Universidade de Brasília - UnB

Instituto de Ciências Exatas - IE

Departamento de Estatística - EST

Redes Neurais Profundas para o problema de Classificação

José Guilherme Ribeiro Lopes

Brasília

2019

José Guilherme Ribeiro Lopes

Redes Neurais Profundas para o problema de Classificação

Trabalho de Conclusão de Curso apresentado ao Departamento de Estatística da Universidade de Brasília, como parte dos requisitos para a obtenção do título de Bacharel em Estatística.

Orientador: Prof. Dr. Donald Matthew Pianto

Brasília

2019

Resumo

Este trabalho apresenta um estudo teórico e prático sobre modelos de Redes Neurais Profundas aplicados em problemas de classificação. Inicialmente, é feita uma abordagem sobre Aprendizado de Máquina e o seu método de validação mais utilizado. Nos fundamentos de Aprendizagem Profunda, entra-se em detalhes sobre os componentes principais de uma Rede Neural e as formas de pré-processamento e representações de dados para se trabalhar com estes modelos. Além disso, há um tópico que trata sobre a arquitetura das Redes Neurais Convolucionais, muito utilizadas em aplicações de Visão Computacional. O último tópico teórico faz uma introdução ao uso da Computação em Nuvem, uma tecnologia muito útil para se trabalhar com grandes bases de dados e utilizar processadores avançados. A última parte do trabalho apresenta os resultados de três aplicações de Redes Neurais Profundas para classificação, onde foram utilizadas bases de dados do tipo imagem, texto e áudio. Todas as aplicações foram realizadas utilizando-se computação em nuvem e tiveram resultados satisfatórios.

Palavras-chave: Redes Neurais Profundas, Aprendizagem Profunda, Aprendizado de Máquina, Computação em Nuvem, Visão Computacional, Rede Neural Convolucional, Classificação de Imagem, Classificação de Texto, Classificação de Áudio.

Abstract

This work presents a theoretical and practical study on the models of Deep Neural Networks applied in classification problems. Initially, an approach is taken on Machine Learning and its most commonly used validation method. In the Fundamentals of Deep Learning, we go into detail about the main components of a Neural Network and the forms of preprocessing and representations of data to work with these models. In addition, there is a topic that deals with the architecture of the Convolutional Neural Networks, much used in Computational Vision applications. The last theoretical topic is an introduction to the use of Cloud Computing, a very useful technology for working with large databases and using advanced processors. The last part presents the results of three applications of Deep Neural Networks for classification, where image, text and audio type databases were used. All applications were performed using cloud computing and had satisfactory results.

Keywords: Artificial Neural Network, Deep Learning, Machine Learning, Cloud Computing, Computational Vision, Convolutional Neural Network, Image Classification, Text Classification, Audio Classification.

Sumário

1. Introdução **1**

2. Objetivos **1**

3. Metodologia **1**
3.1 Metodologia de Estudo 1
3.2 Metodologia de Aplicação 2

4. Fundamentos de Aprendizado de Máquina **2**
4.1 O Aprendizado de Máquina como novo paradigma de programação 2
4.2 As Categorias de Aprendizado de Máquina 3
4.2.1 Aprendizagem Supervisionada 3
4.2.2 Aprendizagem Não-Supervisionada 3
4.2.3 Aprendizagem Semi-Supervisionada 3
4.2.4 Aprendizagem de Reforço ou *Reinforcement Learning* 4
4.3 Validação de Modelos de Aprendizado de Máquina 4

5. Fundamentos de Aprendizado Profundo e Redes Neurais **4**
5.1 Redes Neurais Artificiais 5
5.2 Função de Ativação 7
5.3 Treinamento e Ajuste de Redes Neurais 8
5.3.1 Função Perda 8
5.3.2 Método da Descida de Gradiente 9
5.3.3 Descida por Gradiente Estocástico 10
5.3.4 Otimização por *Back-Propagation* 11
5.4 Arquitetura da Rede Neural Convolucional 11
5.4.1 A Operação de Convolução 12
5.4.2 Convolução em Imagens 13
5.5 Representações de dados para Redes Neurais 14
5.5.1 Dados Vetoriais 14
5.5.2 Dados Sequenciais ou em Série Temporal 15
5.5.3 Imagens 15
5.5.4 Vídeos 16
5.6 Pré-processamento de dados para Redes Neurais 16
5.6.1 Vetorização 16
5.6.2 Normalização dos valores 16
5.6.3 Manipulação de Dados Faltantes 17

6. Fundamentos de Computação em Nuvem **17**
6.1 Tipos de Computação em Nuvem 17
6.1.1 Infraestrutura como Serviço 17
6.1.2 Plataforma como Serviço 18
6.1.3 Software como Serviço 18

7. Resultados da Aplicação **18**

7.1 Dados do tipo Imagem: Classificação de Cães e Gatos 18
7.2 Dados do tipo Texto: Classificação da Crítica de um filme 22
7.3 Dados do tipo Áudio: Classificação de voz 24

8. Conclusão **27**

Referências Bibliográficas **27**

Apêndice A - Código utilizado na aplicação de Classificação de Imagem **28**

Apêndice B - Código utilizado na aplicação de Classificação de Texto **32**

Apêndice C - Código utilizado na aplicação de Classificação de Áudio **34**

1. Introdução

Nos últimos anos, os modelos em Redes Neurais Profundas representaram um progresso extraordinário no campo da Inteligência Artificial, e as consequências deste progresso se estendem a praticamente qualquer área.

Com o desenvolvimento da capacidade de processamento e armazenamento dos computadores, em especial quando a computação pode ser realizada em um servidor remoto, denominado de *Computação em Nuvem*, novos dados de diferentes formatos passaram a ser gerados e processados de forma massiva. Esta mudança tecnológica permitiu que modelos estatísticos clássicos e modernos pudessem ser automatizados e aplicados para novos tipos de dados, em especial para quando as bases de dados são muito grandes.

Este Trabalho de Conclusão de Curso consiste no estudo sobre modelos de Redes Neurais e na implementação destes modelos para o problema de classificação, utilizando dados do tipo imagem, texto e áudio, sendo todas as aplicações executadas em um ambiente de Computação em Nuvem.

2. Objetivos

Os objetivos gerais deste trabalho são:

- Realização de um estudo teórico e matemático sobre o funcionamento das Redes Neurais Profundas;
- Apresentação dos fundamentos de Aprendizado de Máquina e Aprendizagem Profunda;
- Abordagem da arquitetura das Redes Neurais Convolucionais;
- Introdução sobre o que é Computação em Nuvem e os tipos de serviços oferecidos;
- Uso de modelos de Redes Neurais Profundas para classificação em bases de dados do tipo imagem, texto e áudio.

3. Metodologia

3.1 Metodologia de Estudo

A parte teórica do Trabalho consiste inicialmente na apresentação dos fundamentos de Aprendizado de Máquina, suas diferentes categorias e método de validação.

Em seguida há o estudo principal do trabalho, que consiste nos fundamento de Aprendizado Profundo e Redes Neurais, o qual aborda os conceitos fundamentais do funcionamento destes modelos, os formatos de dados que podem ser utilizados, como realizar o processamento dos dados para a execução dos algoritmos, além da conceituação da arquitetura Convolucional, uma das principais arquiteturas em Redes Neurais e que será usada em uma das aplicações.

O último capítulo teórico apresenta uma introdução sobre o que é Computação em Nuvem e os tipos de serviços oferecidos.

3.2 Metodologia de Aplicação

O Trabalho consiste na aplicação de Redes Neurais Profundas em três diferentes tipos de dados: imagem, texto e áudio.

Para os dados do tipo imagem, foi utilizada a base de imagens **Dogs-vs-Cats**, disponível no site *Kaggle*, que consiste em 25.000 imagens de cachorros e gatos. Neste problema, foi desenvolvido um modelo em Rede Neural, com arquitetura Convolucional, que teve como objetivo a classificação do animal (cachorro ou gato) contido em uma imagem.

Para os dados do tipo texto, foi utilizada a base de textos **IMDB Movie Reviews**, disponível para *download* através do *framework Keras*, que consiste em 25.000 críticas de filmes. Neste problema, foi desenvolvido um modelo em Rede Neural que teve como objetivo a classificação de uma crítica em positiva ou negativa.

Para os dados do tipo áudio, foi utilizada a base de áudios **Speech Commands**, disponível para *download* no site do *Tensorflow*, que consiste em 65.000 áudios de um segundo de duração com pessoas pronunciando 30 diferentes palavras em língua inglesa. Neste problema, foi desenvolvido um modelo em Rede Neural que tem como objetivo a classificação da palavra dita em um áudio.

O *framework* utilizado para execução das funções de Redes Neurais Profundas foi o *Keras*, biblioteca de funções desenvolvida em linguagem *Python.* Entretanto, a implementação, ajuste e validação dos modelos foram feitas utilizando a linguagem *R.*
O armazenamento dos conjuntos de dados, execução dos cálculos e computação dos modelos foram feitos através do serviço de computação em nuvem da *Amazon Web Services.*

4. Fundamentos de Aprendizado de Máquina

4.1 O Aprendizado de Máquina como novo paradigma de programação

Segundo [**1**], o Aprendizado de Máquina surge da seguinte questão: *"uma máquina pode ir além do que podemos ordená-la a fazer, e aprender por si própria a executar uma tarefa?"*.

Essa questão criou um novo paradigma de programação. Em programação clássica, o usuário desenvolve as regras e fornece os dados para serem processados de acordo com estas regras, e assim espera-se as respostas desejadas. Em Aprendizado de Máquina, o usuário fornece os dados e as respostas desejadas, e espera-se que a máquina retorne as regras que transformam os dados nas respostas.

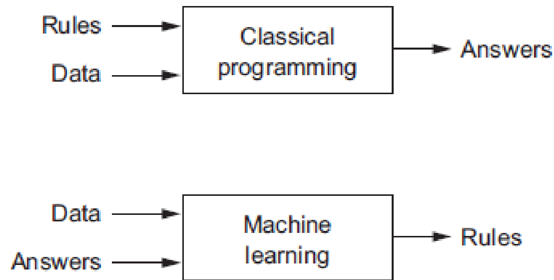

Figura 4.1.1: O Aprendizado de Máquina apresenta um novo paradigma de programação.

4.2 As Categorias de Aprendizado de Máquina

4.2.1 Aprendizagem Supervisionada

É o caso mais comum e pode ser aplicável em problemas de classificação e regressão.

Quando se conhece a variável a ser predita, a aprendizagem supervisionada consiste em mapear e atribuir pesos às variáveis auxiliares, ou explicativas, de modo a maximizar a capacidade preditiva do modelo.

As aplicações realizadas neste Trabalho de Conclusão de Curso são casos de Aprendizagem Supervisionada. No geral, quase todas as aplicações de Redes Neurais Profundas no atual momento pertencem a esta categoria.

