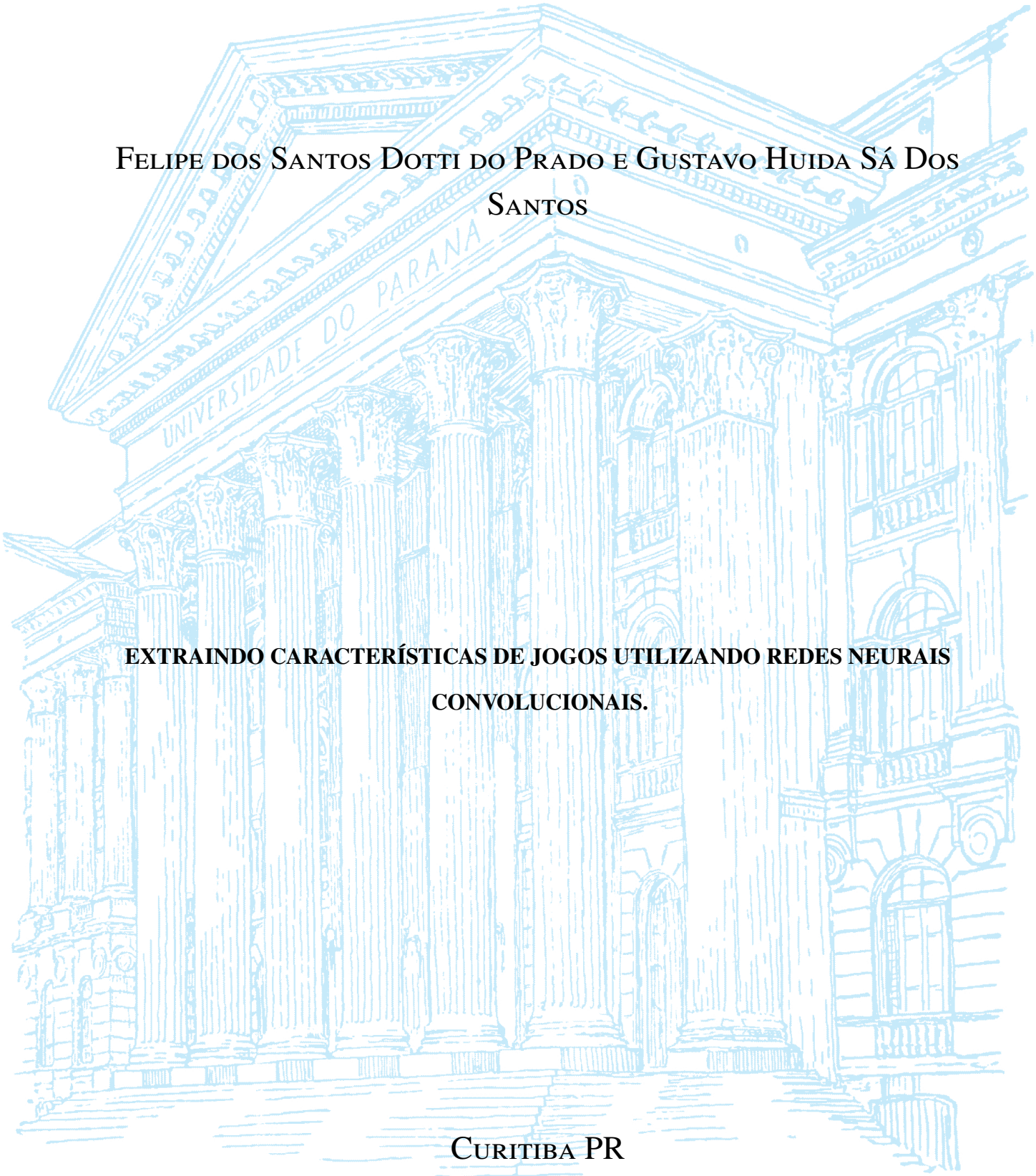
Universidade Federal do Paraná

Felipe dos Santos Dotti do Prado e Gustavo Huida Sá Dos Santos



EXTRAINDO CARACTERÍSTICAS DE JOGOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS.

Curitiba PR
2018

Felipe dos Santos Dotti do Prado e Gustavo Huida Sá Dos Santos

EXTRAINDO CARACTERÍSTICAS DE JOGOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS.

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Fabiano Silva.

Curitiba PR
2018

Ficha catalográfica

Substituir o arquivo 0-iniciais/catalografica.pdf (PDF em formato A4) pela ficha catalográfica fornecida pela Biblioteca da UFPR a pedido da secretaria do PPGInf/ UFPR.

O conteúdo exato da ficha catalográfica é preparado pela Biblioteca Central da UFPR, a pedido da secretaria do PPGINF. Portanto, não "invente" um conteúdo para ela.

ATENÇÃO: por exigência da Biblioteca da UFPR, esta ficha deve ficar no verso da folha de rosto (que contém o nome do orientador e área de concentração). Cuide desse detalhe quando imprimir as cópias finais.

Ficha de aprovação

Substituir o arquivo 0-iniciais/aprovacao.pdf pela ficha de aprovação fornecida pela secretaria do programa, em formato PDF A4.

Dedico à minha família, em especial meus pais, Francisco e Miriam. Dedico à meus irmãos Henrique e Guilherme.

Agradecimentos

Às nossas famílias, pelo apoio em toda a nossa formação educacional e nos momentos difíceis durante esses anos.

Aos nossos amigos Wanderlan Carvalho e Alexey Papalexiou, por aceitarem desenvolver em conjunto o tema proposto para o Trabalho de Conclusão de Curso.

Aos nossos colegas de graduação, pelos incontáveis momentos de apoio e descontração durante os anos de curso.

Ao nosso orientador Fabiano Silva, pelos ensinamentos, sugestões e correções realizadas ao longo do desenvolvimento deste trabalho.

A todos os professores do curso de Ciência da Computação da Universidade Federal Paraná, pelo grande conhecimento transmitido.

À UFPR, por proporcionar a nossa formação educacional.

Resumo

Utilizando-se de técnicas de aprendizado de máquina, o presente trabalho tem como objetivo desenvolver, treinar e testar uma rede neural artificial para que ela seja capaz de, a partir de imagens, extrair características relevantes de jogos digitais.

São apresentadas as técnicas modernas para processamento de redes neurais, e como elas impactam neste projeto. A partir disto, são discutidos outros trabalhos que também fizeram uso de redes neurais e aprendizado de máquina para perseguirem objetivos parecidos ao nosso.

Para a execução deste projeto, foi projetada uma Rede Neural Convolucional utilizando a arquitetura ResNet, com algumas modificações em sua saída final. Tal rede foi treinada com uma base de dados obtida através do Internet Games Database. Primeiramente, os dados foram pré-tratados de forma a facilitar o aprendizado, e, posteriormente, a rede foi treinada com esses mesmos dados.

Após o treinamento, testes foram executados e demonstraram que a rede não obteve sucesso em aprender e extrair as características de maneira geral. No entanto, apresentou resultados regulares em dados uniformes e características isoladas. A baixa taxa de acerto se deve, principalmente, pelo grande número de variáveis de classificação e pela alta variabilidade entre jogos de múltiplas plataformas e gêneros, onde jogos de mesmas características possuem padrões de imagem muito distintos. Tais observações tornam a obtenção de características de jogos com base em imagens um problema de grande complexidade.

Palavras-chave: jogos digitais, aprendizado de máquina, deep learning, redes neurais convolucionais, CNN, imagens, single-label, multi-label.

Abstract

By using machine learning techniques, the present work aims to develop, train and test an artificial neural network, so that it is able to extract relevant characteristics of digital games from images.

First of all, the modern techniques for processing neural networks are presented, and it is explained how they impact on this project. Following that, other works that also have made use of neural networks and machine learning to met their objectives are discussed.

For the execution of this project, a Convolutional Neural Network was designed using the ResNet architecture, with modifications in its final output. The network was trained with a database obtained through the Internet Games Database. Initially, the data has been pre-processed in order for the network to have an improved training process, and, later, the network was trained with the same data that was pre-processed.

After the training, the tests performed have shown that the network was not very successful in learning and extracting characteristics in general. However, it has got average results when only using uniform data and isolated characteristics.

The low success rate is mainly due to a large number of classification variables. The high variability between games of multiple platforms and genres, where games which share the same characteristics have very different image patterns, is also a concern. Such observations make the problem addressed in this project a problem of high complexity.

Keywords: digital games, machine learning, deep learning, convolutional neural networks, CNN, images, single-label, multi-label.

Lista de Figuras

Lista de acrônimos

GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
RGB	Red, Green and Blue
RNA	Redes Neurais Artificiais
CNN	Convolutional Neural Networks
FC	Fully Connected Layer
API	Application Programming Interface
LR	Learning Rate
IGDB	Internet Games Database
JPG	Joint Photographic Experts Group
CSV	Comma-separated Values
PEGI	Pan European Game Information
SNES	Super Nintendo Entertainment System
PS1	Playstation 1
PS2	Playstation 2
PS3	Playstation 3
PS4	Playstation 4

Sumário

1 Introdução

Este capitulo apresenta a introdução ao trabalho, contendo as motivações que incentivaram o desenvolvimento da solução (seção 1.1), os desafios consequentes do problema a ser resolvido, as tecnologias utilizadas e o desenvolvimento (seção 1.2), a solução proposta e a organização deste documento (seções 1.3 e 1.4).

1.1 Motivação

Seja por meio de pinturas, brincadeiras, encenações ou outras atividades, o ser humano, desde os primórdios, procura maneiras de entreter a si e sua comunidade, trazendo com essas atividades múltiplos benefícios sociais para as áreas de saúde, comportamento social e educação [Rau04]. As atividades não possuem limites físicos, logo, com o desenvolver de novas tecnologias, há também o surgimento de novos meios de divertir e entreter, tornando o mercado de entretenimento um ambiente muito dinâmico.

O mercado de entretenimento vem em grande crescente com o avanço da tecnologia e a facilidade de acesso à Internet. No período de 2011 à 2016, atingiu uma receita de $1,9 trilhões de dólares, e *games* representam metade desta indústria multi trilionária, como mostra [Adm17]. Estima-se que, em 2018, 2,3 bilhões de jogadores ao redor do mundo terão gasto $137,9 bilhões de dólares em jogos, o que representa um aumento de 13,3% em relação ao ano anterior [Wij18].

Jogos de vídeo baseiam-se em um ou vários usuários interagindo com uma interface, e gerando respostas que são apresentadas em algum dispositivo de mídia visual (TV, telas, monitores, etc.). Muito populares, principalmente por serem de baixo custo a longo prazo, de fácil acesso e divertidos, não é de se espantar que jogos compõem 51% do mercado de entretenimento atual [Adm17]. Entretanto, com a alta variância de gráficos, jogabilidade, plataforma, gêneros e outras características, é muito difícil prever a viabilidade e até o sucesso de um jogo. Não é um caso incomum jogos que, aparentemente possuíam baixo potencial, tornarem-se estrondosos sucessos [War14] e também vice-versa [Val17], tornando o desenvolvimento de jogos uma área de alto risco. Durante os últimos anos, foram desenvolvidas várias ferramentas, metodologias e tecnologias para auxiliar o desenvolvimento de jogos e aumentar as taxas de sucesso [TKF09], [AdSP13].

O ato de classificar coisas do dia a dia, historicamente, sempre foi um trabalho humano. Porém, com a evolução das redes neurais artificiais, classificação passou a ser um trabalho passível de execução por um computador. Inspiradas na estrutura neural de organismos animais, as redes neurais artificiais trouxeram avanços significativos no campo do aprendizado de máquina. Através delas, é possível desenvolver sistemas computacionais capazes de reconhecer padrões, prever séries temporais, analisar relações entre dados e outras diversas tarefas [Gil17].

Um campo de estudo considerável e que teve uma grande expansão após a popularização das redes neurais, é o reconhecimento de imagens. Ao analisar uma imagem, é possível inferir diversas características da mesma. Por exemplo, ao verificarmos a imagem do rosto de uma

pessoa, podemos extrair a idade, o sentimento, o gênero, etc.. A extração destas características possibilita diversas análises estatísticas em cima de uma base de dados de imagens.

Durante toda a sua história, a indústria dos jogos digitais passou por várias transformações e reinvenções, com cada época tendo um estilo predominante e tendências que poderiam ditar o sucesso de um jogo. Sendo assim, surgiu-se a necessidade de extrair as características relevantes de jogos. O reconhecimento de imagens de vídeo games pode ser útil para analisar quais combinações de características podem levar um futuro jogo ao sucesso.

1.2 Desafio

A classificação de características de jogos puramente baseada em imagens envolve certos desafios. Um deles é a falta de informações sobre o problema de classificação em questão. Definir diversas características do jogo inteiro, com base em apenas uma imagem, pode ser uma tarefa complicada para uma rede neural. Um mesmo jogo pode trazer imagens contendo seções muito distintas uma da outra, causando divergências no aprendizado.

Por outro lado, há também uma imensa quantidade de dados a se analisar. A descentralização dos dados pode causar resultados não esperados. Um jogo pode estar classificado de diferentes formas, dependendo da base de dados utilizada.

A análise técnica também levanta desafios relacionados ao problema de classificação. A classificação dos gêneros de um jogo, por exemplo, é um problema de multi-classificação. Logo, qual é a melhor função de perda para usar? Além disso, qual critério utilizar para medir a acurácia da predição, vide que vários valores estarão na saída (como gênero, data de lançamento, temática, etc.)?

1.3 Proposta

Para este trabalho, foi montada uma base de dados de imagens de jogos obtida através da API do *Internet Games Database* [IGD18b]. A base de dados gerada conta com imagens de diversos jogos lançados desde 1991 até 2018. Todos os jogos possuem características definidas, como data de lançamento, gênero e nota.

O estudo de reconhecimento de imagens digitais ganhou grande impulso através do estudo das redes neurais convolucionais e sua execução em GPUs. As Redes Neurais Convolucionais e suas diversas técnicas possibilitam ao computador realizar análises importantes em imagens. E, em vantagem, devido à natureza numérica da análise, o computador pode detectar correlações entre os dados que um ser humano não seria capaz. Visto isso, nesse projeto utilizamos uma arquitetura moderna de Rede Neural Convolucional, levemente modificada, a fim de extrair características de um jogo com base em suas imagens.

Este trabalho é complementar ao trabalho Predição de Desempenho em Avaliação de Jogos Digitais Utilizando Redes Long Short-Term Memory [[WA18]], que possui a proposta de extrair características de jogos com base em processamento de linguagens naturais. A extração de características, realizada por ambos trabalhos, permite a análise de viabilidade de sucesso para um jogo em questão.

