

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO  
INSTITUTO DE COMPUTAÇÃO  
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FILLIPE RODRIGUES COSTA  
JOÃO PEDRO DE PAULA OLIVEIRA  
PHILIPY SIQUEIRA DA SILVA

ESTUDO DE CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES NEURAS  
CONVOLUCIONAIS

RIO DE JANEIRO  
2022

FILLIPE RODRIGUES COSTA  
JOÃO PEDRO DE PAULA OLIVEIRA  
PHILIPPI SIQUEIRA DA SILVA

ESTUDO DE CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES NEURAIAS  
CONVOLUCIONAIS

Trabalho de conclusão de curso de graduação  
apresentado ao Instituto de Computação da  
Universidade Federal do Rio de Janeiro como  
parte dos requisitos para obtenção do grau de  
Bacharel em Ciência da Computação.

Orientador: Prof. João Carlos Pereira da Silva

RIO DE JANEIRO

2022

C837e

Costa, Fillipe Rodrigues

Estudo de classificação de imagens utilizando redes neurais convolucionais / Fillipe Rodrigues Costa, João Pedro de Paula Oliveira e Philipi Siqueira da Silva. – 2022.

48 f.

Orientador: João Carlos Pereira da Silva.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação)  
- Universidade Federal do Rio de Janeiro, Instituto de Computação, Bacharel em Ciência da Computação, 2022.

1. Fuzzing. 2. Aplicações web. 3. Segurança. 4. Segurança da informação. 5. Aprendizado de máquina. 6. Firewalls. I. Oliveira, Daigoro Alencar. II. Aguiar, Paulo Henrique Rodrigues (Orient). III. Farias, Claudio Miceli de (Coorient.). IV. Universidade Federal do Rio de Janeiro, Instituto de Computação. V. Título.

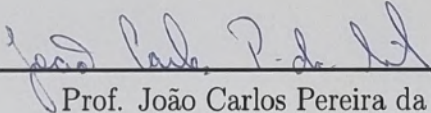
FILLIPE RODRIGUES COSTA  
JOÃO PEDRO DE PAULA OLIVEIRA  
PHILIPY SIQUEIRA DA SILVA

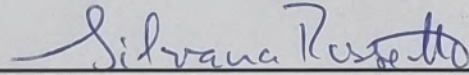
ESTUDO DE CLASSIFICAÇÃO DE IMAGENS UTILIZANDO REDES NEURAIAS  
CONVOLUCIONAIS

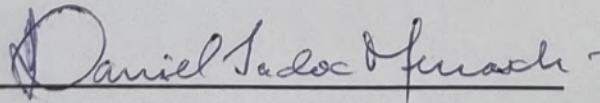
Trabalho de conclusão de curso de graduação  
apresentado ao Instituto de Computação da  
Universidade Federal do Rio de Janeiro como  
parte dos requisitos para obtenção do grau de  
Bacharel em Ciência da Computação.

Aprovado em 03 de outubro de 2022

BANCA EXAMINADORA:

  
Prof. João Carlos Pereira da Silva  
D.Sc. (UFRJ)

  
Profa. Silvana Rossetto  
D.Sc. (PUC-RJ)

  
Prof. Daniel Sadoc Menasché  
Ph.D. (UMASS)

## RESUMO

Este trabalho visa realizar um estudo sobre construção de redes neurais voltadas ao reconhecimento de imagens, buscando compreender como cada parâmetro da rede é capaz de influenciar o modelo como um todo. Neste trabalho, foram realizados experimentos sobre dois bancos de dados de imagens, Cifar-10 e Cifar-100, com execuções diversas do algoritmo, utilizando variações distintas da arquitetura. Para os modelos de aprendizado, utilizamos a linguagem de programação Python em conjunto com a biblioteca Keras, escolhida devido à simplicidade e legibilidade do código. Ao longo do trabalho, foram realizados diversos experimentos sob condições diferentes e utilizando diferentes algoritmos e técnicas com finalidade de obtenção de melhores resultados. A métrica utilizada para a avaliação dos resultados foi a acurácia. Cada experimento teve seus parâmetros e resultados devidamente detalhados e documentados. Ao final dos experimentos, foi possível analisar os impactos que cada parâmetro tem sobre os modelos construídos. Foi possível construir uma arquitetura satisfatória para o Cifar-10, porém para o Cifar-100, o modelo não alcançou resultados excelentes como o esperado.

**Palavras-chave:** Redes neurais convolucionais; Cifar; Classificação de imagens.

## ABSTRACT

This paper aims to carry out a study on image classification using convolutional neural networks, seeking understanding on how each parameter is able to influence the architectural model as a whole. The experiments were carried out on two image datasets, cifar-10 and cifar-100, each with different variations of the architecture. The learning models were built with the python programming language and the keras library, chosen for its code's simplicity and readability. Several experiments were executed under distinct conditions with the usage of multiple algorithms and techniques in order to obtain better results. The metric used to evaluate the results was the accuracy. Each experiments had its parameters and results duly detailed and documented. At the end, it was possible to analyze the impacts that each parameter had on the models built. It was possible to build a satisfactory architecture for cifar-10, however for Cifar-100, the model did not achieve excellent results as expected.

**Keywords:** Convolutional neural networks; Cifar; Image classification.

## LISTA DE ILUSTRAÇÕES

Figura 1 – Exemplo de conjunto de imagens com suas respectivas categorias . . . . .	10
Figura 2 – Desafios encarados na classificação de imagem . . . . .	10
Figura 3 – Um desenho teórico de um neurônio biológico . . . . .	11
Figura 4 – Modelo de um perceptron básico . . . . .	12
Figura 5 – Modelo de um perceptron com função de ativação . . . . .	13
Figura 6 – Forma da função sigmoide . . . . .	13
Figura 7 – Forma da função ReLU . . . . .	14
Figura 8 – Arquitetura de uma rede neural com 2 duas camadas ocultas . . . . .	15
Figura 9 – Uma iteração da camada de convolução . . . . .	18
Figura 10 – Max <i>pooling</i> com filtro 2x2 . . . . .	19
Figura 11 – Achatamento da matriz de entrada . . . . .	19
Figura 12 – Uma rede densamente conectada com duas camadas ocultas . . . . .	20
Figura 13 – Exemplo de uma rede neural convolucional . . . . .	21
Figura 14 – Boxplot dos experimentos . . . . .	31
Figura 15 – Boxplot dos experimentos: Separados por classes . . . . .	32
Figura 16 – Boxplot dos experimentos: Sem animais, apenas classe gato . . . . .	33
Figura 17 – Imagem original, flip horizontal e flip vertical, respectivamente . . . . .	39

## LISTA DE TABELAS

Tabela 1 – Parâmetros Cifar 10 . . . . .	28
Tabela 3 – Resultados Cifar 10 . . . . .	30
Tabela 4 – Experimento 11: Resultados separados por Classe . . . . .	30
Tabela 5 – Experimento 11: Resultados apenas com a classe Gato . . . . .	31
Tabela 6 – Experimento 11: Resultados apenas com os animais . . . . .	32
Tabela 7 – Resultados apenas com as superclasses do cifar 100 . . . . .	34
Tabela 8 – Cifar-100 sem utilizar superclasses e utilizando superclasses . . . . .	35
Tabela 9 – Resultados com as superclasses do cifar 100, utilizando data augmentation . . . . .	36
Tabela 10 – Cifar-100 sem e com <i>data augmentation</i> . . . . .	36
Tabela 11 – Resultados apenas com a superclasse dos invertebrados . . . . .	37
Tabela 12 – Resultados apenas com a superclasse das árvores . . . . .	37



## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>9</b>
<b>2</b>	<b>REDES NEURAIS . . . . .</b>	<b>11</b>
2.1	NEURÔNIO PERCEPTRON . . . . .	11
2.2	FUNÇÕES DE ATIVAÇÃO . . . . .	12
2.2.1	Função Sigmoide . . . . .	13
2.2.2	Função ReLU . . . . .	14
2.2.3	Função <i>softmax</i> . . . . .	15
2.3	REDES NEURAIS . . . . .	15
2.3.1	Arquitetura de uma rede neural . . . . .	15
2.3.2	Modelando uma rede neural . . . . .	16
2.4	REDES NEURAIS CONVOLUCIONAIS . . . . .	16
2.4.1	Tipos de camadas em uma Rede Neural Convolutacional . . . . .	17
2.4.1.1	Camadas de Convolução . . . . .	17
2.4.1.2	Camadas de <i>pooling</i> . . . . .	18
2.4.1.3	Camada de achatamento . . . . .	18
2.4.1.4	Conjunto de camadas densamente conectadas . . . . .	19
2.4.2	Estrutura final da rede . . . . .	20
2.5	FUNÇÃO DE CUSTO E GRADIENTE DESCENDENTE . . . . .	20
<b>3</b>	<b>TREINAMENTOS . . . . .</b>	<b>23</b>
3.1	FUNÇÕES E PARÂMETROS . . . . .	23
3.2	EXPERIMENTOS SOBRE O CIFAR 10 . . . . .	25
3.2.1	Base de Dados do Cifar 10 . . . . .	25
3.2.2	Experimentos . . . . .	26
3.2.3	Resultados . . . . .	27
3.2.4	Experimentos do Cifar10 Separados por Classes . . . . .	29
3.3	CIFAR 100 . . . . .	31
3.3.1	Base de Dados do Cifar100 . . . . .	31
3.3.2	Experimentos cifar-100 . . . . .	33
3.3.3	Experimentos Utilizando Data Augmentation . . . . .	35
3.3.4	Treinando Classes Isoladamente . . . . .	36
3.3.5	Conclusão dos experimentos . . . . .	37
<b>4</b>	<b>CONCLUSÃO . . . . .</b>	<b>40</b>

<b>REFERÊNCIAS</b> . . . . .	<b>42</b>
<b>APÊNDICE A – HIERARQUIA DE CLASSES CIFAR 100</b> . . . . .	<b>45</b>

## 1 INTRODUÇÃO

Vivemos em um mundo moderno onde utilizamos computadores não apenas para facilitar nossa vida com tarefas simples e repetitivas, pois em muitos aspectos da nossa sociedade, grandes decisões são tomadas por computadores, sem interação humana, após análise de vários dados disponíveis ao mesmo. De simples cálculos matemáticos e análises probabilísticas para tomada de decisões, temos também métodos de inteligência artificial e finalmente, aprendizado de máquina e processamento de imagens que torna os computadores cada vez mais indispensáveis para nossa sociedade.

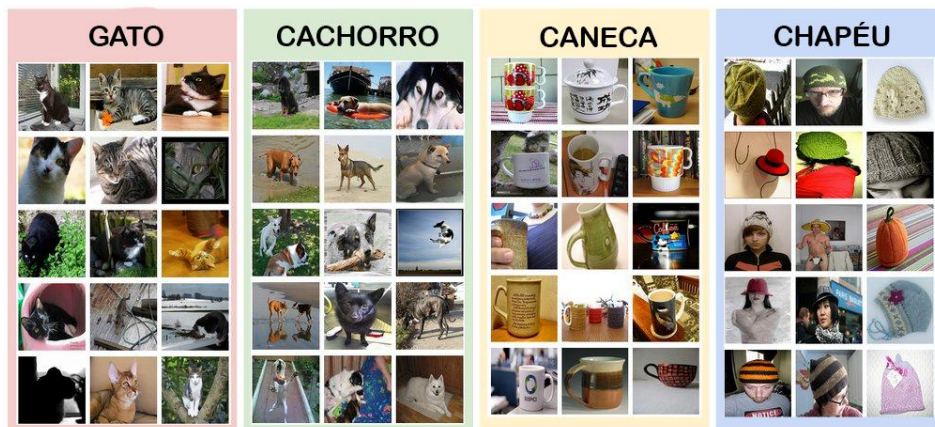
O uso de processamento e classificação de imagens para tomada de decisões já faz parte da rotina da população em geral, como o detector de sorrisos em uma câmera e sistemas que utilizam reconhecimento facial. Porém, o computador não tem a capacidade de abstração humana de visualizar uma imagem e tirar conclusões como, por exemplo, dizer se há ou não um cachorro na imagem. Para que o computador consiga extrair informações de uma imagem, existe a necessidade de modelos computacionais para treinamento e análise da composição digital da imagem e de técnicas para otimizar o processamento e resultados.

