learning problems with restricted computational resources. This work also aims at presenting applications of deep learning in industry.

The first contribution is a new activation function for deep learning networks: the *modulus* function. The experiments show that the proposed activation function achieves superior results in computer vision tasks when compared with state of the art alternatives.

The second contribution is a new strategy to combine pre-trained models using knowledge distillation. The results of this chapter show that it is possible to significantly increase the accuracy of the smallest pre-trained models, allowing for computational savings and improved performance.

The first application covered in this dissertation tackles the problem of sales forecasting in the field of logistics. Two end-to-end systems with two different deep learning techniques (sequence-to-sequence models and transformers) are proposed. The results of this chapter conclude that it is possible to build end-to-end systems to predict the sales of multiple individual products, at multiple points of sale and different times with a single machine learning model. The proposed model beats the state of the art alternatives found in the bibliography.

Finally, the last two applications belong to the speech technology field. The former studies how to build a *Keyword Spotting* speech recognition system using an efficient version of a convolutional neural network. In this study, the proposed system is able to beat the performance of all the benchmarks found in the literature when tested against the most complex subtasks. The latter study proposes a standalone state of the art *text-to-speech* model capable of synthesizing intelligible voice in thousands of voice profiles, while generating speech with meaningful and expressive prosody variations. The proposed approach, removes the dependency of previous models on a production voice system, which makes it more efficient at training and inference time, and enables offline and on-device operations.

Resumen

Los algoritmos de aprendizaje profundo representan el estado de la cuestión en lo que a aprendizaje automático se refiere. Muchas de sus aplicaciones requieren una gran cantidad de recursos computacionales, la cual limita su uso a dispositivos de alto rendimiento. El objetivo principal de esta tesis es estudiar métodos y algoritmos que permitan abordar problemas de aprendizaje profundo cuando se tienen recursos computacionales limitados. Este trabajo también tiene como objetivo presentar aplicaciones de aprendizaje profundo en la industria.

La primera contribución consiste en una nueva función de activación para redes de aprendizaje profundo: la función *módulo*. A partir de los experimentos realizados se observa que la función de activación propuesta logra resultados superiores en tareas de visión artificial en comparación con las alternativas más avanzadas.

En segundo lugar, se presenta un nuevo método para combinar modelos pre-entrenados usando técnicas de destilación de conocimiento. Los resultados de este capítulo muestran el uso de las técnicas propuestas permite aumentar significativamente el desempeño de los modelos pre-entrenados más pequeños. Esto proporciona mejoras computacionales y de rendimiento.

La tercera aportación de esta tesis aborda el problema de la predicción de ventas en el campo de la logística. Se proponen dos sistemas de basados en dos técnicas diferentes de aprendizaje profundo (modelos de secuencia-a-secuencia y *transformers*). De los resultados de este capítulo se concluye que es posible construir sistemas integrales para la predicción de ventas de múltiples productos individuales, en múltiples puntos de venta y en diferentes momentos en el tiempo, mediante el uso de un único modelo de aprendizaje automático. Los resultados del modelo propuesto superan significativamente a las alternativas encontradas en la literatura.

Finalmente, las dos últimas contribuciones pertenecen al campo de la tecnología del habla. El primero estudia cómo construir un sistema de reconocimiento de comandos de voz (*Keyword Spotting*) utilizando una versión eficiente de una red neuronal convolucional. En este estudio, el sistema propuesto es capaz de superar el rendimiento de las alternativas encontrados en la literatura, en las tareas más complejas. El último estudio propone un modelo independiente de generación de habla capaz de sintetizar voz natural e inteligible usando miles de perfiles de voz distintos, generando habla expresiva con variaciones de prosodia significativas. El enfoque propuesto elimina la dependencia de los modelos anteriores en un sistema de voz de producción, lo que lo hace más eficiente en el tiempo de entrenamiento e inferencia.

Contents

List of Figures

List of Tables

Notation

This section summarizes the notation conventions followed in this dissertation. As a general note, bold lowercase symbols (e.g. $\mathbf{x}$) have been used to denote vectors, while bold uppercase (e.g. $\mathbf{X}$) symbols represent matrices. Non bold symbols (e.g. x) have been used to represent scalars. This convention will be followed throughout the dissertation unless specified.

Data

$\mathbf{X}$	a matrix with shape $N \times D$ containing N input row vectors, represented as $\{\mathbf{x_1}, ..., \mathbf{x_i}, ..., \mathbf{x_N}\}$, where each vector $\mathbf{x_i} \in \mathbb{R}^D$.
$\mathbf{x_i}$	a vector of length D where each element represents a particular feature (e.g. age, weight, or IQ of a person).
$\mathbf{Y}$	a matrix with shape $N \times K$ containing desired output vectors rows $\{\mathbf{y_1}, ..., \mathbf{y_i}, ..., \mathbf{y_K}\}$, where each output vector $\mathbf{y_i} \in \mathbb{R}^K$.
$\mathbf{y_i}$	a vector of length K where each element is one of the scalar desired outputs (e.g. the probability of an image containing a dog).
$\mathbf{T}$	a dataset of examples with inputs and desired outputs for supervised learning tasks, composed of a set of pairs of vectors from $\mathbf{X}$ and $\mathbf{Y}$, grouped as follows: $\{(\mathbf{x_1}, \mathbf{y_1})..., (\mathbf{x_i}, \mathbf{y_i}), ..., (\mathbf{x_N}, \mathbf{y_N})\}$, where each tuple represents a training example.
$\mathbf{U}$	a dataset of examples for unsupervised learning tasks, composed of a set of input vectors from $\mathbf{X}$, disposed as follows: $\{\mathbf{x_1}, ..., \mathbf{x_i}, ..., \mathbf{x_N}\}$, where each element $\mathbf{x_i} \in \mathbb{R}^D$ represents a training example.
N	number of examples in a dataset.
D	number of features in a dataset or in a feature vector.
K	dimensionality of a desired output variable (multivariate) of a dataset.
T	length of a time series $(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}, ..., \mathbf{x}^{(T)})$, where $\mathbf{x}^{(t)} \in \mathbb{R}^D$ and the super-index refers to the time step the observation belongs to.
$\mathbf{C}$	set of possible labels associated with the examples of a dataset, where $\|\|C\|\|$ is used to represent the cardinality of that set.

Neural networks and machine learning

$\hat{\mathbf{y}}$	predicted response variable, generally the output of a machine learning model.
$\mathbf{z}$	vector representing a variable in a latent space.
b	bias term of a neuron.
θ, ϕ	vector containing all the parameters of a neural network model.
$f_\theta(\cdot)$	neural network model with parameters θ.
$J(\cdot, \cdot)$	cost function, i.e. function quantifying the error intended to be minimized, often using gradient descent.
$\mathbf{W}, \mathbf{U}$	weight matrices of a neural network layer which scalar components are denoted by w_{ij}.
$g(\cdot)$	activation function.
$G(\cdot)$	multilayer *perceptron*.
$\mathbf{S}$	feature map in a convolutional neural network, result of performing a cross-correlation operation on an input $\mathbf{X}$ with a kernel $\mathbf{W}$.
t	index that refers to time or sequence steps. Generally used for referring to an optimization step or to a time series element.
λ	learning rate.
m	mini-batch size.
L	number of layers of a deep learning model.
$\mathbf{h}$	output of a hidden layer.
$\mathbf{a}$	the attention vector.
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	the query, key and value matrices in the attention mechanism context, respectively.
d_k	the length of a sequence in the attention mechanism context.
Nx	the number of blocks in a transformer.
f_e, f_d	encoder and decoder networks of an auto-encoder, respectively.
p_θ, q_ϕ	encoder and decoder networks of a variational auto-encoder, respectively, with parameter sets θ and ϕ.
$\mathcal{L}$	variational lower bound.
f_g, f_d	generator and discriminator in the context of generative adversarial networks.
$\mathcal{T}$	temperature parameter of a *softmax* operation.

Others

D_{KL}	*Kullback-Leibler* (KL) divergence between two distributions.
$p(\cdot)$	probability distribution.
$H(\cdot)$	*Heavyside* function.
$\sigma(\cdot)$	*sigmoid* function.
$\tau(\cdot)$	hyperbolic tangent function ($\tanh(\cdot)$).
$\text{sgn}(\cdot)$	Sign function.
$\odot$	*Hadamard* product (also known as element-wise multiplication).
$\circ$	composition operation.
$\nabla_\theta f$	gradient of the function f with respect to parameters θ.
^{T}	transposition operation.

Chapter 1

Introduction

1.1 Overview

1.1.1 Early artificial intelligence references

Humans are curious by nature. We have the desire to understand everything around us, from the smallest particle to the vastness of the universe. Our quest for knowledge is what has allowed us to progress as a species and keep evolving. In this journey, we have always looked up to the stars, dreaming of discovering other intelligent beings like us. In this search, we have also looked inward, trying to understand our thoughts, emotions, and the source of our intelligence and consciousness.

Since ancient times, the human being has dreamed of artificial intelligence (AI). One of the first existing records dates back to *Aristotle* (384–322 BCE) in his book *The Politics*, where the author imagined machines that would think by themselves and act autonomously, with the purpose of allowing humans enjoy leisure (Nilsson, 2009). In 10-70 CE, the mathematician and engineer *Hero of Alexandria* designed several ancient automata (Greenwood & Woodcroft, 1851), among which stands out an automated theater that would play short performances in front of the audience.

Later, in the 9th century, the three Persian brothers known as *Banū Mūsā* wrote the *book of ingenious devices* (Ibn Shākir, 1979). In their book, they illustrated hundreds of automata along with other mechanical devices (timing and delay devices, automated valves, etc.) and described how those would be used.

In the middle ages, the philosopher, scientist and bishop *Albert Magnus* (13th century) manufactured several automata. One of the most notorious ones was an artificial talking head able to imitate human voice and breadth (Lacey, 1828).

Later on, in the renaissance (15th century), the polymath *Leonardo da Vinci* sketched various automata (Nilsson, 2009). His mechanical knight, is one of the first anthropomorphic automata we have record of. This automaton was designed to perform basic human-like motions through a system of pulleys and cables. More recently, these sketches have been studied and the mechanical knight has been built faithfully following the original design (Rosheim, 2006), finding that the automaton was fully functional.

During the 18th century, automata reached a new level of sophistication, the

modern history becoming the golden era of automata. One of the most prolific creators was the watchmaker *Pierre Jaquet-Droz* (18th century), who built a number of automata, including a three-year-old child that could write any letter of the alphabet (Carrera et al., 1979).

Years later, other illustrious automata builders of the era were *Wolfgang von Kempelen* (1734-1804), who built *the Turk*, an automaton that could beat any human at chess[1] (Jay, 2000), and *Jacques de Vaucanson* (1709-1782), who built a number of automata, including a duck that could eat and drink (Murthy, 2022; Nilsson, 2009). Despite the complexity and ingenuity gained over the years, these automata were all purely mechanical and human operated, without any form of artificial intelligence.

Fiction stories have often been used as a way to explore the idea of artificial intelligence, and the earliest references in literature date back to the middle ages. In *The City of Brass*, one of the tales included into the *One Thousand and One Nights* (around the 10th century), the main character, a thief, comes across a city ruled by a wizard who created a brass humanoid automaton that could talk and move like a human. In the 19th century, *E.T.A. Hoffmann* (1776-1822) wrote a story entitled *The Sandman* (Hoffmann, 1816), where the protagonist falls in love with an automaton created by a professor, without realizing that she was actually a machine. He suffered a mental breakdown after founding out. This automaton, known as Olympia, was able to move, talk and sing. In one of the most famous science fiction novels of all times, *Frankenstein*, written by *Mary Shelley* in 1818 (Shelley, 1994), the scientist *Victor Frankenstein* creates a human-like creature out of body parts of different people. Although, strictly speaking, Frankenstein's creature is not a machine, it is often seen as one of the first examples of intelligent creations in literature.

A century later, the Czech writer, playwright and critic *Karel Čapek* (1920), wrote a play named *R.U.R.* (*Rossum's Universal Robots*), which is considered to be the first recorded use of the term *robot*[2] (Nilsson, 2009). In that play, robots were manufactured as slaves to do the manual labour that humans disliked. This is often seen as the beginning of the modern science fiction genre.

In 1941, the science fiction writer *Isaac Asimov* published a short story called *Runaround* (Nilsson, 2009), in which he introduced the three laws of robotics (depicted below), which are still considered the basis for the ethical design of robots.

1. *A robot may not injure a human being or, through inaction, allow a human being to come to harm.*

2. *A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.*

3. *A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.*

The above examples are only a few of the many references to artificial intelligence that have been made over the years (Nilsson, 2009). Even though

[1] Although after his death, it was discovered that it was actually nothing more than a machine operated by a person from inside a wooden cabinet

[2] The word *robot* is derived from the Czech word *robota*, which means forced labour or slaves.

none of these automata were truly intelligent, they represent the beginning of the long and arduous process of building machine intelligence.

The yearning for artificial intelligence has come and gone in waves over the years, but it has never lost its appeal to the human imagination. Each time a new wave of artificial intelligence hits, it brings with it renewed enthusiasm for the possibility that machines can think and act by themselves.

Recently, artificial intelligence has begun to show significant promise for actually becoming a reality. The history of artificial intelligence is a long one, and its future is still to be written.

1.1.2 Modern artificial intelligence

The dream of AI started to become a reality in the 20th century, with the development of the first computers. In the 1950s, the idea of artificial intelligence was rediscovered by *Alan Mathison Turing* (1912 - 1954), today known as the father of computer science. He secretly defeated the German intelligence's cryptography system (*Enigma* machines) (Hodges, 2000) and provided a proof showing that it was not possible to find a general solution for the *Hilbert's Entscheidungsproblem* (an important symbolic logic challenge) (Turing, 1936). Later, he published the famous article entitled *Computing Machinery and Intelligence* (Turing, 1950) in which he described a game as a test for machine intelligence. The currently known as *Turing test* consists of an interrogation between a human and an entity (that can either be another human or a machine). The human interrogator's objective is to try to determine, by asking a series of questions, whether the entity it is talking to is a human or a machine. If the interrogator cannot tell the difference (70% of the times after multiple 5 minutes conversations), then the machine is said to have passed the test, and hence it can be considered intelligent.

The "artificial intelligence" term was not coined until 1956 when *John McCarthy* (1927 - 2011) *Nathaniel Rochester* (1919 - 2001) and *Claude Shannon* (1916 – 2001) gave a conference at *Dartmouth College* proposing a 2 month workshop, called the *Dartmouth Summer Research Project on Artificial Intelligence*. This workshop was organized to fund the artificial intelligence as an academic discipline (see the picture in figure 1.1) and the project was funded by the *United States Office of Naval Research*. This historical event brought together some of the most prominent computer scientists of the time, including *Marvin Minsky* (1927–2016), *Arthur L. Samuel* (1916–1990), *Ray Solomonoff* (1926-2009) and *John Nash* (1928-2015), between others[3].

The recent development of artificial intelligence has not been a linear process, and during its evolution, there have been ups and downs along the way, with several so-called *AI winters*. The *first AI winter* took place between the late 1970s and early 1980s, when research on artificial intelligence came to a standstill. Later, *Sir James Lighthill*, a well-known British scientist, published a report, currently known as *the Lighthill report*, concluding that artificial intelligence was a waste of time and money (Lighthill, 1973). This publication, together with the oversized expectations for artificial intelligence at the time (Russell, 2003),

[3] *John McCarthy*, *Claude Shannon*, *Trenchard More*, *Nathaniel Rochester*, *Oliver Selfridge*, *Julian Bigelow*, *W. Ross Ashby*, *W.S. McCulloch*, *Abraham Robinson*, *Tom Etter*, *David Sayre*, *Kenneth R. Shoulders*, *Alex Bernstein*, *Herbert Simon* and *Allen Newell*

Figure 1.1: From left to right: *Oliver G. Selfridge*, *Nathaniel Rochester*, *Ray Solomonoff*, *Marvin Minsky*, *Trenchard More*, *John McCarthy* and *Claude Shannon* at the *Dartmouth Summer Research Project on Artificial Intelligence* (Photo: *Margaret Minsky*).

led to a decrease of investment for AI research. As a result, the field of artificial intelligence went into decline, stalling for several years.

After the first *AI winter*, the field experienced a revival when several companies started investing in the development of *expert systems*[4]. Software and hardware companies such as *Teknowledge* and *LISP Machines Inc.* grew rapidly during this time in order to meet the rising demand for artificial intelligence technology, which motivated the AI research community to continue with their work. The most remarkable inventions of the 80s was the successful application of the *backpropagation* algorithm to neural networks by *David Rumelhart* and *Geoffrey Hinton* (Rumelhart et al., 1986), which revived the study of artificial neural networks. All this sudden success, together to the collapse of *LISP Machines Inc.* in 1987, led to the *second AI winter*.

The 1990s started with a renewed interest in artificial intelligence and, motivated by the increasing computational power, machine learning algorithms started to be applied to a wider range of tasks (Tesauro, 1995). Many interesting applications were developed during that decade across several industries like medical diagnosis (Cinar et al., 1999; DeClaris, 1991; Klein & Shortliffe, 1991; Punch, 1992), psychology (Denby & Gammack, 1999; Dorrer et al., 1995; Ogawa et al., 1999; Perlovsky, 1999), finance and logistics (Benaroch & Dhar, 1991; Falas et al., 1994; Johnson & Hoback, 1991; Lipshutz et al., 1991) and many others (Koyama et al., 1998; Mashaly et al., 1994; Smithers et al., 1993; Yoo et al., 1994). Finally, the most sounded event of the 1990s was the victory of the IBM super-computer *Deep Blue* (Campbell et al., 2002) over *Garry Kasparov*, the world chess champion, which took place in *New York City* in 1997. In 1998, *Yann Lecun* and his collaborators published their work on convolutional neural networks (Y. LeCun et al., 1999), a fundamental advance in machine learning which has been widely used in computer vision and other fields.

The beginning of the 21st century is one of the most fruitful periods for artificial intelligence, with several major achievements in different areas. Further research in neural networks (G. E. Hinton et al., 2006; G. E. Hinton et al., 2012)

[4] A rebranded form of artificial intelligence

gave birth to novel techniques that allowed training deeper models, leading to a rebranding of neural networks as deep learning. In 2011, *IBM's* artificial intelligence program *Watson* won the quiz game *Jeopardy!* against two of the best human players of all time.

Research in neural networks gave birth to novel techniques that allowed researchers to train deeper models, leading to another rebranding of neural networks as deep learning. In 2012, *AlexNet* (Krizhevsky et al., 2017), a deep learning model developed by *Geoffrey Hinton* and his collaborators, achieved state of the art performance on the *ImageNet* classification task (Russakovsky et al., 2015), kickstarting the current deep learning revolution.

In 2016, *Google's* artificial intelligence program *AlphaGo* (Silver et al., 2016) defeated the world champion *Lee Sedol* in *the game of Go*, a feat that was considered impossible a few years earlier, by using *reinforcement learning* algorithms. *Alpha Go Zero*, the *Alpha Go's* big brother, proved in 2017 to be more powerful than its predecessor while not needing human interaction to learn (Silver, Hubert, et al., 2017; Silver, Schrittwieser, et al., 2017). In 2016, *DeepMind* researchers published *WaveNet* (van den Oord, Dieleman, et al., 2016), a deep auto-regressive model that was able to produce natural raw audio waveforms faithfully generating human speech (a modern version of *WaveNet* has been used in the experiments described in chapter 7). One year later, the authors of (Vaswani et al., 2017) published the transformer, a new architecture designed for sequence transduction task that allowed parallel training, as opposed to its sequence-to-sequence predecessors (Sutskever et al., 2014). The transformer architecture has been used in the experiments described in chapter 5. In 2020, *OpenAI* researchers published *GPT-3* (Brown et al., 2020), a massive neural network with 175 billion parameters which was able to achieve strong performance on many NLP tasks[5]. Lately, in 2021, *AlphaFold* was published by *DeepMind* researchers (Jumper et al., 2021) as a method for inferring the 3D structure of a biological protein based on its genetic sequence, representing one of the major contributions of artificial intelligence to scientific discovery.

In addition to the mentioned achievements, many algorithms have been published in the generative modeling field. Generative adversarial networks (I. Goodfellow et al., 2014), variational auto-encoders (Kingma & Welling, 2019), normalizing flows (Kingma et al., 2016; Kobyzev et al., 2021) and diffusion models (Dhariwal & Nichol, 2021) are some of the most remarkable examples. These topics will be discussed in detail in section 2.3.5.

Many more advances have been made in AI in the past few years, which has led to an increased interest in the technology from both the private and public sectors. However, the application of artificial intelligence to complicated and important tasks still faces many challenges. Here is where deep learning comes into play, as it has shown the ability to achieve state of the art results on a wide range of tasks. Therefore, the development of democratized and low-resource deep learning applications is essential for the future of artificial intelligence.

Many of the last advances in the deep learning research community report prohibitive amounts of computation needed to train deep learning models (Brown et al., 2020; Floridi & Chiriatti, 2020; Kechyn et al., 2018; Silver et al., 2016). The authors of (Strubell et al., 2019) estimated the amount of CO_2 emissions

[5] A live example of the power of this model can be found in some of the lines of this chapter, which have been revisited and rephrased with its help.

from training a large transformer network to be equivalent to the emissions of 5 average cars during all their lifetime, or those of a human during 60 years.

Research in low-resource deep learning (Gao et al., 2018; S. Han et al., 2017; Howard et al., 2017; Sanchez-Iborra & Skarmeta, 2020; So et al., 2021) has shown the possibility of training deep learning models with a low amount of computation, and its necessity for the future of artificial intelligence, as well as for the health of our planet.

1.2 Contributions

This dissertation encompasses five contributions to the state of the art of the field of deep learning. Two of them contribute around training methods, and the remaining three are examples of applications of deep learning algorithms to industry problems. All the contributions are related to the efficient use of computational resources. Each of these studies is written as a different chapter.

The first contribution (chapter 3) proposes a new activation function for deep learning models: the *modulus* function. The experiments show that, in line with the current research trends, non-monotonic activation functions generally produce better results than monotonic ones. Additionally, the *modulus* activation function is very efficient to compute, as it consists of a single-bit operation, and its derivative (being either 1 or -1) has constant 1-norm. These properties are specially useful for embedded applications. Moreover, the results show that the proposed activation function achieves superior results in computer vision tasks when compared with state of the art alternatives.

The second contribution (chapter 4) proposes combining the knowledge of several large pre-trained models in order to improve the performance of small low-resource pre-trained models. The results of this chapter show that it is possible to significantly increase the accuracy of the smallest pre-trained models, allowing for computational savings and improved performance.

The first application covered in this dissertation (chapter 5) tackles the problem of sales forecasting in the field of logistics. Two end-to-end systems with two different deep learning techniques (sequence-to-sequence models and transformers) are proposed. The results of this chapter conclude that it is possible to build end-to-end systems to predict the sales of multiple individual products, at multiple points of sale and different times with a single machine learning model. The proposed model beats the state of the art alternatives found in the bibliography. This work has been published in *Elsevier*'s journal *Expert Systems with Applications* (Iván Vallés-Pérez and Emilio Soria-Olivas and Marcelino Martínez-Sober and Antonio J. Serrano-López and Juan Gómez-Sanchís and Fernando Mateo, 2022).

Finally, the last two applications belong to the speech technology field. The former (chapter 6) studies how to build a *Keyword Spotting* speech recognition system using an efficient version of a convolutional neural network. In this study, the proposed system is able to beat the performance of all the benchmarks found in the literature when tested against the most complex subtasks. This work has been published in the proceedings of *European Symposium of Artificial Neural Networks (ESANN 2020)* (Vallés-Pérez et al., 2021a). The latter study (chapter 7) proposes a standlalone state of the art *text-to-speech* model capable of synthesizing intelligible voice in thousands of voice profiles, while generating speech with

meaningful and expressive prosody variations. The proposed approach, removes the dependency of previous models on a production voice system, which makes it more efficient at training and inference time, and enables offline and on-device operations. Device-embedded TTS models are important for voice-activated user interfaces, as they provide a more natural user experience due to the reduced latency, while using a fraction of the energy of cloud TTS services. This study was done as part of the work of the author at *Alexa AI* and has been published in the proceedings of *Interspeech 2020* conference (Vallés-Pérez et al., 2021b).

The unpublished works referenced in this section have already been submitted to a journal and, at the time of writing this paragraph, are under revision.

1.3 Thesis structure

This thesis is organized in individual chapters as follows.

- *Chapter 2*: covers the common background needed to understand the methods and algorithms applied in the subsequent chapters.
- *Chapter 3*: introduces the modulus as activation function, showing its benefits over other alternatives.
- *Chapter 4*: studies how to combine pre-trained models using knowledge distillation.
- *Chapter 5*: explores the application of sequence-to-sequence models and transformers to approach the sales forecasting problem, from an end-to-end perspective.
- *Chapter 6*: presents an end-to-end keyword spotting system with convolutional networks.
- *Chapter 7*: proposes a state of the art multi-speaker text-to-speech (TTS) system with prosody modeling.
- *Chapter 8*: wraps the general conclusions of the studies presented in the previous chapters.

Chapter 2

Background

2.1 Machine learning

Human beings learn by experience, part of which is inherited from previous generations. However, in the digital world, experiences can be stored in form of data, which can be later processed and analyzed.

We live in the middle of a data deluge. The technological progress and the internet have boosted our logging and communication capacities. At the time of writing this paragraph[1], every single second 10,000 new tweets are written, 100,000 search queries are sent to *Google*, 100,000 videos are being viewed in *YouTube*, and 3,000,000 emails are sent. All amounts to approximately a 140 terabytes of internet traffic per second.

This *Brobdingnagian* amount of data cannot be analyzed without the help of automated computational assisted tools, and this is exactly the purpose of machine learning. More formally, we define machine learning as a set of computational methods designed to automatically learn hidden structures and patterns from the data and its origin (Murphy, 2012; Theodoridis, 2015). Machine learning algorithms can serve multiple purposes ranging from informing decision making under uncertainty to understand and simulate natural processes. Sometimes, machine learning algorithms are inspired in biological processes or in how the brain works and learns (Haykin, 1999) (e.g. *self-organizing maps* (Kohonen, 2000)). Other times, machine learning is driven by specific needs arising from data analysis problems (e.g. binary decision trees (Hastie et al., 2009; James et al., 2017)).

[1] https://www.internetlivestats.com/one-second/ on February 6th 2022

2.2 Types of learning

Machine learning algorithms are designed to learn from data. However, there are many ways these data can be treated in the learning process. In this section, the most common types of learning are described at a high level.

2.2.1 Supervised learning

Supervised learning is the most widely employed methodology to train machine learning models. It is based on a function-fitting perspective, where the function f_θ is adjusted (or trained, in machine learning jargon) to map a set of input vectors $\mathbf{X}$ to the corresponding output vectors $\mathbf{Y}$ ($f_\theta : \mathbf{X} \rightarrow \mathbf{Y}$), given a set of N input pairs $\mathbf{T} = \{(\mathbf{x_i}, \mathbf{y_i})\}_{i=0}^{N}$ known as the training dataset (Theodoridis, 2015). The learning algorithm adjusts the parameters θ of a function f_θ according to the minimization of a predefined cost function J (for example the mean squared error between the predicted values and the labels) (Hastie et al., 2009). The vector $\mathbf{x_i}$ (with length D, $\mathbf{x_i} \in \mathbb{R}^D$) represents a set of features (e.g. the age and the income of a person) and $\mathbf{y_i}$ (with length K, $\mathbf{y_i} \in \mathbb{R}^K$) is the vector of response variables (representing for instance the probabilities to buy a set of products)[2].

There are two main forms of supervised learning (Murphy, 2012).

- *Regression*, where the task consists of mapping each input vector $\mathbf{x_i}$ to a real-valued vector $\mathbf{y_i} \in \mathbb{R}^K$. An example of this task would be predicting the age of an *abalone*[3] based on physical measurements of the different parts of its body (Dua & Graff, 2017).

- *Classification*, where a task consists of mapping the input vectors $\mathbf{x_i}$ to nominal variables from a finite set $\mathbf{C_j}$, $y_{i,j} \in \{1, 2, ..., ||\mathbf{C_j}||\}$, where $||\mathbf{C_j}||$ is the cardinality of the j-th response set. An example of a classification task would be determining if a mushroom is poisonous or edible based of several physical characteristics (Dua & Graff, 2017).

2.2.2 Unsupervised learning

Unsupervised learning techniques are employed when no labeled data is available. The training dataset is composed of a set of input vectors $\mathbf{U} = \{(\mathbf{x_i})\}_{i=0}^{N}$, and the objective consists of finding interesting patterns in the data. Compared to supervised learning, unsupervised learning comprises a wider range of techniques and its objective is less well defined: the models have no clear desired output nor obvious error metric (I. Goodfellow et al., 2016). However, the unsupervised learning paradigm seems to be closer to the way animals and humans learn. These algorithms also provide a cheaper framework for data exploitation, given that no data annotation is required by human experts, which is generally expensive.

Some of the most common applications of unsupervised learning are described below.

[2]Notice that we represent the output $\mathbf{y_i}$ as a vector although supervised models can be univariate. However the multivariate form is a more general case.

[3]a type of marine snail

- *Clustering*: consists of finding dissimilar subpopulations in the data (also known as clusters or groups), where the elements within a subpopulation are more similar between them than to elements in other subpopulations.

- *Density or probability mass estimation*: the machine learning algorithm is trained to learn the probability density function of the data (or the probability mass function in case $\mathbf{X}$ is discrete) $p_{model}(\mathbf{X}) : \mathbb{R}^N \rightarrow \mathbb{R}$ (I. Goodfellow et al., 2016). For this, the model needs to learn the underlying structure of the data $\mathbf{X}$. The techniques laying in this family can be used for many downstream applications, such as clustering (Yuzhong et al., 2006), missing data imputation (Ding et al., 2015) or generation (Yang & Chakraborty, 2020).

- *Manifold learning*: is a set of techniques consisting of learning the structure of high-dimensional data, where the data is assumed to lie on a low-dimensional manifold in a high-dimensional space (Murphy, 2012). The objective of these techniques is to discover latent structures in the data that can be exploited for tasks such as data compression, dimensionality reduction, feature extraction or data visualization. One example of this task could be reducing the dimensionality of a dataset using *principal components analysis* (PCA). That would project the original dataset into a linear lower dimensional one with orthogonal axes, where the structures in the data could presumably be more easily discernible.

- *Data completion*: consists of imputing the missing values of a given dataset (Buuren, 2018). This can be done with different purposes such as inferring the unfilled optional answers of a survey, or filling the gaps of a time series with low sampling frequency to get a higher time resolution representation. Some forms of collaborative filtering (Falk, 2019), for example matrix factorization algorithms (Koren et al., 2009), can also be seen as a data completion task where the algorithm needs to fill the blanks of a matrix representing the ratings of products by customers. In these cases, the algorithm needs to answer a question similar to: what would be the rating that a given customer would assign to a given product if they had the chance?

- *Associative learning*: is a type of unsupervised learning where the goal is to discover the relationships between objects in the data (S. Zhang, 2002). These relationships can be expressed in terms of associations (e.g. if A then B), correlations (e.g. A is positively correlated with B) or co-occurrence (e.g. A and B are often observed together). One example of associative learning would be applying the *Apriori* algorithm (Agrawal et al., 1996) to a supermarket database in order to discover the most interesting associations between different products with the aim of deriving attractive offers for customers, or optimize product placement to improve customer experience.

- *Generative modeling*: many forms of generative model also rely on unsupervised learning techniques (Bishop, 2011). This task consists of learning to approximate $p(\mathbf{X})$ with the objective of generating data that is indistinguishable from the original distribution. It is often done by maximizing the likelihood of the data given the model $\text{argmax}_\theta p(\mathbf{X}, \theta)$, however, in

cases where the explicit density function is not needed, other methods may apply (this topic will be covered more in depth in section 2.3.5). One example of application of these techniques would be in the field of natural language processing, where the goal is to learn a model that can generate text (Kamath et al., 2019) that is realistic and linguistically plausible (these are commonly known as language models).

2.2.3 Reinforcement learning

Reinforcement learning is a family of machine learning algorithms which, in contrast to the other types of learning, does not necessarily rely on any previously gathered knowledge about the task at hand. Instead, the reinforcement learning agents (or decision makers) learn what to do by mapping situations to actions (Sutton & Barto, 2018) so that they maximize a numerical reward metric, usually in presence of uncertainty (Haykin, 1999). For the agent to learn successful behaviors (referred commonly as policies), it needs to balance exploration and exploitation while interacting with the environment (Sutton & Barto, 2018), in simpler terms, reinforcement learning algorithms learn by trial and error.

More formally, the environment is commonly formulated as *finite-discrete-time Markov decision process* (Haykin, 1999), which can be represented as a 4-tuple: $(S,\ A,\ P_a,\ R_a)$ where S represents the state space, A is the action space, $P_a(s, s')$ is the probability of transitioning from state s to state s' after performing the action a, and $R(s, s', a)$ is the reward received when transitioning from state s to state s' after performing action a.

The objective of the learning algorithm is to build an agent such that its policy $\pi_\theta(s)$ maximizes the expected sum of discounted rewards $\mathbb{E}\left[\sum_{t=0}^{T} \gamma^t R(s_t, s_{t+1}, a)\right]$, where γ is usually a scalar number between 0 and 1. The reinforcement learning theory is originally based upon dynamic programming (Szepesvári & Bartok, 2010).

A classical example of a successful reinforcement learning application can be found in (Tesauro, 1994), where an agent is trained to play *Backgammon* game.

Detail treatment of the reinforcement learning field lies far beyond the scope of this thesis, a more detailed introduction is given in (Sutton & Barto, 2018; Szepesvári & Bartok, 2010).

2.2.4 Other types of learning

There are other learning paradigms (Raghu & Schmidt, 2020) that are worth mentioning, but either it is not clear where they lay, or they combine elements from various of the previously discussed types of learning. The following list describes the most important ones.

- *Semi-supervised learning algorithms* learn from both labeled and unlabeled data. This is beneficial in problems where it is difficult or costly to label the data, and hence the amount of labeled data is scarce (Raghu & Schmidt, 2020). One example of field where semi-supervised learning has many potential applications is fraud detection (D. Wang et al., 2020), where these cases are uncommon by nature, and difficult to spot.

- *Self-supervised learning algorithms* aim to solve what is known as a *pretext task*: a supervised problem where the data can be automatically labeled

without human intervention, without extra cost and directly from the raw instances (Raghu & Schmidt, 2020). One example of *pretext task* could be determining the missing word in a partially masked sentence, given a set of sentences extracted from a collection of books (Devlin et al., 2019), with the aim of learning latent representations of the words. Other example could be determining the degree of rotation of an image (Gidaris et al., 2018) for biasing the model towards learning the latent structure of the images.

- *Transfer learning* is a discipline solely applicable to deep learning models. This methodology consists of two steps: pre-training a model to solve a large and generic task (e.g. classify large and full-color images into 1000 categories (J. Deng et al., 2009)) and then fine-tuning the pre-trained model to solve a different target task (Raghu & Schmidt, 2020). This paradigm has a lot of benefits in multiple applications (for instance when restricted amounts of labeled data are available, or when the computational resources available are limited). As an example, the authors of (Souza & Filho, 2022) show how they got successful results in performing sentiment analysis over user reviews by using pre-trained word embeddings based on *BERT* (Devlin et al., 2019). Further details about transfer learning will be covered in the chapter 4 of this thesis.

2.3 Deep learning

Deep learning algorithms were motivated by the failure of classical machine learning algorithms on solving central problems on AI (e.g. speech recognition, object recognition, text generation, etc). These algorithms have a long history (figure 2.1 summarizes the most important events in the development process of deep learning), and have been named differently along the years: connectionist models, artificial neural networks, deep learning, etc.

Deep learning is a subfield of artificial intelligence and machine learning as shown in the *Venn* diagram of figure 2.2 (taken from I. Goodfellow et al., 2016), and provides a very flexible framework for different machine learning tasks, spanning all the aforementioned types: supervised, unsupervised, reinforcement learning and others.

2.3.1 From the perceptron to its multilayer version

This section introduces the basic feed-forward neural network, from its origin to the modern trends. The basic component of a modern deep learning model is the artificial neuron (sometimes called unit). The idea of an artificial neuron has its origin in the *McCulloch-Pitts* model from 1943, an attempt to mathematically model the functionality of a biological neuron (McCulloch & Pitts, 1943). The *McCulloch-Pitts* neuron consisted of a linear function of a set of binary inputs $\mathbf{x}$ that are multiplied by a set of weights $\mathbf{W}$ (which values are either excitatory or inhibitory, i.e. 1 or -1), the result is added together, a threshold scalar is subtracted to the result, and a sign function is applied to produce a binary output y (see figure 2.3 for a graphical description). The whole model is described in equation 2.1. This weights and the threshold were meant to be adjusted manually by an operator.

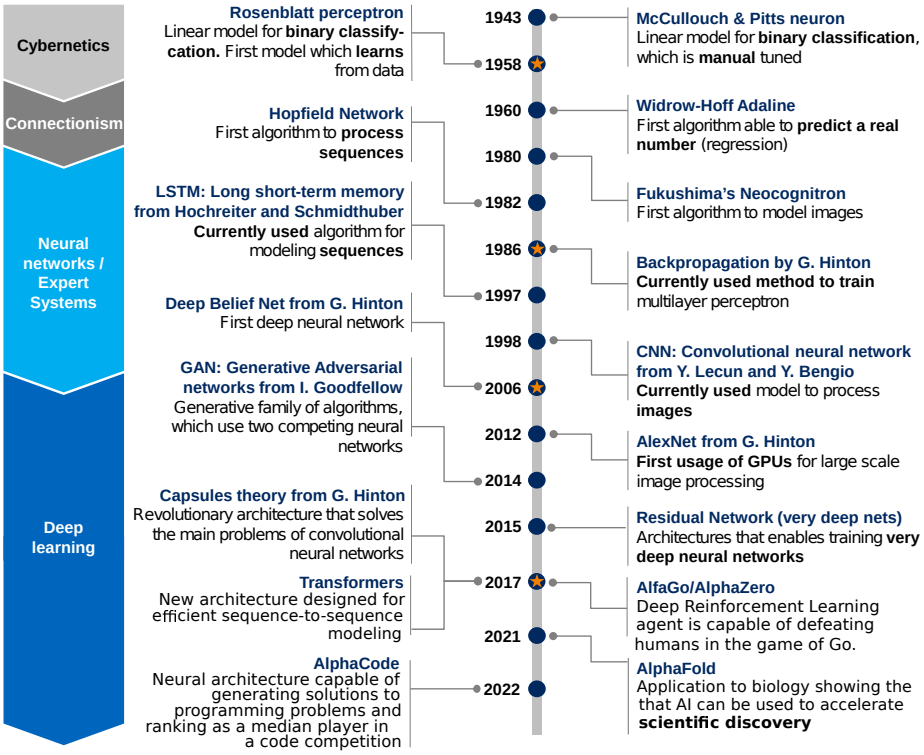

Figure 2.1: Timeline showing the most important achievements in the research of what is currently known as deep learning.

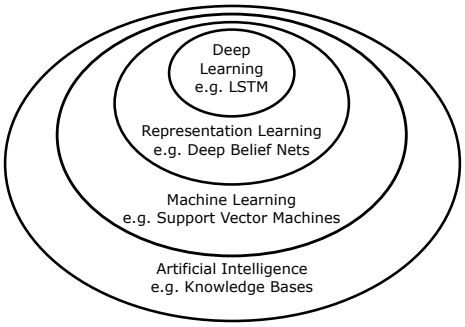

Figure 2.2: Deep learning context within the artificial intelligence field (I. Goodfellow et al., 2016)

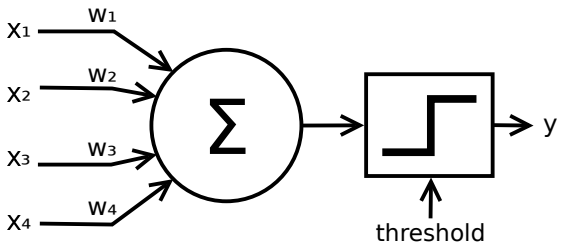

Figure 2.3: *McCulloch-Pitts* neuron model, with 4 input variables $\{x_i\}$ and one output y. $\{w_i\}$ represent the synaptic weights of the neuron.

$$y_i = \text{sgn}\left(\sum_{j=1}^{D} x_{i,j} \cdot w_j - \text{threshold}\right) \tag{2.1}$$

Some years later (1958), *Frank Rosenblatt* introduced the *perceptron* (Rosenblatt, 1958). His idea builds upon the *McCulloch-Pitts* model, proposing a simple method to automatically learn the weights of the model (see equation 2.2, where the desired response is represented by y_j, the predicted one is represented by $\hat{y}_j$ and λ is a scalar that controls the size of the weight updates, commonly referred as the learning rate). This is considered the first primitive neural network.

$$\mathbf{w_j}(\mathbf{t}+\mathbf{1}) = \mathbf{w_j}(\mathbf{t}) + \lambda[y_j - \hat{y}_j(t)] \cdot \mathbf{x_j} \tag{2.2}$$

A couple of years later, *Bernard Widrow* and his student *Ted Hoff* proposed the *ADALINE* model (ADAptive LINear Element) (Widrow, 1960), a modification of the *McCulloch-Pitts* model that removed the sign function. *ADALINE* was trained using gradient descent (Ham & Kostanic, 2000), as described in equations 2.4 and 2.5, where λ is the learning rate and N is the number of training examples (these equations are commonly known as *the delta rule*).

$$y_i = \sum_{j=1}^{D} x_{i,j} \cdot w_j + b \tag{2.3}$$

$$\frac{\partial J}{\partial w_j} = \frac{1}{N} \sum_{i=1}^{N} x_{i,j} \cdot (\hat{y}_i - y_i) \tag{2.4}$$

$$\mathbf{w_j}(\mathbf{t}+\mathbf{1}) = \mathbf{w_j}(\mathbf{t}) - \lambda \cdot \frac{\partial J}{\partial w_j} \tag{2.5}$$

The combination of multiple *ADALINE*-style perceptrons with activation functions such as the sigmoid function (see equation 2.6, where g represents a non-linear activation function), builds a *multilayer perceptron* (*MLP*). More specifically, a *MLP*, also known as *fully-connected* neural network, is a neural architecture which building blocks are perceptrons (called neurons in this scenario) which are disposed in layers so that all the elements from a layer l are connected with all the elements in the next layer $l+1$ (refer to figure 2.4 for a visual example)

$$h_i = g\left(\sum_{j=0}^{D} x_{i,j} \cdot w_j + b\right) \tag{2.6}$$

The *delta rule*, described in equations 2.3 and 2.4, built the basis for the *backpropagation* algorithm, a widely used methodology nowadays as standard method to train neural networks. The *backpropagation* algorithm (Rumelhart et al., 1986) was published by *David Rumelhart* and *Geoffrey Hinton* in 1986 as a method to optimize the parameters of *multilayer perceptrons*. This algorithm consists of two steps (Haykin, 1999):

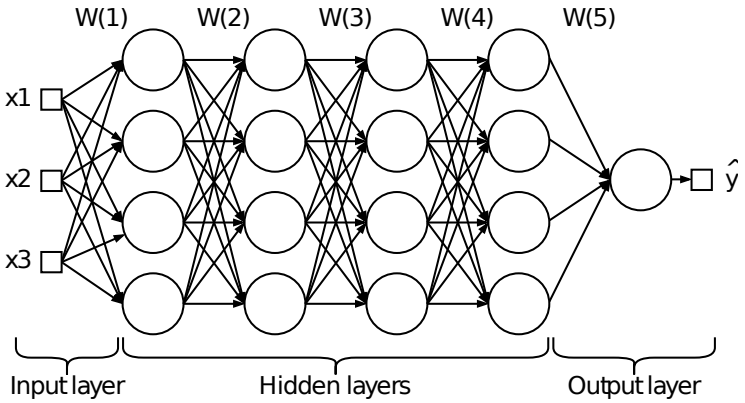

Figure 2.4: Example of *multilayer perceptron* with 3 inputs, 1 output and 3 hidden layers. Each bubble represents a neuron (figure 2.3), and has an associated bias term. Each arc represents a weight. In each neuron, the inputs multiplied by their corresponding weights are added to the neuron's bias, and then an activation function is applied to produce the output, according to equation 2.6.

