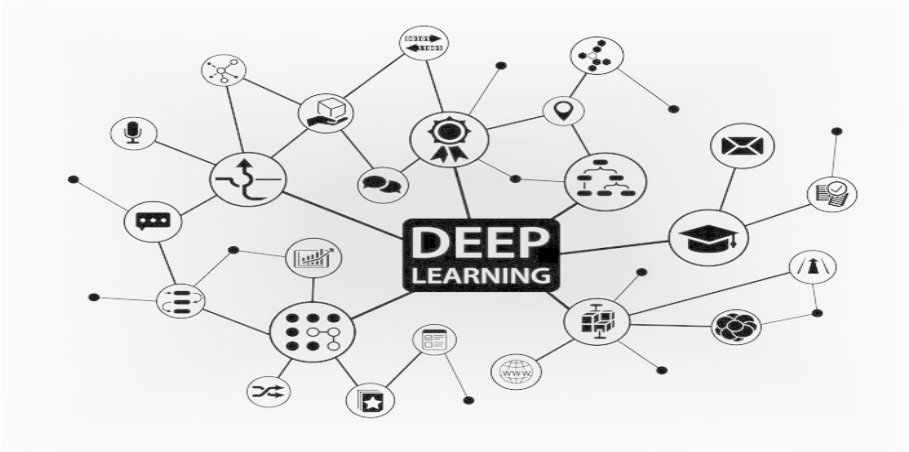
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Transferência de Aprendizagem em Arquiteturas de Aprendizagem Profunda

DIOGO MIGUEL DO NASCIMENTO HORTA

(Licenciado)

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Prof. Doutor Luís Filipe Graça Morgado

Júri:

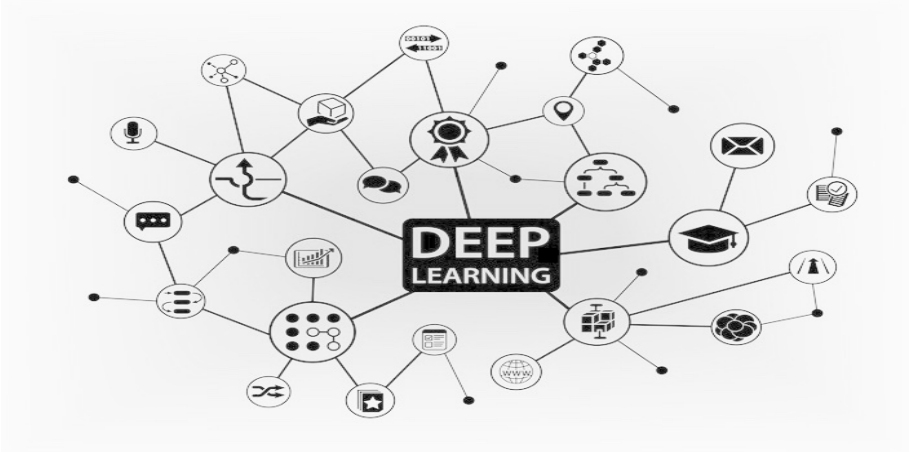
Presidente: Prof. Doutor Nuno Miguel Soares Datia

Vogal: Prof. Doutor Gonçalo Caetano Marques

Junho, 2021

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores



Transferência de Aprendizagem em Arquiteturas de Aprendizagem Profunda

DIOGO MIGUEL DO NASCIMENTO HORTA

(Licenciado)

Dissertação para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientador : Prof. Doutor Luís Filipe Graça Morgado

Júri:

Presidente: Prof. Doutor Nuno Miguel Soares Datia

Vogal: Prof. Doutor Gonçalo Caetano Marques

Junho, 2021

Para a minha mãe e melhor amiga

Agradecimentos

Ao Professor Doutor Luís Morgado, por ter assumido a orientação deste trabalho, pela colaboração e apoio que me proporcionou e pela disponibilidade que sempre mostrou para ajudar na compreensão e análise dos resultados. O seu apoio e orientação foram fundamentais para a realização deste trabalho.

Ao ISEL - Instituto Superior de Engenharia de Lisboa, que ocupará sempre um lugar de destaque na minha formação pelo excelente envolvimento profissional e humano de todos os docentes e colegas com quem tive o privilégio de estudar. Um destaque especial ao Professor Doutor Mário Véstias pela disponibilização do *hardware* que facilitou a realização deste trabalho.

À minha familia e amigos que sempre me apoiaram e motivaram nesta etapa particularmente trabalhosa da minha vida. Uma menção especial aos meus amigos, Daniel Rodrigues e Luís Pinto, cuja amizade permitiu que momentos difíceis se tornassem mais fáceis.

Finalmente, à minha mãe Aida e aos meus avós, Silvino e Celeste, que sempre fizeram tudo o possível para que eu prosseguisse sempre mais além na minha formação académica e pessoal. Esta é a concretização de mais um sonho que eles tinham para mim e que a eles lhe dedico.

Resumo

A aprendizagem profunda (*deep learning*) é uma forma de aprendizagem baseada em redes neuronais organizadas de forma hierárquica em diferentes níveis de competência.

Neste tipo de redes, cada nível de competência tem a capacidade de aprender através da abstração de características de forma modular.

Deste modo, torna-se possível a transferência do conhecimento adquirido em níveis de abstração mais gerais, cujas características não sejam específicas de um domínio particular, de modo a utilizar esse conhecimento em domínios relacionados, nomeadamente, no sentido de acelerar o processo de aprendizagem em domínios distintos do domínio inicial.

O objetivo deste trabalho é o estudo de métodos de transferência de aprendizagem com base em arquiteturas de aprendizagem profunda, nomeadamente em redes convolucionais, no sentido de avaliar o âmbito de aplicação destes métodos, bem como a definição das condições em que essa aplicação é viável.

Palavras-chave: Aprendizagem Profunda, Redes Neuronais, Transferência de Aprendizagem, Inteligência Artificial

Abstract

Deep learning is a learning technique based on hierarchical neural networks with different competence levels.

In this type of networks each hierarchical level has the ability to learn through the abstraction of characteristics.

The transfer of general knowledge adquired previously, which is not specific to certain domain, allows the training time of other neural networks to be reduced allowing those other networks to become operational faster.

The purpose of this dissertation is the study of methods for transfer learning based on deep learning architectures, namely on convolutional networks, with the goal of evaluating its applications as well as the conditions in which its use is desired.

Keywords: Deep Learning, Neural Network, Transfer Learning, Artificial Inteligence

Índice

Lista de Figuras

Lista de Tabelas

1

Introdução

A aprendizagem profunda, também conhecida como *Deep Learning*, no original em inglês, é um dos vários ramos de Aprendizagem Automática, que através de um conjunto de algoritmos procura modelar abstrações de alto nível através de transformações lineares e não-lineares.

A aprendizagem profunda é atualmente utilizada nas mais variadas áreas, desde a visão computacional para reconhecimento de faces, veículos, estradas, comportamentos suspeitos (para situações em que existe uma fonte de vídeo como as caixas *self-checkout* presentes em supermercados), reconhecimento automático de fala e processamento de linguagem natural em assistentes digitais, digitalização de documentos e na área da cibersegurança, onde é utilizada para reconhecer padrões de ataques DoS a sistemas informáticos vitais (centrais elétricas, instituições financeiras entre outras).

Uma rede neuronal profunda pode ser composta por uma hierarquia de várias sub-redes, permitindo assim criar redes com capacidades complexas a partir de redes mais simples e ao mesmo tempo otimizar cada uma destas redes para a(s) sua(s) característica(s) em específico. Apesar do excelente desempenho obtido por algumas implementações de redes neuronais, existem alguns desafios na sua conceção, como o elevado custo computacional durante a fase de treino, devido à necessidade de atualizar cada peso de cada neurónio de cada camada para cada iteração, o cálculo do número de camadas e do número de neurónios por camada, qual o valor inicial dos pesos de cada neurónio e evitar a sobre-especialização

(*overfitting*) da rede, na qual a rede aprende não as características gerais mas sim as características específicas associadas ao conjunto de treino.

Por sua vez, a utilização de redes neuronais profundas requer enorme poder computacional e acesso a dados de treino de grandes dimensões, o que torna a utilização deste tipo de redes difícil a nível comercial, exceto a empresas com capacidades financeiras elevadas ou instituições académicas, e diminui o número de vezes que estas empresas/instituições podem treinar de novo as suas próprias redes, contudo através de composição/hierarquização é possível adquirir conhecimentos mais complexos a partir de redes menos complexas.

A transferência de aprendizagem consiste em utilizar o resultado da aprendizagem, adquirido no âmbito de um determinado problema, em outro problema distinto mas relacionado. No caso de redes neuronais profundas, consiste em transferir a capacidade de reconhecer/identificar determinadas características, no contexto de uma rede neuronal, para outra rede, de modo a permitir reduzir o tempo necessário à aprendizagem de características mais complexas.

1.1 Motivação

O aumento das funcionalidades inteligentes disponíveis nos dispositivos atuais, graças à aprendizagem profunda, torna a possibilidade de investigar novos métodos para melhorar o seu desempenho bastante interessante.

A transferência de aprendizagem constitui um desses métodos, pelo que o seu estudo é de grande relevância no contexto do desenvolvimento de arquitecturas de redes neuronais profundas.

1.2 Objectivos

Esta dissertação propõe os seguintes objetivos:

- Compreender e definir o que é a transferência de aprendizagem no contexto de redes neuronais profundas;
- Definir quais os métodos e abordagens necessários para tornar a transferência possível;
- Criar protótipos e avaliar o desempenho da transferência de aprendizagem em redes neuronais profundas convolucionais.

1.3 Organização do Documento

Para além deste capítulo de introdução, esta dissertação encontra-se organizada em mais seis capítulos, num total de sete:

Capítulo 2: Redes Neuronais
Capítulo de enquadramento, onde são apresentados os conceitos de neurónio biológico, perceptrão, redes multi-camada, a retropropagação e o processo de treino.

Capítulo 3: Redes Neuronais Profundas
Capítulo de enquadramento, onde é apresentada a estrutura de uma rede neuronal profunda, os diferentes tipos de camadas, a rede neuronal profunda convolucional e como é realizada a aprendizagem em redes neuronais profundas.

Capítulo 4: Transferência de Aprendizagem
Neste capítulo é apresentado o conceito de transferência de aprendizagem, quais os diferentes tipo de transferência e as abordagens para realizar a transferência.

Capítulo 5: Arquiteturas de Redes Desenvolvidas
Neste capítulo são apresentadas as arquiteturas desenvolvidas, as suas formas de organização e as parametrizações definidas.

Capítulo 6: Concretização Experimental
Neste capítulo são apresentados os conjuntos de dados a utilizar, o *hardware* utilizado para treinar as redes, o *software* desenvolvido para realizar a transferência de aprendizagem e os resultados obtidos e a sua interpretação.

Capítulo 7: Conclusão
Neste capítulo é realizada uma síntese das conclusões obtidas a partir dos resultados e, são apresentadas questões a explorar em trabalhos futuros.

1.4 Convenções de Escrita

No sentido de facilitar a leitura, ao longo da dissertação são utilizadas as seguintes convenções de escrita:

- As traduções de termos ou expressões originalmente em língua inglesa, na sua primeira ocorrência no texto, surgem seguidas da designação original, em itálico;

- Optou-se por não traduzir alguns termos de língua inglesa de utilização generalizada, como "hardware" ou "software";
- Em relação às siglas utilizadas, dado o seu uso generalizado na comunidade técnico-científica, são mantidas tal como aparecem no original;
- O grafismo em itálico é utilizado para destacar palavras individuais ou conjuntos de palavras no texto, como é o caso das designações de conceitos, que surgem com grafismo em itálico na primeira ocorrência que seja relevante para a sua descrição;
- Surgem igualmente com grafismo em itálico as expressões latinas como *a priori*.

2

Redes Neuronais

O aumento da utilização de redes neuronais profundas poderia indicar que a tecnologia de redes neuronais artificiais é uma nova tecnologia, contudo as redes neuronais profundas são uma tecnologia com aproximadamente 80 anos.
As redes neuronais artificiais surgem na década de 40 do século XX com a **cibernética**, área científica que tem por objectivo o estudos dos fenómenos de controlo e comunicação, quer em contextos naturais, biológicos, quer em sistemas artificiais [1].

Os primeiros modelos propostos são modelo lineares simples, que associam um determinado valor de saída a um conjunto de valores de entrada através de uma função linear. Estes modelos são designados como neurónios artificiais (*artificial neurons*) devido à inspiração no modelo do neurónio biológico.

O primeiro modelo, o Neurónio *McCulloch-Pitts* é proposto por *McCulloch* e *Pitts* em 1943 [2], com a capacidade de distinguir entre duas categorias distintas de valores de entrada, desde que a configuração dos pesos seja adequada para as categorias pretendidas. A configuração dos pesos é realizada manualmente e o modelo não possui a capacidade de aprender a configuração dos pesos correta a partir dos valores de entrada.

O psicólogo *Frank Rosenblatt* [3, 4] propõe em 1958 um novo modelo linear designado por Perceptrão, o primeiro neurónio artificial com a capacidade de aprender

a configuração dos pesos a partir dos valores de entrada associados a cada categoria, *Rosenblatt* utilizou o teorema de convergência do Perceptrão para demonstrar a respetiva capacidade de aprendizagem. Este modelo foi desenvolvido para Marinha Americana e implementado fisicamente no *Mark 1 Perceptron* [5].

Em 1960 *Widrow* e *Hoff* propõem o modelo adaptativo linear ou *ADALINE* [6], um modelo que também possui a capacidade de aprender a configuração dos pesos e utiliza como algoritmo de aprendizagem um caso específico do algoritmo **descida de gradiente estocástica**. Variantes modernas deste algoritmo de **descida de gradiente estocástica** constituem os algoritmos de treino mais utilizados atualmente em redes neuronais [7]. Estes primeiros modelos suscitaram um enorme entusiasmo com artigos de jornal a indicarem que sistemas baseados no Perceptrão teriam a capacidade de andar, ver, escrever e conversar entre outras capacidades [8], contudo, estes modelos apresentam limitações como a incapacidade de aprender funções não-linearmente separáveis como a função lógica *XOR*.

Em 1969 *Minsky* e *Papert* lançam o livro *Perceptrons: an introduction to computational geometry* [9] no qual se focam, maioritariamente, nas limitações associadas ao Perceptrão como a função lógica *XOR*, entre outras.
Estas limitações levaram a uma diminuição do entusiasmo no estudo das redes neuronais artificiais e consequentemente no desenvolvimento de novos modelos.

Em 1975 *Fukushima* apresenta o Cognitrão [10] seguido pelo Neocognitrão [11] em 1980, um modelo inspirado na estrutura do sistema visual de mamíferos e que viria a servir de base no desenvolvimento de redes convolucionais utilizadas atualmente.

No inicio da década de 80 a maioria dos investigadores na área da ciência cognitiva, ciência que procura compreender a mente e o seu funcionamento combinando vários níveis de análise, investigava modelos de cognição baseados em raciocínio simbólico, populares na altura, contudo era difícil explicar como estes modelos poderiam ser implementados no cérebro através de neurónios.

Em 1986 *Rumelhart* apresenta um método de treino de redes multicamada, baseado no conceito de retropropagação e propõe o conceito de **conexionismo** [12], enfantizando a característica de processamento paralelo distribuído das redes neuronais, procurando investigar modelos de cognição baseados em redes neuronais. A ideia central do **conexionismo** é que um grande conjunto de unidades computacionais simples podem apresentar comportamentos inteligentes quando interligadas entre si, uma ideia que se aplica igualmente a neurónios biológicos e a neurónios artificiais.

Do **conexionismo** surgiram alguns conceitos fundamentais para redes neuronais, ainda hoje relevantes, como a representação distribuída que enfatiza a ideia de que cada valor de entrada deve ser representado por várias características e que cada característica deve estar presente no maior número de valores de entrada, para que seja possível à rede aprender cada característica de forma independente.

Na década de 90 são realizados avanços importantes na modelação de sequências em redes neuronais, com *Hochreiter* e *Bengio* a identificarem as principais dificuldades matemáticas na modelação de sequências longas culminando na introdução, por *Hochreiter* e *Schmidhuber*, do modelo de *Long Short-Term Memory* (*LSTM*), um modelo bastante utilizado para o processamento de fala.

A popularidade das redes neuronais veio a diminuir ao longo da década de 90 com as dificuldades de treino destas redes devido ao seu custo computacional elevado, bem como devido à incapacidade de concretização das promessas realizadas por empresas da área e aos avanços realizados em outras áreas de aprendizagem automática como as *support vector machines*.

Durante este período de menor desenvolvimento, o Instituto Canadiano para Pesquisa Avançada (*Canadian Institute for Advanced Research, CIFAR*) promoveu a Computação Neuronal e Percepção Adaptativa, (*Neural Computation and Adaptative Perception, NCAP*), uma iniciativa de investigação que permitiu a colaboração entre diferentes grupos de trabalho de várias universidades, liderados por *Geoffrey Hinton* da Universidade de Toronto, *Yoshua Bengio* da Universidade de Montreal e *Yann LeCun* da Universidade de Nova Iorque, mantendo assim a investigação ativa.

Em 2006 *Geoffrey Hinton* publica o artigo "*To recognize shapes, first learn to generate images*" [13] que iria popularizar, de novo, as redes neuronais e iniciar o período atual de investigação na área da apredizagem profunda (*Deep Learning*).
Este artigo veio demonstrar a possibilidade de treinar, de forma eficiente, cada camada individual para uma rede de crenças profundas (*Deep Belief Network*), com *Bengio* [14] e *Ranzato* [15] a demonstrarem que esta técnica de treino era também aplicável a outro tipo de redes neuronais.

A capacidade de treinar redes neuronais com um elevado número de camadas, aliada ao aumento da disponibilidade e capacidade computacional, permitiu aumentar a popularidade das redes neuronais artificiais e aumentar o seu desempenho, com estas a tornarem-se uma área de investigação muito ativa, quer ao nível académico quer ao nível empresarial.

2.1 Neurónio Biológico

O desenvolvimento de redes neuronais artificiais é baseado no conceito de neurónio artificial, sendo este inspirado no neurónio biológico.

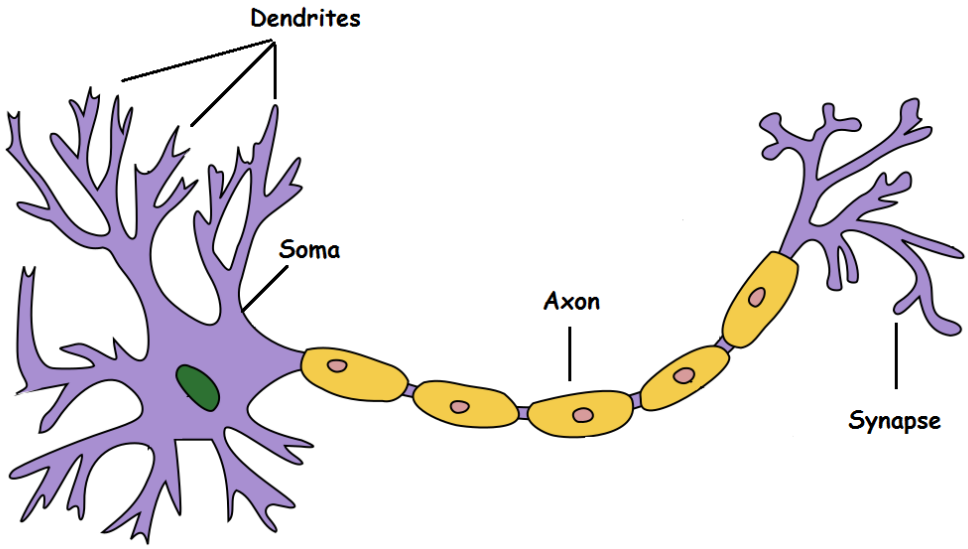


Figura 2.1: Representação de um Neurónio Biológico [16]

A estrutura e modo de funcionamento das redes neuronais e dos neurónios biológicos é ainda hoje largamente desconhecido, no entanto, na sua forma geral, um neurónio biológico é composto por quatro componentes principais:

Dendrites - Recebem informação a partir de outros neurónios.

Soma - O corpo do neurónio, responsável por processar a informação recebida a partir das dendrites e produzir um resultado, por norma na forma de um impulso elétrico.

Axónio - Responsável por transmitir o resultado produzido.

Sinapse - Ponto de ligação às dendrites de outros neurónios.