O problema da Classificação de Imagens é a tarefa de atribuir a uma imagem de entrada um rótulo pertencente a um conjunto fixo de categorias (Figura 1), o que apesar de ser natural ao ser humano, não é trivial para uma máquina. Alguns dos desafios que podem dificultar a interpretação de uma imagem por algoritmos são, por exemplo, mas não se limitando a: variação do ponto de vista, variação de escala, condições de iluminação, dentre outros (Figura 2). As redes neurais convolucionais (seção 2.4) são apropriadas para classificação de imagens.

Sendo assim, o objetivo deste estudo é avaliar o impacto que cada hiperparâmetro da rede neural convolucional possui sobre o modelo, levando em consideração o equilíbrio entre a acurácia dos testes e tempo de execução do algoritmo. Para isto, a rede precisará classificar corretamente as imagens de dois bancos de dados diferentes, Cifar-10 e Cifar-100. Durante o estudo, serão testados diferentes hiperparâmetros e em alguns casos haverá utilização de técnicas auxiliares, na tentativa de obter melhores resultados de classificação a cada experimento. Os modelos serão avaliados a partir de sua acurácia, pois essa métrica é a que melhor generaliza a performance entre todas as classes quando estas são de igual importância e estão igualmente representadas. (BROWNLEE, 2020b)

No próximo capítulo, será melhor explicada a base teórica de classificação de imagens e redes neurais convolucionais. No terceiro capítulo, será detalhada a metodologia do estudo, os experimentos realizados e seus resultados. Também serão feitas avaliações sobre estes resultados, e se eles atingiram o objetivo desejado. Finalmente, no capítulo 4, discutiremos as conclusões, dificuldades encontradas e ideias para continuação do estudo.

Figura 1 – Exemplo de conjunto de imagens com suas respectivas categorias



Fonte: <https://cbmm.mit.edu/>

Figura 2 – Desafios encarados na classificação de imagem



Fonte: [IntroToDeepLearning.com](http://IntroToDeepLearning.com)

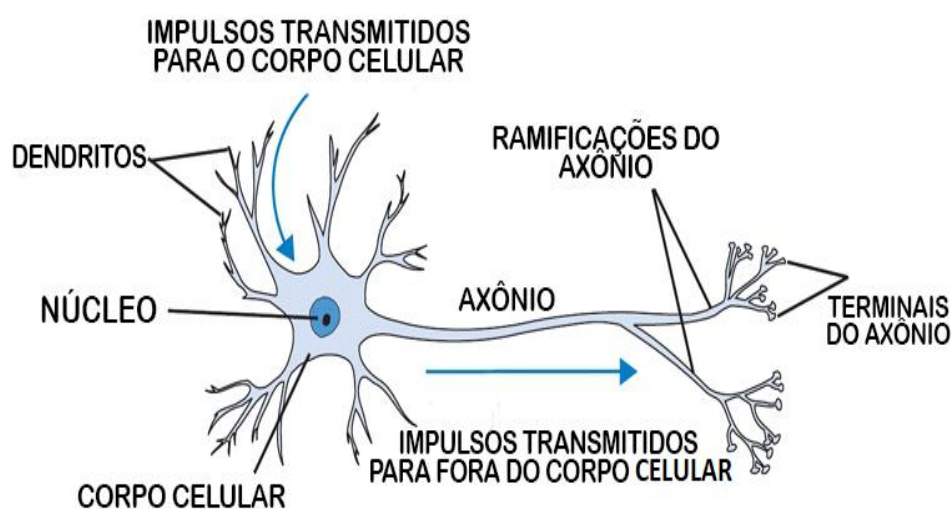
## 2 REDES NEURAIS

Neste capítulo introduziremos o conceito de redes neurais convolucionais (CNN), um algoritmo baseado na estrutura e funcionamento do cérebro humano. Seu objetivo é fazer com que uma máquina emule o comportamento do cérebro, aprendendo através de treinamento. Este algoritmo é apropriado para o problema da classificação de imagens. (LANG, 2021)

### 2.1 NEURÔNIO PERCEPTRON

O sistema nervoso do ser humano é composto de aproximadamente 86 bilhões de neurônios e entre  $10^{14}$  e  $10^{15}$  sinapses os conectando. Cada neurônio recebe sinais de entrada por seus dendritos e produzem sinais de saída pelo seu axônio, que eventualmente se conecta a dendritos de outros neurônios. Através desse sistema, os neurônios podem ou não ser ativados, devido aos sinais de entrada que recebem, e enviam o resultado pelo axônio.

Figura 3 – Um desenho teórico de um neurônio biológico



Fonte: (Stanford university, 2022)

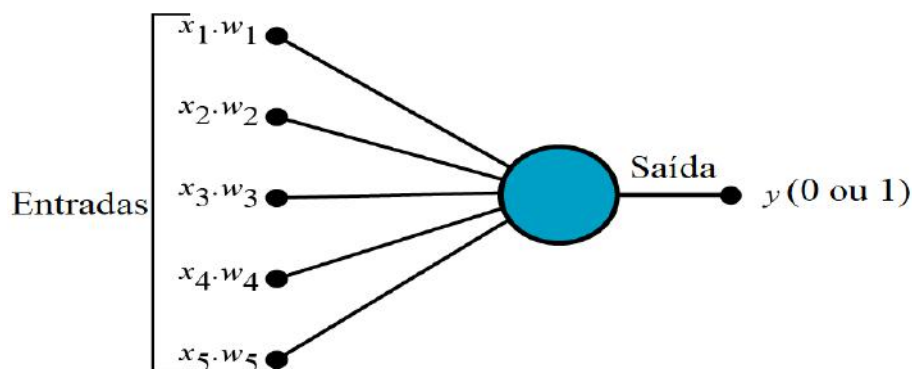
Um neurônio artificial tenta simular o comportamento de um neurônio biológico através de um modelo matemático. Um modelo simples e comumente utilizado é o perceptron. Este modelo possui entradas e saída com valores binários e determina se um neurônio é ativado ou não através do somatório dos valores de entrada multiplicados aos seus pesos correspondentes. Caso este somatório resulte em um valor maior ou igual a um dado valor limite, o neurônio é ativado com saída igual 1, caso contrário o neurônio não é ativado e tem saída 0. (NIELSEN, 2015a).

A função principal do perceptron pode ser descrita como uma função linear onde uma entrada  $x$  é multiplicada por um peso  $w$  e então somada à um viés  $b$  (Equação 2.1). O viés é equivalente ao inverso de um valor limite pré-definido. A saída do neurônio é decidida pelo somatório das funções (Equação 2.2). O modelo de um perceptron básico pode ser visualizado na Figura 4.

$$f(x, w, b) = \begin{cases} 0, & \text{se } x * w + b \leq 0; \\ 1, & \text{se } x * w + b > 0. \end{cases} \quad (2.1)$$

$$\text{saída} = \begin{cases} 0, & \text{se } \sum x_i * w_i + b \leq 0; \\ 1, & \text{se } \sum x_i * w_i + b > 0. \end{cases} \quad (2.2)$$

Figura 4 – Modelo de um perceptron básico



Fonte: (NIELSEN, 2015b)

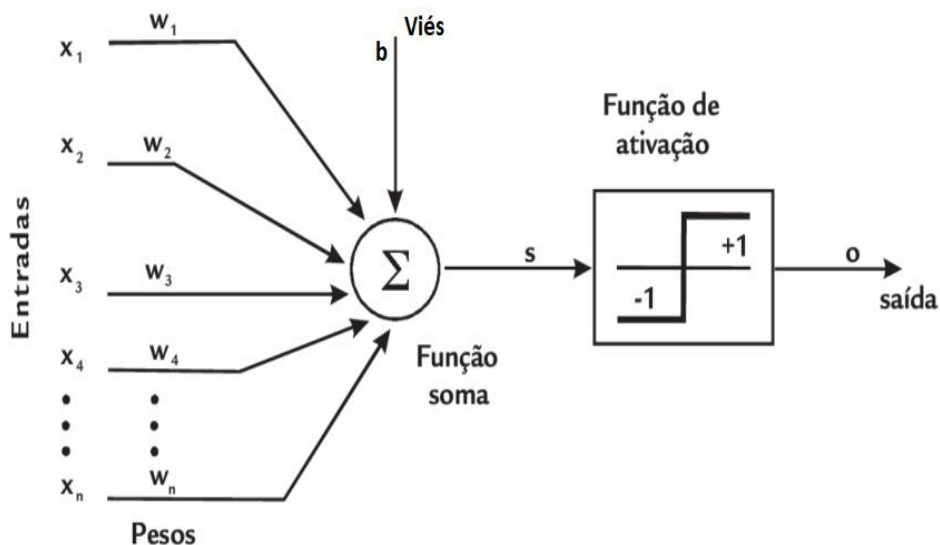
## 2.2 FUNÇÕES DE ATIVAÇÃO

O perceptron em sua concepção é limitado pelo fato de ser um modelo linear. Como sua saída é binária (0 ou 1), uma pequena alteração pode fazer com que o resultado se inverta caso o resultado da função ultrapasse o limiar definido. Portanto, ele é incapaz de modelar relações não-lineares entre entradas e saídas. Para que o neurônio consiga aprender padrões mais complexos, aplicaremos funções de ativação não lineares. Funções de ativação não lineares possuem vantagem em relação às lineares, pois é possível realizar ajustes finos com relação à saída com pequenos ajustes nos argumentos, permitindo que os neurônios consigam lidar com problemas mais complexos. A função de ativação funciona como uma camada adicional do processo matemático de decisão de um neurônio (Equação 2.3).

$$Y = \text{Ativação} \left( \sum (x_i * w_i) + b \right) \quad (2.3)$$

A seguir, veremos algumas funções de ativação comumente utilizadas. A representação de um perceptron com função de ativação pode ser visualizado na Figura 5.

Figura 5 – Modelo de um perceptron com função de ativação

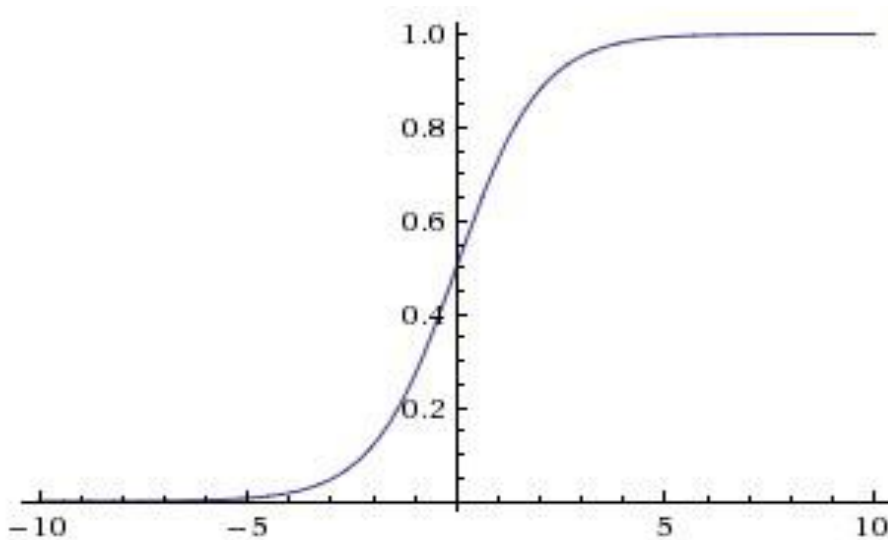


Fonte: (ROSA; LUZ, 2012)

### 2.2.1 Função Sigmoide

A função *sigmoide* é definida pela equação (2.4) e sua representação gráfica pode ser observada na Figura 6.