1. *Forward pass*: consisting of a simple model inference operation, where a set of features $\mathbf{x}_i$ are feed to the network as input to get the output $\hat{\mathbf{y}}_\mathbf{i}$. In this phase, some of the values of the intermediate neurons can be cached to use them in the next step.

2. *Backward pass*: a metric J (often referred as cost or loss function) is used to compare the outputs of the model $\hat{\mathbf{y}}_\mathbf{i}$ with the desired outputs (sometimes called targets) $\mathbf{y}_\mathbf{i}$ and then propagate the gradient of the error backwards (from the output to the input), by using the chain rule, to adjust the weights of the model.

Before *backpropagation*, there was no algorithm for training *multilayer perceptrons* in an end-to-end manner. The only way to train those models was to fix the weights of all but one layer, and train the free one with gradient descent or other methods. These models were called feature analyzers (Rumelhart et al., 1986), and one of the most interesting examples is the *Gamba* perceptrons, described in (Minsky & Papert, 1969). Although it is out of the scope of this thesis, it may be worth mentioning that modern versions of the *Gamba perceptron* (known as *Extreme Learning Machines*) are still in the research community spectrum as alternative training methods to *backpropagation* see (G.-B. Huang et al., 2012; G.-B. Huang et al., 2006).

The introduction of *backpropagation* enabled the neural networks to learn their own hidden representations automatically, allowing for more complex and abstract models. One of the most important pieces of *multilayer perceptrons* and other modern architectures are the neuron *activation functions* (also referred sometimes as *nonlinearities*). An *ADALINE* style neuron is a linear function, and linear functions are closed under composition, therefore the composition of several *ADALINE* neurons is a linear function. To break the linearity of the neurons, the *activation functions* are introduced. They consist of non-linear functions which are applied to the output of each neuron. The authors of (Rumelhart et al., 1986) formulated the *backpropagation* algorithm with sigmoid activation functions (defined in equation 2.7), as a differentiable alternative to the

classical sign function. Later, it was discovered that unbounded and non-smooth *nonlinearities* like the *Rectified Linear Unit* (*ReLU*) (Nair & Hinton, 2010) (defined in equation 2.8) were more convenient for training deep architectures (I. Goodfellow et al., 2016). Activation functions are discussed in depth in chapter 3.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

$$f(x) = \max(x, 0) = \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.8}$$

The *backpropagation* algorithm has certain rules that need to be met (Rumelhart et al., 1986): (1) connections from higher level neurons to lower level ones are forbidden, but connections that skip layers are totally permitted, (2) the architecture must be fully differentiable to be able to backpropagate the errors and (3) the weights must not be initialized to constant values, but they must be set to random values instead, to break the symmetrical weights between layers (which would cause the optimization to stall, see (Rumelhart et al., 1986) for more details).

Despite meeting these rules, there are no theoretical guarantees for the algorithm to find the global minimum, it can get stuck in local minima. One possible way to avoid this problem consists of running the optimization several times with different random parameter initializations (Haykin, 1999). The usage of gradient-free methods such as evolutionary optimization techniques (Sivanandam & Deepa, 2008) have also been explored by the deep learning research community, sometimes leading to promising results (David & Greental, 2014; Vallés-Pérez, 2012). However, these algorithms are usually less computationally efficient than gradient-based ones, making them unfeasible when the training data or the model size are large.

2.3.2 Neural networks as universal approximators

Given any continuous function $f(x)$ with arbitrary complexity, it is always possible to find a multilayer perceptron with a single hidden layer and sigmoid activations that approximates that function to any desired degree of accuracy.

This problem was originally formulated and solved by *G. Cybenko* (Cybenko, 1989). In particular, he proved that:

Theorem 1 (2 - Cybenko, 1989) *Let σ be any continuous sigmoidal function. Then finite sums of the form*

$$G(x) = \sum_{j=1}^{N} \alpha_j \sigma(w_j^T x + \theta_j)$$

are dense in $C(I_n)$. In other words, given any $f \in C(I_n)$ and $\epsilon > 0$, there is a sum, $G(x)$, of the above form for which

$$|G(x) - f(x)| < \epsilon \quad \forall x \in I_n$$

For the sake of gaining intuition (refer to (Cybenko, 1989) for a formal proof), let G be a *multilayer perceptron* with a single hidden layer, which neurons have a sigmoid activation. Assuming the weights of the hidden layer are set to a sufficiently large number, it can be easily seen that the sigmoid activations approximate a *Heavyside step function* H (see equation 2.9, where δ represents a very large number). Then, by adding many *Heavyside* functions with the proper shift and scaling, one can easily approximate any continuous function. It can be also seen that the shift and scaling operations correspond to the bias of the neurons in the hidden layer and the weights of the output layer, respectively. See figure 2.5 for a graphical example. As it can be seen, by increasing the number of neurons one can easily control the fidelity of the approximation.

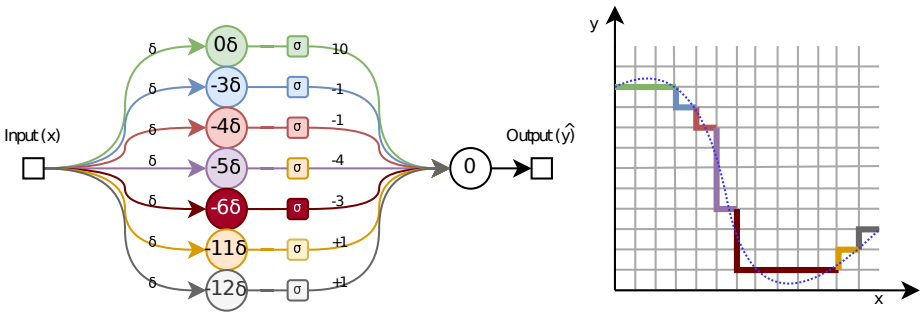

Figure 2.5: In the left side, a toy *multilayer perceptron* with a single hidden layer, a single input and a single output, and with the weights of the hidden layer set to a very large number δ. The bias terms have been indicated inside the bubbles. In the right hand side, a target function $f(x)$ to be approximated (smooth dashed blue line) and the approximation $G(x)$ (thick solid line) achieved given the weights and biases in the network of the left. The different segments of the approximation have been colored with the same color as the last neuron that fired to set that value, as the value of x increases.

$$\lim_{\delta \to \infty} (\sigma(\delta x)) = \mathrm{H}(x) \tag{2.9}$$

After *Cybenko*, other studies (Leshno et al., 1993; Pinkus, 1999) proved that the theorem holds for non-sigmoid activation functions as well. Despite the universal approximation theorem proving that a single hidden layer is enough to model any arbitrarily complex continuous function, deeper neural networks are motivated by the fact that more sophisticated functions may approximate complex problems more easily and efficiently, perhaps even needing less parameters (Nguyen et al., 2021).

2.3.3 Deeper neural networks

Regardless the emergence of the *backpropagation* algorithm, training a *multilayer perceptron* with many layers was *challenging*. The first successful methodology for training deep neural networks consisted on pre-training the weights of the network in an unsupervised fashion, using stacks of *restricted Boltzmann machines* (RBM) (Smolensky, 1986) known as deep belief networks (Bengio et al., 2006; G. E. Hinton et al., 2006). A *restricted Boltzmann machine* (initially called *Harmonium*) is a type of neural network built using a bidirectional bipartite graph architecture, with symmetric connections of neurons between the two layers and without connections between neurons within the same layer (as shown in figure 2.6). This model is trained to learn hidden abstract representations of the input, from which it is possible to recover the original probability distribution $p_\theta(x|h) \approx p(x)$. The training procedure is based on an approximate *maximum-likelihood* method called *contrastive divergence* (CD) (G. E. Hinton, 2002).

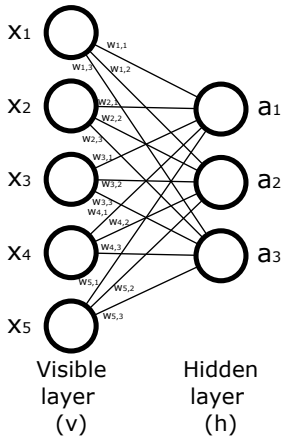

Figure 2.6: *Restricted Boltzmann machine* with 5 visible units and 3 hidden units.

Deep belief networks are stacks of RBMs that are trained using a greedy layer-wise strategy, in which the output of a trained RBM becomes the input of the following RBM (G. E. Hinton et al., 2006) (see figure 2.7). It was shown in (Bengio et al., 2006) that by following this procedure and then fine-tuning the weights of the full network using the backpropagation algorithm, deeper networks could be trained.

At the time of writing this thesis, unsupervised pre-training methods are no longer needed to train deep neural networks. This is thanks to a set of techniques that have been recently developed (in the 21st century) and that when combined together, facilitate the convergence of the *backpropagation* algorithm when used to optimize deep architectures. The first technique was the *ReLU* (Nair & Hinton, 2010) activation functions (see equation 2.8), a non-saturating alternative to the classical functions like *sigmoid* (see equation 2.7) or *tanh*, that showed to be effective at favoring sparse connectivity, and helped overcome the saturating gradients problem, a well known failure mode of neural architectures with saturating *nonlinearities* when trained by *backpropagation* (H. H. Tan & Lim, 2019). Another simple technique that helped training deep neural networks is known as *Dropout*, a regularization technique that consists of randomly zeroing

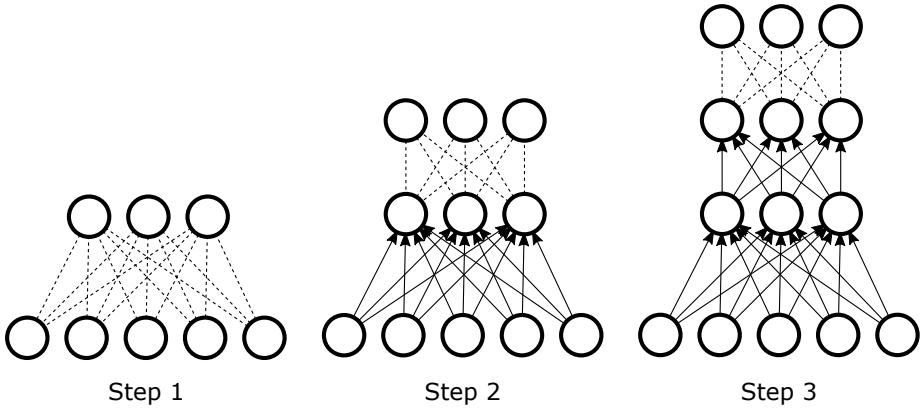

Figure 2.7: Example of *deep belief network* architecture with three feature detector layers. In each of the steps shown above, the dashed connections between units represent the *RBM* being trained, while the solid connections represent the previously trained *RBMs* that are used to compute the input of the next *RBM*.

out a fraction p of neurons from each layer in each training step (G. E. Hinton et al., 2012; Srivastava et al., 2014). These two techniques (among others) allowed *Alex Krizhervsky* and collaborators to successfully train *AlexNet* (Krizhevsky et al., 2017) without unsupervised layer-wise pre-training, a deep neural network that won the *ImageNet* (J. Deng et al., 2009) computer vision contest in 2012, a problem consisting of classifying millions of images into 1,000 categories. These techniques are still used today in the majority of the deep learning models that are published.

Other tricks that are commonly used nowadays to facilitate the parameters optimization of deep architectures are batch normalization (Ioffe & Szegedy, 2015) and residual learning (He et al., 2016).

- *Batch normalization* consists of standardizing the output vectors from hidden layers using the first and the second statistical moments (mean and variance) of the current mini-batch (Ioffe & Szegedy, 2015). This method has proved to increase the training stability when high learning rates (λ) are used (I. Goodfellow et al., 2016). Additionally, it has been shown that it provides regularization (Dauphin & Cubuk, 2021) as a side effect, due to the random fluctuations in the statistical moments from one batch to another.

- *Residual learning* consists of adding skip connections between layers of the neural network, so that the output of one layer l is fed as input to layer $m > (l+1)$. Figure 2.8 shows an example of a graph with a residual block skipping two layers. More formally, the output of the residual block becomes $\mathbf{H}(\mathbf{x}) = \mathbf{F}(\mathbf{x}) + \mathbf{x}$ where $\mathbf{F}(\mathbf{x})$ is the function learned by the composition of the two layers and the *ReLU* function[4]. Obviously one can see that $\mathbf{F}(\mathbf{x})$ is learning a residual mapping $\mathbf{F}(\mathbf{x}) = \mathbf{H}(\mathbf{x}) - \mathbf{x}$ (He et al., 2016). This method has empirically shown substantial improvements of the *backpropagation* optimization process given that the residual connections allow gradients to flow more easily, avoiding vanishing gradients (I. Goodfellow et al., 2016).

[4]notation disambiguation: $H(x)$ here does not refer to the *Heavyside* function

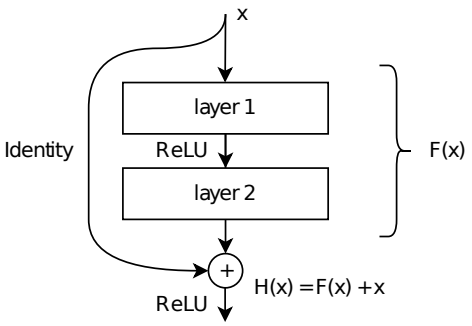

Figure 2.8: Example of a residual block.

The authors of (He et al., 2016) were able to get the first place in the 2015 *ImageNet* contest, improving the performance of *AlexNet*. Recent studies found that residual connections help reform the loss landscape leading to more convex optimization surfaces (Freeman & Bruna, 2017; L. Wang et al., 2020).

Finally, another difference between modern and classical deep learning models is the extended use of *mini-batch* stochastic gradient descent (see equation 2.12), where successive optimizations steps are performed by *backpropagation* using small (m-sized) random subsamples of the dataset named *mini-batches* (Ruder, 2016). Previous alternatives were stochastic gradient descent, where the updates are performed for every individual sample, and batch gradient descent, where the updates are performed over the full dataset $\mathbf{T}$. *Mini-batch* gradient descent has shown generalization improvements over the batch method (Hoffer et al., 2017), while being computationally more efficient than the *stochastic gradient descent* method.

$$\theta(t+1) = \theta(t) - \lambda \cdot \nabla_\theta J(X, Y|\theta(t)) \tag{2.10}$$

$$\theta(\mathbf{t+1}) = \theta(t) - \lambda \cdot \nabla_\theta J(x_i, y_i|\theta(t)) \qquad \text{where} \quad (\mathbf{x_i}, \mathbf{y_i}) \sim \mathbf{T} \tag{2.11}$$

$$\theta(t+1) = \theta(t) - \lambda \cdot \nabla_\theta J(x_{i:i+m}, y_{i:i+m}|\theta(t)) \text{ where} \quad (\mathbf{x_{i:i+m}}, \mathbf{y_{i:i+m}}) \sim \mathbf{T} \tag{2.12}$$

2.3.4 Modern architectures

In this subsection we describe three modern architectures from a general perspective: convolutional neural networks (CNN), recurrent neural networks (RNN) and transformers.

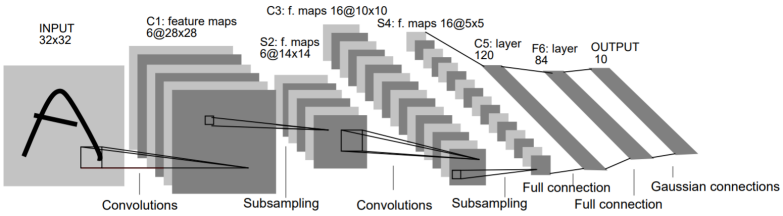

Figure 2.9: *LeNet-5* neural architecture, with 7 layers, capable of recognizing handwritten digits.

Convolutional neural networks

A CNN is a type of feed-forward neural network that is commonly used in problems where the input data have grid-like topology (I. Goodfellow et al., 2016). Common examples of these data are time-series (1D), images (2D) or videos (3D). CNNs are not new, and one of the most important primitive CNN is known as *neocognitron*, a neural architecture published by *Kuniko Fukushima*, as a model inspired in the primary cortex of the human brain that was able to recognize Japanese handwritten characters (Fukushima, 1980). This model was similar to modern convolutional neural networks, and even featured similar properties like weight sharing and translation equivariance (these properties are discussed below). The *neocognitron* inspired future works like *LeNet-5*, a 7-layer convolutional neural network (see figure 2.9) trained with *backpropagation* to recognize handwritten digits (Lecun et al., 1998).

CNNs use cross-correlation operations[5] instead of the general matrix multiplication used in fully connected networks. In this context, a convolution is a linear operation where an input $\mathbf{X}$ is correlated with a *kernel* $\mathbf{W}$ to produce a *feature map* $\mathbf{S}$ (see equation 2.13, where g represents the *nonlinearity*). The task of the algorithm is to learn the *kernel* to solve the target task (Haykin, 1999). In other words, equation 2.13 describes how the *kernel* is displaced over the input image $\mathbf{X}$ to determine its similarity with the different regions of the full-color image.

$$S_{i,j.k} = g\left(\sum_{l,m,n} X_{i+l,j+m,k+n} \cdot W_{l,m,n} + b\right) \tag{2.13}$$

The convolutional neural networks are composed (sometimes partially) of convolutional layers (where several convolutional *kernels* are applied in parallel). These layers have several properties that differenciate them from the classical dense layers, and that become advantageous when the input data can be arranged into a grid structure. These properties are discussed below (I. Goodfellow et al., 2016).

- *Sparse interactions*: the units in a convolutional network are connected to a small region of neighboring inputs. The size of that region is commonly referred as *receptive field*. This property drastically reduces the amount of parameters of the neural network, and enables parameters sharing.

[5]Formally, the operation is called cross-correlation, however they are more commonly referred as convolutions by the machine learning community.

- *Parameter sharing*: consists of using the same parameters for more than one function in the model. This is also known as *tied weights*. In a CNN, each member of the kernel is used at each position in the input (except in the special case of the boundaries, depending on the setting).
- *Equivariance to translation*: the convolution operation builds a map representing the positions where a certain feature appears in the input (e.g. a vertical border in the case of an image). If the feature is moved in the input, its representation will be moved the same amount in the output representation. Notice that the convolution operation is not equivariant to other transformations such as rotation and scaling. This lack of properties inspired the development of the *capsule networks* (Sabour et al., 2017).

Apart from the convolutions, there is another operation that is commonly used in CNNs known as subsampling. Its goal is to reduce the size of the feature maps as more layers are added, so that the representations become more generic. This helps achieve approximate invariance to translation. There are two main versions of this operation: pooling or strided convolutions. The pooling operation (I. Goodfellow et al., 2016) consists of computing a reducing statistic (e.g. the max function in max-pooling) over small neighboring regions. The strided convolutions (Ayachi et al., 2020) are standard convolutions that skip some of the inputs.

In some CNN architectures, a couple of fully-connected layers are added on top of the convolutional layers. Although, recent advances in the field have found that this is not necessary (Long et al., 2015), this pattern is commonly seen in modern architectures.

Recurrent neural networks

RNNs are one type of neural networks that are designed to process sequential data such as time-series: $x^{(1)}, x^{(2)}, ..., x^{(T)}$, where T represents the time series length. One of the most important primitive version of RNNs is known as the *Hopfield* network (Hopfield, 1982) and was published in 1982. This network incorporated a memory cell that allowed it to process sequential data. However, it was initially designed to work with binary data.

Inspired by the *Hopfield* network and its variants, the current RNNs are also incorporate memory cells (sometimes referred as the RNN internal state) that allow them to process variable-length sequences such as text sentences or audio clips (Haykin, 1999).

A RNN shares its weights across several time steps. This may sound similar to 1-dimensional CNNs (1D-CNNs), but there is one important difference (I. Goodfellow et al., 2016). In a 1D-CNN layer, each of the elements of the output sequence is function of a small number of neighboring elements in the input sequence, whereas in the basic RNNs each element in the output is function of all the previous elements in the sequence; in other words, RNNs are causal.

A recurrent neural network takes one input each time step, and then passed to a hidden layer that produces an output, see equation 2.14 for a basic example of recurrent neural network ($\mathbf{U}$ and $\mathbf{W}$ represent the trainable weight matrices, $\mathbf{b}$ is a trainable bias vector, and $\mathbf{h}$ represents the hidden intermediate representation). These hidden representations are designed to retain the important information of the previous sequence steps in order to solve the required task.

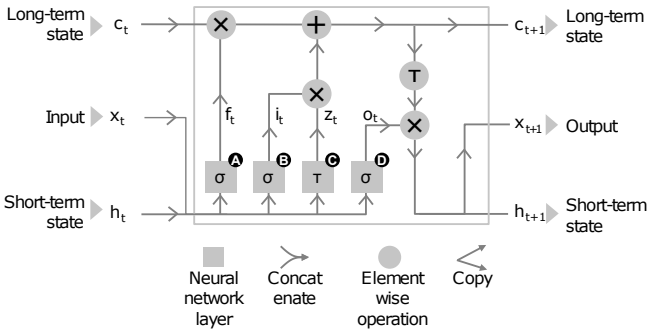

Figure 2.10: LSTM cell structure. $\mathbf{c_t}$ and $\mathbf{h_t}$ represent the long-term and short-term states, respectively, that the network uses as memory. Its operation is based on 3 gates and an output unit. (A) is a layer that acts as forget gate, represented as $\mathbf{f_t}$, which is responsible for erasing memory which will no longer be used. The layer (B) is the input gate, represented as $\mathbf{i_t}$, and controls how much input goes through the long-term line. The layer (C), represented by $\mathbf{z_t}$, controls the new contribution to the cell state that, in conjunction with the layer B form the memory addition system. The layer (D), represented as $\mathbf{o_t}$ is the output gate, which controls which values are outputted. In the diagram, σ stands for the logistic function and τ for the hyperbolic tangent.

$$\mathbf{h^{(t)}} = \mathbf{g}(\mathbf{U}\mathbf{h^{(t-1)}} + \mathbf{W}\mathbf{x^{(t)}} + \mathbf{b}) \tag{2.14}$$

The most commonly used type of recurrent neural network nowadays is the *long short term memory* (LSTM). LSTMs were published by *Hochreiter* and *Schmidhuber* (Hochreiter & Schmidhuber, 1997) in an attempt to solve the issues of RNNs when dealing with sequences in problems that required long-term dependencies. LSTM models contain two hidden state representations: one intended to retain long term memory and other for short term memory. The architecture of a LSTM cell is composed of three gates: input gate $\mathbf{i_t}$, output gate $\mathbf{o_t}$, forget gate $\mathbf{f_t}$. These gates control how the information flows through the network, allowing to write, output and delete the states as needed. This is made possible due to the sigmoid functions, which act as valves for the different operations. Figure 2.10 and equations 2.15 describe the LSTM cell more formally. In the equations, the symbol "$\odot$" refers to the *Hadamard* product, $\mathbf{U}$ and $\mathbf{W}$ are trainable weight matrices, and $\mathbf{b}$ are the biases.

$$\begin{aligned}
\mathbf{f_t} &= \sigma(\mathbf{W_f}\mathbf{x_t} + \mathbf{U_f}\mathbf{h_{t-1}} + \mathbf{b_f}) \\
\mathbf{i_t} &= \sigma(\mathbf{W_i}\mathbf{x_t} + \mathbf{U_i}\mathbf{h_{t-1}} + \mathbf{b_i}) \\
\mathbf{o_t} &= \sigma(\mathbf{W_o}\mathbf{x_t} + \mathbf{U_o}\mathbf{h_{t-1}} + \mathbf{b_o}) \\
\mathbf{c_t} &= \mathbf{f_t} \odot \mathbf{c_{t-1}} + \mathbf{i_t} \circ \tau(\mathbf{W_c}\mathbf{x_t} + \mathbf{U_c}\mathbf{h_{t-1}} + \mathbf{b_c}) \\
\mathbf{h_t} &= \mathbf{o_t} \odot \sigma(\mathbf{c_t})
\end{aligned} \tag{2.15}$$

Apart from the LSTM, there are other more modern variants that are gaining popularity. One of them is the *gated recurrent unit* (GRU) (Chung et al., 2014) a recurrent cell similar to the LSTM but more efficient and with a single state signal that showed to be as effective as its predecessor. Depending on the application, the recurrent layers can be bidirectional, allowing the network to process the sequences in a forward and backward fashion (Schuster & Paliwal, 1997).

Transformer

A transformer (Vaswani et al., 2017) is a neural architecture with encoder-decoder structure that allows mapping sequence-to-sequence (Sutskever et al., 2014) problems without the need of sequential models such as RNNs. These models make use of attention and self-attention mechanisms (Bahdanau et al., 2015) to process and align the input and output sequences, and allow processing the sequential data in parallel at training time, at the cost of a higher memory consumption[6]. The basic transformer architecture is shown in 2.11. The original proposal is defined for natural language processing tasks, however it has been shown that it can be used for other purposes with simple modifications (Bi et al., 2021; Chen et al., 2021; N. Li et al., 2019).

As opposed to RNNs, that use their state vectors to process the sequence steps in a sequential manner, transformers use *multi-head attention* (MHA) directly on the projected inputs (input embeddings). This removes the sequential dependencies of the algorithm (needing to run part of the computation graph to be able to compute the following piece) allowing parallel computation (Kamath et al., 2019). In this setting, the input and output sequences are computed using self-attention and then, the encoder and decoder vector spaces are combined with another attention mechanism.

Multi-head attention is defined as an operation over three matrices: the query $\mathbf{Q}$, the key $\mathbf{K}$ and the value $\mathbf{V}$. The name of these matrices comes from an analogy to information retrieval systems, where input queries (usually in form of a text sequence) are used to find the best matching key and value (representing a document name and content, respectively) (Manning et al., 2008). In the attention mechanism a similar process happens, where the query is compared against all the keys to produce an *attention vector* $\mathbf{a} \in \mathbb{R}^{d_{\text{model}}}$ such that $\sum_i a_i = 1$ and $0 < a_i < 1 \ \forall \ i$, which is used to weight the values corresponding to the keys (Vaswani et al., 2017). See equation 2.16 for a more formal definition. Notice that the original definition of the transformer (Vaswani et al., 2017) uses *scaled dot product* to calculate the similarity between the queries and the keys. This operation is defined in 2.18 and does not require any parameter: it is a dot-product operation normalized by the length of the sequences d_k (see figure 2.12 for a visual description). Refer to table 2.1 for alternative similarity metrics that are commonly used in the attention mechanism (Kamath et al., 2019).

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concatenate}(\mathbf{head_1}, \mathbf{head_2}, ..., \mathbf{head_s}) \tag{2.16}$$

$$\mathbf{head_i}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Attention}(\mathbf{Q}\mathbf{W_i^Q}, \mathbf{K}\mathbf{W_i^K}, \mathbf{V}\mathbf{W_i^V}) \tag{2.17}$$

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right) \cdot \mathbf{V} \tag{2.18}$$

Figure 2.11 shows how the different modules are arranged in the transformer architecture. In particular, Nx MHA blocks with residual connections form the

[6] the self-attention operation computational and memory complexity depend quadratically on the length of the sequences

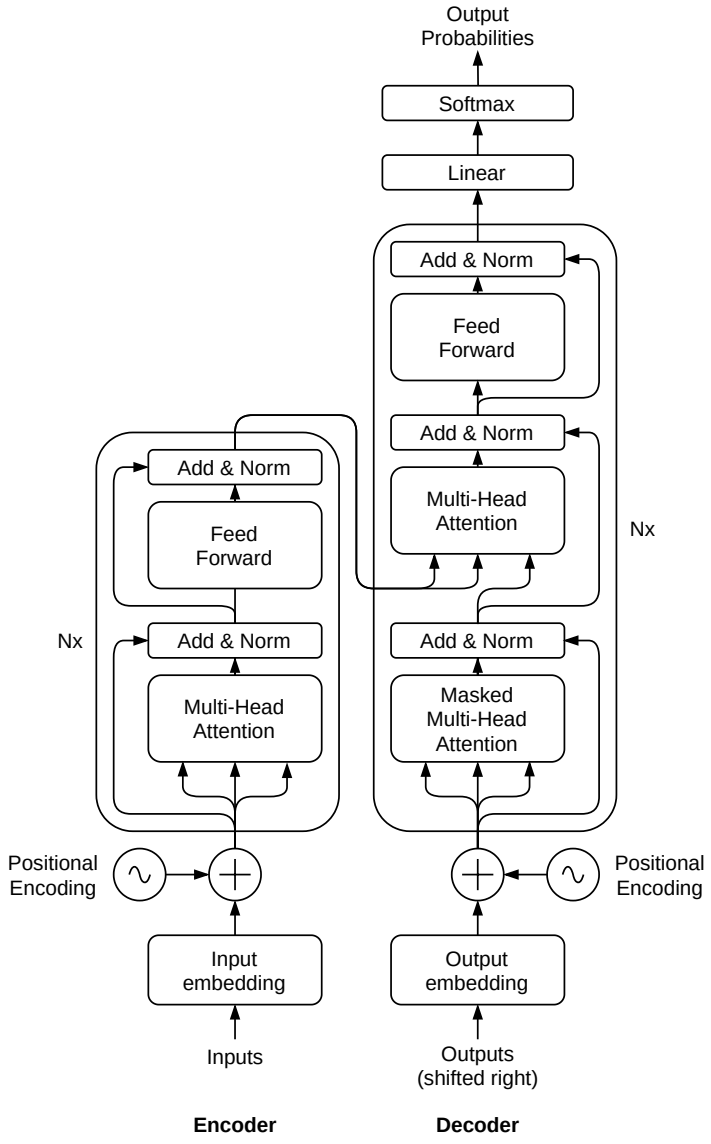

Figure 2.11: Transformer architecture as defined in (Vaswani et al., 2017). The left module represents the encoder and the right module represents the decoder. In the figure, Nx represents the number of times the encoder and decoder blocks are repeated in cascade. Typically $Nx = 8$.

Table 2.1: Attention similarity metrics (Kamath et al., 2019).

Name	Definition	Parameters	Ref
Concat	$score(\mathbf{q}, \mathbf{k}) = \mathbf{v}^\mathbf{T} \tanh(\mathbf{W}([\mathbf{q}; \mathbf{k}])$	$\mathbf{v}, \mathbf{W}$	(Luong et al., 2015)
Linear	$score(\mathbf{q}, \mathbf{k}) = \mathbf{v}^\mathbf{T} \tanh(\mathbf{W}\mathbf{q} + \mathbf{U}\mathbf{k})$	$\mathbf{v}, \mathbf{W}, \mathbf{U}$	(Bahdanau et al., 2015)
Bilinear	$score(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\mathbf{T}\mathbf{W}\mathbf{k}$	$\mathbf{W}$	(Luong et al., 2015)
Dot	$score(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\mathbf{T}\mathbf{k}$	None	(Vaswani et al., 2017)
Scaled dot	$score(\mathbf{q}, \mathbf{k}) = \mathbf{q}^\mathbf{T}\mathbf{k}/\sqrt{d_k}$	None	(Luong et al., 2015)

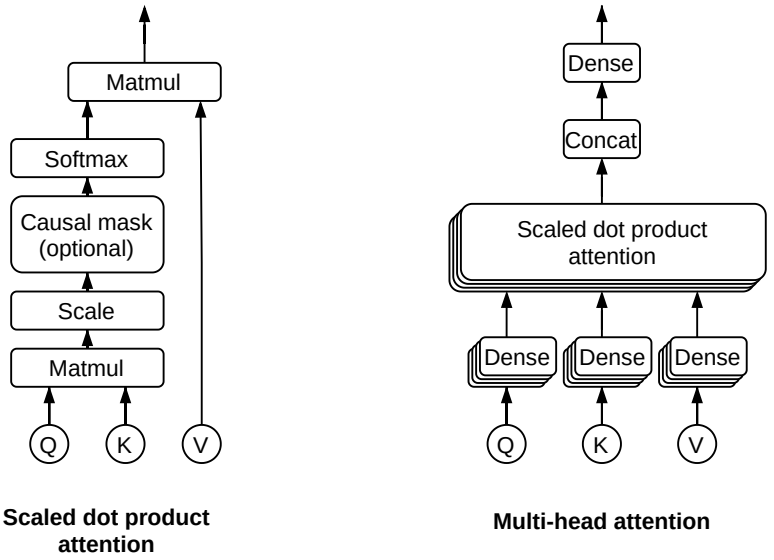

Figure 2.12: Left: the scale dot product computation graph (same as equation 2.18). Right, the multi-head attention module (see equation 2.16).

encoder and the decoder modules, and each MHA block is followed by a fully connected layer that combines the output of all the heads of the MHA. A very important detail is that the decoder self-attention needs to be *masked* so that the whole operation is causal (i.e. each output element strictly depends on the previous input elements, and not on the present or future ones) (Vaswani et al., 2017). The mask, together with the shift of the input sequence (so that the output of the transformer at time step t is calculated taking the $0..t-1$ input elements into account), allow the parallel training of the transformer.

Given that the scaled dot product operation is not location-aware, positional encodings are added to the embedding inputs, to allow the model to sort correctly the input signals if needed.

As it can be noticed in equation 2.18, the dot-product used to compute the attention similarity scores makes quadratic the memory requirements and computational cost (for the case of the self-attention) on the length of the input and output sequences. This is not a desirable property, and the authors warn about it in the paper (Vaswani et al., 2017), becoming one of the major limitations of this approach. There are already studies in the literature that discuss how to reduce that cost (Jaegle et al., 2021; So et al., 2021).

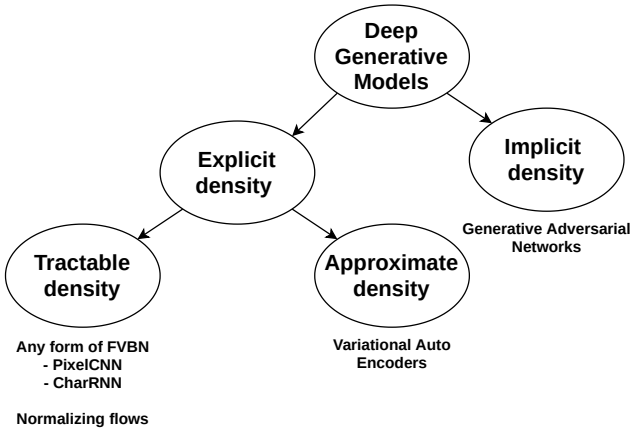

Figure 2.13: Taxonomy of the most common deep generative models.

2.3.5 Deep generative models

Generative modeling with deep learning is one of the most current trending topics in the machine learning research community. The models in this family learn the probability distribution over multiple variables of the data, in one way or another. From an intuitive perspective, to generate new data a generative model needs to grasp a deep understanding of the structure of that data distribution (I. Goodfellow et al., 2016).

The goal of the deep generative models is to learn to approximate the probability of the data (p_{data}) in an unsupervised way. In other words, the data generator needs to find θ so that $p_{\text{data}}(x) \approx p^{\theta}_{\text{model}}(x)$ given a metric of similarity (I. Goodfellow et al., 2016). The general framework consists of collecting a large amount of data samples from a specific domain and train a deep learning model that is able to generate new data that *looks like*[7] the original data.

The family of deep generative models is very diverse, and the models can be categorized into different types depending on the way they solve the generative task. Figure 2.13 shows the taxonomy of the type of models discussed in this subsection. Some of the models allow evaluating the learned probability of data $p^{\theta}_{\text{model}}(x)$ explicitly, others give an approximation for that distribution, and the others do not provide a way to interact with the probability density, yet allowing operations such as sampling from the implicit probability distribution (I. Goodfellow et al., 2016). This subsection provides a description of the following families of deep generative models: *fully visible belief networks* (FVBN), *variational auto-encoders* (VAE), *generative adversarial networks* (GAN) and *normalizing flows* (NF).

[7] we will revisit the problem at the end of this chapter.

Fully Visible Belief Networks

The *fully visible belief networks* (FVBN) are a family of probabilistic models that use the chain rule of probabilities to model the probability density of the data ($p_{\text{model}}(x) \approx p_{\text{data}}(x)$) (Smith, 2018). For that, $p_{\text{model}}(x)$ is decomposed into a set of conditional probabilities $p(x^{(t)}|x^{(1)}, x^{(2)}, \ldots, x^{(t-1)})$ that when multiplied together form $p_{\text{model}}(x)$, as shown in equation 2.19.

$$p_{model}(x) = \prod_{i=1}^{T} p(x^{(t)}|x^{(1)}, x^{(2)}, ..., x^{(t-1)}) \tag{2.19}$$

The FVBN models provide a fully tractable probability density function (I. Goodfellow et al., 2016), and are well suited for modeling sequential data such as text or speech (R. Liu et al., 2019; Shen et al., 2018; Y. Wang et al., 2017). They have also been applied to model images, by turning them into sequences of pixels (van den Oord, Kalchbrenner, Espeholt, et al., 2016; van den Oord, Kalchbrenner, & Kavukcuoglu, 2016).

The basic operation of FVBN is very simple. At training time, a model is trained to predict the next sample of a sequence $x(t)$, given the previous samples $x^{(1)}, x^{(2)}, ..., x^{(t-1)}$. This operation is known as *teacher forcing* (I. Goodfellow et al., 2016; Goyal et al., 2016; Williams & Zipser, 1989), as the model is feed with the original samples of the sequence being modeled. At inference time, a start sample $\hat{x}^{(1)}$ is provided as input for the model to predict a probability distribution $p(x^{(2)}|\hat{x}^{(1)})$ from which the next sample is drawn $\hat{x}^{(2)}$. Subsequently, $\hat{x}^{(1)}$ and $\hat{x}^{(2)}$ are feed into the model to get the probability distribution of the third sample $p(x^{(3)}|\hat{x}^{(1)}, \hat{x}^{(2)})$, from where $x^{(3)}$ is drawn. The process continues until a stop criterion is reached. The inference operating mechanism is commonly referred *free running mode* (I. Goodfellow et al., 2016), as the model is feed with previously generated samples, and does not depend on the ground truth signal. Normally, a deep learning model is used to train FVBN, in particular RNNs and transformers are some of the most common choices. Figure 2.14 shows an example of *char-rnn* (A. Graves, 2013; Sutskever et al., 2011), a FVBN used to generate natural language.

As it can be noticed, the training process, provided with the right algorithms, can be performed in parallel. However, the inference process needs to be done sequentially, given that to generate future samples, all the past samples need to have been previously generated. For this reason, FVBN are known to be inefficient at inference time when used to generate long sequences.

As an example, a shallow version of the *char-rnn* model of order[8] 100 has been trained using 800 books from the open source *Gutenberg* project[9] (Gerlach & Font-Clos, 2020). The model consists of a recurrent neural network that is trained to predict the *Multinoulli* distribution for the next character, conditioned to the previous 100 characters as input. The code of this model can be found in the repository linked in the footnote[10]. The following is an example of generation where the first 100 characters of the *War of the Worlds* book from *Herbert George Wells* are feed as input.

[8] the probability distribution of the next character is conditioned to the previous 100 characters

[9] https://www.gutenberg.org/

[10] https://github.com/ivallesp/simple_chatbot

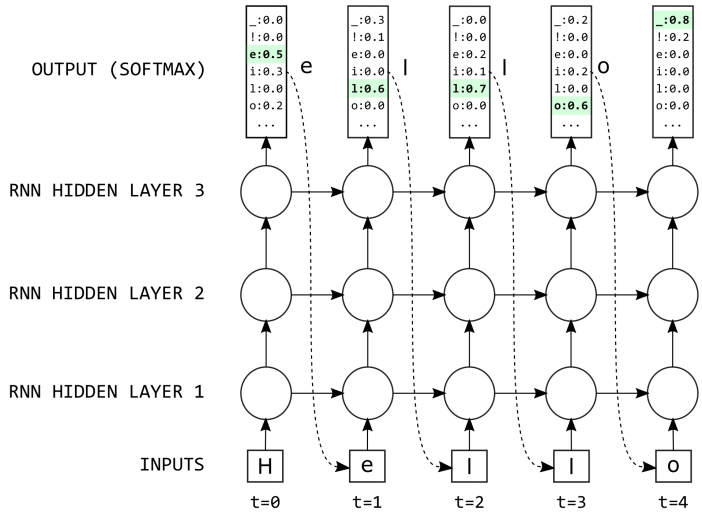

Figure 2.14: Architecture of *char-rnn*, a deep learning model used for natural language generation at character level, based on a multilayer recurrent neural network. The nodes represent RNN cells (e.g. GRU cells) and the rectangles are the inputs and outputs of the model. The dashed arrows represent the character sampling process.