4.2.2 Aprendizagem Não-Supervisionada

Esta categoria consiste em aplicar transformações nos dados sem o auxílio de uma variável resposta, com o propósito de visualização e compreensão dos dados, ou para melhor entender as correlações presentes.

A Aprendizagem Não-Supervisionada é frequentemente uma aplicação necessária para melhor entender o conjunto de dados, antes que se possa iniciar a resolução de um problema de Aprendizagem Supervisionada.

Redução de Dimensionalidade e Agrupamento, ou *clustering*, são aplicações comuns de Aprendizagem Não-Supervisionada

4.2.3 Aprendizagem Semi-Supervisionada

Esta categoria pode ser interpretada como uma Aprendizagem Supervisionada, porém os valores da variável resposta são definidos pela própria máquina, tipicamente via um algoritmo heurístico.

A distinção entre Semi-Supervisionada, Supervisionada e Não-Supervisionada às vezes pode não ser tão clara. A Aprendizagem Semi-Supervisionada pode ser interpretada como qualquer

uma das outras, dependendo de como se cria o mecanismo de aprendizagem ou o contexto da aplicação.

4.2.4 Aprendizagem de Reforço ou *Reinforcement Learning*

Em Aprendizagem de Reforço, um agente recebe as informações sobre o ambiente da aplicação em que se está trabalhando e ele aprende a tomar ações de modo a maximizar algum resultado.

Como exemplo, uma Rede Neural que trabalha sobre um jogo de *video-game* e retorna ações a serem tomadas de modo a maximizar sua pontuação, pode ser treinada via Aprendizagem de Reforço.

Atualmente, esta categoria encontra-se mais frequentemente em áreas de pesquisa do que em aplicações reais. Entretanto, é esperado que em alguns anos a Aprendizagem de Reforço seja utilizada em aplicações sobre carros autônomos, robótica, entre outras.

4.3 Validação de Modelos de Aprendizado de Máquina

Em Aprendizado de Máquina, o objetivo é encontrar modelos que se generalizam para quaisquer outros dados da mesma natureza, sendo o sobreajuste o maior obstáculo neste objetivo, que é quando o modelo tem uma boa *performance* nos dados utilizados durante o ajuste do modelo, mas não faz boas predições em dados novos. Como só se pode controlar os dados em que o modelo está sendo treinado, é importante ser capaz de medir o poder de generalização do modelo. A validação do modelo em Aprendizado de Máquina consiste, na maior parte dos casos, em dividir os dados disponíveis em três conjuntos: treinamento, validação e teste. O modelo é treinado no conjunto de treinamento e validado no conjunto de validação. Uma vez que o modelo esteja bem ajustado, seu poder de previsão é mensurado no conjunto de teste.

Segundo [**1**], a proporção da divisão do conjunto de dados em treinamento, validação e ajuste não possui regra definida, mas em geral, a divisão segue as seguintes medidas: 60% para treinamento, 10% para validação e 30% para teste.

A razão para o uso do conjunto de validação é que o desenvolvimento de um modelo envolve o ajuste de seus parâmetros, e o conjunto de validação serve como um parecer destes ajustes. Em essência, estes ajustes são uma forma de aprendizado do modelo, pois é a busca de uma configuração ideal dentro de um espaço paramétrico.

5. Fundamentos de Aprendizado Profundo e Redes Neurais

Segundo [2], Aprendizado Profundo é um subcampo específico de Aprendizado de Máquina, e consiste em uma abordagem das representações de aprendizagem a partir de dados que enfatizam o aprendizado de camadas sucessivas com representações cada vez mais significativas.

A camada é a estrutura de dado fundamental nas Redes Neurais, e consiste de um processamento de dados que recebe um ou mais vetores e retorna um ou mais vetores de dados transformados. As camadas possuem *pesos*, que especificam as transformações a serem feitas

nos dados, de modo a criar o modelo preditivo. Estes pesos são atualizados via Descida por Gradiente Estocástico. Neste contexto, a aprendizagem do modelo consiste em encontrar um conjunto de valores para os pesos de todas as camadas de uma rede, de modo a maximizar o poder preditivo deste modelo. Juntos, camadas e pesos contêm o *conhecimento* de uma Rede Neural Profunda.

O termo 'Profundo' é proveniente da ideia de sucessivas camadas de representação. O número de camadas que contribui para o modelo denomina a profundidade do modelo. Construções modernas de Aprendizagem Profunda muitas vezes envolvem dezenas e até centenas de camadas sucessivas de representação.

5.1 Redes Neurais Artificiais

Em Aprendizado Profundo, estas representações por camadas são treinadas por via de modelos denominados **Redes Neurais**, ou Redes Neurais Artificiais, estruturadas em camadas empilhadas umas sobre as outras.

O termo Rede Neural é uma referência à Neurobiologia, pois parte dos conceitos centrais em Aprendizado Profundo foram desenvolvidos tendo como inspiração o desenho do entendimento sobre neurônios e o cérebro humano. Todavia é importante ressaltar que modelos em Aprendizado Profundo não são modelos cerebrais. Para os propósitos de modelagem preditiva, Aprendizado Profundo é uma estrutura matemática com o objetivo de aprender representações e padrões nos dados.

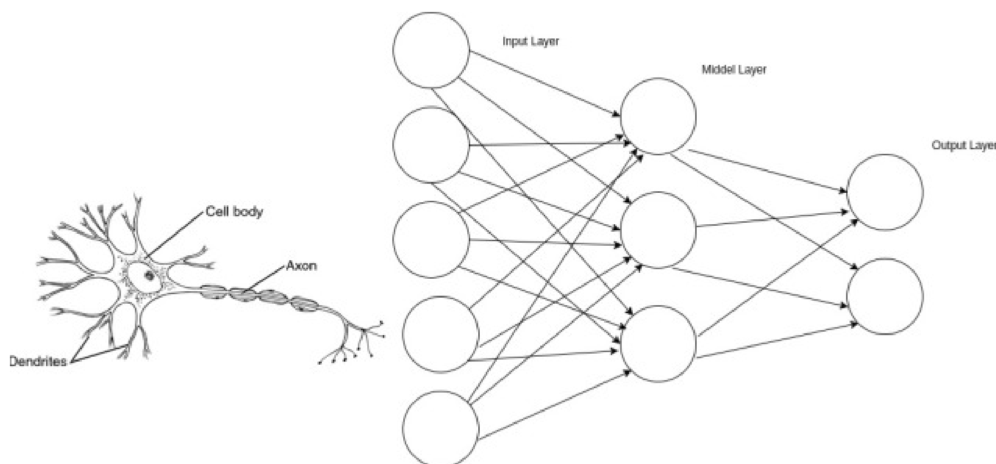

Figura 5.1.1: A estrutura do neurônio biológico é a inspiração dos modelos de Redes Neurais Artificiais.

Cada camada em uma Rede Neural Profunda aplica uma transformação que desfragmenta os dados. Assim, as Redes Neurais podem ser interpretadas como uma cadeia de complexas transformações geométricas em um espaço multi-dimensional, implementadas por longas séries de pequenos passos.

As *unidades* de uma camada consistem na dimensão da representação espacial da camada. A dimensionalidade da representação espacial de uma camada pode ser intuitivamente

compreendida como o tamanho da liberdade que a rede terá na etapa de aprendizagem. Ter mais unidades permite à rede aprender representações mais complexas nos dados, porém torna a rede computacionalmente mais cara e também pode levá-la a aprender padrões não desejados nos dados, consequentemente levando ao sobreajuste do modelo. Uma unidade pode ser representada matematicamente pela seguinte fórmula:

$$y = f\left(\sum_i w_i * x_i + b\right) = f(w'x) \tag{1}$$

no qual $x = (1, x_1, x_2, \dots)$ representa as entradas do neurônio, que são as saídas de neurônio da camada anterior; y a saída desse neurônio; $w = (b, w_1, w_2, \dots\ )$ os pesos associados às entradas; b o viés do neurônio, que representa o valor ao qual a saída tende na ausência de entrada, semelhante a um intercepto. O conjunto dos pesos w e dos vieses b para todos os neurônios são os parâmetros da Rede Neural a serem estimados. Finalmente, f é a função de ativação (ou função de ligação) do neurônio, usada para introduzir uma não-linearidade do modelo [**2, 5**].

Há duas decisões fundamentais de arquitetura da rede a serem feitas na construção de um modelo [**1**]:

- Quantas camadas usar.
- Quantas unidades escolher para cada camada.

O treinamento de uma Rede Neural envolve os seguintes objetos:

- *Camadas*, que são combinadas em uma rede.
- Os *Dados de Entrada*, ou *inputs* ou observações, com as respectivas variáveis respostas.
- A *Função Perda*, ou Função Custo, que define o parecer usado para o aprendizado.
- O *Otimizador*, que determina como o aprendizado deve proceder.

A figura 5.1.2 mostra a interação entre estes objetos.

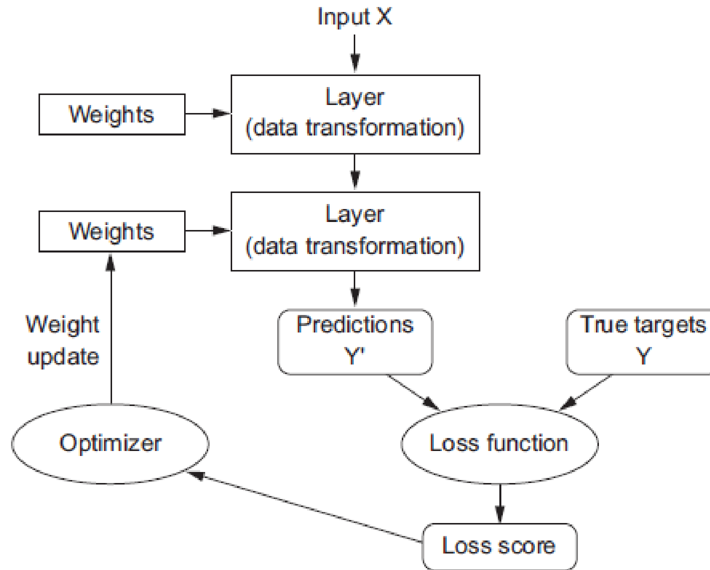

Figura 5.1.2: Diagrama do Aprendizado de uma Rede Neural.

A rede, composta por camadas que são encadeadas, mapeia os dados de entrada às previsões. A Função Perda então compara estas previsões com os valores reais, produzindo um valor de perda, ou resíduo, que é uma medida do quanto as previsões da rede correspondem ao que é esperado. O Otimizador então usa o valor de perda para atualizar os pesos da rede.

Inicialmente, os pesos da rede são valores definidos aleatoriamente. Naturalmente, a predição inicial é muito distante do que deveria ser e a função perda é muito alta. Ao realizar iterações no processo de ajuste do modelo, espera-se que os pesos sejam ajustados na direção correta, e que a função perda decaia.

O Otimizador determina como a rede vai ser atualizada, baseando-se na Função Perda.

