APRENDIZADO PROFUNDO PARA CLASSIFICAÇÃO DE MICROORGANISMOS

Renan Carlos Prata de Faria

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Ricardo Cordeiro de Farias

Rio de Janeiro

Setembro de 2018

APRENDIZADO PROFUNDO PARA CLASSIFICAÇÃO DE MICROORGANISMOS

Renan Carlos Prata de Faria

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Ricardo Cordeiro de Farias, PhD

Prof. Felipe Maia Galvão França, PhD

Prof. Paulo Sérgio Salomon, PhD

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2018

Faria, Renan Carlos Prata de

Aprendizado Profundo para Classificação de Microorganismos / Renan Carlos Prata de Faria. – Rio de Janeiro: UFRJ / COPPE, 2018.

XIII, 54 p.: il.; 29, 7cm.

Orientador: Ricardo Cordeiro de Farias

Dissertação (mestrado) – UFRJ / COPPE / Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 51 – 54.

1. Machine Learning. 2. Deep Learning. 3. Microrganismos. 4. Redes Neurais. I. Farias, Ricardo Cordeiro de. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Aos meus pais, Celso e Alzira, que me educaram a enveredar o mundo que está pelo caminho que é. À minha avó, Caetana, que com sua luta me fez chegar onde estou.

A todos os meus amigos que me apoiaram nesse caminho.

Agradecimentos

Gostaria de agradecer primeiramente ao Professor Ricardo Farias, do Departamento de Engenharia de Sistemas e Computação da UFRJ, PESC, pela paciência com que soube conduzir esse trabalho, além de conselhos certeiros para um fim produtivo. Agradeço também ao Professor Paulo Salomon pelas imagens e dados dos fitoplanctons usados nesse trabalho. Por fim, agradeço ao PESC pela oportunidade de fazer o mestrado em computação gráfica.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APRENDIZADO PROFUNDO PARA CLASSIFICAÇÃO DE MICROORGANISMOS

Renan Carlos Prata de Faria

Setembro/2018

Orientador: Ricardo Cordeiro de Farias

Programa: Engenharia de Sistemas e Computação

Neste trabalho apresentamos uma comparação entre três métodos de aprendizado de máquina (ML) aplicados para identificar *Chatonella* localizadas imagens microscópicas. Analisamos o algoritmo de vizinhos de k mais próximos (KNN), o algoritmo de rede neural sem peso (WNN) e o algoritmo de rede neural convolucional (CNN). Este último sendo o estado da arte para classificação de imagens. O objetivo deste trabalho é identificar o melhor método para contabilizar diferentes tipos de microrganismos de amostras de água oceânica, com a finalidade de detectar poluição. Essa comparação leva em consideração a precisão da taxa de acertos. O algoritmo que teve o melhor desempeho foi a rede neural convolucional, como será detalhado neste trabalho.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DEEP LEARNING FOR MICROORGANISMS CLASSIFICATION

Renan Carlos Prata de Faria

September/2018

Advisor: Ricardo Cordeiro de Farias

Department: Systems Engineering and Computer Science

In this work we present a comparison among three machine learning (ML) methods applied to identify *Chattonella* based on microscopic images. We analyze K-Nearest Neighbors algorithm (KNN), Weightless Neural Network algorithm (WNN) and Convolutional Neural Network algorithm (CNN). The latter being the state of art to image classification. The goal of this work is to identify the best method to count different types of microorganisms in oceanic water samples, with the goal for detecting pollution. This comparison takes into account accuracy of the hit rate. The best result was reached by Convolutional Neural Network algorithm, as will be described in this work.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Inteligência Artificial (IA) é atualmente um assunto que é tendência na ciência da computação. Embora pertença ao campo da computação e engenharia, a IA tem sido utilizado de forma multidisciplinar, como geologia, física, biologia, medicina, entre outras áreas. O verdadeiro desafio para a inteligência artificial provou ser a solução de tarefas fáceis para o ser humano, mas difíceis de descrever formalmente para uma máquina. Problemas que podem ser resolvidos intuitivamente, e que parecem automáticos, como reconhecer palavras faladas ou rostos em imagens [4], são complexos de serem implementados para que uma máquina os resolva. Devido a isso, existe um enorme campo de pesquisa que estuda algoritmos e formas de solucionar e tornar mais simples esses tipos de problemas.

Definição

Inteligência artificial, às vezes chamada de inteligência de máquina, é inteligência demonstrada por máquinas, em contraste com a inteligência natural exibida por humanos e outros animais. Na ciência da computação, a pesquisa em IA é definida como o estudo de "agentes inteligentes": qualquer dispositivo que perceba seu ambiente e realize ações que maximizem sua chance de atingir seus objetivos com sucesso. Coloquialmente, o termo "inteligência artificial"é aplicado quando uma máquina imita funções "cognitivas"que os humanos associam a outras mentes humanas, tais como "aprendizagem"e "resolução de problemas".