Figura 6 – Forma da função sigmoide



$$\sigma(x) = 1/(1 + e^{-x}) \quad (2.4)$$

Como os valores de saída do *sigmoide* variam de 0 a 1, o valor da classificação também representa a probabilidade de corretude da classificação, quanto mais o resultado se aproxima de 1, maior a probabilidade dele pertencer à classe. O valor 0.5 representa o limite de aceitação como membro pertencente à classe, então se o valor de saída for menor que 0.5, o neurônio não será ativado, enquanto que se for maior ou igual a 0.5, o neurônio será ativado.

Apesar de popular, a função de ativação *sigmoide* possui algumas desvantagens, como por exemplo, existe a possibilidade de que ocorra o problema da dissipação do gradiente (HOCHREITER, 1998), caracterizado pela diminuição, de forma exponencial, da derivada da função à medida que o tamanho da entrada aumenta (ver seção 2.5). Outra desvantagem é ser uma função computacionalmente custosa, pois o computador precisa realizar operações exponenciais.

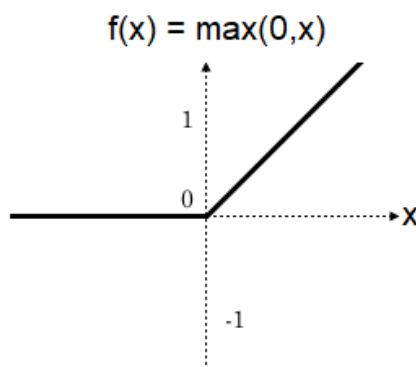
### 2.2.2 Função ReLU

Uma alternativa para a função *sigmoide* é a função de ativação *ReLU* (*Rectified Linear Unit*), definida na Equação 2.5. (SCHMIDT-HIEBER, 2020)

$$g(u) = \max(0, u) \quad (2.5)$$

Analisando a equação, vemos que se  $x$  for maior do que zero, a função retorna o próprio valor de  $x$ . Caso contrário, a função retorna 0. Esta função é linear quando a entrada é maior do que zero, porém é não linear quando a entrada é um valor menor do que 0. A função *ReLU* é muito utilizada por ser de fácil implementação computacional, preservar propriedades lineares e não apresentar o problema de dissipação de gradiente.

Figura 7 – Forma da função ReLU



Fonte: (PAULY et al., 2017)



### 2.2.3 Função *softmax*

A função de ativação *softmax* transforma um vetor de números em um vetor de probabilidades (BROWNLEE, 2020a). A função *softmax* é aplicada em cada elemento do vetor, transformando-os em probabilidades. Como o vetor resultante é formado por probabilidades, a soma de todos os elementos é 1. A fórmula é definida na equação (2.6).

$$f(x)_i = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} \quad (2.6)$$

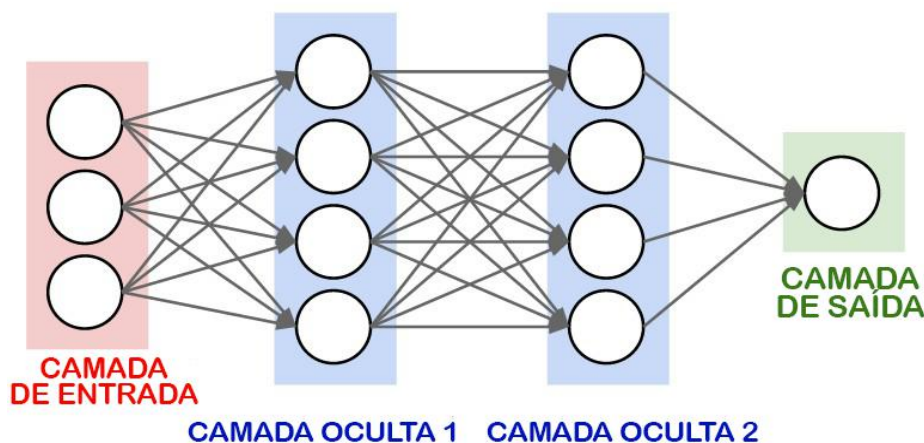
## 2.3 REDES NEURAIIS

### 2.3.1 Arquitetura de uma rede neural

Redes neurais são uma coleção de neurônios modelados na forma de um grafo acíclico dirigido. Em outras palavras, a saída de um neurônio torna-se a entrada de outro, sem retornar à um neurônio anterior. As redes são organizadas em diferentes camadas que normalmente são do tipo totalmente conectadas. Neste modelo, os neurônios entre duas camadas são conectados a todos os outros neurônios das camadas anterior (entrada) e posterior (saída), porém não são conectados a outros neurônios pertencentes à mesma.

Uma rede neural possui três tipos de camadas. A primeira é a camada de entrada, que é onde ficam os neurônios que recebem os parâmetros de entrada. A última se chama camada de saída, que é onde ficam os neurônios de saída, responsáveis pelas classificações atribuídas pelo modelo. Entre as camadas citadas anteriormente, ficam as camadas ocultas. Estas têm este nome pois seus neurônios estão escondidos entre as camadas de entrada e saída, podendo haver múltiplas camadas ocultas em um modelo. Todas as camadas que não são de entrada ou saída, são consideradas camadas ocultas. Um esquema de uma rede neural pode ser observado na Figura 8.

Figura 8 – Arquitetura de uma rede neural com 2 duas camadas ocultas



Fonte: (Stanford university, 2022)

### 2.3.2 Modelando uma rede neural

A modelagem das camadas de entrada e saída são geralmente simples. O tamanho da camada de entrada é baseado no dado a ser inserido. Por exemplo, se a rede recebe como parâmetros a idade, a altura e o peso de uma pessoa, a camada de entrada possuirá 3 neurônios (ou nós).

Quando se trata da camada de saída, é importante garantir que seu número de nós seja igual à quantidade de valores que desejamos classificar. Por exemplo, uma rede neural cujo objetivo é classificar uma imagem dentre 10 classes possíveis, teria dez saídas.

A modelagem de camadas ocultas possui maior nível de complexidade, pois não é possível resumir seu processo com regras básicas simples. De forma geral, é preferível que ela tenha mais capacidade, ou seja, possua mais neurônios e conseqüentemente mais camadas. Quanto mais camadas e neurônios uma rede neural possuir, melhor ela irá aprender sobre os dados. Porém, a escolha do número de camadas ocultas não é uma simples questão de "quanto mais, melhor". Uma quantidade excessiva de neurônios para um conjunto de dados pequeno pode causar um problema chamado *overfitting*, ou sobre-ajuste.

Quando ocorre o sobre-ajuste, o modelo torna-se tão bom na classificação dos dados utilizados durante a etapa de treinamento, que tem dificuldades em generalizar os resultados e classificar dados de teste. De certa forma, é como se o modelo tivesse decorado o conjunto de dados. Tipicamente utilizam-se técnicas de regularização ou descarte para evitar o sobre-ajuste.

Convencionalmente, classifica-se uma rede neural pela quantidade de camadas que ela possui, excluindo a camada de entrada. Por exemplo, se uma rede neural possui a camada de entrada, a camada de saída e mais duas camadas ocultas, a chamamos de uma rede neural de 3 camadas. A camada de saída normalmente não possui uma função de ativação pois normalmente ela é usada para representar os resultados de uma classificação.

## 2.4 REDES NEURAIAS CONVOLUCIONAIS

Redes neurais convolucionais são uma classe de redes neurais tipicamente utilizada para classificação de imagens. Estas redes possuem arquitetura similar a redes neurais tradicionais. Elas ainda são compostas por neurônios organizados em camadas, havendo as camadas de entrada, ocultas e de saída.

As redes neurais simples, porém, apresentam uma dificuldade ao lidar com o processamento de imagens. Por exemplo, um conjunto de dados clássico de classificação de imagens, o MNIST (DENG, 2012), é composto por imagens de números de dimensões 28x28 em escala cinza, sem informações de cores. Uma rede neural tradicional trabalharia com uma matriz de 784 pesos, onde cada elemento representa um *pixel* da imagem de entrada, o que ainda é um tamanho razoável. Ao aumentar as dimensões das imagens

de entrada, a complexidade computacional crescerá de forma exponencial. Se as imagens forem coloridas, onde temos informações dos canais de cores RGB (vermelho, verde, azul), a complexidade fica ainda maior.

Um modelo de segurança em aeroportos que tente identificar armas de fogo a partir de vídeos em *Full HD* em cores precisaria processar, por segundo, 24 imagens de dimensões  $1920 \times 1080 \times 3$ , equivalente a um total de 6.220.800 pesos para cada neurônio, em apenas um quadro da gravação, uma tarefa que se torna bem complicada computacionalmente.

As redes convolucionais buscam superar esta barreira analisando apenas pedaços da imagem por vez, em vez da imagem como um todo. A rede cria um filtro de dimensões menores e vai dando passos pela imagem, tentando identificar dentro daquele filtro algum elemento que caracterize a imagem como uma dada classe.

As entradas de uma rede neural convolucional são imagens representadas como matrizes de *pixels*. Um *pixel*, ou *picture element*, é o elemento básico de uma imagem. Cada pixel é preenchido com uma cor. Se a imagem estiver em escala cinza, o *pixel* é um único valor numérico que define sua intensidade. Caso a imagem seja colorida, então ele é representado como um vetor de tamanho 3, cada valor numérico representando a intensidade de uma das 3 cores básicas (vermelho, verde, azul). A cor do pixel então, é uma combinação destas 3 cores. Por exemplo, uma imagem de dimensões  $32 \times 32$  tem um total de 1024 *pixels*, e é representada como uma matriz de dimensões  $32 \times 32$ . Cada elemento dessa matriz é um valor numérico (escala cinza) ou um vetor de 3 elementos (colorida). Como a imagem pode ter largura e altura diferentes, a matriz nem sempre será quadrada.

### 2.4.1 Tipos de camadas em uma Rede Neural Convolucional

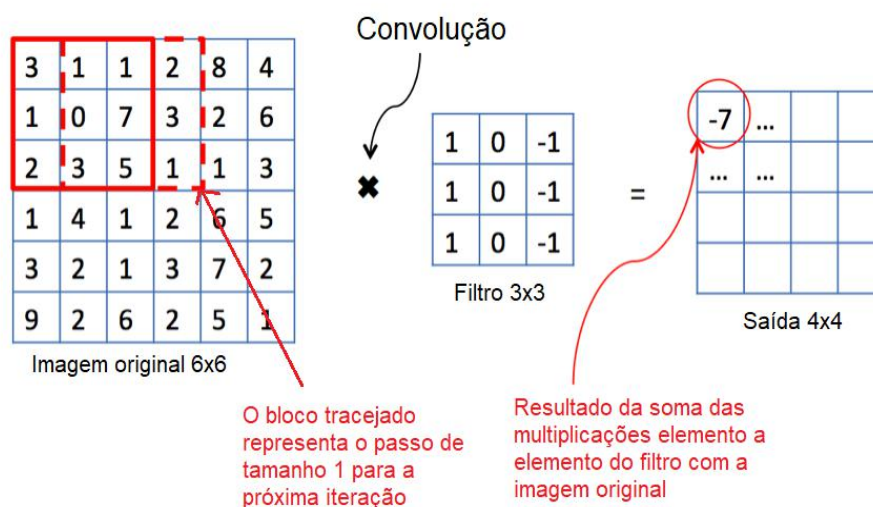
As Redes Convolucionais apresentam novos tipos de camadas especializadas para o tratamento de imagem. A seguir introduzimos a camada de convolução, *pooling* e densamente conectadas.

#### 2.4.1.1 Camadas de Convolução

A camada de convolução é um filtro que possui o objetivo de compactar a imagem em uma de menor complexidade, porém preservando suas propriedades originais. Os principais argumentos desta camada são as dimensões do filtro e o salto. O filtro, ou *kernel*, é uma matriz de pesos.

O procedimento é percorrer a imagem de forma iterativa pegando fragmentos da mesma dimensão do filtro e fazendo uma multiplicação de matrizes elemento a elemento com a matriz que representa o *kernel*. Ao fim de cada iteração, o incremento é o valor do salto. A imagem resultante chama-se *feature map* (mapa de características). Na Figura 9, podemos observar uma iteração da camada de convolução.