Uma rede treinada é aquela com perda mínima, ou seja, em que os valores preditos são os mais próximos que podem ser dos valores reais.

5.2 Função de Ativação

Sem uma função de ativação, as camadas de uma Rede Neural aplicariam apenas transformações lineares nos dados, o que limitaria muito o espaço de hipóteses das camadas. Assim, para se obter um espaço de hipóteses mais amplo e que se beneficia de representações profundas, é preciso uma função não-linear, também chamada de função de ativação. Além da não-linearidade, a função de ativação também afeta o treinamento dos pesos das unidades.

Uma das funções mais usadas em modelos de Redes Neurais é a **Sigmoide Logística**, representada pela seguinte equação:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

A Sigmoide Logística tende a 0 para valores muito pequenos e tende à 1 para valores muito grandes.

Outra função de ativação utilizada em Redes Neurais é a **Tangente Hiperbólica**, representada pela seguinte equação:

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{3}$$

A Tangente Hiperbólica tem curva similar à Sigmoide Logística, mas com a diferença que seus valores vão de -1 a 1.

Uma função que é uma alternativa às duas funções anteriores é a função **Retificadora Linear**, ou simplesmente *relu*, que é definida como:

$$r(x) = max(0, x) \tag{4}$$

A função Retificadora é igual à função identidade para valores positivos, e igual à zero para os valores negativos.

A figura 5.2.1 mostra os gráficos para cada uma destas funções.

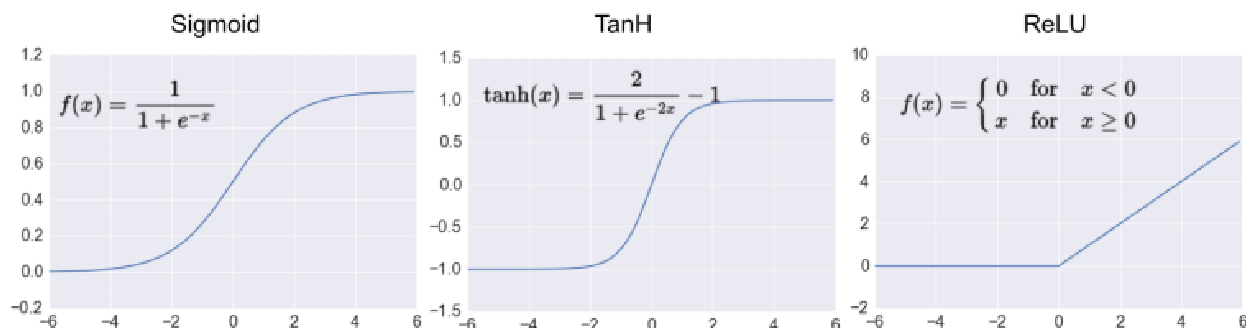

Figura 5.2.1: Gráficos e fórmulas das principais Funções de Ativação.

5.3 Treinamento e Ajuste de Redes Neurais

5.3.1 Função Perda

Para controlar a saída de uma rede neural, é necessário que se possa mensurar o quanto esta saída diverge do valor esperado. Para isso usa-se a **Função Perda**, ou Função Custo, da rede, que calcula o escore da distância entre o que foi predito e o que era esperado.

A perda deve ser minimizada durante o treinamento, pois ela representa uma medida de erro do modelo.

A escolha correta da Função Perda para o problema é muito importante, pois a rede irá tomar todo o atalho que puder para minimizá-la. Portanto, se a Função Perda não correlacionar corretamente com a tarefa a ser realizada, a rede pode acabar realizando coisas não desejadas.

As Funções Perda mais utilizadas são:

- *Crossentropy binária*: para problemas de classificação de duas classes
- *Crossentropy multiclasses*: para problemas de classificação de muitas classes
- *Erro Quadrático Médio*: para problemas de Regressão em variáveis com distribuição Normal

Crossentropy é uma quantidade oriunda do campo da Teoria da Informação que mede a distância entre funções de probabilidade, que no caso de Redes Neurais com objetivo de classificação, a distância entre a verdadeira distribuição dos dados e as previsões do modelo [**1**].

No contexto de regressão linear, uma função perda $J : \mathbb{R}^n \rightarrow \mathbb{R}$ pode ser baseada no erro quadrático médio das predições geradas pelo modelo e o valor verdadeiro presente no conjunto de entrada [**6**]:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(X^{(i)}) - y^{(i)})^2 \tag{5}$$

Tal que $X \in \mathbb{R}^{nm}$ corresponde a uma base de dados com m exemplos de n dimensões, $y \in \mathbb{R}^m$ ao vetor de valores "alvo" para cada exemplo do conjunto, e θ aos parâmetros aprendidos pelo modelo.

A partir disso, o objetivo de otimização do algoritmo de regressão linear é encontrar um conjunto de parâmetros ótimo θ^* que minimize a função perda J. Um algoritmo iterativo conhecido como método da descida de gradiente é capaz de determinar um θ^* ótimo encontrando o mínimo local da função perda J [**6**].

O processo de treinamento em modelos de Redes Neurais Profundas consiste em usar a função perda como um parecer para ajustar os valores dos pesos, de modo a reduzir os erros de previsão do modelo.

5.3.2 Método da Descida de Gradiente

O método da descida de gradiente é um algoritmo de otimização utilizado para minimizar uma função diferenciável $f(x)$ baseado em seu gradiente $\bigtriangledown_x f$, que é definido como um vetor de derivadas parciais tal que cada elemento corresponde a uma dimensão do domínio da função [**6**]. Por exemplo, suponha uma função de custo $J(\theta)$ tal que $\theta \in \mathbb{R}^n$, $\bigtriangledown_\theta J$ é definido por:

$$\bigtriangledown_\theta J = \left(\frac{\partial J}{a\theta_1}, \frac{\partial J}{a\theta_2}, ..., \frac{\partial J}{a\theta_n} \right) \tag{6}$$

O gradiente de uma função $J(\theta)$ corresponde à direção em que os parâmetros devem mudar para $J(\theta)$ sofrer algum acréscimo. Como o objetivo é de minimização, usamos a direção contrária à dada por $\bigtriangledown_\theta J$. Portanto, atualizamos os parâmetros θ usando:

$$\theta_{i+1} = \theta_i - \alpha \bigtriangledown_\theta J(\theta_i) \quad (7)$$

Em que α corresponde à taxa de aprendizado, sendo um escalar positivo que determina o tamanho de cada passo realizado pelo algoritmo. Quanto menor a taxa de aprendizado, mais iterações serão necessárias para atingir o mínimo local. No entanto, um valor deα muito alto pode causar divergência. Portanto, um valor de α pode ser escolhido usando busca binária e selecionando aquele valor que possui melhor desempenho em tempo de treinamento e que não tem problemas de convergência.

As iterações do método da descida de gradiente podem ser visualizadas no exemplo fictício apresentado na figura 5.3.2.1, que consiste em um gráfico de superfície de uma função $J : \mathbb{R}^{\not\vDash} :\rightarrow \mathbb{R}$. Cada seta representa a transição dos parâmetros θ após cada iteração do método da descida de gradiente. O ponto amarelo se refere ao custo com os pesos iniciais. A cada iteração do método, os parâmetros se ajustam de forma a diminuir o valor da função. O método é interrompido quando o mínimo local, representado pelo ponto verde, é atingido.

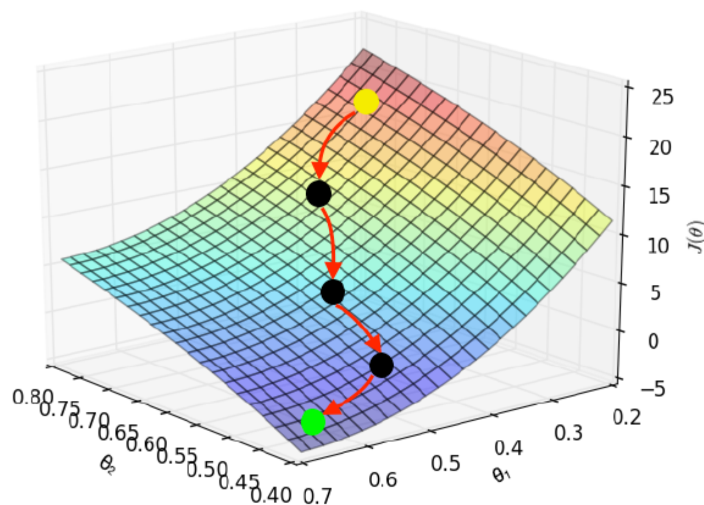

Figura 5.3.2.1: Visualização do método da descida de gradiente para encontrar o mínimo local de uma função perda $J(\theta_1, \theta_2)$.

5.3.3 Descida por Gradiente Estocástico

O algoritmo iterativo de Descida por Gradiente Estocástico é fundamental na etapa de treinamento de um modelo de Rede Neural. Em cada iteração, um pequeno lote do conjunto de dados é usado para se obter o valor e o gradiente da Função Perda, que será usada para a estimação de sua esperança [**5**].

Esta divisão do conjunto de dados em pequenos lotes permite que o algoritmo demande menos esforço computacional, tenha uma convergência mais veloz e não se limite a mínimos locais.

Para cada lote do conjunto de dados, o gradiente da função perda é definido do seguinte modo:

$$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta) \tag{8}$$

em que m é o número de observações do lote, e $L(x^{(i)}, y^{(i)}, \theta) = J(\theta)$ é a função perda para a observação i. Em seguida, com a obtenção do gradiente aproximado g, as estimações dos parâmetros podem ser atualizadas por

$$\theta \leftarrow \theta - \epsilon g \tag{9}$$

em que ϵ é a taxa de aprendizado do algoritmo. Essa taxa influencia no tempo, ou número de iterações, necessário para se chegar num ponto de mínimo da função perda, local ou absoluto, quando o gradiente calculado for zero.

5.3.4 Otimização por *Back-Propagation*

O Back-Propagation, algoritmo central em Aprendizado Profundo, é um método em que o gradiente da função perda vai sendo calculado de camada em camada a partir do final e voltando para a entrada da rede.

Este método usa a regra da cadeia do Cálculo para obter o gradiente de funções encadeadas. Por exemplo, tendo $y = g(x)$ e $z = f(y) = f(g(x))$, tem-se pela regra da cadeia que

$$\frac{dz}{dx} = \frac{dz}{dy} . \frac{dy}{dx} \tag{10}$$

Com isso, o Back-Propagation irá procurar o gradiente da função perda para cada peso w da Rede.

5.4 Arquitetura da Rede Neural Convolucional

De acordo com [**6**], Rede Neural Convolucional (CNN) é um tipo de modelo de Aprendizado Profundo bastante utilizado em aplicações de Visão Computacional. Foi introduzido em 1989 por LeCun et al. através de um trabalho desenvolvido para reconhecer dígitos escritos à mão em uma imagem de entrada, sendo uma aplicação de bastante relevância para o serviço de correio norteamericano da época.