1.4 Organização do documento

Primeiramente, serão apresentadas e explicadas as tecnologias e conhecimentos necessários para o desenvolvimento de uma rede neural convolucional. Depois, há breves resumos e análises de trabalhos correlacionados ao nosso. Após isso, será apresentado como a nossa rede neural foi construída, incluindo a geração e tratamento dos dados para treino. Por fim, há uma análise dos resultados obtidos.

2 Fundamentação teórica

Para esta seção, nos aprofundaremos na Representação de Imagens Digitais, nas técnicas pilares para classificação de imagens e no estudo das Redes Neurais Convolucionais, a fim de demonstrar o alicerce deste projeto.

2.1 O Problema da Classificação de Imagens

Dado um conjunto fechado contendo N classes, o problema da classificação nasce ao ser requerido identificar a qual classe $K \in N$ uma determinada imagem pertence, como mostrado na figura 2.1. Para um humano, a tarefa é trivial em muitos casos. No entanto, como transferir o conhecimento para um computador, cujo núcleo é composto de operações lógicas e aritméticas?

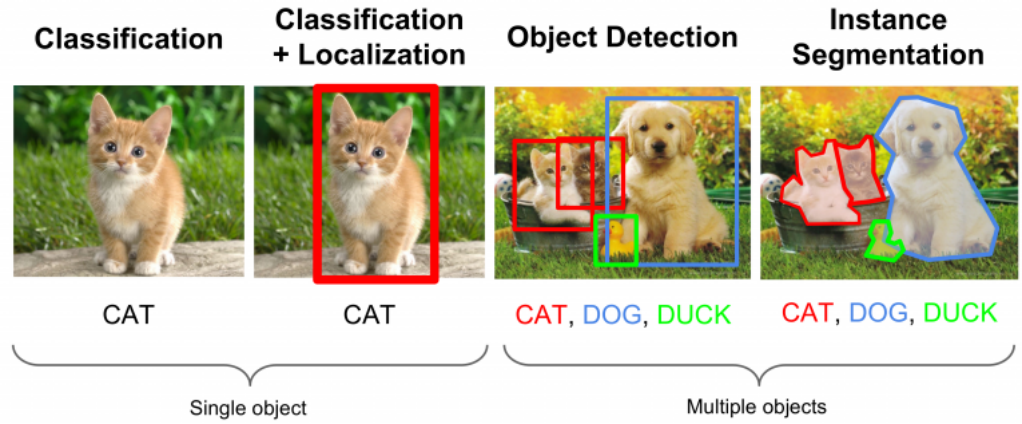


Figura 2.1: Exemplo de classificador de imagem
Fonte: Analytics India Mag.

O estudo da classificação de imagens é um dos pilares do desenvolvimento das Redes Neurais, e trouxe mudanças significativas nas maneiras e hábitos do relacionamento entre seres humanos e tecnologia. Seus usos envolvem reconhecimento facial (figura 2.2), segurança, busca através de imagens, descrição automática de imagens (*image captioning*) e até mesmo desenvolvimento de carros autônomos.

Além disso, o problema da classificação de imagens também representa vários desafios comuns no campo da visão computacional, como variação intraclasse, oclusão, deformação, variação de escala, variação de ponto de vista e iluminação [Pad16].

Figura 2.2: O reconhecimento facial é de grande uso nas redes sociais
Fonte: Tech Crunch.

2.2 Representação de Imagens Digitais

Imagem é a representação de uma pessoa ou coisa [Aur18]. Uma imagem digital é uma imagem gerada através de um computador. A geração e representação de imagens em um computador pertence à área de Computação Gráfica.

As imagens digitais representam uma das utilizações mais comuns de vetores e matrizes de dados na computação. Uma matriz bidimensional é um vetor que contém duas ou mais linhas. Por exemplo, uma matriz de dimensões 3x3:

$$M = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Por outro lado, uma matriz tridimensional pode ser definida como sendo uma matriz com mais de uma camada de profundidade. Um exemplo de matriz de dimensões 3x3x3 está abaixo, na figura 2.3:

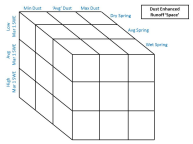


Figura 2.3: Matriz tridimensional
Fonte: Codos.

Para retratar uma imagem em um computador, são utilizados *pixels*. *Pixels* são valores numéricos presentes em cada campo de uma matriz, empregados para definir a intensidade da cor no campo em questão, com base em uma escala pré-definida.

O sistema de cores mais comum é o sistema RGB, abreviatura de *Red, Green and Blue*. Neste sistema, é utilizada uma matriz de dimensões $i * j * 3$ para representar a imagem, onde cada camada da terceira dimensão representa os valores de intensidade para as cores vermelho, verde e azul, respectivamente (figura 2.4). A mistura das três cores traz a cor final do pixel $i * j$ em questão, exemplificado pela figura 2.5.

A representação numérica padronizada para o sistema RGB é a escala de 0 a 255 (valor decimal para 1 *byte*), onde 0 é o valor mínimo e 255 é o valor máximo. Nesse caso, por exemplo, para representar o azul intenso em um pixel, deve ser definido o valor máximo na camada que representa a cor azul (camada 2), e o valor mínimo para as outras. Assim, seriam utilizadas as tuplas $T(i, j, 0) = 0$, $T(i, j, 1) = 0$ e $T(i, j, 2) = 255$.

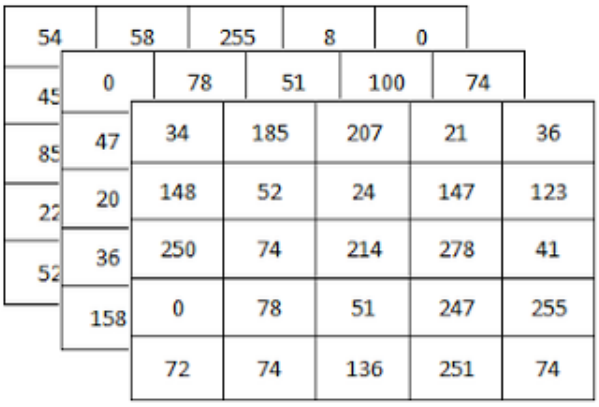


Figura 2.4: Exemplo de imagem RGB visualizada em forma de matriz
Fonte: BogoToBogo

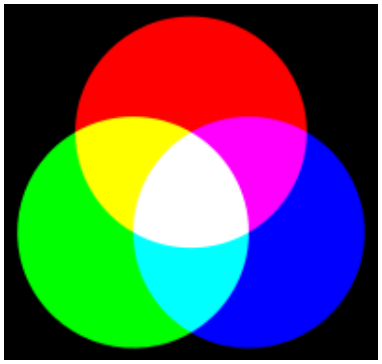


Figura 2.5: Mistura das cores Vermelho, Verde e Azul
Fonte: Wikipedia

2.3 Classificação Linear

Vamos supor que queiramos classificar a qual classe, de um conjunto C de imagens, uma imagem pertence, tendo K classes disponíveis em C. Podemos, inicialmente, tratar a classificação de imagens como uma função linear:

$$f(x, W, b) = Wx + b \tag{2.1}$$

Onde, em, (2.1):

- x: é uma matriz (imagem). Por exemplo: uma imagem x, de dimensões $D = 32x32x3 = 3072$. Consideremos que a imagem, ao invés de estar dimensionada em uma matriz $32x32x3$, está com seus *pixels* "esticados" em uma matriz de 3072 linhas por 1 coluna $(3072x1)$.
- W: uma matriz de pesos de dimensões KxD, com W_i representando o peso da classe C_i para a classificação linear. Por exemplo: matriz de pesos W, de tamanho $5x3072$. Nesta matriz, cada linha i possui os valores de pesos para a respectiva classe C_i.
- B: um vetor opcional, de tamanho K, para auxiliar no resultado final da classificação.

Logo, o resultado de (2.1) traz o produto escalar da multiplicação de W por x, acrescido do viés b. O resultado é um vetor de tamanho K, contendo o valor de pontuação atribuído para cada classe C_i em relação a imagem x.

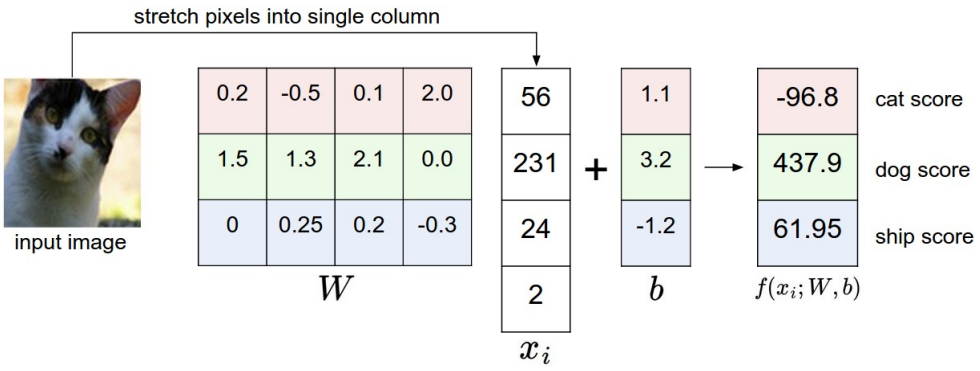


Figura 2.6: Um exemplo de mapeamento de uma imagem para pontuações de classe
Fonte: cs231 - GitHub

A classe com maior pontuação no vetor de resultado seria a escolhida pelo computador como a que representa a imagem em questão. Todo esse processo está exemplificado pela figura 2.6.

Intuitivamente, observamos que o objetivo é encontrar os melhores valores para os pesos em W, de tal forma que maximizem a pontuação da classe correta [Uni17c].

2.4 O cálculo da perda

Uma função de perda é caracterizada como uma função que, dado valores de uma ou mais variáveis, define, em um número real, o quanto os valores atribuídos nas variáveis estão em "desacordo" com os valores esperados [Wik17].

Imaginemos, que, para o exemplo da seção 2.3, fosse usado o classificador linear em uma imagem, para emitir um vetor de pontuações para K classes. A função de perda deverá emitir um valor alto (ruim) caso a pontuação para a classe correta esteja em um valor longe do esperado. Em outra situação, caso a pontuação de uma classe não esperada tenha um valor alto, o valor da perda também deverá ser alto.

Existem várias funções de perda conhecidas, como L1 Loss, L2 Loss e Cross Entropy Loss [Che17a].

2.5 Otimização

Tendo em vista o contexto das funções de classificação e perda, o objetivo é, com base nos valores obtidos através do cálculo da perda, otimizar a função de classificação de forma a obter uma perda menor. Ou seja, encontrar W em (2.1) de tal forma que W minimize a função de perda.

Na matemática, a derivada de uma função f no valor X pode ser interpretada como o quanto a função f irá mudar em respeito a variável X. Através do cálculo da derivada, obtemos o valor para "atualizar" a variável X. Como queremos ir pelo caminho negativo, ou seja, minimizar a função para convergir ao valor mínimo, aplicamos a atualização com o valor oposto do valor derivada, utilizando a seguinte fórmula:

$$X_{j+1} = X_j - f(X_j)' \tag{2.2}$$

Em (2.2), $f(X_j)'$ é a derivada de $f(X_j)$ e X_{j+1} é a variável X_j após a aplicação da atualização.

A derivada garante a mudança da função em passos extremamente pequenos, com os valores dos passos tendendo a zero. Para termos controle do quão grande o passo da atualização irá ser, é necessário introduzir um novo parâmetro, chamado taxa de aprendizado [[Bus17]]. A taxa de aprendizado é um parâmetro com valor entre o intervalo [0, 1], utilizado para controlar o tamanho da atualização:

$$X_{j+1} = X_j - (LR * f(X_j)') \tag{2.3}$$

Onde LR é a taxa de aprendizado (do inglês *Learning Rate*).

Como a variável W em (2.1) é uma matriz , não teremos apenas uma derivada, mas sim vetores de derivadas parciais, um conjunto o qual denominamos de gradiente [Wik18a]. No gradiente, há derivações parciais a respeito de cada variável da função utilizada.

2.6 Backpropagation

A técnica algorítmica fundamental para calcular a gradiente de uma função, e que forma a base de uma Rede Neural Artificial, é a técnica de *Backpropagation*. O procedimento de *Backpropagation* faz uso da regra da cadeia, apresentada no Cálculo [Aca17], para decompor uma expressão em várias expressões.

$$f(x, y, z) = (x * y) + z \tag{2.4}$$

As derivadas parciais da expressão (2.4) podem ser obtidas analiticamente, tendo como resultado o vetor de derivadas parciais $[df\,dx, df\,dy, df\,dz] = [y, x, 1]$. No entanto, o cálculo analítico é um processo custoso e demorado, recomendado, por exemplo, para medir a precisão de um resultado obtido numericamente. Ao invés disso, podemos usar a regra da cadeia e decompor (2.4) em duas expressões distintas:

$$g = (x * y) \tag{2.5}$$

$$h = g + z = f \tag{2.6}$$

As derivadas parciais de g, em (2.5), podem ser facilmente obtidas, pois, pelo cálculo, sabemos que a derivada de uma função em relação a uma variável a, quando a sozinha está sendo multiplicada por outra variável b, é b. Logo, $[dg\,dx, dg\,dy] = [y, x]$.

Para o caso de h, em (2.6) as derivadas parciais novamente são prontamente obtidas, visto que a derivada de uma função em relação a uma variável a, quando a sozinha está sendo somada a uma variável b, é 1. Assim, $[dh\,dg, dh\,dz] = [1, 1]$.

Por fim, a regra da cadeia diz que a derivada da função f, em (2.4), é calculada pela "junção" das funções decompostas g e h, através da multiplicação. Para o caso de x:

$[df\,dx] = [dh\,dg] * [dg\,dx] = 1 * y = y$.

Replicando o raciocínio para as variáveis y e z:

$[df\,dy] = [dh\,dg] * [dg\,dy] = 1 * x = x$.

$[df\,dz] = [dh\,dz] = 1$.