Para que o **Soma** produza um impulso elétrico é necessário que a informação recebida através das dendrites atinja ou ultrapasse o limiar de activação do neurónio. Sempre que este limiar de activação é atingido, o **Soma** produz um impulso elétrico que é enviado através do **Axónio**, com o neurónio a possuir um activação binária: ou se encontra activo, ou seja, produz um impulso elétrico ou encontra-se inactivo e não é produzido um impulso elétrico.

Como ponto de ligação entre neurónios, as sinapses controlam a influência que um impulso elétrico de um neurónio tem sobre outro neurónio através da sua permeabilidade à passagem do impulso elétrico. É a variabilidade da permeabilidade de cada sinapse que permite a aprendizagem entre os diferentes neurónios.

Um neurónio pode assim ser representado por um conjunto de valores de entrada (representam os impulsos elétricos produzidos pelo **Soma** e transportados pelo **Axónio**), em que cada um possui um peso associado (representando a permeabilidade da Sinapse) que controla a influência do neurónio anterior sobre o actual, esta influência corresponde ao produto entre os dois valores. O conjunto de todos estes produtos é combinado através de uma função de activação que determina se é produzido um impulso elétrico ou não. A Figura 2.2 apresenta um exemplo deste tipo de modelo.

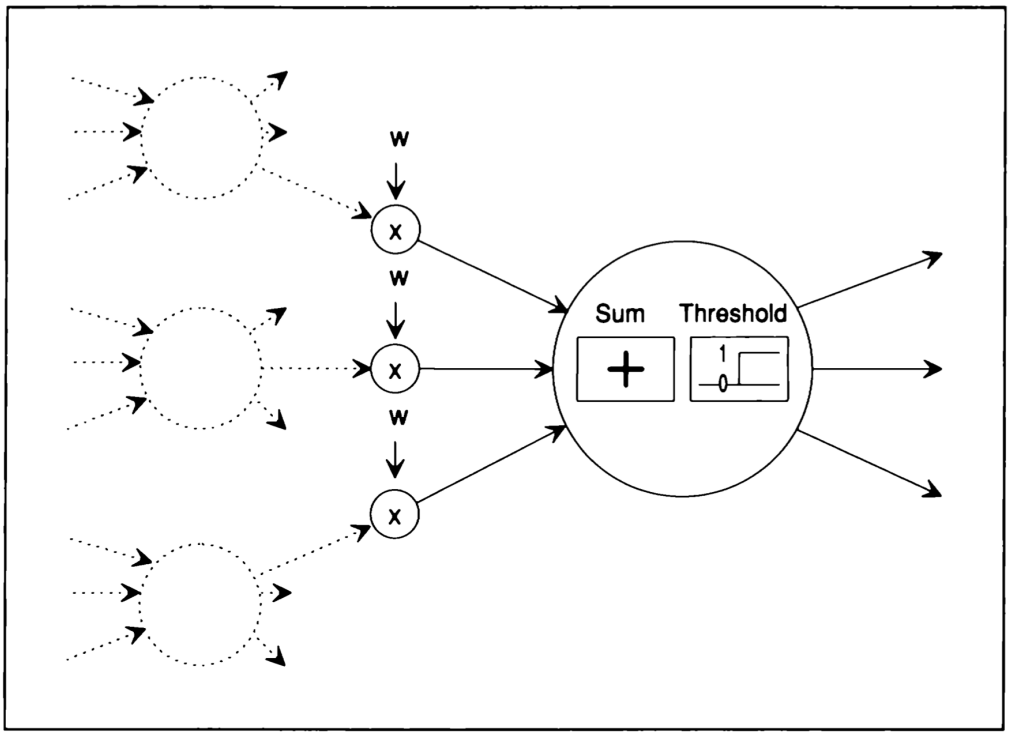


Figura 2.2: Modelo de um neurónio [17]

2.2 Perceptrão

O Perceptrão segue um modelo semelhante ao apresentado na Figura (2.2) com algumas alterações: cada valor de entrada do perceptrão pode ser o resultado de uma combinação de n entradas através de uma função lógica e todos os valores de

entrada são binários. O Perceptrão foi inspirado no modelo biológico da retina, com o intuito de reconhecer padrões a duas dimensões. A Figura (2.3) apresenta o modelo do Perceptrão.

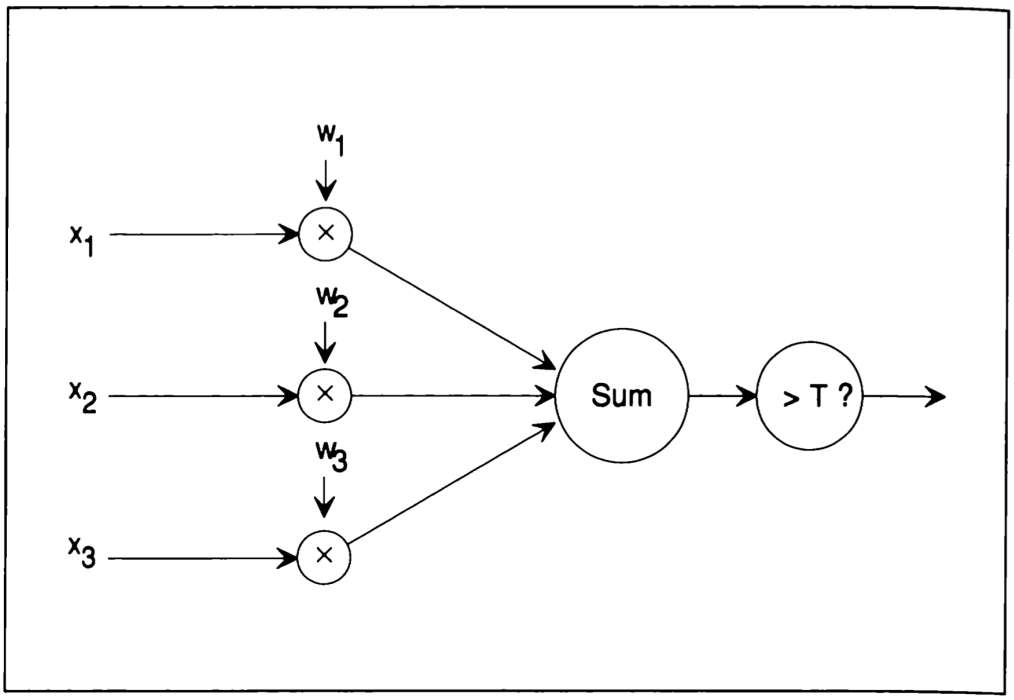


Figura 2.3: Modelo de um Perceptrão [17]

O valor produzido pelo Perceptrão para cada conjunto de valores de entrada é dado por:

$$P = \begin{cases} 1 & \text{se} \quad \sum_i w_i \times x_i > T \\ 0 & \text{caso contrário} \end{cases}$$

O Perceptrão recorre a um algoritmo para aprender o conjunto correto para os valores dos pesos, para uma determinada tarefa, caso ele exista, o qual consiste no seguinte:

1. O valor dos pesos é inicializado com o valor 0.
2. Para cada exemplo é produzido o valor pelo Perceptrão.
3. Caso o Perceptrão produza o valor errado, se o valor produzido foi 0 mas deveria ser 1 então é adicionado o valor de entrada ao peso correspondente, no caso contrário o valor de entrada é subtraído ao peso correspondente.
4. Os passos dois e três são repetidos até que o Perceptrão produza sempre os resultados correctos para todos os exemplos.

Este algoritmo é conhecido como o algoritmo de convergência do Perceptrão.

O Perceptrão, através do seu algoritmo de convergência, deveria conseguir aprender qualquer tipo de função desde que exista um valor T que permita separar os resultados. Na prática verifica-se que, por exemplo, a função XOR não pode ser aprendida pelo Perceptrão, pois requer mais do que um valor T para produzir o resultado correcto.

Para que seja possível ao Perceptrão aprender, é necessário que os padrões possam ser separados linearmente, sendo esta a principal limitação do Perceptrão e de todos os modelos de apenas um neurónio artificial.

2.3 Problema da Separabilidade Linear

O problema da separabilidade linear ocorre para todos os modelos de apenas um neurónio artificial quando a função que procuram aprender é não-linear. A Figura (2.4) apresenta duas funções: uma linearmente separável e uma não-linearmente separável. Para a primeira função é possível determinar, visualmente, que existe uma fronteira linear entre os exemplos representados a vermelho e a azul, o mesmo já não acontece para a segunda função.

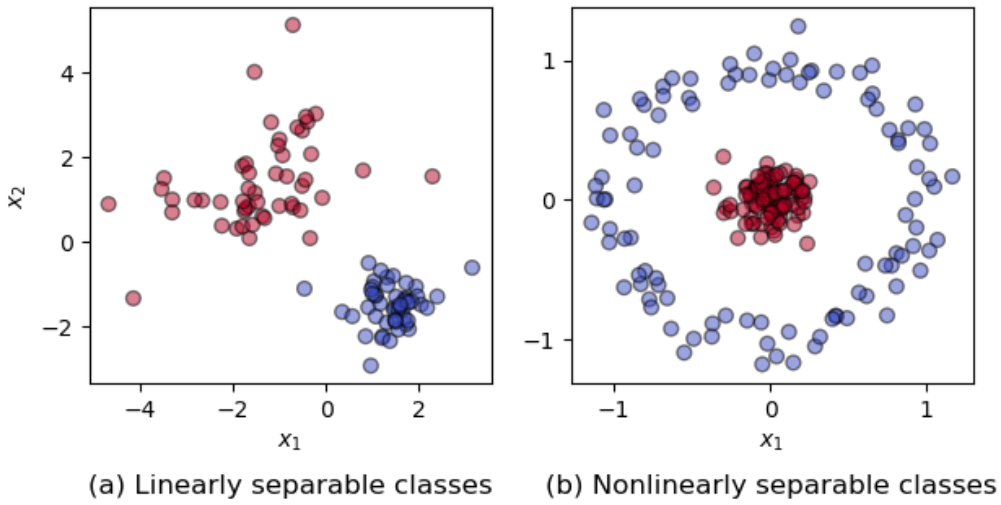


Figura 2.4: Funções linearmente separáveis e não-linearmente separáveis [18]

A incapacidade dos modelos de apenas um neurónio artificial aprenderem funções não-lineares torna-se limitativa com o aumento do número de valores de entrada, para um modelo que aprenda uma função de quatro variáveis binárias cerca de 97% das funções produzidas são não-linearmente separáveis.

Para resolver este problema Teurotzky e Pomerleau propõem, em 1989 [19], a utilização dos valores produzidos por neurónios artificiais como entrada para outros neurónios artificiais, esta forma de organização de neurónios artificiais constitui o que é designado por rede multi-camada.

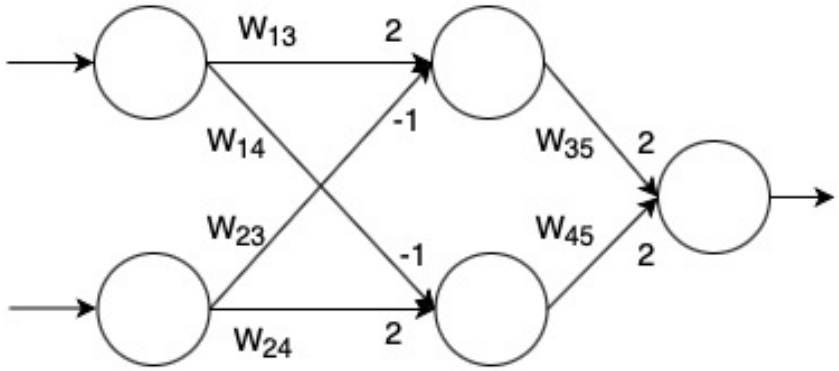


Figura 2.5: Rede Multi-Camada para o problema do XOR [20]

A Figura 2.5 apresenta a configuração e parametrização de uma rede multi-camada para o problema do XOR, as redes multi-camada serão abordadas na secção seguinte.

2.4 Redes Multi-Camada

As redes multi-camada são redes neuronais que produzem resultados a partir da composição de várias camadas de neurónios artificiais.

As redes multi-camada são caracterizadas pelo número de camadas, sendo cada camada caracterizada pelo número de neurónios. Estas características são consideradas a *profundidade* e *largura* da rede e da camada, respectivamente.

As redes multi-camada possuem três tipos camadas:

Camada de Entrada - Este tipo de camada normalmente não realiza operações, é apenas utilizada para receber os valores de entrada do modelo. Por norma, não são aplicadas transformações aos valores recebidos.

Camada de Saída - Este tipo de camada é a última camada da rede e tem como resultado o valor produzido pela rede para os valores de entrada recebidos.

Camada Escondida - Este tipo de camada engloba as transformações que ocorrem entre a camada de entrada e a camada de saída.

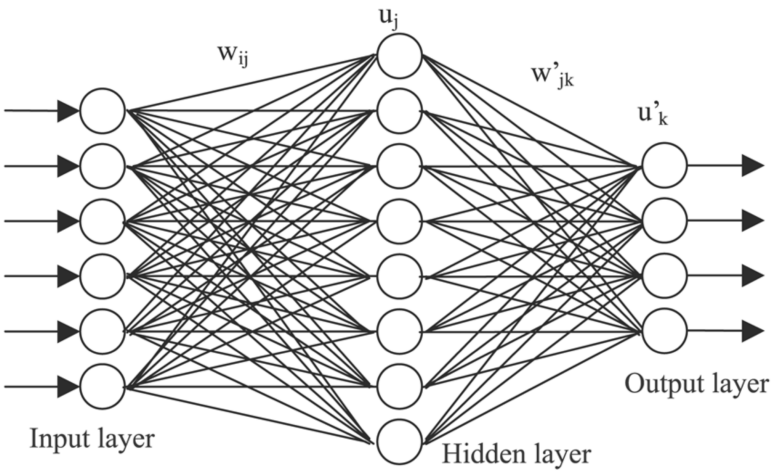


Figura 2.6: Exemplo de Rede Multi-Camada [21]

A Figura (2.6) apresenta um exemplo de uma rede multi-camada.

Com este tipo de redes surge a questão de como propagar o erro obtido na camada de saída através da camada escondida, em 1986 *Rumelhart* propõe o algoritmo de retropropagação [22] que aborda esta questão.

2.5 Retropropagação

Nas redes multi-camada é necessário propagar o erro de modo a que todos os pesos na rede possam ser actualizados em função desse erro, para além disto, cada peso deve ser actualizado de acordo com a sua contribuição para o erro, ao contrário do mecanismo utilizado no Perceptrão que soma ou subtrai o valor 1 ao peso de acordo com o erro.

A Figura (2.7) apresenta uma representação em termos de funções de uma rede neuronal multi-camada, na qual $\boldsymbol{x}$ é um vector que representa os valores de entrada na rede, $\boldsymbol{A}$ e $\boldsymbol{b}$ são vectores que representam os pesos e o pendor, respectivamente, de cada camada i, $f_i(\boldsymbol{x}_{i-1}) = \sigma(\boldsymbol{A}_{i-1}\boldsymbol{x}_{i-1} + \boldsymbol{b}_{i-1})$ representa a transformação que ocorre em cada camada i, $\boldsymbol{x}_{i-1}$ representa o resultado da camada $i-1$ e σ representa a função de activação utilizada. Para uma rede multi-camada a função $\boldsymbol{f}_i$ pode também ser representada como:

$$\boldsymbol{f}_i = \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), i = 1, ..., K, \boldsymbol{f}_0 = \boldsymbol{x} \tag{2.1}$$

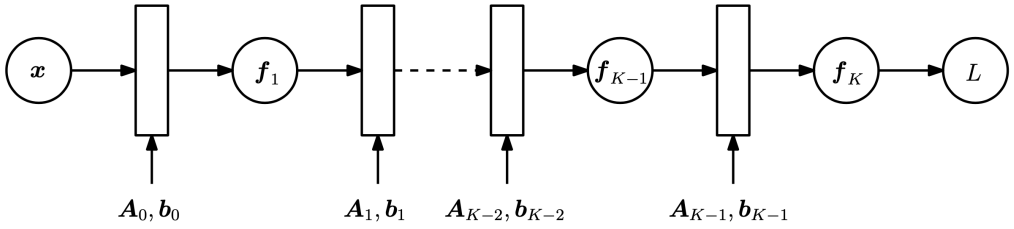


Figura 2.7: Representação em termos de funções de uma rede neuronal multicamada [23]

Para que a aprendizagem ocorra é necessário encontrar o parâmetro $\boldsymbol{\theta}$ que minimize a função de erro $L(\boldsymbol{\theta})$, também designada função de perda (*loss*).

$$\boldsymbol{\theta} = \{\boldsymbol{A}_0, \boldsymbol{b}_0, ..., \boldsymbol{A}_{K-1}, \boldsymbol{b}_{K-1}\} \tag{2.2}$$

Para obter o gradiente da função L é necessário o cálculo da derivada parcial de L para cada $\boldsymbol{\theta}_j$ com $j = 0, ..., K-1$. A regra de cadeia de derivadas indica que o cálculo de uma derivada pode ser composto pelo cálculo das suas derivadas parciais, possibilitando assim a propagação do erro ao longo das diferentes camadas, tal como de seguida indicado:

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} ... \frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_i} \tag{2.3}$$

A Figura (2.8) apresenta o processo de retropropagação do erro através das camadas da rede.

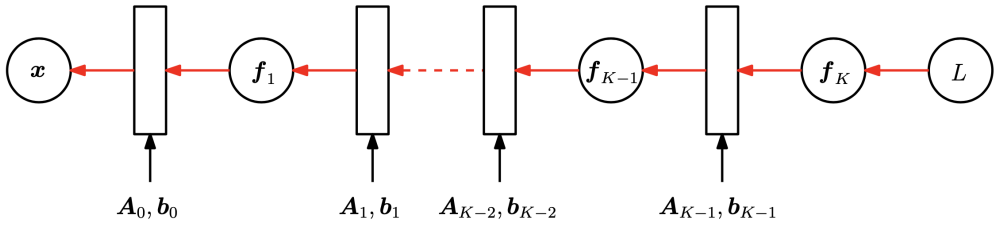


Figura 2.8: Representação da retropropagação de uma rede neuronal multicamada [23]

A retropropagação é desta forma baseada numa técnica de análise numérica designada *diferenciação automática* em modo invertido que calcula o valor de gradientes através de valores intermédios com a aplicação da regra de cadeia de derivadas, a qual permite otimizar os pesos e o pendor de cada neurónio na rede.

2.6 Métodos de Otimização para Treino de Redes Neuronais

De seguida são descritos alguns dos principais métodos de otimização utilizados para o treino de redes neuronais a partir do valor do gradiente obtido na retropropagação.

2.6.1 Descida de Gradiente Estocástica (*SGD*)

O método de otimização mais comum, Descida de Gradiente Estocástica [24], subtrai a cada peso de ligação entre neurónios o valor do gradiente pesado pelo hiperparâmetro taxa de aprendizagem. A equação (2.4) apresenta o novo valor do peso w_j, com η a representar o hiperparâmetro taxa de aprendizagem e $\frac{\partial L}{\partial \boldsymbol{w}_j}$ a representar o gradiente.

$$\boldsymbol{w}_j = \boldsymbol{w}_j - \eta \frac{\partial L}{\partial \boldsymbol{w}_j} \tag{2.4}$$

O valor da taxa de aprendizagem é um dos hiperparâmetros mais difíceis de determinar, pois a utilização de um valor demasiado grande não permite que o valor do peso possa convergir para o valor ideal e um valor demasiado pequeno pode levar a um tempo de treino demasiado longo.

2.6.2 Descida de Gradiente Estocástica com inércia (*SGD with momentum*)

Este método de otimização [25] utiliza uma média móvel exponencial dos valores de gradientes obtidos para atualizar o valor de cada peso, permitindo assim que a atualização do peso seja menos sensível a cada gradiente e mais sensível ao

comportamento do gradiente ao longo do treino. A equação (2.5) apresenta a forma como se calcula o valor médio do gradiente V_t, com o hiperparâmetro β a controlar a proporção entre o valor do gradiente atual e o valor anterior da média utilizado, t representa a iteração atual.

$$\boldsymbol{V}_t = \beta \boldsymbol{V}_{t-1} + (1-\beta)\frac{\partial L}{\partial \boldsymbol{w}_t} \tag{2.5}$$

A equação de atualização do peso é atualizada para utilizar V_t, como apresentado na equação (2.6).

$$\boldsymbol{w}_j = \boldsymbol{w}_j - \eta \boldsymbol{V}_t \tag{2.6}$$

O hiperparâmetro β utiliza valores no intervalo $[0, 1]$, com o valor mais comum a ser $0,9$.