A principal diferença entre uma camada densamente conectada e uma camada convolucional é que camadas densas aprendem via padrões globais dos dados de entrada (por exemplo, de uma imagem, padrões que envolvem todos os pixels), enquanto camadas convolucionais aprendem via padrões locais dos dados de entrada.

Esta característica fornece às Redes Convolucionais duas propriedades:

- *Os padrões aprendidos são invariantes.* Por exemplo, depois de aprender um certo padrão no canto inferior direito de uma imagem, uma rede convolucional pode reconhecê-lo em qualquer outro lugar, como no canto superior esquerdo. Por outro lado, uma rede densamente conectada teria que aprender o padrão novamente caso ele aparecesse em um novo local. Isso torna as redes convolucionais muito eficientes ao processar imagens, pois o mundo visual é fundamentalmente invariante. Consequentemente, redes convolucionais precisam de menos amostras de treinamento para aprender representações que têm alto poder de generalização.
- *É possível aprender hierarquias espaciais de padrões (ver figura 5.4.1).* Uma primeira camada de convolução vai aprender pequenos padrões locais, como bordas, uma segunda camada de convolução pode aprender padrões maiores oriundos da primeira camada, e assim por diante. Isto permite que as redes convolucionais aprendam com eficiência conceitos visuais cada vez mais complexos e abstratos, tendo em vista que o mundo visual é fundamentalmente espacialmente hierárquico.

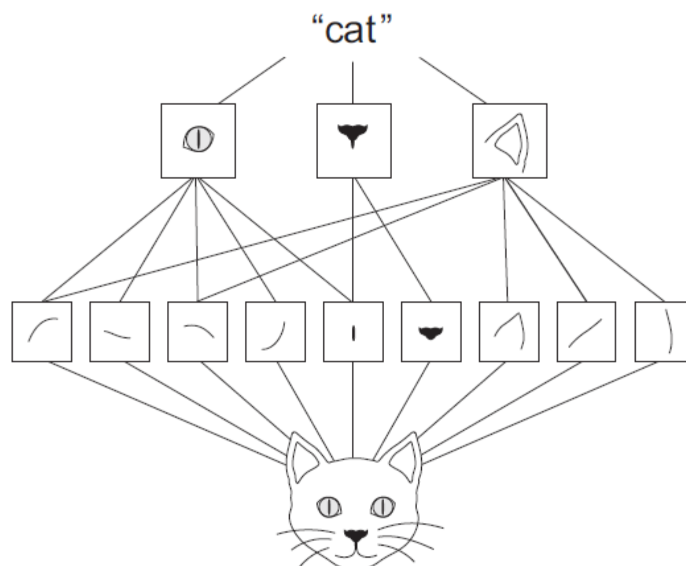

Figura 5.4.1: O mundo visual forma uma hierarquia espacial de módulos visuais.

Redes convolucionais operam sobre tensores tri-dimensionais, sendo dois eixos espaciais (altura e largura) e um eixo de profundidade. Para uma imagem do tipo RGB, a dimensão do eixo de profundidade é três, pois a imagem possui três canais de cores: vermelho, verde, e azul. Para uma imagem em preto e branco, a profundidade é única e definida para tons de cinza.

5.4.1 A Operação de Convolução

Sejam as funções $p : \mathbb{R} \to \mathbb{R}$ e $q : \mathbb{R} \to \mathbb{R}$. A convolução é definida como uma operação linear $*$ que computa a superposição de duas funções em função de um deslocamento τ:

$$(p * q)(t) = \int_{-\infty}^{+\infty} p(\tau)q(t-\tau)d\tau \tag{11}$$

A equação acima descreve a definição de uma convolução contínua. Para o contexto de redes neurais artificiais, na qual a entrada é discreta (por exemplo uma imagem digital), descrevemos a definição de convolução discreta como:

$$(p * q)(t) = \sum_{\tau=-\infty}^{\infty} p(\tau)q(t-\tau) \tag{12}$$

5.4.2 Convolução em Imagens

No contexto do domínio espacial, a filtragem de imagens pode ser realizada através de convoluções. Um filtro de imagem é um procedimento que recebe como entrada a imagem original I e uma matriz bidimensional K, também chamada de *kernel*, de dimensões $m \times n$, e produz uma imagem de saída I'. Usando a equação (12) como referência, a convolução pode ser descrita por:

$$I'(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{13}$$

Em outras palavras, a operação de convolução em imagens pode ser compreendida pela ação de movimentar o kernel sobre todas as posições da imagem, computando o produto escalar entre K e $I(i,...,i+m, j,...,j+n)$, ou seja, a parte da imagem coberta pelo *kernel*.

Como exemplo, considere o filtro Laplaciano, que pode ser descrito pelo *kernel* $K_{Laplacian}$:

$$K_{Laplacian} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{14}$$

Um exemplo de aplicação do filtro Laplaciano sobre uma imagem pode ser visto na Figura 5.4.2.1. Como é possível notar, o filtro foi capaz de destacar as bordas presentes na imagem de entrada.

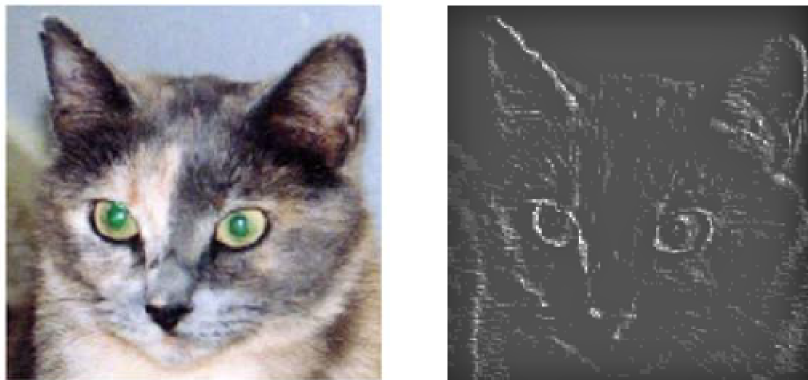

Figura 5.4.2.1: Aplicação do filtro Laplaciano sobre uma imagem.

Além do kernel utilizado, a saída do processo de convolução é determinada por duas variáveis: uso de *padding* e o tamanho do passo.

O *padding* é caracterizado pela inserção simétrica de novos *pixels* ao redor da imagem de entrada. No contexto de filtros, é comum a inserção de *pixels* de valor 0 (*0-padding*). Isso é realizado para que tenhamos um controle sobre o tamanho da saída e que as bordas da imagem original sejam corretamente consideradas no processo de convolução.

O tamanho do passo (também chamado de *stride*) consiste no número de *pixels* que o kernel irá mover sobre a imagem de entrada para realizar cada produto escalar. Ou seja, se o *stride* utilizado for 1, o *kernel* não irá "saltar" *pixels* durante a sua passagem sobre a imagem, percorrendo-a *pixel* a *pixel*. Por outro lado, caso maiores tamanhos sejam utilizados, menos produtos escalares serão computados, e com isso, o tamanho da saída será menor.

Considere que o tamanho da imagem de entrada I seja $W \times H$, que o tamanho do *kernel* K seja $F \times F$, que o tamanho do *padding* seja P, e que o tamanho do passo seja P. A partir disso, a largura W' e a altura H' da imagem filtrada são dadas por:

$$W' = \frac{(W - F + 2P)}{S} + 1 \tag{15}$$

$$H' = \frac{(H - F + 2P)}{S} + 1 \tag{16}$$

5.5 Representações de dados para Redes Neurais

Um *tensor*, ou *array*, é um vetor multidimensional de dados; é uma generalização de vetores e matrizes para um número arbitrário de dimensões. Um tensor é definido por três atributos-chave [**1**]:

- *Número de eixos*
- *Formato* ou *Shape*: é um vetor de valores inteiros que descreve quantas dimensões o tensor possui ao longo de cada eixo.
- *Tipo de dado*: Valores inteiros ou contínuos.

Os dados utilizados em Aprendizado Profundo em geral pertencem à uma das seguintes categorias:

5.5.1 Dados Vetoriais

É o caso mais comum, em que cada observação pode ser codificada como um vetor no formato *(amostra, valor)*. Assim, um conjunto de dados pode ser caracterizado por um tensor bi-dimensional (2D), onde o primeiro eixo representa a amostra e o segundo eixo representa os atributos, ou variáveis, da respectiva amostra.

Por exemplo:

- Um conjunto de dados atuarial considera, para cada pessoa, sua idade, local de residência e renda. Cada pessoa pode ser caracterizada como um vetor de 3 valores, e assim, um conjunto de dados de 100.000 pessoas pode ser representado por um tensor de formato (100000, 3).

5.5.2 Dados Sequenciais ou em Série Temporal

Sempre que o tempo ou a noção de sequência for importante, os dados podem ser armazenados em um tensor tri-dimensional no formato *(amostra, sequência ou tempo, valor)*.

Por exemplo:

- Em um conjunto de dados sobre ações de uma empresa, a cada minuto, é registrado o preço atual da ação e também o maior e menor preço no minuto anterior. Considerando que um dia de movimentação de ações possui 390 minutos, uma amostra de 250 dias desses registros pode ser representada por um tensor no formato (250, 390, 3).

5.5.3 Imagens

Imagens no geral possuem três dimensões: altura, largura e profundidade de cor. Embora imagens em escala de cinza possuam apenas uma profundidade de cor, por convenção os tensores de imagens são representados por tensores de quatro dimensões, no formato *(amostra, altura, largura, profundidade de cor)*.

Por exemplo:

- Um conjunto de 100 imagens de tamanho 256 x 256 no formato RGB (*Red-Green-Blue*) podem ser representadas por um tensor no formato (100, 256, 256, 3).

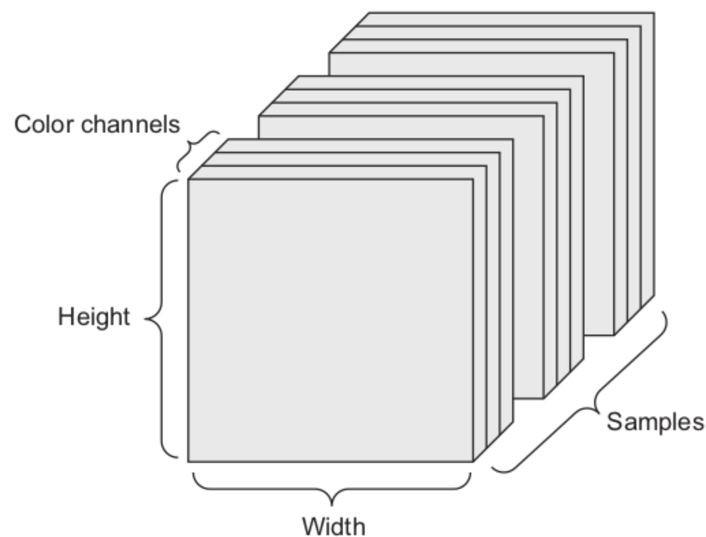

Figura 5.5.3.1: Um tensor de quatro dimensões para uma amostra de imagens.