> ***"No one would have believed in the last years of the nineteenth century that this world was being wat****ched together far out to the thought, 'that will cled it announced for them bloody I have last instant all the strayes as thomass us? This is, and the single War camp, until we proguted it toward the mouncin to lint of the enemy respectful, then the ribonament had been their courts and papers she been ended bent tense freely after good to the eyes to avole. The gather flooded by Wayer a great time home engaged in the exhausting day of the dutie of summers and jangers untourant altogether of mountains. But was the mystery arising half die for some regarding the raider. This top was a Catelumberhand life by the river were business, and other ebbsake and septimum at the campaich, wa, and he showed my brethren."*
>
> *- char-rnn, 2017*

It can be noticed that the model has been able to generate many grammatically correct words, however the text generated is not coherent. For that, more sophisticated approaches would be needed. *GPT-3* (Floridi & Chiriatti, 2020) is a modern example of deep language model.

Other example of modern FVBN application is (van den Oord, Kalchbrenner, Espeholt, et al., 2016; van den Oord, Kalchbrenner, & Kavukcuoglu, 2016), an extension of *char-rnn* to two dimensions, where a set of pixels of an image are given as input to the model, and its task is to predict the remaining pixels in an auto-regressive manner.

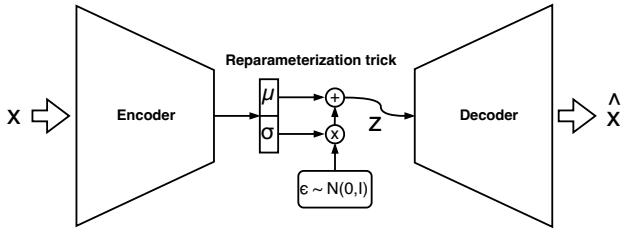

Figure 2.15: Diagram showing how a *variational auto-encoder* is structured.

Variational auto-encoders

Variational auto-encoders (VAE hereafter) are another alternative family of methods used to approximate $p_{\text{data}}(x)$ (Kingma & Welling, 2019). VAEs are the probabilistic versions of *auto-encoders* (AE). An AE is a deep learning model that is trained to reconstruct its input $\mathbf{x}$ in its output $\mathbf{y}$. The input vector $\mathbf{x}$ is transformed into a compressed representation $\mathbf{h} = f_e(\mathbf{x})$ using an encoder f_e, and that vector $\mathbf{h}$ is feed into the decoder f_d to produce $\hat{\mathbf{y}} = f_d(\mathbf{h}) = f_d(f_e(\mathbf{x}))$. The representation $\mathbf{h}$ is a vector of lower dimension than $\mathbf{x}$, and the encoder is forced to prioritize which aspects of the input should be included in $\mathbf{h}$ so that the decoder can recover $\mathbf{y} \approx \mathbf{x}$ (I. Goodfellow et al., 2016) with the minimum error. The encoder and the decoder are generally deep learning models.

The main idea of VAEs consists on replacing the $\mathbf{h}$ vector by the parameters of a distribution (υ), generally chosen to be a *Gaussian* $N(\mu, \sigma)$, from which a latent vector is sampled $\mathbf{z} \sim p(\upsilon)$. Once the model is trained, new samples $\hat{\mathbf{x}}$ can be generated by decoding samples drawn from the prior distribution $\hat{\mathbf{x}} = f_d(\mathbf{z})$ where $\mathbf{z} \sim p(\upsilon)$. The structure of this deep learning architecture is shown in figure 2.15.

VAEs provide an explicit but intractable density function which cannot be directly optimized (I. Goodfellow et al., 2016). Instead, variational *Bayes* methods are used to approximate the intractable probability distributions. In this setting, the posterior probability $p_\theta(\mathbf{z}|\mathbf{x})$ is intended to be computed. Applying the *Bayes* theorem, we can express $p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z}) \cdot p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}$. In this equation, $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}) \cdot p_\theta(\mathbf{x}|\mathbf{z})\, d\mathbf{z}$ is intractable, and hence it is not possible to optimize the parameters of a model that approximates that distribution using maximum likelihood. Here is where variational methods take place.

A known probability distribution q with parameters ϕ will be used to approximate p so that $q_\phi(\mathbf{z}|\mathbf{x}) \approx p_\theta(\mathbf{z}|\mathbf{x})$. For that, the *Kullback-Leibler* divergence (D_{KL}) metric will be used to minimize the differences between the two distributions, as shown in equation 2.20 (Kingma & Welling, 2019). $\mathcal{L}(\mathbf{x}, \theta, \phi)$, from equation equation 2.21, represents a lower bound (often referred as evidence lower bound, *ELBO*, or negative free energy), given that $D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})\right)$ is always greater or equal to zero (see the optimization function in equation 2.22). As the lower bound is tractable, the optimization problem can be approximated as shown in equation 2.23. In other words, maximizing the lower bound assures that the log-likelihood of the data is at least as large as its lower bound (Wei & Mahmood, 2021).

$$\min_{\phi} D_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x}), p_\theta(\mathbf{z}|\mathbf{x})\right) \tag{2.20}$$

$$\log\left(p_\theta(\mathbf{x})\right) = \underbrace{D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z} \mid \mathbf{x})\right)}_{>0} \ldots + \underbrace{\mathbb{E}_{\mathbf{z}} \log p_\theta(\mathbf{x} \mid \mathbf{z}) - D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})\right)}_{\mathcal{L}(\mathbf{x}, \theta, \phi)} \tag{2.21}$$

$$\log p_0(\mathbf{x}) \geq \mathcal{L}_{\theta,\phi}\left(\mathbf{x}\right) \tag{2.22}$$

$$\theta, \phi \leftarrow \arg\max_{\theta,\phi} \sum_{i=1}^{N} \left(\mathcal{L}_{\theta,\phi}(\mathbf{x})\right) \tag{2.23}$$

The first term of the lower bound of equation 2.21 ($\mathbb{E}_{\mathbf{z}} \log p_\theta(\mathbf{x} \mid \mathbf{z})$) is known as the reconstruction error, and measures how dissimilar the input and the output are. The second term ($D_{KL}\left(q_\phi(\mathbf{z} \mid \mathbf{x}) \| p_\theta(\mathbf{z})\right)$), usually referred as the divergence error, measures how dissimilar are the prior and the posterior distribution of the encoder. The probability distributions p_θ and q_ϕ are two separated deep learning models (with parameter sets θ and ϕ). Normally, the prior distribution over the latent variables $P(\mathbf{z})$ is chosen to be an *isotropic Gaussian* (Wei & Mahmood, 2021) and in order to ensure that the encoder-decoder architecture is differentiable everywhere, the *re-parameterization trick* is used as a mechanism to replace the classical non-differentiable sampling procedure $\mathbf{z} \sim N(\mu, \sigma)$ by an equivalent version $\mathbf{z} = \mu + \epsilon \cdot \sigma$ where $\epsilon \sim N(0, I)$ in which z is differentiable with respect to μ and σ (Kingma & Welling, 2019).

Generative adversarial networks

Generative adversarial networks (GAN hereafter) are a type of deep generative models first published in 2014 (I. Goodfellow et al., 2014). They consist of a dual deep learning model, where the first component, known as the generator, tries to generate realistic data samples, and the discriminator (the generator's adversary) attempts to determine if a given sample is real or generated. As an analogy, one can think of the generator as playing the role of a counterfeiter, and the discriminator playing the role of a police officer. The role of the counterfeiter is to try to fool the police officer, while the latter will try to catch the counterfeiter. Figure 2.16 shows this process graphically.

The training process of a GAN is generally a zero-sum game, where the optimization succeeds when the system reaches the *Nash* equilibrium (Nash, 1950). The goal of the algorithm is to learn to represent an estimated distribution (p_{model}) of a given data distribution (p_{data}). Moreover, GANs are designed to be unbiased (I. Goodfellow et al., 2016), in the sense that provided with enough data, a model with enough capacity and the proper learning algorithm, the true probability distribution of the data can be perfectly recovered: $p_{\text{model}} = p_{\text{data}}$. As an important practical aspect of GANs, once the algorithm has converged the generator is able to synthesize samples of p_{model}, however the learned probability density function (PDF) is implicit. GANs provide no access to the learned PDF.

In the GAN systems, the generator f_g is a deep learning model with parameters $\theta^{\mathbf{g}}$ that takes a latent vector $\mathbf{z} \sim p(\mathbf{z})$ as input, where p is a prior distribution (usually chosen to be a *Gaussian* distribution), and produces a sample $\hat{\mathbf{x}} = f_g(\mathbf{z})$. The discriminator f_d is another deep learning model with

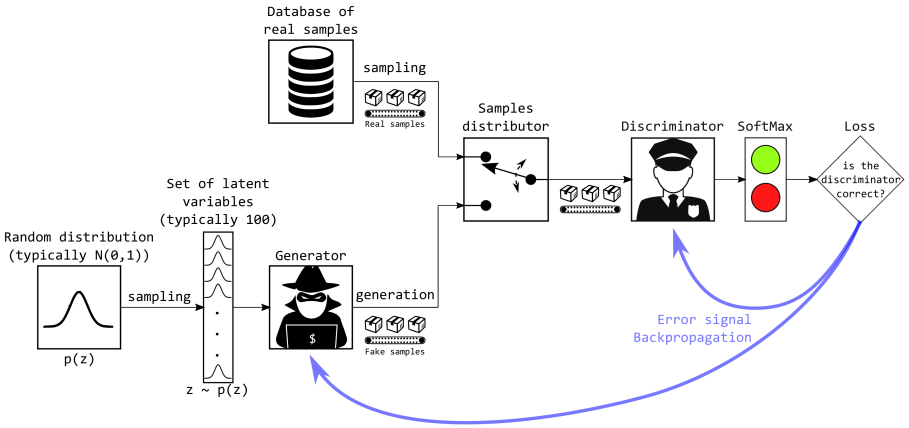

Figure 2.16: *Generative adversarial network* diagram, showing the role of the generator and the discriminator and the error signals backpropagated through the network.

parameters $\theta^{\mathbf{d}}$ that takes a sample $\mathbf{x}$ as input, and produces a binary output $f_d(\mathbf{x})$ that models the probability of the input sample $\mathbf{x}$ being real or generated (I. Goodfellow et al., 2014). In this setting, the GAN objective can be formulated as a *minimax* objective, as shown in equation 2.24. The default loss function (represented as J), is defined in equation 2.25. Unfortunately, the training objective is non-convex in θ^g and θ^d, which makes training difficult when using complex neural networks, often leading to underfitted models (I. Goodfellow et al., 2016; I. J. Goodfellow, 2016). In a simpler manner, when one of the two networks becomes too strong, the system diverges. Stabilization of the GANs remains an open problem, although some advances have been recently made (Arjovsky et al., 2017; Cui & Jiang, 2017; Z. Wang et al., 2022).

$$f_g^*, f_d^* = \arg\min_{\theta_g} \max_{\theta_d} J(f_g, f_d) \tag{2.24}$$

$$J(\theta^g, \theta^d) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log(f_d(\mathbf{x})) + \mathbb{E}_{\mathbf{x}\sim p_{model}} \log(1 - f_d(\mathbf{x})) \tag{2.25}$$

The generator and discriminator models must be differentiable everywhere, otherwise the model cannot be trained with backpropagation. As an important detail that can be noticed in equation 2.24, both players have a cost function (equation 2.25) that depends on the superset of parameters $\{\theta^g, \theta^d\}$, but every player can only change its own parameters, and not the adversary's.

Normalizing flows

Normalizing Flows are another type of deep generative models that are used to approximate data distributions p_{data}. These models describe the transformation of a prior density function $p(z)$ into an arbitrarily complex probability distribution p_{model}, through a series of invertible mappings, by repeatedly applying the rule for change of variables (Rezende & Mohamed, 2015). The output of the series of invertible functions applied to the prior distribution is still a valid probability distribution, and hence this type of models is known as normalizing flow.

A flow $f : \mathbb{R}^d \to \mathbb{R}^d$ is an invertible mapping (i.e. $\exists f^{-1}$) such that given a random variable $\mathbf{x} \sim p(\mathbf{x})$, then $f(f^{-1}(\mathbf{x})) = \mathbf{x}$. The probability density function $p(\mathbf{x})$ could be approximated by applying the rule for change of variables, as shown in equation 2.26, where $p(z)$ is known. Notice that the right hand side of the equation 2.26 comes from the application of the inverse function theorem (Rezende & Mohamed, 2015).

Invertible and differentiable flows are closed under composition (Kobyzev et al., 2021). This means that when composing multiple flows, the resulting function is still a flow. That property is very important for building deep normalizing flows capable of modeling complex data distributions such as images. It can be easily seen from equation 2.26 (Rezende & Mohamed, 2015) that given a composition of F flows $\mathbf{x} = f_F \circ ...f_1(\mathbf{z})$, the probability distribution $p_F(\mathbf{z_F})$ can be easily obtained through the application of the change of variables formula, as shown in equation 2.27 (expressed as log-probability for mathematical convenience), where h_k is the output vector of the flow f_k.

$$p_\theta(\mathbf{x}) = p_\theta(f^{-1}(\mathbf{x})) \cdot \left|\det \frac{\partial f^{-1}(\mathbf{x})}{\partial \mathbf{x}}\right| = p(\mathbf{z}) \cdot \left|\det \frac{\partial f(z)}{\partial \mathbf{z}}\right|^{-1} \tag{2.26}$$

$$\log p_\theta(\mathbf{x}) = \log p_\theta(\mathbf{z}) + \log \left|\det \frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right| = \log p_\theta(\mathbf{z}) + \sum_{k=1}^{F} \log \left|\det \frac{\partial \mathbf{h}_k}{\partial \mathbf{h}_{k-1}}\right| \tag{2.27}$$

The normalizing flow is trained by maximizing the PDF of the samples $\mathbf{x}$, which can be calculated as the probability density of the normalized samples $p(f^{-1}(\mathbf{x}))$ with a volume correction term derived from the change of variables formula, as expressed in 2.27 (Papamakarios et al., 2017). Once the model is trained, one can sample from the prior distribution $\mathbf{z} \sim p(\mathbf{z})$ and transform that latent variable into a data sample $\hat{\mathbf{x}} = f(\mathbf{z})$ (Rezende & Mohamed, 2015).

There are two main restrictions that have to be taken into account when designing neural architectures for normalizing flows. First, each flow block f_i needs to be invertible, and second, the determinant of the *Jacobian* of each flow block must be fast to compute. These two properties are absolutely necessary for being able to train deep learning based normalizing flows, and they represent the major difficulty currently under research. The most common solution at the time of writing this dissertation is to define the flows f_i as affine-coupling layers (Dinh et al., 2016). Their structure is represented in figure 2.17. More formally, given an input vector $\mathbf{x}$, of size $\mathbf{D}$, an integer $d \in [2, D-1]$, a deep learning model s and another l, the affine-coupling layers can be defined as shown in equation 2.28. It can be shown that the determinant of the *Jacobian*

of the affine-coupling layer transformation does not depend on the parameters of the networks l and s, so they can be as complex as one wants (Dinh et al., 2016). The whole operation can be seen as a scale and shift operation applied over one part of the input vector $\mathbf{x_{d+1:D}}$, and conditioned over the other $\mathbf{x_{1:d}}$. The composition of multiple affine-coupling layers with interleaved shuffling operations builds *RealNVP* (Dinh et al., 2016), a deep normalizing flow that achieved remarkable results in computer vision.

$$\begin{aligned} \mathbf{y} &= \begin{cases} \mathbf{y_{1:d}} & = \mathbf{x_{1:d}} \\ \mathbf{y_{d+1:D}} & = \mathbf{x_{d+1:D}} \odot s\left(\mathbf{x_{1:d}}\right) + l\left(\mathbf{x_{1:d}}\right) \end{cases} \\ \mathbf{x} &= \begin{cases} \mathbf{x_{1:d}} & = \mathbf{y_{1:d}} \\ \mathbf{x_{d+1:D}} & = \left(\mathbf{y_{d+1:D}} - l\left(\mathbf{y_{1:d}}\right)\right) \odot s\left(\mathbf{y_{1:d}}\right) \end{cases} \end{aligned} \tag{2.28}$$

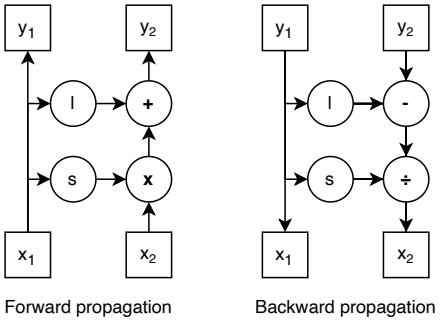

Figure 2.17: Forward and backward computation graphs for *affine-coupling layers*. Notice that $\mathbf{x_1}$ and $\mathbf{x_2}$ represent the input vector $\mathbf{x}$ split in two parts, and the affine coupling operation produces two vectors $\mathbf{y_1}$ and $\mathbf{y_2}$ of the exact same size as $\mathbf{x_1}$ and $\mathbf{x_2}$, respectively. The nodes with the letters l and s are arbitrarily complex neural networks that, conditioned to the first vector, produce scaling and translation weights to be applied over the second vector.

Other common solutions to the same problem are known as masked auto-regressive flows (Papamakarios et al., 2017), inverse auto-regressive flows (Kingma et al., 2016) or *glow* (Kingma & Dhariwal, 2018). They will not be covered here because it is out of the scope of this dissertation. More details about *glow* will be given in chapter 7.

Evaluation

One of the most important aspects of any discipline in science is the ability to properly measure the phenomenon under study, and the field of generative models is not an exception. However, measuring how well a generative model performs can be an extremely difficult task (I. Goodfellow et al., 2016). There are several approximate approaches that aim to quantify the performance of a generator (Theis et al., 2016), but we still do not have a general way to tackle this problem.

Many of the applications of generative models deal with multimedia data such as images, music, speech, etc. Our way of determining if a set of synthetic media has good or bad quality is through perceptual judgement. Generative models can be evaluated using perceptual judgement, but as any type of judgement, it may be subject to biases. To avoid a biased evaluation, the evaluation set and the jury

needs to be carefully chosen. Even if these pieces are designed carefully, there is still risk that this subjective evaluation completely fails (I. Goodfellow et al., 2016). One possible case is the hypothetical example in which a model learns to memorize the training data. In that case, the model will not be succeeding at the generation task, but a perceptual test would score the model with very high performance. Other example could be a model which is collapsed into a mode in the data. For example, given a model that is trained to generate images of cats and dogs, if the model only learns to generate high quality images of cats and never generates dogs', it may get a high score out of a subjective test, while it is completely failing to capture the variability of the data. For these reasons, it is well known that subjective quality of samples is not a reliable way of evaluating generative models (Denton et al., 2015).

Other common way to evaluate generative models consists of measuring the log-likelihood of the p_{model} over a test set, in the cases in which the log-likelihood is tractable (I. Goodfellow et al., 2016). This approach may be informative in some cases, but it is not a solution at all, as it often may not be correlated with the perceptual quality. One example could be a speech synthesis model that models the silences in the recordings with a very small variance. The probability density in those regions would be extremely high, while perceptually we would not probably be interested on accurately reproducing background noise. Additionally, the comparison of log-likelihoods of different models should be done under the same conditions. Aspects like data processing, the type of algorithm that is used to approximate the log-likelihood or the choice of the prior distribution can bias the results (I. Goodfellow et al., 2016).

This topic is still an open research problem, and many studies have already provided some insights about it (Sajjadi et al., 2018; Theis et al., 2016). However, the problem remains still unsolved.

Chapter 3

The modulus as activation function

3.1 Overview

The idea of decreasing the size of neural networks has been studied since long time ago (Frankle & Carbin, 2019; Hassibi et al., 1993; Y. LeCun et al., 1989). Many applications may require to be efficient at training or at inference times, especially the models that need to be executed on mobile and embedded devices (Ooko et al., 2021; Zapico et al., 2021). One of the most impressive examples is the wake word detection system of the *Google* assistant, which was reported to weight only 14 kilobytes in size (Warden & Situnayake, 2020).

The subfield of machine learning that researches how to minimize the computational cost is commonly known as *TinyML* (H. Han & Siebert, 2022). *TinyML* not only brings obvious energy consumption benefits but it can also be used to reduce the latency of the response of such systems because the devices would no longer need to query large cloud servers, removing the dependency on internet connections and allowing them to work offline. Furthermore, the common privacy concerns of machine learning applications (Ha et al., 2019) can be easily diluted if all the sensible data collected are processed solely into the device, and never leaves home.

This chapter introduces a new non-monotonic activation function –the *modulus*– which simplicity makes it specially suitable for *TinyML* and other low resource applications: the modulus and its derivative are simply bit-size operations. Interestingly, the gradient norm does not depend on the input, which removes some of the problems reported on other nonlinearities (Hochreiter et al., 2001; Hochreiter, 1998; Lu, 2020; Pascanu et al., 2013). The experiment results provide evidence that when using the *modulus* activation function on computer vision tasks the models generalize better than with other nonlinearities - up to a 15% accuracy increase in *CIFAR100* dataset and 4% in *CIFAR10* dataset (with respect to the benchmark activations tested).

The code used along this work has been made available in a public repository[1].

[1] https://github.com/ivallesp/abs

3.2 Introduction

As discussed in section 2.3.1, the core piece of all deep learning models is the activation function. They enable the models to produce non-linear abstract representations when applying linear transformations in cascade (I. Goodfellow et al., 2016). The most common choice in modern deep learning models is the Rectified Linear Unit (*ReLU*) (Nair & Hinton, 2010). Among other advantages, *ReLU* nonlinearities allow to train deeper neural networks (B. Xu et al., 2015).

3.3 Previous work

Apart from the *ReLU* in the last years there have appeared many alternative activation functions. The following studies represent some of the most popular examples: *Leaky-ReLU* and *PR-ReLU* (B. Xu et al., 2015), *ELU* (Clevert et al., 2016), *Swish* (Ramachandran et al., 2018), *SELU* (Klambauer et al., 2017), *F/PFLU* (M. Zhu et al., 2021), *RSigElu* (Kiliçarslan & Celik, 2021) etc. All those activation functions have the following properties in common: nonlinearity, continuity, differentiability and low computational cost (being *Swish* the most computationally expensive option). The majority of the activation functions are monotonic, *Swish* (Ramachandran et al., 2018), *GELU* (Hendrycks & Gimpel, 2016), *S-ReLU* (Jin et al., 2016) and *Mish* (D. Misra, 2020) are some of the most popular modern exceptions. Despite the large pool of alternatives, *ReLU* still appears as the default choice in many applications due to its simplicity and low computational cost (Nair & Hinton, 2010).

This chapter delves into the world of the non-monotonic nonlinearities and proposes the *modulus* function (aka absolute value) as an activation function, providing empirical evidences proving that this activation function allows deep learning models to converge to better solutions in *CIFAR10*, *CIFAR100* and *MNIST* datasets.

Compared with other modern non-monotonic nonlinearities such as *Mish* (D. Misra, 2020), *PFLU* and *FPFLU* (M. Zhu et al., 2021) or *Swish* (Ramachandran et al., 2018), the *modulus* has a very low computational cost (equivalent to *ReLU*). This property makes the *modulus* specially useful for *TinyML* (Sanchez-Iborra & Skarmeta, 2020) and hardware (J. Misra & Saha, 2010) applications.

3.4 Methods

3.4.1 Modulus activation function

This section introduces the proposed activation function, the *modulus*: $f(x) = |x|$. This nonlinearity is a continuous, piecewise-linear function consisting of an identity mapping for positive values of x, and a negative identity mapping for negative values of x. Its derivative (defined below) as well as the modulus function itself, are bit-size operations. This is extremely useful for hardware implementations. Besides, the modulus function can be expressed as $f(x) = \text{sgn}(x) \cdot x$, where $f'(x) = \text{sgn}(x)$. In practice, this means that the derivative can be calculated as part of the forward pass and cached for the *backpropagation*, reducing the total computation. This form is also specially useful in hardware

implementations, given that the whole activation function is fully represented as a bit-size operation (the sign function).

$$f'(x) = \begin{cases} 1 & \text{if} \quad x > 0 \\ -1 & \text{if} \quad x < 0 \end{cases}$$

Notice that the *modulus* function is differentiable everywhere except in $x = 0$. For practical purposes, $f'(0) := 1$ so that the derivative is defined for all the range of x values, similar to the case of $ReLU$ (I. Goodfellow et al., 2016). See figure 3.1h for a graphical representation.

The *modulus* activation function belongs, in essence, to the family of rectifier functions (Glorot et al., 2011) . In fact, it is equivalent to a *Leaky-ReLU* with $\alpha = -1$. However *Leaky-ReLU* was originally defined to take values of alpha strictly higher than 0 (B. Xu et al., 2015). Furthermore, *modulus* achieves significantly superior results than the *Leaky-ReLU*.

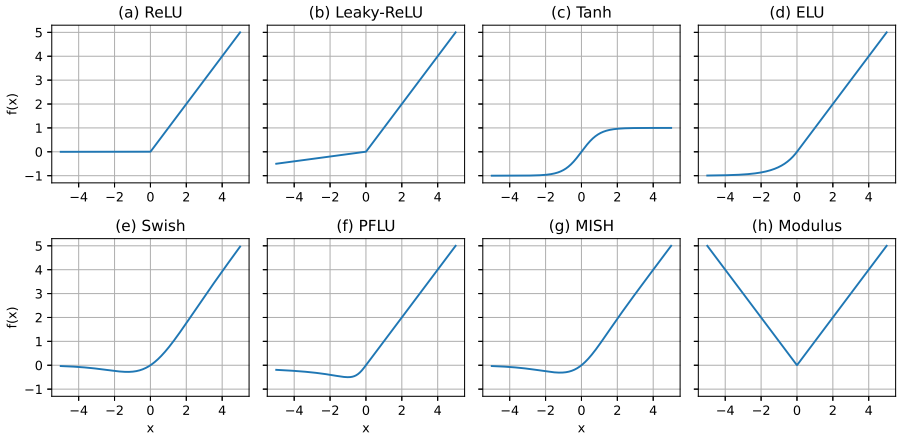

Figure 3.1: Nonlinearities used along this study. (a-g) are benchmark activation functions that showed good performance in previous studies. (h) is the *modulus* activation function proposed in this chapter. For this example, the default hyper-parameters have been used for the *Leaky-ReLU*, *ELU* and *Swish* activation functions, respectively.

The benefit of this activation function with respect to the other rectifiers is that the norm of its gradient is constant ($||\nabla_x f|| = 1 \quad \forall x$) and hence it does not depend on the input value. This property is desirable when optimizing the parameters of a neural network with gradient descent algorithms, as there are no input values for which the neuron saturates (Glorot & Bengio, 2010) (i.e. values of x for which the gradient of the activation function is close to zero). This naturally removes the dying neurons (Lu, 2020) and vanishing gradient problems (Hochreiter et al., 2001; Hochreiter, 1998; Pascanu et al., 2013).

3.4.2 Smooth approximations of the modulus function

The *modulus* function is not differentiable when $x = 0$. To study if this property harms the performance of the models in any way, two alternative smooth approximations of the *modulus* function have been tested as an additional experiment. See figure 3.2 for a visual representation. The two approximations are defined below.

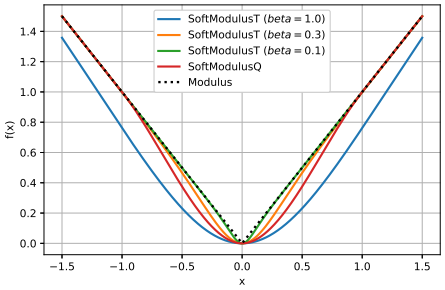

Figure 3.2: Representation of the two *SoftModulus* activation functions compared with the original *modulus*: *SoftModulusQ* and *SoftModulusT*.

Quadratic approximation

A fuzzy set approach is chosen as a method to combine the modulus function with a quadratic function (see figure 3.3) to get a smooth approximation. For that, three membership functions are defined in figure 3.4 and equations 3.1, 3.2, 3.3.

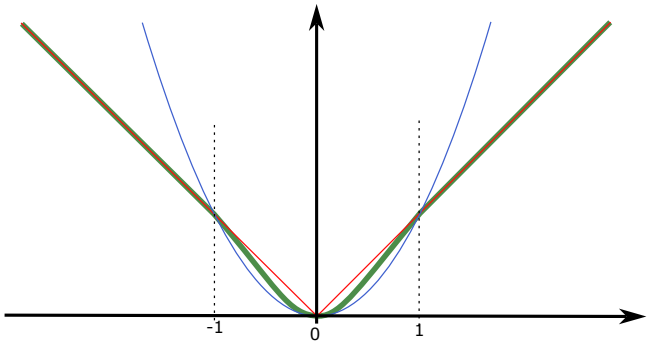

Figure 3.3: Quadratic and modulus functions, together with the combined one (in blue, red and green, respectively).

$$\mu^x_{\text{low}} = \begin{cases} 1 & \text{if} \quad x < -1 \\ -x & \text{if} \quad -1 \leq x < 0 \\ 0 & \text{if} \quad x \geq 0 \end{cases} \tag{3.1}$$

$$\mu^x_{\text{med}} = \begin{cases} 0 & \text{if} \quad x < -1 \\ x+1 & \text{if} \quad -1 \leq x < 0 \\ 1-x & \text{if} \quad 0 \leq x < 1 \\ 0 & \text{if} \quad x \geq 1 \end{cases} \tag{3.2}$$

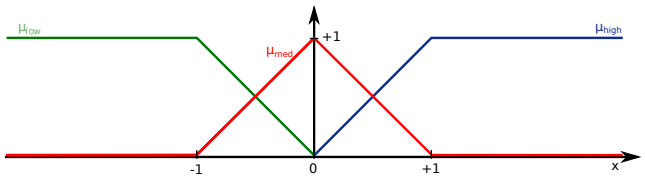

Figure 3.4: Membership functions used to combine the quadratic and modulus functions.

$$\mu^x_{\text{high}} = \begin{cases} 0 & \text{if} \quad x < 0 \\ x & \text{if} \quad 0 \leq x < 1 \\ 1 & \text{if} \quad x \geq 1 \end{cases} \tag{3.3}$$

Let $f^x_{\text{low}} = -x$, $f^x_{\text{med}} = x^2$ and $f^x_{\text{high}} = x$. These functions are combined with the membership functions to get the quadratic approximation.

- For $x < -1$: $\hat{f}(x) = \frac{f^x_{\text{low}} \cdot \mu^x_{\text{low}}}{\mu^x_{\text{low}}} = -x$
- For $-1 \leq x < 0$: $\hat{f}(x) = \frac{f^x_{\text{low}} \cdot \mu^x_{\text{low}} + f^x_{\text{med}} \cdot \mu^x_{\text{med}}}{\mu^x_{\text{low}} + \mu^x_{\text{med}}} = \frac{(x+1)x^2 + (-x)(-x)}{1+x-x} = x^3 + 2x^2$
- For $0 \leq x < 1$: $\hat{f}(x) = \frac{f^x_{\text{med}} \cdot \mu^x_{\text{med}} + f^x_{\text{high}} \cdot \mu^x_{\text{high}}}{\mu^x_{\text{med}} + \mu^x_{\text{high}}} = \frac{(1-x)x^2 + x \cdot x}{1-x+x} = -x^3 + 2x^2$
- For $x \geq 1$: $\hat{f}(x) = \frac{f^x_{\text{high}} \cdot \mu^x_{\text{high}}}{\mu^x_{\text{high}}} = x$

Combining and simplifying all the above expressions the final activation function is obtained. This approximation is subsequently referred as *SoftModulusQ*.

$$f(x) = \begin{cases} x^2 \cdot (2 - |x|) & \text{if} \quad |x| \leq 1 \\ |x| & \text{if} \quad |x| > 1 \end{cases}$$

Hyperbolic tangent approximation

Starting from the modulus function $f(x) = sgn(x) \cdot x$, the *sgn* function is approximated as follows $sgn(x) \approx \tanh(x/\beta)$, where $\beta \in [0, 1]$ is a tunable hyperparameter that controls how acute the "V" transition is. The smaller the β, the closest the approximation is to the *modulus*. A value of $\beta = 0.01$ is used along all the trained models. In the rest of the chapter, this approximation is referred as *SoftModulusT*.

$$f(x) = x \cdot \tanh(x/\beta)$$

3.4.3 Benchmark activation functions

The following activation functions have been used as a benchmark to compare the performance of the proposed one.

- *Tanh*: the hyperbolic tangent has been one of the most popular choices (together with the *sigmoid*), before *ReLU* was proposed (Y. A. LeCun et al., 2012). Among many other desirable properties, its derivative is very simple: $(\tanh x)' = 1 - \tanh^2 x$. This property was very beneficial specially before automatic differentiation tools appeared. Besides, similar to the proposed activation function, the derivative can be easily calculated during forward pass and cached for the backward pass.

- *ReLU*: this activation function was published in 2010 as an alternative to train *Restricted Boltzman Machines* (Nair & Hinton, 2010). It is defined as $f(x) = \max(0, x)$. It allowed training deeper neural networks by solving the vanishing gradient problems typically happening with saturating activation functions. As it can be seen in the formula, ReLU outputs zero if the input is negative. This brings sparsity to the non-linear representation, at the cost of potentially finding optimization problems due to the fact that the derivative when $x < 0$ is zero (dying neurons, for instance).

- *Leaky-ReLU*: the *Leaky-ReLU* (B. Xu et al., 2015) attempted to solve the problem known as *dying neurons* in the *ReLUs* by adding a small linear term in the negative side of x: $f(x) = \max(x/\beta, x)$, where β is a hyperparameter to be tuned: $\beta > 1$ and σ is the logistic function.

- *ELU*: this is a smooth version of *ReLU* (Clevert et al., 2016) that approaches asymptotically to -1 when $x = -\inf$. Unlike ReLUs, it is differentiable everywhere and can produce negative values. This activation function is formally defined as follows. α is a tunable hyperparameter.

$$f(x) = \begin{cases} x & \text{if} \quad x > 0 \\ \alpha(e^x - 1) & \text{if} \quad x \leq 0 \end{cases}$$

- *Swish*: this is one of the few non-monotonic activation functions formally published (Ramachandran et al., 2018). It is defined as $f(x) = x \cdot \sigma(\beta x)$, where β is a hyperparameter to be tuned. It gained its popularity by showing promising results in multiple applications.

- *Mish*: this is a smooth, non-monotonic and self-regularized activation function, similar to *Swish*, showing superior performance in different benchmarks (D. Misra, 2020). It is defined as $f(x) = x \tanh[\text{softplus}(x)]$, where $\text{softplus}(x) = \log(1 + e^x)$ (Dugas et al., 2001).

- *PFLU*: the Power Function Linear Unit (PFLU) is a non-monotonic activation function that showed good performance in convolutional architectures (M. Zhu et al., 2021). It is defined as follows: $f(x) = x \cdot \frac{1}{2}\left(1 + \frac{x}{\sqrt{1+x^2}}\right)$.

3.5 Experiments and results

3.5.1 Setup

For the following experiments, *CIFAR10*, *CIFAR100* (Krizhevsky, 2009) and *MNIST* (L. Deng, 2012) datasets have been used. *CIFAR10* and *CIFAR100* images are labeled into 10 and 100 classes, respectively. They are of size 32x32 and full color.

In both cases, the datasets contain 50,000 images for training and 10,000 images for testing purposes. *MNIST* images belong to one of 10 classes and are in grey scale and 28x28 size. *MNIST* comes with 50,000 images for training and 10,000 images for testing purposes.

The images of the datasets have been normalized so that the minimum and the maximum values are -1 and 1.

Four different architectures have been used to test the performance of the *modulus* activation function against the other nonlinearities. These architectures, described in Table 3.1, are inspired on the ones used in the *Lottery Ticket Hypothesis* study (Frankle & Carbin, 2019).

Table 3.1: Deep learning architectures used to experiment with different activations. In the dense layers row, the output size has been represented as C. The Fully connected architecture (*FC*) is a multilayer perceptron with 2 hidden layers + the output layer. The *Conv2* and *Conv6* architectures are shallow variants of *VGG* described here: (Simonyan & Zisserman, 2015). The last pooling layer of the *VGG-16* original architecture has been trimmed in order to allow this model to work with smaller image sizes.

Network	Fully Connected	Conv2	Conv6	VGG-16
Conv layers		64,64,pool	64,64,pool 128,128,pool 256,256,pool	64,64,pool 128,128,pool 256,256,256,pool 512,512,512,pool 512,512,512
Dense Layers	256,256,C	256,256,C	256,256,C	4096,4096,C
Filter sizes		3x3	3x3	3x3
Pooling type		max	max	max
# Parameters	269k-878k	3.3M-4.3M	1.8M-2.3M	33.6M-40.3M
Size	3.1-11MB	38-50MB	21-27MB	385-462MB

The same experimental setup has been used across models, activations and datasets. All the networks have been trained for 100 epochs, and the best accuracy of each run has been recorded. Each run has been repeated 30 times with different random weight initializations. No *dropout* (Srivastava et al., 2014) nor *batch normalization* (Ioffe & Szegedy, 2015) have been used. *Adam* (Kingma & Ba, 2014) has been used as optimizer, with a learning rate of 10^{-4}, a *gradual warmup* (Gotmare et al., 2019) during the first 5 epochs starting from 10^{-5}, and a *cosine annealing* (Loshchilov & Hutter, 2017) with a target learning rate of 10^{-6} in the last epoch. The software versions used in all the chapter were: *Python*

Table 3.2: Classification accuracy for all the datasets and activation functions tested (rows), and for all the models (columns). The results are expressed as mean ± standard deviation across the 30 random initializations. To facilitate the reading of the table, for each dataset-model combination, the results have been colored as follows: the best result (highest accuracy) has been colored in blue, the second in green, the third in yellow and the fourth in red (first > second > third > fourth). Additionally, the cases where any of the proposed activation functions achieved significantly higher accuracy (with a significance level of $\alpha = 0.05$) than the benchmarks is marked in bold. A *Wilcoxon one-sided Rank Sum* test has been used to quantify the statistical significance.

Dataset	Activation	FC	Conv2	Conv6	VGG16
CIFAR10	ReLU	54.65 ± 0.22	71.40 ± 0.26	77.09 ± 1.21	83.66 ± 0.41
	LeakyReLU	54.71 ± 0.24	71.65 ± 0.32	77.38 ± 1.13	83.98 ± 0.34
	Tanh	49.95 ± 0.23	67.46 ± 0.34	77.67 ± 0.24	79.69 ± 0.26
	Swish	55.39 ± 0.29	69.09 ± 0.27	71.66 ± 0.62	80.77 ± 0.50
	ELU	55.16 ± 0.21	69.34 ± 0.34	78.98 ± 0.33	81.21 ± 0.37
	PFLU	55.43 ± 0.30	70.34 ± 0.33	80.77 ± 0.40	81.58 ± 0.38
	Mish	55.44 ± 0.22	69.66 ± 0.35	75.66 ± 0.53	80.98 ± 0.64
	Modulus	53.97 ± 0.24	**73.93 ± 0.42**	**84.22 ± 0.29**	**84.86 ± 0.32**
	SoftModulusQ	54.07 ± 0.29	71.49 ± 0.37	**81.01 ± 1.27**	10.00 ± 0.00
	SoftModulusT	54.04 ± 0.24	**73.95 ± 0.40**	**84.36 ± 0.28**	**85.34 ± 0.36**
CIFAR100	ReLU	27.33 ± 0.25	36.72 ± 0.31	36.35 ± 0.89	44.61 ± 1.11
	LeakyReLU	27.24 ± 0.22	37.03 ± 0.40	37.15 ± 0.77	45.19 ± 1.47
	Tanh	23.63 ± 0.20	35.29 ± 0.47	42.15 ± 0.49	44.14 ± 0.37
	Swish	27.59 ± 0.25	35.20 ± 0.34	35.75 ± 0.41	46.02 ± 1.10
	ELU	27.92 ± 0.26	35.68 ± 0.31	40.74 ± 0.48	47.63 ± 0.71
	PFLU	27.73 ± 0.21	37.51 ± 0.42	42.25 ± 0.45	48.22 ± 0.63
	Mish	27.68 ± 0.25	36.04 ± 0.41	37.63 ± 0.75	48.69 ± 0.69
	Modulus	26.29 ± 0.26	**38.66 ± 0.56**	**48.73 ± 0.62**	45.83 ± 0.80
	SoftModulusQ	26.23 ± 0.25	37.48 ± 0.44	**48.16 ± 1.97**	1.00 ± 0.00
	SoftModulusT	26.32 ± 0.24	**38.69 ± 0.56**	**48.63 ± 0.83**	48.47 ± 0.68
MNIST	ReLU	98.35 ± 0.07	99.27 ± 0.04	99.53 ± 0.03	99.58 ± 0.04
	LeakyReLU	98.37 ± 0.06	99.27 ± 0.04	99.53 ± 0.03	99.58 ± 0.03
	Tanh	98.34 ± 0.07	99.06 ± 0.05	99.48 ± 0.04	99.48 ± 0.04
	Swish	98.36 ± 0.05	99.24 ± 0.04	99.52 ± 0.03	99.53 ± 0.03
	ELU	98.31 ± 0.04	99.16 ± 0.04	99.54 ± 0.03	99.54 ± 0.03
	PFLU	98.42 ± 0.05	99.21 ± 0.04	99.56 ± 0.03	99.57 ± 0.03
	Mish	98.41 ± 0.05	99.23 ± 0.04	99.56 ± 0.03	99.57 ± 0.04
	Modulus	**98.47 ± 0.07**	**99.38 ± 0.04**	**99.60 ± 0.03**	**99.63 ± 0.04**
	SoftModulusQ	**98.51 ± 0.06**	**99.37 ± 0.03**	**99.62 ± 0.03**	11.35 ± 0.00
	SoftModulusT	**98.47 ± 0.06**	**99.39 ± 0.04**	**99.61 ± 0.03**	**99.62 ± 0.03**

3.7.3, *Pytorch* 1.7.1 and *TorchVision* 0.8.2. The code used in the experiments of this chapter can be found in the url included in the footnote [2].