Portanto, chegamos ao resultado desejado, que é $[df\,dx, df\,dy, df\,dz] = [y, x, 1]$. Esse processo pode ser visualizado através de um grafo computacional. Cada rede neural possui seu

próprio grafo computacional, gerado através da técnica de *backpropagation*. Para o caso de (2.4), o processo de *backpropagation* gerou o grafo presente na figura 2.7:

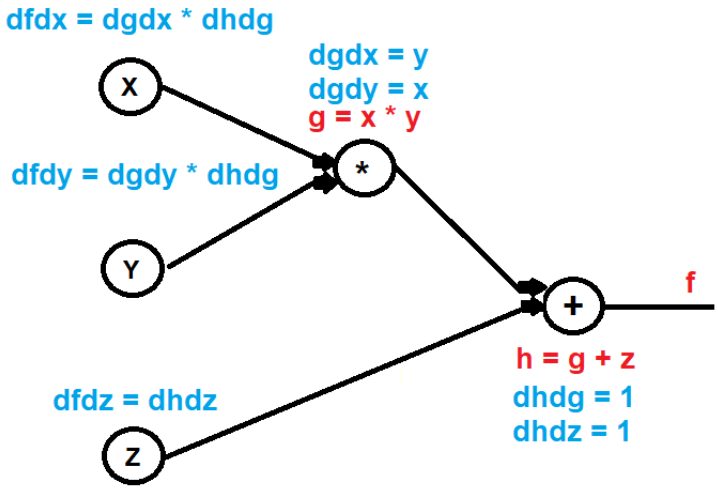


Figura 2.7: Grafo computacional da função $f(x, y, z) = (x * y) + z$

Mesmo funções complexas e extensas, como as utilizadas no processo das Redes Neurais, podem ser decompostas em várias partes, de forma a permitir o uso de *backpropagation*. Muitos padrões estão presentes no cálculo das gradientes, os quais são inseridos nos algoritmos de *backpropagation* e ajudam a otimizar o processo.

2.7 Redes Neurais

Redes Neurais Artificais, ou RNAs, são sistemas computacionais inspirados nas Redes Neurais biológicas [Gei93a]. Através do uso de neurônios artificiais, as RNAs são capazes de simular o processo de aprendizado dos seres vivos e aprender tarefas como reconhecimento de fala e classificação de imagens.

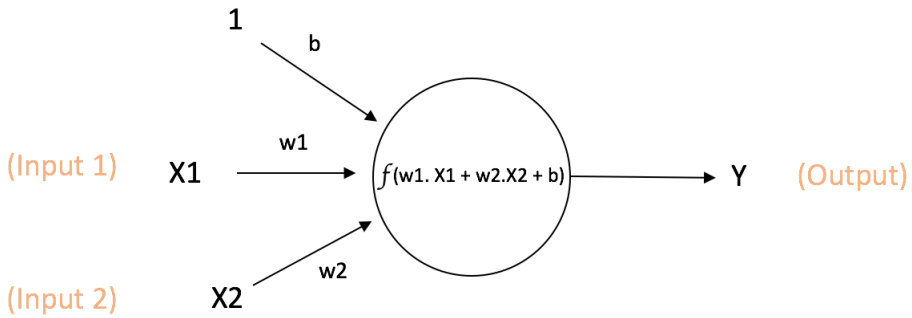


Figura 2.8: Exemplificação de um neurônio em uma RNA
Fonte: KDNuggets

Os principais componentes de uma RNA são os neurônios. Um neurônio, no contexto das RNAs, é um módulo que recebe, como entrada, informações de outros nodos (podendo ser outros neurônios). Para cada entrada, há um vetor de pesos w associado, como mostra a figura 2.8. É aplicada uma função f com o somatório dos valores de entrada multiplicados pelos seus respectivos pesos:

$$f(W_1X_1 + W_2X_2 + ... + W_nX_n) \tag{2.7}$$

A saída Y do neurônio é computada como mostrado em (2.7). A função f é não-linear e é chamada de Função de Ativação. O objetivo da função de ativação é introduzir a não linearidade na saída de um neurônio. Isso é importante devido ao fato de que a maioria dos dados do mundo real não é linear, e queremos que os neurônios aprendam essas representações não lineares [Ujj16b]. Exemplos de funções comuns para ativação são as funções *Sigmoid*, *ReLU* e *tanh* [Sha17].

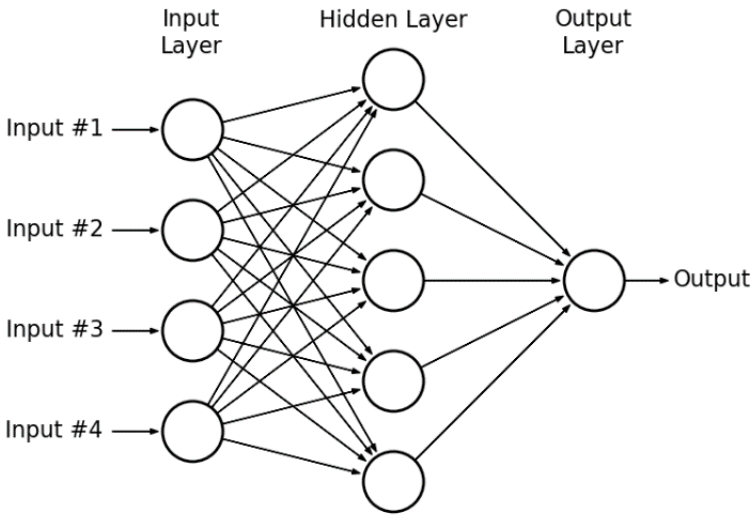


Figura 2.9: Um exemplo de RNA Feed-Forward
Fonte: ResearchGate

Uma RNA *feed-forward*, vista na figura 2.9, é composta por vários neurônios conectados, formando camadas diferentes de ativação, ou *layers*:

- O primeiro *layer* possui o trabalho de receber as entradas e passa-lás a frente, para os neurônios do próximo *layer*, sem realizar computações nas entradas.
- Os layers situados no meio são chamados de *hidden layers* e possuem a função de realizar cálculos e ativações nas informações recebidas, para assim transferir as informações consideradas úteis até a saída rede.
- O *layer* final, ou *Output Layer*, é responsável por coletar as informações geradas pelos *hidden layers*, e, através de uma função de ativação final, produzir os resultados finais da RNA.

A função de perda é aplicada nestes resultados finais, e, após isso, é realizado o *backpropagation* desde tal ponto até o início da RNA, de forma a atualizar todos os pesos W.

A forma com que os neurônios estão interligados e o número de *hidden layers* presentes caracterizam a RNA *feed-forward* como sendo do tipo *Single-layer perceptron* ou *Multi-layer perceptron* [Sch15].

2.8 Redes Neurais Convolucionais

Ao adentrarmos as áreas de Visão Computacional e Classificação de Imagens, a subcategoria de Rede Neural mais eficiente para tais assuntos é a Rede Neural Convolucional, ou *Convolutional Neural Network* (CNN) [AKH12].

2.8.1 Convolução

Uma CNN recebe como uma entrada uma matriz representando uma imagem, conforme definido na seção 2.2 deste capítulo. A partir disso, realiza algumas operações em cima dessa mesma matriz. Uma delas é a convolução, usada para extrair características úteis da imagem em questão.

O processo de convolução utiliza uma outra matriz, chamada de filtro, e de tamanho menor do que a matriz da imagem. O processo de convolução aplica o filtro em toda a matriz da imagem, "deslizando-o"de forma a cobri-lá completamente. Em cada posição aonde o filtro é aplicado, é computado o produto escalar entre valores da matriz da imagem e os valores do filtro. O resultado é adicionado em uma nova matriz, chamada de *Feature Map*.

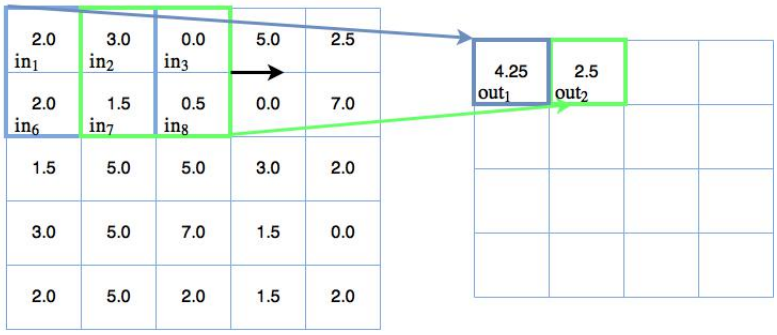


Figura 2.10: Exemplificação do processo de convolução
Fonte: Adventures in Machine Learning

Na figura 2.10, há um filtro $2x2$ sendo aplicado em uma matriz de imagem $5x5$. Nota-se que, primeiramente, foi computado o produto escalar entre os valores do filtro e os valores das posições (em azul) $[in1, in2, in6, in7]$ da matriz de imagem. Esse cálculo gerou o valor 4,25, adicionado na primeira posição, $out1$, do *feature map*, à direita na imagem. Após isso, o filtro foi "deslizado"uma posição à direita (agora em verde), e mais um produto escalar foi computado.

As propriedades matemáticas do *feature map* (como o número de dimensões) podem ser controladas através de parâmetros definidos durante a operação de convolução. Tais parâmetros incluem, por exemplo, o número de filtros que serão aplicados no decorrer da convolução e a quantidade de *pixels* que o filtro deve deslizar a cada convolução (chamado de *stride*).

Os valores contidos no filtro são inicializados de forma pré-definida ou aleatória. Durante seu processo, a CNN aprende por conta própria quais são os melhores valores para os filtros.

Ao final de um processo de convolução, é aplicada uma função não-linear em cima do *feature map* obtido, de forma a introduzir não-linearidade no processo. A função mais utilizada para esse caso é a ReLU, que substitui todos os valores negativos do *feature map* por zero. A figura 2.11 exemplifica uma imagem de entrada e a saída da mesma após passar por um filtro de convolução.

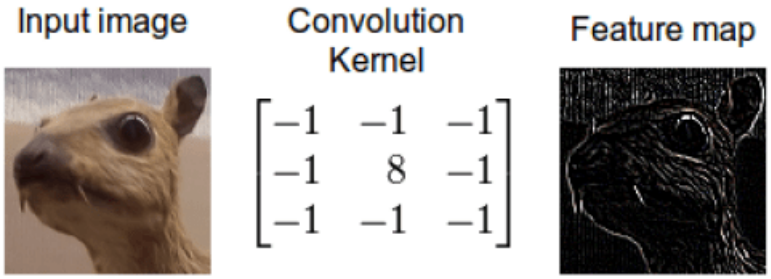


Figura 2.11: Exemplificação prática do processo de convolução
Fonte: Tim Dettmers

2.8.2 Pooling

O Pooling constitui outro processo de uma Rede Neural Convolucional. Através da técnica de pooling, é possível diminuir a dimensionalidade de um *feature map*, mas mantendo as informações mais importantes [Ujj16a].

A técnica de Pooling mais utilizada é o Max Pooling, onde é definido um tamanho espacial, e para cada janela contendo esse tamanho espacial no *feature map*, é retirado o pixel de maior valor, como visto na figura 2.12.

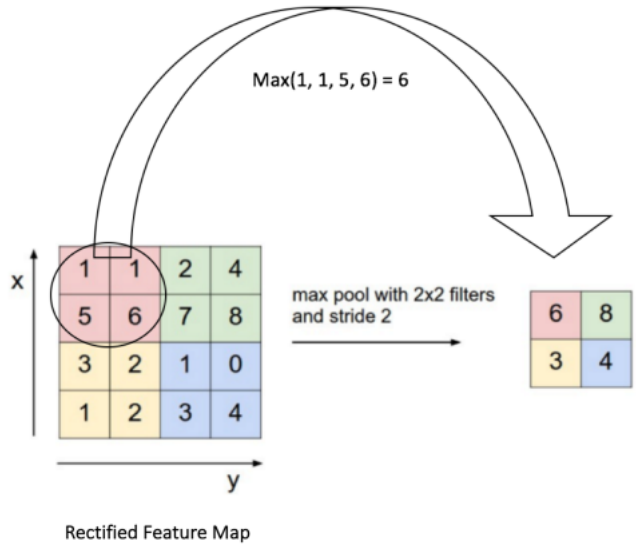


Figura 2.12: Exemplificação de max pooling com uma janela de tamanho 2x2
Fonte: The Data Science Blog

2.8.3 Etapas da Rede Neural Convolucional

A arquitetura de uma CNN utiliza os processos descritos nas seções 2.8.1 e 2.8.2, de forma a obter a classificação final para uma imagem em questão.

Na figura 2.13, é apresentada a arquitetura em alto-nível da LeNet, uma das primeiras Redes Neurais Convolucionais desenvolvidas [LeC13].

Observa-se que, após a entrada, de tamanho $32x32$, são aplicados 6 filtros convolucionais de tamanho $28x28$, obtendo 6 *feature map* de tamanho 28x28. Tais *feature map* também passaram pela função ReLU, cujo objetivo foi eliminar os valores negativos. Após isso, é aplicado o

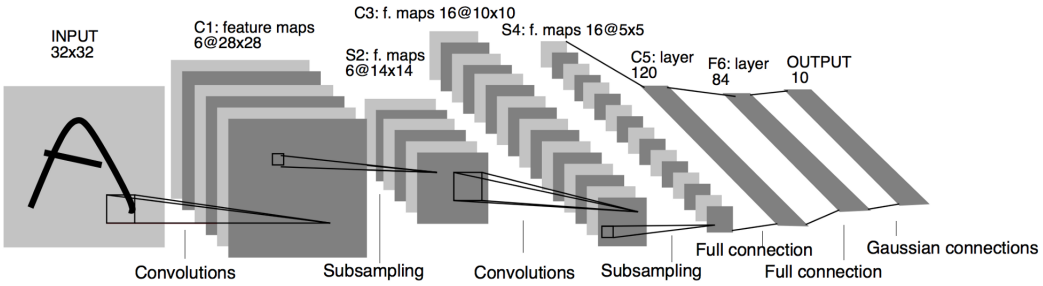


Figura 2.13: Arquitetura da LeNet
Fonte: Medium

pooling (na figura 2.13, esta etapa está representada como *subsampling*), resultando em novos 6 *feature map* de tamanho 14x14. Repetindo os processos, chegamos à etapa S4, onde há 16 *feature map* de dimensões 5x5.

A partir desta etapa, as características extraídas são passadas para um *Fully Connected Layer*, referenciado como *Full connection* na figura 2.13. Um *Fully Connected Layer* é um *Multi-layer perceptron*, conforme definido na seção 2.7 deste capítulo. O nome *Fully-Connected Layer*, ou FC, vem do fato de que todos os neurônios de cada *layer* do *Multi-layer perceptron* estão interligados com todos os neurônios do próximo *layer* [Sak18].

O FC, então, utiliza as características obtidas pelos processos anteriores para classificar a imagem de entrada em alguma das classes pertencentes ao respectivo *dataset*. No caso da LeNet, a saída da etapa S4, contendo 400 características (16 *feature map* de dimensões $5x5$), é utilizada como entrada em um FC, que gera 120 saídas. Outro FC transforma estas 120 saídas em 84 saídas. Por fim, um último FC utilizada as 84 saídas anteriores para gerar as saídas finais, contendo valores de classificação para 10 classes.

3 Revisão de Literatura

Este capítulo apresenta estudos publicados que possuam relação com este trabalho.