5.5.4 Vídeos

Dados de Vídeos são representados por uma sequência de quadros, ou *frames*, onde cada quadro é uma imagem colorida. Portanto, um conjunto de vídeos pode ser representado por um tensor de cinco dimensões, no formato *(amostra, quadro, altura, largura, profundidade de cor)*.

Ao se trabalhar com vídeos é importante dar atenção ao espaço necessário para armazenamento dos dados e da capacidade de processamento do computador. Por exemplo, uma amostra de 4 vídeos de 60 segundos de duração, com 4 quadros por segundo e de tamanho 144 x 256, seria armazenada em um tensor de formato (4, 240, 144, 256, 3). Este tensor representa um total de 106 milhões de valores, que se forem armazenados por valores contínuos (*doubles*), representariam 810 MB de dados. Uma forma de contornar este problema é pela compressão do formato do vídeo, como por exemplo o formato *MPEG*.

5.6 Pré-processamento de dados para Redes Neurais

De acordo com [**1**], o pré-processamento dos dados visa transformar os dados da forma em que foram coletados para um formato mais receptível à Rede Neural. Esta tarefa envolve, entre outras coisas, a vetorização, normalização e manipulação de dados faltantes.

5.6.1 Vetorização

Todos os dados de entrada e classes em uma Rede Neural devem ser tensores de dados em ponto flutuante, ou em alguns casos específicos, tensores de inteiros. Independente do tipo dos dados que se deseja processar, o primeiro passo a ser feito é transformá-los em tensores, uma etapa chamada de Vetorização de dados.

5.6.2 Normalização dos valores

Quando os dados possuem valores muito extensos, uma amplitude muito grande ou são heterogêneos, não é seguro aplicá-los diretamente na Rede Neural, pois tal aplicação pode desencadear atualizações muito altas do Gradiente e consequentemente a rede pode não convergir.

Para tornar o aprendizado mais fácil para a Rede Neural, os dados devem ter as seguintes características:

- Valores pequenos: tipicamente, a maior parte dos valores deve estar dentro de um intervalo 0-1.
- Homogeneidade: todos os valores devem estar aproximadamente dentre do mesmo intervalo.

Apesar de não ser sempre necessária, a normalização dos dados de forma que tenham média igual a zero e desvio-padrão igual a 1, é uma prática bastante útil e comumente aplicada.

5.6.3 Manipulação de Dados Faltantes

É muito comum em aplicações reais que existam dados faltantes. Em geral, é seguro tratar um dado faltante como igual a zero em Redes Neurais, com a condição de que o zero já não seja um valor com algum significado. Na exposição aos dados, a rede aprende que o valor 0 significa dado faltante e começa a ignorar o valor.

É importante ressaltar que, quando espera-se dados faltantes no conjunto de teste mas a rede foi treinada em um conjunto sem nenhum dado faltante, resultados equivocados e inesperados podem ser gerados. Nestas situações, é necessário que se gere artificialmente dados que sejam interpretados como faltantes no conjunto de treinamento.

6. Fundamentos de Computação em Nuvem

A *computação em nuvem* consiste essencialmente no acesso e uso de recursos computacionais via Internet. Esta tecnologia fornece uma maneira simples de utilizar servidores, armazenamento, bancos de dados e um amplo conjunto de serviços por meio de um aplicativo Web [**8, 9**].

Uma das vantagens da computação em nuvem é que o usuário não precisa fazer grandes investimentos em hardware, além de evitar todo o processo de instalação, gerenciamento e manutenção dos seus equipamentos. Em vez disso, o usuário seleciona exatamente o tipo e tamanho correto de todos os recursos computacionais necessários e, para a maioria das plataformas de serviços na nuvem, paga somente pelo o que utilizar [**8, 9**].

A computação em nuvem é um recurso muito útil para o trabalho estatístico atual, tendo em vista o crescimento de conjuntos de dados massivos, que muitas vezes são dados coletados em tempo real, e que extrapolam a capacidade de armazenamento de um computador pessoal comum. Além disso, existem algoritmos que demandam muito esforço computacional ou hardwares de alto custo, como por exemplo os algoritmos necessários para modelos de Redes Neurais que trabalham com dados do tipo imagem e exigem uma Unidade de Processamento Gráfico muito potente.

6.1 Tipos de Computação em Nuvem

6.1.1 Infraestrutura como Serviço

A Infraestrutura como Serviço, ou *Infrastructure as a Service* (IaaS), fornece os recursos para a construção de um sistema de Tecnologia da Informação na nuvem, como recursos de rede, computadores (virtuais ou em hardware dedicado) e espaço de armazenamento de dados.

Entre os benefícios da Infraestrutura como Serviço estão a dispensa de investir em hardware próprio e a escalabilidade de infraestrutura sob demanda para suportar cargas de trabalho dinâmicas.

6.1.2 Plataforma como Serviço

A Plataforma como Serviço, ou *Platform as a Service* (PaaS), fornece um ambiente baseado na nuvem com tudo o que é necessário para suportar o ciclo de vida completo da criação e entrega de aplicativos baseados na nuvem, sem o custo e a complexidade de comprar e gerenciar o hardware, software, provisionamento e hospedagem subjacentes.

6.1.3 Software como Serviço

O Software como Serviço, ou *Software as a Service* (SaaS), fornece um produto completo que é executado e gerenciado pelo provedor de serviços. Na maioria dos casos, as pessoas que se referem ao Software como Serviço estão se referindo aos aplicativos do usuário final. Com uma oferta de SaaS, o usuário não precisa pensar sobre como o serviço é mantido ou como a infraestrutura subjacente é gerenciada; ele só precisa pensar em como usar esse software específico. Um exemplo comum de um aplicativo SaaS é o e-mail baseado na Web, no qual é possível enviar e receber e-mails sem precisar gerenciar adições de recursos ao produto de e-mail ou manter os servidores e sistemas operacionais nos quais o programa de e-mail está sendo executado.

7. Resultados da Aplicação

As três aplicações a seguir foram executadas em um servidor em nuvem da *Amazon Web Services*. A máquina virtua utilizada, denominada *p2.xlarge*, teve as seguintes características:

- Sistema Operacional Linux/Ubuntu, versão 16.04;
- Processador 64-bit (x86), Amazon Machine Image (AMI);
- 61GiB de memória;
- Processador Gráfico (GPU) NVIDIA K80 (GK210).

O custo desta máquina virtual, ou instância, foi de 0,99 dólares americanos (US$) por hora utilizada. O tempo necessário para escrita e execução dos algoritmos e códigos em linguagem R foi de aproximadamente 2 horas para cada aplicação. Assim, considerando a cotação do dólar atual, o custo de cada aplicação foi de aproximadamente R$7,00.

Entretanto, é importante ressaltar que no desenvolvimento de um modelo de Redes Neurais Profundas é vantajoso escrever o código e realizar testes em uma máquina local antes de ir para um serviço de computação em nuvem, tendo em vista que etapas como o pré-processamento dos dados e testes do modelo podem demandar bastante tempo.

7.1 Dados do tipo Imagem: Classificação de Cães e Gatos

Nesta aplicação foi desenvolvida uma Rede Neural de arquitetura Convolucional para a classificação de imagens de cães e gatos. O conjunto de dados utilizado foi o *Dogs vs Cats*, que contém 25.000 imagens (12.500 para cada classe de animal). As imagens são de formato

JPEG e com resolução média. A dimensão das imagens foi posteriormente padronizada para 150x150 pixels.

Figura 7.1.1: Exemplos de imagens contidas no conjunto de dados *Dogs vs Cats*.

O conjunto de dados foi divido em três conjuntos: um conjunto de treinamento com 1000 amostras de cada classe, um conjunto de validação com 500 amostras de cada classe, e um conjunto de teste com 500 amostras de cada classe. O uso de somente 4.000 amostras foi para redução do custo com o serviço de computação em nuvem.

A rede neural inicia com quatro camadas convolucionais e função de ativação Retificadora Linear, e por último, uma camada de unidade única e função de ativação Sigmoide. Como a rede finaliza com uma única unidade Sigmoide, foi usada uma função perda *Crossentropy binária*.

A escolha da última camada é devido ao objetivo do modelo ser de classificação binária, assim, a última unidade codifica a probabilidade da imagem ser de uma classe ou outra (no caso, a probabilidade da imagem ser um de cachorro ou de um gato).

O pré-processamento dos dados foi realizado pelos seguintes passos:

- Leitura dos arquivos de imagem;
- Transformação do formato JPEG para grades RGB de pixels;
- Conversão para tensores de ponto flutuante;
- Reescalonamento dos valores de pixels (entre 0 e 255) para o intervalo [0, 1].

O modelo foi então treinado com 30 iterações *(epochs)* sobre todas as amostras do conjunto de treinamento. Para cada iteração, o modelo usava o conjunto de validação para verificar a qualidade do ajuste.

Após o término do treinamento da rede, tem-se um gráfico com os valores da acurácia e perda para ambos os conjuntos em cada iteração.

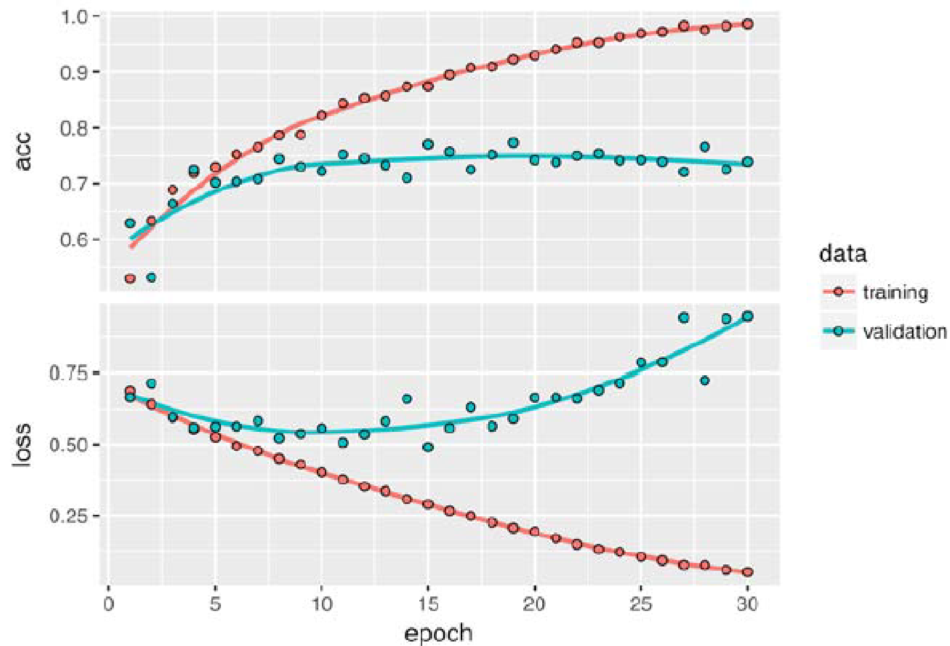

Figura 7.1.2: Acurácia e Perda nos conjuntos de treinamento e validação, em cada iteração.