[2] https://github.com/ivallesp/abs

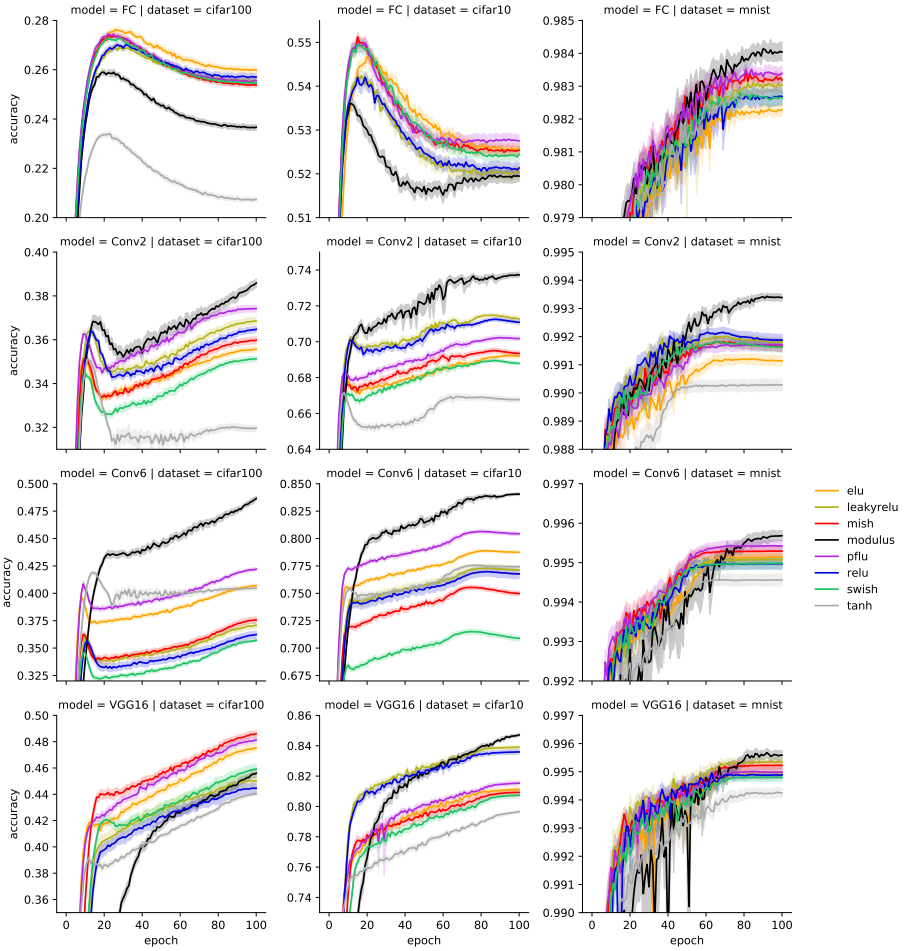

Figure 3.5: Training curves of all the models. Each line represents the average accuracy of 30 runs of a model. The shade beahind to each line shows the 95% confidence interval around the mean. Notice that the scales in the y-axis are not shared to allow for better zoom in each figure, comparing the performance across models is not the objective of this study. Details like the shades are better viewed in a screen.

3.5.2 Results

Table 3.2 summarizes the results of the classifiers for *CIFAR10*, *CIFAR100* and *MNIST* datasets. As it can be seen from the tables, the *modulus* activation function outperforms significantly the benchmark activations in 9 out of 12 experiments (4 architectures x 3 datasets). In 4 of these cases, the accuracy improvement is 3% or higher, in relative terms. The *SoftModulusT* approximation significantly outperforms the benchmark in 10 out of 12 experiments.

Figure 3.5 shows the average training curves for the 12 experiments (the curves of the smooth approximations have been included in figure 3.6 to compare them with the *modulus*). As it can be noticed in several cases (e.g. *CIFAR10-Conv2*, *CIFAR10-Conv6*, *CIFAR100-Conv6*), the accuracy of the networks with *modulus* activation function keeps increasing at the 100th epoch while the benchmarks

have already stabilized. That observation suggests that, if trained for longer, probably the accuracy would increase even more.

Table 3.3: Accuracy comparison for the soft approximations of the *modulus* function, compared with the original *modulus* definition. The results are expressed as mean ± standard deviation across the 30 random initializations. The models with highest accuracy are highlighted in bold. A star is added to those cases where a *SoftModulus* activation function achieved significantly higher results than the *modulus*, with a significance level of $\alpha = 0.05$.

Dataset	Activation	FC	Conv2	Conv6	VGG16
CIFAR10	Modulus	53.97 ± 0.24	73.93 ± 0.42	84.22 ± 0.29	84.86 ± 0.32
	SoftModulusQ	$\mathbf{54.07 \pm 0.29}*$	71.49 ± 0.37	81.01 ± 1.27	10.00 ± 0.00
	SoftModulusT	54.04 ± 0.24	$\mathbf{73.95 \pm 0.40}$	$\mathbf{84.36 \pm 0.28}$	$\mathbf{85.34 \pm 0.36}*$
CIFAR100	Modulus	26.29 ± 0.26	38.66 ± 0.56	$\mathbf{48.73 \pm 0.62}$	45.83 ± 0.80
	SoftModulusQ	26.23 ± 0.25	37.48 ± 0.44	48.16 ± 1.97	1.00 ± 0.00
	SoftModulusT	$\mathbf{26.32 \pm 0.24}$	$\mathbf{38.69 \pm 0.56}$	48.63 ± 0.83	$\mathbf{48.47 \pm 0.68}*$
MNIST	Modulus	98.47 ± 0.07	99.38 ± 0.04	99.60 ± 0.03	$\mathbf{99.63 \pm 0.04}$
	SoftModulusQ	$\mathbf{98.51 \pm 0.06}*$	99.37 ± 0.03	$\mathbf{99.62 \pm 0.03}*$	11.35 ± 0.00
	SoftModulusT	98.47 ± 0.06	$\mathbf{99.39 \pm 0.04}$	99.61 ± 0.03	99.62 ± 0.03

The results of the *SoftModulus* approximations compared with the *modulus* activation function are summarized in table 3.3. Additionally, figure 3.6 shows the training curves for the same activations. The *SoftModulusQ* approximation achieved significantly better results than the original modulus in several cases (*FC* model over *CIFAR10* and *MNIST* dataset, and *Conv6* over *MNIST* dataset); however it seems to be much more unstable: on VGG16, the gradients of the models with *SoftModulusQ* activations vanish at the beginning of the training process due to the fact that the weights are initialized very close to zero and the gradients are too small, causing numerical issues. This problem may be solved tweaking the initialiation. However, the *SoftModulusT* approximation seems to be on-par with the original *modulus* in the majority of cases, while it performs significantly better for deep architectures like *VGG16*. A plausible hypotyhesis explaining that observation is that the difference between the two approximations is due to the width of the zero gradient region around $x = 0$: wider regions lead to more training problems (see figure 3.2). The β parameter in the hyperbolic tangent approximation allows for easily adjust this zero-gradient region. Different values of beta were informally tested concluding that for high values of β (e.g. $\beta = 1.0$) the model struggles to train due to numerical precision problems (small values of weights lead to tiny gradients). A value of $\beta = 0.01$ seems to work well for all the tested experiments. Finally, the *SoftModulusT* approximation achieves superior results than the original *modulus* when used in the *VGG16* architecture.

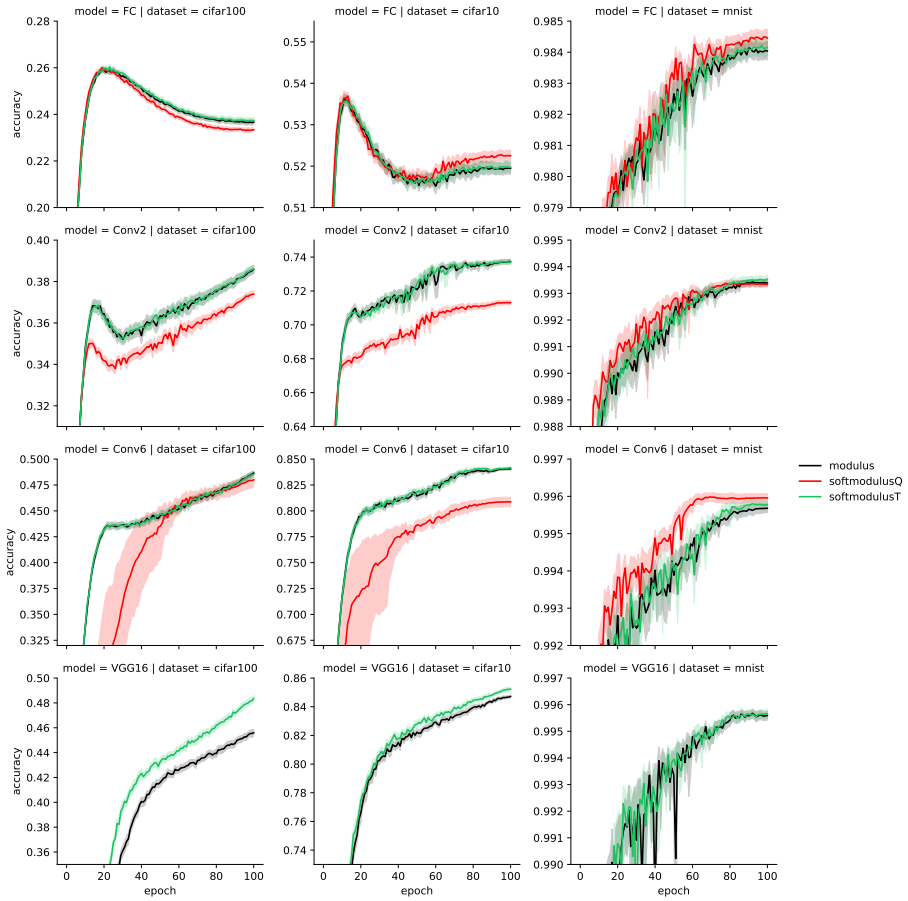

Figure 3.6: Training curves smooth approximations compared with the *modulus*. Each line represents the average accuracy of 30 runs of a model. The shade behind to each line shows the 95% confidence interval around the mean. Notice that the scales in the y-axis are not shared to allow for better zoom in each figure, comparing the performance across models is not the objective of this study. Details like the shades are better viewed in a screen.

3.6 Conclusions

This chapter provides evidences of how the proposed *modulus* activation function outperforms the benchmark activation functions in 75% of the experiments. The improvement achieved by the networks with this activation function is often significantly higher (3% or higher in 33.3% of the experiments conducted). Additionally, a smooth version of the *modulus* activation function is proposed, even performs better than the first one, at a slightly higher computational cost.

The results of this chapter motivate exploring non-monotonic activation functions as alternatives to the classical choices.

Chapter 4

Distilling the knowledge of pre-trained models

4.1 Overview

Since the 2012 *ImageNet* competition, many convolutional neural networks based architectures for computer vision have been proposed leading to a large number of open source resources in the form of pre-trained models. In this chapter, the idea of combining the knowledge of multiple pre-trained models into another simpler pre-trained model using knowledge distillation techniques is studied.

The experiments included in this chapter show improvements in the accuracy of the lightest pre-trained models when fine-tuning them using the soft-labels of their superior counterparts. The knowledge transfer is performed by using solely unlabeled data. Up to a 3.0% and a 1.7% absolute improvement in top-1 and top-5 accuracy is achieved across several models, suggesting that the trained architectures are still not at their capacity limit. This technique allows having a significant improvement of performance in mobile and other *TinyML* applications (Sanchez-Iborra & Skarmeta, 2020).

All the code used in this chapter is publicly available in a repository[1].

4.2 Introduction

Computer vision systems have evolved dramatically in the last decade due to the rise of deep learning technologies. In 2012, *AlexNet* (Krizhevsky et al., 2017) achieved the first position in the *ImageNet* yearly challenge (Russakovsky et al., 2015), and that was the first time a neural network got such position. Since then, the neural solutions prevailed, and many different architectures were proposed, each of them being better than the previous ones (Algan & Ulusoy, 2021; Khan et al., 2020). The parameters of many of the best deep learning models solving the *ImageNet* problem are publicly available (e.g. (Chollet, 2017; He et al., 2016; Howard et al., 2017; Pham et al., 2018; Szegedy et al., 2016; Szegedy et al., 2017; M. Tan & Le, 2019)), in form of pre-trained models. One of the most common applications of pre-trained models is transfer learning (Zhuang et al., 2021), as

[1] https://github.com/ivallesp/distillnet

briefly discussed in section 2.2.4, where the weights of a model trained to solve a large scale task (commonly referred to as pretext task, such as the *ImageNet* classification task) are re-utilized and fine-tuned to exploit them for another task. Usually, although not always, the first layers of the network are frozen and only the last few layers are adjusted. This simple idea has been developed in the last years, and many further studies have been published (see (Evci et al., 2022; X. Wu et al., 2021; P. Zhao et al., 2021; Y. Zhu et al., 2018)). Transfer learning works under the hypothesis that lower layers learn simpler and widely applicable patterns that are used by higher level layers to solve the objective task.

Another very powerful technique, named knowledge distillation, allows transferring knowledge from a teacher model to a student model. In 2015, a study showed that it is possible to improve the performance of a simple model (known as the student) by distilling the knowledge of a more sophisticated model (named the teacher) (G. Hinton et al., 2015). The technique proposed in the work of *Hinton et al.* is very simple and consists on training the student with the soft-targets of the teacher, given a transfer dataset. The soft-targets are the raw predictions of the model, which are expressed as probabilities of belonging to each class. The authors suggest to minimize the cross-entropy with soft-targets, after dividing the logits by a temperature constant T (process known as warming the logits, extra details in section 4.4.1) so that the probability distribution gets less sparse (i.e. more distributed across the classes that are different than the most probable class). This technique is built under the hypothesis that there is more information in the soft targets than in the hard targets (i.e. the labels). This additional information is sometimes known as *dark knowledge* (Gou et al., 2021). Knowledge distillation is still a hot topic in nowadays research (see (H. S. Lee & Wallraven, 2021; C. Tan et al., 2021; H. Zhao et al., 2021)).

This work explores the following hypothesis: the performance of a small pre-trained model can be improved by leveraging the knowledge of one or multiple larger pre-trained models, while keeping the computational budget small and using solely unsupervised data. An ensemble of pre-trained models may be the most powerful solution in terms of performance but it is obviously energy-inefficient and may not be applicable for mobile applications, tasks that require a time-sensitive inference (Sanchez-Iborra & Skarmeta, 2020) or other *TinyML* tasks. Therefore, the problem is approached from a knowledge distillation perspective, with the aim to distill the knowledge of multiple heavy pre-trained models into a light-weight base model. The experiment results show that not only it is possible to improve its original accuracy but that some techniques for combining the knowledge of multiple teachers are better than others.

There is a plethora of applications that can benefit from the results of this study. The first one is data intense applications such as video event detection (Chakraborty et al., 2021), for instance in autonomous driving (Swaminathan et al., 2019), where a large number of frames need to be processed in real time and with a high performance. The second one is tasks that need to run necessarily in a low-resource device, like a mobile device (e.g. augmented reality for map localization (Limmer et al., 2016)), or the time-sensitive tasks (e.g. facial recognition for security (Aung et al., 2021)). Finally, the proposed models are more environmental-friendly, as they require less energy for training and inference (C.-J. Wu et al., 2021).

4.3 Previous work

A few previous studies have been published where the objective was to combine multiple deep learning models. The authors of (L. Liu et al., 2019) distill the knowledge of several teachers into a multitask student in the computational linguistics domain. Apart from the different domain of application, their approach differs from the one described in this chapter in the fact that their student and teachers goals differ: their student learns to combine the different objectives of the teachers. In (Geyer et al., 2019), the authors present a new technique that allows merging multiple models using a parameter fusing technique based on the incremental mode matching (IMM) procedure (S.-W. Lee et al., 2017). This methodology has an important limitation that makes it unsuitable for this use case: the pre-trained and the target architecture must have the same structure. The objective is to improve the performance of a light-weight model using the knowledge of its greater siblings, which normally have more complex architectures.

In the work of (Asif et al., 2020), the authors define a framework to learn a small ensemble of students from a large ensemble of teachers which are combined linearly. For that, they propose to define a neural network architecture with as many student branches as teachers. The student branches are trained to minimize their *Kullback-Leibler* (KL) divergence with their corresponding teacher branch, as well as minimizing the KL-divergence between the linear combination of the students, and the linear combination of the teachers. The approach proposed in this chapter differs fundamentally in the fact that the base architecture size of the proposal is independent of the number of teachers. In addition, instead of using KL-divergence losses, the cross-entropy with soft-targets is used, as defined by (G. Hinton et al., 2015).

Additionally, multiple works have been found in the literature where the objective is to design lightweight convolutional neural networks for multiple tasks (Hui et al., 2018; Jeon et al., 2021; Zhou et al., 2020). Different techniques can be used to lighten up the high computational cost and memory requirements of the classical convolutional neural networks architectures. These techniques range from efficient modifications of the convolution operation, such as depthwise-separable convolutions (Chollet, 2017), squeeze operations (Qiang et al., 2021) or dilated convolutions (Yu & Koltun, 2016), to higher level architecture design choices such as systematically balancing the size of the architecture building blocks (M. Tan & Le, 2019) or pyramidal feature extraction (T.-Y. Lin et al., 2017), typically applied in object detection or optical flow inference. In this work, given the restriction of using publicly available student models pre-trained on *ImageNet* dataset, the architecture design is limited to the current available choices, hence innovative architecture design is out of the scope of this work. However, the chosen student architectures feature various of the efficiency techniques discussed in the literature, such as 1x1 convolutions (Szegedy et al., 2017), depthwise-separable convolutions (Chollet, 2017), efficient model width, height and resolution scaling (M. Tan & Le, 2019), and compression-expansion blocks (Howard et al., 2017; Sandler et al., 2018), among others.

4.4 Methods

This section describes the training methods used as well as the knowledge distillation framework and the techniques employed to blend the predictions of different teachers.

4.4.1 Knowledge distillation

The original knowledge distillation method, as defined by (G. Hinton et al., 2015), consists on using the C class probabilities (known as soft-targets) produced by a machine learning model (named the teacher) as the training objective for another, often simpler, machine learning model (named the student). The probabilities of a deep learning model $\mathbf{p_i}$ (where $i \in \{1, ..., C\}$) are normally calculated by applying the *softmax* function over the logits vector $\mathbf{z_i}$ (I. Goodfellow et al., 2016). Usually, a temperature parameter T is introduced with the aim of producing a softer[2] probability distribution over the classes (see equation 4.1, where i and j represent class indices, and t represents a specific teacher).

$$p_{i \in \{1..C\}} = \frac{\exp(z_{ti}/T)}{\sum_j^C \exp(z_{tj}/T)} \tag{4.1}$$

The authors of (G. Hinton et al., 2015) also recommend combining two loss functions: cross-entropy with soft-targets $\mathcal{L}_S$ and cross-entropy with hard-targets $\mathcal{L}_H$. The first objective is computed with the probabilities of the student $p(\mathbf{z_s}, T)$ against the probabilities of a teacher model $p(\mathbf{z_t}, T)$, where the temperature T is artificially increased to inflate the weight of the *dark knowledge* (G. Hinton et al., 2015). The second objective is computed using $T = 1$ on the probabilities of the student $p(\mathbf{z_s}, T = 1)$ against the ground-truth label y. See the losses definitions in equations 4.2 and 4.3, where $\mathbf{z_t}$ and $\mathbf{z_s}$ denote the logits for the teacher and the student models, respectively, and $\mathbf{y}$ denotes the ground truth label. The losses are combined as shown in equation 4.4, where the α parameter is intended to balance between the two losses (Gou et al., 2021).

$$\mathcal{L}_S \left[p(\mathbf{z_t}, T), p(\mathbf{z_s}, T)\right] = -\sum_i p_i(z_{ti}, T) \log\left(p_i(z_{si}, T)\right) \tag{4.2}$$

$$\mathcal{L}_H \left[\mathbf{y}, p(\mathbf{z_s}, T = 1)\right] = -\sum_i y_i \log\left(p_i(z_{si})\right) \tag{4.3}$$

$$\mathcal{L} = \alpha \mathcal{L}_S + (1 - \alpha)\mathcal{L}_H \tag{4.4}$$

Along all the chapter, α is set to 1 given that only unlabeled data is used for training, and hence no hard targets are available.

[2] Closer to a uniform distribution.

4.4.2 Teachers blending methods

Different models may produce harder or softer posterior probability distributions. In order to be able to successfully combine different teacher models for building the teacher signal that will be distilled into the student, the posterior probability distributions needs to be calibrated.

The hardness or softness of a distribution can be controlled with the temperature parameter T (see equation 4.1). In this case, as different models have been combined to build the soft target signal, the temperature of each model has been chosen so that the average probability of the most probable class becomes S. Hence S represents a hyperparameter that must be chosen before training. The term s is used to denote a specific choice of S. T has been determined independently for each pre-trained model so that $S = s$, on average over the transfer dataset. The value of T has been found using the bisection method[3], given that the equation 4.1 is continuous in T.

Once the probability distributions have been calibrated, the following techniques have been defined as teacher blending proposals. The term θ_n is used to denote the parameter set of the teacher n.

- *Mean*: following the same idea of ensemble models (Kuncheva, 2004), averaging all the teachers probabilities is proposed as the simplest method to combine all teachers knowledge. This is done by computing the arithmetic mean across the set of N teachers, for every instance x and class c.

$$p_{\text{mean}}(x, c) = \frac{1}{N} \sum_{n=1}^{N} p(c|x, \theta_n)$$

- *Median*: with the idea of improving the mean blending method with robust statistics, the same calculation is repeated but using the median across the set of N teachers, for every instance x and class c. The result of this operation needs to be normalized so that the probabilities across the C different classes sum to 1.

$$p_{\text{median}}(x, c) = \frac{\text{median}_n \; p(c|x, \theta_n)}{\sum_k^C \text{median}_n \; p(k|x, \theta_n)}$$

- *Random*: randomly choosing the teacher that provides the soft-target for each training example, in expectation, should lead to the mean of the teachers predictions. However, given that stochastic gradient descent based optimizers are used, this method is included with the hypothesis that it will increase the variability of the gradients, potentially leading to a more robust exploration. For that, the class probabilities for each instance x are selected from a random teacher $\hat{t}$. The randomization is recalculated after every training epoch.

$$\hat{t} \sim U(1, T) \rightarrow \mathbf{p}_{\text{random}}(x) = \mathbf{p}_{\hat{\mathbf{t}}}(x)$$

- *Maximum correlation*: selecting the teacher that has the maximum correlation with the rest of teachers for a given training example will potentially

[3] $T = 1$ and $T = 6$ are used as starting points for the bisection method

lead to a more refined training signal. To account for that, the *Pearson*'s correlation coefficient between all the possible pairs of teachers (correlation matrix) is calculated, for every training example. Then, the average of the correlation coefficients of every teacher with the rest is calculated, in order to determine to what degree that teacher "agrees" with the rest for that given training example.

- *Minimum entropy*: the probability distributions of the teacher predictions for a given training example can be more or less spread. One possible interpretation of that phenomenon is that the higher the spread of the output classes distribution is, the lower the confidence of the model prediction will be. With that assumption in mind, the use of the *Shannon* entropy is proposed as a measure of the spread (the higher the entropy, the closer the class distribution will be to a uniform distribution), and hence choose, for every training example, the teacher which class probability distribution has the lowest *Shannon* entropy. That teacher will be the one that provides a more confident prediction, under the mentioned assumption.

The techniques described above can be categorized in aggregative and selective technique depending on their operating principle. The "mean" and "median" techniques are considered aggregative techniques because for a given training example x_i, they merge information from different teachers to get the final soft-target probabilities. The "random", "maximum correlation" and "minimum entropy" are categorized as selective techniques given that for a training example x_i, they just select which teacher provides the soft-target signal.

4.4.3 Pre-trained models

Some of the pre-trained models included in the *Keras* library for *Python* (Chollet et al., 2015) have been used along this study. One model from each family is selected, excluding the VGG group in favour of more modern and efficient alternatives. A short description about each of those architectures is included below. Further details, including the number of parameters and the computational cost of each model, are included in table 4.1.

1. *ResNet* (He et al., 2016): convolutional neural network with multiple blocks where the output of the l^{th} layer is added to the output of the $(l+1)^{th}$. This structure is known as *residual connection* (or *skip connection*), and leads to the following transition: $x_{l+1} = H_{l+1}(x_l) + x_l$, where H represents a convolutional layer. Described in the section 2.3.3.

2. *Inception ResNet* (Szegedy et al., 2017): introduction of the *residual connection* structure from (He et al., 2016) to the classical *inception* convolutional neural network model, combined with several efficiency tricks. The *inception* model is based on the idea of using different convolution operations (with different receptive field sizes and pooling operations) in every layer, and concatenating the result together. Additionally, before each regular 2D convolution, a 1x1 convolutional layer is introduced to reduce the number of operations (see figure 4.1). The authors of (M. Lin et al., 2014) provide a deeper study on 1x1 convolutions.

3. *DenseNet* (G. Huang et al., 2017): convolutional neural network inspired by the *ResNet* model (He et al., 2016) that introduces direct connections from every layer to all the subsequent ones leading to the following transition: $x_{l+1} = H_l([x_0, x_1, ...x_{l-1}])$. The square brackets in the previous expression mean concatenation.

4. *NASNet* (Pham et al., 2018): convolutional neural network designed using neural architecture search (NAS) with reinforcement learning algorithms. The final architecture structure resembles to *inception ResNet*, but has been optimized to have a higher inductive bias.

5. *Xception* (Chollet, 2017): convolutional architecture inspired in *inceptionV3* (Szegedy et al., 2016) that features *depthwise-separable convolutions* for higher computational efficiency.

6. *MobileNet V1* and *V2* (Howard et al., 2017; Sandler et al., 2018): convolutional architecture designed to be efficient and scalable with the objective of being implemented into mobile devices. These networks feature *depthwise-separable convolutions* for reducing the number of parameters, compression-expansion blocks and the introduction of two parameters α and ρ, to control the depth of the network and the input image resolution, respectively.

7. *EfficientNet* (M. Tan & Le, 2019): highly scalable convolutional architecture that attempts to tie the network depth, width and the input image resolution together into a compound single parameter referred as ϕ. The architecture of the base model (aka *EfficientNet*) has been designed using neural architecture search (NAS) techniques.

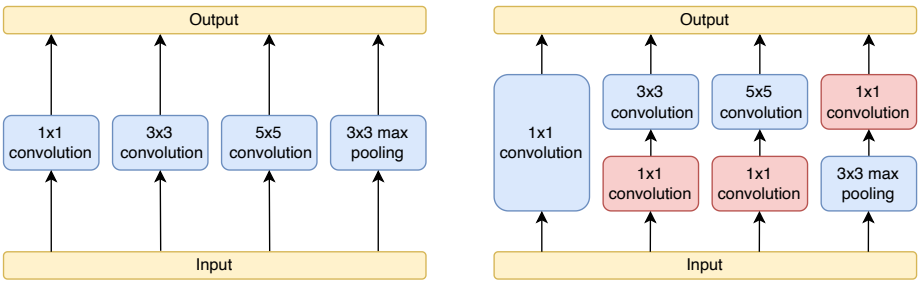

Figure 4.1: Building block of inception architecture (Szegedy et al., 2015). Left: inception block without the dimensionality reduction trick. Right: inception block with convolutions 1D to reduce the total amount of operations.

Table 4.1: Pre-trained models used as teachers along this study, taken from *Keras* implementations. Some of the input resolutions shown in the table may not correspond to the resolutions of the original papers, the models have been run in the resolutions indicated in the table as they showed a substantial performance improvement. The performance metrics reported in the table are empirical accuracies obtained by measuring the performance of the models against the *ImageNet* 2012 validation dataset, using the *Keras* implementations. They may differ from the performance reported in the original studies. *m* means millions (10^6) and b means billions (10^9). The models that were used as students are formatted in bold.

Model	Input size	#Params	FLOPs	Top-1 acc.	Top-5 acc.
NasNetLarge	331x331	89m	24b	82.44%	96.02%
InceptionResNetV2	299x299	56m	13b	80.44%	95.31%
Xception	299x299	23m	8.4b	78.92%	94.47%
EfficientNetB7	256x256	67m	37b	77.88%	93.87%
DenseNet201	256x256	20m	8.8b	77.75%	93.83%
DenseNet169	256x256	14m	6.7b	76.60%	93.39%
ResNet50	256x256	26m	4.1b	75.53%	92.53%
DenseNet121	256x256	8m	5.7b	75.47%	92.68%
EfficientNetB0	256x256	5m	0.4b	75.17%	92.34%
MobileNetV2	256x256	4m	1.3b	73.11%	91.29%
MobileNetV1	256x256	4m	2.3b	71.73%	90.44%

4.5 Experiments and results

This section provides an overview of the dataset that has been used in the experiments included in this chapter, describe the experiments performed and show the results achieved.

4.5.1 Setup

The *ILSVRC2012 ImageNet* dataset has been used in all the experiments (Russakovsky et al., 2015), given its large popularity and numerous available benchmarks. It is composed of 1.3 million images, each of which belongs to only one class out of 1000 available classes. The data is provided in three separated subsets: train, validation and test, with 1.2M, 50,000 and 100,000 images in each set, respectively. The ground truth labels are provided for the train and validation sets, but not for the test set (i.e. the test set is unlabeled).

The original dataset comes with images at different sizes and aspect ratios. Three different sets with the following sizes have been generated: 256x256, 299x299 and 331x331 (see table 4.1 for the detailed model input sizes specifications). For that, first the images were resized so that their short edge matches the desired size and then center-cropping has been applied to get a square image, as it is common in these cases. Pixel values centering and scaling have been applied as per the functions provided with *Keras* along with each pre-trained model. No data augmentation has been used along this study.

In this work, the unlabeled test set is used as transfer dataset and the original validation set to measure and report performance (leaving 5% out of the latter for development purposes).

Several experiments have been conducted to prove that the methodology described in this chapter scales to different settings. In those experiments, the following factors have been varied:

- *Student*: different small models are used as students to analyze how the methodology works with different students. The models used as students are: *EfficientNetB0*, *MobileNetV2*, *DenseNet121* and *Xception*. The reason those models are chosen as students and not the strictly smallest models is to favour variability in the architectures (e.g. MobileNetV1 and MobileNetV2 are very similar architectures, so only one of them was chosen).

- *Teachers set*: according to the performance shown in table 4.1, different sets of teachers are selected to build the distillation target: the "best" teacher (i.e. *NasNetLarge*), the "3-best" teachers (i.e. *NasNetLarge*, *InceptionResNetV2* and *Xception*) and "all" the 11 teachers.

- *Teachers blending methods*: the different sets of teachers are combined with the methods described in the section 4.4.2. These methods are referred in the tables below as "mean", "median", "random", "maximum correlation" and "minimum entropy". In the results section, the "best" teacher set is treated as a blending method[4], to facilitate the reader to compare how different blending methods compare with just taking the best teacher as target (i.e. the *NasNetLarge* model).

The set of experiments conducted in this study is represented in the schema of figure 4.2.

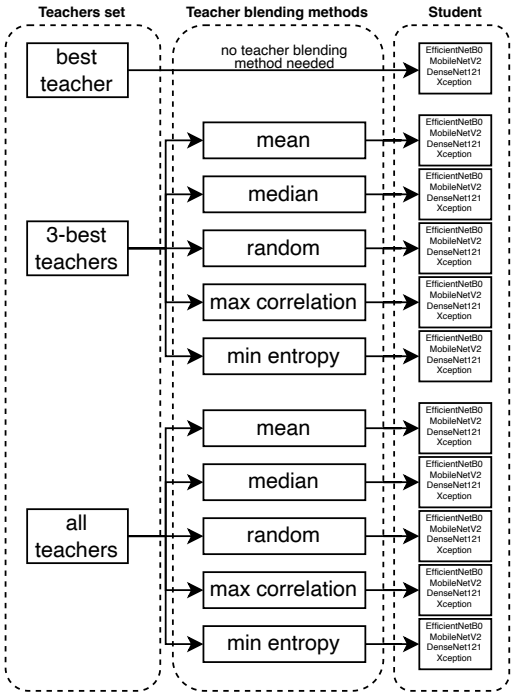

Figure 4.2: Schema representing the combination of techniques tested in this study.

[4]indeed, it can be seen as a special case of a selective blending method where the best teacher model is always selected

Each experiment has been repeated 5 times to add more robustness to the results, totalling to 220 trained models. The students' initial weights have always been fixed to the pre-trained weights for the *ImageNet* classification task, across all the experiments. During training, no layers have been frozen, allowing the optimizer to fine-tune all the weights in the student network. The temperature of the softmax in the output of the last layer has been increased in all the cases to match the normalized soft-targets distribution, as described in section 4.4.2. Each model has been trained for 100 epochs, and the results are presented in the next section at the early stopping point (as per its performance in the development set). An *Adam* optimizer (Kingma & Ba, 2014) and a learning rate of 10^{-6} have been used along all the experiments.

In the experiments the hyperparameter $S = 0.35$ is fixed, and the value of T is determined for every pre-trained model using the bisection method with the transfer dataset. This value of S has been chosen so that it produces an average temperature across the models of 2.0, which is close to the range recommended in (G. Hinton et al., 2015). Doubling the temperature parameter showed no noticeable improvements.

To run the experiments, a single *Nvidia RTX 2080ti* GPU has been used, and the code has been implemented in *Tensorflow 2.0*. The repository of code is public and can be found following the URL of the footnote[5].

4.5.2 Results

Table 4.2 summarize the results achieved by each of the experiments run over the *ImageNet* dataset. The metrics reported are the top-1 accuracy and the top-5 accuracy, as it is standard in *ImageNet* benchmarks, at the early stopping epoch. The results are expressed as the average metric ± the standard error.

As it can be seen in the table, the proposed methodology achieves an absolute accuracy increase over the student's original top-1 accuracy of up to +3.10%, +1.01%, +0.97% and +0.38% for the *EfficientNetB0*, *MobileNetV2*, *DenseNet* and *Xception* students, respectively.

The experiments showed that building a teacher signal by combining "all" the pre-trained models, or combining the "3-best" ones, leads to similar results. However, in some cases the results of distilling the combination of "all' the teachers is superior than the "3-best" (e.g. *DenseNet121* as student), but also the contrary occurs (e.g. *EfficientNetB0* as student). Additionally, using only the "best" model as teacher often leads to overfitting issue, as it can be seen in figure 4.3 (notice how the accuracy quickly degrades after reaching the maximum).

Regarding the teacher combination method, the results show that the "median" method gets, in general, worse results than the "mean" method. Besides, the "min entropy" method achieves many of the best results when applied over the 3-best teachers (in 3 out of 4 student models as per the top-1 accuracy), but it is the winner blending method only once when applied with "all" the teachers. The results of the "max correlation" method, although they are superior to the original performance and competitive with the other methods, they do not seem to work as well as other methods. The techniques that better perform are the "min entropy" blending method followed by the "mean" blending method (both of them being the winner methods in 7 out of 8 teachers set and students

[5] https://github.com/ivallesp/distillnet

combinations, as per top-1 accuracy).

Figure 4.3 shows the training curves for all the cases. As it can be seen in the figure, there are models for which the accuracy degrades more than the others after reaching the maximum accuracy (for instance this is the general trend of the "best" teacher).

Choosing a very small learning rate was crucial to achieve the performance reported. Higher learning rates (10^{-5} and 10^{-3}) were tested leading to worse performances (sometimes even worse than the original pre-trained model). However no exhaustive studies are performed around learning rate sensitivity due to the high resource requirements involved. This effect can be explained by the presence of catastrophic forgetting (French, 1999), which leads to overfitting to the transfer set when the learning rate is too large. The results are expressed as the average metric ± the standard error across the five repeated experiments.

Table 4.2: Results in top-1 and top-5 accuracy (%) for all the experiments. Each column in the table represents a different teacher set. The experiment that achieved the best performance for each student is highlighted in bold, a star symbol is added $\star$ in case the difference is significant with respect to the original accuracy with $\alpha = 0.05$ (as per a one-sided one-sample t-test conducted separately for the Top-1 and Top-5 accuracy results). A second star symbol distinguishes the experiments that significantly beat the distillation with the best teacher (as per a one-sided two-samples t-test with $\alpha = 0.05$)

Student (Orig accuracy)	Blending method	Top 1 accuracy		Top 5 accuracy	
		3-best teachers	all teachers	3-best teachers	all teachers
EfficientNetB0	Best	78.14 ± 0.02⋆	**78.14 ± 0.02**⋆	94.17 ± 0.00⋆	**94.17 ± 0.00**⋆
(top1: 75.17)	Mean	78.20 ± 0.02 ⋆ ⋆	78.07 ± 0.01⋆	**94.17 ± 0.01**⋆	94.01 ± 0.00⋆
(top5: 92.34)	Median	78.18 ± 0.01 ⋆ ⋆	77.98 ± 0.01⋆	94.10 ± 0.01⋆	93.93 ± 0.01⋆
	Random	78.21 ± 0.01 ⋆ ⋆	78.05 ± 0.01⋆	94.08 ± 0.01⋆	93.99 ± 0.01⋆
	Min Entropy	**78.27 ± 0.01** ⋆ ⋆	77.94 ± 0.01⋆	94.16 ± 0.01⋆	94.07 ± 0.01⋆
	Max Correl.	78.18 ± 0.02⋆	77.79 ± 0.01⋆	94.13 ± 0.01⋆	93.89 ± 0.01⋆
MobileNetV2	Best	73.85 ± 0.01⋆	73.85 ± 0.01⋆	91.83 ± 0.01⋆	91.83 ± 0.01⋆
(top1: 73.11)	Mean	73.99 ± 0.01 ⋆ ⋆	**74.11 ± 0.02** ⋆ ⋆	91.86 ± 0.01 ⋆ ⋆	**92.02 ± 0.01** ⋆ ⋆
(top5: 91.29)	Median	73.92 ± 0.00 ⋆ ⋆	74.04 ± 0.01 ⋆ ⋆	91.84 ± 0.01⋆	91.91 ± 0.01 ⋆ ⋆
	Random	73.96 ± 0.01 ⋆ ⋆	74.03 ± 0.01 ⋆ ⋆	**91.88 ± 0.00** ⋆ ⋆	91.96 ± 0.01 ⋆ ⋆
	Min Entropy	**74.12 ± 0.01** ⋆ ⋆	73.91 ± 0.02 ⋆ ⋆	91.87 ± 0.01 ⋆ ⋆	91.84 ± 0.01⋆
	Max Correl.	73.94 ± 0.01 ⋆ ⋆	73.84 ± 0.01⋆	91.78 ± 0.01⋆	91.77 ± 0.01⋆
DenseNet121	Best	75.83 ± 0.01⋆	75.83 ± 0.01⋆	92.95 ± 0.01⋆	92.95 ± 0.01⋆
(top1: 75.47)	Mean	75.92 ± 0.01 ⋆ ⋆	**76.44 ± 0.02** ⋆ ⋆	93.04 ± 0.01 ⋆ ⋆	93.30 ± 0.01 ⋆ ⋆
(top5: 92.68)	Median	75.87 ± 0.02⋆	76.42 ± 0.02 ⋆ ⋆	92.98 ± 0.01 ⋆ ⋆	93.29 ± 0.01 ⋆ ⋆
	Random	75.92 ± 0.02 ⋆ ⋆	76.43 ± 0.01 ⋆ ⋆	93.03 ± 0.01 ⋆ ⋆	93.30 ± 0.01 ⋆ ⋆
	Min Entropy	**76.00 ± 0.01** ⋆ ⋆	76.09 ± 0.01 ⋆ ⋆	**93.10 ± 0.01** ⋆ ⋆	**93.27 ± 0.01** ⋆ ⋆
	Max Correl.	75.84 ± 0.02⋆	76.21 ± 0.02 ⋆ ⋆	92.96 ± 0.01⋆	93.20 ± 0.02 ⋆ ⋆
Xception	Best	79.11 ± 0.01⋆	79.11 ± 0.01⋆	94.55 ± 0.01⋆	94.55 ± 0.01⋆
(top1: 78.92)	Mean	**79.30 ± 0.02** ⋆ ⋆	79.16 ± 0.02 ⋆ ⋆	**94.70 ± 0.01** ⋆ ⋆	94.65 ± 0.00 ⋆ ⋆
(top5: 94.47)	Median	79.18 ± 0.01 ⋆ ⋆	**79.19 ± 0.01** ⋆ ⋆	94.65 ± 0.01 ⋆ ⋆	94.63 ± 0.01 ⋆ ⋆
	Random	79.27 ± 0.02 ⋆ ⋆	79.14 ± 0.01 ⋆ ⋆	94.68 ± 0.01 ⋆ ⋆	94.59 ± 0.01 ⋆ ⋆
	Min Entropy	79.31 ± 0.02 ⋆ ⋆	**79.17 ± 0.01** ⋆ ⋆	94.64 ± 0.01 ⋆ ⋆	**94.69 ± 0.01** ⋆ ⋆
	Max Correl.	79.19 ± 0.01 ⋆ ⋆	78.82 ± 0.01	94.58 ± 0.00 ⋆ ⋆	94.44 ± 0.00

Finally, the accuracy increase depends substantially on the model being used as student, where the smaller student models are the ones that show the greatest accuracy increases.

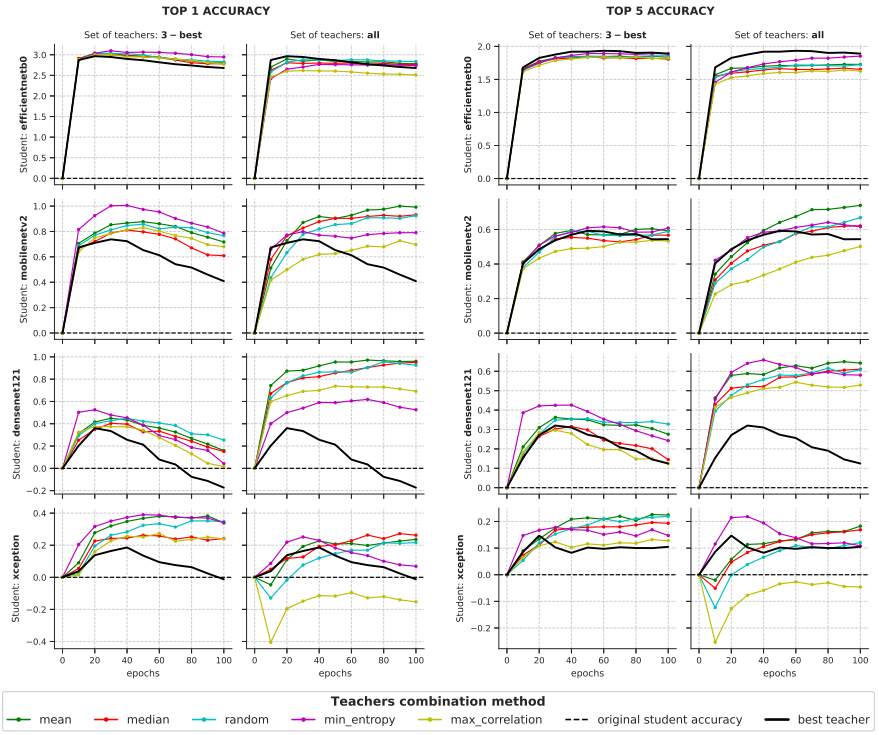

Figure 4.3: Absolute difference of top-1 accuracy (left) and top-5 accuracy (right) with respect to the baseline. The x-axis represents the number of training epochs, and the y-axis shows the absolute difference in performance with respect to the baseline. Each row in the grid represents a different student model. For comparison purposes, the training curve of the student trained by distilling the knowledge of the best teacher is included as a black line.