Figura 9 – Uma iteração da camada de convolução



Fonte: (LENDAVE, 2021)

Neste exemplo, aplicamos um filtro de dimensões 3x3 com *passo* 1 a uma imagem de 6x6. Obteremos uma saída de dimensões 4x4 ao final de todas as iterações. Com isso, obtém-se um decréscimo de complexidade.

#### 2.4.1.2 Camadas de *pooling*

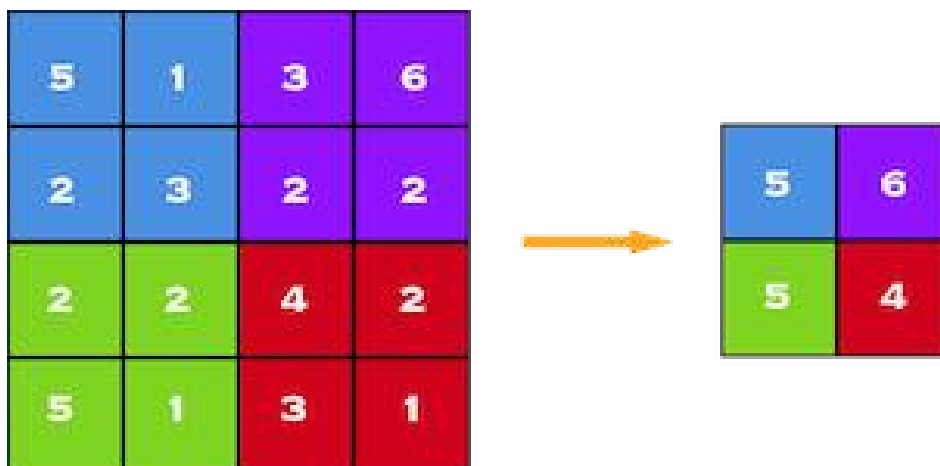
As camadas de *pooling* têm o intuito de reduzir ainda mais a complexidade espacial da representação da imagem, o que reduz a quantidade de parâmetros contabilizados e a complexidade computacional. É uma prática comum colocar camadas de *pooling* após cada camada de convolução.

A camada de *pooling* consiste em descartar elementos da imagem. Existem várias técnicas para selecionar quais itens serão descartados, como o *max*, *average* ou *sum pooling*. Os argumentos da camada são o *tamanho do filtro* e *passo*, assim como na camada de convolução. Os *pixels* da imagem de entrada são divididos de acordo com o *tamanho do filtro* e a heurística de *pooling* é aplicada. No *max pooling*, mantêm-se apenas os *pixels* com o maior valor. No *average pooling*, a média entre os valores. No *sum*, a soma entre todos os valores.

A Figura 10 mostra um exemplo da camada de *pooling* com filtro de dimensões 2x2 e *passo* 2. A técnica utilizada foi *max pooling*. Foi possível reduzir uma imagem 4x4 (16 *pixels*) para uma imagem 2x2 (4 *pixels*).

#### 2.4.1.3 Camada de achatamento

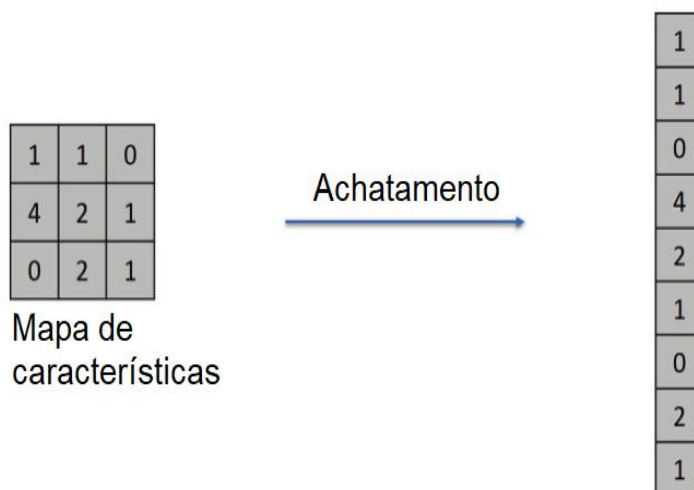
O achatamento consiste em transformar os dados em um vetor unidimensional. Esta camada fica entre a camada de *pooling* e as camadas densamente conectadas (ver seção

Figura 10 – Max *pooling* com filtro 2x2

Fonte: IntroToDeepLearning.com

2.4.1.4).

Figura 11 – Achatamento da matriz de entrada

Fonte: <https://www.superdatascience.com/blogs>

Devido à estrutura das camadas densamente conectadas, se sua entrada for um matriz, o resultado final seria também uma matriz. Porém, por ser a última camada, sua saída deve ser um vetor de uma dimensão, já que estamos interessados apenas na classificação da imagem. Logo, a camada de achatamento precisa ser incluída antes das camadas densas.

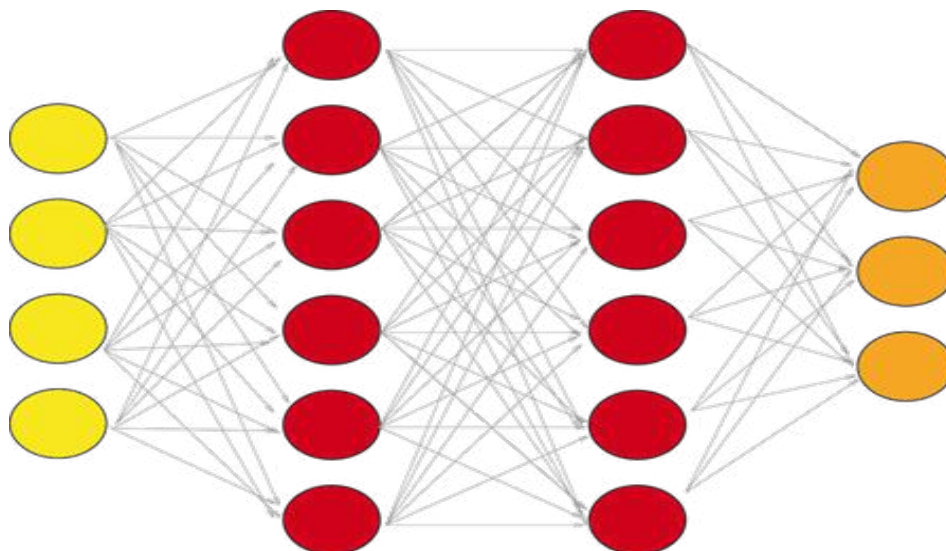
#### 2.4.1.4 Conjunto de camadas densamente conectadas

As camadas densamente conectadas são estruturadas de forma que cada perceptron de uma camada está conectado a todos os outros neurônios da camada posterior. Ou seja,

um neurônio de uma camada densa recebe como entrada as saídas de todos os neurônios da camada anterior. As camadas densas são a etapa final da rede neural convolucional.

Camadas densas são computacionalmente custosas, porém, este problema é mitigado reduzindo a complexidade da imagem através da utilização das camadas de convolução e de *pooling*. A estrutura de camadas densamente conectadas pode ser observada na Figura 12.

Figura 12 – Uma rede densamente conectada com duas camadas ocultas



Fonte: IntroToDeepLearning.com

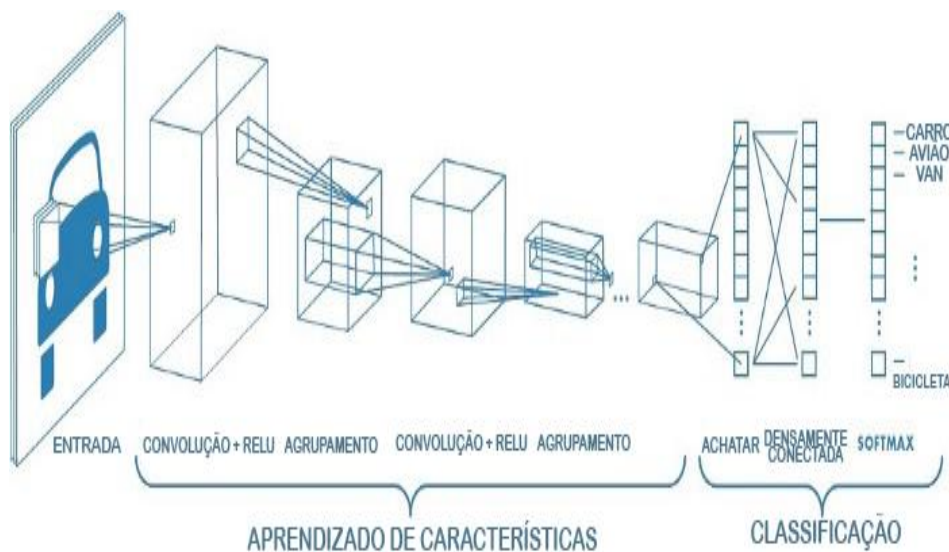
#### 2.4.2 Estrutura final da rede

Com os tipos de camadas apresentados na seção anterior é possível montar uma Rede Neural Convolucional completa. A Figura 13 ilustra uma rede com duas camadas de convolução e duas camadas de *pooling*. Observe que após as camadas de convolução é aplicada uma função de ativação, que no caso é a *ReLU* e que antes de se passar para a rede densamente conectada, os dados tratados são “achatados” em um único vetor de uma dimensão e no final, é aplicada uma função de perda para avaliar o desempenho da classificação.

### 2.5 FUNÇÃO DE CUSTO E GRADIENTE DESCENDENTE

A função de custo utilizada em nossos experimentos será a função de entropia cruzada, pois ela é capaz de minimizar a distância entre duas distribuições de probabilidade (VOVK, 2015). Esta função compara a classe prevista com a classe real e calcula uma

Figura 13 – Exemplo de uma rede neural convolucional



Fonte: IntroToDeepLearning.com

pontuação de acordo com a distância ao valor real. Quanto menor for a perda, melhor é o modelo.

$$Perda(y_{real}, y_{retornado}) = - \sum_{i=1}^n y_{reali} * \log y_{retornadoi} \quad (2.7)$$

A fórmula que define a entropia cruzada pode ser observada na equação (2.7), onde  $y_{esperado}$  é a classe real da imagem, enquanto  $y_{retornado}$  é classificação retornada pelo modelo e  $n$  é o tamanho da saída. Existem vários algoritmos para minimizar a perda, o mais comum sendo o gradiente descendente, visto a seguir.

A premissa do Gradiente Descendente é calcular de que forma os pesos do modelo devem ser alterados de modo que a derivada da função de perda atinja um ponto de mínimo.

O algoritmo é inicializado com um conjunto de pesos iniciais aleatórios. Então, encontramos o gradiente na posição atual e o multiplicamos por um valor pré-definido  $\alpha$  chamado de taxa de aprendizado. Ao final da apresentação de todos os exemplos de treinamento para a rede, a saída é comparada com o valores desejados e calcula-se o valor da função de perda. Esta perda é utilizada para atualizar o valor dos pesos da rede para a próxima iteração.

O passo de iteração do algoritmo está representado na fórmula 2.8. As iterações repetem-se até que o valor mínimo seja encontrado ou o tamanho do *passo* ultrapasse uma tolerância.

$$w_{n+1} = w_n - \alpha \nabla(w_n) \quad (2.8)$$

A escolha da taxa de aprendizado é delicada, pois determina o quão rápido o gradiente descendente irá convergir para o mínimo. Uma escolha inapropriada pode fazer com que sejam necessárias muitas iterações para a convergência do algoritmo, ou até mesmo causar divergência. Tradicionalmente, utilizam-se valores baixos para taxa de aprendizado, como 0.001.

Como em uma rede de processamento de imagens há muitas camadas e por ser aplicado em cada peso, o gradiente descendente pode ser muito custoso computacionalmente. Uma otimização que resolve esse problema é o gradiente descendente estocástico, que consiste em escolher apenas alguns dos pesos aleatoriamente em cada iteração, diminuindo significativamente o custo computacional (SRINIVASAN, 2019).

Outro algoritmo utilizado em nossos experimentos é o Adam, uma alternativa ao gradiente descendente de fácil implementação e computacionalmente eficiente (KINGMA; BA, 2014). Embora este não generalize tão bem quanto o gradiente descendente estocástico, ele possui convergência mais rápida, resultando em um menor tempo de execução, como observaremos mais adiante em nossos experimentos (ZHOU et al., 2020).