Um problema que é fácil para os humanos e difícil para as máquinas é identificar com precisão microrganismos *Chattonella* (mostrado pela figura 1.1) presentes em

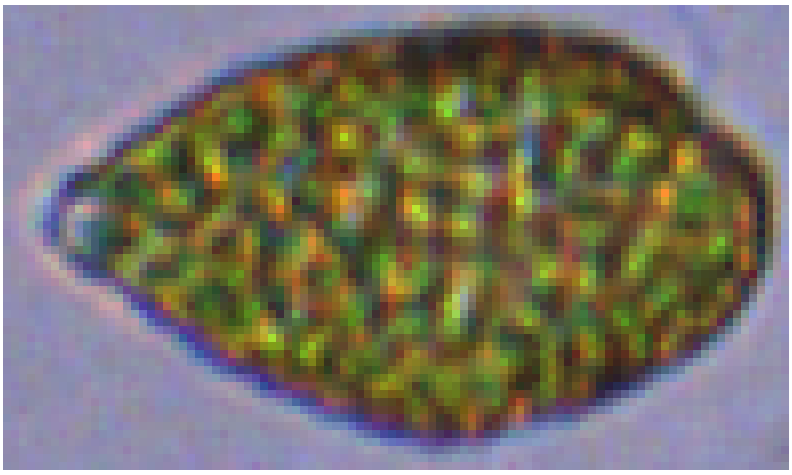

Figura 1.1: Exemplo de uma *Chattonella*[1].

uma dada amostra de água, ou seja, em meio à uma amostra complexa com células de diversas morfologias. A *Chattonella* é um gênero de rapidófitas marinhas associadas às marés vermelha [1]. As Chattonellas são encontradas em várias regiões balneáreas do mundo, no entanto, neste trabalho apresentamos os resultados de *Chattonella* coletados na baía de Guanabara, localizada no Rio de Janeiro. Sua ploriferção ocorre em função de poluição e variação climática do ecossistema. Os efeitos dessa proliferação é a contaminação do ecossistem e a redução da troca de oxigênio entre a atmosfera e o mar, causando assim morte dos peixes que vivem naquele meio[5]. A proliferação causa o efeito conhecido como "maré vermelha", e o impacto não é apenas ambiental, também é econômicos em nossa sociedade. Por isso é de extrema importância identificar de maneira rápida e precisa a proliferação da *Chattonella*.

1.1 Motivação

Em uma amostra de água é possível ter milhares de microorganismos. Embora a classificação visual por humanos seja simples, isso é muito trabalhoso e o tempo

para quantificar e catalogar os microrganismos de uma única amostra é bastante demorado, podendo demorar dias, e há também a possibilidade de erro humano durante a realização desta tarefa. O uso de máquinas para este tipo de classificação mostra-se ideal, pois levaria alguns milissegundos para categorizar os microrganismos de uma amostra.

Através de imagens com microrganismos microscópicos, foi-se desenvolvido um banco de dados de imagem rotoulado (dataset) e também aplicado algoritmos de inteligência artificial para a identificação automática da *Chattonella*. O motivo de usar três métodos distintos é a possibilidade de buscar qual mais eficiênte para que se tenha o melhor resultado para a classifcação da *Chattonella*.

1.2 Objetivo

Nosso objetivo é criar um método mais eficiente para a identificação automática de *Chattonella sp* em amostras de água para auxiliar o trabalho de pesquisadores da área de fitoplâncton marinho. Para isso será desenvolvido um algoritmo implementado uma parte em linguagem de programação C++ e outra parte em Python para a segmentação e classificação das imagens desses microorganismos. Também será desenvolvido um Dataset com imagens do microrganismo alvo que queremos classificar automaticamente, que no caso é a Chattonella.

1.3 Trabalhos Relacionados

Há uma variedade de métodos para classificar imagens de microorganismos [6] [7]. O sistema visual humano reconhece e localiza eficientemente objetos em cenas desordenadas. Para sistemas artificiais, no entanto, isso ainda é difícil. No campo da biologia, a classificação de microrganismos é geralmente realizada através da contagem manual por humanos. Diversos trabalhos têm seus dados coletados dessa forma, o que implica um maior gasto de tempo para a contagem de cada microrganismo a ser analisado. Os primeiros trabalhos desenvolvidos para catalogar dados estatísticos de *Chattonella* usaram este método manual[1].

A classificação de imagens digitais é o processo de extrair informações em imagens para reconhecer padrões e objetos homogêneos, e cada pixel da imagem é um

rótulo que descreve um objeto real. Atualmente, o método de classificação digital de imagens é amplamente utilizado, pois acelera o processo de classificação e reduz a chance de erro humano, proporcionando maior quantidade de dados, além de maior precisão com os dados adquiridos. Esses dados são compostos por até dois tipos de etapas: o pré-processamento da imagem e a aplicação de um algoritmo estatístico para extrair as características destes.