Challenging the initial assumptions

With the aim of de-biasing the initial assumptions and gaining a better understanding of the dynamics of the proposed methods, this section includes the results of combining the teachers with the "minimum correlation" and with the "maximum entropy" methods (as opposed to the "maximum correlation" and "minimum entropy" methods described in section 4.4). The results are summarized in table 4.3, which shows that in the vast majority of cases the original definition gets significantly higher accuracy than the opposite case. This observation demonstrates that the assumptions taken upon the original methods definitions were accurate. The training curves of these experiments can be seen in figure 4.4, included in the appendix.

4.5.3 Discussion

This chapter shows how it is possible to improve current low-resource pre-trained models by fine-tuning them on larger teacher soft-targets. This is, presumably, the first work that attempts to apply knowledge distillation techniques to pre-trained models. The aforementioned observations lead to the following conclusions: (1) the smaller the student model, the higher the accuracy increase, (2) increasing the number of teachers often leads to more stable training process, reducing the overfitting, and (3) the "min entropy" and the "mean" teacher blending methods often show significantly superior results compared to the rest of methods.

Smaller pre-trained models generally have lower performance than their larger counterparts. This is due, among other possible reasons, to the fact that smaller models have less modeling capacity. Consequently, smaller models have the highest performance gap with the larger pre-trained models, and this has a strong implication in conclusion (1). If the performance gap between the original student model and the teacher signal is large, the student has a larger room for improvement, and hence the observed accuracy increase is larger.

In conclusion (2), distilling the knowledge of multiple teachers often leads to better results than using a single one. The intuition behind that observation is that the variability in the soft-targets increases as more models are combined, and that increases the amount of *dark knowledge*, helping the student model reach a better solution.

In addition, from the training curves of figure 4.3 can be concluded that the training processes of the experiments that use larger sets of teachers seem to be more robust, in the sense that the accuracy does not quickly degrade after reaching the maximum accuracy. The hypothesis behind this phenomenon is that when several teachers are combined to build the distillation target, the complexity of the classification task increases and that has, in essence, a regularization effect.

Table 4.3: Results in top-1 and top-5 accuracy (%) comparing the original "min entropy" and "max correlation" methods with their opposite counterparts. In each comparison, the method that achieved the maximum accuracy is highlighted in bold, and a star symbol $\star$ is added where the difference was statistically significant with $\alpha = 0.05$ (as per a one-sided two-sampled t-test).

Student	Blending method	Top 1 accuracy		Top 5 accuracy	
		best-3 teachers	all teachers	best-3 teachers	all teachers
EfficientNetB0	Min Entropy (original)	**78.27 ± 0.01**$\star$	**77.94 ± 0.01**$\star$	**94.16 ± 0.01**$\star$	**94.07 ± 0.01**$\star$
	Max Entropy (opposite)	78.0 ± 0.01	77.56 ± 0.02	94.0 ± 0.01	93.75 ± 0.00
	Max Correl. (original)	**78.18 ± 0.02**$\star$	**77.79 ± 0.01**$\star$	**94.13 ± 0.01**$\star$	**93.89 ± 0.01**$\star$
	Min Correl. (opposite)	78.07 ± 0.02	77.26 ± 0.01	94.08 ± 0.01	93.66 ± 0.01
MobileNetV2	Min Entropy (original)	**74.12 ± 0.01**$\star$	**73.91 ± 0.02**$\star$	**91.87 ± 0.01**$\star$	**91.84 ± 0.01**$\star$
	Max Entropy (opposite)	73.69 ± 0.01	73.38 ± 0.02	91.65 ± 0.01	91.44 ± 0.01
	Max Correl. (original)	**73.94 ± 0.01**$\star$	**73.84 ± 0.01**$\star$	**91.78 ± 0.01**	**91.77 ± 0.01**$\star$
	Min Correl. (opposite)	73.83 ± 0.01	72.57 ± 0.01	91.77 ± 0.02	91.29 ± 0.01
DenseNet121	Min Entropy (original)	**76.00 ± 0.01**$\star$	**76.09 ± 0.01**$\star$	**93.10 ± 0.01**$\star$	**93.27 ± 0.01**$\star$
	Max Entropy (opposite)	75.55 ± 0.02	75.49 ± 0.02	92.8 ± 0.01	92.86 ± 0.01
	Max Correl. (original)	**75.84 ± 0.02**$\star$	**76.21 ± 0.02**$\star$	**92.96 ± 0.01**	**93.20 ± 0.02**$\star$
	Min Correl. (opposite)	75.66 ± 0.01	75.29 ± 0.02	92.94 ± 0.01	92.83 ± 0.01
Xception	Min Entropy (original)	**79.31 ± 0.02**$\star$	**79.17 ± 0.01**$\star$	**94.64 ± 0.01**$\star$	**94.69 ± 0.01**$\star$
	Max Entropy (opposite)	78.78 ± 0.02	78.40 ± 0.01	94.46 ± 0.01	94.16 ± 0.01
	Max Correl. (original)	**79.19 ± 0.01**$\star$	**78.82 ± 0.01**$\star$	**94.58 ± 0.00**$\star$	**94.44 ± 0.00**$\star$
	Min Correl. (opposite)	78.76 ± 0.02	77.45 ± 0.02	94.44 ± 0.01	93.82 ± 0.0

Another important observation is that the standard error of the results is very small (see table 4.2) because the initial weights in each student are the same across the 5 repetitions: the weights of the pre-trained model. The variability

reported is produced by stochastic processes other than the initialization, such as dropout, the mini-batches arrangement, or the random teacher selection.

In conclusion (3), the "min entropy" and the "mean" methods seem to be the ones producing the best results more often. Also, the "median" and "max correlation" methods do not perform as well as the rest. In the "median" case, this may be happening because, when using this blending method to combine the teacher predictions, the probability distribution is not preserved (i.e. the output probabilities for every class do not sum to 1), having to apply a posterior renormalization step. This process may add some noise to the new distillation target. However, it is not clear why the "max correlation" method is not as good as the rest of methods.

The results reported by this study are relevant given that the methods described are able to increase the performance of pre-trained models, leading to classifiers with significantly higher performance for the same computational budget. For instance, it was shown that by distilling the knowledge of the *3-best* teachers into the *EfficientNetB0* student using the "random" teachers combination method (one of the best results achieved, as shown in table 4.2), the student ends up performing better than the *EfficientNetB7*, as per the top-1 accuracy (see table 4.1), while only requiring less than 7.5% of the parameters and 1.05% of the computation (around 100× fewer FLOPs) than *EfficientNetB7* (M. Tan & Le, 2019).

The training process described on this study is also advantageous from the computational standpoint. The *ImageNet* training dataset is roughly 12.8× larger than the transfer dataset used to fine-tune the students. Additionally, the number of *FLOPs* of the students is, on average, 3.5× smaller than the teachers'. The compound effect of needing less data to train and having less computationally intensive models, leads to a process that needs $3.5 \cdot 12.8 = 44.8\times$ less computation. This means that provided with same-size batch sizes and the same number of training steps, the training process of the teachers would need, on average, about 44.8× more operations than the students fine-tuning.

4.6 Conclusions

The experiments of this chapter show how by using simple knowledge distillation techniques, the accuracy of the smallest pre-trained models increase in just few training epochs. This discovery opens potential new research lines towards more sophisticated teacher blending techniques or distillation methodologies. This can also motivate the research of novel training techniques, alternative or complementary to *backpropagation* given that, as empirically demonstrated, the pre-trained models still did not reach their maximum capacity.

4.7 Appendix

This section includes the training curves for comparing the "min entropy" and "max correlation" methods with their opposite counterparts.

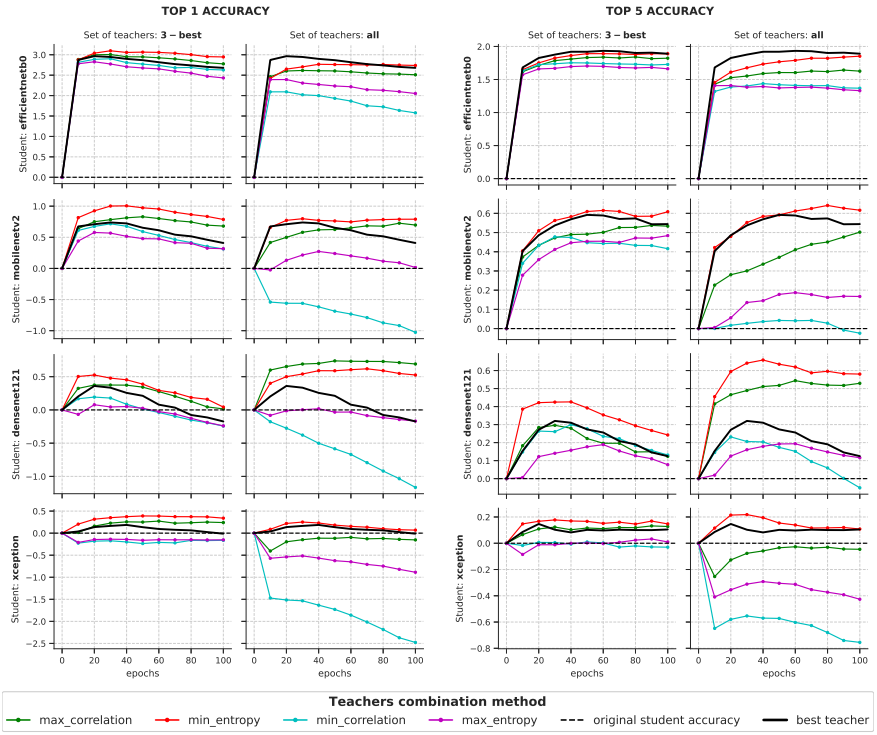

Figure 4.4: Absolute difference of top-1 accuracy (left) and top-5 accuracy (right) with respect to the baseline. The x-axis represents the number of training epochs, and the y-axis shows the absolute difference in performance with respect to the baseline. Each row in the grid represents a different student model.

Chapter 5

End to end sales forecast with deep learning models

5.1 Overview

Accurate and fast demand estimation is one of the key processes in supply chain that enables the precise execution of the corresponding downstream processes (inbound and outbound planning, inventory placement, network planning, etc).

In this chapter, three alternatives are proposed to tackle the problem of forecasting the customer sales at day/store/item level using deep learning techniques and the *Corporación Favorita* dataset, published as part of a *Kaggle* competition. This chapter focuses on building a single end-to-end model, which is more convenient than a multi-model strategy from energy efficiency and model maintenance perspectives. The empirical results show how good performance can be achieved by using a simple *sequence-to-sequence* architecture with minimal data pre-processing effort. Additionally, a training trick for making the model more time independent is described, and it showed evidences towards improving generalization over time. The proposed solution achieves a RMSLE of around 0.54, which is competitive with other more specific solutions to the problem proposed in the *Kaggle* competition.

The development and usage of a standalone forecasting model that generalizes across time, locations and items requires less computational resources than fitting models for all the combinations. One of the main reasons is that a single global hyper-parameters search effort is required. In that search it is shown that for this particular case study simpler models (RNN-based) perform better or equal than large ones (transformer-based). In addition, evidences are found about the ineffectiveness of providing the models with historical information beyond the last 75 days, and this drastically reduces the amount of computation needed at inference time.

All the code used in the experimentation part of this work has been made publicly available in a repository[1].

[1] https://github.com/ivallesp/cFavorita

5.2 Introduction

The retail industry economic activity is moving online. In the last years, e-commerce companies are gaining more and more adepts every day. *E-Marketer* (Cramer-Flood, 2020; Lipsman, 2019) reported consistent year on year growths in number of sales, being more than 13% in the last 5 years in the US. The percentage of total sales in the US owned by e-commerce companies increased from 8,9% to 14,5% in 2020 (Cramer-Flood, 2020; Lipsman, 2019).

The continuously increasing demand requires the online industries to constantly improve their supply chain systems. This process entails multiple challenges such as: optimal inventory placement (S. Graves & Willems, 2008)accurate network expansion (Badri et al., 2017), precise inbound and outbound planning (Kaipia, 2009), etc. One of the most important wires that enables all those improvements is the ability to accurately forecast the customer demand for different products, locations and times (Forslund & Jonsson, 2007).

This study proposes several alternatives to solve the demand forecast problem in an end-to-end manner using deep learning techniques (I. Goodfellow et al., 2016). The generalization power of these algorithms enables solving the problem using a single model for all the different locations and products time series, while other algorithms like the *ARIMA* family of models (Hyndman, 2018) only can tackle one product-location time series at a time.

Two approaches are described in this chapter: (1) a *sequence-to-sequence* (*seq2seq* hereafter) architecture with product and (2) location conditioning and an adapted *transformer* architecture for time series forecasting.

5.2.1 Data

The *Corporación Favorita Grocery Sales* dataset has been used to conduct this study (Corporación Favorita, 2018). *Corporación Favorita*, an Ecuadorian company owner of multiple supermarkets across Latin America, released this dataset around 2017 as a *Kaggle* competition to challenge the community to forecast their sales. It contains daily sales records for 4,400 unique items, in 54 different Ecuadorian stores from January 1st 2013 to August 15th 2017. Additional data provided along with the number of sales are described below.

- *ID variables*: date, store number and item number.
- *Promotions*: a binary variable indicating if a given item, in a given store at a given time was on promotion.
- *Store information*: location (city and state) and segment (type and cluster).
- *Item information*: item family, class, and a binary variable indicating if the item is perishable.
- *Transactions*: number of total sales for each store at each date.
- *Oil price*: price of the oil on each date.
- *Holidays and events*: dates, locations and types of holidays.

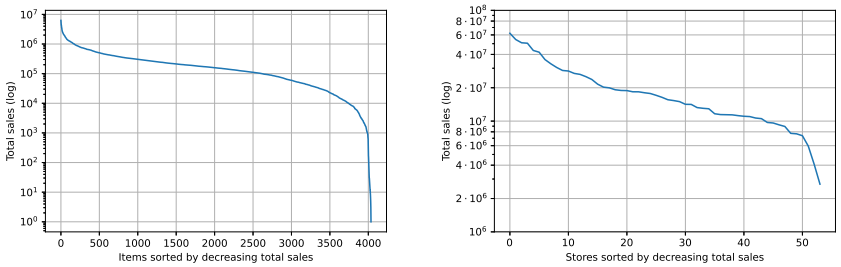

Figure 5.1: Long tail distributions for items (left) and stores (right). Y axes represent total sells across all the history (around 5 years).

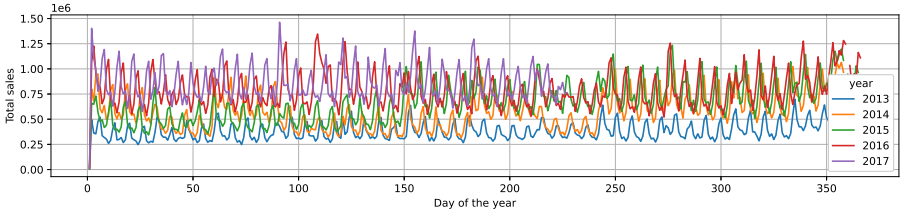

Figure 5.2: Daily total sales for the 5 years included in the data.

As it is usual in the supply chain organizations, the distribution of sales across products is very uneven (see Figure 5.1-left where the top 10% of the products bring the 44% of the sales). The same characteristic is observed in the case of the stores (see Figure 5.1-right where the top 10 stores bring 40% of the sales). The sales for some products are very sparse: during the test period, the probability of having one or more sells for a given product in a given day is 47.6%, which in practice means that more than half of the days in the time series will have value of zero.

As it can be noticed in Figure 5.2, the total sales show clear year on year trends as well as a very remarkable weekly seasonality (see Figure 5.3). There is also a peak of sales at the end of each year.

The dataset does not contain records for items in days when there were zero unit sales. It also lacks information about the available inventory. These two facts together make the forecast effort more complicated, given that when there are zero sales of a given product in a store at a given date, it can be either because there was not available inventory, or because there was inventory but not demand (or both of them at the same time).

Estimating the actual demand of a retailer is not a straightforward task (Deep & Salhi, 2018). In this case, the quantity being forecasted is the number of sales. That would have an important implication in the demand estimation: the number of sales represent the demand as long as there exists available inventory. Hence, the quantity estimated by the machine learning model will be the demand bounded by the inventory $min(demand(i, s, d), inventory(i, s, d))$, where i, s and d are the inventory, the store and the date. There are techniques to correct the demand in these cases (e.g. (Bell, 2000)), however as the objective of this chapter is to build the forecast model, it is out of the scope to deal with that

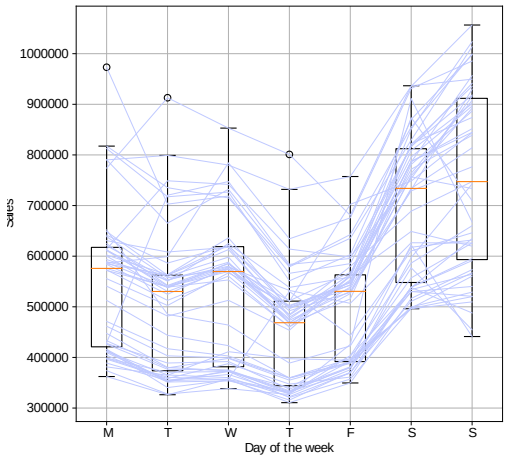

Figure 5.3: Weekly pattern detail, including the 2014 sales.

inconvenience.

5.3 Previous work

Several works have been published discussing the application deep learning techniques for demand and sales forecast in the supply chain environment. The authors of (Kilimci et al., 2019) use several *multilayer perceptrons* (MLP) to build a unified forecast and compare it with more classical techniques. The main disadvantage of this approach is that it requires heavy feature engineering work, as the MLPs are not suitable to deal with time series. In a supply chain problem, this is not practical given the huge quantity of data normally available. In the work by (Talupula, 2018), different deep learning architectures based on *CNNs*, *LSTMs* and MLPs are compared over an outbound demand forecast task, using data from a retailer.

Similarly, (Helmini et al., 2019) compare a deep learning model based on LSTMs with peephole connections with more classical approaches using tree-based models. The difference with the proposed approach is that it offers a set of flexible architectures capable to deal with multi-modal data more efficiently, while the models proposed in these works can only deal with time series. Furthermore, the proposed approach uses sequence-to-sequence modeling, which minimizes the error across all the predicted sequence. The authors of the aforementioned papers use a next-step prediction auto-regressive model, which only optimizes the error of the next time step.

However, only a few reports were found using this dataset for benchmarking. In (Kechyn et al., 2018), the authors used the *WaveNet* (van den Oord, Dieleman, et al., 2016) architecture to build an auto-regressive forecast. They achieved the 2nd position in the Kaggle competition that published the *Corporación Favorita* data. The authors only show some charts summarizing their results, pointing

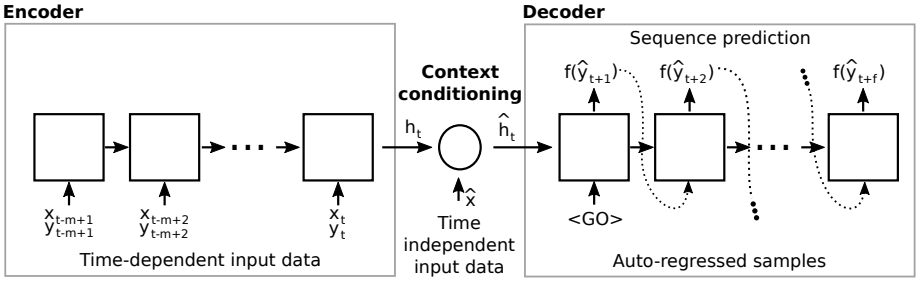

Figure 5.4: *Seq2seq* architecture diagram. The left box shows the encoder, which takes the historical actual sales y_i as input as well as other exogenous time-dependent features (i.e. oil price, holidays and events, transactions and promotions) denoted by x_i, and returns the hidden vector h_t of the last recurrent module as output (aka *context vector*). The middle of the figure shows the context conditioning module, which is the variation proposal over the original sequence-to-sequence proposal. This module receives the *context vector* h_t from the encoder module and concatenates it with static data $\hat{x}$ producing a conditioned context vector $\hat{h}_t = (h_t, \hat{x})$. Finally, on the right, the decoder module receives as initial state the conditioned context vector and provides the first recurrent cell with a *go symbol* (constant input indicating the input of the decoded sequence). The decoder generates the sequence prediction in an auto-regressive way and f is a stack of two fully-connected layers applied to each output of the decoder.

to root mean squared weighted logarithmic errors (RMSWLE) of around 0.578 (see metrics definitions in section 5.5.2. However, they do not give many details about the architecture used and the experimental framework. They also do not specify the period of time used to measure the errors.

The authors of (Calero & Caro, 2018) used multiple classical techniques (historical average, ARIMA/SARIMA, seasonal-naïve and exponential smoothing) to forecast the daily sales. These techniques do not consider multiple sale points and products at the same time. They reach a minimum RMSWLE of 0.555. These solutions achieve competitive results, however, they require training one model per item and store (around 238,000 models), which is not a scalable approach for a production environment and constitute a non energy efficient alternative. The methods proposed in this chapter consist of a single model that is used over all the time periods, items and points of sale.

Other studies using the same dataset have been found: (Curtin et al., 2020; Khamis et al., 2020; Kuleshov et al., 2018; Lim et al., 2019; Malik et al., 2019; Schleich et al., 2019; Shaikhha et al., 2020; J. Wang et al., 2020). Although some of them may be interesting for the supply chain goals, they deviate from the demand/sales forecast goal, hence they will not be summarized.

5.4 Methods

The size of the dataset chosen (around $4 \cdot 10^8$ samples) enables the use of deep learning models. Two different neural architectures have been designed: a *seq2seq* model and a *transformer* model.

5.4.1 Seq2seq

A sequence-to-sequence architecture (abbreviated as *seq2seq*) is a model that is trained to map an input sequence to an output sequence (also known as sequence transduction), without any length restriction in both sides (input and output sequences can be of different length) (Sutskever et al., 2014). This architecture contains two main blocks: one encoder and one decoder. The encoder consists of a recurrent neural network which processes the input sequence, one sample at a time, condensing all the relevant input sequence information into a fixed length *context vector*. This vector is usually the last hidden state of the encoder $\mathbf{h_t}$ (Kamath et al., 2019). The decoder, also consisting of a recurrent neural network, generates the output sequence conditioned to the *context vector*. Both modules are trained together to minimize an error term over the output.

For the purpose of the current objective, the original *seq2seq* architecture has been slightly modified to condition the *context vector* to a set of static (non time-dependent) features (see Figure 5.4 for a graphical representation of the architecture). To achieve that, the original context vector $\mathbf{h_t}$ has been concatenated with the static features (item and store embeddings and related features) and then passed into a feed-forward neural network with two hidden layers. The context conditioning module is an important part of the network because it allows the model to adapt the predictions to each product and store. The output of the feed-forward neural network has been used as the initial hidden state of the decoder. Other possible alternative would be to concatenate the static information along with every element of the input sequence. However the former approach was favoured given its benefit in terms of energy efficiency. The input of the first recurrent cell of the decoder is a special symbol that indicates the model that it is the first step of the output sequence. In this case, the special symbol is a vector containing all zeros.

The decoder module is an auto-regressive model trained to predict the immediate next step, i.e. the predicted value for time step t is used as input for the prediction of time step $t + 1$.

5.4.2 Transformer

Transformer architectures were firstly published in 2017 (Vaswani et al., 2017). This architecture removes the need to use recurrent neural networks by implementing attention and self-attention mechanisms (Bahdanau et al., 2015). Like *seq2seq* architectures, the transformers are able to map an input sequence to an output sequence, with potentially different lengths. Similar to the *seq2seq* models, they also consist of two blocks: the encoder and the decoder; which function similarly as in the *seq2seq* architecture.

The attention mechanism can be described as shown in equation 2.16. Where Q, K, V stand for *query*, *key* and *value*, respectively. These three pieces represent an analogy, introduced by (Vaswani et al., 2017), of the information retrieval systems where a *query* is used in order to look for the matching *key* (or the most similar one) and retrieving its *value*. The attention mechanism working principle is similar to those systems (refer to section 2.3.4 for a deeper review of the attention mechanism). There are many possible differentiable similarity functions (f) that can be used (Kamath et al., 2019). The authors of (Vaswani et al., 2017) propose the Scaled dot product as similarity function, given that

it is scaled so that different length sequences can be easily compared together. The Scaled dot product is defined in equation 2.18, where d_K represents the length of the key vector K. This version was adopted as it showed to work well in the initial transformer publication. The transformer architecture is described in detail in section 2.3.4.

A slight modification has been performed over the original transformer, removing the *softmax* operation of the output and only using the categorical embeddings for the categorical inputs. This is necessary in this case because the task is a regression and not a classification.

Following the information retrieval analogy and as illustrated in the figure 5.5, there are two types of attention being used in this architecture.

- *encoder-encoder attention*: this is a form of self-attention that is used in the encoder module. In it, the *query*, the *key* and the *value* come from the same time series.
- *decoder-decoder masked attention*: also a form of self-attention with the particularity that the operation is forced to be causal, i.e. it only uses time steps from the past, the future ones are masked out. The *query*, *key* and *value* come from the same time series.
- *encoder-decoder attention*: this attention mechanism compares the decoder information with the encoder one, hence it is not self-attention. The *query* comes from the decoder while the *key* and *value* are taken from the encoder output.

To train the *transformer* architecture the *teacher forcing* technique is used (Goyal et al., 2016; Williams & Zipser, 1989). It consists of feeding the decoder with the target sequence, right-shifted by one sample so that the training can be done in one single calculation per batch. This technique is commonly used in auto-regressive models to improve the speed of training, and showed good results in the literature (Vaswani et al., 2017).

At inference time, *teacher forcing* is no longer available (because the future time steps of the time series are unknown) so auto-regression is used to compute the next steps recursively (i.e. the predicted sample is fed back to the input in order to predict the next sample). For a more detailed discussion of the transformer architecture, refer to section 2.3.4.

5.4.3 Random max time step trick

At training time and with the aim of improving generalization over different periods of time, each *mini-batch* has been constructed so that the maximum time step (the most recent one) is drawn randomly from the time line. This trick allows the algorithm to learn a model that generalizes over different periods of time, preventing it to overfit to a single time span.

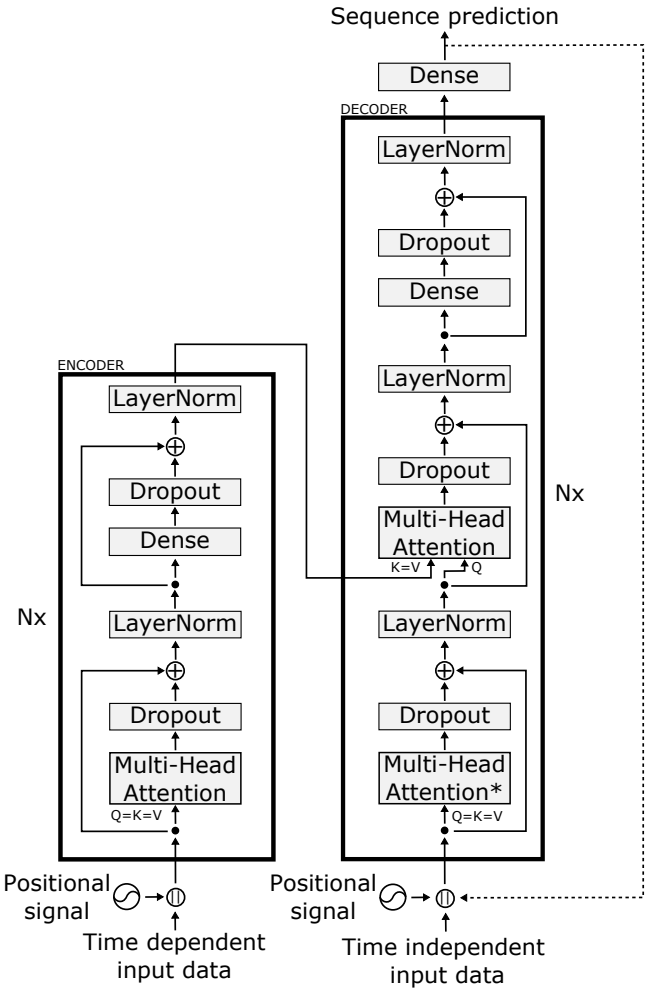

Figure 5.5: Transformer architecture used to perform time series forecasting. In the diagram, the || symbol stands for the concatenation operation, and similarly as in (Vaswani et al., 2017), N_X represents the number of repetitions of the encoder and decoder blocks.

5.5 Experiments and results

5.5.1 Setup

The dataset provided has intentionally been minimally pre-processed as one of the goals of the current study is to provide a simple and flexible solution to the sales forecast problem. The most important transformation consisted of filling the zero sales records, as the dataset was provided without them. The numerical input variables have been standardized ($\mathbb{N}(0, 1)$) while the categorical variables have been turned into *one hot encoding* representations. The target variable has been scaled using a logarithmic transformation, as suggested by the authors of the dataset in the *Kaggle* competition (Corporación Favorita, 2018). The ID variables corresponding to the store and the item have been used as an input to an embedding lookup layer to give the model the opportunity to learn store or

item related information.

The model has been trained using daily data from January 1st 2013 to May 27th 2017, to produce daily forecasts of the next 16 days[2] from any present day. Data from June 13th 2017 to June 28th 2017 have been used for validation purposes and the next 3 16-days time spans (June 29th to July 14th, July 15th to July 30th and July 31st to August 15th, referred subsequently as period 1, 2 and 3 respectively) have been used to test the performance of the algorithms.

The *random max time step* has been constrained not to lay before October 29th 2013, to assure that the model has at least 300 days of history to learn from.

In the *seq2seq* model all the history (from January 1st 2013) has been used as input. In the *transformer* model, given the quadratic computational complexity dependence on the length of the input sequence, the history had to be shortened to 200 days. In the spirit of fair comparison, an alternative *seq2seq* version (referred subsequently as *seq2seq trimmed*) has been trained using the 200 most recent time steps in every *mini-batch*. To facilitate the interpretation of the results, two baselines have been included: *random* and *average*. The first one consists of measuring the accuracy of a naïve prediction built by randomly permuting the target variable. The second one consists of predicting the average of the target variable for all the instances.

5.5.2 Results

The results have been measured using the *root mean squared logarithmic error* (*RMSLE*, defined in equation 5.1), *root mean squared weighted logarithmic error* (*RMSWLE*, defined in equation 5.2, where the perishable items are given a weight of 1.25, and 1.0 to the rest), and *mean absolute logarithmic error* (*MALE*, defined in equation 5.3).

$$RMSLE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\log(\hat{y}_i + 1) - \log(y_i + 1)\right)^2} \tag{5.1}$$

$$RMSWLE = \sqrt{\frac{\sum_{i=1}^{n} w_i \left(\log(\hat{y}_i + 1) - \log(y_i + 1)\right)^2}{\sum_{i=1}^{n} w_i}} \tag{5.2}$$

$$MALE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left|\log(\hat{y}_i + 1) - \log(y_i + 1)\right|} \tag{5.3}$$

In the previous equations, $\hat{y}_i$ represents the predicted sales, y_i represents the actual sales and N is the total number of samples. The logarithmic component of the error metrics was introduced because different products at different shops have arbitrarily different demand levels. The usage of the logarithm scales the unit sales distribution and makes the whole problem easier to measure. A natural

[2]this is not an arbitrary decision, the 16 days were chosen because that was the requirement in the *Kaggle* competition. That choice would make sense for bi-monthly forecast publications, as it would be applicable for months with 28, 29, 30 and 31 days

logarithm has been used along all this chapter. The *RMSWLE* error metric has been introduced for easier comparison and benchmarking with future studies.

Figure 5.6 shows how the errors evolve in every epoch. Figure 5.7 decomposes the error at store and item level, in order to show in detail how the errors vary along these dimensions. Finally, Figures 5.8 and 5.9 show examples of actual and forecasted time series in log and linear scales, respectively.

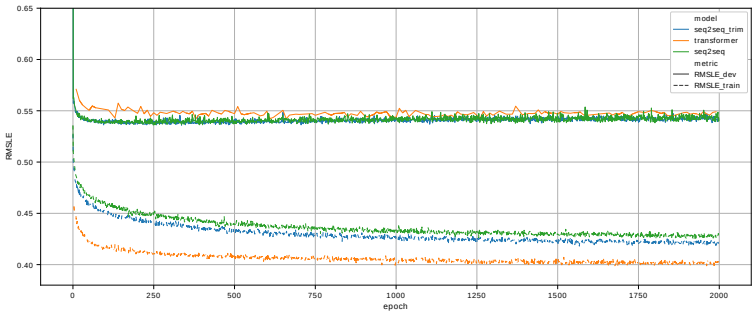

Figure 5.6: Evolution of the train and validation (dev) error during the process of training, for all the models.

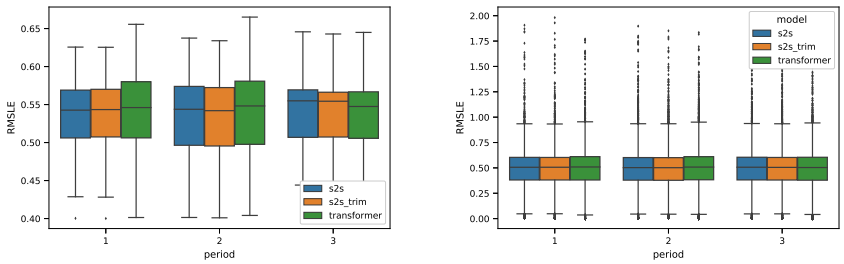

Figure 5.7: Distribution of the error across stores (left) and accross items (right) for every model and for the three different test periods used.

As shown in figure 5.7, the performance is very similar across models. A numerical comparison of the errors obtained for every model is presented in Table 5.1. Additionally, a deeper daily analysis is provided in Figure 5.10. From these figures, it can be noticed that the error is not distributed uniformly across products, stores and time.

Table 5.1 shows that the three models perform similarly. However, the daily figures show that the transformer error has more variability around the second and fourth day of forecast. This may be due to the fact that the model has been trained using *teacher forcing* (Goyal et al., 2016; Williams & Zipser, 1989), and at inference time, an auto-regressive strategy has been used to compute the forecasted sales. This may cause distribution shifts that impact the quality of the forecast. Besides, the *seq2seq* models were much faster at training and inference time. This is due to the quadratic complexity dependence on the sequence length in the *transformer* architecture; in *seq2seq* it is linear. The

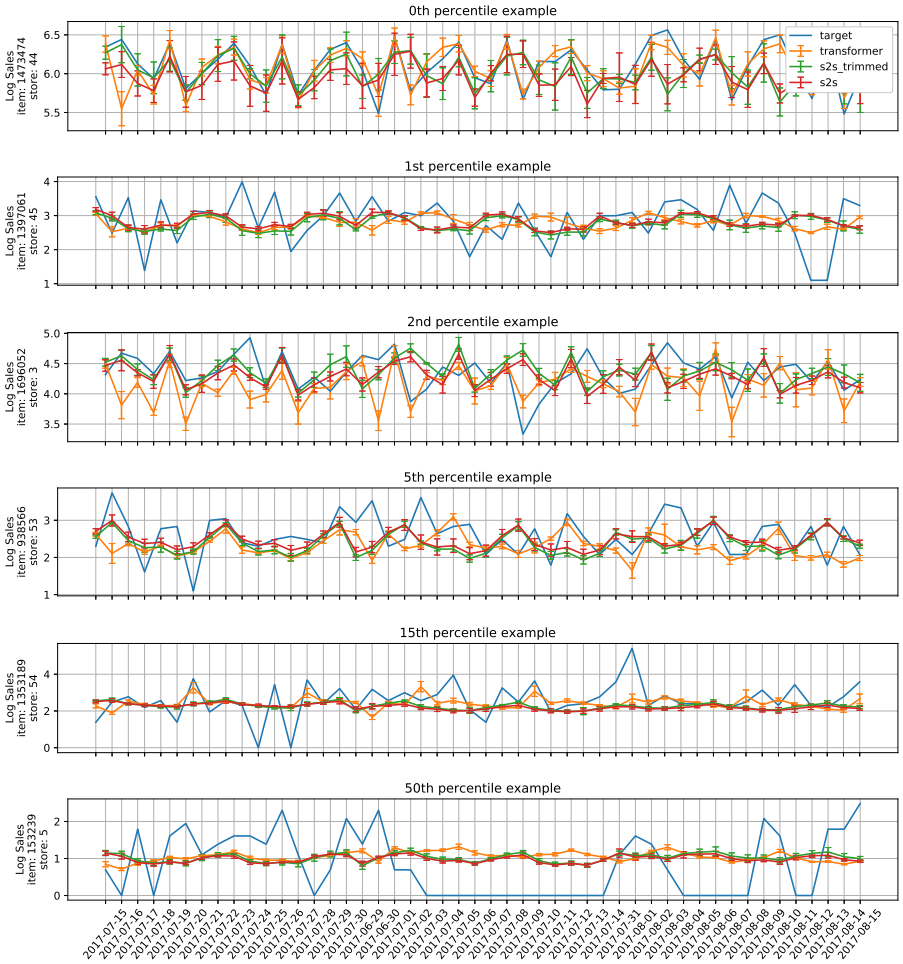

Figure 5.8: Actual and forecasted sales in log scale for six examples of store-item combinations representing the 0th (best prediction), 1st, 2nd, 5th 15th and 50th percentiles of RMSLE (relative to the target variable average) from top to bottom. The three test periods have been concatenated along the X axis. The error bars show the standard deviation across the 5 runs.

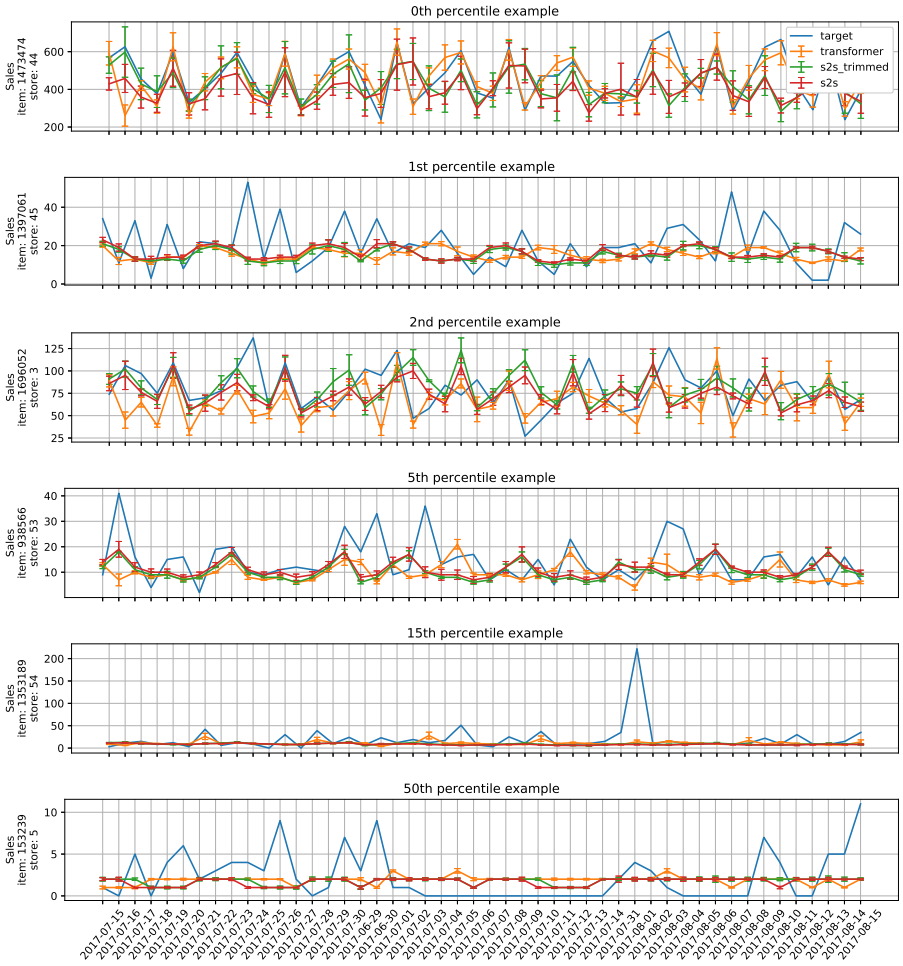

Figure 5.9: Actual and forecasted sales in linear scale for six examples of store-item combinations representing the 0th (best prediction), 1st, 2nd, 5th 15th and 50th percentiles of RMSLE (relative to the target variable average) from top to bottom. The three test periods have been concatenated along the X axis. The error bars show the standard deviation across the 5 runs. Some of the examples are better viewed in log-scale (refer to figure 5.8).

simplest model (*seq2seq trimmed*) was the fastest of the three alternatives, with no noticeable decrease in performance, either in the general picture or in the daily figures.

Table 5.1: Results of the models trained for three different time spans. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. The rows corresponding with the models that achieved the lowest RMSLE have been highlighted in bold. At the bottom of the table two benchmark metrics extracted from the previous works have been included. However, the authors of those works do not clearly specify the period of time used to measure the results, hence they should be used only as a reference.