Para as primeiras iterações de treino não existe ainda informação suficiente para que o valor da média obtido possa representar o comportamento do gradiente, por essa razão é possível alterar V_t para que nas primeiras iterações o seu valor seja reduzido, esta versão é apresenta na equação (2.7).

$$\boldsymbol{V}'_t = \frac{\boldsymbol{V}_t}{1-\beta^t} \tag{2.7}$$

2.6.3 *Root Mean Square Propagation (RMSProp)*

Este método de otimização [26] mantêm a média móvel exponencial do quadrado dos gradientes, dividindo a taxa de aprendizagem pela raíz quadrada desta média, permitindo assim escalar a taxa de aprendizagem de acordo com a situação. A equação (2.8) apresenta a média móvel do quadrado dos gradientes com o hiperparâmetro β, t a representar a iteração atual e g a representar o gradiente.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\frac{\partial L}{\partial \boldsymbol{w}_t})^2 \tag{2.8}$$

A equação (2.9) apresenta a atualização do valor do peso para este método.

$$\boldsymbol{w}_t = \boldsymbol{w}_t - \frac{\eta}{\sqrt{E[g^2]_t}} \frac{\partial L}{\partial \boldsymbol{w}_t} \tag{2.9}$$

2.6.4 *Adaptive Moment Estimation (Adam)*

Este método de otimização [27] pode ser interpretado como uma combinação do método de otimização Descida de Gradiente Estocástica com inércia (*SGD with momentum*) e *Root Mean Square Propagation (RMSProp)*, utilizando a média móvel exponencial dos gradientes como o valor do gradiente a utilizar e a raíz quadrada da média móvel exponencial do quadrado dos gradientes para adaptar a taxa de aprendizagem, a equação (2.10) apresenta a média móvel exponencial dos gradientes com o hiperparâmetro β_1 e a equação (2.11) apresenta a média móvel exponencial do quadrado dos gradientes com o hiperparâmetro β_2.

$$\boldsymbol{V}_t = \beta_1 \boldsymbol{V}_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial \boldsymbol{w}_t} \tag{2.10}$$

$$E[g^2]_t = \beta_2 E[g^2]_{t-1} + (1 - \beta_2) (\frac{\partial L}{\partial \boldsymbol{w}_t})^2 \tag{2.11}$$

A equação (2.12) apresenta a equação atualizada do valor do peso.

$$\boldsymbol{w}_t = \boldsymbol{w}_t - \frac{\eta}{\sqrt{E[g^2]_t}} \boldsymbol{V}_t \tag{2.12}$$

2.6.5 *Adaptive Gradient Algorithm (AdaGrad)*

Este método de otimização [28], ao contrário do método Descida de Gradiente Estocástica com inércia (*SGD with momentum*) que procura substituir o valor do gradiente pela sua média móvel exponencial, este método procura adaptar a taxa de aprendizagem através da acumulação do produto externo dos gradientes, como apresentado na equação (2.13), com $\boldsymbol{G}_t$ a representar uma matriz dos gradientes.

$$\boldsymbol{G}_t = \sum_{\tau=1}^{t} \frac{\partial L}{\partial \boldsymbol{w}_\tau} \left(\frac{\partial L}{\partial \boldsymbol{w}_\tau}\right)^T \tag{2.13}$$

A equação (2.14) apresenta a equação atualizada do novo valor do peso, com ε a representar um valor bastante pequeno para evitar a divisão por 0 e $diag(\boldsymbol{G}_t)$ a diagonal de $\boldsymbol{G}_t$.

$$\boldsymbol{w}_t = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\varepsilon + diag(\boldsymbol{G}_t)}} \frac{\partial L}{\partial \boldsymbol{w}_t} \tag{2.14}$$

Este método de otimização poderá, dependendo da situação, impedir a aprendizagem pois a acumulação de valores muitos elevados poderá diminuir a taxa de aprendizagem para valores muito próximos de 0.

2.7 O Processo de Treino

Em aprendizagem automática é necessário treinar o sistema para que este possa aprender, crianda um modelo interno, este treino tem por base o conjunto de dados disponível.

O conjunto de dados é separado em dois subconjuntos: o conjunto de dados de treino e o conjunto de dados de teste, o conjunto de treino é utilizado para treinar o sistema e o conjunto de teste é utilizado para verificar o seu desempenho. Caso o conjunto de dados possua o valor de classificação, etiqueta (*label*), associado a cada exemplo, a aprendizagem é considerada supervisionada, caso contrário a aprendizagem é não-supervisionada.

De modo a garantir que o modelo criado produz o melhor desempenho possível, por vezes são realizadas alterações ao conjunto de treino como o aumento de dados (*data augmentation*), uma técnica que procura aumentar o número de exemplos para permitir um melhor treino do modelo, ou novas divisões no conjunto de treino para permitir mais iterações de treino, como acontece com o método *K-fold cross validation*, que divide o conjunto de treino em *K* conjuntos de treino e teste e que treina o modelo em *K-1* conjuntos e utiliza o restante conjunto para treino.

Após o treino do modelo com o conjunto de treino, é verificado o desempenho do modelo no conjunto de teste para garantir que o modelo possui um bom desempenho para exemplos onde não foi treinado.

Em algumas situações o modelo pode apresentar um excelente desempenho no conjunto de treino mas esse desempenho não ocorre no conjunto de teste o que significa que existe um problema de especialização excessiva ao conjunto de treino. Esse problema é designado de *sobreparametrização* ou sobre-especialização (*overfitting*).

2.8 O Problema da Sobreparametrização

O problema da sobreparametrização ocorre quando o modelo aprende as especificidades dos exemplos no conjunto de treino e não as características genéricas dos exemplos, de modo que quando é apresentado um exemplo que não foi treinado o modelo não consegue obter o mesmo desempenho.

A Figura (2.9) apresenta o desempenho de dois modelos, o modelo sem sobreparametrização (linha negra) e com sobreparametrização (linha verde), o modelo com sobreparametrização apresenta um desempenho perfeito para os exemplos de treino, contudo o modelo sem sobreparametrização irá obter melhor desempenho para novos exemplos.

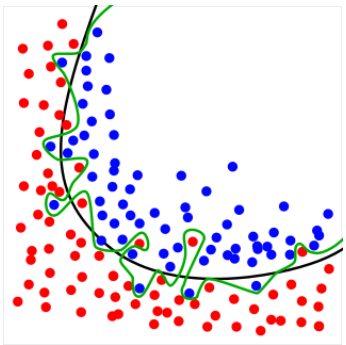


Figura 2.9: Exemplo de dois modelos, com e sem sobreparametrização [29]

Para redes neuronais profundas é possível utilizar algumas técnicas, como o decaimento dos pesos da rede (*Weight Decay*), constrangimento dos valores de ativação da rede (*Activity Regularization*), constrangimento dos valores dos pesos da rede (*Weight Constraint*), a remoção probabilística de alguns valores de entrada durante o treino (*Dropout*) e a utilização de paragem antecipada (*Early Stopping*).

No decaimento dos pesos da rede é adicionada uma penalização na função de perda de acordo com a magnitude dos pesos, forçando a aprendizagem a otimizar também para valores de menor magnitude. A equação (2.15) apresenta a fórmula genérica para adicionar o decaímento dos pesos na rede, com o valor λ, a taxa de decaímento, a representar o nível de penalização e a função L a representar a função de perda original.

$$L' = L + \lambda w \tag{2.15}$$

No constrangimento dos valores de ativação da rede é aplicada uma função de regularização ao valor de ativação das camadas escolhidas ou a função de regularização pode ser aplicada antes da função da ativação da camada. As funções de regularização mais comuns são: L1 (soma dos valores absolutos) e L2 (soma do quadrado dos valores).

No constrangimento dos valores dos pesos da rede é aplicada uma função aos pesos das camadas escolhidas que, caso o valor dos pesos ultrapasse os limites definidos, estes são alterados de modo a respeitarem os limites definidos. Esta técnica é, por vezes, utilizada como alternativa ao decaímento dos pesos.

A remoção probabilística de alguns valores de entrada durante o treino permite reduzir possíveis dependências entre camadas, forçando a rede a aprender uma representação esparsa dos dados pois estes podem não estar disponíveis. A utilização desta técnica pode forçar a utilização de redes mais largas devido à remoção temporária de alguns valores.

A utilização de paragem antecipada consiste na avaliação do desempenho da rede e caso se detete uma perda de desempenho o treino da rede é terminado, contudo, devido à natureza estocástica da rede é possível que uma ligeira perda de desempenho termine o treino demasiado cedo, por essa razão, a perda de desempenho é medida ao longo de várias iterações de treino para garantir que a perda de desempenho que se verificou não é uma situação pontual. A medição de perda de desempenho pode consistir na perda ao longo de várias iterações, uma perda no valor médio ao longo de várias iterações ou até a estabilidade do desempenho.

2.9 Conclusão

Neste capítulo descreveu-se os componentes principais do neurónio biológico, que serve de inspiração para o neurónio artificial, o Perceptrão, o algoritmo de aprendizagem utilizado e as limitações associadas ao Perceptrão.
Foi também abordado o problema da separabilidade linear para sistemas de apenas um neurónio e a introdução de redes multi-camada, com os diferentes tipos de camada possíveis.
Foi apresentado o algoritmo de retropropagação, necessário em redes multi-camada para permitir a aprendizagem em todas as camadas, bem como alguns dos principais métodos de optimização para o treino de redes multi-camada através da retropropagação.
Foi também abordado o processo de treino em redes multi-camada e o problema da sobreparametrização que pode surgir no processo de treino.

3

Redes Neuronais Profundas

Este capítulo descreve a estrutura associada a redes neuronais profundas, os diferentes tipos de camadas e como se processa a aprendizagem neste tipo de redes, com especial foco em redes neuronais convolucionais.

3.1 Estrutura de Redes Neuronais Profundas

As redes neuronais profundas são caracterizadas pelas suas múltiplas camadas, desde configurações totalmente homogéneas como o Perceptrão Multi-Camada, até redes com realimentação entre a saída de uma camada e a entrada desta camada ou de uma camada anterior, com este tipo de redes a ser designada por redes neuronais recorrentes (*Recurrent Neural Network* ou *RNN*).

Com o aumento do poder computacional tornou-se possível a utilização de redes cada vez mais profundas e com maior heterogeneidade no tipo e organização das suas camadas, permitindo desenvolver redes mais complexas. Algumas destas redes possuem mais de mil camadas [30], existindo redes com aproximadamente novecentos e vinte e oito milhões de parâmetros [31].

As redes neuronais profundas são compostas por quatro tipos de camadas mais gerais: *totalmente ligada*, o tipo de camada mais comum, *convolucional*, um tipo de camada utilizado para extrair características através de filtros, *deconvolucional*, um tipo de camada utilizado para aumentar a resolução da informação e, por último,

recorrente, um tipo de camada que possui memória e que utiliza essa memória como valores de entrada para além dos dados disponíveis.

Para interligar as camadas existem várias opções, desde uma ligação simples, ou seja, sem qualquer modificação da informação, a ligações que modificam a informação como *dropout* que "desliga" certas ligações durante o processo de treino e *pooling* que reduz a dimensão dos dados recorrendo à média (*average pooling*), ao valor máximo (*max pooling*) ou ao valor mínimo (*min pooling*).

Na arquitetura de redes neuronais profundas, procurou-se inicialmente utilizar redes mais largas e menos profundas, mas redes mais profundas e menos largas têm vindo a obter melhor desempenho [32], permitindo ao mesmo tempo reduzir o número de parâmetros a ser treinado.

3.2 Tipos de Camadas

São de seguida apresentados com maior detalhe os diferentes tipos de camadas existentes em algumas redes neuronais profundas.

3.2.1 Camada Totalmente Ligada

Uma camada totalmente ligada é uma camada em que cada neurónio da camada estabelece uma ligação a cada neurónio da camada seguinte, sendo o tipo de camada mais comum e utilizada em todo o tipo de redes.

Este tipo de camada requer um aumento de capacidade computacional elevado com o aumento do número de neurónios presentes, devido ao aumento do número de operações a realizar, o que dificulta a sua utilização em redes com dados de elevada dimensionalidade e, por isso, são maioritariamente utilizadas no fim da rede.

Esta camada possui como parâmetros o número de neurónios presentes na camada, a função de ativação e a utilização de *dropout*.

3.2.2 Camada Deconvolucional

Uma camada deconvolucional é uma camada utilizada para aumentar a resolução dos dados, quer de imagens quer de mapas de características.

É maioritariamente utilizada para adaptar dados de menor resolução para camadas que requerem dados de maior resolução.

Esta camada possui como parâmetros a dimensionalidade, a estratégia de *padding*, a função de ativação e a utilização de *dropout*.

3.2.3 Camada Recorrente

Uma camada recorrente é uma camada que possui memória, ou seja, consegue utilizar como valor de entrada valores produzidos na sua saída.

É bastante utilizada em dados sequenciais como o processamento de linguagem natural.

Esta camada possui como parâmetros a dimensionalidade, o tipo de rede recorrente e a utilização de *dropout*.

3.2.4 Camada Convolucional

A camada convolucional é uma camada utilizada para detetar características em imagens através de mapas de características. Cada mapa de características é produzido através do processamento de pequenas regiões da imagem com um filtro, reduzindo assim o número de parâmetros a ser calculado.

É maioritariamente utilizada para reconhecimento e/ou classificação de imagens.

Esta camada possui como parâmetros a dimensionalidade, a dimensão de processamento de cada região, o número de pixeis que avança entre cada processamento (*stride*), o número de mapas de características a gerar, a estratégia de *padding* a utilizar, a função de ativação, a utilização de *dropout* e a utilização de *pooling*.

Este tipo de camada e redes que utilizam este tipo de camada vão ser apresentados na secção 3.3.

3.3 Redes Convolucionais

As redes convolucionais são um tipo de rede neuronal em que pelo menos uma das suas camadas recorre à convolução e é normalmente utilizada para extrair características presentes em imagens e em sequências temporais.

3.3.1 Convolução

A operação de *convolução* é uma operação sobre duas funções, x e w, tal como descrito na equação (3.1), simplificada na equação (3.2) e para valores discretos (3.3). Para valores discretos a duas dimensões a convolução é descrita por (3.4).

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da \tag{3.1}$$

$$s(t) = (x * w)(t) \tag{3.2}$$

$$(x * w)(n) = \sum_{m=-\infty}^{\infty} x(m)w(n-m) \tag{3.3}$$

$$S(i,j) = (X * W)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(i,j)W(i-m, j-n) \tag{3.4}$$

A convolução é comutativa, ou seja, $(x * w)(t)$ é igual a $(w * x)(t)$, a equação (3.5) apresenta a versão comutativa da equação (3.4).

$$S(i,j) = (W * X)(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} X(i-m, j-n)W(i,j) \tag{3.5}$$

Para além da convolução existe também a correlação cruzada, definida na equação (3.6) para valores contínuos e na equação (3.7) para valores discretos e a sua relação com convolução na equação (3.8).

$$(x \star w)(t) = \int_{-\infty}^{\infty} \overline{x(a)}w(t+a)da \tag{3.6}$$

$$(x \star w)(n) = \sum_{m=-\infty}^{\infty} \overline{x(m)} w(n+m) \tag{3.7}$$

$$[x(t) \star w(t)](t) = [\overline{x(-t)} * w(t)](t) \tag{3.8}$$

Para valores reais a correlação cruzada produz um resultado simétrico ao da convolução.

Em redes neuronais convolucionais o argumento X é designado por valor de entrada (*input*) e utiliza o símbolo I e a argumento W é designada por filtro (*kernel*) e utiliza o símbolo K, com o resultado da convolução a ser designado por mapa de características (*feature map*). Por norma o filtro K possui dimensões inferiores ao valor de entrada I.

3.3.2 Aplicações em Redes Neuronais

As redes convolucionais representam uma das primeiras aplicações de redes neuronais profundas bem sucedidas, com a aplicação do algoritmo de retropropagação em redes neuronais com atraso (*Time-Delay Neural Network*), redes neuronais multi-camada que utilizam camadas convolucionais, em 1988 [33], no ano seguinte [34] é desenvolvida a rede neuronal convolucional moderna agora aplicada em imagens.

O aumento do interesse em redes neuronais convolucionais profundas ocorre em 2012 [35] devido à obtenção do melhor desempenho, na altura, para o conjunto de dados de referência *ImageNet* [36].

3.3.2.1 Inspiração Biológica

As redes convolucionais representam um dos maiores sucessos de aplicação de métodos de inspiração biológica em sistemas de inteligência artificial. Após vários anos de estudo *Hubel* e *Wiesel* [37], [38], [39] determinaram as características básicas do funcionamento do sistema de visão de uma mamífero, com a maior influência em redes neuronais a surgir do estudo da actividade de neurónios individuais em gatos. Desses estudos, observou-se, por exemplo, que no sistema visual os primeiros neurónios respondem com maior intensidade a padrões muito específicos, como linhas verticais, mas ignoram outros padrões.

3.3.2.2 Motivação

A utilização de redes convolucionais procura tirar partido de três características tornadas possíveis pelo uso da convolução: **conectividade esparsa**, **partilha de parâmetros** e **representações equivariantes**.

Nas camadas de redes neuronais totalmente ligadas existe um parâmetro para cada interação entre um valor de entrada e um valor de saída para cada neurónio, o que requer $m \times n$ parâmetros, no caso de camadas de redes neuronais convolucionais cada valor de saída apenas interage com k valores de entrada reduzindo assim o número de parâmetros a $k \times n$, com $k \ll m$. A redução no número de parâmetros reduz os requisitos de memória e o tempo necessário para realizar o processamento dos valores de entrada pela camada. A Figura (3.1) apresenta um exemplo na redução do número de parâmetros possível pela utilização de conectividade esparsa.

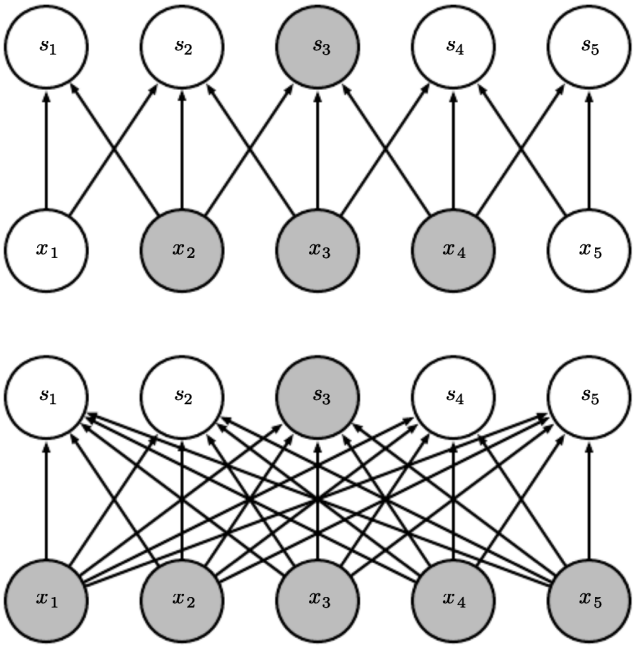


Figura 3.1: No primeiro exemplo, visto de cima para baixo, é possível verificar que cada valor de saída apenas depende de três valores de entrada sendo apenas necessários três parâmetros por valor de saída, no segundo exemplo, devido à conectividade totalmente ligada cada valor de saída depende dos cinco valores de entrada e consequentemente são necessários cinco parâmetros por valor de saída [40]

A **partilha de parâmetros** refere-se à reutilização de um parâmetro para mais do que um valor de saída. No caso de uma camada totalmente ligada cada parâmetro é utilizado apenas uma vez, em camadas convolucionais cada parâmetro

é utilizado em todos os valores de entrada, dependendo do método de processamento de valores nas bordas é possível que nem todos os valores utilizem todos os parâmetros, permitindo assim aprender apenas um conjunto de parâmetros para todos os valores de entrada. A partilha de parâmetros permite assim reduzir os requisitos de memória. A Figura (3.2) apresenta um exemplo da partilha de parâmetros para camadas convolucionais quando comparadas com camadas totalmente ligadas.