### 3 TREINAMENTOS

Neste Capítulo, apresentaremos os experimentos de treinamento utilizando uma rede neural de convolução para o reconhecimento de imagens. Os experimentos têm como objetivo chegar a um modelo que tenha uma acurácia aceitável para a classificação das imagens. Os bancos de imagens utilizados nos experimentos foram o Cifar-10 e o Cifar-100.

A linguagem utilizada nos experimentos foi o Python em sua versão 3.8, com uso da biblioteca Keras do Tensorflow<sup>1</sup>. O Keras é uma biblioteca de código aberto que fornece uma interface de desenvolvimento de aplicações de redes neurais utilizando a biblioteca TensorFlow. O Keras tem o foco em ser uma biblioteca amigável ao usuário, modular e extensível, permitindo experimentação rápida de modelos de aprendizado profundo. Além disso, é amplamente referenciada na literatura e seu código possui alta legibilidade.

Na próxima seção, apresentaremos as funções usadas para implementar o modelo nos nossos experimentos. Todas as funções foram obtidas da documentação da biblioteca Keras(CHOLLET, 2015).

A máquina utilizada para os experimentos tem as seguintes especificações: processador Intel Xeon series 5500, Placa de Vídeo integrada e memória RAM de 8GB.

#### 3.1 FUNÇÕES E PARÂMETROS

Esta seção aborda como as funções do Keras funcionam e como seus parâmetros influenciam no tratamento dos dados.

- a) *sequential()*: Com o Keras, existem duas formas de se construir modelos de redes neurais, o modelo sequencial e o modelo funcional. Este trabalho visa manter o foco no modelo sequencial.

O modelo sequencial é a arquitetura mais comum para o desenvolvimento de redes convolucionais. Ele permite a criação de modelos separados em camadas, que são limitadas a manterem um relacionamento linear de entrada / saída apenas com camadas adjacentes, ou seja, uma camada só pode transmitir dados para a camada de nível subsequente.

- b) *conv2D()*: Esta função é usada para criar uma camada de convolução bidimensional (Ver seção 2.4.1.1). Os argumentos desta função são:
- *filtros*: É um inteiro que define a dimensionalidade do espaço da saída.
  - *tamanho do kernel*: Um inteiro, ou tupla de dois inteiros, que definem a altura e largura da matriz de convolução. O tamanho do *kernel* é o parâmetro usado

---

<sup>1</sup> <https://www.tensorflow.org/>

durante a convolução para reduzir a complexidade no espaço das imagens do treinamento.

- *formato de entrada*: O formato de entrada é referente à dimensionalidade dos dados de entrada. Como o conjunto de dados estudado é composto por imagens coloridas de 32x32 pixels, o computador faz o uso de três matrizes para armazenar estes dados. Cada uma delas possui dimensão 32x32, correspondendo a um dos canais do espaço RGB, o que faz com que a entrada seja uma matriz tridimensional no formato (32,32,3). Cada elemento da matriz é um número entre 0 e 255, correspondendo à intensidade de um pixel em 8-bits. O formato de entrada é um argumento que precisa necessariamente ser passado na primeira camada do modelo, uma vez que é responsável por explicitar o formato das matrizes que virão a ser multiplicadas.
  - *padding*: É um parâmetro que faz o tratamento das bordas em imagens adicionando valores 1 às extremidades, de modo que a saída mantenha a mesma razão de largura / altura do que a imagem de entrada. Os valores do argumento podem ser *valid* (sem preenchimento, valor padrão) ou *same* (com preenchimento)(BROWNLEE, 2018).
  - *activation*: Define qual função de ativação será usada na camada de convolução.
- c) *maxpooling2D()*: Função para criar uma camada de *pooling* bidimensional usando a heurística MAX (Ver seção 2.4.1.2). O argumento desta função é:
- *Pool Size*: Um inteiro, ou um tupla de dois inteiros, que define a dimensão da janela do *pooling*.
- d) *dropout()*: A camada de *dropout* define o valor da porcentagem de neurônios que serão descartados. Seu argumento pode assumir valores de 0 a 1.
- e) *flatten()*: Camada de achatamento do vetor de entrada. Achar um vetor multidimensional, significa transformá-lo em um vetor unidimensional. Por exemplo, nosso vetor de entrada com dimensões(*none*, 32, 32, 3) torna-se um vetor com uma única dimensão de 3072 (=32\*32\*3) posições, representado por (*none*, 3072). O "*none*" é um valor que recebe o tamanho do lote (*batch size*, definido abaixo. Por exemplo, se o tamanho do lote for 32, então (*none*, 3072) será (32, 3072). Uma boa prática é achar o vetor de entrada antes de passar para a camada densamente conectada, evitando-se que hajam múltiplas saídas para cada neurônio.
- f) *dense()*: Cria uma camada densamente conectada de um modelo de rede neural simples. Seu argumento é:
- *unidades*: Dimensionalidade da saída. Indica quantos neurônios existirão na passagem para a próxima camada. Na última camada, o valor deste argumento deve ser necessariamente igual ao número de classes que desejamos classificar.

- g) *compile()*: Método para compilar o modelo. Os argumentos são:
- *otimizador*: Define o otimizador utilizado. Nos experimentos, utilizaremos os otimizadores *gradiente descendente estocástico* e *adam* (ver seção 2.5).
  - *perda*: Função de perda utilizada.
  - *métrica*: Métricas usadas para avaliar o modelo. Em nossos experimentos, estaremos interessados em avaliar a acurácia.
- h) *fit()*: Método para treinar o modelo.
- *época*: Uma época é um parâmetro que define o número de vezes que o algoritmo de aprendizado vai executar sobre o conjunto de dados (BROWNLEE, 2019). Utilizando os pesos atualizados na época anterior como ponto de partida na época atual, garantimos que uma época sempre tente obter melhores resultados em relação ao que foi feito na época anterior, ao invés de começar do zero com novos valores aleatórios. Valores de épocas tradicionalmente utilizados na literatura são múltiplos de 10.
  - *tamanho da amostra*: O tamanho do lote é um parâmetro que define o número de amostras que são treinadas antes de atualizar os outros parâmetros do modelo. Um lote é como um *loop* que itera sobre cada elemento da amostra fazendo previsões. Ao final de cada lote, as previsões são comparadas aos valores de saída esperados e o erro é calculado. A partir deste erro, o modelo é melhorado. Tamanhos de lote pequenos costumam produzir resultados melhores, pois há mais atualizações nos parâmetros (BROWNLEE, 2019). Se houver 100 imagens no treinamento, e o tamanho do lote for 10, haverá 10 iterações durante cada época. Existem três tipos de tamanho do lote. No *batch mode*, o tamanho do lote é igual ao número de imagens. Neste caso, haverá uma única iteração por época. No *stochastic mode*, o tamanho do lote é igual a um. O número de iterações por época será igual ao tamanho do conjunto de dados. No *mini-batch mode*, o tamanho do lote é maior do que 1, porém menor do que o tamanho do conjunto de dados. Os valores mais utilizados na literatura são 32, 64 e 128. Portanto, usaremos estes valores nos experimentos.

## 3.2 EXPERIMENTOS SOBRE O CIFAR 10

### 3.2.1 Base de Dados do Cifar 10

O Cifar-10 (KRIZHEVSKY, 2009) é um conjunto de 60.000 imagens em cores, com 32 *pixels* de altura por 32 *pixels* de largura, dividido em dez classes compostas por 6.000 imagens cada. As classes presentes no conjunto de dados são: *avião*, *automóvel*, *pássaro*, *gato*, *veado*, *cachorro*, *sapo*, *cavalo*, *navio* e *caminhão*, e não existem imagens que pertençam a mais de uma classe ao mesmo tempo. Para realizar os experimentos, o conjunto é

separado em 50.000 imagens para treino e 10.000 imagens para teste, mantendo a mesma proporção de separação para todas as classes presentes.

### 3.2.2 Experimentos

Foram realizados experimentos alterando-se os valores dos parâmetros e a quantidade de camadas utilizadas.

Nossa primeira arquitetura foi baseada em valores referenciados pela literatura, baseado em experimentos que realizamos sobre a base MNIST (YALÇIN, 2018). Por se tratar de uma base mais simples e mais bem comportada do que o cifar, é mais fácil de se obter resultados desejáveis.

No experimento, utiliza-se uma camada de convolução, seguida por uma camada de *pooling* e uma camada densamente conectada. Neste momento, a intenção era apenas observar o comportamento da rede, para fazer nos experimentos seguintes, um estudo de quais parâmetros contribuem mais para a obtenção de resultados melhores. Assim, construiremos incrementalmente uma arquitetura melhor. Com base nos parâmetros do experimento original, foram propostas várias modificações.

A primeira modificação testada (experimento 2) foi a alteração do tamanho do lote de modo estocástico para *mini-batch mode*, usando valor 32. Com esta mudança, temos 32 imagens sendo processadas de uma vez. Devido a isto, espera-se que o treinamento seja mais rápido, já que há menos iterações por época.

No Experimento 3, alteramos a função de ativação das camadas densas de Sigmoid para ReLU, afim de contornar o problema de dissipação do gradiente (*Vanishing Gradient Problem*) e, com isso, aumentar a acurácia dos resultados. A seguir, no experimento 4, retornamos o tamanho do lote para 32 a fim de melhorar o tempo de execução.

No experimento 5, foi alterado o tamanho do Kernel de (5,5) para (3,3). Com base na teoria, diminuí-lo deve reduzir, durante o passo de convolução, a perda de informação dos pixels da imagem original em relação a seus vizinhos. Com menos perda de informação durante o passo da convolução, também se espera a diminuição do sobreajuste (ver seção 2.3.2), e um aumento na acurácia. A seguir, no experimento 6, alteramos o *batch size* (tamanho do lote) para 64, a fim de avaliar se obtemos nova melhora no tempo de execução.

O sétimo experimento introduziu o uso de camadas de *dropout*. O objetivo é de que, desligando alguns neurônios aleatoriamente, a rede se torne mais sensível aos dados de entrada e menos enviesada em relação a certas entradas, diminuindo o problema do sobreajuste.

No oitavo experimento, inserimos três blocos VGG. A arquitetura VGG é nomeada em homenagem ao *Visual Geometry Group* da universidade de Oxford (SIMONYAN; ZISSERMAN, 2014) e consiste em blocos de uma ou mais camadas convolucionais de *kernels* pequenos (como 3x3), com passo 1 e *padding "same"*. Essas camadas são então

seguidas por uma camada de *Max Pooling* com tamanho 2x2 e tamanho do passo na mesma dimensão. No nosso caso, utilizamos três blocos VGG, com 32, 64 e 128 filtros, respectivamente. Cada bloco é composto por 2 camadas de convolução, que não utilizam a função de ativação, e por uma camada de *pooling*, seguidas por uma camada de *dropout*.

No nono experimento, mantivemos os parâmetros aumentando o número de épocas de 10 para 50, afim de verificar se o maior tempo de treinamento acarretaria em uma melhor taxa de acurácia. No experimento 10, foi acrescentada a função de ativação ReLU às camadas de convolução e voltamos ao número de 10 épocas para o treinamento.

Por fim, no experimento 11, alteramos o otimizador de gradiente descendente estocástico para o adam, a fim de obter um melhor tempo de execução, pois este otimizador converge mais rapidamente, e podemos obter resultados satisfatórios com menos épocas.

### 3.2.3 Resultados

A métrica utilizada para avaliar o treinamento é a acurácia. É uma métrica suficiente para este conjunto de dados pois ele é simétrico (mesmo número imagens por classe). Caso houvesse desequilíbrio entre as classes, sob risco de muitos falsos-positivos ou muitos falsos-negativos, outras métricas seriam necessárias. Cada experimento foi executado dez vezes, e obteve-se a acurácia média de cada execução.

No experimento 1, a acurácia média foi de 46.110. No experimento 2, observamos uma pequena melhora na acurácia, que passou a ser 52.005, mas uma melhora muito significativa e também esperada, no tempo de execução, que passou de 2 horas para 10 minutos.