Period	Model	RMSLE	RMSWLE	MALE
1	**Seq2seq**	**0.5380 ± 0.0016**	**0.5376 ± 0.0016**	**0.3450 ± 0.0024**
	Seq2seq trimmed	0.5381 ± 0.0008	0.5377 ± 0.0008	0.3442 ± 0.0008
	Transformer	0.5439 ± 0.0024	0.5436 ± 0.0023	0.3386 ± 0.001
	Baseline: random	1.474 ± 0.0003	1.4795 ± 0.0003	1.0691 ± 0.0002
	Baseline: average	1.0422	1.05	0.8744
2	Seq2seq	0.5431 ± 0.0014	0.5421 ± 0.0013	0.3475 ± 0.0012
	Seq2seq trimmed	**0.5413 ± 0.0019**	**0.5403 ± 0.0018**	**0.3444 ± 0.0012**
	Transformer	0.5495 ± 0.0021	0.5486 ± 0.0021	0.3415 ± 0.0012
	Baseline: random	1.4649 ± 0.0002	1.4702 ± 0.0002	1.0577 ± 0.0003
	Baseline: average	1.0358	1.0433	0.8655
3	Seq2seq	0.544 ± 0.0021	0.5431 ± 0.0021	0.3502 ± 0.0028
	Seq2seq trimmed	0.5423 ± 0.0015	0.5414 ± 0.0016	0.3481 ± 0.0017
	Transformer	**0.5414 ± 0.0015**	**0.5407 ± 0.0014**	**0.3366 ± 0.0012**
	Baseline: random	1.4555 ± 0.0002	1.4606 ± 0.0002	1.0517 ± 0.0002
	Baseline: average	1.029	1.0363	0.8616
Benchmark	(Kechyn et al., 2018)	-	0.5780	-
Benchmark	(Calero & Caro, 2018)	-	0.5550	-

5.5.3 Ablation study

This subsection contains an analysis of the effect of two core pieces of the proposed architecture: the *random max time step trick* and the length of the input sequences in the *seq2seq trimmed* model.

Random max time step trick

The *seq2seq trimmed* model is retrained without the *random max time step trick*. Table 5.2 summarizes the errors obtained at the best iteration of each model, averaged across five repeated runs. From the results one can conclude that when the *random max time step trick* is used, the model achieves significantly superior performance. This is due to the fact that randomizing the max time step helps the model to capture behaviors of the target signal at different times. Figure 5.11, shows the training curves when the trick is used and when it is not used, suggesting that the trick may also act as a regularization technique, as the model is much less prone to overfitting when the trick is used.

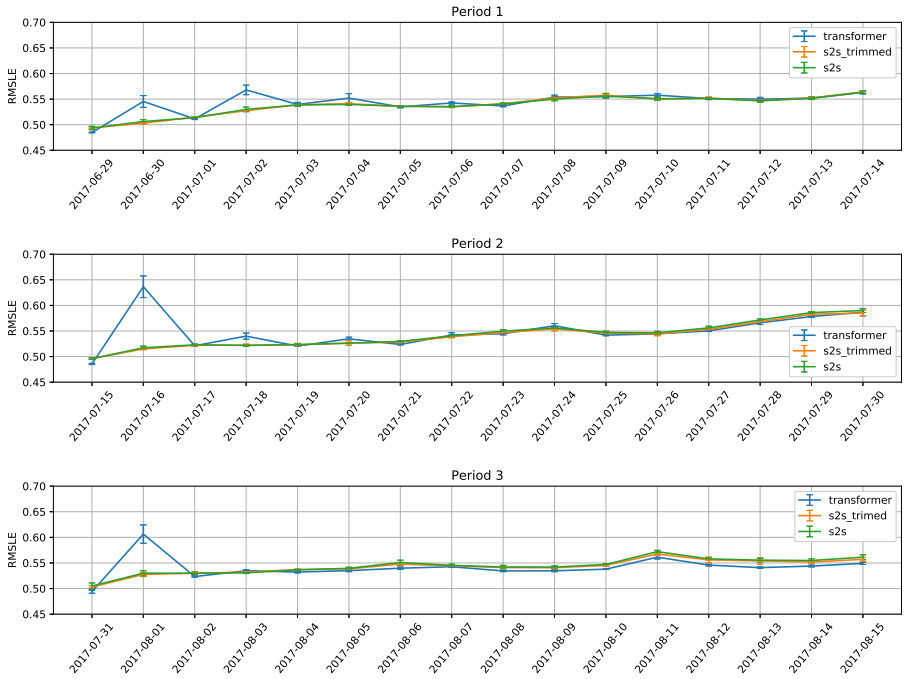

Figure 5.10: Daily *RMSLE* for the three test periods (in chronological order, from top to bottom). The error bars shown in the figures represent the standard deviation of the three runs. Despite the error spikes in the 2nd day of the forecast, an ANOVA test shows non-significant differences between the average performance of the three models (with the following p-values: 0.0947, 0.1823 and 0.6181 for periods 1, 2 and 3, respectively).

Table 5.2: Results of the models trained with and without the *random max time step trick*. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. The rows corresponding with the models that achieved the lowest RMSLE have been highlighted in bold.

Period	Trick/No trick	RMSLE	RMSWLE	MALE
1	Trick	**0.5381 ± 0.0008**	**0.5377 ± 0.0008**	**0.3442 ± 0.0008**
	No trick	0.6077 ± 0.0055	0.6073 ± 0.0054	0.4037 ± 0.0171
2	Trick	**0.5413 ± 0.0019**	**0.5403 ± 0.0018**	**0.3444 ± 0.0012**
	No trick	0.5895 ± 0.0042	0.5886 ± 0.0042	0.3892 ± 0.0216
3	Trick	**0.5423 ± 0.0015**	**0.5414 ± 0.0016**	**0.3481 ± 0.0017**
	No trick	0.5929 ± 0.0127	0.5922 ± 0.0125	0.3938 ± 0.0318

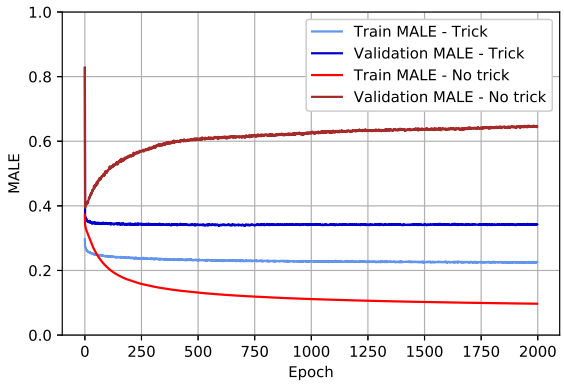

Figure 5.11: Training and validation *MALE* curves of the *seq2seq trimmed* when using the *random max time step trick* and when disabling it.

Input sequences length

As showed in the Table 5.1, the *seq2seq* model can be further simplified by trimming the length of the input sequences. This subsection studies the minimum length of the input sequence without significant performance degradation. For that, the *seq2seq* model was retrained with different input sequence lengths to determine where is the optimum. The Table 5.3 shows the results of the model with the full sequences, and with sequences trimmed to 200, 75, 10, 1 and 0 time steps (where 0 time steps means not using any input sequence information at all, only static features).

The results show that it is possible to reduce the sequence lengths to up to 75 time steps without losing performance (and even slightly improving the generalization). Further reductions to 10 and 1 time steps start showing performance degradation. When not using input sequences (length=0) the performance degrades notably, as compared to using only one time step. The most plausible hypothesis is that this happens because the model needs some reference level of number of sales per product to produce accurate forecasts.

Table 5.3: Results of the *seq2seq* models trained with different input sequence lengths. All the models have been trained five times to reduce the effect of different random initialization. The errors are represented as mean ± standard deviation, across the five runs. The rows corresponding with the models that achieved the lowest RMSLE have been highlighted. An asterisk is added to the experiments where the metric is significantly different than the full sequence error (as per a two-tail T-test with $\alpha = 0.05$)

Period	Sequence length	RMSLE	RMSWLE	MALE
1	Full	0.5380 ± 0.0016	0.5376 ± 0.0016	0.3450 ± 0.0024
	200	0.5381 ± 0.0008	0.5377 ± 0.0008	0.3442 ± 0.0008
	75	$\mathbf{0.5372 \pm 0.0016}$	$\mathbf{0.5369 \pm 0.0016}$	$\mathbf{0.3458 \pm 0.0034}$
	10	$0.5452 \pm 0.0008*$	$0.5447 \pm 0.0009*$	0.3478 ± 0.0017
	1	$0.5812 \pm 0.0042*$	$0.5807 \pm 0.0042*$	$0.3795 \pm 0.0029*$
	0	$0.8411 \pm 0.0016*$	$0.8453 \pm 0.0015*$	$0.5408 \pm 0.0028*$
2	Full	0.5431 ± 0.0014	0.5421 ± 0.0013	0.3475 ± 0.0012
	200	0.5413 ± 0.0019	0.5403 ± 0.0018	$0.3444 \pm 0.0012*$
	75	$\mathbf{0.5400 \pm 0.0010}*$	$\mathbf{0.5392 \pm 0.001}*$	$\mathbf{0.3458 \pm 0.0009}$
	10	$0.5510 \pm 0.0049*$	$0.5501 \pm 0.0048*$	0.3509 ± 0.0038
	1	$0.6162 \pm 0.0035*$	$0.6156 \pm 0.0037*$	$0.4011 \pm 0.0032*$
	0	$0.8426 \pm 0.0016*$	$0.8461 \pm 0.0015*$	$0.5388 \pm 0.0027*$
3	Full	0.5440 ± 0.0021	0.5431 ± 0.0021	0.3502 ± 0.0028
	200	0.5423 ± 0.0015	0.5414 ± 0.0016	0.3481 ± 0.001
	75	$\mathbf{0.5418 \pm 0.0026}$	$\mathbf{0.5411 \pm 0.0025}$	$\mathbf{0.3499 \pm 0.0047}$
	10	$0.5560 \pm 0.0051*$	$0.5548 \pm 0.0051*$	0.3562 ± 0.0049
	1	$0.6360 \pm 0.0058*$	$0.6352 \pm 0.0062*$	$0.4163 \pm 0.0041*$
	0	$0.8387 \pm 0.0017*$	$0.8419 \pm 0.0016*$	$0.5373 \pm 0.0027*$

5.6 Conclusions

Along this chapter, a *seq2seq* and a transformer architecture were proposed as a set of models capable of solving the problem of sales forecasting for the *Corporación Favorita* problem. Additionally, a trick (named *random max time step trick*) is introduced, allowing to train the model to adapt to different time steps, not requiring to retrain the model every time a prediction is needed.

It has been empirically proven, through the experiments described in this chapter, that it is possible to build a forecast for different products, at different points of sale and at different points in time using a single model. The *seq2seq trimmed* model achieved the best performance at the lowest theoretical computational cost. For that reason, its usage is recommended for this type of use cases.

Exploring smaller and more efficient architectures is left as future work (for instance, smaller RNNs with shorter historical sequences), to further reduce the computational cost of the proposed solution. In a real case, where more information about the input variables is available, feature engineering may also be useful in order to help finding better representations. Finally, more sophisticated normalization methods for the target variable might be useful to deal with different magnitudes and sparsity.

Chapter 6

Keyword Spotting with efficient convolutions

6.1 Overview

Speech commands recognition is a very relevant task for human-computer interaction at this time, due to the increase of the global interest in personal assistants. The field of conversational agents is growing fast and there is an increasing need for algorithms that perform well in order to enhance natural interaction.

This chapter shows that state of the art results can be achieved by adapting and tweaking the *Xception* algorithm (Chollet, 2017), which achieved outstanding results in several computer vision tasks. This architecture is designed to exploit the computational benefits and efficiency of the *depthwise-separable convolutions*, leading to a solution with significantly lower computational requirements than a regular convolutional neural network. The solution obtained about 96% accuracy when classifying audio clips belonging to 35 different categories, beating human performance at the most complex benchmarks proposed. The source code that has been used for this chapter has been uploaded to *GitHub* and can be publicly found here: https://github.com/ivallesp/Xception1d

6.2 Introduction

The world of voice-activated virtual assistants is booming mainly due to the fact that several giant technological companies, (such as *Amazon*, *Google*, *Microsoft*, *Apple* and *Baidu*) have already developed their own version. There is a huge research community surrounding this field, potentially promoted by the recent growth of the artificial intelligence (AI) paradigm.

There has been an outstanding evolution in this field, leveraging AI and machine learning (ML) advances for making virtual assistants behave as close as humans as possible. There are multiple open research lines, such as increasing the accuracy and the relevance of the responses (Serban et al., 2017), reducing the answer delay (S. Han et al., 2017) or increasing their variability of the responses (J. Li et al., 2017). In particular, deep learning (DL) models have

revolutionized the field of automatic speech recognition (Nassif et al., 2019), as language features are highly hierarchical.

This chapter focuses on studying the usage of DL models to increase the accuracy of a voice command recognition task using a limited vocabulary.

An important factor to take into account is the small size and low power specifications of the usual virtual assistants, which limit the complexity of the models to implement. Furthermore, a low latency is needed to avoid harming the users' experience. DL models usually contain millions of parameters. For that reason, the right choice of number of layers and their architecture is critical. Using the cloud for processing audio commands can partially mitigate this. However, continually transmitting to and from the cloud increases latency and may not be an energy-efficient solution. Therefore, at least the models that process the most common words in a vocabulary, such as keyword spotting (KWS), should be implemented on-device.

6.3 Previous work

The current commercial virtual assistants are still not as accurate as humans at identifying human voice commands (Michaely et al., 2017). Although substantial efforts have been made and nearly human performance has been reported in several studies (Coimbra de Andrade et al., 2018; McMahan & Rao, 2017; Warden, 2018; Y. Zhang et al., 2017), there is still room for improvement.

Bidirectional recurrent models with attention have been used (Coimbra de Andrade et al., 2018) as these kind of structures are able to accurately model past and future dependencies in the time domain, while attention focuses on important parts of the audio clip. Howewer, the authors state that there are word pairs that are difficult to identify without extra context.

Gated convolutional long-short term memory (LSTM) structures have also proven useful by other authors (D. Wang et al., 2018) to improve the state of the art results on the same data. According to this work, gated convolutions help further learn the local features of speech, improving the model prediction accuracy.

Due to their architecture, convolutional neural networks (CNNs) provide a good approach to optimize computational resources for KWS. In (Sainath & Parada, 2015), CNNs outperform other deep neural networks (DNNs) architectures, such as recurrent reural networks (RNN), for the constrained KWS task. Other works focus on the hardware implementation of neural networks for KWS (Y. Zhang et al., 2017), comparing not only their accuracy, but also their memory usage and computation efficiency. According to this work, depthwise-separable (DS) CNNs achieve both the best accuracy and scalability among the tested architectures. Transfer learning has also been applied to CNN architectures (McMahan & Rao, 2017), showing substantial improvements in accuracy.

6.4 Materials and methods

6.4.1 Data set

One of the major contributions to the collection of open datasets in the field of speech commands recognition has been made by *Google* with the release of the *Google Tensorflow speech commands dataset* (Warden, 2017; Warden, 2018). It consists of a collection of thousands of utterances with a length of one second containing short words, recorded by thousands of people. The recordings were collected remotely, in uncontrolled environments (i.e. without any specific quality requirements). The subjects were asked to say a chain of words during a minute. Then, the recordings were split in clips of one second by extracting the loudest sections of the signal (Warden, 2017; Warden, 2018).

Two different versions of the *Google Tensorflow speech commands dataset* have been used for quantifying the performance of the algorithm across the different tasks, described subsequently. The versions 0.01 and 0.02 of the dataset contain 64,721 and 105,829 audio clips of one second (each one containing the recording of one voice command), respectively, with a sample rate of 16 kHz and 16-bit resolution. Each of them is stored in *wav* format. The first data version has up to 30 different commands while the second one has 35 of them. The frequency distribution of the labels is described in Figure 6.1. For simplicity, versions 0.01 and 0.02 will be referred as as V1 and V2, respectively, from now on.

Four different tasks have been defined in order to benchmark the proposed algorithm. They are thoroughly described below in decreasing complexity order.

- *35-words-recognition*: consists of classifying all the different existing clips (commands and plain words) in each of the different categories (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”, “zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”, “dog”, “cat”, “wow”, “house”, “bird”, “happy”, “sheila”, “marvin”, “bed”, “tree”, “visual”, “follow”, “learn”, “forward”, “backward”). In the version V1 of the dataset there are 5 missing commands and hence, the task consists of a 30-class classification task even though it is called *35-words-recognition*.

- *20-commands-recognition*: entails the categorization of all the clips representing the most commonly used commands in robotics (Warden, 2018) and numbers (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”, “zero”, “one”, “two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”), while the remaining words are grouped together in a synthetic category named “unknown”.

- *10-commands-recognition*: requires categorizing all the clips representing the typical commands in robotics (“left”, “right”, “yes”, “no”, “down”, “up”, “go”, “stop”, “on”, “off”), while the remaining clips are grouped together in a synthetic category named “unknown”.

- *left-right-recognition*: consists of categorizing the clips belonging to the “left” and “right” categories, while the rest of clips are grouped under a new “unknown” category.

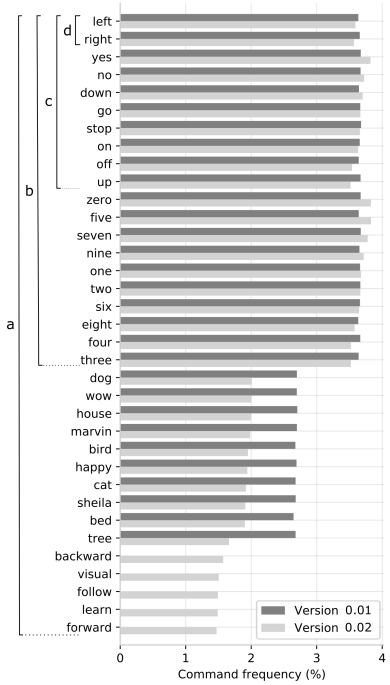

Figure 6.1: Command frequency distribution for both versions of the dataset. The V2 is a refined and extended version of V1. In the left, the four different tasks that have been benchmarked in this work: (a) referred as *35-words-recognition* and comprising in both cases all the words for classification, (b) referred as *20-commands-recognition* (c) referred as *10-commands-recognition* (d) referred as *left-right* recognition.

While it is true that, as more unrecognized words are grouped under the "unknown" category the imbalance grows, in this kind of systems (speech recognition agents), the precision should be optimized at the expense of a worse recall (in general, the cost of a false positive is higher than that of a false negative). Thus, for this purpose, having a positive imbalance towards the "unknown" class does not represent an inconvenience. A summary of the percentage of words grouped under the "unknown" category for each task is shown in Table 6.1.

Table 6.1: Percentage of words represented by the "unknown" category in each one of the proposed speech recognition tasks.

Data set version	*35-words*	*20-commands*	*10-commands*	*left-right*
V1	0.00%	26.84%	63.41%	92.71%
V2	0.00%	26.81%	63.58%	92.84%

6.4.2 Data augmentation

Five different augmentations[1] consisting of the application of a set of distortions have been performed to each and every audio clip. These distortions consist of a set of transformations being applied together over all the clips randomizing their parameters and intensities in each case. The different distortions used in this step are described below.

- *Resampling*: the audio clip is resampled extending or contracting its length and hence changing its pitch (Proakis, 2007). If the resampling factor is lower than one, the audio clip is contracted and then the audio duration is zero-padded to keep the original duration of the original clips. On the contrary, if the resampling factor is greater than one, the audio clip is extended and a *center-crop* operation is performed in order to keep the original length of one second.
- *Saturation*: the amplitude of the clip is increased by a given factor, potentially saturating the audio clip. The higher the factor, the larger the saturation of the clip (Proakis, 2007).
- *Time offset*: the audio clip is displaced in time by appending a set of zeros to the beginning of the signal and cropping the end (right offset), or by cropping a set of samples from the beginning, and adding the same number of zeros at the end (left offset) (Proakis, 2007).
- *White noise addition*: white noise (with gaussian distribution) is added to the clips with a given amplitude (Proakis, 2007).
- *Pitch shift*: the pitch of the clips is increased or decreased a given amount, producing higher or lower-pitched sounds (Buś & Jedrzejewski, 2016; Proakis, 2007).

All the distortions are applied together with random intensities only to the training data, producing 5 new transformations of the original recordings with high variability of results. These new versions are appended to the original dataset.

6.4.3 Depthwise-separable convolutions

The use of a CNN with *depthwise-separable convolution* layers and *residual connections* is proposed, based on the *Xception architecture* described by *François Chollet* in 2017 (Chollet, 2017). This model is a CNN-based architecture which achieved state of the art results in multiple computer vision tasks (C. Liu et al., 2019; Nazaré et al., 2018; Song et al., 2018).

The *regular convolution* operation consists of the application of a filter over a signal along the spatial/temporal dimension(s) and along the channels in a single operation.

A *depthwise-separable convolution* performs an operation which, given an input tensor, produces an output tensor of the same shape that the regular convolution would produce, but in a more efficient way; i.e. it reduces the number of sums and multiplications needed to produce the output (Chollet, 2017). This

[1] meaning distorted versions of each of the clips in the original data

is achieved by following the two steps described below (the whole operation is shown in the Figure 6.2).

1. *Depthwise convolution* (Guo et al., 2019): consists of convolving a separate filter per channel. It produces an output tensor with potentially a different spatial/temporal dimensions size than the input (depending on the stride, the size and the padding of the convolution operation to be applied). However, the number of channels of the output tensor is constrained to be the same as the input.

2. *Pointwise convolution* (Gao et al., 2018): consists of applying a set of size-1 convolutions (as many as number of output channels desired). This operation can modify the channels dimension, leaving the spatial/temporal dimensions intact.

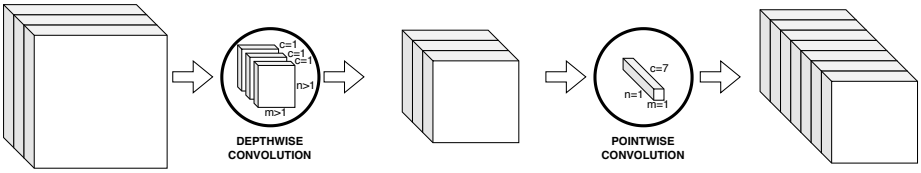

Figure 6.2: *Depthwise-separable convolution* diagram showing how the whole computation is done in two steps: the *depthwise convolution* followed by the *pointwise convolution*.

The *depthwise convolution* does not combine different channels for producing the output, and hence always produces an output tensor which has the same number of channels as the input tensor. This difference makes the operation much more efficient than the *regular convolution*, at the cost of loosing the ability to generate new features by combining different channels. That is the reason why the *pointwise convolution* is applied after the *depthwise convolution* constituting the *depthwise-separable convolution*. The *separable* term is used in the name of the operation because it is effectively separating between the channel-wise and the spatial/temporal-wise computations.

The number of sums and multiplications required by this operation is $\frac{1}{R} \cdot \frac{1}{O}$ times the number of operations required by a regular convolution (details below) (Howard et al., 2017), where O is the size of the *depthwise convolution* filters and R is the number of output channels after the *pointwise convolution* is applied. This represents a meaningful performance improvement for big networks.

More formally, the *regular convolution* and *depthwise convolution* operations (Howard et al., 2017) are defined[2] in equations 6.1 and 6.2 for illustrative purposes, where

- $\mathbf{X}$ is the feature map over which the convolution operation is intended to be applied. It has a shape of $D \times M$, where D represents the spatial/temporal dimension and M the number of input channels.

- $\mathbf{W}$ is the convolution kernel and has a shape of $O \times M$ where O is the size of the kernel.

[2]in both cases assuming a stride of one, *SAME* padding and odd size convolution kernels

- $\mathbf{S}$ is the output tensor produced by applying the *regular convolution* operation with the filter W over the input X resulting in a vector of length D.
- $\hat{\mathbf{S}}$ is the output tensor produced by applying the *depthwise convolution* with the filter W over the input X, resulting in a matrix with shape $D \times M$. The number of output channels is restricted to be equal to the number of input channels M.

$$\mathbf{S}_x = \sum_{o,m}^{O,M} \mathbf{W}_{o,m} \cdot \mathbf{X}_{x+o-\frac{O-1}{2},m} \tag{6.1}$$

$$\hat{\mathbf{S}}_{x,m} = \sum_{o}^{O} \mathbf{W}_{o,m} \cdot \mathbf{X}_{x+o-\frac{O-1}{2},m} \tag{6.2}$$

See that $\mathbf{S}_x = \sum_m^M \hat{\mathbf{S}}_{x,m}$. A *pointwise convolution* would be equivalent of applying a *regular* convolution with $O = 1$.

The *depthwise-separable convolution* is composite function of a *pointwise convolution* and a *depthwise convolution* as follows: $PW(DW(X, W_d), W_p)$ where PW and DW refer to the *pointwise* and *depthwise convolutions*, and W_p, W_d to their weights, similarly.

The *regular convolution* layer targeting R output channels has a computational cost of $D \cdot M \cdot R \cdot O$. The *depthwise convolution* has a computational cost of $D \cdot M \cdot O$, and the *separable convolution* $D \cdot M \cdot R$. Therefore the *depthwise-separable convolution* has a computational cost of $D \cdot M \cdot O + D \cdot M \cdot R$, i.e. the sum of its two constituting operations. Hence, by using the *depthwise-separable convolution* in place of the *regular* one, the cost reduces as follows:

$$\frac{D \cdot M \cdot O + D \cdot M \cdot R}{D \cdot M \cdot R \cdot O} = \frac{1}{R} + \frac{1}{O}$$

6.4.4 *Xception-1d* architecture

The proposed architecture exploits the gain in computational efficiency of the *depthwise-separable convolution* operation over the *regular convolution* in the same way original *Xception* does (Chollet, 2017). That allows a more efficient resource and time management and hence, a more complex architecture can be defined.

The *Xception-1d* architecture is shown in Figure 6.3. It has a total of 37 layers, 34 of which are *depthwise-separable convolutional layers*, 2 of them are *regular convolutions*, and the last one is a dense layer performing a logistic regression. The network contains 12 residual connections, one in each residual block, as shown in Figure 6.4. All of it builds up a network with up to 21 million parameters in the case of the *left-right recognition* task and 23 million parameters in the *35-words recognition* one[3].

[3]The difference lays on the last layer implementing the logistic regression of the architecture. Depending on the number of outputs required by each task, the number of parameters in this layer varies. The maximum difference occurs between the *35-words recognition* task and the *left-right recognition*

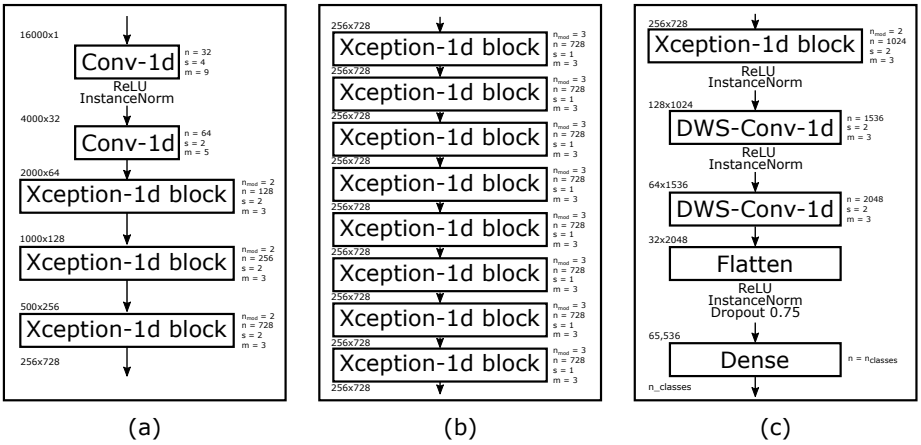

Figure 6.3: Diagram of the architecture used in this work where n refers to the number of channels, s is the stride of the convolutions (which in the case of the *Xception-1d blocks* is applied through the final pooling operation), m is the size of the convolution filters, n_{mod} is the number of the *depthwise convolutional* layers stacked in the *Xception-1d blocks* (see Figure 6.4) and $n_{classes}$ is the number of outputs of the network (i.e. the number of classes to predict, which depends on the task being solved). The architecture is built up in 3 main modules: (a) the entry module is the part of the network responsible for adapting the input wave into a condensed representation (by applying strides after each operation), (b) the middle module is responsible for learning the representation extracting the useful features that will allow the next module to distinguish between classes and (c) the classification module is responsible for mapping the extracted features into the number of outputs required for every task.

Considering that the last dense layer can be prone to overfitting due to the high number of parameters ($\sim 65,000$), a strong dropout (I. Goodfellow et al., 2016; Srivastava et al., 2014) has been applied after the last convolution ($p = 75\%$). In addition, a small *weight decay* (I. Goodfellow et al., 2016; Haykin, 1999; Krogh & Hertz, 1991) has been applied over all the network weights (specifically $\lambda = 10^{-3}$) in order to enhance regularization. *Adam* optimizer has been used to train the network (Kingma & Ba, 2014). The initial learning rate has been fixed to $\eta = 10^{-4}$. It has been decreased by a factor of $\frac{1}{2}$ when no improvement was observed (i.e. when a *plateau* is reached) with a patience of 4 epochs.

Instance normalization has been used to normalize the intermediate outputs of each convolution (Ulyanov et al., 2016; Z. Xu et al., 2018). It simply consists of standardizing separately each of the channels of every instance. Although it is typically used in generative modeling and style transfer efforts, it demonstrated to be very useful to improve training time of convergence and it has no undesirable effects at test time[4] (Ulyanov et al., 2016); it is considered a key element of this architecture. The input of the dense layer has been normalized using *layer normalization* (Ba et al., 2016). *Batch normalization* (Ioffe & Szegedy, 2015) has been avoided because it is prone to introduce *covariance shift*, degrading generalization (Ba et al., 2016). In figure 6.5 the way these three normalization techniques operate is represented.

[4]In the case of *Batch Normalization*, there can appear big differences of performance between train time and test time (Ba et al., 2016)

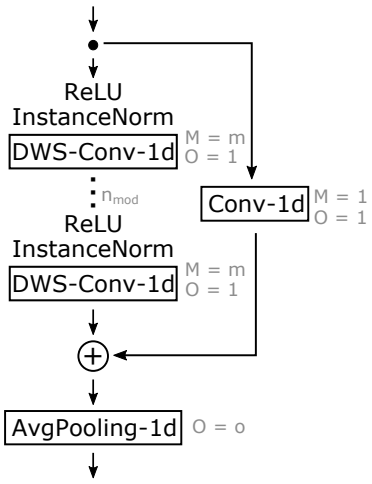

Figure 6.4: Neural network module refered as *Xception-1d block*. The *Xception-1d block* represents the building block of the architecture proposed in this work. It consists of a set of n_{mod} *1-D depthwise-separable convolution* layers with a *residual connection* and with a *1-D average pooling* layer at the end. The activation function used is the *rectified linear unit* (ReLU) and the normalization procedure after each convolutional layer is the *instance normalization* for one dimensional sequence.

6.5 Experiments and results

6.5.1 Setup

The train/development/test split provided by the authors of the dataset (Warden, 2017) has been adopted as *cross validation* (CV) setting in order to facilitate future benchmarking efforts. A simple split CV has been used instead of k-fold CV for the following reasons: (1) these models are expensive to train. Forty deep models (2 dataset versions $\times$ 4 tasks $\times$ 5 random initializations) were trained per experiment. If k-fold CV was used it would become unfeasible; (2) for the sake of reproducibility and benchmarking, as the authors of the dataset provide a default list of clips to use as validation and test.

Versions V1 and V2 of the dataset have 16,000 and 9,981 hold out samples for development purposes and 16,000 and 11,005 hold out samples for testing purposes, respectively. The development set has been used to manually tune the hyper-parameters and for *early stopping* purposes. The model has been trained for 50 epochs in each case, with a batch size of 32 clips, and the weights of the epoch that achieved the best performance in the development set were checkpointed. The checkpointed models have been used to calculate and report the performance of the algorithm.

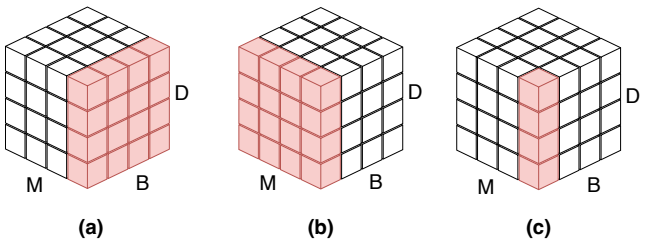

Figure 6.5: Each cube represents a batch of data where B is the instances axis (with the batch size), M is the channels axis and D is the spatial/temporal dimension. (a) represents *batch normalization*, (b) is *layer normalization* and (c) shows *instance normalization*. The shaded areas in the cubes represent the dimensions that are aggregated in each case for computing the statistics used to normalize the data (generally the mean and the standard deviation). As it can be noticed, the only normalization technique that depends on instances of data in the batch is the *batch normalization*, that is why it is more sensitive to train-test distribution differences. As it can be noticed in the picture, *instance normalization* is a specific version of *layer normalization* where each channel is normalized separately.

6.5.2 Results

All the results shown in this section have been measured over the test set. Five different models have been trained for each task in order to explore and report the effect of different random initializations of the weights of the network. With the aim of providing a baseline, human performance has been measured by 4 human subjects, who manually labeled ~ 1000 commands achieving different results. These results are reported in Table 6.2 along with the results of the proposed algorithm and the results reported by (Coimbra de Andrade et al., 2018; McMahan & Rao, 2017; Warden, 2018; Y. Zhang et al., 2017) on the matching tasks.

Besides the global results, figure 6.6 shows the precision and the recall obtained for the most complex model (*35-words-recognition* for data version V2). In conjunction with this figure, precision, recall and f1-score metrics for the task left-right and 35-commands have been included in the tables 6.3 and 6.4, respectively. These results are discussed in detail in section 6.6.

6.6 Discussion

The presented method, *Xception-1d*, offers better performance than the existing methods in the literature for three out of the four tested tasks, using different sets of limited-size vocabularies. According to the results in Table 6.2, *Xception-1d* performed voice recognition better than the state of the art methods (Coimbra de Andrade et al., 2018; McMahan & Rao, 2017; Warden, 2018; Y. Zhang et al., 2017) in three out of four tasks: *35-words-recognition*, *20-commands-recognition*, and *10-commands-recognition*. In the only task where *Xception-1d* did not achieve the best results (left-right), the leading method was the one proposed by Andrade *et al.* (Coimbra de Andrade et al., 2018) which was only marginally better (<0.5% performance difference on the test set) than the presented method. *Xception-1d* even surpassed human performance (with statistical significance level) in the two first tasks, including the most difficult one (*35-words-recognition*).

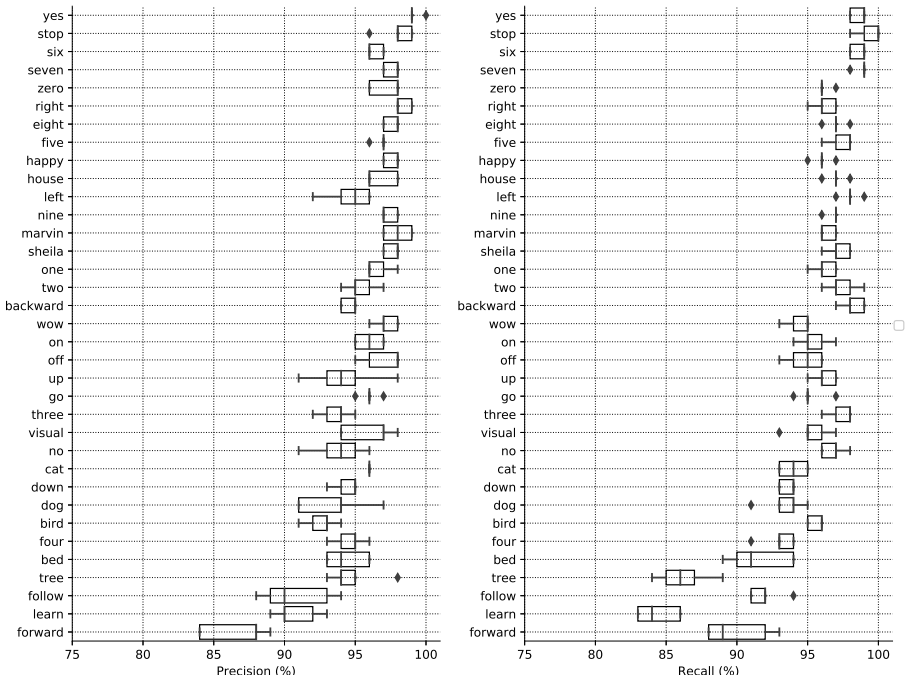

Figure 6.6: Precision and recall for each of the different classes using the *35-words-recognition* model trained with data version V2. Classes are sorted by descending f1-score.

Table 6.2: Accuracy (mean ± standard deviation) obtained by the proposed solution on the different tasks compared to other benchmark algorithms (described in (Coimbra de Andrade et al., 2018; McMahan & Rao, 2017; Warden, 2018; Y. Zhang et al., 2017)) and compared to the measurement of human accuracy (through 4 manual evaluations). The results of best performing algorithms for each task have been highlighted in bold in each case. Results better than human performance (given a statistical significance level of $\alpha = 0.05$) have been tagged with a star mark (*).

(a) Results for version 1 of the dataset.

	Coimbra	*McMahan*[a]	*Warden*	*Xception-1d*	Human	p-value[b]
35-words	94.30	84.35	-	**95.85 ± 0.12 ***	94.15 ± 1.03	$1.46 \cdot 10^{-2}$
20-commands	94.10	85.52	-	**95.89 ± 0.06 ***	94.56 ± 0.98	$3.14 \cdot 10^{-2}$
10-commands	95.60	-	85.40	**97.15 ± 0.03**	97.22 ± 0.85	$8.75 \cdot 10^{-1}$
left-right	**99.20**	95.32	-	98.96 ± 0.09	99.54 ± 0.16	$5.24 \cdot 10^{-4}$

(b) Results for version 2 of the dataset.

	Coimbra	*Zhang*[a]	*Warden*	*Xception-1d*	Human	p-value [b]
35-words	93.90	-	-	**95.85 ± 0.16 ***	94.15 ± 1.03	$1.50 \cdot 10^{-2}$
20-commands	94.50	-	-	**95.96 ± 0.16 ***	94.56 ± 0.98	$2.70 \cdot 10^{-2}$
10-commands	96.90	95.40	88.20	**97.54 ± 0.08**	97.22 ± 0.85	$4.84 \cdot 10^{-1}$
left-right	**99.40**	-	-	99.25 ± 0.07	99.54 ± 0.16	$1.27 \cdot 10^{-2}$

[a] the best results obtained among all the trials performed by the autors have been selected
[b] Student's t-test for the comparison of two means. $\alpha = 0.05$

With regard to per-class accuracy, it can be noticed from Figure 6.6 that the algorithm performs generally well for all the classes in the most difficult

scenario (35-words task) as the majority of precision and recall values lay between 90-100%. Nonetheless, the algorithm has more difficulties differentiating some groups of similar words like the following pairs: "three" and "tree", "follow" and "four", "bed" and "bird", etc. No comparison with other existing models has been included because no such detailed results have been found in the related work.

Finally, the findings show that the per-class performance of the speech commands in the *left-right* task is lower than for the other tasks (in particular the recall values) (see tables 6.4 and 6.3). However, the per-class performance for these two classes was higher when they were included in a multiclass classification task like the *35-words-recognition* task (Caruana, 1997). This evidence motivates the hypothesis that auxiliary tasks (in this case the 35-words vocabulary task) are benefiting primary tasks (left-right), as more features may be extracted from more complex tasks. This hypothesis has been proven for reinforcement learning (Jaderberg et al., 2016), but may also apply to DL efforts.

6.7 Conclusions

This chapter provided the insights of the study and implementation of an *Xception* based architecture (Chollet, 2017), named *Xception-1d*, to the speech commands recognition problem.

The experiments conducted give empirical evidence on how a neural network architecture which succeeded in the computer vision field, with an adaption and a set of tweaks, is able to surpass human performance at a speech recognition task with limited vocabulary achieving state of the art results. This motivates the suggestion of *Xception-1d* as the *de facto* architecture when facing a voice command recognition task with restricted vocabulary. The algorithm presented can have multiple applications for improving voice-controlled systems.

A possible future line of work could be the use of *Xception-1d* architecture with a global pooling layer at the end as an encoder of a *sequence-to-sequence* architecture for tackling a speech recognition task with free vocabulary (e.g. a *speech to text* engine). In addition to the usage of efficient convolutions, exploring pruning and complexity reduction techniques is recommended to further reduce the computational cost of the proposed solution.

Table 6.3: Detailed results for task *left-right* and data version V2, sorted by decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
left	95.80±0.98	94.00±0.89	95.00±0.63	412
right	98.20±1.47	90.60±2.65	94.00±1.10	396
unknown	99.40±0.49	100.00±0.00	100.00±0.00	10197

Table 6.4: Detailed results for task *35-words-recognition* and V2 dataset, in decreasing f1-score order. The columns "precision", "recall" and "f1-score" have been represented as the mean ± the standard deviation in percentage scale.

	precision	recall	f1-score	support
yes	99.20±0.40	98.60±0.49	99.00±0.00	419
stop	98.00±1.10	99.40±0.80	98.60±0.49	411
seven	97.60±0.49	98.80±0.40	98.40±0.49	406
six	96.40±0.49	98.60±0.49	97.60±0.49	394
right	98.40±0.49	96.20±0.75	97.40±0.49	396
sheila	97.60±0.49	97.20±0.75	97.20±0.75	212
nine	97.40±0.49	96.80±0.40	97.20±0.40	408
eight	97.60±0.49	97.00±0.63	97.20±0.40	408
marvin	98.00±0.89	96.40±0.49	97.20±0.40	195
five	96.80±0.40	97.40±0.80	97.00±0.00	445
house	96.80±0.98	97.00±0.63	96.80±0.75	191
happy	97.60±0.49	96.00±0.63	96.80±0.40	203
zero	97.20±0.98	96.20±0.40	96.60±0.49	418
left	94.60±1.50	98.00±0.63	96.40±0.80	412
backward	94.60±0.49	98.20±0.75	96.40±0.49	165
one	96.60±0.80	96.20±0.75	96.40±0.49	399
two	95.40±1.02	97.40±1.02	96.20±0.75	424
wow	97.20±0.75	94.40±0.80	96.00±0.89	206
off	97.00±1.26	94.80±1.17	95.80±0.75	402
on	96.00±0.89	95.40±1.02	95.80±0.40	396
visual	96.00±1.67	95.20±1.33	95.60±0.80	165
go	96.00±0.63	95.20±0.98	95.40±0.49	402
no	93.80±1.72	96.80±0.75	95.40±0.49	405
up	94.20±2.32	96.20±0.75	95.20±0.98	425
cat	96.00±0.00	94.00±0.89	95.20±0.75	194
three	93.60±1.02	97.40±0.80	95.20±0.75	405
four	94.60±1.02	93.00±1.10	94.00±0.63	400
bird	92.60±1.02	95.60±0.49	94.00±0.63	185
down	94.40±0.80	93.60±0.49	94.00±0.00	406
dog	93.40±2.24	93.40±1.36	93.40±0.80	220
bed	94.40±1.36	91.60±2.06	93.20±1.17	207
follow	90.80±2.32	92.00±1.10	91.60±1.36	172
tree	94.80±1.72	86.20±1.72	90.40±1.20	193
forward	86.60±2.15	90.00±2.10	88.20±1.72	155
learn	90.80±1.47	84.40±1.36	87.60±1.02	161

Chapter 7

Multi-speaker text-to-speech modeling

7.1 Overview

Text-to-speech systems recently achieved almost indistinguishable quality from human speech. However, the prosody of those systems is generally flatter than natural speech, producing samples with low expressiveness. Disentanglement of speaker id and prosody is crucial in speech generation systems to improve on naturalness and produce more variable syntheses (Marković et al., 2015).