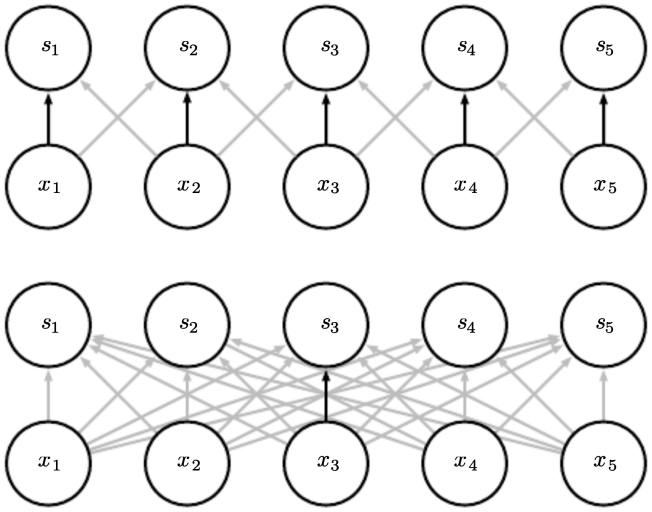


Figura 3.2: Cada seta negra indica a utilização do mesmo parâmetro entre valores de entrada e saída diferentes. No primeiro exemplo, visto de cima para baixo, verifica-se a partilha de um parâmetro para uma camada convolucional entre todos os valores de entrada, no segundo exemplo, para uma camada totalmente ligada o parâmetro representado pela seta negra apenas é utilizada para um valor de entrada [41]

Devido à partilha de parâmetros a camada convolucional possui também **equivariância** a deslocações, ou seja, para qualquer deslocação que ocorra nos valores de entrada essa deslocação estará também presente nos valores de saída, por exemplo, se os valores de entrada representarem uma sequência temporal, se um evento ocorre mais recentemente nos valores de entrada a mesma representação desse evento irá surgir mais cedo, se os valores de entrada representarem uma imagem, a deslocação de um determinado objecto na imagem irá produzir a deslocação da sua característica no resultado produzido. A camada convolucional não apresenta **equivariância** para rotações e alterações de escala, pelo que técnicas como o *zoom* numa imagem irão produzir resultados diferentes em relação ao processamento da imagem sem a aplicação dessa técnica.

Estas três características associadas à convolução tornam as camadas convolucionais mais eficientes na utilização de memória e reduzem o tempo de computação necessário para calcular os seus resultados, permitindo assim o processamento de um maior volume de dados de entrada e a utilização de mais camadas.

3.3.2.3 *Pooling*

Uma camada convolucional típica é composta por três etapas: a etapa de cálculo que realiza as convoluções, denominada por etapa convolucional, a etapa de activação que utiliza a função de ativação específicada para cada valor produzido na etapa anterior, denominada de etapa de deteção, e a etapa de redução que utiliza uma função de redução, esta função substitui um conjunto de valores produzidos por outros. Algumas das funções de redução mais populares incluem a função *max pooling*, a média dos valores do conjunto ou a distância ponderada ao valor central do conjunto. A Figura 3.3 apresenta um exemplo de *max pooling* e a Figura 3.4 apresenta a composição de uma camada convolucional com as três etapas acima descritas.

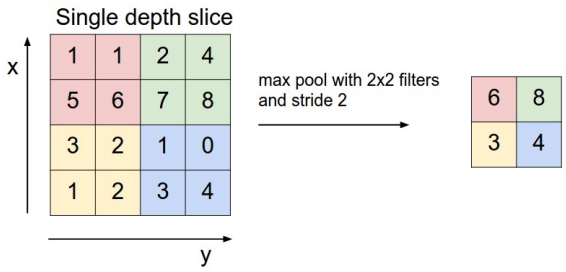


Figura 3.3: Exemplo de *max pooling*[42]

A utilização da função de redução pretende aumentar a invariância a pequenas deslocações no valor de entrada, pois para pequenas deslocações o valor resultante não é alterado. Para situações em que o mais importante é a presença de uma característica e não a sua localização em concreto, esta invariância é fundamental na melhoria de desempenho. A etapa de redução (*pooling*) também permite reduzir a dimensionalidade dos dados, reduzindo assim os requisitos de memória e o tempo de cálculo da camada seguinte.

3.3.2.4 *Padding*

Na operação de convolução, o filtro K apenas pode ser utilizado em posições para o qual existam sempre valores em I, reduzindo o número de valores utilizados na

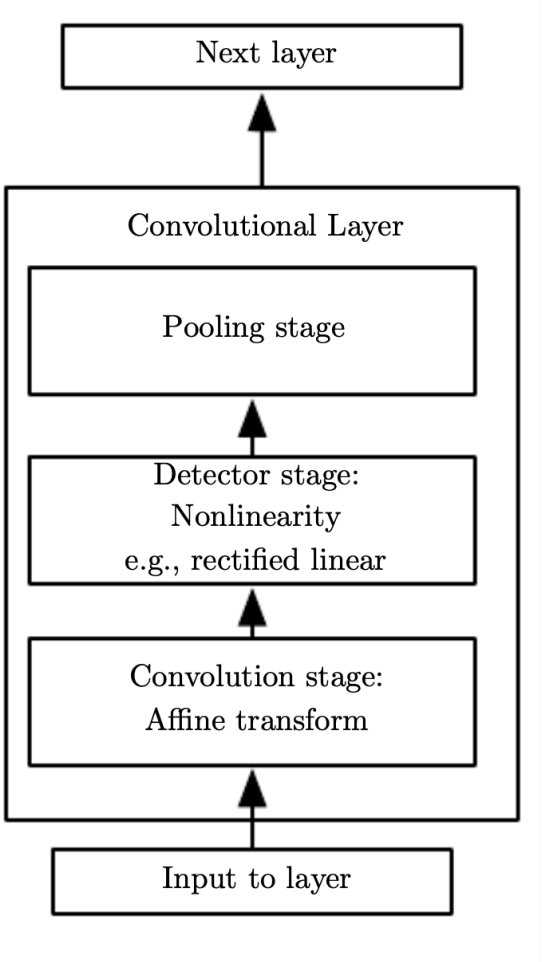


Figura 3.4: Composição de uma camada convolucional com as etapas, de baixo para cima, etapa de cálculo (*Convolution Stage*), etapa de deteção (*Detector Stage*) e etapa de redução (*Pooling Stage*)[43]

convolução o que produz como resultado um mapa de característica de menores dimensões comparativamente ao valor de entrada I, esta convolução é designada como convolução *válida* (*valid*). A diminuição do mapa de característica produzido é dada por $m-k+1$, com m largura do valor de entrada I, k a largura do filtro K, para valores bidimensionais o mesmo acontece com a altura. Esta redução irá diminuir a dimensão do valor de entrada I até que não seja possível realizar a convolução limitando o número de camadas convolucionais que podem ser utilizadas numa rede.

Para garantir que não existe redução de dimensões é possível utilizar *padding*, que adiciona o mínimo de elementos possível ao valor de entrada I para que realização da convolução ocorra sem uma redução de dimensões do mapa de características em relação ao valor de entrada I. Uma convolução que utiliza este tipo de

padding é designada como uma convolução *idêntica* (*same*) pois não existem diferenças de dimensões entre o valor de entrada I e o mapa de características obtido. A utilização deste tipo de convolução remove a limitação no número de camadas convolucionais que podem ser utilizadas numa rede, contudo os valores que se encontram nos limites do valor de entrada I possuem menor influência no mapa de características pois são utilizados menos vezes, o que pode ser prejudicial caso estes valores possuam informação relevante.

Para garantir que todos os valores presentes no valor de entrada I possuem a mesma influência no mapa de características são adicionados o número de elementos necessários, uma convolução que utiliza este tipo de *padding* é designada como uma convolução *completa* (*full*). Esta convolução leva a um aumento das dimensões do mapa de características em relação ao valor de entrada I, com um aumento de $m + k - 1$, com m largura do valor de entrada I e k a largura do filtro K.

O *padding* é uma técnica que adiciona elementos de modo a garantir que uma determinada dimensão é atingida. O valor utilizado para realizar o *padding* pode variar de acordo com a situação mas é comum que o valor utilizado seja zero.

Em redes convolucionais o tipo de *padding* que possui melhor desempenho varia entre a ausência de *padding* (convolução **válida**) e a utilização do *padding* mínimo (convolução **idêntica**).

3.4 Aprendizagem em Redes Neuronais Profundas

3.4.1 Parametrização da Camada Convolucional

Cada camada convolucional possui múltiplos hiperparâmetros: a sua dimensionalidade, a dimensão do filtro, o número de filtros utilizados, qual a função de *pooling* utilizada, a distância que o filtro se deve deslocar para cada convolução, qual o tipo de *padding* utilizado, qual a percentagem de *dropout* a utilizar e a função de ativação.

A dimensionalidade da camada é normalmente definida pelas dimensões do valor de entrada, embora em algumas situações a dimensionalidade da camada seja pré-determinada e os valores de entrada alterados para a dimensionalidade da camada.

A dimensão do filtro está relacionada com a dimensão da área em que o filtro

procura por uma característica, com um filtro de maiores dimensões a possuir um maior custo computacional.

O número de filtros depende do número de características que se pretende extrair, por norma, camadas mais perto da camada de entrada possuem menor número de filtros que camadas mais perto da camada de saída.

3.4.2 Parametrização da Rede

O processo de aprendizagem consiste em iterar sobre o conjunto de treino na rede, cada iteração processa um conjunto de exemplos, denominado por *batch*, em que o número de exemplos é denominado por tamanho do *batch*, de cada vez, ativando a rede para cada exemplo e acumulando o erro associado a cada exemplo, quando termina o *batch* é realizado o processo de aprendizagem através de retropropagação, com os pesos de cada camada a serem atualizados para reduzir o erro. A ordem dos exemplos de iteração pode variar de modo a aumentar a resiliência da rede.

Antes de iniciar o processo de treino é necessário definir alguns hiperparâmetros, como o número de iterações (*epochs*), a função de otimização da rede, a função de erro, a taxa de aprendizagem e a taxa de decaimento dos pesos.

No processo de aprendizagem da rede existem vários métodos de optimização disponíveis como mencionado na secção Métodos de Otimização para Treino de Redes Neuronais.

A função de erro calcula qual o valor do erro entre o valor predito pela rede e o valor correto (para aprendizagem supervisionada). E existem vários tipos de funções de erro, nomeadamente, probabilísticas e de regressão. As funções de regressão mais comuns são: Erro Quadrático Médio (*MSE*), Erro Absoluto Médio (*MAE*) e o Erro Logarítmico Quadrado Médio (*MSLE*), entre outras. As funções probabilísticas mais comuns são: Erro de Entropia Cruzada Binária (*BCE* ou *Binary Cross Entropy*), Erro de Entropia Cruzada Categórica (*CCE* ou *Categorial Cross Entropy*), entre outras.

A inicialização dos pesos da rede é, por norma, obtida aleatoriamente através de uma distribuição normal, com o valor mínimo e máximo a serem hiperparâmetros desta função. A inicialização dos pesos da rede pode contribuir para o problema de redução extrema (*vanishing*) dos gradientes, impedindo assim que a rede aprenda durante o processo de treino.

3.5 Conclusão

Neste capítulo descreveu-se a estrutura de uma rede neuronal profunda, os diferentes tipos de camadas e as suas características e a aprendizagem para estas redes. No caso de redes convolucionais, que servem de base ao trabalho experimental realizado, foi descrita a operação de convolução e como esta é utilizada no contexto de redes convolucionais, as diferentes aplicações deste tipo de redes, a inspiração em modelos biológicos, as características associadas e as diferentes etapas associadas a camadas convolucionais.

4

Transferência de Aprendizagem

Atualmente o uso de aprendizagem automática continua a aumentar na sociedade, desde dispositivos móveis como o *iPhone*, o *iPad* e dispositivos *Android* com assistentes pessoais (*Siri [44], Google Assistant [45], Alexa [46] e Cortana [47]*) cada vez melhores, equipamentos que simulam a percepção de profundidade [48] a novas bibliotecas que permitem treinar/utilizar modelos diretamente em dispositivos móveis [49, 50], contudo a maioria dos algoritmos utilizados operam sobre uma assunção comum: o conjunto de dados de treino e os dados reais possuem o mesmo conjunto de características e a mesma distribuição. Sempre que esta assunção não se verifica, é necessário voltar a treinar o classificador com os novos dados e/ou distribuição o que implica um esforço elevado. A capacidade de tirar partido de modelos pré-existentes e do conhecimento neles contido permitiria obter melhores soluções e/ou reduzir mais rapidamente os custos associados a re-treinar o classificador.

A transferência de aprendizagem aborda este problema, permitindo que os domínios, tarefas e distribuições utilizados para treino sejam diferentes dos dados de teste, uma situação que se verifica no dia-a-dia, quando qualquer indivíduo tira partido do seu conhecimento em diferentes áreas para apoiar as suas decisões como, por exemplo, a capacidade de reconhecer maçãs facilita a capacidade de um indivíduo reconhecer outro tipo de frutas como laranjas.

A transferência de aprendizagem é definida como a capacidade de um sistema reconhecer e aplicar conhecimento e competências adquiridos previamente em

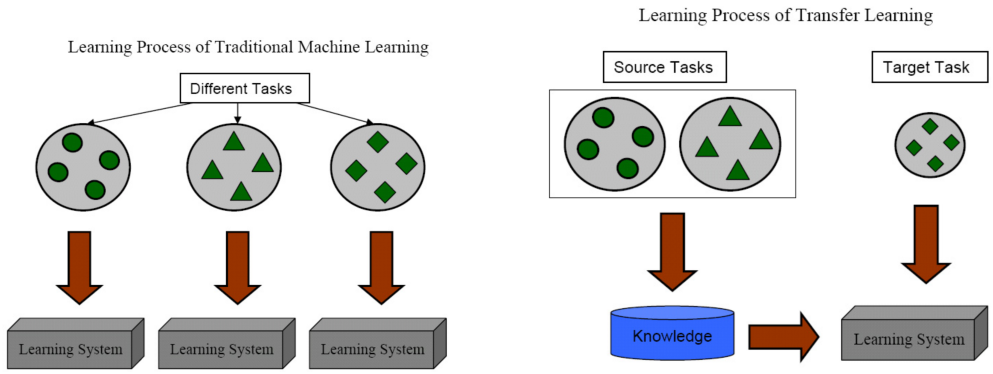
novas tarefas [51], ou seja, o processo de transferência de aprendizagem procura extrair conhecimento a partir de uma ou mais tarefas de origem e aplicá-lo na tarefa de destino.

Na transferência de aprendizagem existem três focos principais de investigação:

O que transferir - Quais as partes do conhecimento que devem ser transferidas, como distinguir entre o conhecimento específico a cada tarefa e o conhecimento comum entre as diferentes tarefas.

Como transferir - Como se transfere o conhecimento, quais os algoritmos a utilizar para realizar esta transferência.

Quando transferir - Determinar em que situações é que a transferência deve ser realizada e também quando não deve ser realizada[1].



(a) *Aprendizagem Automática* Tradicional (b) Transferência de Aprendizagem

Figura 4.1: Diferenças entre *aprendizagem automática* tradicional e transferência de aprendizagem [51]

A Figura (4.1) apresenta as diferenças entre um sistema de aprendizagem automática tradicional (4.1(a)), que procura aprender cada tarefa individualmente ao contrário de um sistema de transferência de aprendizagem (4.1(b)) que procura tirar partido do conhecimento já obtido nas tarefas de origem.

[1]Situações em que a transferência é forçada podem levar a perdas de desempenho.

4.1 Introdução

Como referido anteriormente, a transferência de aprendizagem permite que diferentes domínios, tarefas e distribuições sejam utilizados, de seguida encontram-se as definições e as notações utilizadas para definir estes conceitos.

$$\mathcal{D} \text{ representa o domínio} \tag{4.1}$$

$$\mathcal{D} = \{\mathcal{X}, \mathcal{P}(X)\} \tag{4.2}$$

$$\mathcal{P}(X) \text{ é a distribuição de probabilidade} \tag{4.3}$$

$$X = \{x_1, ..., x_n\} \quad \epsilon \quad \mathcal{X} \tag{4.4}$$

$$X \text{ representa o espaço de características} \tag{4.5}$$

Como indicado na expressão (4.2) um **domínio** $\mathcal{D}$ é constituído por dois componentes: o espaço de características $\mathcal{X}$ e distribuição de probabilidades $\mathcal{P}(X)$.

Definição de Tarefa

$$\mathcal{T} = \{\mathcal{Y}, f(x)\} \tag{4.6}$$

$$\mathcal{T} = \{\mathcal{Y}, \mathcal{P}(Y|X)\} \tag{4.7}$$

$$Y = \{y_1, ..., y_n\} \quad \epsilon \quad \mathcal{Y} \tag{4.8}$$

Como indicado na expressão (4.6) uma **tarefa** $\mathcal{T}$ é constituída por dois componentes: o espaço de etiquetas/classes $\mathcal{Y}$ e função de predição $f(x)$. A função de predição pode ser escrita, em termos probabilísticos, como $\mathcal{P}(Y|X)$.

Dado um domínio $\mathcal{D}_s$ e tarefa $\mathcal{T}_s$ de origem e um domínio $\mathcal{D}_t$ e tarefa $\mathcal{T}_t$ de destino, a transferência de aprendizagem procura melhorar a função de predição $f_t(x)$ a partir do conhecimento obtido do domínio e tarefa de origem se $\mathcal{D}_s \neq \mathcal{D}_t$ ou $\mathcal{T}_s \neq \mathcal{T}_t$.

A partir desta definição é possível concluir:

- Para situações em que os domínios e as tarefas são iguais ($\mathcal{D}_s = \mathcal{D}_t$ e $\mathcal{T}_s = \mathcal{T}_t$) então o problema é um caso tradicional de aprendizagem automática.

- A condição $\mathcal{D}_s \neq \mathcal{D}_t$ implica que ou o espaço de características não é partilhado entre os domínios ($\mathcal{X}_s \neq \mathcal{X}_t$) ou que a distribuição de probabilidade no espaço de características é diferente embora o espaço de características seja partilhado ($\mathcal{P}_s(X) \neq \mathcal{P}_t(X)$, $\mathcal{X}_s = \mathcal{X}_t$). Por exemplo na classificação de documentos podem surgir situações em que os termos são diferentes (texto sobre diferentes temas) ou termos iguais mas com uma diferente distribuição (textos acerca do mesmo tema mas escritos por diferentes indivíduos).

- A condição $\mathcal{T}_s \neq \mathcal{T}_t$ implica que ou o espaço de etiquetas não é partilhado entre as tarefas ($\mathcal{Y}_s \neq \mathcal{Y}_t$) ou a distribuição de probabilidades condicionadas é diferente ($\mathcal{P}_s(Y|X) \neq \mathcal{P}_t(Y|X)$. Por exemplo a transferência de aprendizagem de uma tarefa de origem com espaço de etiquetas binário para uma tarefa de destino com espaço de etiquetas multi-classe.

Cada uma destas condições requer um diferente tipo de transferência que será descrito de seguida.

4.2 Tipos de Transferência

O tipo de transferência é caracterizado pela relação entre ambos os domínios e entre ambas as tarefas.

A Tabela (4.1) apresenta os três diferentes tipos de transferência de aprendizagem: indutiva, transdutiva e não supervisionada, a ausência de transferência corresponde à situação tradicional.

Tipo Transferência	**Domínios de origem e destino**	**Tarefas de origem e destino**
Sem Transferência[2]	Iguais	Iguais
Indutiva	Iguais/Diferentes[3]	Diferentes
Não Supervisionada	Diferentes	Diferentes
Transdutiva	Diferentes	Iguais

Tabela 4.1: Tipos de Transferência de Aprendizagem

A transferência indutiva ocorre quando as tarefas de origem e destino são diferentes e é necessário que existam alguns dados etiquetados na tarefa de destino para que seja possível induzir um função de predição $f_t(Y|X)$ objetiva. É possível distinguir ainda situações em que os domínios são iguais, ou seja, existem dados

[2] Aprendizagem Automática tradicional.
[3] A transferência indutiva pode ser utilizada em ambos os casos.

etiquetados no domínio de origem e podem ser utilizados para melhorar a função de predição, ou os domínios são diferentes e não é possível tirar partido do domínio de origem diretamente.

A transferência transdutiva ocorre quando as tarefas de origem e destino são iguais mas os seus domínio diferentes, ou seja, não é possível tirar partido das características do domínio de origem sendo por isso necessário realizar uma adaptação de domínio.

A transferência não supervisionada é semelhante à transferência indutiva quando os domínios são diferentes exceto que o foco neste tipo de transferência é resolver tarefas não supervisionadas no domínio de destino como agrupamentos ou redução de dimensionalidade.

4.3 Abordagens de Transferência de Aprendizagem

Cada abordagem de transferência de aprendizam procura responder à questão "o que transferir".