A figura 7.1.2 permite visualizar que houve um sobreajuste do modelo. A acurácia no conjunto de treinamento atinge quase 100%, enquanto que no conjunto de validação fica em aproximadamente 75%. Em relação à perda, no conjunto de validação o mínimo é atingido após cinco iterações apenas e então cresce, enquanto que a perda no conjunto de treinamento continua a decair até atingir quase zero.

Para lidar com o problema de sobreajuste neste caso, foi utilizada uma técnica de correção bastante comum em modelos de classificação de imagens, denomidada *data augmentation*. Esta técnica consiste em gerar novas imagens para o conjunto de treinamento, a partir de imagens já existentes no mesmo conjunto, por meio de pequenas transformações.

Transformações como rotacionar a imagem, cortar parte da imagem e aplicar um filtro de assimetria fazem com que o modelo, durante a etapa de treinamento, jamais veja a exata mesma imagem duas vezes. Isto ajuda a expor o modelo a variações nos dados e assim ser capaz de fazer maiores generalizações.

A figura 7.1.3 exibe um exemplo de *data augmentation* aplicado à uma amostra do conjunto de treinamento.

Figura 7.1.3: Aplicação da técnica data augmentation em uma amostra do conjunto de treinamento, com o propósito de correção de sobreajuste.

Além da técnica *data augmentation*, foi adicionada uma camada com a técnica *Dropout* para também correção de sobreajuste. *Dropout* é uma técnica estocástica de regulação, que consiste em diluir a Rede durante o treinamento com intuito de reduzir a dependência entre unidades para extrair características.

Com o uso das técnicas descritas acima, a rede foi novamente treinada e o resultado foi que o sobreajuste de fato foi corrigido. A figura 7.1.4 exibe a acurácia e a perda nos conjuntos de treinamento e validação, para cada iteração.

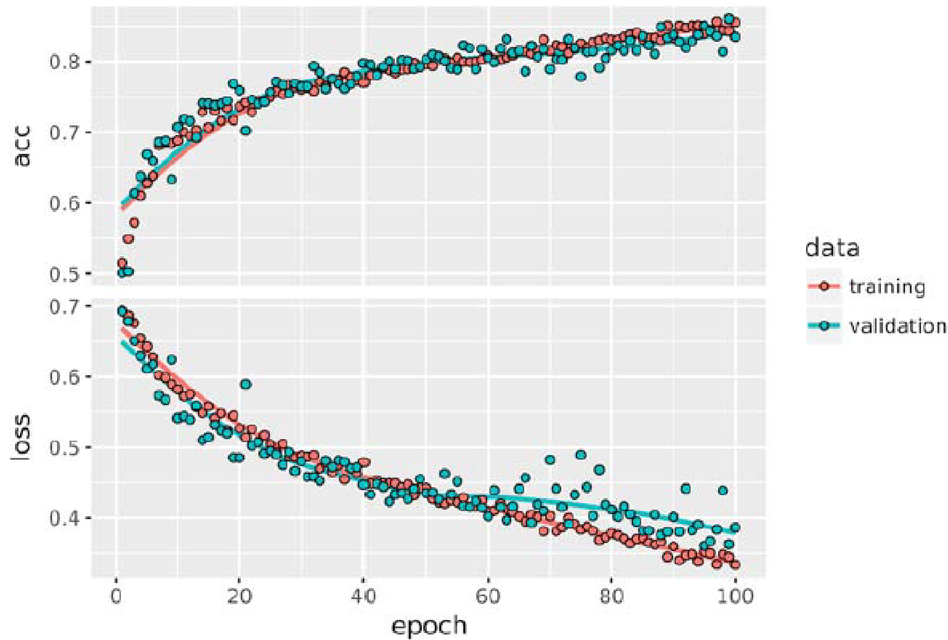

Figura 7.1.4: Métricas para o conjunto de treinamento e validação, após uso de técnicas para correção de sobreajuste.

Pela figura acima, nota-se a similaridade entre as curvas, um sinal de que o modelo ajustado no treinamento está generalizando bem para novos dados. Considerando também o valor da acurácia de aproximadamente 85%, foi considerado que esta aplicação de modelo Rede Neural Convolucional para classificação de imagens teve um desempenho satisfatório.

7.2 Dados do tipo Texto: Classificação da Crítica de um filme

Nesta aplicação, foi feita uma classificação de críticas de filmes, em positiva ou negativa, baseada no conteúdo textual das críticas.

O conjunto de dados utilizado é o fornecido pelo *IMDB - Internet Movie Database*, que consiste de 50.000 críticas. Estes textos foram divididos em 25.000 críticas para treinamento e 25.000 para teste, cada uma consistindo de 50% críticas positivas e 50% críticas negativas.

★ 1/10

Massively overrated.
jojofla 5 August 2002

I continually fail to understand why The Godfather is hailed as "The Greatest Movie of All Time". I've seen it twice--a second time just to make sure--and I have to tell you that I sat there in a stupor, bored out of my mind. And I'm not a teenager raised on MTV; I'm in my 30s and am absolutely devoted to movies--I've seen as many classics (American & foreign) that I can get my hands on. But, for me, The Godfather ranks alongside Singin' in the Rain as the most overrated films of all time.

Singin' in the Rain, at least, I get (it's just my intense dislike for Donald O'Connor that makes me dislike this film). But The Godfather? It's just a bland epic about a bunch of moronic gangsters, with Marlon Brando giving a campy performance, and riddled with repulsive violence. Give me a break. The fact that this movie is so "beloved" has had the direct result that nowadays we got absurdly worse and worse films every year, created by

★ 10/10

An Iconic Film
alexkolokotronis 21 June 2008

Tell me a movie that is more famous than this. Tell me a movie that has had more parodies spinned off its storyline than this. Tell me one movie that has been as quoted as a much as this. The answer is you can't. No movie has had as much of an impact as The Godfather has had ever since it was released.

The acting was simply amazing, what else could you say. What could be more appealing to people(even today) than watching actors like Al Pacino, Marlon Brando, James Caan, Diane Keaton, Talia Shire and Robert Duvall. This is like heaven for someone who is a fan of movies. With this movie Brando was able to bring himself back into the limelight. His performance as the godfather alone is iconic. His character has been recreated so much in films that it has almost if it has not already become a cliché. His performance though was not a cliché. His performance was subtle and breathtaking. It was so genuine and

Figura 7.2.1: Exemplo de crítica negativa e crítica positiva de um filme.

O processamento do texto é feito da seguinte forma: as críticas, que são sequências de palavras, são transformadas em sequências de números inteiros, onde cada inteiro significa uma palavra específica em um dicionário.

Na importação dos dados, foram mantidas somente as 10.000 palavras mais frequentes no conjunto de treinamento. As palavras raras, fora deste conjunto de 10.000, foram descartadas.

Assim, os dados com as críticas de filmes são listas de índices de palavras; e os dados com a classificação das críticas são listas de zeros e uns, onde 0 é para críticas negativas e 1 para críticas positivas, de acordo com a nota atribuída pelo usuário.

Por exemplo:

- Estrutura de uma crítica: *14 298 65 4291 530 43 1622 321 7 1 ...*
- Estrutura da classificação da crítica: *1*

O IMDB fornece um índice das palavras decodificadas, logo é facilmente possível transformar uma crítica de volta à língua Inglesa.

Como não é possível fornecer listas para uma rede neural, foi necessário transformar as sequência de números inteiros em um tensor. Esta transformação foi feita realizando uma vetorização das sequências para um vetor de tamanho 10.000 composto apenas de zeros e uns.

Isso significa que, suponha que uma crítica contenha apenas duas palavras, a terceira e a quinta palavra mais comuns. A sequência de números inteiros seria *[3, 5]*, e após a vetorização, a crítica iria consistir em um vetor de tamanho 10.000 com todos os índices iguais a 0, exceto os índices 3 e 5, que seriam iguais a 1.

Como a aplicação se trata de um problema de classificação binária e a resposta esperada pela rede é uma probabilidade, a função de perda utilizada foi a Crossentropy binária.

Para poder monitorar a acurácia do modelo, foi criado um conjunto de validação, que reune 10000 amostras do conjunto de treinamento.

O modelo foi treinado com 20 iterações sobre todas as amostras do conjunto de treinamento, agrupadas em sub-amostras de 512 observações. Ao mesmo tempo, foram monitoradas a perda e a acurácia das 10.000 amostras de validação.

Na figura 7.2.2, o gráfico da acurácia se encontra no painel superior e o gráfico da perda no painel inferior.

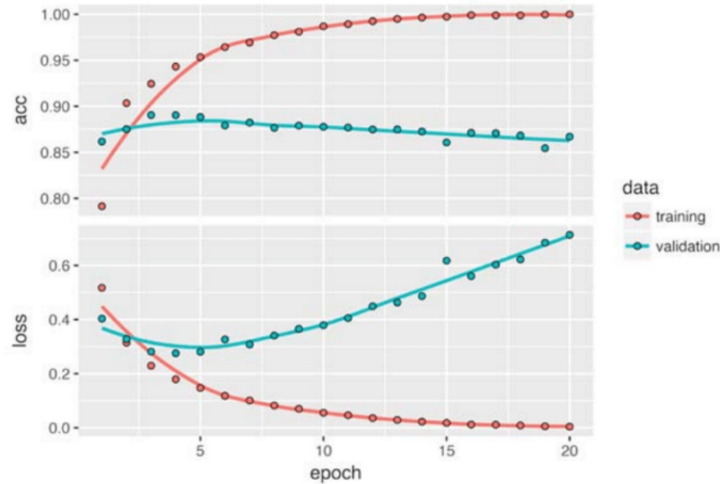

Figura 7.2.2: Métricas de Validação e Treinamento.

É notável que a perda vai decaindo a cada iteração, bem como a acurácia de treinamento aumenta. Este é de fato o resultado esperado quando se está realizando uma otimização via Gradiente Estocástico.

Entretanto, o comportamento da validação começa a divergir do treinamento a partir da quarta iteração. Este comportamento é esperado, pois um modelo que faz boas previsões no conjunto de treinamento não necessariamente irá fazer boas previsões em dados jamais treinados.

Neste caso, houve um sobreajuste do modelo. Após a terceira iteração, a rede neural sobreotimiza os dados de treinamento, e acaba aprendendo representações que são específicas ao conjunto de treinamento e que não se generalizam à outros dados.

Para corrigir o sobreajuste do modelo, o treinamento da Rede Neural foi realizado novamente, mas com apenas quatro iterações de ajuste. Em seguida, a capacidade preditiva do modelo foi avaliada no conjunto de teste e os resultados finais foram uma perda de *0.2765* e uma acurácia de *0.9142*, ou seja, em 91% das observações do conjunto de teste, o modelo classificou corretamente a crítica textual.