3.1 Predicting video game properties with deep convolutional neural networks using screenshots

O artigo [BS16] possui disponível ao público apenas uma breve página de introdução. No entanto, algumas observações notáveis são feitas. Os autores utilizaram o banco de dados da plataforma Steam para obterem informações acerca do gênero, data de lançamento, classificação etária, etc. para milhares de jogos.

É citada a dificuldade de inferir propriedades em um jogo de video-game apenas com base em uma imagem, pois ela possui poucas informações acerca do problema de classificação. Algumas irregularidades foram descobertas, como o fato de a qualidade e resolução da imagem terem importância na classificação do ano de lançamento. Por exemplo, uma imagem de um jogo recente, porém de qualidade e resolução baixa, pode ser confundida pela rede neural como a de um jogo de uma ou duas décadas atrás. O contrário também ocorre, visto que um jogo antigo, porém em uma imagem de baixa resolução, provoca o mesmo efeito do *anti-aliasing* [[Ste18]], onde os modelos parecem mais suaves e com quantidade maior de polígonos.

Também é lembrado o fato de que humanos possuem dificuldade na mesma tarefa de classificação, visto que a pessoa pode ter pouco conhecimento acerca dos jogos mostrados.

3.2 Recognizing Game Genres From Screenshots using CNNs

O trabalho Recognizing Game Genres From Screenshots using CNNs [Huw17], possui objetivos parecidos com o nosso trabalho: extrair características de jogos com base em imagens. No caso, o autor buscou reconhecer o(s) gênero(s) de um jogo com base em uma imagem. A análise levanta pontos interessantes, como qual função de perda usar e como medir o desempenho do modelo.

Para os dados de treino, o projeto utilizou-se de capturas de tela de jogos da loja virtual Steam, totalizando cerca de 12.000 imagens. Juntamente, há os gêneros associados a cada jogo. Como um jogo pode ter mais de um gênero, isso caracteriza um problema *multi-label* [Wik18b].

Por não se tratar de um problema *single-label*, não seria possível utilizar uma função de ativação comumente usada para a camada final, como a *Softmax*. A mesma produz como saída da rede um valor de probabilidade para cada *label*, que somados totalizam 1. Logo, o rótulo com maior probabilidade é o único rótulo de saída da rede. Para contornar tal limitação, o autor utilizou a função *Sigmoid*, que emite um vetor de probabilidades independentes como saída da rede neural. Além disso, técnicas de estado-da-arte comuns em redes neurais modernas, como *Dropout* e *Binary Crossentropy Loss*, fazem-se presentes.

Seja L o número de vezes que um *label* foi adivinhado corretamente, Q a quantidade de vezes que o *label* aparece no *dataset*, e P a quantidade de vezes em que o *label* foi ativado pela rede neural. Para o cálculo do desempenho, foram introduzidas duas variáveis:

$$Recall = \frac{L}{Q} \tag{3.1}$$

$$Precision = \frac{L}{P} \tag{3.2}$$

Utilizando em conjunto as variáveis em (3.1) e (3.2), é feito o cálculo final da performance, chamado de $F1$. O $F1$ *score* é obtido utilizando a média harmônica das duas variáveis anteriores:

$$F1 = \frac{2 * Recall * Precision}{(Recall + Precision)} \tag{3.3}$$

Os resultados finais foram satisfatórios, com o modelo obtendo um $F1$ *score* final de 0,522. Analisando especificamente cada gênero e seu respectivo *score*, alguns gêneros obtiveram ótimas performances para a classificação, equivalentes a humanos. No entanto, devido a natureza multi-label da classificação, gêneros que aparecem poucas vezes no dataset se tornam inerentemente mais difíceis de obter um bom $F1$ score, e possuíram performances ruins.

3.3 Machine Learning for Predicting Success of Video Games

Embora não trate de classificação de imagens, esta publicação [Trn17] realiza várias análises, em especial no tratamento de dados. Tais análises nos ajudaram a montar a base para o nosso trabalho. Inicialmente, o autor discute o ponto de o que e como algo pode influenciar no futuro sucesso de um jogo de videogame. Alguns dos pontos levantados são:

- Número de cliques em motores de busca.
- Análises e informativos publicados pela mídia.
- Serviços de streaming de mídia.
- Discussão em fóruns e redes sociais.

Para a montagem do banco de dados, foram utilizados dados da API da Steam (como preço, gêneros, desenvolvedores, etc.). Porém, como dados de vendas não são disponibilizados pela Steam, foram utilizados como substitutos o número de jogadores concorrentes em um jogo, fornecido pelo Steam Charts, e o número de donos de cada de jogo, fornecido pelo Steam Spy.

Para medir o sucesso de um jogo, o autor classificou "sucesso" como o número de jogadores que possuem o jogo após dois meses de lançamento do mesmo, e a média do número de jogadores concorrentes também após dois meses de lançamento do jogo.

Foi realizada uma filtragem nos dados obtidos, para evitar ruídos durante o treinamento:

- Jogos sem dados de preço e jogos marcados como Early Access e Free to Play foram removidos do *dataset*.
- Somente jogos publicados na loja a partir de setembro de 2013 em diante foram considerados, pois nesse período a maneira como os jogos podiam ser publicados sofreu mudanças significativas.

- Alguns gêneros, pouco presentes, foram removidos. Ao final, os seguintes gêneros estiveram presentes: *Strategy, Adventure, Action, Simulation, Racing, Casual, Sports, Massively Multiplayer, Education, Indie*.

- Novos atributos foram adicionados: sequência, indicando se um jogo é continuação de outro; customizável, indicando se o jogo é aberto para customizações pela comunidade.

Ao final, 4.267 jogos estiveram presentes no *dataset*. Análises estatísticas em cima desses 4.267 itens revelaram características interessantes:

- Preço é um fator decisivo.

- Jogos com gráficos mais avançados e requisitos complexos de *hardware* são, geralmente, mais bem sucedidos.

- Poucos jogos alcançam popularidade alta. A maioria dos jogos possui menos de 10 mil jogadores.

- Na média, o segundo jogo desenvolvido por uma desenvolvedora vende melhor do que o primeiro.

Para os experimentos, o *dataset* foi dividido em 60% treino, 20% validação e 20% teste. Quatro cenários foram criados para a avaliação do modelo:

- Regressão - predizer um valor numérico exato de jogadores para um jogo.

- Três classes binárias: (1, -), indicando jogos que pelo menos alcançaram algum público; (10, -), indicando jogos que obtiveram mais de 10 jogadores; (100, -), indicando jogos que obtiveram mais de 100 jogadores.

Algoritmos RPART (Recursive Partitioning), Random Forest, Gaussian Process e SVM (Support Vector Machine) [Le16] tiveram aplicações nos experimentos. Para a tarefa de regressão, o algoritmo de Random Forest gerou as melhores predições: cerca de 50% dos resultados previstos estavam corretos, com um desvio de no máximo 1, e cerca de 92% com um desvio de no máximo 3.

Nas tarefas envolvendo as classes binárias, o algoritmo de Random Forest também se saiu melhor: para as três classes binárias, o $F1$ *score* foi de 67%, 53% e 64%, respectivamente.

Em conclusão, os resultados obtidos foram regulares. O autor cita que o fator mais determinante no sucesso dos jogos é o fato de sua desenvolvedora já possuir experiência com jogos anteriores. Também conclui que, embora o mercado de jogos esteja em crescimento contínuo, os estudos sobre previsão de sucesso em jogos ainda são escassos.

4 Solução proposta

Nossa solução é composta por uma Rede Neural Convolucional ajustada e treinada em uma base de dados customizada. Tal solução será descrita aqui. O capítulo está dividido em 5 seções, de forma a explicar o trabalho desde a obtenção dos dados até o treinamento final.

4.1 Obtenção dos dados

Todas as imagens e dados foram obtidos pela plataforma Internet Games Database. A IGDB é uma plataforma online cujo intuito é centralizar todas informações relevantes sobre jogos em um só lugar, e, a partir dessas informações, construir recursos sociais e exploratórios [IGD18a]. O site apresenta, para cada jogo, diversas informações, como: nota, gêneros, data de lançamento, análises oficiais da crítica, console, etc..

O banco de dados da plataforma conta com dados sobre 188.939 jogos, e possui 265.285 imagens armazenadas. Para manipulação e extração desses dados, é disponibilizada a API IGDB API [IGD18b]. A IGBD API possui um plano de licença gratuito, que foi utilizado por este trabalho. Tal plano permite uso comercial e acadêmico da aplicação.

Para a obtenção dos dados, desenvolvemos um script na linguagem Python que utiliza as funções da IGDB API, de forma a obter dados e imagens de jogos específicos filtrados por nós. A decisão foi obter dados de todos os jogos das seguintes plataformas: *Super Nintendo*, *Playstation*, *Playstation 2*, *Playstation 3* e *Playstation 4*. O motivo é que tais consoles foram plataformas de grande sucesso em suas respectivas gerações, com uma vasta biblioteca de jogos. Além disso, nos dá uma boa distribuição de jogos em relação à data de lançamento, pois contém jogos desde 1990 até 2018. Logo, há jogos dos mais variados estilos e tendências que ditaram as diferentes épocas.

Para cada requisição de jogo realizada pelo *script*, eram transferidos os seguintes dados pela API:

- Nome
- Data de lançamento (no formato *UNIX Epoch Date*)
- Lista de gêneros
- Lista de temas
- Média da nota dos usuários (*rating*)
- Número de usuários que deram nota
- Modos de jogo
- Visão/perspectiva do jogo

- Classificação etária (*PEGI*)
- Todas as *screenshots* do jogo presentes no site, no formato JPG e em resolução 555 x 312. Cada jogo possui entre 1 a 6 *screenshots*.

As imagens foram salvas separadamente, em uma pasta específica. O restante dos dados foi salvo em um arquivo CSV, como mostra a figura 4.1. Cada linha representa um jogo, separando o tipo do dado por coluna. Há uma coluna *screenshots* para cada jogo, que contém uma lista com os nomes dos arquivos das imagens referentes ao jogo em questão.

	A	B	C	D	E	F	
1	name	first_release_date	genres	themes	rating	rating_count	g
2	Wing Commander	631152000000	['Simulator']	['Action', 'Science fiction']	80.0	8	['S
3	ACA NEOGEO NAM-1975	633312000000	['Shooter', 'Arcade']	['Action', 'Warfare']	63.0	0	['S
4	Metal Gear 2: Solid Snake	648432000000	['Adventure']	['Action', 'Horror', 'Stealth']	84.82215341862441	17	['S
5	Raiden	652147200000	['Shooter', 'Arcade']	['Action', 'Science fiction']	79.7650666276352	8	['S
6	Mega Man 3	652752000000	['Platform']	['Action', 'Science fiction']	81.6253469186386	42	['S
7	F-Zero	659145600000	['Racing']	['Science fiction']	81.44195568399735	33	['S
8	Super Mario World	659145600000	['Platform']	['Action']	95.86025561390684	436	['S
9	Dragon's Lair	660009600000	['Platform', 'Adventure']	['Fantasy', 'Historical']	55.0	2	['S
10	ActRaiser	661305600000	['Fighting', 'Simulator', 'Strategy']	['Action', 'Fantasy']	80.9079482168462	12	['S
11	Pilotwings	661737600000	['Simulator', 'Sport']	['Warfare', 'Action']	39.6222143896071	9	['S
12	Pit Fighter	662601600000	['Fighting', "Hack and slash/Beat '[	['Horror']	50.22665365936091	7	['S
13	Alien Storm	662601600000	['Shooter', "Hack and slash/Beat '[	['Action', 'Science fiction']	69.813395361658	6	['S
14	Ultima VI: The False Prophet	662601600000	['Role-playing (RPG)']	['Fantasy']	87.0958300482104	4	['S

Figura 4.1: Arquivo CSV contendo os dados do experimento

4.2 Tratamento dos dados

Inicialmente, foram realizados ajustes com base em dados faltantes ou dados duplicados. As regras foram as seguintes:

- Retirados jogos que não possuíam imagens.
- Retirados jogos que não possuíam dados sobre data de lançamento.
- Retirados jogos que não possuíam dados sobre gêneros.
- Retirados jogos que não possuíam dados sobre modos de jogo.
- Retirados jogos que vieram com dados inválidos.
- Retirados jogos que apareciam mais de uma vez.
- Retirados jogos que possuíam *collection*, *HD*, *edition*, *DLC* ou *remaster* no nome, por não serem jogos originais.
- Retirados jogos que não possuíam dados de *rating* nem tema.
- Retirados jogos que nao possuíam dados de tema nem perspectiva.
- Retirados jogos que nao possuiam dados de tema nem modos de jogo.
- Retirados jogos que ainda não foram lançados.
- Retirados jogos que foram lançados antes de 1990.

Para as imagens, foram retiradas todas imagens que não demonstravam conteúdo *in-game*. Ou seja, imagens que eram de menus, telas-título, seleção de personagens e afins, foram removidas.

Mesmo com os filtros aplicados acima, algumas entradas ainda continham dados faltantes (*Missing Data*). Conduzimos as seguintes estratégias para estes casos:

- Campos númericos faltantes: substituídos pela média aritmética das entradas com o campo presente.
- Campos categóricos faltantes: substituídos pelo valor mais comum.
- Em alguns casos, os valores faltantes foram inseridos manualmente, com base em conhecimento prévio e pesquisas na internet.

Devido a grande variedade de dados para a classificação, tornou-se necessária uma simplificação das categorias, de forma a otimizar o desempenho da rede neural. Abaixo seguem as simplificações realizadas:

- A data de lançamento, que continha um *range* de 29 valores (1990 até 2018), teve mudanças de forma que fosse possível simplifica-lá para um *range* contendo 5 valores. Abaixo seguem as regras aplicadas:
  - Se a data de lançamento for menor ou igual a 1995, a data de lançamento para tais jogos recebe o valor 0, significando que são jogos da geração SNES.
  - Se a data de lançamento for menor ou igual a 2000, porém maior do que 1995, a data de lançamento para tais jogos recebe o valor 1, significando que são jogos da geração PS1.
  - Se a data de lançamento for menor ou igual a 2006, porém maior do que 2000, a data de lançamento para tais jogos recebe o valor 2, significando que são jogos da geração PS2.
  - Se a data de lançamento for menor ou igual a 2013, porém maior do que 2006, a data de lançamento para tais jogos recebe o valor 3, significando que são jogos da geração PS3.
  - Se a data de lançamento for maior do que 2013, a data de lançamento para tais jogos recebe o valor 4, significando que são jogos da geração PS4.