Em transferência de aprendizagem existem quatro tipos de abordagens:

Transferência de instâncias - Esta abordagem assume que parte dos dados no domínio de origem podem ser reutilizados no domínio de destino desde que adaptados, esta adaptação ocorre através da alteração do peso das características ou amostragem da importância de cada característica nos diferentes domínios.

Transferência de representação de características - Esta abordagem assume que uma boa representação de características no domínio de origem pode ser partilhada com o domínio de destino permitindo assim que a aprendizagem seja mais eficiente.

Transferência de parâmetros - Esta abordagem assume que os modelos criados pelas tarefas de origem e destino partilham alguns parâmetros, desta forma a transferência destes parâmetros transfere o conhecimento obtido, codificado nos parâmetros.

Transferência relacional - Esta abordagem assume que existem relações entre os dados dos domínios de origem e destino que são semelhantes e que o conhecimento a transferir é essa relação.

Híbridas - Estas abordagens procuram integrar um conjunto de abordagens para otimizar o desempenho.

4.4 Conclusão

Neste capítulo definiu-se o conceito de transferência de aprendizagem, quais as principais questões relacionadas com a transferência de aprendizagem, os diferentes tipos de transferência de acordo com os domínios e tarefas associados e as abordagens possíveis.

Na realização do trabalho experimental é utilizada a abordagem transferência de parâmetros, esta abordagem permite assim transferir os pesos associados à rede neuronal profunda, pesos esses que codificam o conhecimento aprendido no contexto da tarefa de origem.

5

Arquitecturas de Rede Desenvolvidas

Neste capítulo é apresentada a arquitetura concebida para suportar a realização das redes neuronais implementadas na concretização experimental, incluindo a organização da biblioteca desenvolvida e a forma como é utilizada. São também apresentadas as duas arquiteturas de rede desenvolvidas para testes e como se encontram organizadas as suas camadas.

5.1 Arquitetura de Suporte às Redes Desenvolvidas

Para suportar a concretização das arquiteturas de rede definidas foi criada uma biblioteca que define uma arquitetura de suporte para permitir especificar a configuração e os parâmetros associados a cada arquitetura de rede, a definição de valores por omissão, a transferência de aprendizagem entre diferentes arquiteturas e a definição de ficheiros de teste que permitam executar o treino e transferência de aprendizagem das diferentes arquiteturas.
A Figura 5.1 apresenta os três módulos principais da biblioteca os quais são descritos de seguida.

Para além destes componentes, foram criados *scripts* que permitem a execução de um ou mais testes a partir da linha de comandos.

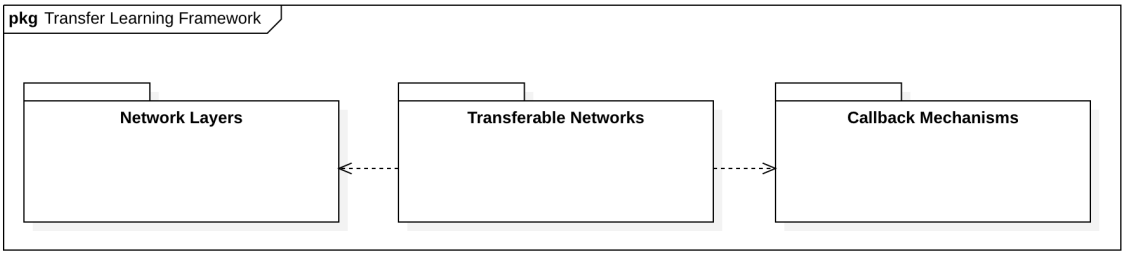


Figura 5.1: Diagrama de estrutura da biblioteca desenvolvida

5.1.1 Módulo *Transferable Networks*

A Figura 5.2 apresenta o detalhe deste módulo. O módulo *Transferable Networks* é responsável pela configuração e especificação de cada arquitetura e a transferência de aprendizagem.

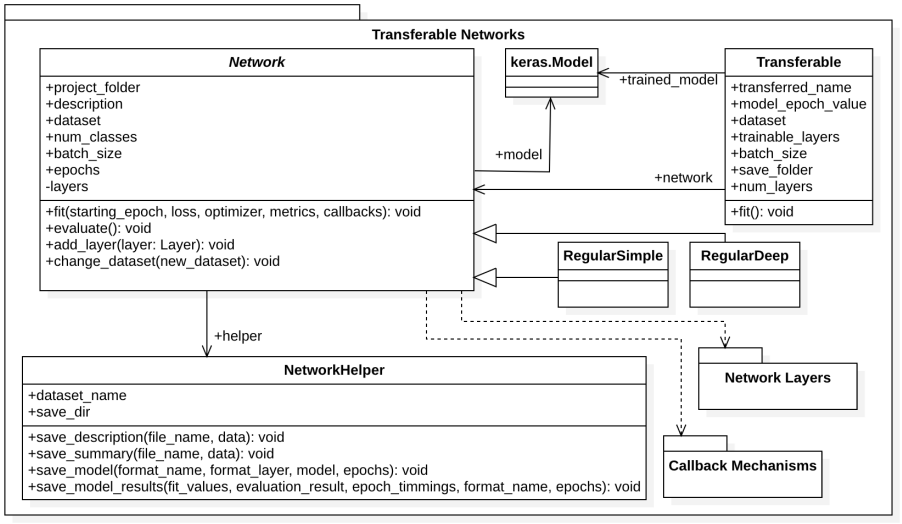


Figura 5.2: Diagrama de classes referente ao módulo *Transferable Networks*

A classe *Network*, serve de ponto de entrada para representar uma rede neuronal, permitindo especificar o conjunto de dados a ser utilizado, o número de iterações (*epochs*) a realizar durante o treino, a dimensão do *batch* a utilizar, o nome da rede, entre outras opções.
A classe *NetworkHelper* interage com o sistema de ficheiros para persistir dados relacionados com uma rede neuronal, como os resultados obtidos, ou carregar uma rede neuronal a partir de um ficheiro.
A classe *Transferable* permite realizar a transferência de aprendizagem entre duas redes neuronais diferentes através da transferência de parâmetros, utilizando um ficheiro de uma rede treinada previamente e transferindo os pesos para uma nova

rede.
As classes *RegularSimple* e *RegularDeep* representam a Arquitetura de Rede de 4 Camadas (Base) e a Arquitetura de Rede de 6 Camadas (*Deep*), respectivamente, estas especificam as suas camadas e a sua parametrização.

5.1.2 Módulo *Network Layers*

A Figura 5.3 apresenta o diagrama de classes deste módulo. O módulo *Network Layers* é responsável pela especificação das camadas utilizadas nas arquiteturas de rede.

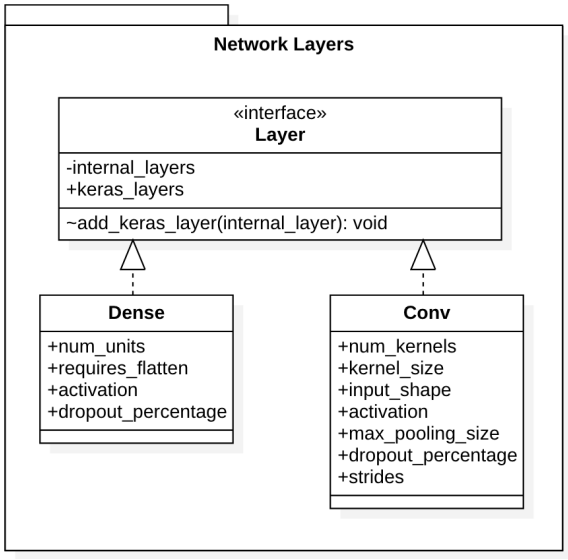


Figura 5.3: Diagrama de classes referente ao módulo *Network Layers*

Para simplificar a parametrização de camadas convolucionais e densas, foram criadas as classes *Dense* e *Conv*, visíveis na Figura 5.3, que representam camadas densas e convolucionais, respectivamente, permitindo especificar os parâmetros necessários como o *pooling* ou *padding* de uma camada convolucional, ou a dimensionalidade de uma camada densa. Todas as arquiteturas de rede desenvolvidas utilizam estas classes para especificar as suas camadas.

Cada camada convolucional é parametrizada por o número de filtros, a dimensão do filtro (a dimensão da matriz, quadrada e com valor ímpar para permitir centrar a matriz, por exemplo 3x3 ou 5x5). Para além disso, a cada camada convolucional pode ainda ser adicionado *max pooling*, uma técnica de redução de dimensionalidade que mantém apenas o valor máximo associado para uma matriz quadrada de tamanho **D** e *dropout*, uma técnica que permite reduzir a sobreparametrização

e aumentar a capacidade de generalização da rede através da remoção aleatória de valores na saída, simulando assim diferentes configurações para a mesma camada.

Cada camada densa é parametrizada por o número de neurónios, é também possível adicionar *dropout* na saída da camada. Cada arquitetura de rede possui uma camada de saída densa, em que o número de neurónios presente é o número de classes diferentes no conjunto de dados e a função de ativação é *softmax*.

5.1.3 Módulo *Callback Mechanisms*

A Figura 5.4 apresenta o diagrama de classes deste módulo. O módulo *Callback Mechanisms* é responsável pelo registo de informação relacionada com as redes implementadas, como o seu desempenho e tempo de treino.

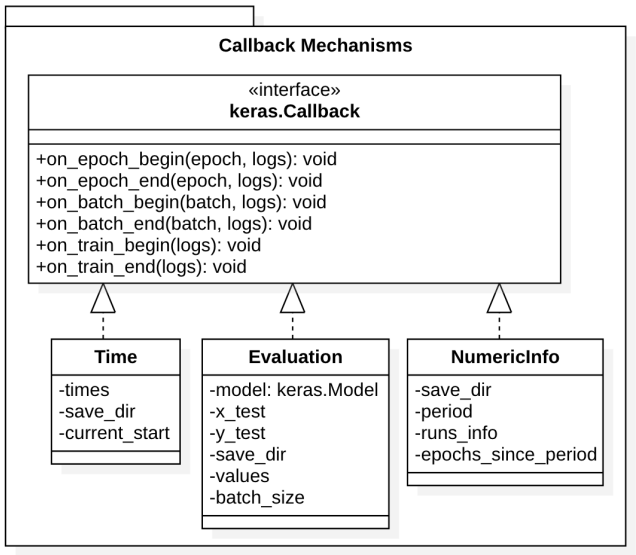


Figura 5.4: Diagrama de classes referente ao módulo *Callbacks Mechanisms*

O objectivo deste tipo de mecanismos é interceptar o treino de uma arquitetura em diferentes momentos. A classe *Time* é utilizada para registar o tempo utilizado para treinar cada *epoch*, a classe *NumericInfo* regista o desempenho da rede para um determinado conjunto de *epochs* e calcula valores como máximos e mínimos nesse conjunto e a classe *Evaluation* que avalia o desempenho da arquitetura no final de cada *epoch*. Todas estas classes persistem a sua informação para o sistema de ficheiros para permitir a avaliação da arquitetura após o treino.

5.2 Arquitetura de Rede de 4 Camadas (Base)

Esta arquitetura é composta por duas camadas convolucionais e duas camadas densas, com *kernels* pequenos, ideais para extrair características muito localizadas. Nas duas camadas convolucionais são utilizados mecanismos de *dropout* com o objectivo de reduzir a sobre-especialização destas camadas.

A primeira camada convolucional possui 32 filtros, *kernels* de dimensão 3x3, um *max pooling* de 2, ou seja, o resultado é reduzido para um quarto da dimensão dos dados de entrada e *dropout* de 20%. A segunda camada convolucional possui uma parametrização semelhante à primeira camada mas com 128 filtros. A primeira camada densa possui 256 unidades e a camada de saída possui 10 unidades.

A Figura 5.5 apresenta o diagrama desta arquitetura com as parametrizações para cada camada.

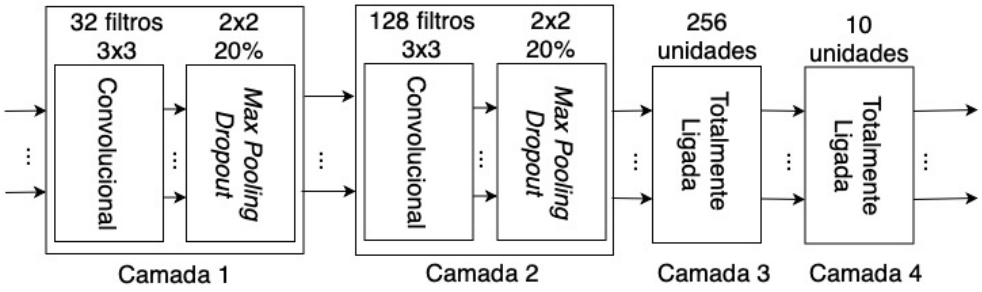


Figura 5.5: Diagrama da Arquitetura de Rede de 4 Camadas (Base)

Esta arquitetura foi desenvolvida para utilização com conjuntos de dados de menor complexidade e serve de base a todas as arquiteturas desenvolvidas. Para conjuntos de dados de maior complexidade foi desenvolvida a arquitetura descrita na secção seguinte.

5.3 Arquitetura de Rede de 6 Camadas (*Deep*)

Esta arquitetura é baseada na arquitetura apresentada em [52], com a remoção da normalização dos valores de saída de cada camada e o aumento no valor de *dropouts* na segunda e quarta camada de 20% para 25%, esta arquitetura foi escolhida pois permite estudar a transferência de aprendizagem em arquiteturas com maior nível de complexidade.

Esta arquitetura possui quatro camadas convolucionais e duas camadas densas. A primeira e a terceira camada utilizam o *padding same*, ao contrário da segunda e

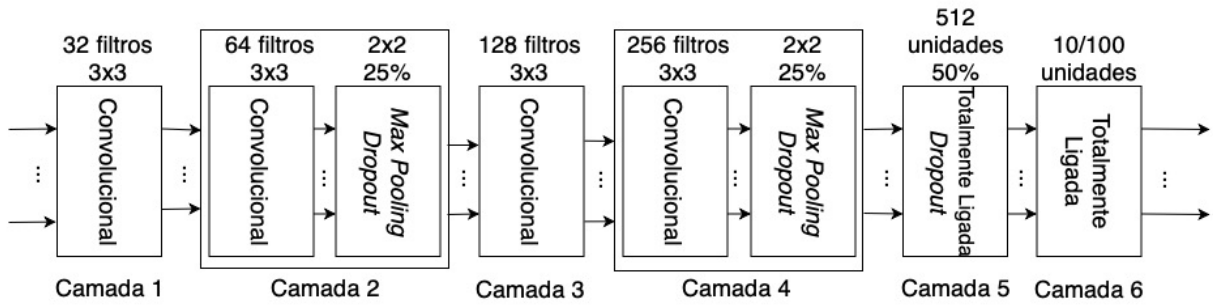


Figura 5.6: Diagrama da Arquitetura de Rede de 6 Camadas (*Deep*)

quarta camada que utilizam o *padding valid*, desta forma na primeira e terceira camada não existe redução de dimensionalidade dos dados, permitindo à segunda e quarta camada processar mais informação.

A Figura 5.6 apresenta o diagrama desta arquitetura de rede com as parametrizações para cada camada. A camada de saída (camada 6) possui 10 unidades para o conjunto de dados *Cifar10* e 100 unidades para o conjunto de dados *Cifar100*.

Esta arquitetura foi desenvolvida para conjuntos de dados mais complexos, que requerem um arquitetura mais profunda, para obter um melhor desempenho.

5.4 Conclusão

Neste capítulo apresentou-se a biblioteca criada que suporta as arquiteturas de rede desenvolvidas, com os seus três módulos principais: *Transferable Networks*, *Network Layers* e *Callback Mechanisms*, detalhou-se quais as classes que compõem cada módulo e o seu propósito.
Foram também apresentadas as duas arquiteturas de rede desenvolvidas, Arquitetura de Rede de 4 Camadas (Base) e Arquitetura de Rede de 6 Camadas (*Deep*), para utilização com conjuntos de dados de menor e maior complexidade, respectivamente.

6

Concretização Experimental

Neste capítulo é apresentado o contexto experimental, com uma descrição dos conjuntos de dados utilizados, o *software* e o *hardware* utilizados no ambiente de testes durante a concretização experimental e a análise comparativa dos resultados obtidos.

6.1 Contexto Experimental

O contexto experimental realizado teve por objectivo a transferência de aprendizagem do tipo indutivo, ou seja, as tarefas de origem e destino são consideradas diferentes, sendo a transferência de parâmetros a abordagem selecionada, descrita na secção Abordagens de Transferência de Aprendizagem. A transferência de parâmetros, neste caso os pesos de cada camada, é possível devido à utilização da mesma arquitetura de rede, garantindo assim que os parâmetros são partilhados entre as tarefas de origem e destino.

Os conjuntos de dados (*datasets*) selecionados são o *MNIST* [53], o *Fashion-MNIST* [54], o *Cifar10* [55] e o *Cifar100* [55].

O *dataset MNIST* é um dos mais antigos *datasets* utilizados, criado em 1998 por Yann LeCun [56], representa uma modificação do *dataset NIST*. Este *dataset* possui imagens de 28x28 pixeis a preto e branco de dígitos manuscritos, todas as imagens encontram-se centradas e o tamanho dos dígitos em cada imagem encontra-se normalizado, existem 60000 (sessenta mil) imagens de treino e 10000 (dez mil)

imagens de teste.

O *dataset Fashion-MNIST* foi criado com o objectivo de substituir o *dataset MNIST* devido ao elevado desempenho que redes convolucionais já conseguem obter. Este *dataset* utiliza imagens de 28x28 pixeis a preto e branco de roupa como saias e calçado, entre outros, existem 60000 (sessenta mil) imagens de treino e 10000 (dez mil) imagens de teste.

O *dataset Cifar10* é um *dataset* de imagens de 32x32 pixeis a cor, com 10 classes (avião, automóvel, pássaro, gato, veado, cão, sapo, cavalo, barco e camião), existem 50000 (ciquenta mil) imagens de treino e 10000 (dez mil) imagens de teste, criado a partir de um *dataset* de 80000000 (oitenta milhões) de imagens, em 2006, através da pesquisa de 53464 nomes, copiados do *Wordnet* nos motores de pesquisa, o *dataset* original já não se encontra disponível [57].

O *dataset Cifar100* é um *dataset* de imagens de 32x32 pixeis a cor, com 100 classes criado da mesma forma que o *dataset Cifar10*. Para cada classe existem 500 imagens de treino e 100 imagens de teste.

Estes *datasets* apresentam características muito semelhantes, o *dataset Fashion-MNIST* foi criado para substituir o *MNIST* e o *Cifar100* e *Cifar10* apresentam imagens com as mesmas dimensões apesar do diferente número de classes e imagens de treino/teste, permitindo assim testar a transferência de aprendizagem sem preocupações devido a diferentes formatos dos dados.

6.2 Ambiente de Testes

O ambiente de testes inclui o *hardware* especializado utilizado e o *software* desenvolvido para realizar a transferência de aprendizagem, tal como de seguida apresentado.

6.2.1 Hardware

O processo de treino foi realizado na placa *Jetson TX2*, apresentada na Figura 6.1. Esta placa possui um *GPU* Pascal com 256 núcleos, 2 processadores, um *ARM quad-core Cortex-A57* e um *ARM dual-core Nvidia Denver*, 8GB de *RAM* e 32GB de armazenamento, é também disponibilizado o *software Jetpack*, um kit de desenvolvimento de *software* que inclui uma versão modificada da distribuição *Linux Ubuntu* e todos os *drivers* e bibliotecas necessárias para tirar partido do *hardware*.

Figura 6.1: Placa de desenvolvimento *Nvidia Jetson TX2* [58]

Esta placa da *Nvidia* pode ser utilizada no desenvolvimento de aplicações de inteligência artificial com a capacidade de integrar sensores externos através de entradas *USB 3.0*, *Ethernet* e *PCIe* e realizar todo o processamento local, como apresentado na Figura 6.2.