No experimento 3, esperava-se uma melhora devido a função de ativação ReLU. Porém, a acurácia resultante foi de apenas 0.1. Uma vez que temos dez classes no total e elas estão distribuídas uniformemente nos conjuntos de treino e teste, a hipótese era de que o modelo estava sempre predizendo apenas uma única classe, o que acarretaria no acerto de 10% dos casos. Ao verificarmos a saída das predições, foi observado que, de fato, o algoritmo sempre estava prevendo uma mesma classe arbitrária em todas as instâncias. Tentamos investigar a causa, porém nossos resultados foram inconclusivos.

No experimento 4, esperávamos que ao retornar o tamanho do lote de 1 para 32, obtivéssemos uma melhora no tempo de execução. Porém, não apenas conseguimos esta melhora, como o problema encontrado no experimento anterior (acurácia muito baixa) foi resolvido. Obteve-se uma acurácia média de 60.905, nosso melhor resultado até o momento, com tempo de execução de apenas de 10 minutos. Comparando-se o experimento 2 com o experimento 4, nos quais a única diferença é a função de ativação, podemos concluir que a função de ativação ReLU obtém melhores resultados do que a sigmoíde. Portanto, a partir de então, utilizaremos esta função de ativação.

No experimento 5, esperava-se uma melhora na acurácia devido à redução do tamanho do kernel. Porém, esta melhora não foi observada nos resultados. A acurácia manteve-se

Tabela 1 – Parâmetros Cifar 10

Parâmetros	Exp 1	Exp 2	Exp 3	Exp 4	Exp 5	Exp 6	Exp 7	Exp 8	Exp 9	Exp 10	Exp 11
Camadas de Convolação	X	X	X	X	X	X	X	X	X	X	X
Kernel Size	X	X	X	X	X	X	X	X	X	X	X
Filtros	X	X	X	X	X	X	X	X	X	X	X
Camadas Pool	X	X	X	X	X	X	X	X	X	X	X
Pool Size	X	X	X	X	X	X	X	X	X	X	X
Camadas Dense	X	X	X	X	X	X	X	X	X	X	X
Função de Ativação	X	X	X	X	X	X	X	X	X	X	X
Dropout	X	X	X	X	X	X	X	X	X	X	X
Otimizador	X	X	X	X	X	X	X	X	X	X	X
Função de Perda	X	X	X	X	X	X	X	X	X	X	X
Épocas	X	X	X	X	X	X	X	X	X	X	X
Tamanho do Lote	X	X	X	X	X	X	X	X	X	X	X

similar ao experimento anterior (60.635). Portanto, concluímos que esta alteração não nos trouxe um resultado relevante.

No experimento 6, alteramos o tamanho do lote de 32 para 64, esperando-se uma melhora no tempo de execução. Esta melhora foi muito pequena, de apenas 2 minutos, e observou-se uma pequena diminuição de acurácia, que foi para 56.115. Porém, como os experimentos seguintes serão redes mais complexas, manteremos este tamanho de lote para que o tempo de execução seja mais razoável e não consuma muitos recursos da máquina. A perda de acurácia não foi significativa o suficiente para justificar o uso do tamanho de lote 32.

No experimento 7, esperava-se que com a inclusão da camada de *dropout* implicasse em uma melhora na acurácia. Porém, isto não se refletiu no resultado, que se manteve similar ao experimento anterior. Porém, como os experimentos seguintes terão maior complexidade, manteremos esta camada.

No experimento 8, com a inclusão dos blocos VGG, obtivemos uma melhoria significativa na acurácia média, que passou a ser 66.330, como esperado.

No experimento 9, com o aumento do número de épocas, esperava-se uma melhora na acurácia. Isto se refletiu nos resultados. Obtivemos uma acurácia de 78.170. Porém, o tempo de execução foi 4 horas por rodada do experimento.

No experimento 10, incluímos uma função de ativação na camada de convolução. Para manter um tempo de execução razoável, retornamos a 10 épocas, porém o resultado não foi satisfatório.

No experimento 11, alteramos o otimizador para adam, a fim de obtermos os bons resultados do experimento 9 com um tempo de execução mais razoável. A acurácia média obtida foi de 78.460, resultado similar ao experimento 9, porém com tempo de execução bem menor, de em média uma hora por rodada.

Com base nos testes acima, conseguimos chegar em um conjunto de parâmetros para seguir com a próxima fase dos testes.

### 3.2.4 Experimentos do Cifar10 Separados por Classes

A seguir, separamos os resultados obtidos no experimento 11 por classe, ou seja, medimos o valor da acurácia para cada classe do conjunto de dados individualmente. O experimento foi executado dez vezes, e tiramos a média das acurácias de cada classe, assim como da acurácia geral.

Observamos uma tendência do modelo de não conseguir classificar alguns animais tão bem quanto as demais classes. *Pássaro*, *Gato*, *Veado* e *Cachorro*, por exemplo, obtiveram notas bem inferiores do que as demais classes, e isso influenciou a acurácia geral, chegando ao valor final de 0.786.

A hipótese que temos para justificar esse comportamento é que os animais são muito semelhantes entre si, porém são distintos das demais categorias (como *Avião* e *Navio*),

Tabela 3 – Resultados Cifar 10

Experimentos	Acurácia	Desvio padrão	Tempo de Execução (HH:MM:SS)
Experimento 1	0.46	0.0125	01:42:00
Experimento 2	0.52	0.0137	00:10:38
Experimento 3	0.1	0	01:56:00
Experimento 4	0.6	0.0087	00:10:52
Experimento 5	0.6	0.0109	00:10:16
Experimento 6	0.56	0.0243	00:08:38
Experimento 7	0.55	0.0292	00:09:23
Experimento 8	0.66	0.0192	00:53:46
Experimento 9	0.78	0.005	04:22:46
Experimento 10	0.54	0.0308	00:54:14
Experimento 11	0.78	0.0081	00:58:12

Tabela 4 – Experimento 11: Resultados separados por Classe

Classe	Acurácia	Desvio padrão
Automóvel	0.897	0.0392
Navio	0.893	0.0365
Caminhão	0.874	0.0379
Cavalo	0.816	0.0639
Sapo	0.860	0.037
Avião	0.797	0.0505
Veado	0.745	0.0371
Cachorro	0.686	0.0404
Pássaro	0.655	0.0147
Gato	0.625	0.0462
Média Geral	0.786	0.0403

sendo assim, o modelo tem dificuldade de atribuir a eles uma classe específica e acaba errando a classificação com maior frequência.

Para testar essa hipótese, fizemos dois experimentos em que alteramos o conjunto de dados original. No primeiro, iremos isolar ao animal Gato, que tem o pior desempenho, e treinar nosso modelo usando apenas esta categoria de animal junto com as demais classes que não são animais (Avião, Automóvel, Navio e Caminhão). Em seguida, iremos testar o modelo apenas com os animais, tendo então seis classes: Pássaro, Gato, Veado, Cachorro, Sapo e Cavalo.

No experimento onde removemos todas as classes de animais, exceto a classe gato, obtivemos melhora significativa tanto na classe gato individualmente, como no conjunto de dados em geral. Porém, no experimento onde utilizamos apenas as classes de animais,



Figura 14 – Boxplot dos experimentos

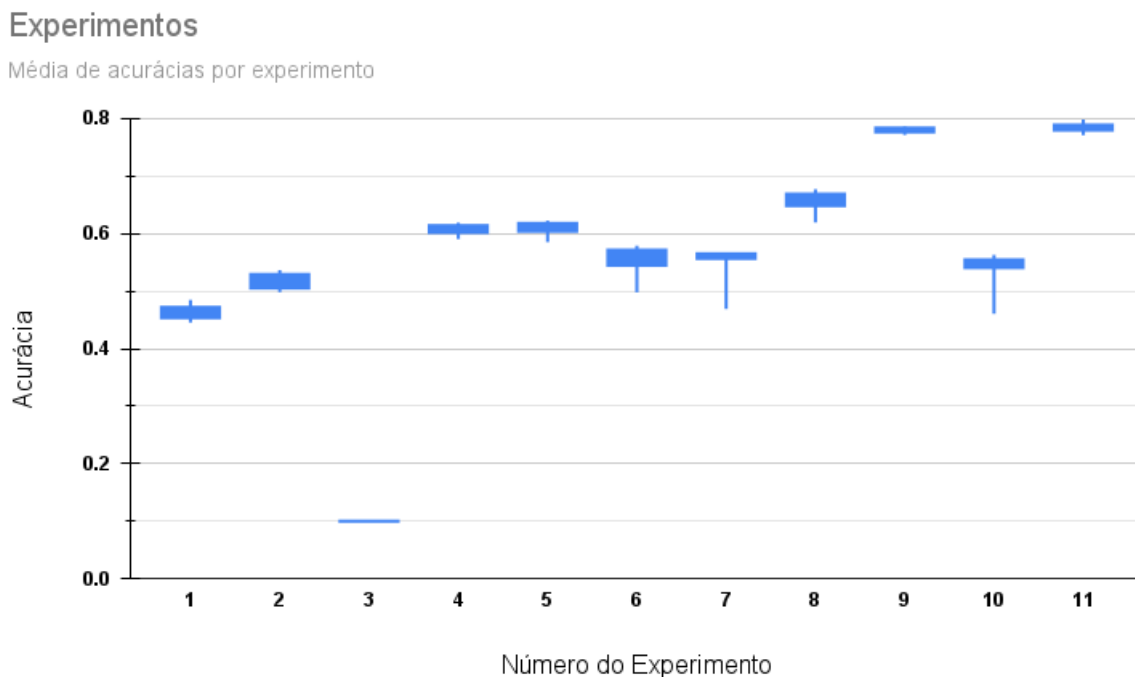


Tabela 5 – Experimento 11: Resultados apenas com a classe Gato

Classe	Acurácia	Desvio padrão
Gato	0.92	0.0739
Automóvel	0.90	0.0216
Navio	0.88	0.0337
Caminhão	0.87	0.0323
Avião	0.85	0.0503
Geral	0.895	0.0423

a acurácia da classe dos gatos manteve-se similar à do experimento 8. Além disso, a acurácia geral também foi inferior. A exceção foi a classe dos cavalos, que no experimento 8, com as dez classes, já destacava-se em relação aos outros animais.