  Jogos que eram de uma geração, porém suas datas estavam fora do *range* definido, tiveram suas datas alteradas manualmente para adentrarem ao *range*. Este processo foi feito de forma manual, com cerca de 60 jogos.
- A lista de gêneros, que inicialmente continha 20 gêneros, teve quatro gêneros removidos pelo fato de aparecerem muito pouco na base de dados: *Pinball*, *Quiz/Trivia*, *Point-and-click* e *Music*. O gênero *Indie* também foi removido, visto que é uma tarefa subjetiva definir se um determinado jogo é do gênero *indie*. Os jogos que continham tais gêneros permaneceram no *dataset*, porém os gêneros foram removidos dos mesmos.
- Os gêneros *Real Time Strategy (RTS), Strategy, Turn-based strategy (TBS)* e *Tactical* foram transformados em apenas um: *Strategy*. Ao total, 12 gêneros continuaram presentes, que são: *Fighting, Shooter, Platform, Puzzle, Racing, Role-playing (RPG), Simulator, Sport, Strategy, Hack and slash/Beat'em up, Adventure e Arcade*.

- A lista de temas passou pela mesma transformação da lista de gêneros, tendo os temas *Thriller, Business, Drama, Historical, Comedy, Educational, Kids, 4X (explore, expand, exploit, and exterminate), Erotic, Mystery, Open world, Warfare e Party* removidos. Os gêneros *Survival* e *Horror* foram fundidos para apenas *Horror*. Ao total, se fazem presentes 7 temas, que são: *Action, Fantasy, Science fiction, Horror, Stealth, Non-fiction, Sandbox*.

- Inicialmente, o *Rating* apresentava-se em uma escala de 0 a 100. O *Rating* foi dividido por 10 para transformá-lo em uma escala de 0 a 10. Após isso, foi convertido para uma escala de 1 a 4, através das seguintes transformações:

  – Se o *Rating* do jogo for menor ou igual a 4.9, o *Rating* para tais jogos recebe o valor 0, significando que possuem nota 1/4.

  – Se o *Rating* do jogo for menor ou igual a 6.9, porém maior do que 4.9, o *Rating* para tais jogos recebe o valor 1, significando que possuem nota 2/4.

  – Se o *Rating* do jogo for menor ou igual a 8.6, porém maior do que 6.9, o *Rating* para tais jogos recebe o valor 2, significando que possuem nota 3/4.

  – Se o *Rating* do jogo for maior do que 8.6, o *Rating* para tais jogos recebe o valor 3, significando que possuem nota 4/4.

- Os dados sobre número de usuários que deram nota, modos de jogo, visões/perspectivas de jogo e classificação etária foram descartados para essa versão do projeto, podendo ser utilizados em versões futuras.

Em resumo, os dados analisados e que serão classificados são os seguintes, para cada jogo:

- Data de lançamento, contendo 5 classes.

- Gêneros, contendo 12 classes.

- Temas, contendo 7 classes.

- *Rating*, contendo 4 classes.

Até esse momento, há, ao total, 3249 jogos listados e 22.626 arquivos de imagem.

4.2.1 Data Augmentation

Após experimentações com o *dataset* formado, notou-se que havia uma baixa variância de dados. Isso era devido, principalmente, ao fato de que cerca de 80% das imagens pertencia a jogos das gerações PS2, PS3 e PS4, e poucas imagens representavam as gerações SNES e PS1, o que causava desbalanceamento no *dataset*. O desbalanceamento resultava em um treinamento enviesado e com alta perda. Portanto, tornou-se necessária a aplicação de técnicas de *Data Augmentation*.

É um fato observado, que, quantos mais dados efetivos para análise um algoritmo de *Deep Learning* possui, melhor tende a ser o seu desempenho. Visto isso, muitas vezes possuímos um *dataset* não suficientemente grande para treinar efetivamente uma Rede Neural. Porém, podemos obter "novos"dados a partir do nosso próprio *dataset*, em um processo chamado de *Data Augmentation*. Como uma rede neural não sabe distinguir uma imagem da outra, podemos aplicar

transformações em uma mesma imagem, como rotação, translação, cortes, redimensionamento, etc., que o modelo as interpretará como diferentes imagens.

Em nosso trabalho, aplicamos técnicas de transformação de imagens em todas as imagens presentes no dataset, utilizando a biblioteca PIL da linguagem Python. As tranformações em si e as regras para a aplicação estão descritas abaixo:

- Para todas as imagens de jogos da geração SNES, foram geradas, a a partir das imagens originais:
  - Imagens rotacionadas aleatoriamente por um *range* entre 20 e 180 graus.
  - Imagens rotacionadas aleatoriamente por um *range* entre 200 e 360 graus.
  - Imagens com o brilho aumentado por um valor fixo.
  - Imagens com o contrate diminuído por um valor fixo.
- Para imagens de jogos da geração PS1, foram geradas:
  - Imagens rotacionadas aleatoriamente por um *range* entre 20 e 180 graus.
  - Imagens rotacionadas aleatoriamente por um *range* entre 20 e 180 graus.
  - Imagens com o brilho aumentado por um valor fixo.
- Para imagens de jogos da geração PS2, foram geradas:
  - Imagens rotacionadas aleatoriamente por um *range* entre 20 e 180 graus.
- Para todas imagens do dataset, foi gerada sua versão refletida, ou seja, um *Flip Left-Right* (figura 4.2).

(a) Imagem original (b) Imagem transformada

Figura 4.2: Uma imagem do *dataset* antes e após a transposição aplicada

Por fim, após o *Data Augmentation*, o *dataset* definitivo contém 3249 jogos e cerca de 40 mil arquivos de imagem.

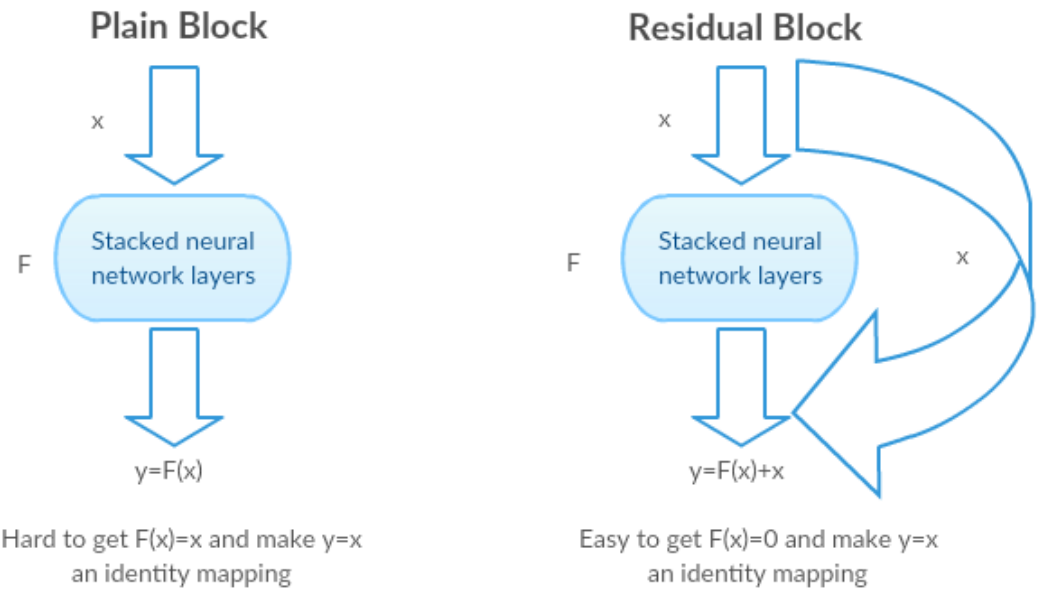
4.3 Arquitetura

4.3.1 ResNet

A arquitetura de rede neural utilizada foi a *ResNet* [KH18], por ser um modelo do estado-da-arte das redes neurais e por obter um dos melhores desempenhos em uma das maiores competições de detecção de imagens, a *Image Large Scale Visual Recognition Challenge* [Lab18].

Em um breve resumo, a *ResNet*, abreviação de *Residual Network*, busca solucionar o problema das *Vanishing / Exploding Gradients* [Nie18]. Quando uma rede neural possui muitas camadas de profundidade, o gradiente pode zerar ou explodir (tornar-se muito grande) durante o *backpropagation*, visto que será multiplicado diversas vezes, e, caso os fatores sejam números muito pequenos ou muito grandes, o problema tende a ocorrer.

A ideia é que, em camadas mais profundas da rede, a identidade x dos pesos calculados pela Rede Neural seja sempre passada adiante (figura 4.3(b)). Juntamente com a identidade, também é calculada a função residual $F(x)$, que verifica se é necessário mudanças no valor provido pela identidade x. Logo, a função de passagem pro próximo *layer* se dá por:



(a) Bloco padrão (b) Bloco residual

Figura 4.3: Comparação entre bloco padrão e bloco residual
Fonte: Medium

$$H(x) = F(x) + x \tag{4.1}$$

Nota-se que $H(x)$ (ou Y) é de fácil otimização, visto que, caso seja necessário passar apenas a identidade x adiante, será necessário somente tornar $F(x) = 0$. Com o uso de blocos residuais, evita-se a perda do gradiente, pois a otimização será feita em $F(x)$, que é apenas o resíduo.

No modelo da Resnet referenciado na figura 4.4, observa-se que o mesmo é composto de 34 *layers* empilhados. Com exceção do *layer* inicial $7x7$, todos os outros realizam convoluções $3x3$ utilizando blocos residuais (sinalização à direita de cada bloco). Ao final, há um *Fully Connected Layer (FC 1000)*, que transforma as convoluções em uma saída com 1000 *features*.

4.3.2 Modificações na arquitetura da ResNet

O propósito deste trabalho é treinar uma rede neural que emita saídas para quatro problemas de classificação (data de lançamento, gêneros, temas e *rating*). Logo, para adaptar a *ResNet* a este trabalho, foram necessárias modificações em seus *layers* finais.

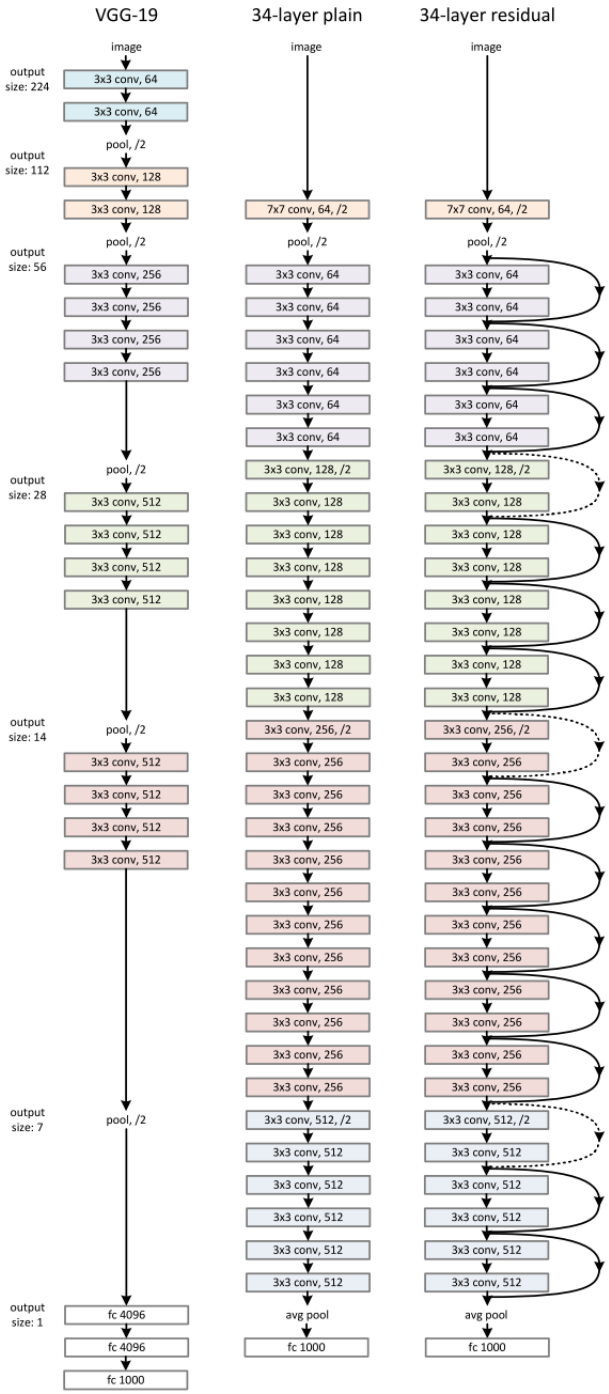


Figura 4.4: Camadas da ResNet
Fonte: Kaggle

Ao final das convoluções, a saída original se dava em um *Fully Connected Layer* com 1000 *features*. Para nosso caso, adicionamos mais quatro *Fully Connected Layer*, um para cada classificação citada acima. Logo, a saída do *FC 1000* descrito anteriormente serve de entrada para os novos quatro *FCs*:

- *Date Fully Connected Layer*, que emite 4 saídas. Usado para a classificação da data de lançamento.

- *Genre Fully Connected Layer*, que emite 12 saídas. Usado para a classificação de gêneros.
- *Theme Fully Connected Layer*, que emite 7 saídas. Usado para a classificação de temas.
- *Rating Fully Connected Layer*, que emite 4 saídas. Usado para a classificação de rating.

A figura 4.5 é uma representação visual dos quatro novos *layers* adicionados.

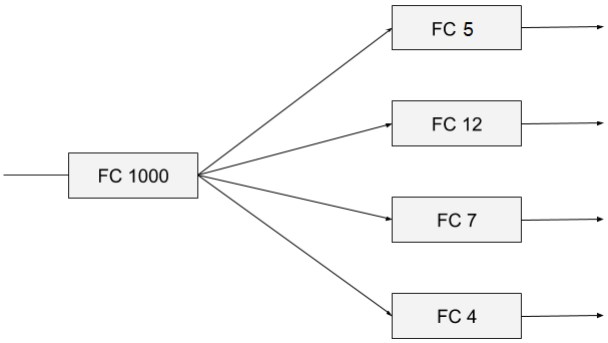


Figura 4.5: Os quatro *layers* finais

4.3.3 Ativação final

Conforme descrito no seção 2.7 do capítulo 2, sobre Redes Neurais, a saída final de uma Rede Neural é dada por uma função de ativação, que é aplicada sobre os valores providos pelo Fully Connected Layer final. Nesta seção, serão descritas as funções de ativação para cada um dos quatro *FCs* citados anteriormente, na seção 4.3.2.

Para o *Date Fully Connected Layer* e o *Rating Fully Connected Layer*, estamos com a classificação de 5 classes mutuamente exclusivas e 4 classes mutuamente exclusivas, respectivamente. Isso significa que são problemas *single-class*: a saída deve ser apenas uma para cada *FC*. Visto isso, utilizamos a normalização *softmax*:

$$softmax(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{4.2}$$

A função Softmax transforma a saída em uma distribuição de probabilidades, tornando os resultados mais intuitivos e de fácil interpretação.