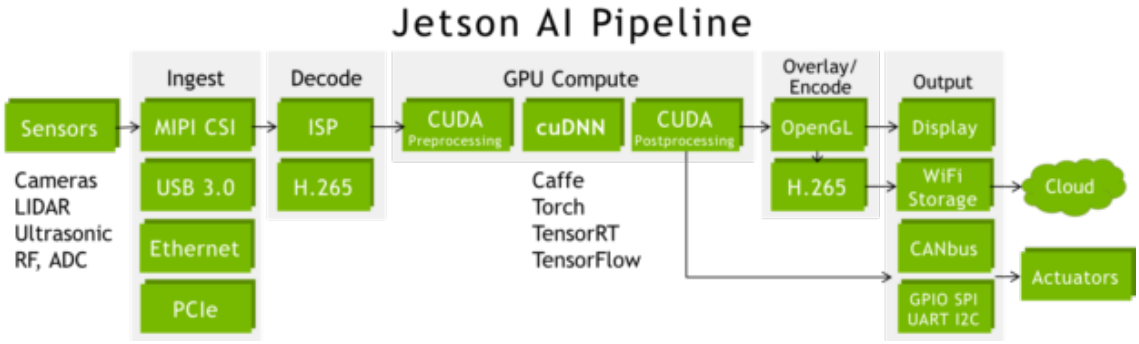


Figura 6.2: Encadeamento de processamento da placa *Nvidia Jetson TX2* [58]

Um dos exemplos em que esta placa é utilizada é a deteção do horizonte, realizada em drones, para garantir a estabilidade e evitar colisões com elementos no terreno como árvores ou elementos no ar como outros drones ou aves, a Figura 6.3 apresenta a deteção em tempo real do horizonte realizada numa placa *Nvidia Jetson TX2*.

Na concretização experimental realizada foi utilizada a versão 4.4[59] do *software Jetpack* que utiliza o *Linux Ubuntu* 18.04 LTS.

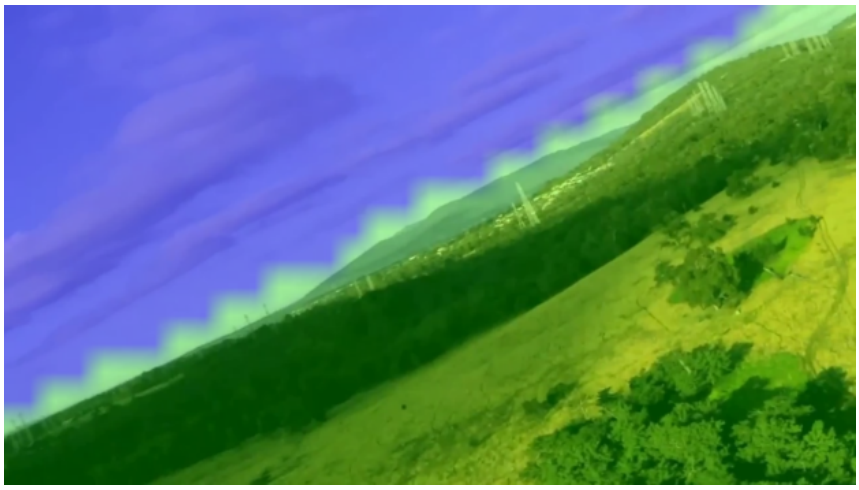


Figura 6.3: Deteção de horizonte através da placa *Nvidia Jetson TX2* [58]

6.2.2 Ambiente de Desenvolvimento

O ambiente de desenvolvimento escolhido foi um ambiente virtual *Python*, permitindo assim validar o *software* desenvolvido num computador e executar esse mesmo *software* no *Jetson TX2* sem problemas de dependências diferentes entre os equipamentos.

Para o desenvolvimento e treino dos modelos criados foi escolhida a plataforma *Keras* [60], que permite descrever a arquitetura de redes neuronais em código. Esta plataforma procura simplificar o desenvolvimento e treino de redes neuronais através da disponibilização de modelos pré-treinados, funções de pré-processamento dos dados, acesso a múltiplos conjuntos de dados prontos a utilizar e utilizar diferentes implementações de sistemas de redes neuronais como o *Tensorflow* ou o *Theano*.

Devido ao *hardware* utilizado, a implementação do sistema de redes neuronais é baseada na plataforma *Tensorflow*, uma versão modificada pela empresa *Nvidia*, que permite utilizar diretamente o *GPU* para executar os modelos desenvolvidos.

6.3 Resultados Obtidos

Todos os treinos foram executados com um *batch*, (ver secção Parametrização da Rede), de 128, os valores associados a cada imagem foram normalizados, a função de perda utilizada é a entropia cruzada entre categorias (*categorical crossentropy*), sendo utilizado algoritmo de otimização *RMSprop* (ver subsecção *Root Mean Square Propagation (RMSProp)*).

A definição dos valores de parametrização das redes implementadas é apresentada no anexo Anexo A - Configuração das Redes Desenvolvidas.

Na apresentação de resultados, os gráficos que apresentam no título "s/ aprend." correspondem a situações em que as camadas transferidas não participam na aprendizagem, os gráficos que apresentam no título "c/ aprend." correspondem a situações em que as camadas transferidas participam na aprendizagem.

6.3.1 Transferência *Fashion-MNIST* para *MNIST*

Nesta secção são apresentados os resultados da transferência de aprendizagem do *dataset Fashion-MNIST* para o *dataset MNIST*.

O *dataset Fashion-MNIST* é considerado mais complexo [61] que o *dataset MNIST* permitindo assim testar se a transferência permite manter o desempenho ao mesmo tempo que se reduz o tempo de treino.

Para esta transferência de aprendizagem foi realizado um treino no *dataset Fashion-MNIST* e os parâmetros obtidos durante o treino da arquitetura foram transferidos para o treino no *dataset MNIST*.

Desempenho da Arquitetura de Rede de 4 Camadas (Base) com Camadas Transferidas Fixas

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas não participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

Com a transferência de aprendizagem o tempo de treino, vísivel na Figura 6.4 e Tabela 6.1, diminui com o aumento do número de camadas transferidas, este comportamento é esperado pois as camadas transferidas não são incluídas durante a aprendizagem. É possível observar uma diminuição no tempo de treino considerável, cerca de 24% para apenas 1 camada transferida até cerca de 50% para 3 camadas transferidas, a diminuição do tempo de treino entre a transferência de duas e três camadas é de apenas 6%, seria expetável que a redução fosse superior pois a terceira camada é a que apresenta o maior número de parâmetros pois é totalmente ligada, é possível que o tempo de treino com três camadas

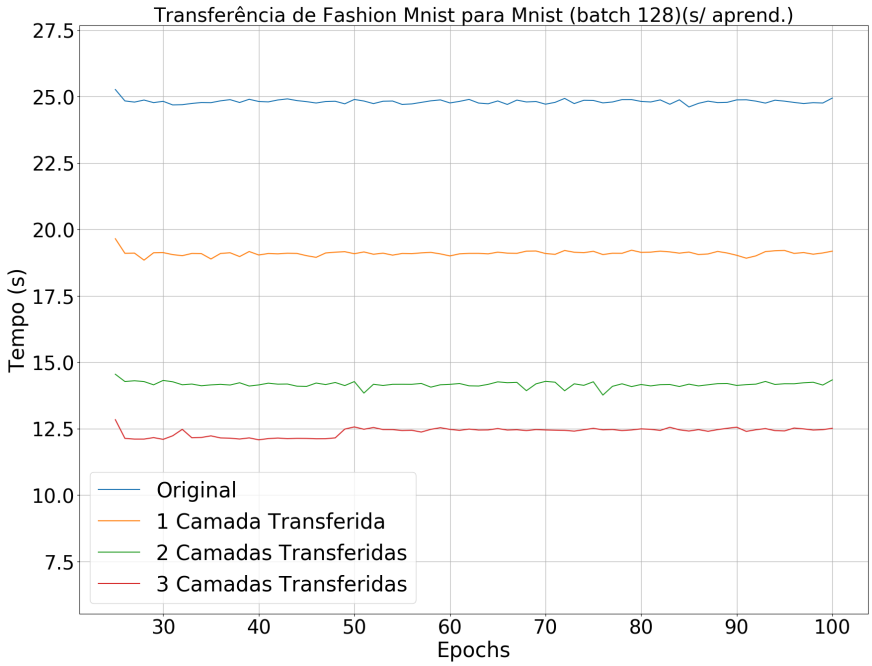


Figura 6.4: Tempo de treino por *epoch* na transferência de **Fashion-MNIST** para **MNIST** sem aprendizagem (apresentado como s/aprend.) das camadas transferidas

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença Tempo Treino Médio (s/*epoch*)
Sem transferência	≈25	0
1 Camada	≈19	≈-6 (-24%)
2 Camadas	≈14	≈-11 (-44%)
3 Camadas	≈12,5	≈-12,5 (-50%)

Tabela 6.1: Resultados do tempo de treino de cada *epoch* na transferência do **Fashion-MNIST** para **MNIST** sem aprendizagem nas camadas transferidas

transferidas seja o tempo mínimo necessário para realizar o treino de arquitetura, o que poderia explicar a diferença de apenas 6%.

Com a transferência de aprendizagem a taxa de acerto dos modelos, vísivel na Figura 6.5 e Tabela 6.2, é semelhante à taxa de acerto sem transferência, com a exceção da transferência de três camadas que apresenta uma queda de desempenho de cerca 3,9%, em média, em relação ao modelo sem transferência. A capacidade de manter e até melhorar ligeiramente o desempenho indica que as características aprendidas no *dataset Fashion-MNIST* são apropriadas para o *dataset MNIST*, mesmo na situação em que as três camadas são transferidas a perda de desempenho é aceitável tendo em conta que o tempo de treino é metade do original.

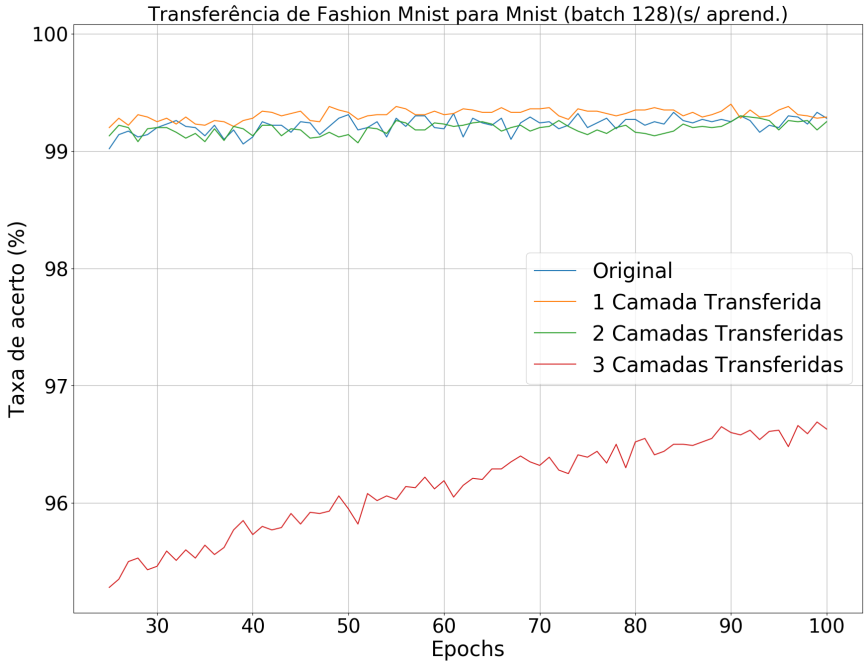


Figura 6.5: Taxa de acerto da classificação com transferência de **Fashion-MNIST** para **MNIST** sem aprendizagem das camadas transferidas

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	99,3	99	0
1 Camada	99,4	99,1	≈+0,1
2 Camadas	99,3	99	≈0
3 Camadas	96,7	95,1	≈-3,9

Tabela 6.2: Resultados da taxa de acerto na transferência do **Fashion-MNIST** para **MNIST** sem aprendizagem nas camadas transferidas

Desempenho da Arquitetura de Rede de 4 Camadas (Base) com Camadas Transferidas que Aprendem

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

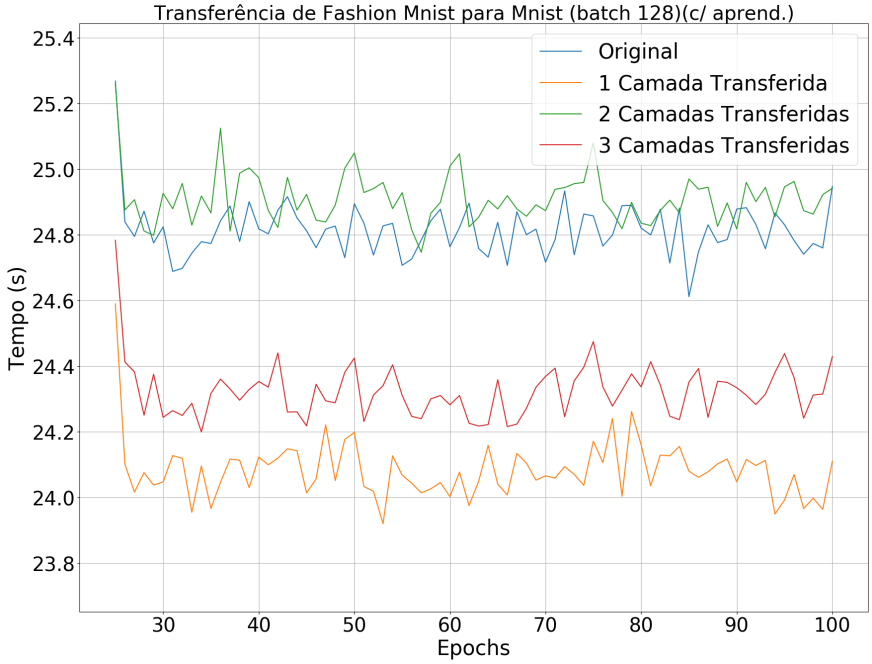


Figura 6.6: Tempo de treino por *epoch* na transferência de **Fashion-MNIST** para **MNIST** com aprendizagem das camadas transferidas

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença T. Treino Médio (s/*epoch*)
Sem transferência	≈25	0
1 Camada	≈24,5	≈-0,5 (-2%)
2 Camadas	≈25	≈0
3 Camadas	≈24,5	≈-0,5 (-2%)

Tabela 6.3: Resultados do tempo de treino de cada *epoch* na transferência do **Fashion-MNIST** para **MNIST** com aprendizagem nas camadas transferidas

Com a transferência o tempo de treino, vísivel na Figura 6.6 e Tabela 6.3, diminui cerca de 2% para uma e três camadas e não existe qualquer redução para duas camadas transferidas, estas pequenas variações são esperadas pois as camadas

transferidas participam no processo de aprendizagem, o que requer que todo o processo de retropropagação e actualização dos pesos seja realizado.

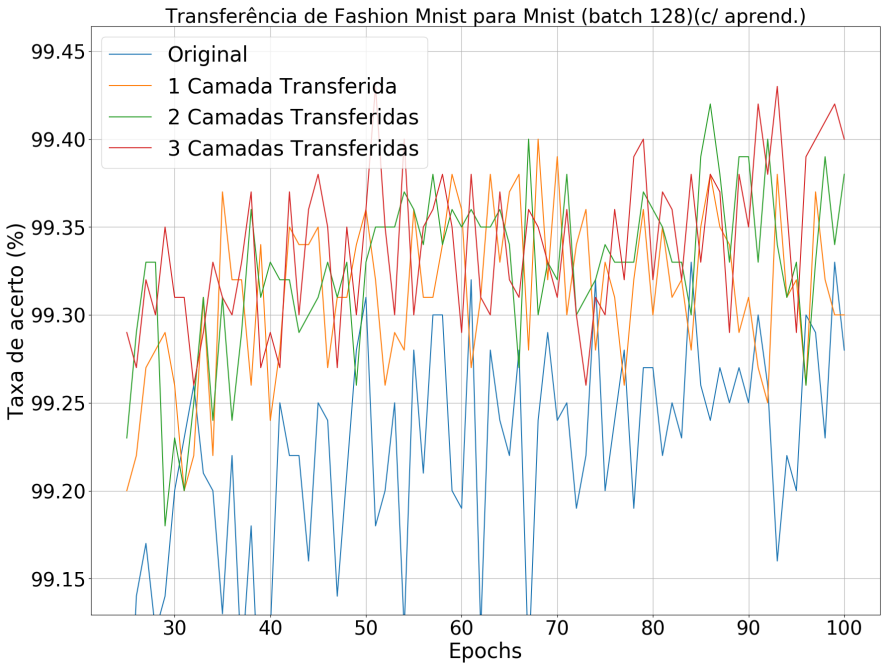


Figura 6.7: Transferência de **Fashion-MNIST** para **MNIST** com aprendizagem das camadas transferidas

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	99,3	99	0
1 Camada	99,4	99,1	≈+0,1
2 Camadas	99,4	99,2	≈+0,2
3 Camadas	99,4	99,2	≈+0,2

Tabela 6.4: Resultados da taxa de acerto na transferência do **Fashion-MNIST** para **MNIST** com aprendizagem nas camadas transferidas

Com a transferência de aprendizagem a taxa de acerto dos modelos, vísivel na Figura 6.7 e Tabela 6.4, é superior, em média, ao do modelo sem transferência, o que é esperado, pois já se verificou que para camadas fixas as características aprendidas são úteis, servindo como um melhor ponto inicial para o modelo.

Síntese de Resultados

A transferência de aprendizagem para esta situação foi bem sucedida, quer quando as camadas transferidas aprendem quer quando não aprendem. Foi possível reduzir o tempo de treino em quase todos os casos, mantendo a mesma taxa de acerto na classificação.

6.3.2 Transferência *MNIST* para *Fashion-MNIST*

Esta secção contém os resultados da transferência de aprendizagem do *dataset MNIST* para o *dataset Fashion-MNIST*.

O *dataset MNIST* é considerado menos complexo [61] que o *dataset Fashion-MNIST* permitindo assim testar se transferência de conjunto de dados mais simples pode ser útil para conjuntos de dados mais complexos.

Para esta transferência foi realizado um treino no *dataset MNIST* e os parâmetros obtidos durante o treino da arquitetura foram transferidos para o treino no *dataset Fashion-MNIST*.

Desempenho da Arquitetura de Rede de 4 Camadas (Base) com Camadas Transferidas Fixas

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas não participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença T. Treino Médio (s/*epoch*)
Sem transferência	≈24,8	0
1 Camada	≈19	≈-5,8 (-23%)
2 Camadas	≈14,9	≈-9,9 (-40%)
3 Camadas	≈12,5	≈-12,3 (-50%)

Tabela 6.5: Resultados do tempo de treino de cada *epoch* na transferência do **MNIST** para **Fashion-MNIST** sem aprendizagem nas camadas transferidas

Com a transferência de aprendizagem o tempo de treino, visível na Figura 6.8 e Tabela 6.5, diminui com o aumento do número de camadas, este comportamento

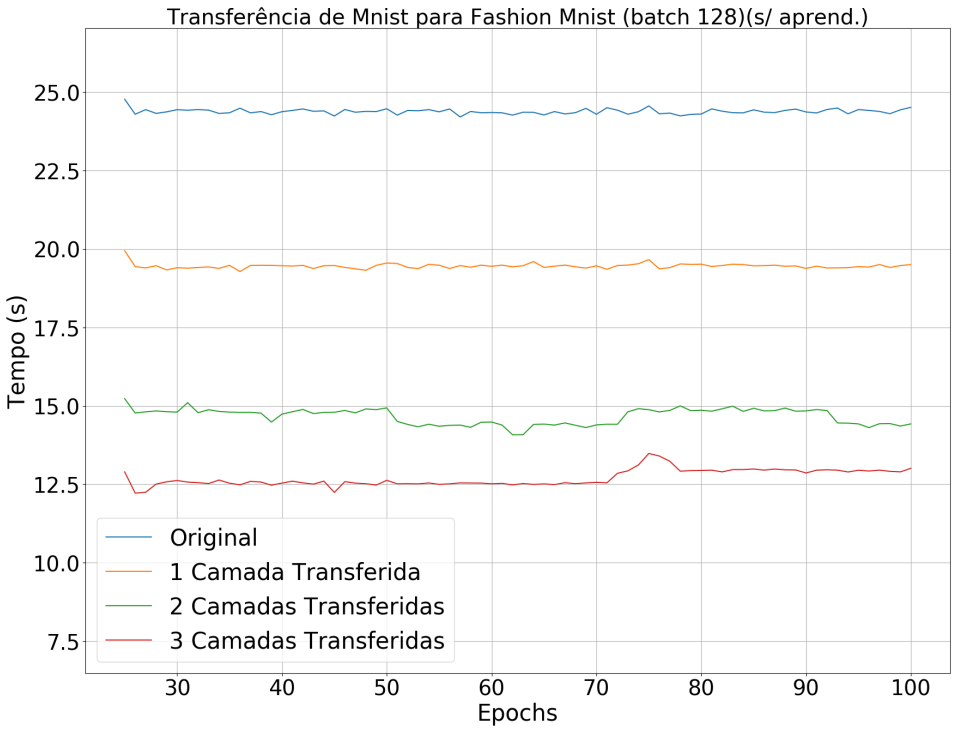


Figura 6.8: Tempo de treino por *epoch* na transferência de **MNIST** para **Fashion-MNIST** sem aprendizagem das camadas transferidas

é esperado pois as camadas transferidas não são incluídas durante a aprendizagem. A diminuição do tempo de treino entre a transferência de duas e três camadas é de 10%, seria expetável que a redução fosse semelhante à redução entre uma e duas camadas transferidas (17%) pois a terceira camada é densa e possui um custo computacional mais elevado, contudo é possível que este seja o tempo mínimo necessário para treinar a rede, não sendo assim possível reduzir mais o tempo de treino.