O pré-processamento da imagem é uma maneira de adaptar a imagem para a aplicação do algoritmo estatístico. No caso, seria uma aplicação de um filtro e/ou redimensionamento na quantidade de pixels desta imagem. A aplicação do algoritmo estatístico é o estágio no qual escolhemos um algoritmo capaz de extrair características e atribuí-lo a um determinado grupo ou padrão. Neste caso, algoritmos de aprendizado de máquina serão utilizados, devido à sua eficiência nos resultados da classificação.

Muitas implementações estão disponíveis para classificação de imagens. Neste trabalho, avaliamos três métodos diferentes para classificar um microorganismo específico, a *Chattonella*.

1.4 Metodologia

Com o intuito de facilitar e acelerar o processo de classificação de microrganismos, revisamos alguns métodos basedos em algoritmos de inteligência artitficial. A Implementação destes foi realizada utilizando as linguangens de programação Python e C++ a fim de comparar suas performances e analisar as características de cada um, tendo como principal objetivo descobrir qual a melhor abordagem para resolver o problema proposto.

Procuramos analisar os possíveis métodos de classificação automática utilizando algoritmos de inteligência artificial. Como dito antes, investigamos três algoritmos diferentes: Algoritmo de vizinhos mais próximos (KNN), algoritmo de rede neural sem peso (WNN) e algoritmo de rede neural convolucional (CNN). Além disso, este trabalho busca comparar a eficiência do uso desses algoritmos com o método manual de classificação, realizado por um ser humano.

Antes de implementar os algoritmos de inteligência artificial foi necessário de-

senvolver um ambiente para pré-processamento. Em outras palavras, foi necessário a construção de um banco de dados de imagens (dataset) de Chattonella, realizada com o auxílio dos alunos e professores do Laboratório de Fitoplâncton Marinho da Universidade Federal do Rio de Janeiro. Estas imagens foram retiradas do dispositivo FlowCam, que está instalado no Laboratório de Fitoplâncton Marinho, sem que as imagens dos microrganismos não tivesse passado por nenhum processamento antes. Este trabalho foi desenvolvido em duas partes: A segmentação da imagem que era composta de diversos microorganismos diferentes, utilizando um algoritmo de segmentação conveniente ao caso, e depois a classificação manual destas imagens. O Dataset é composto por 4000 imagens de microrganismos, de amostras mista complexa.

Uma parte do que foi pretendido a ser desenvolvido teve de ser ponderado/reduzido. Por exemplo, era de interesse o desenvolvimento de uma aplicação web interativa que facilita-se o uso do melhor algoritmo para outros pesquisadores. Também era de interesse tornar esse banco de dados aberto e livre para uso de outras instituições. Este trabalho identifica de maneira precisa Chattonella presentes em uma amostra mista de microrganismos, e tem como perspectiva futura também classificar outros microrganismos de interesse. Por fim, também era pretendido fazer a mesma abordagem utilizando o algoritmo de redes neural convolucional para realizar a segmentação do microorganismo alvo, pois isso reduziria um passo desta aplicação, que consiste em segmentar a imagem antes de realizar a classificação. Todos esses pontos são ponderados para trabalhos futuros que poderão ser desenvolvidos.

Em suma, a metodologia consiste no estudo comparativo e implementação de algoritmos de Inteligencia Artificial para a contagem do número de Chattonellas em uma determinada amostra. Como resultado, tem-se uma ferramenta poderosa para a resolução deste problema.

1.5 Organização da Dissertação

Dos tópicos relevantes sobre o Inteligencia Artificial aplicado para classificação de imagens, escolhemos o subconjunto que deve configurar uma apresentação o mais possível flúida e concisa.

No segundo capítulo, *Revisão Bibliográfica* apresenta a explicação do funcionamento de cada algoritmo de inteligência artificial que foi utilizado neste trabalho, com os conceitos, formulações e heurísticas conforme a literatura acadêmica apresenta. Mencionamos também as características da linguagem de programação C++ e Python, além dos frameworks utilizados para a implementação destes algortimos. Ao final do capítulo apresentamos uma breve explanação de algoritmos de segmentação e de Data Augmentation que são de uso importante para o trabalho proposto.

No terceiro capítulo, *Metodo Proposto* é apresentado a forma que o Dataset foi construído e como foi implementado cada algorítmo de Inteligência Artificial.

No quarto capítulo, *Resultados e Discussões*, apresentamos os dados que fundamentam as conclusões apresentadas adiante.

No quinto capítulo, *Conclusões* apresentamos, além das conclusões, as ressaltas sobre a validade do que se pôde concluir a partir dos testes e também comentamos sobre o que se pretendia fazer, o que foi feito e trabalhos futuros.

No apêndice, *Apêndice A* estão as implementações de cada algorítimo desenvolvido.