Na equação (4.2), z é o vetor de saídas do *FC*. Conforme a fórmula, cada *score* provido pelo *FC* é normalizado pela média exponencial de todos os *scores*, transformando-se em um valor de uma distribuição de probabilidades. A soma de todas as probabilidades emitidas resulta no valor 1. E, como saída final da rede neural, coletamos a classe que possuir maior valor de probabilidade.

No caso dos *FCs Genre Fully Connected Layer* e *Theme Fully Connected Layer*, há um problema de classificação *multi-class*. Cada *FC* emite 12 e 7 saídas, respectivamente, e a função de ativação deve fornecer como saída as classes que considerar corretas, seja 1 ou N classes. A função *sigmoid* possui aplicação para esse caso:

$$sigmoid(z)_j = \frac{1}{1 + e^{-z_j}} \tag{4.3}$$

A função (4.3) emite um valor no intervalo [0,1] para cada valor de z. Intuitivamente, é possível notar que os *labels* verdadeiros devem ter um valor obtido pela *sigmoid* próximo a 1, e, os valores falsos, próximo a 0.

Nota-se que são valores independentes uns dos outros e não somam 1. Dado um número de *threshold*, é possível definir que as classes com um *score* acima do *threshold* serão coletadas como resultado. Por exemplo, utilizando o valor 0.6 como *threshold*, todos os valores emitidos pelo *FC* que, ao passarem pela função *sigmoid*, obterem resultado igual ou superior a 0.6, terão suas respectivas classes ativadas e emitidas como saída de rede neural.

Obter um valor de *threshold* ótimo, que maximize os resultados corretos da rede neural, é um problema que expande para diversos casos de estudo, como em [FL18]. Nosso valor de *threshold* foi definido para 0.55, obtido através de experimentos empíricos com os dados de treino e teste.

Na figura 4.6, é demonstrado qual função de ativação é utilizada para cada *layer* de saída.

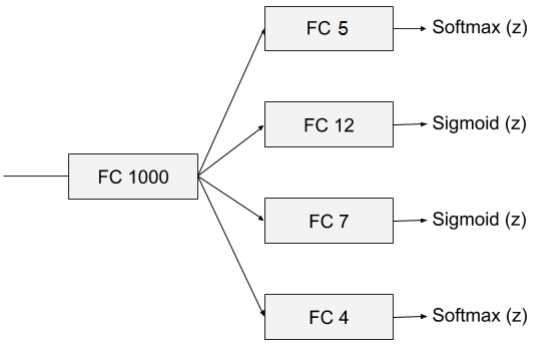


Figura 4.6: As ativações nos quatro layers finais

4.3.4 Perda

O cálculo da perda, visto na seção 2.4 do capítulo 2, também caracteriza uma importante etapa do treinamento de uma rede neural. Como possuímos quatro *layers* de saída, é necessário adaptar o cálculo da perda para que englobe os resultados dos quatro *layers*, e para que também seja possível realizar o *backpropagation* de forma efetiva.

Novamente, há uma função para os *layers* que tratam de classificação *single-label* e outra para os que tratam de classificação *multi-label*.

A função de perda *Negative Log Likelihood Loss*, ou *NLLLoss*, utiliza a saída da função 4.2, descrita na sub-seção anterior. Dada a saída Z da função *Softmax* e a classe correta C da classificação, é feito o seguinte cálculo envolvendo o valor de probabilidade emitido por (4.2) para a classe correta C:

$$NLLL(C) = -\log(Z_{[C]}) \tag{4.4}$$

(4.4) é utilizado para o cálculo da perda dos *layers single-label*. Para os *layers multi-label*, observa-se que cada classe terá apenas duas possibilidades: estar ou não presente no vetor de resultados final. Logo, se trata de uma classificação binária.

Na classificação binária, dado o valor de probabilidade Z_i obtido por (4.3) para cada classe C_i, a *Binary Cross-Entropy Loss* pode ser calculada por:

$$BCELoss(C_i) = -(y \log(Z_{[C_i]}) + (1 - y) \log(1 - Z_{[C_i]})) \tag{4.5}$$

Onde, em (4.5), y é um indicador binário (0 ou 1) indicando se a classe C_i deve estar presente ou não na saída da classificação. Somando-se a *BCELoss* de cada classe C_i, obtem-se o valor final da perda para os *layers* de classificação multi-label [Che17b].

Para obter a perda final da rede, foram somadas as perdas dos quatro *layers* finais, e, por fim, realizado o *backpropagation* a partir desse resultado final, como demonstrado na figura 4.7.

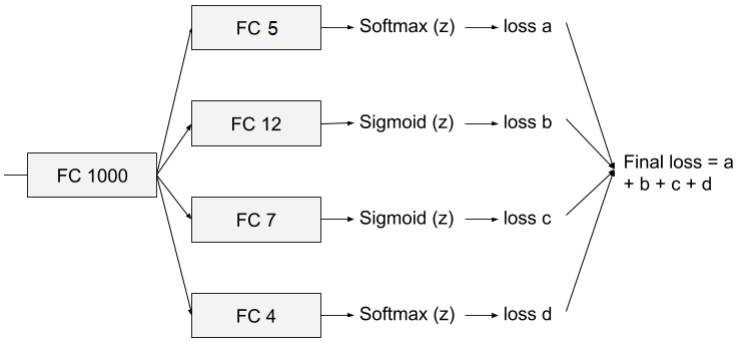


Figura 4.7: O cálculo da perda final

4.3.5 Otimização

Para otimização do treinamento e obtenção de melhores resultados, as seguintes técnicas tiveram aplicações:

- Algoritmo de Otimização *Adam*, ao invés do tradicional procedimento *stochastic gradient descent*, para atualizar os pesos da Rede Neural baseando-se iterativamente nos dados de treino [Bro18]. O procedimento *stochastic gradient descent* mantém a mesma *learning rate* para todas atualizações de pesos, não a mudando durante o treino. O método *Adam* computa *learning rate* individuais e adaptativas para diferentes parâmetros, através de estimativas de um primeiro momento e de um segundo momento de uma gradiente [DPK15].

- Redução da *learning rate* conforme o número de *epochs* aumenta. Isso é baseado na intuição de que uma *learning rate* alta faz com o que o modelo possua alta energia cinética. Como resultado, o vetor de parâmetros é incapaz de se estabelecer em partes mais profundas e estreitas da função de perda (mínimos locais). Por outro lado, se a taxa de aprendizado for muito baixa, o modelo se estabeleceria em partes mais rasas e estreitas da função de perda (mínimos locais) [Sud17]. Para manter o equilíbrio da *learning rate*, foi utilizado um *scheduler*, que diminui a *learning rate* em um fator *gamma* de 0.1 a cada 30 *epochs*.

- Transferência de conhecimento, ou *Transfer Learning*: raramente uma CNN é treinada do zero. Ao invés disso, é comum pré-treinar uma CNN em um *dataset* muito grande, e então usar a rede pré-treinada como inicializadora dos novos pesos ou como extratora de recursos fixos [Uni17b]. Em nosso caso, os pesos da ResNet foram inicializados com os

pesos providos por pré-treinamento no *ImageNet* [Uni17a]. Após tal ato, o treinamento foi realizado de forma a atualizar todos esses pesos pré-inicializados. Essa técnica de *Transfer Learning* é conhecida como *fine-tuning*.

4.4 Implementação

Todo o código e suas respectivas funções para a implementação foram desenvolvidos na linguagem Python, utilizando a biblioteca Pytorch, uma biblioteca *open-source* para desenvolvimento em *deep learning* [AP16].

Para a leitura do arquivo CSV contendo os dados, houve uso da biblioteca Pandas. Todas as variáveis foram convertidas para *one-hot encoding* através de dicionários da linguagem Python. Durante a leitura dos dados, também foi realizado o *Data Augmentation* com auxílio das bibliotecas PIL e SciKit-Learn.

Após a leitura, todos os dados são convertidos para *Tensors* da biblioteca Pytorch, a fim de possibilitar treinamento em GPU. Para montagem do *dataset*, há proveito do módulo TorchVision da biblioteca Pytorch, que fornece métodos para padronizar o carregamento e criação dos *data sets*, com divisão de dados para treino, validação e teste. Além disso, permite a criação de *data-loaders*, para carregamento de *batchs* de dados com o tamanho especificado por uma variável *batch size*. Da mesma forma, provê normalização e tratamento (*resize, scale* e *crop*) para todas as imagens da base.

Utilitários específicos, como o otimizador Adam, o *scheduler* para a *Learning Rate* e as funções de perda e ativação, foram todos providos pelo Pytorch. O Pytorch monta, de forma automática, um grafo computacional dinâmico, e realiza o *backpropagation* com o chamar de um método.

A implementação utilizada para a *ResNet* é o modelo ResNet18, do módulo Torch Vision. Tal modelo foi apenas modificado em sua saída final, de forma a adaptar-se aos quatro *layers* finais propostos na seção 4.3, como mostra o código 4.1.

Listing 4.1: Loop principal do treinamento da CNN

```
for image, release_date, game_genres, game_themes,
   game_rating, game_name, image_name in train_loader:

   optimizer.zero_grad()

   with torch.set_grad_enabled(True):
      outputs = model(image)

      dateFcOutput = dateFc(outputs)
      _, dateOutput = torch.max(dateFcOutput, 1)
      a = criterion(dateFcOutput, release_date)

      genreOutputSigmoid = sigmoid(genreFc(outputs))
      genreOutput = [genreOutputSigmoid > 0.55] # Threshold 0.55
      b = criterion2(genreOutputSigmoid, game_genres)

      themeOutputSigmoid = sigmoid(themeFc(outputs))
      themeOutput = [themeOutputSigmoid > 0.55] # Threshold 0.55
      c = criterion2(themeOutputSigmoid, game_themes)

      ratingFcOutput = ratingFc(outputs)
      _, ratingOutput = torch.max(ratingFcOutput, 1)
      d = criterion(ratingFcOutput, game_rating)
```

```
24
25        loss = a + b + c + d
26
27        loss.backward()
28        optimizer.step()
```

Por fim, a função para testar o modelo treinado foi desenvolvida de forma a, além de verificar a acurácia das previsões, também oferecer alguns dados estatísticos obtidos pelo teste, úteis para análise dos resultados. Essa análise será feita no capítulo 5.

4.5 Treinamento

O treinamento foi executado em uma máquina com processador Dual-Core Intel® Core™ i3-2100 CPU @ 3.10GHz com 3MB de memória cache, 12GB de memória RAM e GPU NVIDIA GeForce GTX 1060 de 3 GB com 1152 núcleos.

O *dataset* foi divido em 80% teste, 10% validação e 10% treino. Para maior eficiência na obtenção de um modelo final, houve aplicação da estratégia de *k-folds cross-validation* [Gei93b], que alterna entre 10 *sets* de validação diferentes. Quando um *set* não está sendo usado como validação, seus dados são transferidos para o *set* de treino. O modelo que obteve melhor desempenho para um determinado *k-set* foi escolhido como o modelo final.

Para cada *fold* de validação treinado, seguem os valores iniciais dos parâmetros:

- Número de épocas, ou *epochs*: 100
- *Learning Rate*: 0.001
- *Batch Size*: 32
- *Scheduler Step Size*: 30 *epochs*

Com as configurações acima, cada *fold* leva cerca de 13 horas para terminar o treinamento, e, ao total, o treinamento inteiro leva cerca de 5 dias e 10 horas.

5 Análise dos Resultados

Neste capítulo, serão apresentados os resultados obtidos a partir de testes com o nosso modelo de CNN. Todos os testes foram realizados com base no modelo de CNN descrito na seção 4.3 do capítulo 4. O treinamento foi desenvolvido a partir da base de dados customizada descrita nas seções 4.1 e 4.2, também do capítulo 4.

5.1 Testes

A base de dados, tratada com dados obtidos pela API do IGDB, foi dividida de modo que 80% dos dados foram utilizados para o treinamento da CNN, 10% utilizado para validação, e os 10% restantes para testes. Os dados de teste serviram como entrada da rede e foram classificados de acordo com plataforma, gênero, tema e nota.

Após os testes, foram comparadas as classificações obtidas pela CNN com as classificações já existentes no IGDB, para assim ser possível avaliar o desempenho da rede.

5.1.1 Rede Completa

Inicialmente, a rede foi treinada contendo os quatro FCs finais ativados. Ou seja, para que, ao mesmo tempo, classificasse plataforma, gêneros, temas e nota. Tal tarefa provou-se ser extremamente difícil para a CNN, pois são diversas variáveis para análise e classificação. Os resultados obtidos não foram satisfatórios, conforme serão descritos nas próximas subseções.

Acertos e erros: Plataforma e Nota

A classificação de plataforma é o equipamento (*console*) ao qual o jogo pertence. Em nosso caso, abstraímos a data de lançamento, para que a mesma representasse a plataforma do jogo. Algumas inconsistências dificultam muito a tarefa, como:

- Conforme o avanço das gerações de *consoles*, a qualidade gráfica dos jogos tende a melhorar e se tornar mais realista, o que é um ponto importante de aprendizado pela rede neural. No entanto, isso não é uma regra, e muitos jogos modernos podem conter gráficos simplistas ou até antigos, de forma proposital.

- Jogos produzidos por desenvolvedores pequenos ou independentes costumam também ser mais simplistas, o que pode confundir a CNN, fazendo-a pensar que são jogos antigos.

- Um determinado jogo pode variar muito em diferentes seções. Logo, mesmo que a rede aprenda que determinada característica de um jogo é mais comum em uma certa geração de *consoles*, o mesmo jogo pode aparentar ser totalmente diferente em outra imagem.

Após os testes, os resultados em relação à plataforma não foram bons, vide figuras 5.1 e 5.2. A taxa de acerto para a maioria das plataformas foi muito baixa, com exceção da classificação para o *Playstation 2* (algo que também pode ser observado nas figuras 5.1 e 5.2).

- *SNES*: 5 acertos de 834.
- *Playstation 1*: 0 acertos de 675.
- *Playstation 2*: 430 acertos de 462.
- *Playstation 3*: 1 acerto de 988.
- *Playstation 4*: 81 acertos de 1366.