### 3.3 CIFAR 100

#### 3.3.1 Base de Dados do Cifar100

Após diversos experimentos com o cifar-10 e de termos encontrado um modelo que obtém resultados satisfatórios, foram iniciados novos experimentos, utilizando o conjunto de dados cifar-100. O cifar-100 é um conjunto de 60.000 imagens, semelhante ao cifar-10, porém, dividido em 100 classes, cada uma constituída por 500 imagens para treino e

Figura 15 – Boxplot dos experimentos: Separados por classes

## Experimento 11

Resultados por Classe

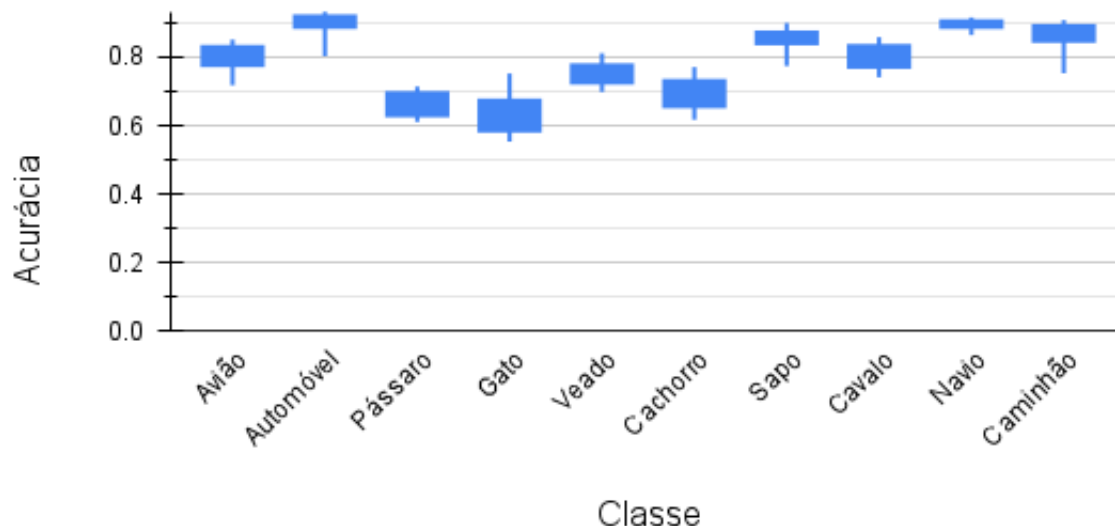


Tabela 6 – Experimento 11: Resultados apenas com os animais

Classe	Acurácia
Cavalo	0.87
Pássaro	0.76
Sapo	0.75
Cachorro	0.69
Veado	0.69
Gato	0.67
Geral	0.7368

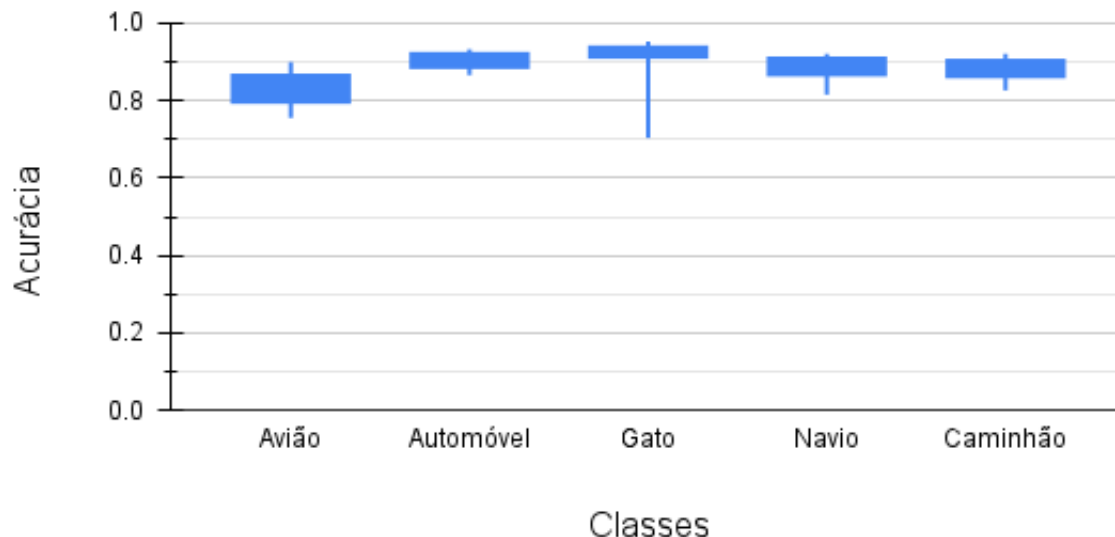
100 imagens para teste, ou seja, mantém-se a quantidade total de imagens, mas existem dez vezes mais classes e dez vezes menos instâncias para cada classe. As classes, estão agrupadas em 20 super-classes, as quais possuem 5 sub-classes cada. Cada imagem possui um rótulo que indica tanto sua sub-classe, quanto a super-classe correspondente. Em relação as características de altura, largura e cores, o cifar-100 herda as características do cifar-10 utilizando as mesmas proporções e não existindo imagens que pertençam a mais de uma classe.

As superclasses incluídas no cifar-100 são: *mamíferos aquáticos, peixes, flores, recipientes de comida, frutas e vegetais, eletrodomésticos, móveis, insetos, carnívoros de grande porte, grandes construções artificiais, grandes cenários naturais, onívoros e herbívoros de*

Figura 16 – Boxplot dos experimentos: Sem animais, apenas classe gato

## Experimento 11

Excluindo Classes Animais



*grande porte, mamíferos de médio porte, invertebrados excluindo insetos, pessoas, répteis, mamíferos de pequeno porte, árvores, veículos 1, veículos 2.*

Todas as classes e superclasses do cifar 100 estão listadas no apêndice A.

### 3.3.2 Experimentos cifar-100

No experimento com o cifar-100, foi utilizada a arquitetura do experimento 11 do cifar-10, na qual obtivemos os melhores resultados nas execuções. A única alteração foi o número de neurônios de saída, que passou a ser 100, pois este parâmetro deve ser igual ao número de classes. Neste experimento, o modelo foi executado com o cifar-100 sem alterações, com as classificações sendo realizadas através das 100 classes. O objetivo era verificar se um modelo que obteve sucesso para o cifar-10, também seria capaz de alcançar bons resultados para o cifar-100 sem que houvessem adaptações críticas. Os resultados do cifar-100 não foram tão satisfatórios quanto os obtidos no cifar-10, pois foi obtido uma acurácia de apenas de 0.37 neste experimento comparado a acurácia de 0.78 com o cifar-10.

Devido a baixa acurácia obtida no experimento, foi decidido fazer um novo experimento onde ao invés de utilizar 100 classes no treinamento e classificação, utilizaríamos apenas as superclasses, de modo a aumentar o número de imagens por classe. O objetivo é verificar se há melhoria na acurácia geral e identificar quais classes se beneficiariam em ser

agrupadas. Utilizamos um mapeamento para agrupar todas as subclasses em suas respectivas superclasses indicadas pela página do cifar-100, pois durante o desenvolvimento foi identificado que na instância do cifar-100 que estava sendo utilizada, as imagens não possuíam o rótulo de superclasse. Devido a forma como o cifar-100 é estruturado, não existia risco de haver quantidades diferentes de imagens para cada superclasse. O experimento foi executado dez vezes.

Tabela 7 – Resultados apenas com as superclasses do cifar 100

Classe	Acurácia	Classe	Acurácia
Animais Aquáticos	0.429	Peixes	0.463
		0.782	
Flores	0.716	Recipientes de Comida	0.576
Frutas e Vegetais	0.661	Eletrodomésticos	0.393
Móveis	0.655	Insetos	0.531
Carnívoros de Grande Porte	0.489	Grandes Construções	0.677
Grandes Cenários	0.735	Onívoros/Herbívoros de Grande Porte	0.439
Mamíferos de Médio Porte	0.357	Invertebrados Não Insetos	<b>0.264</b>
Pessoas	0.676	Répteis	0.321
Mamíferos de Pequeno Porte	0.4	Árvores	<b>0.849</b>
Veículos 1	0.617	Veículos 2	0.53

Observando os resultados obtivemos uma acurácia geral de 0.537. Verificamos que a classe com acurácia mais baixa foi a dos invertebrados, de 0.264, e a com acurácia mais alta foi a das árvores, de 0.849. Na tabela 7, temos a acurácia média de cada superclasse nos dez experimentos. Na Tabela 8, apresentamos a comparação entre as acurácias gerais do experimento realizado com as cem classes com o experimento onde utilizamos apenas as vinte superclasses.

Um problema da base de dados do cifar-100 é sua quantidade de imagens relativamente pequena. Ao passo que o cifar-10 possui 6000 imagens por classe, o cifar-100 possui apenas 600 imagens por classe e 3000 por superclasse. Com o intuito de melhorar nosso modelo, foi decidido utilizar o método de *data augmentation* (definido na seção a seguir), que consiste em aumentar artificialmente a quantidade de imagens em nossa base de dados.

Tabela 8 – Cifar-100 sem utilizar superclasses e utilizando superclasses

Experimentos	Acurácia Geral
Experimento 100 classes	0.373
Experimento 20 superclasses	0.537

### 3.3.3 Experimentos Utilizando Data Augmentation

Este método é útil quando precisamos realizar treinamentos com bases menores. Ele consiste em utilizar imagens já existentes no conjunto de dados e fazer alterações nessas imagens tais como girar, rotacionar, cortar ou mudar escala e acrescentar as imagens alteradas na base de dados.

A biblioteca Keras possui uma implementação de *data augmentation*, chamada pelo método *ImageDataGenerator()*. As técnicas a serem utilizadas na geração de imagens são passadas através do argumentos desta função. Alguns exemplos de argumentos apresentados na documentação do Keras são:

- a) *horizontal flip*: Booleano, vira a imagem horizontalmente de forma aleatória.
- b) *vertical flip*: Booleano, vira a imagem verticalmente de forma aleatória.
- c) *rotation range*: Inteiro, rotaciona a imagem aleatoriamente, até o valor dado (em graus).
- d) *width shift range*: Inteiro, float ou array. Desloca a imagem para esquerda ou para a direita.
- e) *height shift range*: Inteiro, float ou array. Desloca a imagem para cima ou para baixo.

O método gera um número de imagens igual ao tamanho da amostra (*batch size*) para cada iteração. Se temos um conjunto com 600 imagens, e amostras de tamanho 100, teremos 6 iterações. Em cada iteração, serão produzidas 100 imagens novas por amostra, totalizando 600 imagens em uma época.

Em nossos experimentos, utilizaremos o *shift* e o *flip* horizontal. As imagens geradas foram utilizadas apenas para treinamento e os testes foram realizados com o conjunto de dados original.

A partir dos resultados observados nas tabelas, podemos concluir que houve uma melhora pequena em nossa acurácia ao aplicar a técnica de *data agumentation*, com a acurácia geral passando de 0.537 para 0.541.

Observando os resultados separados por classes, também observamos melhorias pequenas, com algumas classes observando um aumento na acurácia, enquanto outras se mantêm ou pioram. A classe de flores destacou-se com uma melhoria expressiva na acurácia, passando de aproximadamente 70% para quase 80%. A classe das árvores, no entanto,

Tabela 9 – Resultados com as superclasses do cifar 100, utilizando data augmentation

Classe	Acurácia	Classe	Acurácia
Animais Aquáticos	0.426	Peixes	0.498
Flores	0.782	Recipientes de Comida	0.592
Frutas e Vegetais	0.608	Eletrodomésticos	0.446
Móveis	0.646	Insetos	0.562
Carnívoros de Grande Porte	0.534	Grandes Construções	0.67
Grandes Cenários	0.742	Onívoros/Herbívoros de Grande Porte	0.46
Mamíferos de Médio Porte	0.362	Invertebrados Não Insetos	<b>0.218</b>
Pessoas	0.656	Répteis	0.306
Mamíferos de Pequeno Porte	0.378	Árvores	<b>0.828</b>
Veículos 1	0.646	Veículos 2	0.494

Tabela 10 – Cifar-100 sem e com *data augmentation*

Experimentos	Acurácia Geral
Sem <i>augmentation</i>	0.537
Com <i>augmentation</i>	0.541

obteve desempenho pior se comparado ao experimento anterior, mas manteve-se como a classe com o melhor desempenho.

### 3.3.4 Treinando Classes Isoladamente

Nosso próximo objetivo é identificar a existência de correlação entre a acurácia geral da superclasse e a acurácia individual de suas classes. Para isto, selecionamos duas superclasses, a que possui o melhor e o pior desempenho. Estas superclasses são as árvores e invertebrados, respectivamente.

Separamos ambas superclasses em suas respectivas subclasses (como vistas no apêndice A), e executamos o mesmo experimento anterior, com *data augmentation*. O experimento foi executado 10 vezes, e obtivemos a acurácia média de cada uma das classes.

Na tabela 11, apresentamos os resultados apenas com a superclasse dos invertebrados e na tabela 12, apenas com a superclasse das árvores.

Tabela 11 – Resultados apenas com a superclasse dos invertebrados

Classe	Acurácia
Caranguejo	<b>0.42</b>
Lagosta	0.495
Lesma	0.5
Aranha	<b>0.615</b>
Minhoca	0.6

Tabela 12 – Resultados apenas com a superclasse das árvores

Classe	Acurácia
Bordo	0.445
Carvalho	<b>0.79</b>
Palmeira	0.705
Pinheiro	<b>0.31</b>
Salgueiro	0.465

Dentre os invertebrados, a melhor acurácia foi a da classe das aranhas, ao passo que o pior desempenho foi da classe dos caranguejos. Dentre as árvores, a classe com o melhor desempenho individual foi a dos carvalhos, enquanto que a pior acurácia foi a dos pinheiros.