7.3 Dados do tipo Áudio: Classificação de voz

O conjunto de dados utilizado nesta aplicação foi o *Speech Commands dataset*, que consiste de 65.000 áudios de um segundo de duração com pessoas pronunciando 30 diferentes palavras em língua inglesa, onde cada áudio contem uma única palavra pronunciada. O objetivo é desenvolver um modelo de Redes Neurais para classificação das palavras.

Nos dados brutos, cada áudio estava definido por um arquivo do tipo *.wav*, que armazena o áudio pelas suas ondas sonoras. Cada onda sonora pode ser representada pelo seu respectivo espectro, e digitalmente pode ser computada usando *Transformada de Fourier* [**7**].

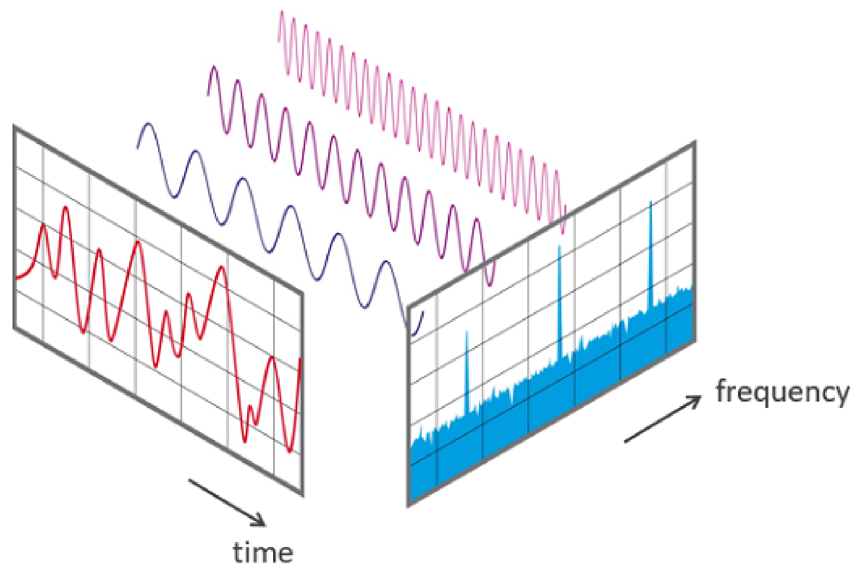

Figura 7.3.1: Representação do áudio.

Uma forma de se trabalhar com dados de áudio é dividi-los em pequenos trechos, que geralmente se sobrepõem. Assim, para cada trecho, é executado um algoritmo que usa a Transformada de Fourier para calcular a magnitude do espectro de frequência. Os espectros são então combinados, lado a lado, para formar o chamado **espectograma** do áudio [**7**].

Nesta aplicação, cada trecho de áudio foi definido por um tamanho de 30 milissegundos. Além disso, a distância entre os centros de trechos adjacentes foi definida por um tamanho de 10 milissegundos, e em seguida foi realizada uma conversão do tamanho dos intervalos de milissegundos para amostras. Assim, cada arquivo de áudio possui 16.000 amostras por segundo.

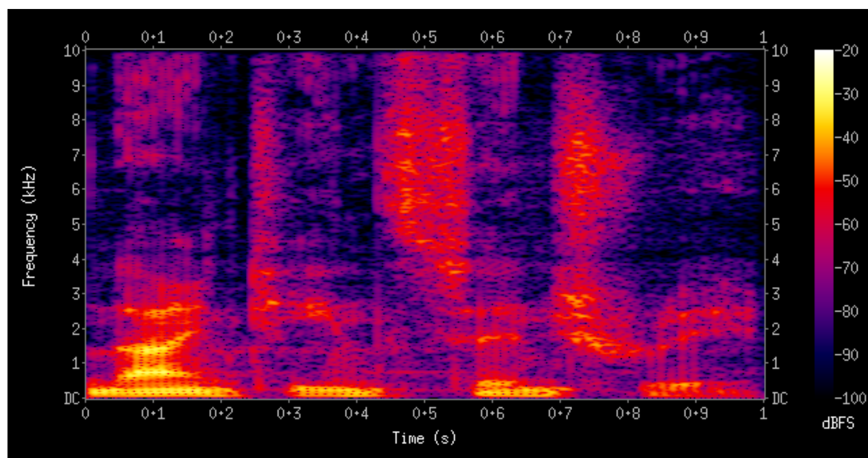

Figura 7.3.2: Espectograma de um áudio.

Após este procedimento, tem-se uma imagem para cada amostra de áudio. Desse modo foram utilizadas **Redes Neurais Convolucionais**, o tipo de arquitetura padrão em modelos de reconhecimento de imagem, para fazer a classificação das palavras pronunciadas nos áudios.

Foram usadas 4 camadas de convoluções combinadas com camadas de agrupamento e duas camadas densas no topo. Como a classificação é em múltiplas categorias, a função de perda utilizada foi a Crossentropy categórica.

O conjunto de dados foi dividido em um conjunto de treinamento, composto por 70% das observações, e em um conjunto de validação, composto por 30% das observações. O modelo foi treinado com 10 iterações sobre todas as amostras do conjunto de treinamento, tendo a acurácia como medida de qualidade do ajuste.

De modo a mensurar a capacidade preditiva, o modelo foi então aplicado no conjunto de validação e o resultado foi uma acurácia de 93%. O diagrama *Alluvial*, exibido abaixo, permite visualizar a matriz de confusão das classes preditas.

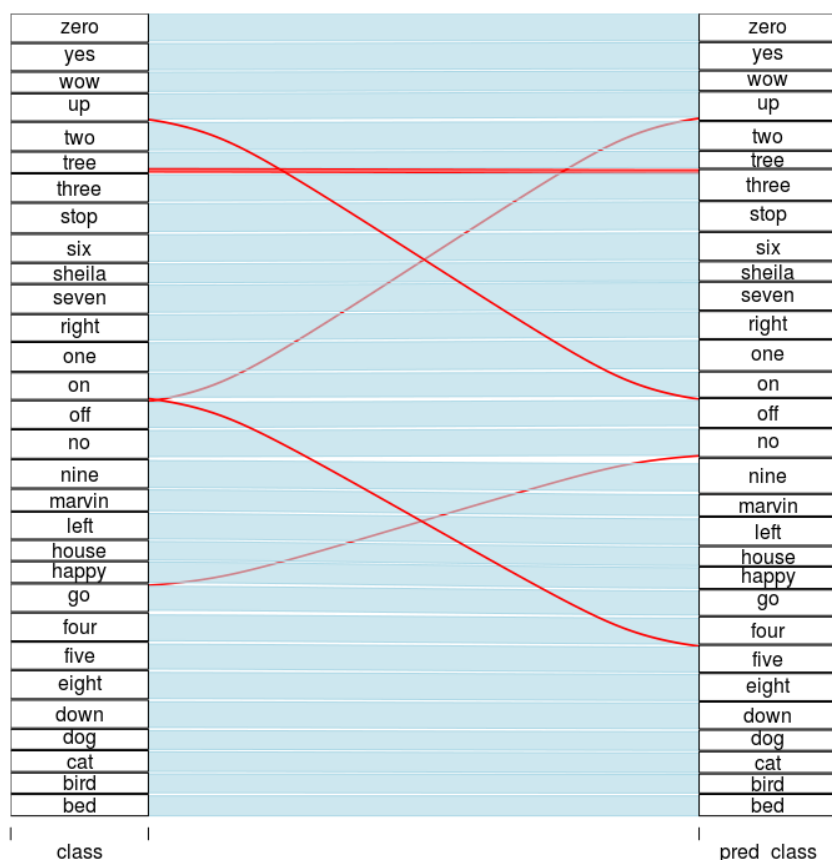

Figura 7.3.3: Diagrama alluvial.

Pelo diagrama, é possível notar que o erro mais comum foi o de classificar a palavra *tree* como *three*, seguido por outros erros como a classificação de *go* como *no*, e *up* como *off*, que são erros compreensíveis e esperados tendo em vista que a pronúcia destas palavras são de fato bastante semelhantes.

Considerando que o conjunto de dados continha 30 diferentes classes, que os erros de previsão do modelo foram majoritariamente em palavras semelhantes, e que a acurácia foi de 93%, pode-se considerar que o modelo teve um ótimo desempenho na tarefa de classificação. Como sugestão de melhoria do modelo para sistemas de reconhecimento de voz, poderiam ser adicionados ruídos em alguns elementos da amostra, de modo a tornar o modelo mais apropriado para aplicações reais.

8. Conclusão

Na Aprendizagem Profunda, é possível criar ótimos modelos de classificação para dados do tipo imagem, texto e áudio ao realizar vetorizações nos dados de entrada e criando camadas que operam transformações geométricas simples nestes vetores, mas que em sequência formam uma transformação geométrica complexa e poderosa. Essa transformação complexa tenta mapear o espaço de entrada para o espaço de destino, um ponto por vez. Esta transformação é parametrizada pelos pesos das camadas (ou parâmetros do modelo), que são iterativamente atualizadas com base em quão bem o modelo está sendo executado.

Uma característica fundamental das transformações geométricas realizadas na Aprendizagem Profunda é que elas devem ser diferenciáveis, o que é necessário para que o modelo possa aprender parâmetros via descida do gradiente. Intuitivamente, isso significa que as passagens geométricas das entradas para as saídas devem ser suaves e contínuas.

Além disso, foi visto que a Computação em Nuvem é uma tecnologia bastante útil e muitas vezes necessária para lidar com grandes bases de dados e para executar algoritmos em hardwares de grande potência.

Sendo assim, com base nos estudos teórico e prático descritos neste trabalho, conclui-se que as Redes Neurais Profundas apresentam um ótimo desempenho na tarefa de classificação, e que a computação em nuvem é um recurso muito proveitoso para o trabalho estatístico atual.

Referências Bibliográficas

[1] CHOLLET, F; ALLAIRE, J.J. **Deep Learning with R**, Manning Publications, 2018.

[2] GOODFELLOW, I; BENGIO, Y; COURVILLE, A. **Deep Learning**, The Mit Press, 2016.

[3] KUTNER, M; NACHTSHEIM, C; NETER, J; LI, W. **Applied Linear Statistical Models**, McGraw-Hill/Irwin, 2004.

[4] WICKHAM, H; GROLEMUND, G. **R for Data Science**, O'Reilly Media, 2017.

[5] RANGEL, P; **Estudo sobre Aprendizado Profundo**, Universidade de Brasília, 2017.

[6] FERREIRA, A; **Estimação do ângulo de direção por Vídeo para Veículos Autônomos utilizando Redes Neurais Convolucionais Multicanais**, Universidade de Brasília, 2017.

[7] FALBEL, D; **TensorFlow for R: Simple Audio Classification with Keras**, https://blogs.rstudio.com/tensorflow/posts/2018-06-06-simple-audio-classification-keras/, 2018.