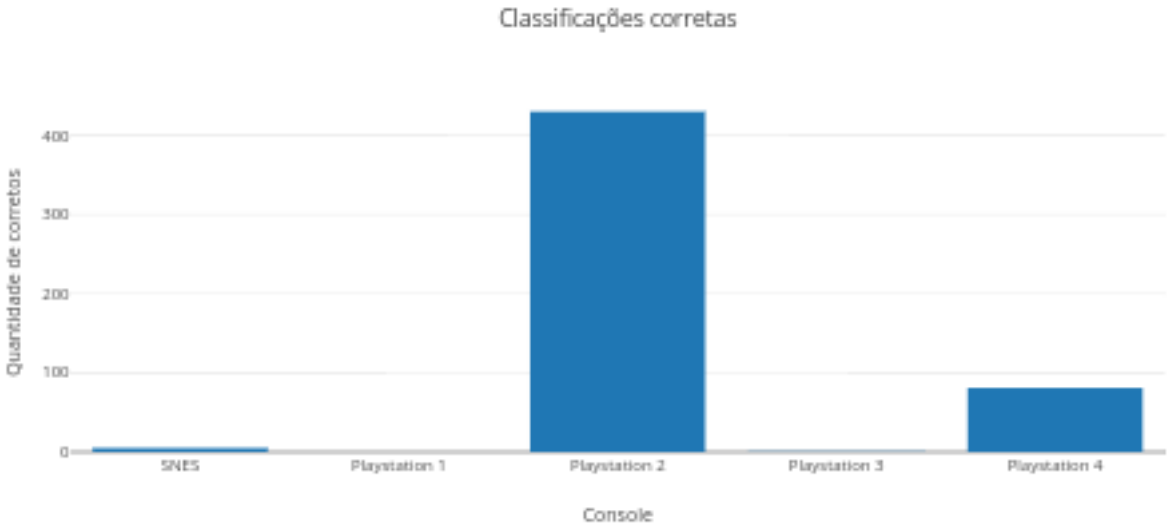


Figura 5.1: Gráfico de classificações corretas por geração (data de lançamento)

A melhor taxa de acerto conseguida pela rede foi de 93% (jogos do *Playstation 2*), e a pior foi de 0% (jogos do *Playstation 1*). Os testes com a nota, ou *rating*, também obtiveram resultados parecidos: classificações erradas e quase totalmente centralizadas apenas em uma das classes (a classe que representa nota $2/4$ para o jogo). Isso indica que a rede neural se perdeu durante a classificação, tanto pelo alto número de variáveis (plataforma, gêneros, temas e nota), como pelas inconsistências presentes nas imagens dos jogos.

Acertos e erros: Gênero

Jogos eletrônicos possuem várias classificações e subclassificações, denominadas gêneros. Essas classificações determinam um certo conjuntos de atributos que o jogo deve possuir para ser considerado de um certo gênero. Tais classificações não são mutuamente exclusivas, logo, um jogo pode ter múltiplos gêneros.

Isso dificulta o reconhecimento, pois, além de serem muito diversos, os gêneros podem ter características que se sobrepõem umas as outras. A identificação dessas características é complexa até para seres humanos. Por exemplo, uma pessoa com conhecimento em certos tipos de jogos poderá ter facilidade em identificar jogos do gênero que ela conhece, porém falhará

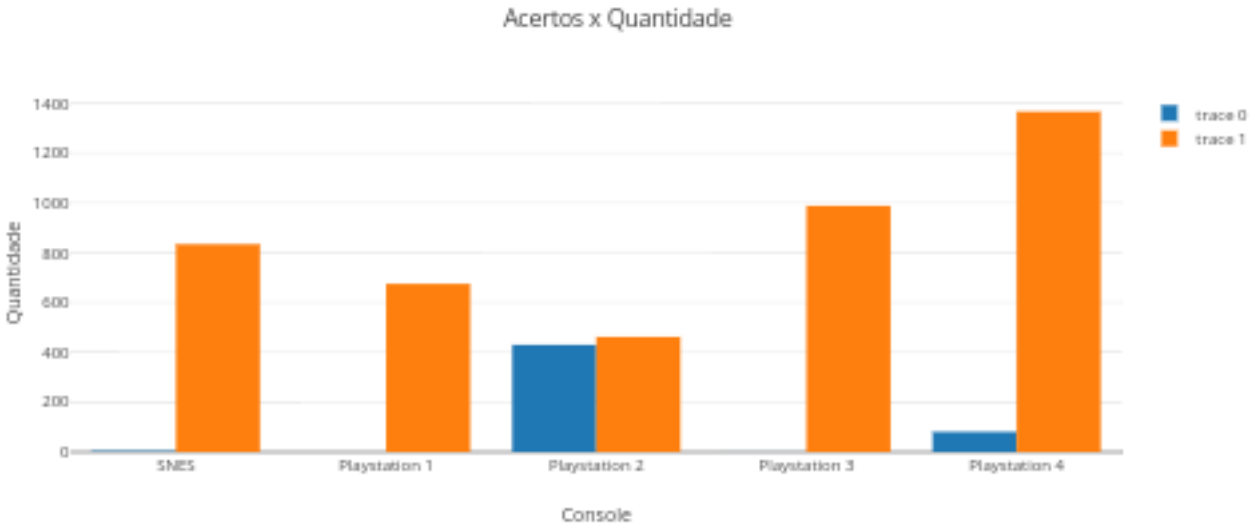


Figura 5.2: Gráfico de classificações corretas por geração x quantidade total

em outros. Outra pessoa, sem conhecimento nenhum sobre jogos, pode acabar errando diversas vezes, em vários jogos.

Outra dificuldade se apresenta durante a análise das imagens. Assim como na data de lançamento, muitas vezes um jogo possui características explícitas que definem os seus gêneros. Porém, em vários casos, a imagem contém a representação de uma porção do jogo que não condiz com suas características de gênero, confundindo a rede neural.

Assim como na classificação de plataforma, o desempenho da rede em relação a gêneros de jogos foi ruim (conforme gráfico da figura 5.3), indicando que a rede não foi capaz de abstrair as informações necessárias.

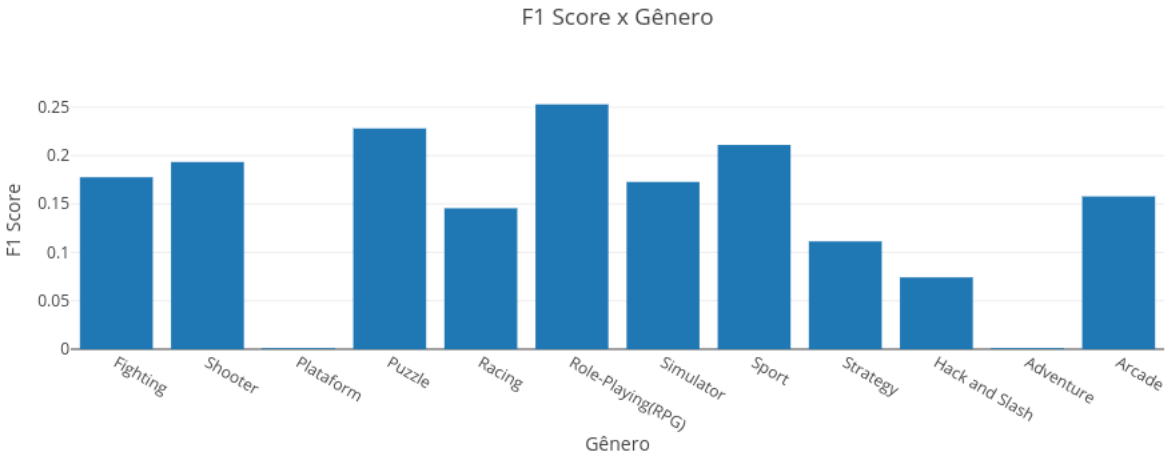


Figura 5.3: Gráfico de pontuação F1 para cada gênero

Muitos desses gêneros são difíceis de classificar, pois vários deles estão mais atrelados à jogabilidade do que com os visuais (imagens). Por exemplo: a velocidade de como os eventos acontecem dentro do jogo. Para esses gêneros, o melhor modo de classificá-los seria por vídeo, ou efetivamente jogando (ex.: *Hack and slash* e *Shooter*).

Acertos e erros: Temas

As mesmas dificuldades encontradas para as classificações anteriores se confrontaram com a classificação de temas. No entanto, os resultados obtidos foram superiores, embora ainda não satisfatórios.

Constata-se, que, para o caso dos temas, a CNN ao menos foi capaz de abstrair as informações necessárias, não realizando as classificações de forma quase imprevisível. No entanto, falhou em generalizar a totalidade dos temas, pois alguns deles quase não foram classificados. Enquanto isso, Ação, que é um dos temas mais comuns, obteve ótimo desempenho, como mostra a figura 5.4.

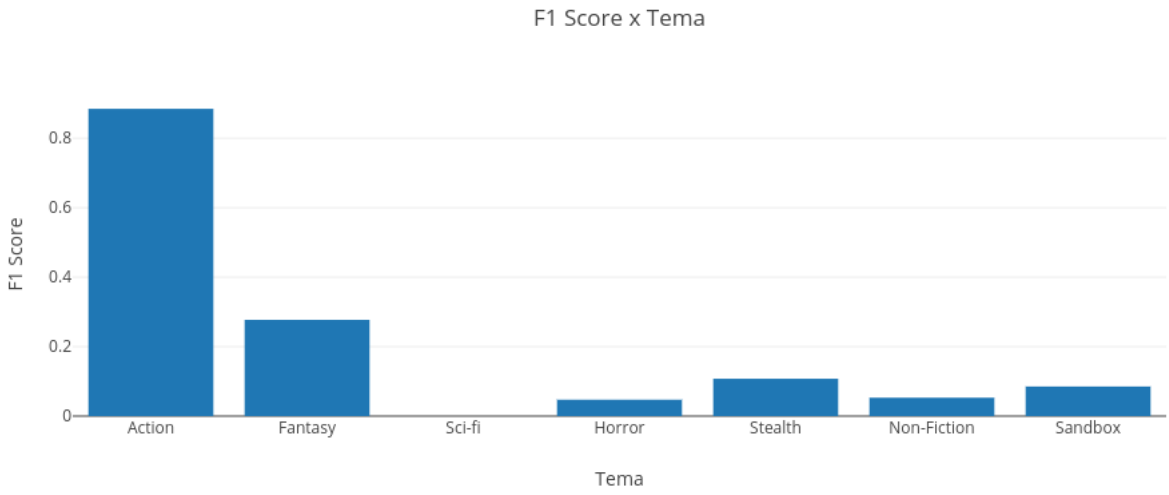


Figura 5.4: Gráfico de pontuação F1 para cada tema

5.1.2 Rede Parcial

Posteriormente, foram congelados todos os FCs finais, exceto o de classificação de gêneros. Após isso, a rede foi treinada novamente classificando apenas gêneros, a fim de verificar se há melhoras com a diminuição do número de variáveis de classificação.

Os testes (figura 5.5) indicam que houve melhora na taxa de acertos. Dois gêneros, *Platform* e *Adventure*, conseguiram ótimos resultados, com pontuação F1 acima de 0.5. *Shooter, Puzzle* e *RPG* também foram regulares. O destaque negativo fica para os temas pouco presentes, como *Sport*. *Strategy* também deveria conseguir resultados melhores, visto que possui boa presença na base de dados.

A mesma estratégia foi aplicada para a classificação de plataforma, agora mantendo de FC final apenas o de plataforma (figura 5.6). Dessa vez, também foi calculado a quantidade de classificações que, ou acertaram o resultado, ou estiveram em uma margem de erro entre -1 e $+1$ (figura 5.7). Os resultados foram melhores (a figura 5.8 traz o exemplo de um jogo classificado corretamente), validando que a rede foi capaz de distinguir as diferentes plataformas, ao contrário do caso verificado na figura 5.2. Porém, os resultados ainda são apenas regulares.

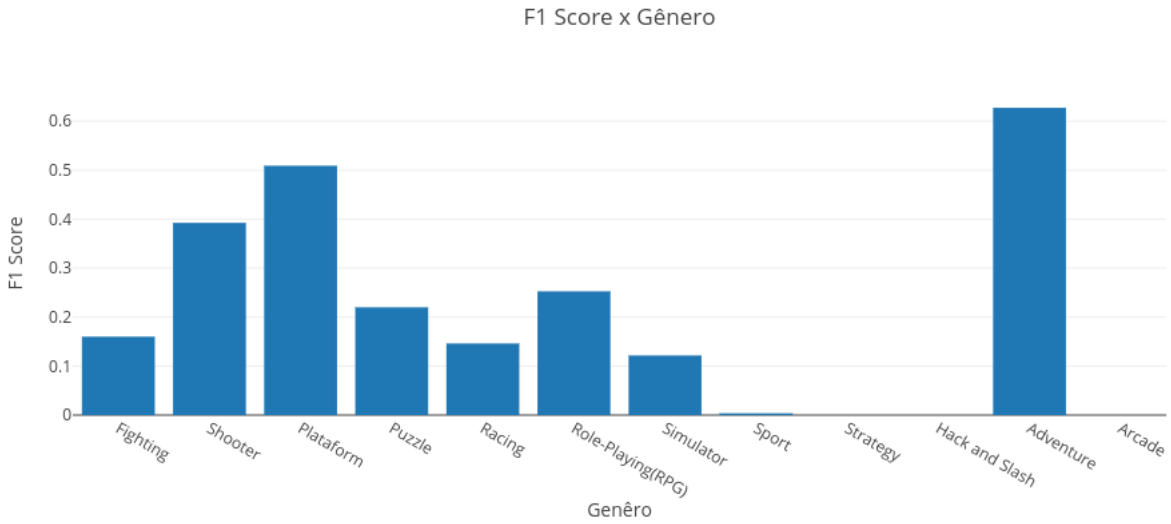


Figura 5.5: Gráfico de pontuação F1 para cada gênero, com treinamento especializado

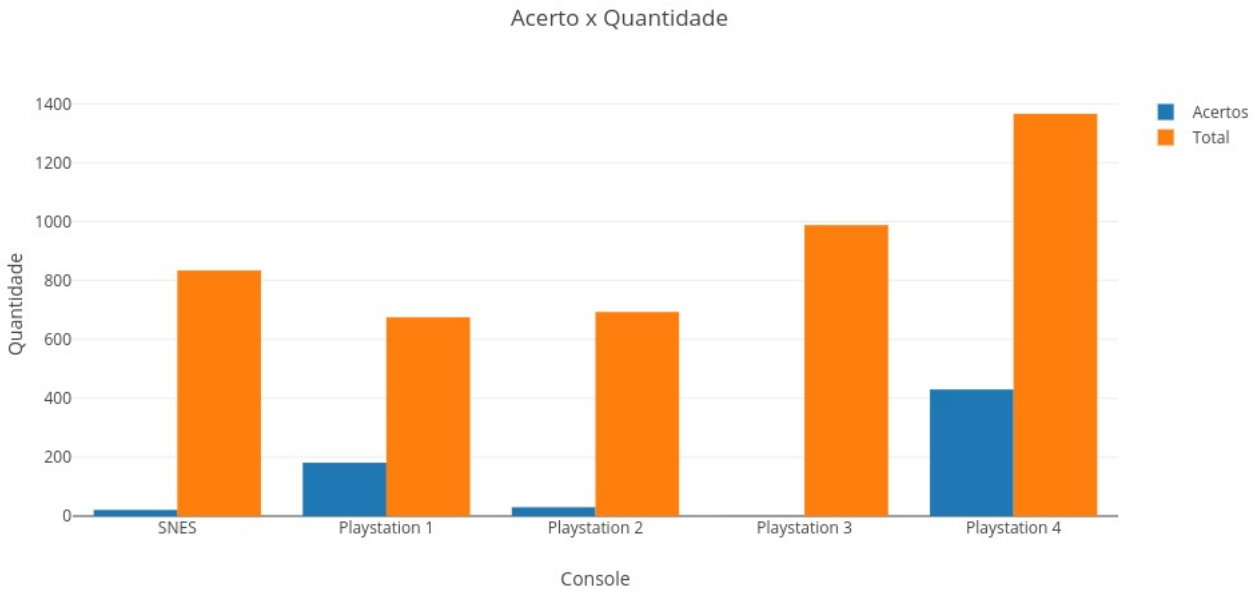


Figura 5.6: Gráfico de classificações corretas por geração x quantidade total, com treinamento especializado

5.1.3 Conclusão dos testes

Através dos testes realizados, conclui-se que a rede projetada ainda não é efetiva para a classificação de múltiplas variáveis. O *overload* de dados, informações e características faz com que a mesma tenha dificuldades em abstrair o conhecimento. No entanto, caso treinemos cada tipo de classificação separadamente, a mesma fornece resultados regulares. Com um *hardware* de maior poder, maior tempo de treinamento e ajustes finos na arquitetura da CNN, poderiam ser obtidos resultados mais atrativos.