This chapter proposes a new neural text-to-speech model that approaches the disentanglement problem by conditioning a *Tacotron2*-like architecture on flow-normalized speaker embeddings, and by substituting the reference encoder with a new learned latent distribution responsible for modeling the intra-sentence variability due to the prosody. By removing the reference encoder dependency, not only the speaker-leakage problem typically happening in this kind of systems disappears, producing more variable syntheses at inference time, but the system becomes much more lightweight, not depending on a production TTS system to be executed. This is a first step that enables low-resource on-device offline execution The new model achieves significantly higher prosody variance than the baseline in a set of quantitative prosody features, as well as higher speaker distinctiveness, without decreasing the speaker intelligibility. Finally, the normalized speaker embeddings enable much richer speaker interpolations, substantially improving the distinctiveness of the new interpolated speakers.

7.2 Introduction

In the last five years speech technologies have improved considerably. The text-to-speech (TTS hereafter) field has been largely benefited by the rise of deep learning (Sisman et al., 2021), which allowed these systems to achieve near-human performance at synthesizing speech that is almost indistinguishable from human's. One of the first achievements was *WaveNet* (van den Oord, Dieleman, et al., 2016), a neural vocoder based on dilated causal convolutions that was able to surpass its predecessors in naturalness in 2016. With the rise of the attention mechanism and its variants (Bahdanau et al., 2015; Chaudhari et al., 2021; Vaswani et al., 2017), *Tacotron* (Y. Wang et al., 2017) and *Tacotron 2* (R. Liu et al., 2019; Shen et al., 2018) proposed a sequence-to-sequence architecture as an end-to-end TTS solution. These models were able to map input text (or phonemes) to a spectrogram that, given the right neural vocoder (*WaveNet* for instance), would be converted into a sequence of waveform samples.

TTS is fundamentally a generative modeling problem because a given sentence can be mapped to multiple utterances with different prosody and speaker characteristics (Taylor, 2009). *Tacotron*, in its initial form, is a supervised model that performs a hard mapping between the input text and the output spectrograms. Therefore, the utterances generated using *Tacotron*-like architectures, although natural, they follow the average prosody of the training set, not allowing to generate utterances of a sentence with multiple speaking styles.

7.3 Previous work

The authors of (Skerry-Ryan et al., 2018) attempt to remediate the lack of expressiveness of the model by introducing a reference encoder consisting of a latent distribution that is conditioned on a reference mel-spectrogram (usually the target spectrogram, at training time). This distribution is learned by the model through a bottleneck that is intended to capture prosody aspects of the target spectrogram, preventing phonetic or speaker information to flow through. In practice, specially when using this approach in multi-speaker settings, the model tends to leak speaker information to the output. This represents a problem at inference time, when a synthetic neutral reference is provided (generally synthesized using a production system similar to *Amazon Polly*), given that the synthesized speaker identity tends to resemble the reference instead of the target voice. This problem is known as speaker leakage and it was already reported in (Skerry-Ryan et al., 2018), where the authors emphasize the importance of properly tuning the size of the reference bottleneck to amend it. In addition, the usual dependence of this model in a production TTS system makes them unfeasible for low resource, on-device or offline implementations, besides their obvious negative impact in energy efficiency.

In (R. Liu et al., 2020), the authors propose using a multi-task version of *Tacotron* to enhance the prosody of the syntheses. The approach described in the paper consists of jointly learning the target mel-spectrogram as well as the probability distribution of the phrase break patterns for each word. The work of (R. Liu et al., 2020) proposes a novel training schema for *Tacotron* where deep style features are extracted using the *SER* framework ((Lotfian & Busso, 2019; S. Zhang et al., 2018)). These features are later used for minimizing the style

differences between the real and synthesized samples.

This chapter's contribution builds upon a multi-speaker *Tacotron* architecture with a reference encoder (similar to (Skerry-Ryan et al., 2018)) and propose two modifications: (1) replaces the pre-trained speaker embedding with a normalized speaker embedding using normalizing flows (Kingma & Dhariwal, 2018), that allows sampling from the learned *Gaussian* distribution and (2) substitutes the reference encoder with a residual encoder, which learns a latent distribution conditioned on the input phonetic information, allowing the model to capture and encode the residual attributes not present in the linguistic input and the speaker embedding.

Inspired by the work of (Raitio et al., 2020), the proposed model achieves significantly higher prosody variation by measuring the difference in variance between the baseline and the proposed model across a set of features derived from the fundamental frequency, the energy, the signal to noise ratio and the speaking rate. These features capture most of the prosody variables (pitch, speed, loudness and timbre) (Raitio et al., 2020) and allow performing objective comparisons between systems.

7.4 Methods

A *Tacotron2*-based sequence-to-sequence model with *location-based attention* (Y. Wang et al., 2017) is used as a baseline throughout this study. This baseline architecture is represented in figure 7.1-top. A couple of encoder branches are added over the initial *Tacotron* formulation to allow the model to synthesize multiple speakers: the speaker branch and the reference branch. The first one takes as input a pre-trained speaker embedding vector representing the speaker characteristics. This vector is obtained from the output of a speaker verification model trained to minimize a triplet loss. This speaker verification model is pre-trained using pairs of utterances, similar to (Ren et al., 2019). For every speaker, the vectors corresponding to all their utterances are pre-computed and then averaged to form the speaker vectors. As a result, a fixed size vector is produced for every speaker in the dataset. Additionally, the proposed architecture includes a reference branch that is conditioned on the target spectrogram (at training time) and is intended to learn a latent distribution summarizing the prosody information, similar to the proposal of (Skerry-Ryan et al., 2018).

The proposed architecture builds upon the baseline model, removing the dependency of the reference branch on the target spectrogram, because it tends to cause speaker/phonetic leakage (read the section 4 of the samples included in (Skerry-Ryan et al., 2018)). It also slows down the inference process as it requires a production TTS system to provide the references. Instead, a learnable variational latent space conditioned on the phonetic information is learned together with a set of learnable free parameters (named *residual branch*). The purpose of this branch is to encode the prosody information not present in the input linguistic features or in the speaker embedding vector in a new latent space (i.e. the residual prosody variance). Moreover, the speaker embedding vectors are normalized using normalizing flows based on the work of (Kingma & Dhariwal, 2018), so that the normalized vectors follow a *Gaussian* distribution.That change enables sampling from the speaker embedding latent space instead of just using the average embedding vector. This increases the coverage of the speaker

embedding space, allowing smoother interpolations between speaker embedding vectors. The architecture proposed is depicted in figure 7.1-bottom

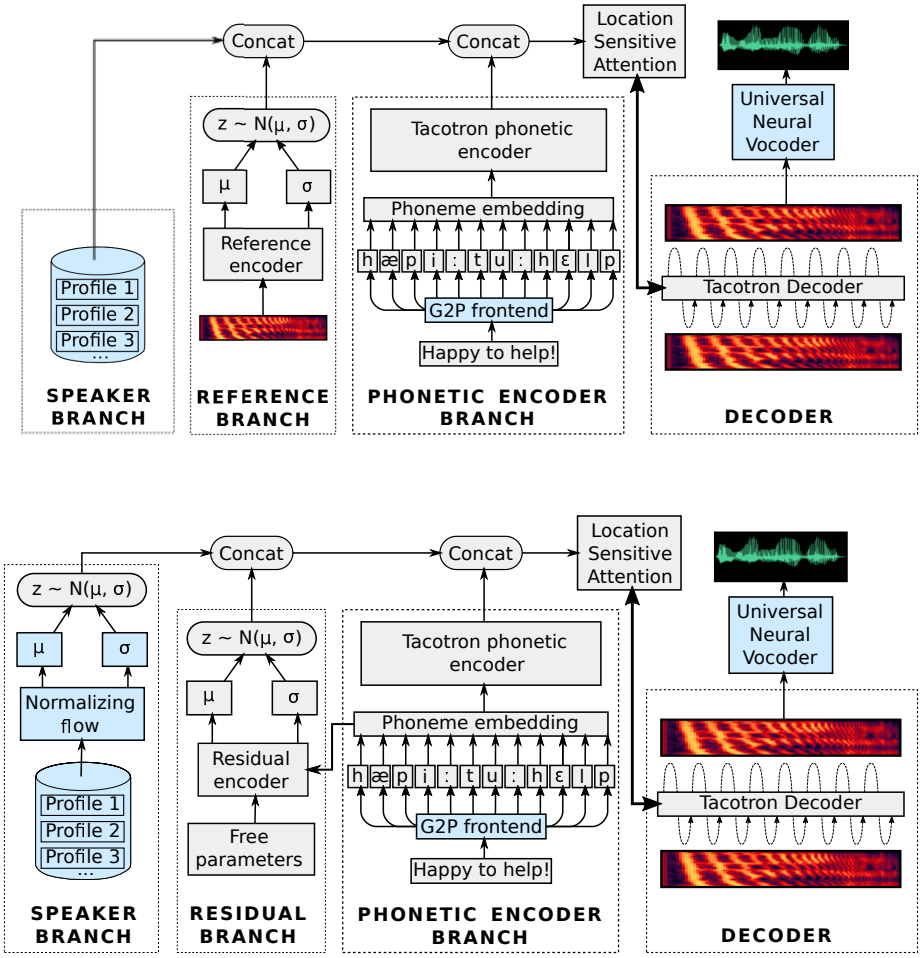

Figure 7.1: Top: the baseline architecture. Bottom: the proposed architecture. The blue blocks represent pre-trained parts of the network and the blocks with round corners are tensor operations.

7.4.1 Speaker embedding normalization using normalizing flows

Normalizing flows (Rezende & Mohamed, 2015) are powerful methods for producing tractable distributions from which one can sample. Through a sequence of invertible transformations they transform a complex data distribution to a tractable probability distribution. The output distribution is usually chosen to be an isotropic unit *Gaussian* to allow for smooth interpolations and efficient sampling.

Glow (Kingma & Dhariwal, 2018), a flow-based generative model, has shown significant improvements in computer vision generative modeling by adding invertible 1x1 convolutions to the sequence of transformations applied to the

input. To train this model on one-dimensional speaker embedding vectors, the 2-dimensional convolutions have been replaced with 1-dimensional ones.

A normalizing flow has been trained to normalize the pre-trained speaker embeddings described previously. The trained model attains a *Gaussian* distributed latent space of speaker embeddings from which one can easily sample to create embeddings representing new, unseen speakers. The proposed architecture samples from the *Gaussian* distribution defined by the normalized embeddings, both at training time and at inference time.

7.4.2 Residual branch

The new residual branch is designed to learn the prosody-induced variance that cannot be explained by the linguistic features and the speaker embedding vector alone. It takes as input the linguistic features and a set of learnable free parameters[1], conditioning on the target sentence and on a global prior. The motivation behind this design is to help the network learn different ways of uttering a given input sentence (linguistic features conditioning), and the potential global biases existing in the training dataset (free parameters).

The output of the phonetic encoder $\mathbf{h}_{\text{ph}}$ is piped into a bidirectional recurrent neural network (A. Graves et al., 2005; Schuster & Paliwal, 1997) in order to get a representation independent of time. The last output of the recurrent neural network of both, forward and reverse passes (represented in equations 7.1 and 7.2 as $\mathbf{o_f}$ and $\mathbf{o_r}$, respectively), are concatenated to form a fixed-size phonetic representation $[\mathbf{o_f}, \mathbf{o_r}]$. That vector is concatenated with a vector of free parameters $\mathbf{v_f}$ and the result is passed through two stacked dense layers to form the parameters of the residual latent distribution (as shown in equation 7.3, where g represents the *ReLU* activation function). The $h_{residual}$ vector is split in two vectors $h^{\mu}_{residual}$ and $h^{\sigma}_{residual}$ from where a latent vector z is sampled using the re-parametrization trick (Kingma & Welling, 2019). Finally, a *Kullback-Leibler* loss is included to assure that the distribution of the latent representation approximates an isotropic *Gaussian* distribution.

$$\mathbf{o_f} = \text{RNN}_f(\mathbf{h_{ph}}) \tag{7.1}$$

$$\mathbf{o_r} = \text{RNN}_r(\mathbf{h_{ph}}) \tag{7.2}$$

$$\mathbf{h_{residual}} = \mathbf{W_2}(g(\mathbf{W_1} \cdot ([\mathbf{o_f}, \mathbf{o_r}, \mathbf{v_f}]) + \mathbf{b1})) + \mathbf{b_2} \tag{7.3}$$

[1]Notice that the dense layers that produce the residual latent distribution parameters do not have bias terms, so that all the bias is learned in the free parameters vector.

7.5 Experiments and results

7.5.1 Setup

A combination of two internal datasets is used to train the models, one of them containing 2860 non-professional speakers, with 200 utterances per speaker on average, and the other containing 10 professional speakers, with 13,000 studio-recorded utterances per speaker – totaling to 2870 speakers and more than 700,000 utterances. The audio utterances are downsampled at 16kHz and 80 dimensional mel-spectrograms are extracted. The frame width is defied as a $50ms$ sequence, with an overlap of $12.5ms$. A universal neural vocoder is used to synthesize the wav samples (Lorenzo-Trueba et al., 2019). The linguistic features have been extracted using an internal front-end which takes the text as input and extracts the phonemes, that are used as input for the model. The speaker embedding, the free parameters and the residual distribution parameters μ and σ vectors were defined to have a length of 192 elements.

The models have been trained for one million steps, with a mini-batch size of 24. Then 50 unseen sentences are synthesized with each of the 2870 speakers voices. The syntheses have been evaluated in 3 ways: intelligibility, distinctiveness and prosody variability.

Additionally, 50 speakers are randomly sampled from the pool of 2870 speakers and generated interpolations (linear) between all the possible pairs, in order to study how the proposed model and the baseline behave in this setting.

7.5.2 Intelligibility

AWS transcribe system is used to transcribe each of the syntheses, and then measure the word error rate (*WER*) between the target sentence and the transcription (Kamath et al., 2019). These metrics are aggregated as the average *WER* per speaker. A median word error rate of 8.5% is achieved in the baseline, while a 8.3% WER is achieved by the proposed architecture. Although significative, this difference is very small, concluding that both models are roughly equivalent in terms of intelligibility.

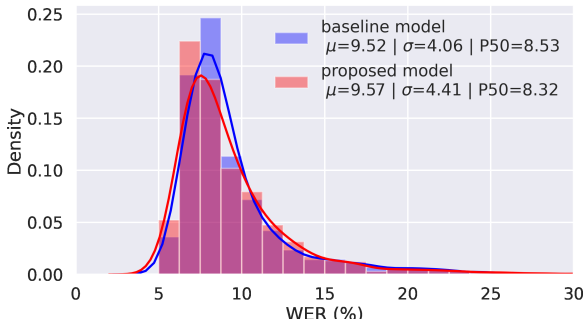

Figure 7.2: Speaker intelligibility of the baseline and residual models, measured as word error rate of the target and the transcribed synthesis, showing a slightly smaller error for the proposed architecture (Wilcoxon's $p = 0.0305$). The solid lines represent a kernel density estimate (KDE) over the histograms.

7.5.3 Distinctiveness

To measure how distinct the synthesized speakers are, *Alexa*'s internal *speaker verification system* is used. Each utterance is compared with 4 samples randomly drawn from the full pool of samples. The false acceptance rate (FAR) is used as a metric to quantify the percentage of speaker pairs incorrectly identified as the same speakers by the speaker verification model. Equation 7.4 shows how FAR is computed in detail, where $\mathcal{C}(i)$ is 1 if the ith pair of samples is misclassified as different speakers and N is the total number of pairs in the pool. Figure 7.3 shows how the FAR metric varies for both, the proposed model and the baseline as the classification threshold varies. Table 7.1 shows the FAR score for four arbitrarily picked thresholds. Lower values of FAR mean higher distinctiveness.

$$FAR = \frac{1}{N}\sum_{i=1}^{N}\mathcal{C}(i) \tag{7.4}$$

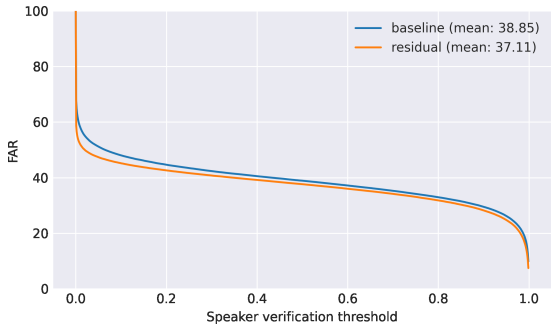

Figure 7.3: false acceptance rate distribution at different *speaker verification model* thresholds.

Table 7.1: false acceptance rate metric for different thresholds

Threshold	FAR-Baseline model	FAR-Proposed model
85%	31.49%	30.27%
90%	29.50%	28.21%
95%	26.23%	24.77%
99%	19.05%	16.93%

7.5.4 Prosody variability

To measure how much prosody and quality variance the new model introduces with respect to the baseline, a set of objective metrics has been defined inspired on the work of (Raitio et al., 2020). Those metrics are listed in table 7.2.

Table 7.2: Metrics used to quantify the prosody variation and quality across different models.

Metric	Description
f0 mean	Mean of the fundamental frequency (calculated using SPTK (Imai et al., 2017) considering only the vocal sounds (non-zero components), with frame skip of 12.5ms.
f0 range	Difference between the 5th and 95th percentiles of f0.
speaking rate	Calculated as the number of phonemes in the sentence divided by total duration of the synthesis.
f0 slope	Calculated as the slope of a linear fit using least squares in the f0 plot.
snr	Signal to noise ratio calculated using SOX (**SOX**) as the difference in RMS dBs between the loudest and the quietest windows, using windows size of 50ms.
power mean	$20 \log 10(\hat{x})$, where $\hat{x}$ is the average absolute amplitude with frame skip of 12.5ms. Only considering the vocal sounds.
power range	Difference between the 5th and 95th percentiles of the power along time.
power slope	Calculated as the slope of a linear fit using least squares in the power plot.

For this experiment, 50 speakers and 50 sentences have been picked. Then, 30 samples have been synthesized for every speaker-sentence pair varying the random seed, so different latent vectors are drawn from the latent distributions in each repetition. This procedure is repeated four times: (1) with the baseline model, (2) with the proposed model multiplying the σ parameter of the speaker embedding distribution by zero (so that the latent vector becomes the mean of the speaker embedding latent distribution), (3) with the proposed model multiplying the σ parameter of the residual distribution by zero, and (4) allowing sampling from both branches in the proposed model. Then the following comparisons are performed: the syntheses of (1) vs (2), (1) vs (3) and (1) vs (4) .

For the comparison, as the focus of this study is to measure differences of variance for the variables defined in the table, *bootstrap* is used to approximate the variance distribution for every variable in every speaker-sentence pair, and then a one-way *Wilcoxon Signed-Rank* sum test between the groups is conducted. A significance level of $\alpha = 5\%$ is used, over which the *Bonferroni* correction ($\alpha = 0.05/2500 = 0.002\%$) is applied. The results of the tests are summarized in the figure 7.4.

Informal listening tests of the syntheses confirm that when sampling from the residual distribution (keeping the speaker embedding distribution constant), the variations in the syntheses are related with prosody aspects like the speaking rate, syllable duration or intonation, keeping the speaker identity untouched. When sampling from the speaker embedding distribution, small variations on the speaker identity are noticed.

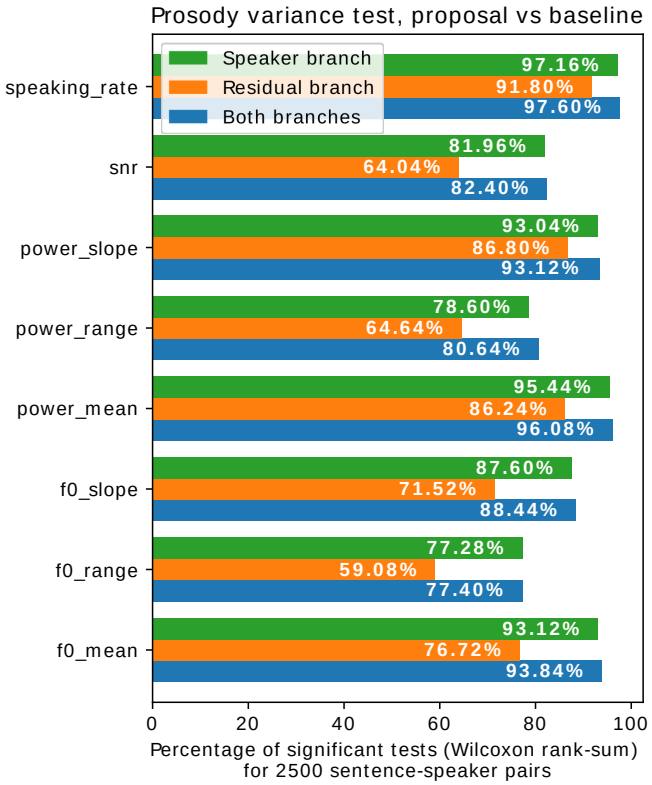

Figure 7.4: Baseline model vs proposed model when sampling from the residual latent distribution (blue) or from the speaker embedding distribution (orange), keeping the other constant. The results are represented as proportion of speaker-sentence pairs for which the proposed model shows significantly higher variance than the baseline, for each of the previously defined prosody features.

7.5.5 Speakers interpolation

The proposed model and the baseline are tested using interpolated speakers to study how the speaker embedding normalization affects the intelligibility and distinctiveness of the new speakers. For that, 50 speakers have been chosen. All the pairs of speakers have been interpolated using linear interpolation, generating 1225 new voice profiles. Fifty sentences have been synthesized using those new voices and then evaluated distinctiveness and intelligibility (results in figure 7.5).

The proposed model achieves 13.11% lower FAR than the baseline. The rationale behind this is that the normalized speaker embedding space has a denser latent distribution than the one in the baseline. Given that the proposed model is trained on samples drawn from the normalized speaker embedding distribution (as opposed of using average vectors as in the baseline model), the denser latent space leads to a better generalization.

From the intelligibility perspective, the baseline shows an average WER of 6.42% while the proposed model achieved a 7.35%. That difference is attributed to the fact that the interpolated speakers in the baseline are less distinctive and resemble much more to one of the two actual speakers, hence its intelligibility is naturally higher at the cost of a worse distinctiveness. Although significative,

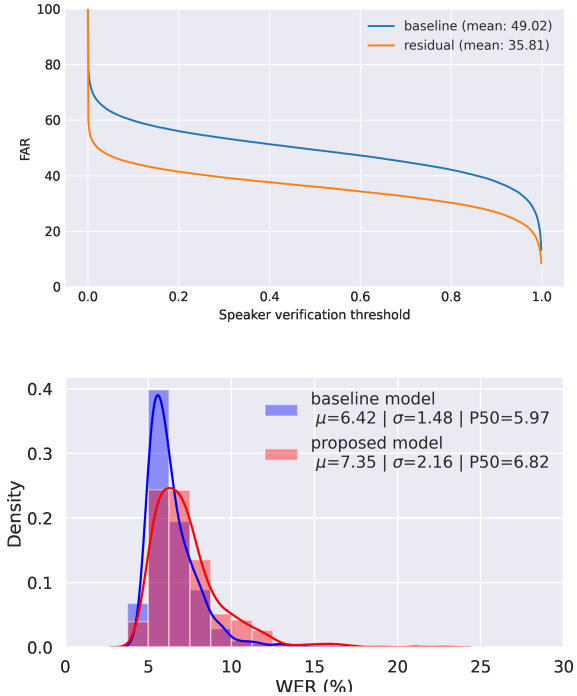

Figure 7.5: Top: false acceptance rate distribution at different speaker verification thresholds, for all the interpolated speakers. Bottom: speaker intelligibility of the baseline and residual models, measured as word error rate of the target and the transcribed synthesis, showing a slightly smaller error for the baseline architecture (Wilcoxon's $p = 5.2263 \cdot 10^{-6}$). The solid lines represent a KDE over the histograms.

the difference in practice is negligible.

7.6 Conclusions

This chapter presents a new TTS architecture that allows increasing the prosody variance of a multi-speaker TTS system by learning the residual prosody into a new latent distribution.

It has been also showed that by sampling from the residual and normalized speaker latent distribution the model produces syntheses with significantly different prosodies as measured by a set of quantitative metrics. The inclusion of the residual distribution also enables removing the reference spectrogram dependency. This does not only allow for a faster and cheaper inference, but also solves potential speaker leakage issues, given that the model does no longer depend on a voice production system to work.

Finally, the normalization of the speaker embedding latent space allows for better speaker interpolation when compared with the baseline model, thus producing more diverse and unseen synthetic speakers.

Chapter 8

General conclusions

This dissertation ecompasses five contributions related to the low-resource deep learning field, showing how improvements of state of the art can be made without necessarily needing to spend a prohibitive amount of resources.

- The modulus activation function, introduced in chapter 3, is a living example of the premise of this thesis, showing that a bit-size operation can outperform the most novel and complex activation functions in 75% of the experiments conducted, despite its simplicity. The computational cost of the modulus activation function is equivalent to the $ReLU$'s, but its gradient constant norm guarantees a better utilization of the network parameters, removing the "dying neurons" problem and often leading to more efficient training processes.

- Transfer learning also showed recently that impressive results can be achieved by just fine-tuning a few thousands of the millions of weights of a large network which has been previously trained on a general pretext task. Knowledge distillation techniques allow leveraging the knowledge of big networks to implant it into small ones. Combining both techniques has been a clear objective for the study line of this thesis. Chapter 4 provides evidences around the benefit of increasing the accuracy of a small pre-trained neural network up to 3% (in absolute terms), by just learning from their large counterparts and using solely unlabeled data. This case study motivates the idea that the small neural networks are not yet at their capacity limit. Based on the results of this study, a possible future research line may be looking for more efficient learning techniques that enable the use all the modeling capacity of the parameters of a neural network. However this was not the objective of this study.

- Thinking of applications, the outstanding generalization capacity of deep learning models, combined with the flexibility and customizability of their architecture, enables one to design models that solve many tasks at once. One of the main benefits of having a single model, as opposed to train standalone models for each individual task, is that the amount of total parameters, the effort required and the computational cost of tuning the hyper-parameters is smaller than tackling each problem with a different model. Besides, recent studies have shown that when a neural network is

asked to perform many tasks, the overall performance increases (Jaderberg et al., 2016) as if the model had more appeal to learn. These ideas motivated the design of the sales forecasting problem as an end-to-end solution, as described in chapter 5. It shows how to train a single model to predict the daily number of sales each of 4400 items in 54 different points of sale, and show that the proposed solution is better than all the published benchmarks. In a production environment, this means that a company like *Corporación Favorita* (the owners of the open-source dataset used in the experiments) would be able to deploy the model in a single machine which would serve all their points of sale (as opposed of having to build 4400 products $\times$ 54 points of sale = 237600 standalone models). Overall, this would result in lower infrastructure and computational costs.

- Speech technologies also got benefited by the fast development of deep learning technologies of the last years. Generally speaking, the computational speech field is divided in two main branches: speech recognition and speech generation; deep learning has revolutionized both of them.
  - The contribution to the first branch, developed in chapter 6 in form of a keyword spotting model, was motivated by the outstanding performance of CNNs in the computer vision field. An architecture designed for computer vision tasks has been adapted to the problem of keyword spotting, achieving state of the art results when compared with all the reported benchmarks. The *Xception* architecture is designed to work with depthwise-separable convolutions, which is an efficient modification of the classical convolution operations used in deep learning (as shown in section 6.4.3). Along with this case comes the design and implementation of a (i) *depthwise-separable* CNN-based architecture able to surpass the current state of the art results and the human performance, (ii) the development of a methodology for augmenting audio data to increase the size of a dataset (5x in this work), and therefore enhance generalization, (iii) the quantification of the human performance across the different classification tasks to use it as an additional baseline for checking the results achieved by the algorithm and (iv) the creation of a public repository where further contributions could be handled to enhance the project functionalities and to facilitate reproducibility.
  - Chapter 7 contributes to the speech generation field by proposing a modification to a baseline architecture that brings computational and quality improvements in form of prosody variation. The computational improvements come from the dependency of the baseline architecture on a production speech generation system: for generating a new audio clip, the baseline model needed to get an example of that utterance as input in order to use it as a reference. For that, a commercial solution is normally used (e.g. *Amazon Polly*). The proposed architecture replaces the reference encoder by a residual branch and a normalizing flow, which together build a new TTS architecture that, as empirically proven in this chapter, generates syntheses with more varied prosody. The proposed architecture is capable of working on a single offline computer, not depending on a production TTS service.

The potential of deep learning has not yet been fully explored, there is still much research to be done. The recent success of deep learning has stood out by the amount of resources these algorithms require, which has made it inaccessible to many organizations. The work described in this dissertation shows that it is not necessarily true. The computational requirements are not an impediment for solving real-life problems. The research on low-resource deep learning has recently started to gain traction and it is expected to snag the attention of many researchers and practitioners in the coming years.

As the momentum grows, the amount and variety of problems that will be solved using deep learning will also increase, bringing new challenges with it. Looking back at the work reported in this dissertation, relevant contributions have been made to the research field focused on lowering the resource requirements of the deep learning models: two general contributions and three applications showing that many problems can be solved while using a reasonable use of resources.

This is a very important area of study, because lowering the bar of the resources requirements will open the door to the use of deep learning technologies in a larger set of use cases and application domains. The area of low-resource deep learning will probably be one of the most important areas of research in the coming years, and hopefully the work reported in this dissertation will contribute to the advancement of this area of study.

Bibliography

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A. I., et al. (1996). Fast discovery of association rules. *Advances in knowledge discovery and data mining*, *12*(1), 307–328.

Algan, G., & Ulusoy, I. (2021). Image classification with deep learning in the presence of noisy labels: A survey. *Knowledge-Based Systems*, *215*, 106771. https://doi.org/10.1016/j.knosys.2021.106771

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein Generative Adversarial Networks. *Proceedings of the 34th International Conference on Machine Learning (ICML)*, *70*, 214–223.

Asif, U., Tang, J., & Harrer, S. (2020). Ensemble Knowledge Distillation for Learning Improved and Efficient Networks. *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI)*.

Aung, H., Bobkov, A. V., & Tun, N. L. (2021). Face Detection in Real Time Live Video Using YOLO Algorithm Based on VGG16 Convolutional Neural Network. *Proceedings of the International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, 697–702. https://doi.org/10.1109/ICIEAM51226.2021.9446291

Ayachi, R., Afif, M., Said, Y., & Atri, M. (2020). Strided Convolution Instead of Max Pooling for Memory Efficiency of Convolutional Neural Networks. In M. S. Bouhlel & S. Rovetta (Eds.), *Proceedings of the 8th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*. Springer International Publishing.

Ba, J., Kiros, R., & Hinton, G. E. (2016). Layer Normalization. *Proceedings of the 30th Neural Information Processing Systems conference (NIPS)*.

Badri, H., Ghomi, S., & Hejazi, T. H. (2017). Supply Chain Network Design: A Value-based Approach. *Transportation Research Part E*, 1–17. https://doi.org/10.1016/j.tre.2017.06.012

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*.

Bell, P. C. (2000). Forecasting Demand Variation when there are Stockouts. *The Journal of the Operational Research Society*, *51*(3), 358–363. http://www.jstor.org/stable/254094

Benaroch, M., & Dhar, V. (1991). An intelligent assistant for financial hedging. *Proceedings of the 7th IEEE Conference on Artificial Intelligence Application*, *1*, 168–174. https://doi.org/10.1109/caia.1991.120865

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2006). Greedy layer-wise training of deep networks. *Proceedings of the 20th Neural Information Processing Systems Conference (NIPS)*.

Bi, J., Zhu, Z., & Meng, Q. (2021). Transformer in Computer Vision. *Proceedings of the IEEE International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology*, 178–188. https://doi.org/10.1109/CEI52496.2021.9574462

Bishop, C. M. (2011). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York Inc. https://www.ebook.de/de/product/5324937/christopher_m_bishop_pattern_recognition_and_machine_learning.html

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., ... Amodei, D. (2020). Language Models are Few-Shot Learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Procedings of the 33rd conference in Neural Information Processing Systems conference (NeurIPS)* (pp. 1877–1901). Curran Associates, Inc. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

Buś, S., & Jedrzejewski, K. (2016). Digital signal processing techniques for pitch shifting and time scaling of audio signals. *Proceedings of SPIE - The International Society for Optical Engineering*, *10031*, 1003157–1. https://doi.org/10.1117/12.2249374

Buuren, S. (2018). *Flexible imputation of missing data*. CRC Press. https://doi.org/10.1201/9780429492259

Calero, A. S. M., & Caro, J. M. B. (2018). *Corporación Favorita Grocery Sales Forecasting* (Master Thesis). Universidad Autónoma de Andalucía. https://dspace.unia.es/handle/10334/3921

Campbell, M., Hoane, A. J., & Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, *134*, 57–83. https://doi.org/10.1016/s0004-3702(01)00129-1

Carrera, R., Loiseau, D., & Roux, O. (1979). *Androides*.

Caruana, R. (1997). Multitask Learning. *Machine Learning*, *28*(1), 41–75. https://doi.org/10.1023/A:1007379606734

Chakraborty, D., Chiracharit, W., & Chamnongthai, K. (2021). Video shot Boundary Detection using Principal Component Analysis (PCA) and Deep Learning. *Proceedings of the 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, 272–275. https://doi.org/10.1109/ECTI-CON51831.2021.9454775

Chaudhari, S., Mithal, V., Polatkan, G., & Ramanath, R. (2021). An Attentive Survey of Attention Models. *ACM Transactions on Intelligent Systems and Technology*, *12*(5), 1–32. https://doi.org/10.1145/3465055

Chen, S., Wu, Y., Chen, Z., Wu, J., Li, J., Yoshioka, T., Wang, C., Liu, S., & Zhou, M. (2021). Continuous Speech Separation with Conformer. *Proceedings of the 46th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5749–5753. https://doi.org/10.1109/ICASSP39728.2021.9413423

Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. *Proceedings of the IEEE Conference on Computer Vision and*

Pattern Recognition (CVPR), 1800–1807. https://doi.org/10.1109/CVPR.2017.195

Chollet, F. et al. (2015). *Keras*. https://github.com/fchollet/keras

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *Proceedings of the 28th Neural Information Processing Systems Conference (NIPS), Deep Learning and Representation Learning Workshop*.

Cinar, A., Tatura, E., DeCicco, J., Raj, R., Aggarwal, N., Chesebro, M., Evans, J., Shah-Khan, M., & Zloza, A. (1999). Automated patient monitoring and diagnosis assistance by integrating statistical and artificial intelligence tools. *2*, 700. https://doi.org/10.1109/IEMBS.1999.803855

Clevert, D., Unterthiner, T., & Hochreiter, S. (2016). Fast and Accurate deep Network Learning by Exponential Linear Units (ELUs). In Y. Bengio & Y. LeCun (Eds.), *Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico*. http://arxiv.org/abs/1511.07289

Coimbra de Andrade, D., Leo, S., Loesener Da Silva Viana, M., & Bernkopf, C. (2018). A neural attention model for speech command recognition. *ArXiv e-print*, *abs/1808.08929*, Article arXiv:1808.08929.

Corporación Favorita, K. (2018). *Corporación Favorita Grocery Sales Forecasting Data Set* [Available in the following link: https://www.kaggle.com/c/favorita-grocery-sales-forecasting/data].

Cramer-Flood, E. (2020). Global Ecommerce 2020: Ecommerce Decelerates amid Global Retail Contraction but Remains a Bright Spot. *E-marketer*.

Cui, S., & Jiang, Y. (2017). Effective Lipschitz constraint enforcement for Wasserstein GAN training. *Proceedings of the 2nd IEEE International Conference on Computational Intelligence and Applications (ICCIA)*, 74–78. https://doi.org/10.1109/ciapp.2017.8167183

Curtin, R. R., Moseley, B., Ngo, H. Q., Nguyen, X., Olteanu, D., & Schleich, M. (2020). Rk-means: Fast Clustering for Relational Data. In S. Chiappa & R. Calandra (Eds.), *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS), Virtual, Palermo, Sicily, Italy* (pp. 2742–2752). http://proceedings.mlr.press/v108/curtin20a.html

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314. https://doi.org/10.1007/bf02551274

Dauphin, Y., & Cubuk, E. D. (2021). Deconstructing the regularization of batchnorm. *Proceedings of the 9th International Conference on Learning Representations (ICLR), Virtual Event, Austria*. https://openreview.net/forum?id=d-XzF81Wg1

David, O. E., & Greental, I. (2014). Genetic Algorithms for Evolving Deep Neural Networks. *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, 1451–1452. https://doi.org/10.1145/2598394.2602287

DeClaris, N. (1991). A systems approach to medical decision aiding. *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, 2103–2107 vol.3. https://doi.org/10.1109/icsmc.1991.169924

Deep, K., & Salhi, M. J. S. (2018). *Logistics, Supply Chain and Financial Predictive Analytics*. Springer Singapore. https://doi.org/10.1007/978-981-13-0872-7

Denby, E., & Gammack, J. (1999). The naming of colours: Investigating a psychological curiosity using AI. *Proceedings of the 6th Neural Information Processing Systems conference (NIPS)*, *3*, 964–973. https://doi.org/10.1109/iconip.1999.844667

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A large-scale hierarchical image database. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 248–255.

Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Denton, E. L., Cintala, S., Szlam, A., & Fergus, R. (2015). Deep Generative Image Models using Laplacian Pyramid of Adversarial Networks. *Proceedings of the 29th Neural Information Processing Systems conference (NIPS)*, *28*. https://proceedings.neurips.cc/paper/2015/file/aa169b49b583a2b5af89203c2b78c67c-Paper.pdf

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 4171–4186. https://doi.org/10.18653/v1/N19-1423

Dhariwal, P., & Nichol, A. (2021). Diffusion Models Beat GANs on Image Synthesis. *Proceedings of the 35th Neural Information Processing Systems conference (NeurIPS)*, *abs/2105.05233*. https://arxiv.org/abs/2105.05233

Ding, Q., Han, J., Zhao, X., & Chen, Y. (2015). Missing-Data Classification with the Extended Full-Dimensional Gaussian Mixture Model: Applications to EMG-Based Motion Recognition. *IEEE Transactions on Industrial Electronics*, *62*(8), 4994–5005. https://doi.org/10.1109/TIE.2015.2403797

Dinh, L., Sohl-Dickstein, J., & Bengio, S. (2016). Density estimation using Real NVP. *Proceedings of 4th International Conference of Learning Representations (ICLR)*. http://dblp.uni-trier.de/db/journals/corr/corr1605.html#DinhSB16

Dorrer, M., Gorban, A., & Zenkin, V. (1995). Neural networks in psychology: Classical explicit diagnoses. *Proceedings of the 2nd International Symposium on Neuroinformatics and Neurocomputers*, 281–284. https://doi.org/10.1109/isninc.1995.480869

Dua, D., & Graff, C. (2017). UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/datasets/abalone

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2001). Incorporating Second-Order Functional Knowledge for Better Option Pricing. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Proceedings of the 15th Neural Information Processing Systems conference (nips)*. MIT Press. https://proceedings.neurips.cc/paper/2000/file/44968aece94f667e4095002d140b5896-Paper.pdf

Evci, U., Dumoulin, V., Larochelle, H., & Mozer, M. C. (2022). Head2toe: Utilizing Intermediate Representations for Better Transfer learning. *ArXiv Preprint*, *abs/2201.03529*. https://arxiv.org/abs/2201.03529

Falas, T., Charitou, A., & Charalambous, C. (1994). The application of artificial neural networks in the prediction of earnings. *Neural Networks*, *6*, 3629–3633. https://doi.org/10.1109/ICNN.1994.374920

Falk, K. (2019). *Practical Recommender Systems* (Illustrated). Manning. https://www.ebook.de/de/product/28452266/kim_falk_practical_recommender_systems.html

Floridi, L., & Chiriatti, M. (2020). GPT-3: Its Nature, Scope, Limits, and Consequences. *Minds and Machines*, *30*(4), 681–694. https://doi.org/10.1007/s11023-020-09548-1

Forslund, H., & Jonsson, P. (2007). The impact of forecast quality on supply chain performance. *International Journal of Operations & Production Management*, *27*, 90–107. https://doi.org/10.1108/01443570710714556

Frankle, J., & Carbin, M. (2019). The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *Proceedings of the 7th International Conference on Learning Representations (ICLR)*.

Freeman, C. D., & Bruna, J. (2017). Topology and Geometry of Half-Rectified Network Optimization. *Proceedings of the 5th International Conference on Learning Representations (ICLR), Toulon, France*. https://openreview.net/forum?id=Bk0FWVcgx

French, R. (1999). Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, *3*(4), 128–135. https://doi.org/10.1016/s1364-6613(99)01294-2

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202. https://doi.org/10.1007/bf00344251

Gao, H., Wang, Z., & Ji, S. (2018). ChannelNets: Compact and Efficient Convolutional Neural Networks via Channel-Wise Convolutions. *Proceedings of the 32nd Neural Information Processing Systems conference (NeurIPS)*.

Gerlach, M., & Font-Clos, F. (2020). A Standardized Project Gutenberg corpus for Statistical Analysis of Natural Language and Quantitative Linguistics. *Entropy*, *22*(1), 126. https://doi.org/10.3390/e22010126

Geyer, R. C., Wegmayr, V., & Corinzia, L. (2019). Transfer Learning by Adaptive Merging of Multiple Models. *Proceedings of the International Conference on Medical Imaging with Deep Learning*.