Capítulo 2

Revisão Bibliográfica

> Intelligence is the ability to adapt to change.
>
> Stephen Hawking [8]

Foi revisada na literatura de Inteligência Artificial, o algoritmo de k-vizinhos mais próximos, o algoritmo de rede neural sem peso e o algoritmo de rede neural convolucional, tal como sua aplicação para a classificação de imagens. Além disso, foi também detalhado característas das linguagens de programação C++ e Python e métodos de segmentação de imagens.

2.1 Métodos de Classificação de Imagem

Como mencionado anteriormente, este trabalho consiste na implementação e comparação de três métodos de classificação. Esta seção tem como objetivo explicar cada método investigado.

2.1.1 Algoritmo de k-vizinhos mais próximos

No reconhecimento de padrões, o algoritmo de k-vizinhos mais próximos (KNN) é um método não paramétrico usado para classificação e regressão [9]. Em ambos os casos, a entrada consiste nos k exemplos de treinamento mais próximos no espaço de recursos. Não é muito usado na prática, mas nos permitirá ter uma idéia sobre a abordagem básica para o problema de classificação de imagem e é rápido e simples

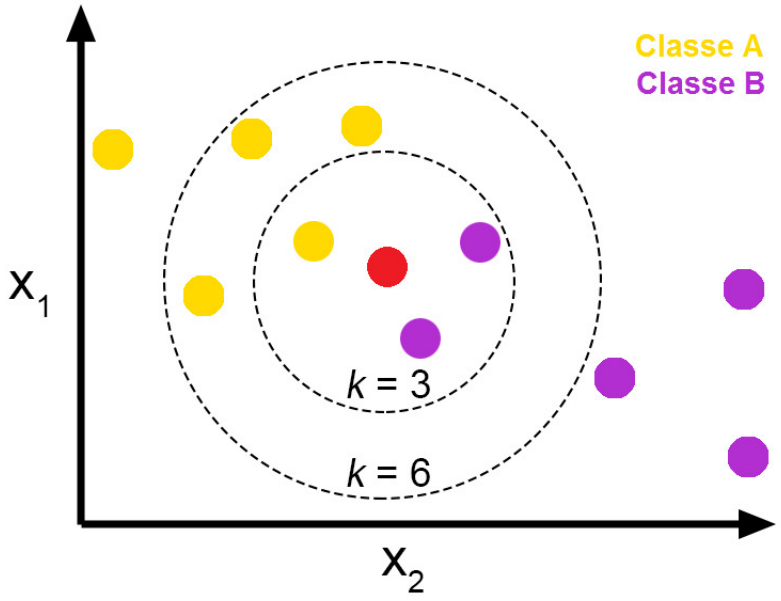

Figura 2.1: Apresentação gráfica do algoritmo KNN para diferentes valores de K

de implementar.

Existem vetores em um espaço de recurso multidimensional denominado exemplo de treinamento. Cada vetor tem um rótulo de classe. A fase de treinamento deste algoritmo consiste apenas em armazenar os vetores e rótulos de classes das amostras de treinamento. Na fase de classificação, k é uma constante definida pelo usuário e um vetor não rotulado é classificado atribuindo-se o rótulo mais frequente entre as k amostras de treinamento mais próximas a esse ponto de consulta.

Os exemplos de treinamento são vetores em um espaço de recurso multidimensional, cada um com um rótulo de classe. A fase de treinamento do algoritmo consiste apenas em armazenar os vetores de recursos e rótulos de classes das amostras de treinamento.

Na fase de classificação, k é uma constante definida pelo usuário e um vetor não rotulado, uma consulta ou ponto de teste, é classificado atribuindo-se o rótulo mais frequente entre as k amostras de treinamento mais próximas a esse ponto de consulta, veja figura 2.1.

Uma métrica de distância comumente usada para variáveis contínuas é a distância euclidiana. Para variáveis discretas, como para classificação de texto, outra métrica pode ser usada, como a métrica de sobreposição, ou distância de Hamming [10]. No contexto de dados de microarranjos de expressão gênica, por exemplo, k-NN também foi empregado com coeficientes de correlação, como Pearson e Spearman.

Freqüentemente, a precisão de classificação de KNN pode ser melhorada significativamente se a métrica de distância for aprendida com algoritmos especializados, como a análise de componentes vizinho maior ou vizinhança de margem maior.

Uma desvantagem da classificação básica de "votação por maioria"ocorre quando a distribuição de classes é distorcida, ou seja, exemplos de uma classe mais frequente tendem a dominar a previsão do novo exemplo, porque tendem a ser comuns entre os k vizinhos mais próximos devido ao seu grande número [11]. Uma maneira de superar esse problema é ponderar a classificação, levando em conta a distância do ponto de teste até cada um de seus vizinhos mais próximos. A classe, ou valor em problemas de regressão, de cada um dos k pontos mais próximos é multiplicada por um peso proporcional ao inverso da distância entre esse ponto e o ponto de teste. Outra maneira de superar a distorção é por abstração na representação de dados, usando por exemplo, em um mapa de auto-organização.