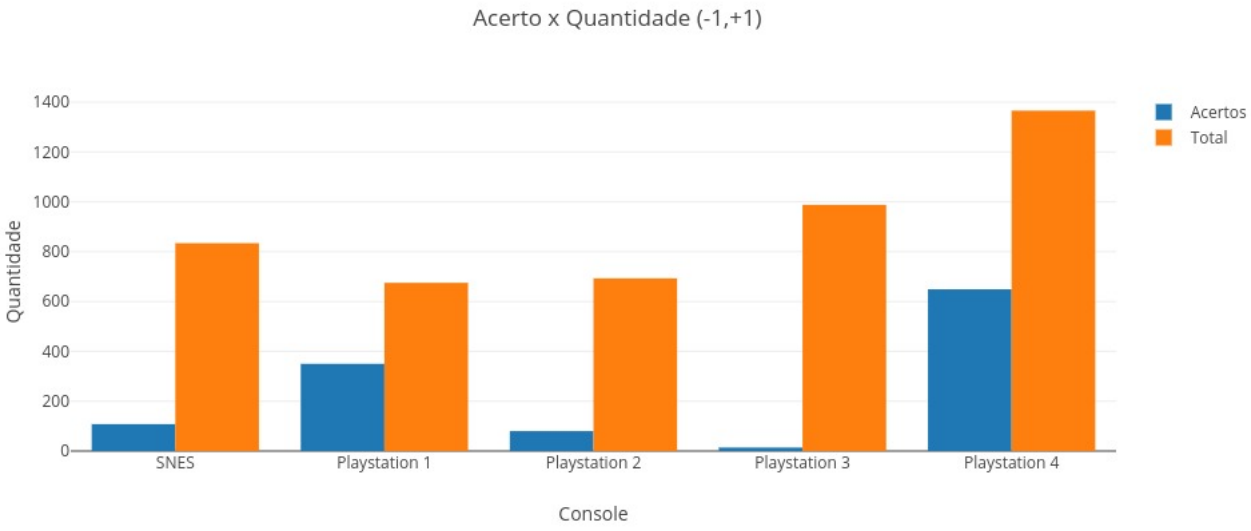


Figura 5.7: Gráfico de classificações corretas por geração x quantidade total, com treinamento especializado e tolerância de erro

Figura 5.8: Na imagem, o jogo Crash Bandicoot, lançado em 1996 para o Playstation 1, teve a sua plataforma corretamente classificada pela CNN

6 Usos e aplicações

Neste capitulo serão exemplificados alguns dos possíveis usos para o trabalho desenvolvido.

6.1 Previsão de sucesso durante o desenvolvimento

Durante o desenvolvimento de um jogo, a CNN apresentada pode ser utilizada para extrair características de um jogo em questão, características essas que podem servir para avaliação da probabilidade de sucesso. Com esses indicadores, os desenvolvedores podem tomar decisões mais informadas sobre as características do jogo, podendo alterá-las e combiná-las, assim melhorando a chance de sucesso antes do mesmo ser finalizado e comercializado. Neste caso, a rede seria utilizada como ferramenta de apoio e até como um indicador qualitativo para desenvolvedores de jogos, com o intuito de maximizar a qualidade final de seu produto.

6.2 Previsão de sucesso para investidores

Quanto mais informações e indicadores sobre um negócio são mostrados e avaliados, mais precisa a decisão sobre a viabilidade do projeto tende a ser. Para investidores, a rede apresentada pode ser utilizada para auxiliar o julgamento e a tomada de decisão sobre a viabilidade de investimento em um determinado jogo.

7 Considerações finais e trabalhos futuros

O presente trabalho consistiu no desenvolvimento e treinamento de uma rede neural artificial, utilizando milhares de imagens, para extrair características de jogos digitais. Ao extrair essas características, os desenvolvedores podem fazer melhorias em seu produto, assim como também há a oportunidade usar esses dados para procurar investidores. Para o investidor, é mais uma ferramenta para auxiliá-lo a tomar a decisão de investir ou não em um determinado projeto. Com todos esses fatores, potencialmente pode-se economizar dinheiro, aumentando a taxa de acerto dos investimentos e diminuindo o prejuízo gerado por jogos mal sucedidos.

7.1 Conclusões

O problema proposto mostrou-se ser um problema de alta complexidade, causando dificuldades no desenvolvimento e na execução. Para as classificações exigidas, a obtenção e tratamento de um *dataset* rico em detalhes e variedade de dados é uma tarefa penosa, pois em várias das imagens pouco pode-se inferir em relação ao que a mesma representa do jogo em questão. As técnicas de filtragem de dados, discretização e *data augmentation* proveram melhora e maior eficácia ao treinamento, no entanto, muitos dados ainda são perdidos ou pouco aproveitados, causando divergências no aprendizado.

Os testes realizados demonstraram que a rede acertou pouco e errou bastante, não sendo bem sucedida em classificar na totalidade as características dos jogos. Dada a alta discrepância entre os dados, visto que pertencem a *consoles* diferentes e possuem grande pluralidade de estilos visuais, é compreensível que a CNN não tenha conseguido detectar muitos padrões.

Ao diminuir o escopo da classificação, foram obtidos resultados melhores. No entanto, ainda são passíveis de muito aperfeiçoamento. Embora tenhamos utilizados diversas técnicas do estado-da-arte das redes neurais, ainda há espaço para aplicação e experimentação de diversos estudos. Ajustes finos também podem ser aplicados em nossa arquitetura, como mudanças no cálculo da perda, definição de um valor diferente de *threshold* para as classificações *multi-class*, treinamento em *hardware* de maior poder, entre outras coisas.

Por fim, os resultados obtidos indicam que classificação de jogos com base em imagens é uma área ainda pouco explorada, que carece de estudos. Este trabalho pode servir de ponto de partida para estudos relacionados na área, pois fornece um modelo base para a análise de imagens de jogos.

7.2 Trabalhos futuros

É possível treinar a CNN deste trabalho com dados mais limitados, escolhendo apenas uma plataforma ou apenas um estilo gráfico, por exemplo. De tal forma, seria viável avaliar se há melhora nos resultados, e, caso haja, desenvolver um estudo sobre quais combinações de dados e

características forneceram melhores resultados. O aprimoramento da mesma CNN também pode ser feito, para que, além de imagens, sejam extraídas características de *trailers* e trechos de video, assim aumentando a capacidade da rede.

As características obtidas através da análise de imagens, que são o objetivo deste presente trabalho, podem ser utilizadas como entrada no projeto Predição de Desempenho em Avaliação de Jogos Digitais Utilizando Redes Long Short-Term Memory. Tal projeto foi desenvolvido em paralelo a este. A unificação deste trabalho com o projeto em questão é uma proposta a ser trabalhada.

Referências

[Aca17] Khan Academy. Regra da cadeia. https://pt.khanacademy.org/math/ap-calculus-ab/ab-differentiation-2-new/ab-3-1a/a/chain-rule-review, 2017.

[Adm17] International Trade Administration. 2017 top markets report media and entertainment sector snapshot. Technical report, U.S. Department of Commerce, United States of America, 2017.

[AdSP13] Ana Beatriz Bahia Luís Carlos Petry Luciana Rocha Mariz Clua e Antônio Vargas Arlete dos Santos Petry, André Luiz Battaiola. Parâmetros, estratégias e técnicas de análise de jogo: o caso a mansão de quelícera. In *3rd XII SBGames*, São Paulo - Brazil, October 2013.

[AKH12] Ilya Sutskever Alex Krizhevsky and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, pages 1097–1105, Lake Tahoe, Nevada - United States of America, December 2012.

[AP16] Soumith Chintala Gregory Chanan Adam Paszke, Sam Gross. Pytorch. https://pytorch.org/, 2016.

[Aur18] Dicionário Aurélio. Definição de imagem. https://dicionariodoaurelio.com/imagem, 2018.

[Bro18] Jason Brownlee. Gentle introduction to the adam optimization algorithm for deep learning. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/, 2018.

[BS16] PrzemysĹ,aw Buczkowski and Antoni Sobkowicz. Predicting video game properties with deep convolutional neural networks using screenshots. In *LWDA 2016*, Potsdam - Germany, September 2016.

[Bus17] Vitaly Bushaev. How do we 'train' neural networks? https://towardsdatascience.com/how-do-we-train-neural-networks-edd985562b73, 2017.

[Che17a] ML Cheatsheet. Loss functions. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html, 2017.

[Che17b] ML Cheatsheet. Loss functions. https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html, 2017.

[DPK15] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2015.

[FL18] Rong-En Fan and Chih-Jen Lin. A study on threshold selection for multi-label classification. , 2018.

[Gei93a] Seymour Geisser. Cross-validation. https://www.teco.edu/~albrecht/neuro/html/node7.html), 1993.

[Gei93b] Seymour Geisser. Cross-validation. https://en.wikipedia.org/wiki/Cross-validation_(statistics), 1993.

[Gil17] Navdeep Singh Gill. Artificial neural networks, neural networks applications and algorithms. https://www.xenonstack.com/blog/data-science/artificial-neural-networks-applications-algorithms/, 2017.

[Huw17] Lukas Huwald. Recognizing game genres from screenshots using cnns. https://vanhavel.github.io/2017/09/12/cnn-games.html/, 2017. Acessado em 29/11/2018.

[IGD18a] IGDB. Internet games database. https://www.igdb.com/about, 2018.

[IGD18b] IGDB. Internet games database api. https://www.igdb.com/api, 2018.

[KH18] Shaoqing Ren Jian Sun Kaiming He, Xiangyu Zhang. Internet games database api. https://arxiv.org/abs/1512.03385, 2018.

[Lab18] Stanford Vision Lab. Ilsvrc. http://image-net.org/challenges/LSVRC/, 2018.

[Le16] James Le. The 10 algorithms machine learning engineers need to know. https://www.kdnuggets.com/2016/08/10-algorithms-machine-learning-engineers.html, 2016.

[LeC13] LeCun. Lenet-5, convolutional neural networks. http://yann.lecun.com/exdb/lenet/, 2013.

[Nie18] Michael Nielsen. Why are deep neural networks hard to train? http://neuralnetworksanddeeplearning.com/chap5.html, 2018.

[Pad16] Sachin Padmanabhan. Convolutional neural networks for image classification and captioning. https://web.stanford.edu/class/cs231a/prev_projects_2016/example_paper.pdf/, 2016. Acessado em 15/11/2018.

[Rau04] Matthias Rauterberg. Positive effects of entertainment technology on human behaviour. *Kluwer Academic Press*, pages 51–58, 2004. invited paper.

[Sak18] Andrey Sakryukin. Under the hood of neural networks. part 1: Fully connected. https://towardsdatascience.com/under-the-hood-of-neural-networks-part-1-fully-connected-5223b7f78528, 2018.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, January 2015.

[Sha17] Sagar Sharma. Activation functions: Neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6, 2017.

[Ste18] Samuel Stewart. What is anti-aliasing? https://www.gamingscan.com/what-is-anti-aliasing/, 2018.

[Sud17] Shreenidhi Sudhakar. Learning rate scheduler. https://towardsdatascience.com/learning-rate-scheduler-d8a55747dd90, 2017.

[TKF09] Jean Carlo Albiero Berni Tiago Keller Ferreira, Marcos Cordeiro d'Ornellas. Planejamento estratégico na produção de jogos eletrônicos. In *3rd XXIX ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO*, Salvador - Brazil, October 2009.

[Trn17] Michal Trněný. Machine learning for predicting success of video games. Master's thesis, Masaryk University - Faculty of Informatics, Brno - Czech Republic, March 2017.

[Ujj16a] Ujjwalkarn. An intuitive explanation of convolutional neural networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, 2016.

[Ujj16b] Ujjwalkarn. A quick introduction to neural networks. https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, 2016.

[Uni17a] Stanford University. Imagenet. http://www.image-net.org/, 2017.

[Uni17b] Stanford University. Learning rate scheduler. http://cs231n.github.io/transfer-learning/, 2017.

[Uni17c] Stanford University. Linear classification. http://cs231n.github.io/linear-classify/, 2017.

[Val17] Adriano Valente. 15 overhyped games that flopped. https://www.thegamer.com/15-overhyped-games-that-flopped/, 2017. Acessado em 05/11/2018.

[War14] Christina Warren. 28 days of fame: The strange, true story of 'flappy bird'. https://mashable.com/2014/02/10/flappy-bird-story/#Zfev0IprggqM, 2014. Acessado em 05/11/2018.

[Wij18] Tom Wijman. Mobile revenues account for more than 50% of the global games market as it reaches $ 137.9 billion in 2018. https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/, 2018. Acessado em 05/11/2018.

[Wik17] Wikipedia. Função de perda. https://pt.wikipedia.org/wiki/Fun%C3%A7%C3%A3o_de_perda, 2017.

[Wik18a] Wikipedia. Gradiente. https://pt.wikipedia.org/wiki/Gradiente, 2018.

[Wik18b] Wikipedia. Multi-label classification. https://en.wikipedia.org/wiki/Multi-label_classification, 2018.