Gidaris, S., Singh, P., & Komodakis, N. (2018). Unsupervised Representation Learning by Predicting image Rotations. *Proceedigns of the 6th International Conference of Learning Representations (ICLR)*.

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), *Proceedings of the 13th international conference on artificial intelligence and statistics* (pp. 249–256). http://proceedings.mlr.press/v9/glorot10a.html

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse Rectifier Neural Networks. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics* (pp. 315–323). http://proceedings.mlr.press/v15/glorot11a.html

Goodfellow, I., Bengio, J., & Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, & K. Q. Weinberger (Eds.), *Proceedings of the 28th Neural Information Processing Systems conference (NIPS)* (pp. 2672–2680). Curran Associates, Inc. http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

Goodfellow, I. J. (2016). Generative Adversarial Networks. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS), tutorial*. http://arxiv.org/abs/1701.00160

Gotmare, A., Shirish Keskar, N., Xiong, C., & Socher, R. (2019). A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation. *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=r14EOsCqKX

Gou, J., Yu, B., Maybank, S. J., & Tao, D. (2021). Knowledge Distillation: A Survey. *International Journal of Computer Vision*, *129*(6), 1789–1819. https://doi.org/10.1007/s11263-021-01453-z

Goyal, A., Lamb, A., Zhang, Y., Zhang, S., Courville, A., & Bengio, Y. (2016). Professor Forcing: A New Algorithm for Training Recurrent Networks. *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS)*, 4608–4616.

Graves, A. (2013). Generating Sequences with Recurrent Neural Networks. *ArXiv e-print*, *abs/1308.0850*. https://doi.org/10.48550/ARXIV.1308.0850

Graves, A., Fernández, S., & Schmidhuber, J. (2005). Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. *Proceedings of the International Conference of Artificial Neural Networks: Formal Models and Their Applications (ICANN)*, 799–804.

Graves, S., & Willems, S. (2008). Strategic Inventory Placement in Supply Chains: Nonstationary Demand. *Manufacturing & Service Operations Management*, *Vol. 10*, 278–287. https://doi.org/10.1287/msom.1070.0175

Greenwood, J., & Woodcroft, B. (1851). *The Pneumatics of Hero of Alexandria: From the Original Greek*. Taylor, Walton; Maberly. https://books.google.co.uk/books?id=O8PVAAAAMAAJ

Guo, Y., Li, Y., Feris, R., Wang, L., & Rosing, T. (2019). Depthwise Convolution is All You Need for Learning Multiple Visual Domains. *Association for the Advancement of Artificial Intelligence*.

Ha, T., Dang, T. K., Dang, T. T., Truong, T. A., & Nguyen, M. T. (2019). Differential Privacy in Deep Learning: An Overview. *Proceedings of the International Conference on Advanced Computing and Applications (ACOMP)*, 97–102. https://doi.org/10.1109/ACOMP.2019.00022

Ham, F. M., & Kostanic, I. (2000). *Principles of Neurocomputing for Science and Engineering* (1st). McGraw-Hill Higher Education.

Han, H., & Siebert, J. (2022). TinyML: A Systematic Review and Synthesis of Existing Research. *Proceedings of the International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, 269–274. https://doi.org/10.1109/ICAIIC54071.2022.9722636

Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., Yang, H., & Dally, W. (J. (2017). ESE: Efficient Speech

Recognition Engine with Sparse LSTM on FPGA. *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 75–84. https://doi.org/10.1145/3020078.3021745

Hassibi, B., Stork, D., & Wolff, G. (1993). Optimal Brain Surgeon and general network pruning. *Proceedings of the IEEE International Conference on Neural Networks*, *1*, 293–299. https://doi.org/10.1109/ICNN.1993.298572

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning : Data mining, inference, and prediction*. Springer. https://books.google.co.uk/books?id=eBSgoAEACAAJ

Haykin, S. (1999). *Neural networks : A comprehensive foundation* (2nd). Prentice Hall.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

Helmini, S., Jihan, N., Jayasinghe, M., & Perera, S. (2019). Sales forecasting using multivariate long short term memory network models. *PeerJ Preprints*, *7*, e27712v1. https://doi.org/10.7287/peerj.preprints.27712v1

Hendrycks, D., & Gimpel, K. (2016). Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units. *Proceedings of the 4th International Conference on Learning Representations (ICLR), San Juan, Puerto Rico*. https://openreview.net/forum?id=Bk0MRI5lg

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *Proceedings of the Neural Information Processing Systems conference (NIPS), Deep Learning and Representation Learning Workshop*.

Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, *14*(8), 1771–1800. https://doi.org/10.1162/089976602760128018

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. https://doi.org/10.48550/ARXIV.1207.0580

Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). A Field Guide to Dynamical Recurrent Networks. *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press. https://ml.jku.at/publications/older/ch7.pdf

Hochreiter, S. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*(2), 107–116. https://doi.org/10.1142/s0218488598000094

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Hodges, A. (2000). *Alan Turing : The enigma*. Walker.

Hoffer, E., Hubara, I., & Soudry, D. (2017). Train longer, generalize better: Closing the generalization gap in large batch training of neural networks.

Proceedings of the 31st Neural Information Processing Systems conference (NIPS). http://arxiv.org/abs/1705.08741

Hoffmann, E. T. A. (1816). *Der Sandmann*.

Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *National Academy of Sciences of the United States of America*, *79*(8), 2554–2558. http://view.ncbi.nlm.nih.gov/pubmed/6953413]

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., & Adam, H. (2017). MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. https://doi.org/10.48550/arXiv.1704.04861

Huang, G., Liu, Z., Maaten, L. V. D., & Weinberger, K. Q. (2017). Densely Connected Convolutional Networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2261–2269. https://doi.org/10.1109/cvpr.2017.243

Huang, G.-B., Zhou, H., Ding, X., & Zhang, R. (2012). Extreme Learning Machine for Regression and Multiclass Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *42*(2), 513–529. https://doi.org/10.1109/tsmcb.2011.2168604

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: Theory and applications. *Neurocomputing*, *70*(1-3), 489–501. https://doi.org/10.1016/j.neucom.2005.12.126

Hui, T.-W., Tang, X., & Loy, C. C. (2018). LiteFlowNet: A Lightweight Convolutional Neural Network for Optical Flow Estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Hyndman, R. (2018). *Forecasting : Principles and practice* (2nd). OTexts.

Ibn Shākir, M. I. M. (1979). *The Book of Ingenious Devices / kitáb al-hiyal* (1979th ed.). Kluwer Academic.

Imai, S., Kobayashi, T., & Tokuda, K. (2017). Speech Signal Processing Toolkit (SPTK) - Version 3.11.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, *37*, 448–456.

Iván Vallés-Pérez and Emilio Soria-Olivas and Marcelino Martínez-Sober and Antonio J. Serrano-López and Juan Gómez-Sanchís and Fernando Mateo. (2022). Approaching sales forecasting using recurrent neural networks and transformers. *Expert Systems with Applications*, *201*, 116993. https://doi.org/10.1016/j.eswa.2022.116993

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *Proceedings of the 5th International Conference of Learning Representations (ICLR), Toulon, France*.

Jaegle, A., Gimeno, F., Brock, A., Zisserman, A., Vinyals, O., & Carreira, J. (2021). Perceiver: General Perception with Iterative Attention. *Proceedings of the 38th International Conference on Machine Learning (ICML)*.

James, G., Witten, D., & Hastie, T. (2017). *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated. https://www.ebook.de/de/product/20292548/gareth_james_daniela_

witten_trevor_hastie_an_introduction_to_statistical_learning_with_applications_in_r.html

Jay, R. (2000). *The Automaton Chess Player, the Invisible Girl, & the Telephone*. R. Jay; W & V Dailey. https://books.google.ie/books?id=fkppYgEACAAJ

Jeon, Y. S., Yoshino, K., Hagiwara, S., Watanabe, A., Quek, S. T., Yoshioka, H., & Feng, M. (2021). Interpretable and Lightweight 3-D Deep Learning Model for Automated ACL Diagnosis. *IEEE Journal of Biomedical and Health Informatics*, *25*(7), 2388–2397. https://doi.org/10.1109/JBHI.2021.3081355

Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., & Yan, S. (2016). Deep Learning with S-Shaped Rectified Linear Activation Units. *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, 1737–1743. https://arxiv.org/abs/1512.07030

Johnson, L., & Hoback, A. (1991). From prototype to production: Expanding expert systems project management planning. *Proceedings of the IEEE International Conference on Developing and Managing Expert System Programs*, 286–294. https://doi.org/10.1109/DMESP.1991.171773

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., Bridgland, A., Meyer, C., Kohl, S. A. A., Ballard, A. J., Cowie, A., Romera-Paredes, B., Nikolov, S., Jain, R., Adler, J., … Hassabis, D. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, *596*(7873), 583–589. https://doi.org/10.1038/s41586-021-03819-2

Kaipia, R. (2009). Coordinating material and information flows with supply chain planning. *The International Journal of Logistics Management*, *20*(1), 144–162. https://doi.org/10.1108/09574090910954882

Kamath, U., Liu, J., & Whitaker, J. (2019). *Deep Learning for NLP and Speech Recognition*. Springer International Publishing. https://doi.org/10.1007/978-3-030-14596-5

Kechyn, G., Yu, L., Zang, Y., & Kechyn, S. (2018). Sales forecasting using WaveNet within the framework of the Kaggle competition. https://doi.org/10.48550/arXiv.1803.04037

Khamis, M. A., Ngo, H. Q., Nguyen, X., Olteanu, D., & Schleich, M. (2020). Learning models over relational data using sparse tensors and functional dependencies. *ACM Transactions on Database Systems*, *45*(2), 1–66. https://doi.org/10.1145/3375661

Khan, A., Sohail, A., Zahoora, U., & Qureshi, A. S. (2020). A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review*, *53*(8), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

Kiliçarslan, S., & Celik, M. (2021). RSigELU: A nonlinear activation function for deep neural networks. *Expert Systems with Applications*, *174*, 114805. https://doi.org/10.1016/j.eswa.2021.114805

Kilimci, Z. H., Akyuz, A. O., Uysal, M., Akyokus, S., Uysal, M. O., Bulbul, B. A., & Ekmis, M. A. (2019). An Improved Demand Forecasting Model Using Deep Learning Approach and Proposed Decision Integration Strategy for Supply chain. *Complexity*, *2019*, 1–15. https://doi.org/10.1155/2019/9067367

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *Proceedings of the 3rd International Conference of Learning Representations (ICLR)*.

Kingma, D. P., & Dhariwal, P. (2018). Glow: Generative Flow with Invertible 1×1 Convolutions. *Proceedings of the 32nd Neural Information Processing Systems conference (NeurIPS)*, *31*, 10236–10245.

Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., & Welling, M. (2016). Improved Variational Inference with Inverse Autoregressive Flow. *Proceedings of the 30th Neural Information Processing Systems conference (NIPS)*, 4743–4751.

Kingma, D. P., & Welling, M. (2019). An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning*, *12*(4), 307–392. https://doi.org/10.1561/2200000056

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-Normalizing Neural Networks. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)* (pp. 971–980). http://dblp.uni-trier.de/db/conf/nips/nips2017.html#KlambauerUMH17

Klein, D. A., & Shortliffe, E. H. (1991). Interactive diagnosis and repair of decision-theoretic models. *Proceedings of the 7th IEEE Conference on Artificial Intelligence Application*, *1*, 289–293. https://doi.org/10.1109/CAIA.1991.120883

Kobyzev, I., Prince, S. J., & Brubaker, M. A. (2021). Normalizing Flows: An Introduction and Review of Current Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *43*(11), 3964–3979. https://doi.org/10.1109/tpami.2020.2992934

Kohonen, T. (2000). *Self-Organizing Maps* (3rd). Springer Berlin Heidelberg. https://www.ebook.de/de/product/3246154/teuvo_kohonen_self_organizing_maps.html

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix Factorization Techniques for Recommender Systems. *Computer*, *42*(8), 30–37. https://doi.org/10.1109/mc.2009.263

Koyama, T., Horie, T., Yoshioka, T., Yoshitani, F., & Takahashi, J. (1998). A highly intelligible speech synthesis for banking services in financial network system ANSER. *Proceedings 4th IEEE Interactive Voice Technology for Telecommunications Applications conference IVTTA, workshop (Cat. No.98TH8376)*, 87–90. https://doi.org/10.1109/ivtta.1998.727699

Krizhevsky, A. (2009). *Learning multiple layers of features from tiny images* (tech. rep.). Computer Science, University of Toronto. https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks (F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger, Eds.). *Communications of the ACM*, *60*(6), 84–90. https://doi.org/10.1145/3065386

Krogh, A., & Hertz, J. A. (1991). A Simple Weight Decay can Improve Generalization. *Proceedings of the 4th Neural Information Processing Systems conference (NIPS)*, 950–957.

Kuleshov, V., Fenner, N., & Ermon, S. (2018). Accurate Uncertainties for Deep Learning Using Calibrated Regression. In J. Dy & A. Krause (Eds.),

Proceedings of the 35th International Conference on Machine Learning (ICML) (pp. 2796–2804). https://proceedings.mlr.press/v80/kuleshov18a.html

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience.

Lacey, K. (1828). *The Worthies of the United Kingdom; or Biographical Accounts of the Lives of the Most Illustrious Men, in Arts, Arms, Literature and Science, Connected with Great Britain. with Numerous Portraits, etc.* Knight&Lacey. https://books.google.ie/books?id=QjhkAAAAcAAJ

Lecun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324. https://doi.org/10.1109/5.726791

LeCun, Y., Denker, J., & Solla, S. (1989). Optimal Brain Damage. In D. Touretzky (Ed.), *Proceedings of the 3rd Neural Information Processing Systems conference (NIPS)*. Morgan-Kaufmann. https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Object Recognition with Gradient-Based Learning. In D. A. Forsyth, J. L. Mundy, V. D. Gesù, & R. Cipolla (Eds.), *Shape, Contour and Grouping in Computer Vision* (pp. 319–345). Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-46805-6_19

LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-R. (2012). Efficient BackProp. *Neural Networks: Tricks of the Trade: Second Edition* (pp. 9–48). Springer-Verlag GmbH. https://doi.org/10.1007/978-3-642-35289-8_3

Lee, H. S., & Wallraven, C. (2021). Visualizing the embedding space to explain the effect of knowledge distillation. *ArXiv e-print*, *abs/2110.04483*. https://doi.org/10.48550/ARXIV.2110.04483

Lee, S.-W., Kim, J.-H., Jun, J., Ha, J.-W., & Zhang, B.-T. (2017). Overcoming Catastrophic Forgetting by Incremental Moment Matching. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)*, 4655–4665.

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867. https://doi.org/10.1016/s0893-6080(05)80131-5

Li, J., Monroe, W., Shi, T., Ritter, A., & Jurafsky, D. (2017). Adversarial Learning for Neural Dialogue Generation. *Proceedings of the Empirical Methods in Natural Language Processing conference (EMNLP)*, 2157–2169.

Li, N., Liu, S., Liu, Y., Zhao, S., & Liu, M. (2019). Neural Speech Synthesis with Transformer Network. *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*. https://doi.org/10.1609/aaai.v33i01.33016706

Lighthill, J. (1973). Artificial Intelligence: A General Survey. *Artificial Intelligence: a paper symposium, Science Research Council*.

Lim, B., Arik, S. O., Loeff, N., & Pfister, T. (2019). Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *ArXiv e-print*, *abs/1912.09363*. https://doi.org/10.48550/ARXIV.1912.09363

Limmer, M., Forster, J., Baudach, D., Schüle, F., Schweiger, R., & Lensch, H. P. (2016). Robust Deep-Learning-Based Road-Prediction for Augmented Reality Navigation Systems at Night. *Proceedings of the 19th IEEE*

International Conference on Intelligent Transportation Systems (ITSC), 1888–1895. https://doi.org/10.1109/ITSC.2016.7795862

Lin, M., Chen, Q., & Yan, S. (2014). Network in network. *Proceedings of 2nd International Conference of Learning Representations (ICLR)*.

Lin, T.-Y., Dollár, P., Girshick, R. B., He, K., Hariharan, B., & Belongie, S. J. (2017). Feature pyramid networks for object detection. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 936–944.

Lipshutz, M., McEntire, R., & McKay, D. (1991). LIMA: A Logistics Inventory Management Assistant. *Proceedings of the 7th IEEE Conference on Artificial Intelligence Application*, *1*, 393–397. https://doi.org/10.1109/caia.1991.120899

Lipsman, A. (2019). Global Ecommerce 2019: Ecommerce Continues Strong Gains amid Global Economic Uncertainty. *E-marketer*.

Liu, C., Chen, L., Schroff, F., Adam, H., Hua, W., Yuille, A. L., & Fei-Fei, L. (2019). Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation. *Computing Research Repository CoRR*, *abs/1901.02985*, Article arXiv:1901.02985.

Liu, L., Wang, H., Lin, J., Socher, R., & Xiong, C. (2019). MKD: A Multi-Task Knowledge Distillation Approach for Pretrained Language Models. *ArXiv e-print*, *abs/1911.03588*. https://doi.org/10.48550/arXiv.1911.03588

Liu, R., Sisman, B., Bao, F., Gao, G., & Li, H. (2020). Modeling prosodic phrasing with multi-task learning in tacotron-based tts. *IEEE Signal Processing Letters*, *27*, 1470–1474. https://doi.org/10.1109/lsp.2020.3016564

Liu, R., Sisman, B., Li, J., Bao, F., Gao, G., & Li, H. (2019). Teacher-Student Training for Robust Tacotron-based TTS. *Proceedings of the 45th IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, *abs/1911.02839*. http://arxiv.org/abs/1911.02839

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965

Lorenzo-Trueba, J., Drugman, T., Latorre, J., Merritt, T., Putrycz, B., Barra-Chicote, R., Moinet, A., & Aggarwal, V. (2019). Towards achieving robust universal neural vocoding. *Proceedings of the Interspeech conference*.

Loshchilov, I., & Hutter, F. (2017). SGDR: Stochastic Gradient Descent with Warm Restarts. *Proceedings of the 5th International Conference of Learning Representations (ICLR), Toulon, France*. https://ml.informatik.uni-freiburg.de/~staeglis/deploy/papers/17-ICLR-SGDR.pdf

Lotfian, R., & Busso, C. (2019). Curriculum Learning for Speech Emotion Recognition from Crowdsourced Labels. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *27*(4), 815–826. https://doi.org/10.1109/taslp.2019.2898816

Lu, L. (2020). Dying ReLU and Initialization: Theory and Numerical Examples. *Communications in Computational Physics*, *28*(5), 1671–1706. https://doi.org/10.4208/cicp.oa-2020-0165

Luong, M. T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *Proceedings of the Empirical*

Methods in Natural Language Processing conference (EMNLP), 1412–1421. https://doi.org/10.18653/v1/d15-1166

Malik, A., Kuleshov, V., Song, J., Nemer, D., Seymour, H., & Ermon, S. (2019). Calibrated Model-Based Deep Reinforcement Learning. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML), long beach, california, USA* (pp. 4314–4323). http://proceedings.mlr.press/v97/malik19a.html

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press. https://www.ebook.de/de/product/7455223/christopher_d_manning_prabhakar_raghavan_hinrich_schuetze_introduction_to_information_retrieval.html

Marković, M., Jakovljević, B., Milićev, T., & Miliević, N. (2015). The Role of Prosody in the Perception of Synthesized and Natural Speech. *Speech and Computer* (pp. 446–453). Springer International Publishing. https://doi.org/10.1007/978-3-319-23132-7_55

Mashaly, H., Sharaf, A., Mansour, M., & El-Sattar, A. (1994). Implementation of an artificial neural network based controller for a photovoltaic energy scheme. *Proceedings of IEEE International Conference on Neural Networks (ICNN)*, *4*, 2545–2549 vol.4. https://doi.org/10.1109/icnn.1994.374621

McCulloch, W., & Pitts, W. (1943). A Logical Calculus of Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics*, *5*, 127–147.

McMahan, B., & Rao, D. (2017). Listening to the World Improves Speech Command Recognition. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*.

Michaely, A. H., Zhang, X., Simko, G., Parada, C., & Aleksic, P. (2017). Keyword spotting for google assistant using contextual speech recognition. *Proceedings of the IEEE Automatic Speech Recognition and Understanding (ASRU), Workshop*, 272–278.

Minsky, M., & Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Misra, D. (2020). Mish: A self regularized non-monotonic neural activation function. *Proceedings of the British Machine Vision Conference (BMVC)*.

Misra, J., & Saha, I. (2010). Artificial neural networks in hardware: A survey of two decades of progress [Artificial Brains]. *Neurocomputing*, *74*(1-3), 239–255. https://doi.org/10.1016/j.neucom.2010.03.021

Murphy, K. P. (2012). *Machine Learning*. MIT Press Ltd. https://www.ebook.de/de/product/19071158/kevin_p_murphy_machine_learning.html

Murthy, S. T. (2022). *Textbook of Elements of Mechanical Engineering*. Medtech.

Nair, V., & Hinton, G. E. (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. In J. Fürnkranz & T. Joachims (Eds.), *Proceedings of the 27th International Conference on Machine Learning (ICML)* (pp. 807–814). https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, *36*(1), 48–49. https://doi.org/10.1073/pnas.36.1.48

Nassif, A. B., Shahin, I., Attili, I., Azzeh, M., & Shaalan, K. (2019). Speech Recognition Using Deep Neural Networks: A Systematic Review. *IEEE Access*, *7*, 19143–19165. https://doi.org/10.1109/access.2019.2896880

Nazaré, T. S., de Mello, R. F., & Ponti, M. A. (2018). Are pre-trained CNNs good feature extractors for anomaly detection in surveillance videos? *ArXiv e-print*, *abs/1811.08495*, Article arXiv:1811.08495.

Nguyen, T., Raghu, M., & Kornblith, S. (2021). Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth. *Proceedings of the 9th International Conference on Learning Representations (ICLR), Virtual Event, Austria*. https://openreview.net/forum?id=KJNcAkY8tY4

Nilsson, N. J. (2009). *The Quest for Artificial Intelligence* (1st). Cambridge University Press.

Ogawa, T., Minohara, T., Kanada, H., & Kosugi, Y. (1999). A neural network model for realizing geometric illusions based on acute-angled expansion. *Proceedings of the 13th Neural Information Processing Systems conference (NIPS)*, *2*, 550–555. https://doi.org/10.1109/ICONIP.1999.845653

Ooko, S. O., Muyonga Ogore, M., Nsenga, J., & Zennaro, M. (2021). TinyML in Africa: Opportunities and Challenges. *Proceedings of the IEEE Globecom (GC) Workshops*, 1–6. https://doi.org/10.1109/GCWkshps52748.2021.9682107

Papamakarios, G., Pavlakou, T., & Murray, I. (2017). Masked Autoregressive Flow for Density Estimation. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)*, 2335–2344.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML)* (pp. 1310–1318). http://proceedings.mlr.press/v28/pascanu13.html

Perlovsky, L. I. (1999). Emotions, learning and control. *Proceedings of the IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics (Cat. No.99CH37014)*, 132–137. https://doi.org/10.1109/ISIC.1999.796643

Pham, H., Guan, M., Zoph, B., Le, Q., & Dean, J. (2018). Efficient Neural Architecture Search via Parameters Sharing. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML)* (pp. 4095–4104). PMLR.

Pinkus, A. (1999). Approximation theory of the MLP model in neural networks. *Acta Numerica*, *8*, 143–195. https://doi.org/10.1017/s0962492900002919

Proakis, J. (2007). *Digital signal processing: Principles, algorithms, and applications* (4th ed.). Pearson Prentice Hall Pearson Education, Ltd.

Punch, W. (1992). Large interactions of compiled and causal reasoning in diagnosis. *IEEE Expert*, *7*(1), 28–35. https://doi.org/10.1109/64.120685

Qiang, B., Zhai, Y., Zhou, M., Yang, X., Peng, B., Wang, Y., & Pang, Y. (2021). SqueezeNet and Fusion Network-Based Accurate Fast Fully Convolutional Network for Hand Detection and Gesture Recognition. *IEEE Access*, *9*, 77661–77674. https://doi.org/10.1109/ACCESS.2021.3079337

Raghu, M., & Schmidt, E. (2020). A Survey of Deep Learning for Scientific Discovery. *ArXiv e-print*, *abs/2003.11755*. https://arxiv.org/abs/2003.11755

Raitio, T., Rasipuram, R., & Castellani, D. (2020). Controllable Neural Text-to-Speech Synthesis Using Intuitive Prosodic Features. *Proceedings of the Interspeech conference*. https://doi.org/10.21437/interspeech.2020-2861

Ramachandran, P., Zoph, B., & Le, Q. (2018). Searching for Activation Functions. *Proceedings of the 6th International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=SkBYYyZRZ

Ren, Z., Chen, Z., & Xu, S. (2019). Triplet Based Embedding Distance and Similarity Learning for Text-independent Speaker Verification. *Proceedings of the A Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, 558–562. https://doi.org/10.1109/APSIPAASC47483.2019.9023253

Rezende, D. J., & Mohamed, S. (2015). Variational Inference with Normalizing Flows. *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 1530–1538.

Rosenblatt, F. (1958). The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, 65–386.

Rosheim, M. E. (2006). *Leonardo's Lost Robots*. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-28497-4

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *ArXiv e-print, abs/1609.04747*. https://doi.org/10.48550/ARXIV.1609.04747

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536. http://dx.doi.org/10.1038/323533a0

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. https://doi.org/10.1007/s11263-015-0816-y

Russell, S. (2003). *Artificial intelligence: A modern approach* (2nd ed.). Prentice Hall/Pearson Education.

Sabour, S., Frosst, N., & Hinton, G. E. (2017). Dynamic Routing between Capsules. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)*, 3859–3869.

Sainath, T. N., & Parada, C. (2015). Convolutional neural networks for small-footprint keyword spotting. *Proceedings of the Interspeech conference*, 1478–1482.

Sajjadi, M. S. M., Bachem, O., Lucic, M., Bousquet, O., & Gelly, S. (2018). Assessing Generative Models via Precision and Recall. *Proceedings of the 32nd Neural Information Processing Systems conference (NeurIPS)*, 5234–5243.

Sanchez-Iborra, R., & Skarmeta, A. F. (2020). TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities. *IEEE Circuits and Systems Magazine*, *20*(3), 4–18. https://doi.org/10.1109/MCAS.2020.3005467

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 4510–4520. https://doi.org/10.1109/CVPR.2018.00474

Schleich, M., Olteanu, D., Abo Khamis, M., Ngo, H. Q., & Nguyen, X. (2019). A Layered Aggregate Engine for Analytics Workloads. *Proceedings of*

the International Conference on Management of Data, 1642–1659. https://doi.org/10.1145/3299869.3324961

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093

Serban, I., Sankar, C., Germain, M., Zhang, S., Lin, Z., Subramanian, S., Kim, T., Pieper, M., Chandar, S., Rosemary Ke, N., Mudumba, S., de Brebisson, A., M. R. Sotelo, J., Suhubdy, D., Michalski, V., Nguyen, A., Pineau, J., & Bengio, Y. (2017). A Deep Reinforcement Learning Chatbot. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)*.

Shaikhha, A., Schleich, M., Ghita, A., & Olteanu, D. (2020). Multi-Layer Optimizations for End-to-End Data Analytics. *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*, 145–157. https://doi.org/10.1145/3368826.3377923

Shelley, M. (1994). *Frankenstein*.

Shen, J., Pang, R., Weiss, R. J., Schuster, M., Jaitly, N., Yang, Z., Chen, Z., Zhang, Y., Wang, Y., Skerry-Ryan, R., Saurous, R. A., Agiomyrgiannakis, Y., & Wu, Y. (2018). Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. https://arxiv.org/abs/1712.05884

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*, 484–503. http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., & Hassabis, D. (2017). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *ArXiv e-print*, *abs/1712.01815*. https://doi.org/10.48550/arXiv.1712.01815

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/nature24270

Simonyan, K., & Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1409.1556

Sisman, B., Yamagishi, J., King, S., & Li, H. (2021). An Overview of Voice Conversion and its Challenges: From Statistical Modeling to Deep Learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *29*, 132–157. https://doi.org/10.1109/TASLP.2020.3038524

Sivanandam, S., & Deepa, S. (2008). *Introduction to Genetic Algorithms*. Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-73190-0

Skerry-Ryan, R., Battenberg, E., Xiao, Y., Wang, Y., Stanton, D., Shor, J., Weiss, R., Clark, R., & Saurous, R. A. (2018). Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML)* (pp. 4693–4702). PMLR. http://proceedings.mlr.press/v80/skerry-ryan18a.html

Smith, P. D. (2018). *Hands-On Artificial Intelligence for Beginners: An Introduction to AI Concepts, Algorithms, and Their Implementation*. Packt Publishing. https://www.ebook.de/de/product/34797426/patrick_d_smith_hands_on_artificial_intelligence_for_beginners.html

Smithers, T., Tang, M. X., & Tomes, N. (1993). An approach to intelligent drug design support. *Proceedings of the 36th Hawaii International Conference on System Sciences*, *1*, 634–645 vol.1. https://doi.org/10.1109/HICSS.1993.270677

Smolensky, P. (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* (D. E. Rumelhart, J. L. McClelland, & C. PDP Research Group, Eds.). MIT Press. https://www.ebook.de/de/product/9337112/david_e_rumelhart_james_l_mcclelland_pdp_research_group_parallel_distributed_processing_volume_1.html

So, D. R., Mańke, W., Liu, H., Dai, Z., Shazeer, N., & Le, Q. V. (2021). Primer: Searching for Efficient Transformers for Language Modeling. *Proceedings of the 35th Neural Information Processing Systems conference (NeurIPS)*.

Song, L., Liu, J., Qian, B., Sun, M., Yang, K., Sun, M., & Abbas, S. (2018). A Deep Multi-Modal CNN for Multi-Instance Multi-Label Image Classification. *IEEE Transactions on Image Processing*, *27*(12), 6025–6038. https://doi.org/10.1109/TIP.2018.2864920

Souza, F., & Filho, J. (2022). BERT for Sentiment Analysis: Pre-trained and Fine-Tuned Alternatives. *Proceedings of the International Conference on the Computational Processing of Portuguese (PROPOR)*. https://doi.org/10.48550/arXiv.2201.03382

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of machine learning research*, *15*(1), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 3645–3650. https://doi.org/10.18653/v1/P19-1355

Sutskever, I., Martens, J., & E. Hinton, G. (2011). Generating text with recurrent neural networks. *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 1017–1024.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *Proceedings of the 27th Neural Information Processing Systems conference (NIPS)*, *2*, 3104–3112.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning*. A Bradford Book. https://www.ebook.de/de/product/32966850/richard_s_university_of_alberta_sutton_andrew_g_co_director_autonomous_learning_laboratory_barto_reinforcement_learning.html

Swaminathan, V., Arora, S., Bansal, R., & Rajalakshmi, R. (2019). Autonomous Driving System with Road Sign Recognition using Convolutional Neural

Networks. *Proceedings of the International Conference on Computational Intelligence in Data Science (ICCIDS)*, 1–4. https://doi.org/10.1109/ICCIDS.2019.8862152

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the Inception Architecture for Computer Vision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2818–2826. https://doi.org/10.1109/CVPR.2016.308

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 4278–4284.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–9. https://doi.org/10.1109/CVPR.2015.7298594

Szepesvári, C., & Bartok, G. (2010). Algorithms for Reinforcement Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *4*, 1–103. https://doi.org/10.2200/S00268ED1V01Y201005AIM009

Talupula, A. (2018). *Demand Forecasting of Outbound Logistics Using Machine learning* (Master Thesis). Faculty of Computing, Blekinge Institute of Technology, 371 79 Karlskrona, Sweden. https://www.diva-portal.org/smash/get/diva2:1367098/FULLTEXT02

Tan, C., Liu, J., & Zhang, X. (2021). Improving knowledge distillation via an expressive teacher. *Knowledge-Based Systems*, *218*, 106837. https://doi.org/10.1016/j.knosys.2021.106837

Tan, H. H., & Lim, K. H. (2019). Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization. *Proceedings of the 7th International Conference on Smart Computing Communications (ICSCC)*, 1–4. https://doi.org/10.1109/ICSCC.2019.8843652

Tan, M., & Le, Q. (2019). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th International Conference on Machine Learning (ICML)* (pp. 6105–6114).

Taylor, P. (2009). *Text-to-Speech Synthesis* (1st). Cambridge University Press. https://www.ebook.de/de/product/7676986/paul_taylor_text_to_speech_synthesis.html

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM*, *38*(3), 58–68. https://doi.org/10.1145/203330.203343

Tesauro, G. (1994). TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play. *Neural Computation*, *6*, 215–219. https://doi.org/10.1162/neco.1994.6.2.215

Theis, L., van den Oord, A., & Bethge, M. (2016). A note on the evaluation of generative models. *Proceedings of the International Conference on Learning Representations*. http://arxiv.org/abs/1511.01844

Theodoridis, S. (2015). *Machine learning : A Bayesian and optimization perspective* (1st). Academic Press.

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *London Mathematical Society*, *2*(42), 230–265. http://www.cs.helsinki.fi/u/gionis/cc05/OnComputableNumbers.pdf

Turing's famous demonstration of the formal limits on computation based on a proof that the *halting problem* is undecidable.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, *59*, 433–460.

Ulyanov, D., Vedaldi, A., & Lempitsky, V. S. (2016). Instance Normalization: The Missing Ingredient for Fast Stylization. *Computing Research Repository CoRR*, *abs/1607.08022*, Article arXiv:1607.08022.

Vallés-Pérez, I. (2012). *Aplicación de Algoritmos Genéticos en Extreme Learning Machine* (Bachelor Degree capstone project report). Intelligent data analysis laboratory, University of Valencia. https://www.uv.es/ivape3/memoria_pfc.pdf

Vallés-Pérez, I., Gómez-Sanchís, J., Martínez-Sober, M., Vila-Francés, J., Serrano-López, A. J., & Soria-Olivas, E. (2021a). End-to-end Keyword Spotting using Xception-1d. *Proceedings of the European Symposium of Neural Networks (ESANN)*. https://www.esann.org/sites/default/files/proceedings/2021/ES2021-21.pdf

Vallés-Pérez, I., Roth, J., Beringer, G., Barra-Chicote, R., & Droppo, J. (2021b). Improving Multi-Speaker TTS Prosody Variance with a Residual Encoder and Normalizing Flows. *Proceedings of the Interspeech conference*, 3131–3135. https://doi.org/10.21437/Interspeech.2021-562

van den Oord, A., Kalchbrenner, N., Espeholt, L., kavukcuoglu koray, k., Vinyals, O., & Graves, A. (2016). Conditional Image Generation with PixelCNN Decoders. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Proceedings of the 30th Neural Information Processing Systems (NIPS)*. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2016/file/b1301141feffabac455e1f90a7de2054-Paper.pdf

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., & Kavukcuoglu, K. (2016). WaveNet: A Generative Model for Raw Audio. *Proceedings of the 9th International Speech Communication Association (ISCA), Speech Synthesis Workshop, Sunnyvale, CA, USA*, 125. http://www.isca-speech.org/archive/SSW%5C_2016/abstracts/ssw9%5C_DS-4%5C_van%5C_den%5C_Oord.html

van den Oord, A., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, *abs/1601.06759*. http://arxiv.org/abs/1601.06759

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you need. *Proceedings of the 31st Neural Information Processing Systems conference (NIPS)*, 6000–6010.

Wang, D., Lin, J., Cui, P., Jia, Q., Wang, Z., Fang, Y., Yu, Q., Zhou, J., Yang, S., & Qi, Y. (2020). A Semi-supervised Graph Attentive Network for Financial Fraud Detection. *Proceedings of the IEEE International Conference on Data Mining (ICDM)*. https://doi.org/10.1109/ICDM.2019.00070

Wang, D., Lv, S., Wang, X., & Lin, X. (2018). Gated Convolutional LSTM for Speech Commands Recognition. *Proceedings of the International Conference of Computer Science*, 669–681.

Wang, J., Cevik, M., & Bodur, M. (2020). On the Impact of Deep Learning-based Time-series Forecasts on Multistage Stochastic Programming

Policies. *ArXiv e-print*, *abs/2009.00665*, Article arXiv:2009.00665, arXiv:2009.00665.

Wang, L., Shen, B., Zhao, N., & Zhang, Z. (2020). Is the Skip Connection Provable to Reform the Neural Network Loss Landscape? *Proceedings of the 29th International Joint Conference of Artificial Intelligence (IJCAI)*, *https://www.ijcai.org/proceedings/2020/0387.pdf*.

Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., & Saurous, R. A. (2017). Tacotron: Towards End-to-End Speech Synthesis. *Proceedings of the Interspeech conference*. https://arxiv.org/abs/1703.10135

Wang, Z., She, Q., & Ward, T. E. (2022). Generative Adversarial Networks in Computer Vision. *ACM Computing Surveys*, *54*(2), 1–38. https://doi.org/10.1145/3439723

Warden, P. (2017). Speech Commands: A public dataset for single-word speech recognition. *Datasets available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz and http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz*.

Warden, P., & Situnayake, D. (2020). *TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-low-power Microcontrollers*. O'Reilly. https://books.google.co.uk/books?id=sB3mxQEACAAJ

Warden, P. (2018). Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-print*, *abs/1804.03209*, Article arXiv:1804.03209. https://doi.org/10.48550/arXiv.1804.03209

Wei, R., & Mahmood, A. (2021). Recent advances in variational autoencoders with representation learning for biomedical informatics: A survey. *IEEE Access*, *9*, 4939–4956.

Widrow, B. (1960). An adaptive 'Adaline' neuron using chemical 'memistors' (S. E. L. Solid-State Electronics Laboratory, Ed.). *Technical report at Stanford university, Solid-State Electronics Laboratory, under an Office of Naval Research contract*.

Williams, R. J., & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation*, *1*, 270–280.

Wu, C.-J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Behram, F. A., Huang, J., Bai, C., Gschwind, M., Gupta, A., Ott, M., Melnikov, A., Candido, S., Brooks, D., Chauhan, G., Lee, B., Lee, H.-H. S., ... Hazelwood, K. (2021). Sustainable AI: Environmental Implications, Challenges and Opportunities. *ArXiv e-print*, *abs/2111.00364*. https://doi.org/10.48550/arXiv.2111.00364

Wu, X., Manton, J. H., Aickelin, U., & Zhu, J. (2021). *A bayesian Approach to (online) Transfer Learning: Theory and Algorithms* (tech. rep.). University of Melbourne, Department of Electronic, Electrical Engineering, Department of Computing, and Information Systems.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical Evaluation of Rectified Activations in Convolutional Network. *Proceedings of the Computer Vision and Pattern Recognition conference (CVPR)*. https://arxiv.org/pdf/1505.00853.pdf

Xu, Z., Yang, X., Li, X., & Sun, X. (2018). The Effectiveness of Instance Normalization: A Strong Baseline for Single Image Dehazing. *ArXiv*

e-print, *abs/1805.03305*, Article arXiv:1805.03305. https://doi.org/10.48550/arXiv.1805.03305

Yang, L., & Chakraborty, R. (2020). A GMM Based Algorithm to Generate Point-Cloud and its Application to Neuroimaging. *Proceedings of the 17th IEEE International Symposium on Biomedical Imaging (ISBI) Workshops*, 1–4. https://doi.org/10.1109/ISBIWorkshops50223.2020.9153437

Yoo, J.-H., Kang, B.-H., & Choi, J.-U. (1994). A hybrid approach to auto-insurance claim processing system. *1*, 537–542 vol.1. https://doi.org/10.1109/ICSMC.1994.399894

Yu, F., & Koltun, V. (2016). Multi-Scale Context Aggregation by Dilated Convolutions. *Proceedings of the 5th International Conference on Learning Representations (ICLR)*.

Yuzhong, W., Jie, Y., & Yue, Z. (2006). Color-texture segmentation using JSEG based on Gaussian mixture modeling. *Journal of Systems Engineering and Electronics*, *17*(1), 24–29. https://doi.org/10.1016/S1004-4132(06)60005-4

Zapico, J. L., Ahlgren, F., & Zennaro, M. (2021). Insect biodiversity in agriculture using IoT: Opportunities and needs for further research. *Proceedings of the IEEE Globecom (GC) Workshops*, 1–5. https://doi.org/10.1109/GCWkshps52748.2021.9682153

Zhang, S., Zhang, S., Huang, T., & Gao, W. (2018). Speech Emotion Recognition Using Deep Convolutional Neural Network and Discriminant Temporal Pyramid Matching. *IEEE Transactions on Multimedia*, *20*(6), 1576–1590. https://doi.org/10.1109/TMM.2017.2766843

Zhang, S. (2002). *Association Rule Mining*. Springer Berlin Heidelberg. https://www.ebook.de/de/product/6701874/shichao_zhang_association_rule_mining.html

Zhang, Y., Suda, N., Lai, L., & Chandra, V. (2017). Hello Edge: Keyword Spotting on Microcontrollers. *ArXiv e-print*, *abs/1711.07128*, Article arXiv:1711.07128. https://doi.org/10.48550/arXiv.1711.07128

Zhao, H., Sun, X., Dong, J., Dong, Z., & Li, Q. (2021). Knowledge distillation via instance-level sequence learning. *Knowledge-Based Systems*, *233*, 107519. https://doi.org/10.1016/j.knosys.2021.107519

Zhao, P., Wu, T., Zhao, S., & Liu, H. (2021). Robust transfer learning based on Geometric Mean Metric Learning. *Knowledge-Based Systems*, *227*, 107227. https://doi.org/10.1016/j.knosys.2021.107227

Zhou, Y., Chen, S., Wang, Y., & Huan, W. (2020). Review of research on lightweight convolutional neural networks. *2020 IEEE 5th Information Technology and Mechatronics Engineering Conference (ITOEC)*, 1713–1720. https://doi.org/10.1109/ITOEC49072.2020.9141847

Zhu, M., Min, W., Wang, Q., Zou, S., & Chen, X. (2021). PFLU and FPFLU: Two novel non-monotonic activation functions in convolutional neural networks. *Neurocomputing*, *429*, 110–117. https://doi.org/10.1016/j.neucom.2020.11.068

Zhu, Y., Hu, X., Zhang, Y., & Li, P. (2018). Transfer learning with stacked reconstruction independent component analysis. *Knowledge-Based Systems*, *152*, 100–106. https://doi.org/10.1016/j.knosys.2018.04.010

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., & He, Q. (2021). A Comprehensive Survey on Transfer Learning. *Proceedings of the IEEE*, *109*(1), 43–76. https://doi.org/10.1109/JPROC.2020.3004555