Podemos calcular a distância entre duas imagens usando a equação 2.1.

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \tag{2.1}$$

Onde a soma é tomada em todos os pixels. Ou usando a equação de distância euclidiana 2.2.

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \tag{2.2}$$

Considerando usar a distância de Hamming, utiliza-se a equação 2.3.

$$HD_{raw} = \frac{\|(codeA \otimes codeB) \cap maskA \cap maskB\|}{\|maskA \cap maskB\|} \tag{2.3}$$

O algoritmo de vizinhos k mais próximos é amplamente utilizado para classificação simples e é um método rápido de ser implementado e testado.

2.1.2 Redes Neurais sem Peso

Redes Neurais sem Peso, anteriormente conhecidas como "N-tuplas", foi originalmente concebida por Bledsoe e Browning [12] em 1959. Passada duas décadas, os estudos relacionados a Redes Neurais sem Peso, Weightless Neural Network (WNN),

retornaram através de Igor Aleksander, Wilkes e Stonham, com o desenvolviemento da WiSARD [3], uma rede neural amplamente estudada até hoje.

Apesar de serem inspiradas em neorônios do sistema nervoso humano, essas redes seguem uma abordagem um pouco diferente das redes com peso. Nelas, a emulação da topologia das conexões entre dendritos e axônios, ou seja, a árvore dendrítica [13], é priorizada. Além disso, seus neurônios artificiais não realizam processamento ou têm conexões com pesos sinápticos, de modo a aguardar informações em suas conexões, por isso são conhecidas como redes sem peso.

A informação aprendida é armazenada em memórias RAM, que funcionam como neurônios artificiais, ou seja, redes neurais baseadas em memórias RAM. Além disso, assim como a arquitetura dos neurônios nas redes de pesos poderia ser modificada, diferentes maneiras de usar as memórias RAM foram desenvolvidas. Tais formulários impactam não apenas o tempo de resposta, como também o grau de generalização e especificidade. As principais redes sem peso disponíveis atualmente são: WiSARD, G-RAM (Generalization RAM), PLN (Probabilistic Logic Node) e GSN (Goal Seeking Neuron) [14].

Discriminadores e WiSARD

Um discriminador de RAM consiste em um conjunto de RAMs de uma palavra X de um bit com n entradas e um dispositivo de soma (Σ). Qualquer tal discriminador de RAM pode receber um padrão binário de bits $X \bullet n$ como entrada. As linhas de entrada da RAM são conectadas ao padrão de entrada por meio de um mapeamento pseudo-aleatório biunívoco. O dispositivo de soma permite que esta rede de RAMs exiba - assim como outros modelos de RNA baseados em pesos sinápticos - generalização e tolerância a ruído.

Neste trabalho, implementamos o modelo WiSARD. WiSARD foi a primeira rede neural artificial a ser patenteada e produzida comercialmente, e também o modelo de rede neural sem peso mais representativo [13]. Originalmente era tudo implementado em hardware, por Bruce Wilkie, mas depois seu algoritmo foi reproduzido na forma de um programa C++ em execução no UNIX. Assim, um laptop pode ser transformado em uma rede neural com considerável poder cognitivo e flexibilidade, o que levou a um nível de prototipagem rápida do sistema antes inconcebível quando

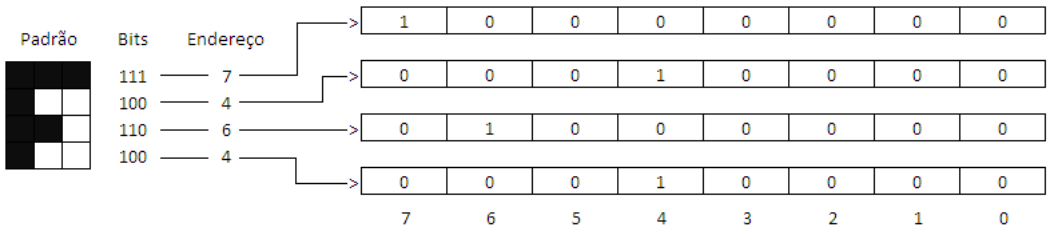

Figura 2.2: Discriminador da categoria "F"de WiSARD com memórias de 8 bits e entrada de 12 bits.

se usa apenas hardware [3].

Nesta rede, a entrada é separada em tuplas. Cada uma dessas tuplas será relacionada a uma RAM e, como mencionado anteriormente, o conhecimento será armazenado nessas memórias. Além disso, cada tupla de entrada será usada para endereçar um local de memória, e assim acessar o conhecimento armazenado nessa RAM. Como as tuplas correspondem a um endereço de RAM, a entrada deve ter valores binários ou ter sofrido algum processamento que converta o valor original em bits [3].