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	92,8	90,7	0
1 Camada	91,8	90,2	≈-0,5
2 Camadas	90,5	89,3	≈-1,4
3 Camadas	83,9	80,9	≈-9,8

Tabela 6.6: Resultados da taxa de acerto na transferência do **MNIST** para **Fashion-MNIST** sem aprendizagem nas camadas transferidas

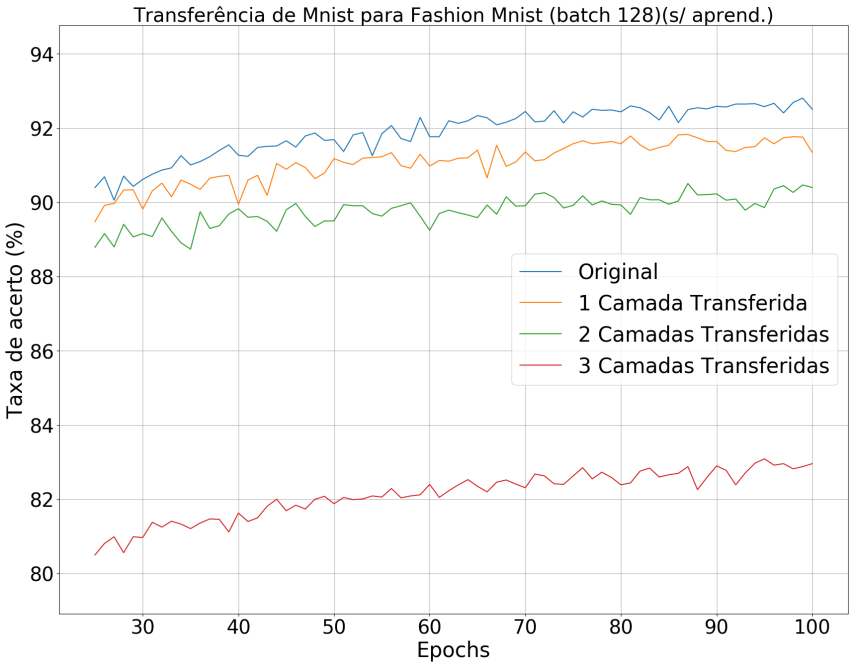


Figura 6.9: Transferência de **MNIST** para **Fashion-MNIST** sem aprendizagem das camadas transferidas

Com a transferência de aprendizagem a taxa de acerto dos modelos transferidos, vísivel na Figura 6.9 e Tabela 6.6, não consegue igualar o desempenho do modelo sem transferência, contudo a perda de desempenho é reduzida, especialmente para uma e duas camadas transferidas, sendo possível obter cerca de 98,5% do desempenho original em apenas 60% do tempo de treino. Apesar de ser considerado um *dataset* com menor complexidade e por isso mais fácil, as características aprendidas no *dataset MNIST* e transferidas para o *dataset Fashion-MNIST* aparentam ser suficientemente abrangentes para obter um desempenho semelhante ao modelo sem transferência, contudo os modelos transferidos nunca conseguiram obter ou ultrapassar o desempenho do modelo sem transferência.

Desempenho da Arquitetura de Rede de 4 Camadas (Base) com Camadas Transferidas que Aprendem

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

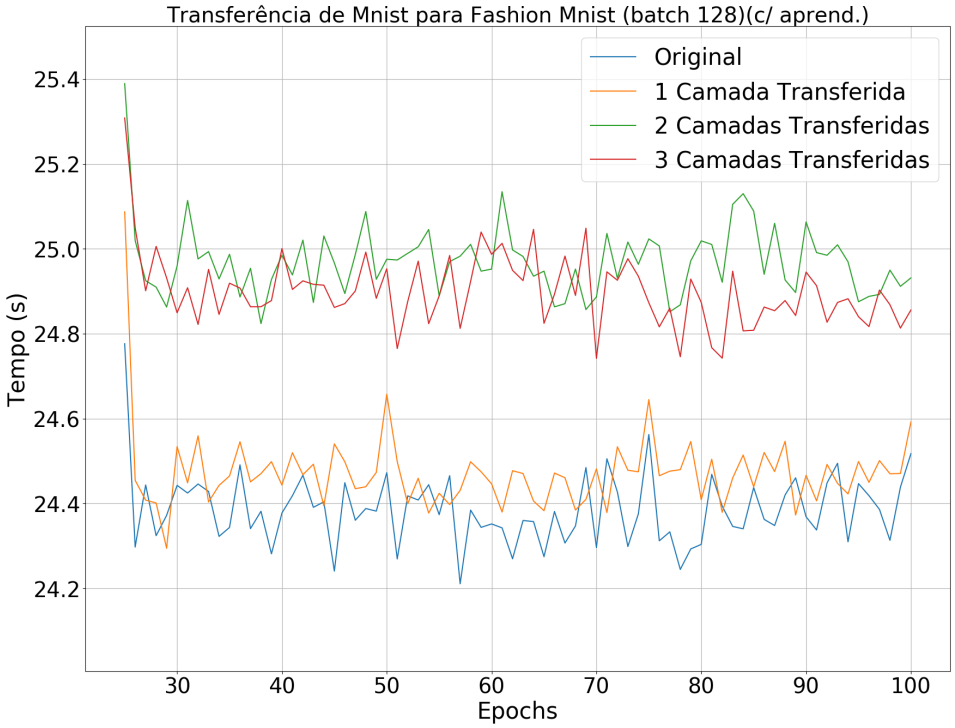


Figura 6.10: Tempo de treino por *epoch* na transferência de **MNIST** para **Fashion-MNIST** com aprendizagem das camadas transferidas

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença T. Treino Médio (s/*epoch*)
Sem transferência	≈24,4	0
1 Camada	≈24,5	≈+0,1 (+0,4%)
2 Camadas	≈24,9	≈+0,5 (+2%)
3 Camadas	≈24,9	≈+0,5 (+2%)

Tabela 6.7: Resultados do tempo de treino de cada *epoch* na transferência do **MNIST** para **Fashion-MNIST** com aprendizagem nas camadas transferidas

Com a transferência de aprendizagem o tempo de treino, visível na Figura 6.10 e Tabela 6.7, aumentou ligeiramente com o número de camadas transferidas. Embora seja expectável que existam algumas variações no tempo de treino, como ocorre com uma camada transferida, para duas e três camadas transferidas existe um aumento de 2% no tempo de treino, é possível que os valores transferidos dificultem a aprendizagem sendo necessário mais tempo devido a possíveis interdependências entre as diferentes camadas.

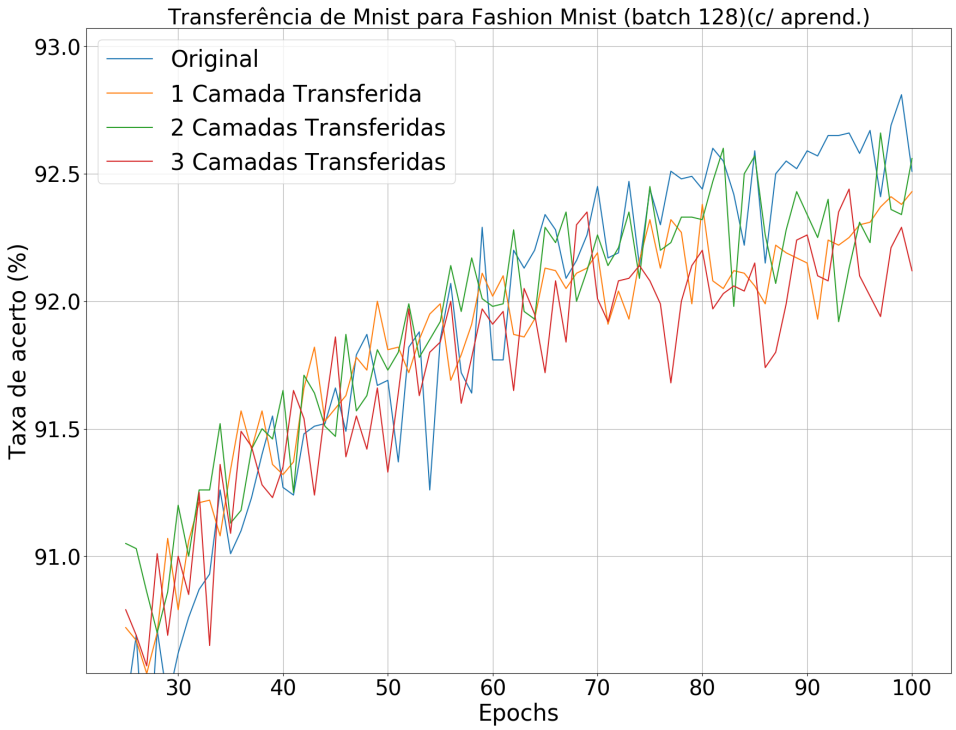


Figura 6.11: Transferência de **MNIST** para **Fashion-MNIST** com aprendizagem das camadas transferidas

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	92,8	90,7	0
1 Camada	92,4	90,9	≈+0,2
2 Camadas	92,6	91,3	≈+0,6
3 Camadas	92,4	91,1	≈+0,4

Tabela 6.8: Resultados da taxa de acerto na transferência do **MNIST** para **Fashion-MNIST** com aprendizagem nas camadas transferidas

Com a transferência de aprendizagem a taxa de acerto dos modelos transferidos, vísivel na Figura 6.11 e Tabela 6.8, é ligeiramente superior, em média, à taxa de

acerto do modelo sem transferência, contudo esta vantagem parece diminuir/-desaparecer com o aumento do número de *epochs*. Os modelos com transferência conseguem obter melhor desempenho com menos iterações, possivelmente tirando partido das características transferidas, contudo essas características parecem impedir esses modelos de se especializarem para o *dataset Fashion-MNIST*.

Síntese de Resultados

A transferência de aprendizagem para esta situação foi bem sucedida, para o caso de camadas transferidas que não participam na aprendizagem, obteve-se uma redução no tempo de treino com uma ligeira redução no desempenho de classificação, para o caso de camadas transferidas que participam na aprendizagem verifica-se um ligeiro aumento no tempo de treino para um ligeiro aumento de desempenho. Ao contrário do verificado na Transferência *Fashion-MNIST* para *MNIST* nenhum dos modelos transferidos consegue igualar a taxa de acerto do modelo sem transferência, com e sem aprendizagem das camadas transferidas, possivelmente devido à menor complexidade do *dataset MNIST* comparativamente ao *dataset Fashion-MNIST*.

6.3.3 Transferência *Cifar100* para *Cifar10*

Nesta secção são apresentados os resultados da transferência de aprendizagem do *dataset Cifar100* para o *dataset Cifar10* na Arquitetura de Rede de 6 Camadas (*Deep*).

O *dataset Cifar100* contém dez vezes mais classes com uma maior variabilidade de dados, apesar de conter menos exemplos de cada, permitindo assim verificar se a obtenção de um conjunto de características mais variadas é útil para o contexto de um *dataset* mais reduzido.

Desempenho da Arquitetura de Rede de 6 Camadas (*Deep*) com Camadas Transferidas Fixas

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas não participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

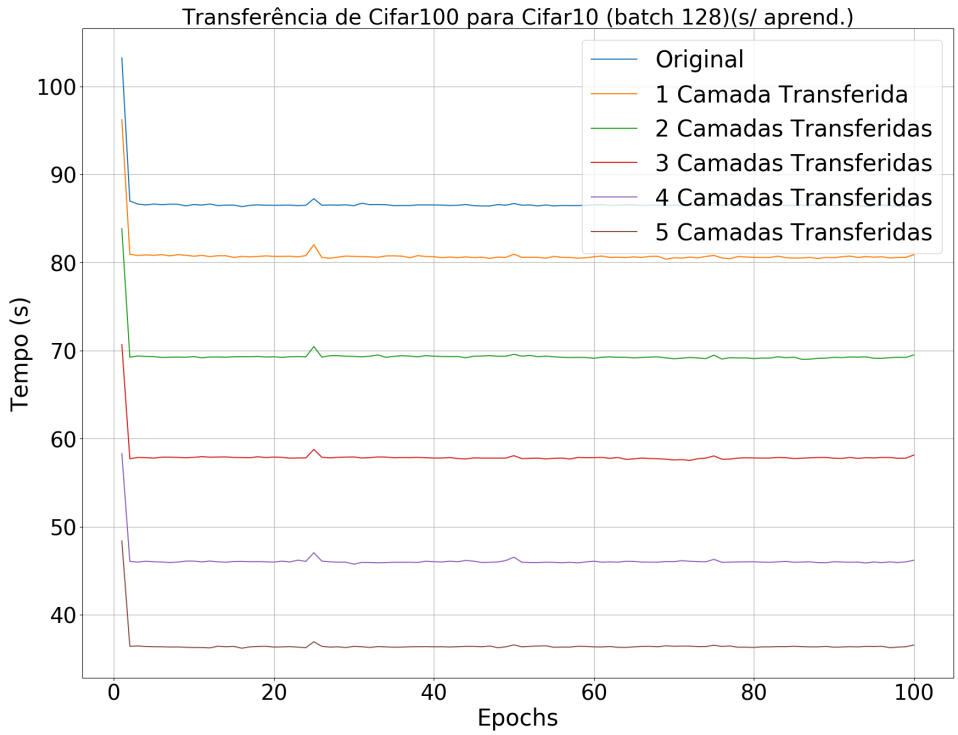


Figura 6.12: Tempo de treino por *epoch* na transferência de **Cifar100** para **Cifar10** sem aprendizagem das camadas transferidas

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença T. Treino Médio (s/*epoch*)
Sem transferência	≈87	0
1 Camada	≈80,7	≈-6,3 (-7%)
2 Camadas	≈69,2	≈-17,8 (-20%)
3 Camadas	≈57,8	≈-29,2 (-33%)
4 Camadas	≈46	≈-41 (-47%)
5 Camadas	≈36,3	≈-50,7 (-58%)

Tabela 6.9: Resultados do tempo de treino de cada *epoch* na transferência do **Cifar100** para **Cifar10** sem aprendizagem nas camadas transferidas

Com a transferência de aprendizagem o tempo de treino, vísivel na Figura 6.12 e Tabela 6.9, é reduzido em todos os casos, a diferença por camada transferida é, aproximadamente, 13%, este comportamento é o esperado pois as camadas transferidas não participam na aprendizagem.

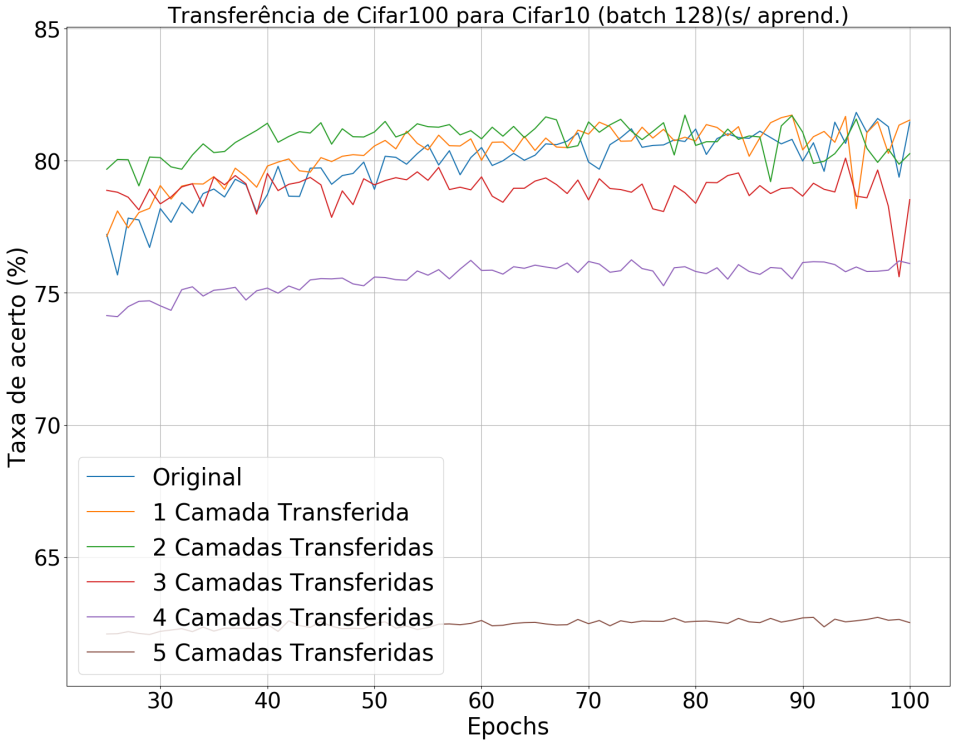


Figura 6.13: Transferência de **Cifar100** para **Cifar10** sem aprendizagem das camadas transferidas

Com a transferência de aprendizagem a taxa de acerto dos modelos transferidos, vísivel na Figura 6.13 e Tabela 6.10, possibilita em alguns modelos, obter

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	81,8	76,8	0
1 Camada	81,7	77,6	≈+0,8
2 Camadas	81,7	79,2	≈+2,4
3 Camadas	80,1	77,9	≈+1,1
4 Camadas	76,3	74,3	≈-2,5
5 Camadas	62,7	61,1	≈-15,7

Tabela 6.10: Resultados da taxa de acerto na transferência do **Cifar100** para **Cifar10** sem aprendizagem nas camadas transferidas

um melhor desempenho que o modelo sem transferência, o que indica que as características obtidas no *dataset Cifar100* são úteis, o mesmo não acontece para a transferência de quatro e cinco camadas, estas camadas possivelmente possuem características demasiado específicas para o *dataset Cifar100* que não são indicadas para o *dataset Cifar10*.

Desempenho da Arquitetura de Rede de 6 Camadas (*Deep*) com Camadas Transferidas que Aprendem

Nesta concretização experimental foi implementada uma transferência de aprendizagem através da transferência dos pesos das camadas, na qual as camadas transferidas participam na aprendizagem, permitindo assim verificar se a transferência produz uma redução no tempo de treino e/ou um aumento na taxa de acerto da classificação.