Não foi possível observar nenhuma correlação aparente entre o desempenho individual das subclasses e sua respectiva superclasse. Uma observação interessante, porém, é a amplitude das acurácias entre as classes de árvores, variando de aproximadamente 30% (pinheiro) a quase 80% (carvalho).

### 3.3.5 Conclusão dos experimentos

Observamos que a premissa de reaproveitar a modelagem abordada no caso de estudo do conjunto de dados MNIST não foi nada promissora. Depois de muitos experimentos, conseguimos construir uma modelagem satisfatória para o CIFAR-10. Porém, esta modelagem, quando aplicada ao conjunto de dados Cifar-100, produziu resultados insatisfatórios, criando a necessidade de novos experimentos.

As diferentes características entre cada conjunto de dados nos possibilitou estudar os impactos de cada hiperparâmetro na rede. Fomos capazes de aplicar este conhecimento para gerar melhorias incrementais nos resultados entre cada experimento. Porém, a modelagem ainda não foi capaz de produzir uma boa acurácia para o Cifar-100. Podemos presumir que o motivo é este conjunto de dados possuir uma menor quantidade de imagens

por classe. Ao passo que no Cifar-10 possuímos 1000 imagens por classe, no Cifar-100 há apenas 100. Foi possível mitigar este problema utilizando os rótulos de superclasse do cifar-100, porém, não foi possível resolvê-lo por completo, pois ainda há apenas 500 imagens por superclasse, metade do conjunto de dados original. Para solucionar o problema da distribuição de imagens por classe no Cifar-100, tentamos implementar a técnica de *data augmentation*, que não produziu as melhorias esperadas.

Durante os experimentos, aprendemos que a quantidade de épocas, tamanho do lote e quantidade de camadas de *pooling* e convolução foram os hiperparâmetros que mais influenciaram no desempenho dos treinamentos. As mudanças obtidas a partir da alteração de outros hiperparâmetros tais como tamanho do *kernel* e inclusão de camadas de *dropout* produziram alterações bem menos perceptíveis. Também aprendemos sobre a influência dos algoritmos de otimização. O adam foi capaz de reproduzir um resultado estatisticamente igual ao gradiente descendente estocástico com um menor número de épocas, resultando em menos tempo de execução.



Figura 17 – Imagem original, flip horizontal e flip vertical, respectivamente



## 4 CONCLUSÃO

Podemos citar como um desafio a transformação do conhecimento teórico em uma modelagem computacional. Começamos utilizando a biblioteca *pytorch*, porém a curva de aprendizado mostrou-se muito complexa. Optamos, então, pela utilização do *Keras*, que possui melhor legibilidade e maior simplicidade. Para nos familiarizarmos com a biblioteca e suas funcionalidades, realizamos experimentos sobre o conjunto de dados MNIST, apenas com objetivo de aprendizado.

Outro desafio técnico encontrado foram as especificações do servidor que tínhamos à disposição, que não possuía placa de vídeo e apresentava intermitências em seu período *online*. No início do trabalho, o objetivo era a realização dos experimentos sobre o conjunto de dados *ImageNet* (DENG et al., 2009), porém este conjunto de dados não era bem estruturado. As imagens disponíveis não tinham dimensões padronizadas e o uso deste conjunto requeria a instanciação de forma avulsa para cada imagem a ser trabalhada. O custo computacional de armazenar as imagens localmente e redimensioná-las em tempo de execução acabou se mostrando muito alto para o servidor disponível. Devido a estes fatores, decidiu-se pela utilização do conjunto de dados Cifar-10 e Cifar-100, cujas dimensões são padronizadas, e menores, para todas as imagens e com a utilização do *Keras* não era necessário instanciá-las localmente no servidor.

Otimizar o tempo de execução também mostrou-se desafiador. Aumentar o número de épocas apresenta um impacto bastante positivo nos resultados, porém, às custas de um aumento considerável no tempo de execução. Um treinamento de 50 épocas leva aproximadamente 4 horas por rodada de execução, enquanto um de 10 épocas leva 50 minutos. Este fator também limitou a quantidade de vezes em que os experimentos foram executados, realizar mais de 10 rodadas por experimento se tornaria muito custoso.

Gostaríamos de ter realizado experimentos com maior número de épocas, principalmente para o Cifar-100, visando a obtenção de melhores resultados. Uma outra oportunidade de estudo, seria realizar mais experimentos com *data augmentation*, analisando os impactos de cada argumento da função geradora. O próximo passo do estudo, seria a construção de um modelo para o conjunto de dados *ImageNet*, nosso objetivo original, analisando também outras métricas, como *precisão*, *recall* e *F1-score* e realizando testes de hipótese sobre os resultados obtidos.

Enxergamos a possibilidade de expandir este trabalho para abranger o estudo da técnica de *transfer learning*, a qual nos daria a flexibilidade para tentar aplicar ao nosso problema o resultado do treinamento de um outro modelo treinado previamente, seja este feito por nós ou por algum outro estudo. Outra técnica que poderíamos explorar em trabalhos futuros seriam as Redes Neurais Adversariais Generativas (GANs) (SAXENA; CAO, 2021), um modelo computacional que coloca duas, ou mais, arquiteturas de redes

neurais para competir entre si e assim chegar a melhores resultados.

## REFERÊNCIAS

- BROWNLEE, J. **Difference Between a Batch and an Epoch in a Neural Network**. 2018. Disponível em: <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>.
- BROWNLEE, J. **Padding and Stride for Convolutional Neural Networks**. 2019. Disponível em: <https://machinelearningmastery.com/padding-and-stride-for-convolutional-neural-networks/>.
- BROWNLEE, J. **Loss and Loss Functions for Training Deep Learning Neural Networks**. 2020. Disponível em: <https://machinelearningmastery.com/softmax-activation-function-with-python/>.
- BROWNLEE, J. **Metrics To Evaluate Machine Learning Algorithms in Python**. 2020. Disponível em: <https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/>.
- CHOLLET, F. **Keras**. [S.l.]: GitHub, 2015. <https://github.com/fchollet/keras>.
- DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: **IEEE. 2009 IEEE conference on computer vision and pattern recognition**. [S.l.], 2009. p. 248–255.
- DENG, L. The mnist database of handwritten digit images for machine learning research. **IEEE Signal Processing Magazine**, IEEE, v. 29, n. 6, p. 141–142, 2012.
- HOCHREITER, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. **International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems**, v. 06, n. 02, p. 107–116, 1998. Disponível em: <https://doi.org/10.1142/S0218488598000094>.
- KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1412.6980>.
- KRIZHEVSKY, A. **Learning Multiple Layers of Features from Tiny Images**. [S.l.], 2009.
- LANG, N. **Using Convolutional Neural Network for Image Classification**. 2021. Disponível em: <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4>.
- LENDAVE, V. What is a convolutional layer? 2021. Disponível em: <https://analyticsindiamag.com/what-is-a-convolutional-layer/>.
- NIELSEN, M. A. **Neural Networks and Deep Learning**. Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com>.
- NIELSEN, M. A. Neural networks and deep learning. Determination Press, 2015. Disponível em: <https://cs231n.github.io/neural-networks-1/>.
- PAULY, L. et al. Deeper networks for pavement crack detection. In: . [S.l.: s.n.], 2017.

ROSA, G.; LUZ, J. da. Mix grinding simulation by artificial neural network. **Rem: Revista Escola de Minas**, v. 65, p. 247–256, 06 2012. Disponível em: [http://old.scielo.br/scielo.php?script=sci\\_arttext&pid=S0370-44672012000200014&lng=en&nrm=iso&tlng=pt](http://old.scielo.br/scielo.php?script=sci_arttext&pid=S0370-44672012000200014&lng=en&nrm=iso&tlng=pt).

SAXENA, D.; CAO, J. Generative adversarial networks (gans): Challenges, solutions, and future directions. **ACM Comput. Surv.**, Association for Computing Machinery, New York, NY, USA, v. 54, n. 3, may 2021. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3446374>.

SCHMIDT-HIEBER, J. Nonparametric regression using deep neural networks with ReLU activation function. **The Annals of Statistics**, Institute of Mathematical Statistics, v. 48, n. 4, p. 1875 – 1897, 2020. Disponível em: <https://doi.org/10.1214/19-AOS1875>.

SIMONYAN, K.; ZISSERMAN, A. **Very Deep Convolutional Networks for Large-Scale Image Recognition**. arXiv, 2014. Disponível em: <https://arxiv.org/abs/1409.1556>.

SRINIVASAN, A. V. **Stochastic Gradient Descent — Clearly Explained**. 2019. Disponível em: <https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31>.

Stanford university. **CS231n Convolutional Neural Networks for Visual Recognition**. 2022. Disponível em: <https://github.com/cs231n/cs231n.github.io>.

VOVK, V. The fundamental nature of the log loss function. In: \_\_\_\_\_. **Fields of Logic and Computation II: Essays Dedicated to Yuri Gurevich on the Occasion of His 75th Birthday**. Cham: Springer International Publishing, 2015. p. 307–318. ISBN 978-3-319-23534-9. Disponível em: [https://doi.org/10.1007/978-3-319-23534-9\\_20](https://doi.org/10.1007/978-3-319-23534-9_20).

YALÇIN, O. G. **Image Classification in 10 Minutes with MNIST Dataset**. 2018. Disponível em: <https://towardsdatascience.com/image-classification-in-10-minutes-with-mnist-dataset-54c35b77a38d>.

ZHOU, P. et al. **Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning**. arXiv, 2020. Disponível em: <https://arxiv.org/abs/2010.05627>.

## APÊNDICES

**APÊNDICE A – HIERARQUIA DE CLASSES CIFAR 100**

## a) Animais Marinhos:

- Castor
- Golfinho
- Lontra
- Foca
- Baleia

## b) Peixes:

- Peixes de aquário
- Peixe chato
- Raia
- Tubarão
- Truta

## c) Flores:

- Orquídea
- Papoula
- Rosas
- Girassol
- Tulipa

## d) Contêiner de Comida:

- Garrafa
- Tigela
- Lata
- Copo
- Prato

## e) Frutas e vegetais:

- Maça
- Cogumelo
- Laranja
- Pera
- Pimentão

## f) Aparelhos eletrodomésticos:

- Relógio
  - Teclado
  - Lâmpada
  - Telefone
  - Televisão
- g) Móveis:
- Cama
  - Cadeira
  - Sofá
  - Mesa
  - Guarda roupas
- h) Insetos:
- Abelha
  - Besouro
  - Borboleta
  - Lagarta
  - Barata
- i) Carnívoros de grande porte:
- Urso
  - Leopardo
  - Leão
  - Tigre
  - Lobo
- j) Construções artificiais:
- Ponte
  - Castelo
  - Casa
  - Estrada
  - Arranha-céus
- k) Cenários naturais:
- Nuvem
  - Floresta
  - Montanha
  - Planície



- Mar
- l) Onívoros e herbívoros de grande porte:
  - Camelo
  - Gado
  - Chimpanzé
  - Elefante
  - Canguru
- m) Mamíferos de médio porte:
  - Raposa
  - Porco-espinho
  - Didelídeos(Gambá)
  - Guaxinim
  - Gambá
- n) Invertebrados não-insetos:
  - Carangueijo
  - Lagosta
  - Lesma
  - Aranha
  - Minhoca
- o) Pessoas:
  - Bebê
  - Menino
  - Menina
  - Homem
  - Mulher
- p) Répteis:
  - Crocodilo
  - Dinossauro
  - Lagarto
  - Cobra
  - Tartaruga
- q) Mamíferos de pequeno porte:
  - Hamster

- Rato
  - Coelho
  - Soricidae
  - Esquilo
- r) Árvores:
- Bordo
  - Carvalho
  - Palmeira
  - Pinheiro
  - Salgueiro
- s) Veículos 1:
- Bicicleta
  - Ônibus
  - Motocicleta
  - Caminhonete
  - Trem
- t) Veículos 2:
- Cortador de grama
  - Foguete
  - Bonde
  - Tanque
  - Trator