O treinamento de um neurônio é feito alterando o conteúdo da localização da memória endereçada pela tupla, que começa com "0"e "1". Dessa forma, cada neurônio é extremamente eficiente em reconhecer um padrão apresentado anteriormente, mas não é capaz de generalizar as informações para reconhecer padrões semelhantes. No entanto, como mencionado, cada tupla representa apenas parte da entrada, de modo que um padrão apresentado será dividido em M partes. Dentro da rede, cada categoria é armazenada em um discriminador, que é composto de M RAM. São esses conjuntos de memórias os responsáveis pela generalização da rede e tolerância a ruído [2].

É importante notar que uma tupla com N bits endereça 2^N posições. Portanto, o número de bits na tupla dependerá do tamanho da memória que está sendo usada na rede. Considerando que a entrada tem MxN bits, serão necessárias M RAM para cada discriminador usado.

Na figura 2.2, podemos observar uma rede que usa memórias de 8 bits (2^3) e, portanto, precisa de quatro memórias por discriminador para trabalhar com uma entrada de 12 bits. Neste exemplo, um padrão exibido para a categoria "F"está sendo armazenado.

Nessa rede, o reconhecimento é feito de maneira muito semelhante ao treinamento. No entanto, a entrada é exibida para todos os discriminadores, cada RAM lê o endereço de memória apontado pela tupla correspondente e retorna o valor armazenado, que pode ser 0 ou 1. Uma vez feito isso, cada discriminador adiciona o número de memórias retornadas 1 e chega ao seu total de pontos. Em seguida, a categoria do discriminador que obteve a pontuação mais alta é apresentada em resposta a esse reconhecimento [3].

Embora esta seja uma forma muito eficiente e bem sucedida, este método tem um aspecto negativo muito relevante: no caso de empate, uma das categorias sorteadas será escolhida de forma aleatória. A figura 2.3 exemplifica o processo de reconhecimento ocorrido dentro de um discriminador com N RAM, que neste caso obteve r pontos. Pode-se observar que neste exemplo, as posições de entrada que compõem as tuplas de cada RAM foram agrupadas de forma completamente aleatória.

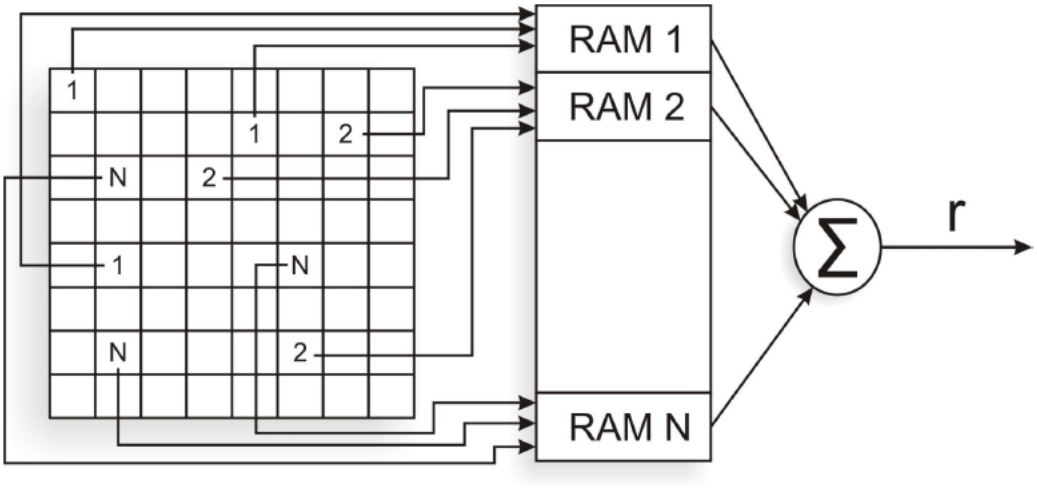

Figura 2.3: Cálculo de pontos de um discriminador durante o processo de reconhecimento.(França, Silva, Lengerke, et al., 2009) [2]

Além do cálculo do ponto de discriminação, este sistema fornece um nível de confiança relativo ao resultado de reconhecimento alcançado. Este parâmetro é calculado pela fórmula 2.4.

$$C = (R_{max} + R_{2max})/R_{max} \tag{2.4}$$

onde R_{max} é a maior pontuação encontrada para os discriminadores e R_{2max} a segunda maior pontuação. Em caso de empate, como os dois discriminadores

obtiveram o mesmo escore, a confiança será igual a zero, o que pode ser usado como um alerta para possíveis categorizações errôneas. A figura 2.4 mostra o resultado do reconhecimento em um WiSARD com múltiplos discriminadores. Neste exemplo R_1 obteve a maior pontuação e R_N o segundo maior.