Nº Transferidas	T. Treino Médio (s/*epoch*)	Diferença T. Treino Médio (s/*epoch*)
Sem transferência	≈87	0
1 Camada	≈84,2	≈-2,8 (-3,2%)
2 Camadas	≈86,4	≈-0,6 (-0,7%)
3 Camadas	≈84,9	≈-2,1 (-2,4%)
4 Camadas	≈84,6	≈-2,4 (-2,75%)
5 Camadas	≈85	≈-2 (-2,3%)

Tabela 6.11: Resultados do tempo de treino de cada *epoch* na transferência do **Cifar100** para **Cifar10** com aprendizagem nas camadas transferidas

Com a transferência de aprendizagem o tempo de treino, vísivel na Figura 6.14 e Tabela 6.11, é inferior para todas as transferências. Seria expectável que o tempo

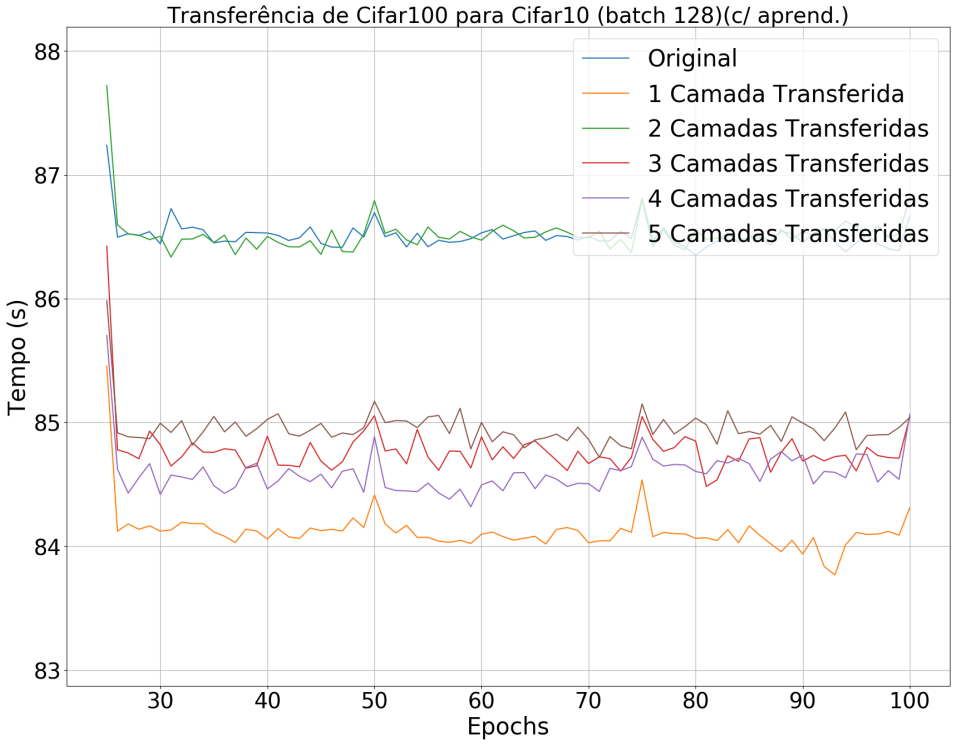


Figura 6.14: Tempo de treino por *epoch* na transferência de **Cifar100** para **Cifar10** com aprendizagem das camadas transferidas

de treino fosse semelhante ao modelo sem transferência de aprendizagem, devido ao facto de as camadas transferidas participarem na aprendizagem, com pequenas variações. É possível que as camadas transferidas, devido às suas características, facilitem, de alguma forma, o processo de aprendizagem permitindo assim reduzir o tempo de treino.

Nº Transferidas	Taxa de Acerto Máxima (%)	Taxa de Acerto Média (%)	Diferença Média (%)
Sem transferência	81,8	76,8	0
1 Camada	81,9	78,3	≈+1,5
2 Camadas	83	79,8	≈+3
3 Camadas	82,5	79,9	≈+3,1
4 Camadas	81,4	79,6	≈+2,8
5 Camadas	81,5	78,4	≈+1,6

Tabela 6.12: Resultados da taxa de acerto na transferência do **Cifar100** para **Cifar10** com aprendizagem nas camadas transferidas

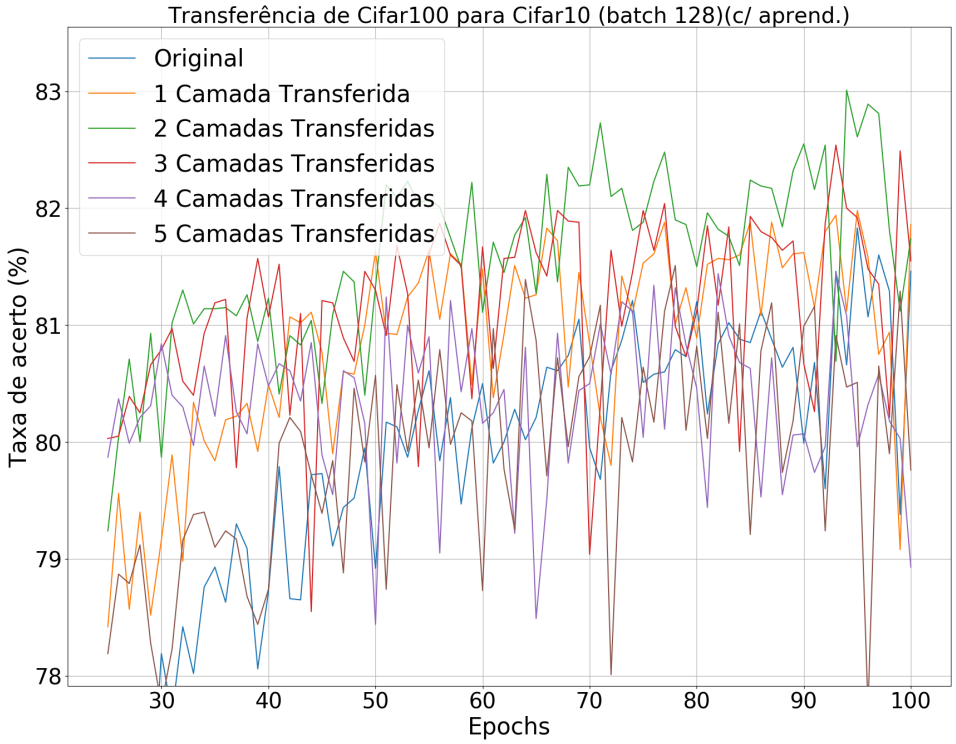


Figura 6.15: Transferência de **Cifar100** para **Cifar10** com aprendizagem das camadas transferidas

Com a transferência de aprendizagem a taxa de acerto dos modelos transferidos, visível na Figura 6.15 e Tabela 6.12, é, em média, superior à taxa de acerto do modelo sem transferência, indicando assim que as características transferidas continuam a ser úteis mesmo que as camadas transferidas participem na aprendizagem.

Síntese de Resultados

A transferência de aprendizagem foi bem sucedida nesta situação, foi possível reduzir o tempo de treino em todos os casos. Para a transferência com camadas que não participam na aprendizagem foi possível verificar que algumas camadas transferidas conseguem igualar/melhorar o desempenho do modelo sem transferência, contudo as camadas transferidas mais profundas não conseguem igualar o desempenho, indicando que as características aprendidas por essas camadas são características especializadas. Para a transferência com camadas que participam na aprendizagem, foi possível verificar que as características transferidas continuam a ser úteis no novo *dataset*, independentemente do número de camadas transferidas.

6.4 Conclusões Obtidas

Com os resultados obtidos na transferência de aprendizagem entre o *dataset Fashion-MNIST* e o *dataset MNIST* e o *dataset Cifar100* e o *dataset Cifar10*, é possível concluir que a transferência de *datasets* mais complexos ou com informação mais variada para *datasets* menos complexos permite obter um desempenho semelhante ou melhor que o treino sem transferência, quer as camadas transferidas participem na aprendizagem no modelo transferido ou não.

Com os resultados obtidos na transferência de aprendizagem entre o *dataset MNIST* e o *dataset Fashion-MNIST* é possível concluir que o desempenho, na transferência de *datasets* menos complexos para *datasets* mais complexos, pode apenas aproximar o desempenho do modelo sem transferência, devido à falta de características suficientemente abrangentes no *dataset* menos complexo.

O aumento do número de camadas transferidas diminui o desempenho obtido, principalmente para situações em que as camadas transferidas não participam na aprendizagem, o que indica que as camadas iniciais aprendem características mais genéricas comparativamente a camadas posteriores, que aprendem características mais específicas ao *dataset* em concreto.

A transferência de aprendizagem permite reduzir o tempo de treino quando as camadas transferidas não participam na aprendizagem, para situações em que participem na aprendizagem, o tempo de treino poderá ser ligeiramente melhor ou pior dependendo de quão apropriados sejam os valores nas camadas transferidas.

A transferência de aprendizagem permitiu verificar duas situações, em relação ao desempenho, na taxa de acerto da classificação dos modelos: se o *dataset* de origem apresentar uma maior complexidade comparativamente ao *dataset* de destino, então o desempenho do modelo transferido poderá ser igual e/ou melhor que o modelo sem transferência, se o *dataset* de origem apresentar uma menor complexidade comparativamente ao *dataset* de destino então, para camadas transferidas que não participem na aprendizagem o modelo transferido poderá possuir um desempenho próximo do modelo sem transferência, para camadas transferidas que participem na aprendizagem, o modelo transferido poderá possuir um desempenho semelhante ao modelo sem transferência.

7

Conclusão

Desde a introdução de modelos de aprendizagem automática que existe a necessidade de informação (dados de treino) para que a aprendizagem possa ser realizada, sendo a obtenção desta informação uma das maiores dificuldades, principalmente em situações de aprendizagem supervisionada que requer que a informação se encontre categorizada/etiquetada.

Com a adopção de modelos de aprendizagem automática, principalmente modelos baseados em redes neuronais artificiais, pela sociedade em geral, a capacidade de obter mais e melhor informação para treinar estes modelos obtém ainda maior importância, contudo, a capacidade de obter esta informação poderá não estar disponível devido a limitações de diversos tipos.

A transferência de aprendizagem procura tirar partido de conhecimento adquirido previamente e reutilizá-lo, permitindo assim utilizar a informação disponível em novos contextos.

No decurso do trabalho realizado, apresentaram-se os conceitos principais de redes neuronais artificiais, de transferência de aprendizagem e como esta se aplica em redes neuronais artificiais.

Realizou-se a implementação [62] de algumas redes neuronais convolucionais e a transferência de aprendizagem entre elas, o que permitiu verificar as situações em que a transferência apresenta vantagens, cumprindo assim com os objectivos de apresentar o conceito de transferência de aprendizagem e demonstrar o seu funcionamento.

Verificou-se que a transferência de aprendizagem em redes convolucionais reduz o tempo de treino sempre que as camadas transferidas não participem no processo de aprendizagem, para situações em que as camadas transferidas participem no processo de aprendizagem o tempo de treino será semelhante ou superior ao modelo sem transferência.

A transferência de aprendizagem de conjuntos de dados mais complexos para menos complexos permite obter melhor desempenho, o que não se verifica na situação contrária, ou seja, de conjuntos de dados menos complexos para mais complexos, onde apenas é possível obter um desempenho semelhante.

O aumento do número de camadas transferidas poderá reduzir o desempenho, indicando que as camadas transferidas mais profundas transferem conhecimento específico e não conhecimento geral, o número de camadas a transferir é dependente de cada modelo mas, regra geral, camadas menos profundas possuem conhecimento genérico que será transferível e camadas mais profundas possuem conhecimento específico do conjunto de dados de treino que não será transferível, ou seja, a sua transferência irá prejudicar o desempenho do modelo.

7.1 Trabalho Futuro

O trabalho efetuado permitiu demonstrar os benefícios que a utilização de transferência de aprendizagem poderá trazer na utilização de redes neuronais profundas envolvendo camadas convolucionais, contudo, existem situações e questões relevantes para trabalho futuro, algumas destas são apresentadas de seguida:

- A possibilidade de introduzir um mecanismo que permita determinar ou estimar, sem a necessidade de realizar a transferência de aprendizagem, se a transferência de aprendizagem será útil;
- O desenvolvimento de uma plataforma, ou extensão de uma plataforma existente, que permita rapidamente testar a transferência de aprendizagem de modelos com arquiteturas heterogéneas. Neste momento já é possível utilizar modelos pré-treinados com camadas fixas.

7.2 Considerações Finais

A utilização de redes neuronais artificiais tem vindo a crescer exponencialmente na última década, com a expectativa de que a sua utilização irá abranger cada vez

mais processos na sociedade.

Este trabalho serviu para apresentar o conceito de transferência de aprendizagem para redes neuronais artificiais, demonstrar a sua utilização correcta e sugerir possíveis linhas de investigação futura que possam expandir a utilização desta.

Bibliografia

[1] Norbert Wiener. *Cybernetics or Control and Communication in the Animal and the Machine*. 2019. (p. 5)

[2] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943. (p. 5)

[3] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. (p. 5)

[4] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961. (p. 5)

[5] Mikel Olazaran. A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26(3):611–659, 1996. (p. 6)

[6] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960. (p. 6)

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 15. MIT Press, 2016. http://www.deeplearningbook.org. (p. 6)

[8] Frank Rosenblatt. New navy device learns by doing; psychologist shows embryo of computer designed to read and grow wiser. 1958. https://www.nytimes.com/1958/07/08/archives/new-navy-device-learns-by-doing-psychologist-shows-embryo-of.html?searchResultPosition=1. (p. 6)

[9] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. 2017. (p. 6)

[10] Kunihiko Fukushima. Cognitron: A self-organizing multilayered neural network. *Biological cybernetics*, 20(3-4):121–136, 1975. (p. 6)

[11] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. (p. 6)

[12] Rumelhart , David E, James Mcclelland, and James L. *Parallel distributed processing: explorations in the microstructure of cognition. Volume 1. Foundations*. 01 1986. (p. 6)

[13] Geoffrey E Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007. (p. 7)

[14] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007. (p. 7)

[15] Marc Ranzato, Christopher Poultney, Sumit Chopra, Yann LeCun, et al. Efficient learning of sparse representations with an energy-based model. *Advances in neural information processing systems*, 19:1137, 2007. (p. 7)

[16] Wikipedia. Neuron. https://en.wikipedia.org/wiki/Neuron, 2019. Acedido: 21-06-2019. (p. 8)

[17] Tony Owen. Artificial intelligence by patrick henry winston addison-wesley publishing company, massachusetts, usa, july 1984 (£ 18.95. *Robotica*, 6(2): 165–165, 1988. (pp. 9 e 10)

[18] Jean-Christophe B. Loiseau. Rosenblatt's perceptron, the first modern neural network. https://towardsdatascience.com/rosenblatts-perceptron-the-very-first-neural-network-37a3ec09038a, 2019. Acedido: 23-06-2019. (p. 11)

[19] David S Touretzky and Dean A Pomerleau. What's hidden in the hidden layers. *Byte*, 14(8):227–233, 1989. (p. 12)

[20] Graham Kendall. G5aiai - introduction to artificial intelligence. http://www.cs.nott.ac.uk/~pszgxk/courses/g5aiai/

006neuralnetworks/neural-networks.htm, 2021. Acedido: 29-05-2021. (p. 12)

[21] Graham Templeton. Artificial neural networks are changing the world. what are they? https://www.extremetech.com/extreme/215170-artificial-neural-networks-are-changing-the-world-what-are-they, 2021. Acedido: 20-04-2021. (p. 13)

[22] David E Rumelhart and Geoffrey E Hintonf. Learning representations by back-propagating errors. *NATURE*, 323:9, 1986. (p. 13)

[23] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong. *Mathematics for machine learning*. Cambridge University Press, 2020. (p. 14)

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 151. MIT Press, 2016. http://www.deeplearningbook.org. (p. 15)

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 296. MIT Press, 2016. http://www.deeplearningbook.org. (p. 15)

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 307. MIT Press, 2016. http://www.deeplearningbook.org. (p. 16)

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 308. MIT Press, 2016. http://www.deeplearningbook.org. (p. 17)

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 307. MIT Press, 2016. http://www.deeplearningbook.org. (p. 17)

[29] Wikipedia. Overfitting. https://en.wikipedia.org/wiki/Overfitting, 2021. Acedido: 29-05-2021. (p. 19)

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. (p. 23)

[31] Image classification on imagenet. https://paperswithcode.com/sota/image-classification-on-imagenet, 2020. Acedido: 21-12-2020. (p. 23)

[32] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on learning theory*, pages 907–940, 2016. (p. 24)

[33] Alexander Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and K Lang. Phoneme recognition: neural networks vs. hidden markov models vs. hidden markov models. In *ICASSP-88., International Conference on Acoustics, Speech, and Signal Processing*, pages 107–108. Citeseer, 1988. (p. 27)

[34] Yann Le Cun, Lionel D Jackel, Brian Boser, John S Denker, Henry P Graf, Isabelle Guyon, Don Henderson, Richard E Howard, and William Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989. (p. 27)

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017. (p. 27)

[36] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. (p. 27)

[37] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959. (p. 27)

[38] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962. (p. 27)

[39] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968. (p. 27)

[40] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 337. MIT Press, 2016. http://www.deeplearningbook.org. (p. 28)

[41] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 338. MIT Press, 2016. http://www.deeplearningbook.org. (p. 29)

[42] Stanford. Cs231n convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks, 2021. Acedido: 03-06-2021. (p. 30)

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, page 341. MIT Press, 2016. http://www.deeplearningbook.org. (p. 31)

[44] Apple. Siri. https://www.apple.com/ios/siri/, 2019. Acedido: 06-06-2019. (p. 35)

[45] Google. Google assistant. https://assistant.google.com/, 2019. Acedido: 07-06-2019. (p. 35)

[46] Amazon. Amazon alexa. https://developer.amazon.com/alexa/, 2019. Acedido: 07-06-2019. (p. 35)

[47] Microsoft. Microsoft cortana. https://www.microsoft.com/en-us/cortana, 2019. Acedido: 07-06-2019. (p. 35)

[48] Yael Pritch Marc Levoy. Portrait mode on the pixel 2 and pixel 2 xl smartphones, 2017. URL https://ai.googleblog.com/2017/10/portrait-mode-on-pixel-2-and-pixel-2-xl.html. Acedido: 08-06-2019. (p. 35)

[49] Apple. Arkit. https://developer.apple.com/arkit/, 2019. Acedido: 06-06-2019. (p. 35)

[50] Brahim Elbouchikhi. Introducing ml kit, 2019. URL https://developers.googleblog.com/2018/05/introducing-ml-kit.html. Acedido: 08-06-2019. (p. 35)

[51] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009. (p. 36)

[52] Jason Brownlee. How to develop a cnn from scratch for cifar-10 photo classification. https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/, 2021. Acedido: 16-05-2021. (p. 45)

[53] Yann LeCun et al. Mnist handwritten digit database. http://yann.lecun.com/exdb/mnist/, 2021. Acedido: 11-05-2021. (p. 47)

[54] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. (p. 47)

[55] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. (p. 47)

[56] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 (11):2278–2324, 1998. (p. 47)

[57] Bill Freeman Antonio Torralba, Rob Fergus. 80 million tiny images. http://groups.csail.mit.edu/vision/TinyImages/, 2021. Acedido: 27-04-2021. (p. 48)

[58] Dustin Franklin. Nvidia jetson tx2 delivers twice the intelligence to the edge. https://developer.nvidia.com/blog/jetson-tx2-delivers-twice-intelligence-edge/, 2021. Acedido: 10-05-2021. (pp. 49 e 50)

[59] Nvidia. Jetpack 4.4 sdk archive. https://developer.nvidia.com/jetpack-sdk-44-archive, 2021. Acedido: 15-05-2021. (p. 49)

[60] Keras. Keras. https://keras.io, 2021. Acedido: 13-03-2021. (p. 50)

[61] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 08 2017. (pp. 51 e 56)

[62] Diogo Horta. Código da dissertação. https://gitlab.com/dmmn1992/keras, 2021. Acedido: 28-05-2021. (p. 69)

Anexo A - Configuração das Redes Desenvolvidas

A.1 Configural Geral

A configuração geral para todos os treinos das redes desenvolvidas é a seguinte:

Dimensão do *Batch* - 128;

Número de *Epochs* - 100;

Função de Perda - Entropia cruzada entre categorias (*categorical crossentropy*);

Algoritmo de Optimização - *RMSProp*.

Taxa de Aprendizagem - 1×10^{-6};

Taxa de Decaimento de Pesos - 1×10^{-4}.

A.2 Configuração Arquitetura de Rede de 4 Camadas (Base)

A configuração da Arquitetura de Rede de 4 Camadas (Base) é a seguinte:

Camada 1 - Camada convolucional

Número de Filtros - 32;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - 2×2;

Padding - *valid*;

Dropout - 20%.

Camada 2 - Camada convolucional

Número de Filtros - 128;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - 2×2;

Padding - *valid*;

Dropout - 20%.

Camada 3 - Camada densa

Número de Unidades - 256;

Dropout - 0%.

Camada 4 - Camada densa

Número de Unidades - 10;

Dropout - 0%.

A.3 Configuração Arquitetura de Rede de 6 Camadas (*Deep*)

A configuração da Arquitetura de Rede de 6 Camadas (*Deep*) é a seguinte:

Camada 1 - Camada convolucional

Número de Filtros - 32;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - Não Aplicado;

Padding - *same*;

Dropout - 0%.

Camada 2 - Camada convolucional

Número de Filtros - 64;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - 2×2;

Padding - *valid*;

Dropout - 25%.

Camada 3 - Camada convolucional

Número de Filtros - 128;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - Não Aplicado;

Padding - *same*;

Dropout - 0%.

Camada 4 - Camada convolucional

Número de Filtros - 256;

Dimensão do Filtro - 3×3;

Dimensão *Max Pooling* - 2×2;

Padding - *valid*;

Dropout - 25%.

Camada 5 - Camada densa

Número de Unidades - 512;

Dropout - 50%.

Camada 6 - Camada densa

Número de Unidades (*Cifar10*) - 10;

Número de Unidades (*Cifar100*) - 100;

Dropout - 0%.