[8] AMAZON WEB SERVICES, **What is Cloud Computing?**, https://aws.amazon.com/what-is-cloud-computing/

[9] IBM, **What is cloud computing?**, https://www.ibm.com/cloud/learn/what-is-cloud-computing

Apêndice A - Código utilizado na aplicação de Classificação de Imagem

```
library(keras)
library(tidyverse)

# Importação dos dados
original_dataset_dir <- "~/Downloads/kaggle_original_data"

base_dir <- "~/Downloads/cats_and_dogs_small"
dir.create(base_dir)

train_dir <- file.path(base_dir, "train")
dir.create(train_dir)
validation_dir <- file.path(base_dir, "validation")
dir.create(validation_dir)
test_dir <- file.path(base_dir, "test")
dir.create(test_dir)

train_cats_dir <- file.path(train_dir, "cats")
dir.create(train_cats_dir)

train_dogs_dir <- file.path(train_dir, "dogs")
dir.create(train_dogs_dir)

validation_cats_dir <- file.path(validation_dir, "cats")
dir.create(validation_cats_dir)

validation_dogs_dir <- file.path(validation_dir, "dogs")
dir.create(validation_dogs_dir)

test_cats_dir <- file.path(test_dir, "cats")
dir.create(test_cats_dir)

test_dogs_dir <- file.path(test_dir, "dogs")
dir.create(test_dogs_dir)
```

```
fnames <- paste0("cat.", 1:1000, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(train_cats_dir))

fnames <- paste0("cat.", 1001:1500, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(validation_cats_dir))

fnames <- paste0("cat.", 1501:2000, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(test_cats_dir))

fnames <- paste0("dog.", 1:1000, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(train_dogs_dir))

fnames <- paste0("dog.", 1001:1500, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(validation_dogs_dir))

fnames <- paste0("dog.", 1501:2000, ".jpg")
file.copy(file.path(original_dataset_dir, fnames),
          file.path(test_dogs_dir))

# Definição da Rede
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(150, 150, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr = 1e-4),
  metrics = c("acc")
)
```

```
# Pré-processamento dos dados
train_datagen <- image_data_generator(rescale = 1/255)
validation_datagen <- image_data_generator(rescale = 1/255)

train_generator <- flow_images_from_directory(
  train_dir,
  train_datagen,
  target_size = c(150, 150),
  batch_size = 20,
  class_mode = "binary"
)

validation_generator <- flow_images_from_directory(
  validation_dir,
  validation_datagen,
  target_size = c(150, 150),
  batch_size = 20,
  class_mode = "binary"


# Treinamento da rede
history <- model %>% fit_generator(
  train_generator,
  steps_per_epoch = 100,
  epochs = 30,
  validation_data = validation_generator,
  validation_steps = 50

plot(history)

# Data augmentation
datagen <- image_data_generator(
  rescale = 1/255,
  rotation_range = 40,
  width_shift_range = 0.2,
  height_shift_range = 0.2,
  shear_range = 0.2,
  zoom_range = 0.2,
  horizontal_flip = TRUE,
  fill_mode = "nearest")

# Treinamento da rede, com uma camada de Dropout
model <- keras_model_sequential() %>%
  layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu",
                input_shape = c(150, 150, 3)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
```

```r
  layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 512, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr = 1e-4),
  metrics = c("acc")
)

test_datagen <- image_data_generator(rescale = 1/255)

train_generator <- flow_images_from_directory(
  train_dir,
  datagen,
  target_size = c(150, 150),
  batch_size = 32,
  class_mode = "binary"
)

validation_generator <- flow_images_from_directory(
  validation_dir,
  test_datagen,
  target_size = c(150, 150),
  batch_size = 32,
  class_mode = "binary"
)

history <- model %>% fit_generator(
  train_generator,
  steps_per_epoch = 100,
  epochs = 100,
  validation_data = validation_generator,
  validation_steps = 50
)

plot(history)
```

Apêndice B - Código utilizado na aplicação de Classificação de Texto

```
library(tidyverse)
library(keras)

# Importando o conjunto de dados ----------------------------------------

# Serão mantidas as 10000 palavras mais frequentes.
# Isto permite que palavras muito raras sejam descartadas.
imdb <- dataset_imdb(num_words = 10000)

# Divisão do conjunto de dados em treinamento e teste.
train_data <- imdb$train$x
train_labels <- imdb$train$y
test_data <- imdb$test$x
test_labels <- imdb$test$y



# Transformação da crítica decodificada em palavras ----------------------

word_index <- dataset_imdb_word_index()

# Mapeamento de índices inteiros à palavras
reverse_word_index <- names(word_index)
names(reverse_word_index) <- word_index

# Decodificação de texto.
# Os índices 0, 1 e 2 estão deslocados pois são reservados para
# "preenchimento", "início de sequência" e "desconhecido".

decoded_review <- sapply(train_data[[1]], function(index) {
  word <- if (index >= 3) reverse_word_index[[as.character(index - 3)]]
  if (!is.null(word)) word else "?"
})



# Vetorizando as sequências de inteiros em matriz binária ----------------

vectorize_sequences <- function(sequences, dimension = 10000){
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in 1:length(sequences))
    results[i, sequences[[i]]] <- 1
```

```r
  results
}


x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)

# Visualizando a transformação:
str(x_train[1,])

# Conversão das classificações, de inteiros para numéricos:
y_train <- as.numeric(train_labels)
y_test <- as.numeric(test_labels)

# Agora os dados estão prontos para serem processados.



# Definição do modelo ----------------------------------------------------

# Arquitetura:
model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")


# Compilação:
model %>% compile(
  optimizer = "rmsprop",
  loss = loss_binary_crossentropy,
  metrics = metric_binary_accuracy
)



# Criação do conjunto para Validação -------------------------------------

val_indices <- 1:10000

# Conjunto de validação
x_val <- x_train[val_indices, ]
y_val <- y_train[val_indices]

# Conjunto para treinamento
partial_x_train <- x_train[-val_indices, ]
```

```r
partial_y_train <- y_train[-val_indices]



# Treinamento do Modelo -----------------------------------------------------

history <- model %>%
  fit(
    partial_x_train,
    partial_y_train,
    epochs = 20,
    batch_size = 512,
    validation_data = list(x_val, y_val)
  )



# Segundo Treinamento do Modelo, evitando o sobreajuste -------------------

model <- keras_model_sequential() %>%
  layer_dense(units = 16, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 16, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")

model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("accuracy")
)

model %>% fit(x_train, y_train, epochs = 4, batch_size = 512)


# Avaliando o modelo ao prever as classes do conjunto de teste:
results <- model %>% evaluate(x_test, y_test)

results
```

Apêndice C - Código utilizado na aplicação de Classificação de Áudio

```r
# Download dos dados -------------------------------------------------------

dir.create("data")
```

```r
download.file(
  url = "http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz",
  destfile = "data/speech_commands_v0.01.tar.gz"
)

untar("data/speech_commands_v0.01.tar.gz",
      exdir = "data/speech_commands_v0.01")


# Importação dos dados para o R -------------------------------------------

library(stringr)
library(dplyr)

files <- fs::dir_ls(
  path = "data/speech_commands_v0.01/",
  recursive = TRUE,
  glob = "*.wav"
)

files <- files[!str_detect(files, "background_noise")]

df <- data_frame(
  fname = files,
  class = fname %>% str_extract("1/.*/") %>%
    str_replace_all("1/", "") %>%
    str_replace_all("/", ""),
  class_id = class %>% as.factor() %>% as.integer() - 1L
)

saveRDS(df, "data/df.rds")

# Pré-Processamento -------------------------------------------------------

library(tfdatasets)

audio_ops <- tf$contrib$framework$python$ops$audio_ops

data_generator <- function(df, batch_size, shuffle = TRUE,
                           window_size_ms = 30, window_stride_ms = 10) {

  window_size <- as.integer(16000*window_size_ms/1000)
  stride <- as.integer(16000*window_stride_ms/1000)
  fft_size <- as.integer(2^trunc(log(window_size, 2)) + 1)
  n_chunks <- length(seq(window_size/2, 16000 - window_size/2, stride))
```

```r
  ds <- tensor_slices_dataset(df)

  if (shuffle)
    ds <- ds %>% dataset_shuffle(buffer_size = 100)

  ds <- ds %>%
    dataset_map(function(obs) {

      # decodificação do arquivo wav
      audio_binary <- tf$read_file(tf$reshape(obs$fname, shape = list()))
      wav <- audio_ops$decode_wav(audio_binary, desired_channels = 1)

      # criação do espectograma
      spectrogram <- audio_ops$audio_spectrogram(
        wav$audio,
        window_size = window_size,
        stride = stride,
        magnitude_squared = TRUE
      )

      spectrogram <- tf$log(tf$abs(spectrogram) + 0.01)
      spectrogram <- tf$transpose(spectrogram, perm = c(1L, 2L, 0L))

      response <- tf$one_hot(obs$class_id, 30L)

      list(spectrogram, response)
    }) %>%
    dataset_repeat()

  ds <- ds %>%
    dataset_padded_batch(batch_size, list(shape(n_chunks, fft_size, NULL),
                                          shape(NULL)))
}


# Modelo ------------------------------------------------------

library(keras)
library(dplyr)
source("02-generator.R")

df <- readRDS("data/df.rds") %>% sample_frac(1)
id_train <- sample(nrow(df), size = 0.7*nrow(df))

ds_train <- data_generator(df[id_train,], 32L)
```

```r
ds_test <- data_generator(df[-id_train,], 32, shuffle = FALSE)


model <- keras_model_sequential()
model %>%
  layer_conv_2d(input_shape = c(98, 257, 1),
                filters = 32, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 128, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 256, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_dropout(rate = 0.25) %>%
  layer_flatten() %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 30, activation = 'softmax')

# Compilação
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)

# Treinamento
model %>% fit_generator(
  generator = ds_train,
  steps_per_epoch = 0.7*nrow(df)/32,
  epochs = 10,
  validation_data = ds_test,
  validation_steps = 0.3*nrow(df)/32
)

save_model_hdf5(model, "model.hdf5")



# Validação ----------------------------------------------------------------

df_test <- df[-id_train,]
n_steps <- nrow(df_test)/32 + 1

predictions <- predict_generator(
```

```r
  model,
  ds_test,
  steps = n_steps)

str(predictions)

classes <- apply(predictions, 1, which.max) - 1

# Criação do diagrama Alluvial
library(dplyr)
library(alluvial)
x <- df_test %>%
  mutate(pred_class_id = head(classes, nrow(df_test))) %>%
  left_join(
    df_test %>% distinct(class_id, class) %>% rename(pred_class = class),
    by = c("pred_class_id" = "class_id")
  ) %>%
  mutate(correct = pred_class == class) %>%
  count(pred_class, class, correct)

alluvial(
  x %>% select(class, pred_class),
  freq = x$n,
  col = ifelse(x$correct, "lightblue", "red"),
  border = ifelse(x$correct, "lightblue", "red"),
  alpha = 0.6,
  hide = x$n < 20
)
```