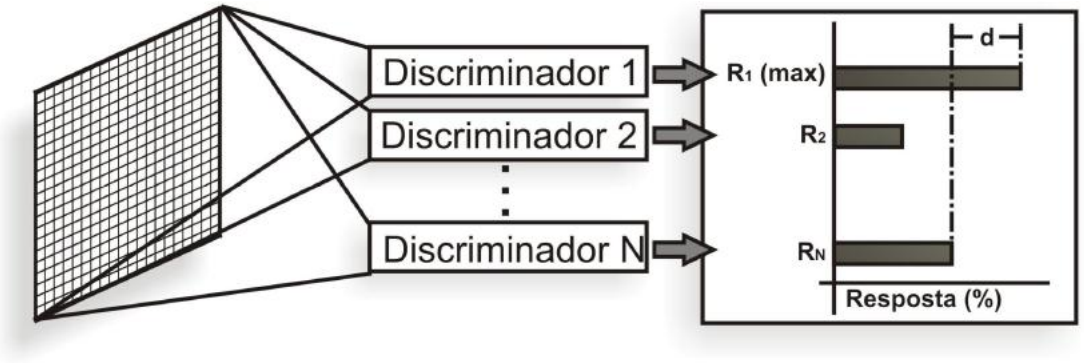

Figura 2.4: Reconhecimento com N discriminadores.[3]

É fácil ver que tanto o treinamento quanto o reconhecimento não dependem do número de padrões aprendidos pela rede. Eles podem ser executados em tempo constante, porque dependem apenas do tamanho da entrada e, no caso do reconhecimento, do número de categorias. Essa é a grande vantagem dessa rede porque permite que as duas etapas sejam executadas quase instantaneamente. Além disso, o grau de generalização da rede pode ser facilmente controlado, dependendo da relação entre o tamanho da memória e o número de memórias do discriminador. Quanto maior a memória, menor a capacidade de generalização da rede [2]

PLNs, MPLNs e *p*RAMs

A maioria dos sistemas neurais baseados em RAM é formada por uma única camada de nós RAM. Aleksander introduziu uma arquitetura multicamadas, pirâmide, formada por novos nós chamados PLN, Probabilistic Logic Node. Um nó PLN armazena um valor de 2 bits em cada um dos seus locais de memória: “0”, “1” e “u”. Este último representa o estado “desconhecido” ou “indefinido”, que é armazenado em todos os locais de memória PLN antes da fase de aprendizado. O estado indefinido “u” permite que um nó PLN imprima aleatoriamente 0 ou 1, com a mesma probabilidade

A arquitetura da pirâmide formada por nós PLN é caracterizada por ter um fan-

out de 1 e baixo fan-in. Com uma baixa fan-in, o nó PLN satura com muita facilidade e não consegue aprender novos padrões antes que o conjunto de treinamento tenha sido completamente apresentado.

O algoritmo de treinamento consiste em substituir u com 0 e 1 da seguinte maneira: um padrão de treinamento é propagado através da base da pirâmide até o topo, onde a saída é produzida; se o valor de saída da pirâmide corresponder à saída desejada, um sinal será propagado de volta para cada nó para que eles armazenem todos os valores de saída gerados aleatoriamente no local endereçado no momento em cada nó. Se a saída produzida não for o valor desejado, o mesmo padrão de entrada será aplicado novamente e o mesmo procedimento será repetido.

Aleksander [2] propôs uma extensão natural do nó PLN: o PLN do estado m (MPLN). Nesse novo modelo, um intervalo discreto mais amplo de valores poderia ser armazenado em cada local de memória. Além disso, a função de saída pode ser até mesmo uma função linear ou a função sigmóide. Nesse modelo, a fase de aprendizado permite mudanças incrementais dos valores armazenados. Novas informações são obtidas após diferentes etapas, pois as informações incorretas são descartadas somente após um certo número de erros.

Para incorporar algumas propriedades dos neurônios vivos, uma evolução desse modelo foi proposta por Taylor [15]. Pode ser demonstrado que este novo modelo é equivalente a uma rede de RAMs (probabilísticas) ruidosas, pRAMs. Neste modelo, os valores pertencentes a [0, 1] podem ser armazenados nos locais da memória, probabilidade contínua. Dada uma certa entrada, o conteúdo de localizações de memória representa a probabilidade de que um valor 1 seja produzido como saída.

GSN e GRAMs

O GSN (Goal Seeking Neuron) foi desenvolvido com o objetivo de preservar a corrupção das informações previamente armazenadas nos nós PLN. Ao contrário de um PLN, um GSN pode inserir, armazenar e gerar resultados de 0, 1 e u. Dois locais diferentes são abordados em um GSN se a entrada contiver um u. Da mesma forma, se um determinado conteúdo de memória for u, o GSN correspondente abordará dois nós diferentes.

O GSN possui três modos diferentes de operações: validação, aprendizado e

recuperação. O propósito da fase de validação é verificar se um novo padrão pode ser aprendido sem corromper o conhecimento armazenado anteriormente. O valor na saída significa que a rede pode aprender qualquer saída desejada. 0 ou 1 como saída sugere que a rede pode aprender apenas o padrão proposto se a saída da rede for a desejada. Caso contrário, haveria uma interrupção das informações armazenadas anteriormente. Em caso de sucesso da fase de validação, uma das seguintes condições ocorrerá (fase de aprendizado): se a saída desejada for 0 (1) e o conteúdo endereçado contiver pelo menos um 0 (1), então um desses locais é selecionado aleatoriamente e seu endereço de entrada torna-se, efetivamente, as saídas desejadas para nós na camada anterior; caso contrário, um local indefinido é selecionado aleatoriamente, do conjunto endereçável, e o desejado na saída é armazenada nesse local e, como acima, o endereço desse local volta para a camada anterior. O objetivo da fase de *recall* é o de produzir o valor mais ocorrente (0 ou 1) nos conteúdos abordados. Se houver números iguais de 0 e 1 no conteúdo endereçado, a saída será u.

Os GRAMs foram introduzidos por Aleksander [2] para aumentar a generalização de WNNs no nível do nó, incluindo uma fase de espalhamento no algoritmo de aprendizado, logo após o armazenamento de um padrão de treinamento. O GRAM se comporta como uma memória de acesso aleatório organizada por bits até o ponto em que há um conflito entre a tentativa de definir um endereço para 0 e 1, ou um endereço não pode ser definido de acordo com a regra de Distância de Hamming. Em tais casos, o endereço permanece indefinido e se comporta como um gerador aleatório de cadeias binárias. Esta forma de generalização é descrita como um processo de disseminação, onde as informações são colocadas na RAM através do conjunto treinado se espalha para outros locais. Em algumas aplicações, o espalhamento limitado a um raio é empregado, no sentido de que distâncias de Hamming maiores que um limite definido são ignoradas.

2.1.3 Redes Neurais Convolucionais

No contexto da inteligência artificial e aprendizado de máquina, uma rede neural convolucional, CNN, é uma classe de rede neural de alimentação artificial, que tem sido aplicada com sucesso no processamento e análise de imagens digitais. As Redes Neurais Convolucionais são muito semelhantes às Redes Neurais comuns: elas são

constituídas de neurônios que possuem pesos e vieses que podem ser aprendidos. Cada neurônio recebe algumas entradas, executa um produto escalar e, opcionalmente, segue-o com uma não linearidade. Toda a rede ainda expressa uma única função de pontuação diferenciável: dos pixels da imagem bruta em uma extremidade às pontuações da classe na outra. E eles ainda têm uma função de perda, por exemplo SVM/Softmax, na última camada, totalmente conectada, e todas as características de Redes Neurais regulares ainda se aplicam [16].

Uma Rede Neural Convolucional usa uma variedade de perceptrons multicamadas 2.5 desenvolvidos para exigir o mínimo possível de pré-processamento. Essas redes também são conhecidas como redes neurais artificiais invariantes a mudanças ou invariantes no espaço, em ambos os casos representados pelo acrônimo SIANN

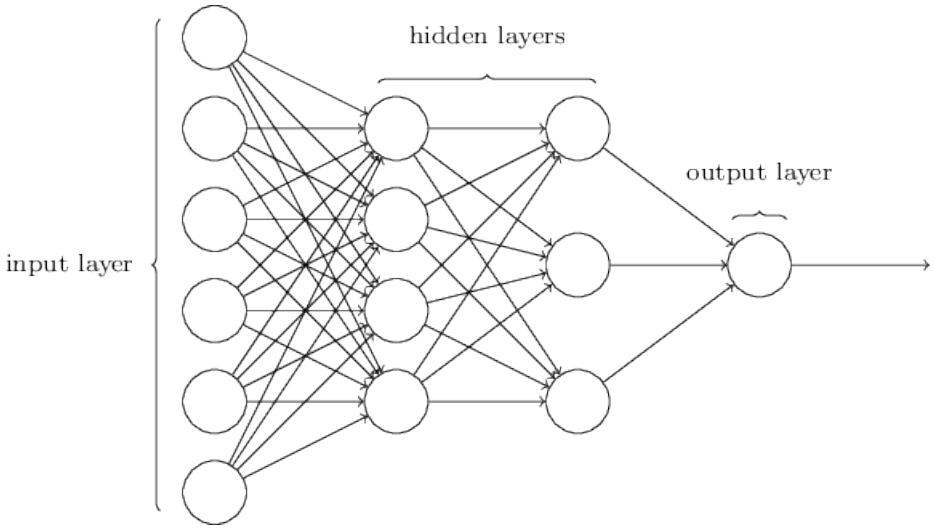

Figura 2.5: Um exemplo de arquitetura Perceptron multicamadas.

Redes Neural Convolucional aproveitam o fato de que a entrada consiste em imagens e elas restringem a arquitetura de uma maneira mais sensata. Em particular, ao contrário de uma Rede Neural regular, as camadas de uma ConvNet possuem neurônios dispostos em 3 dimensões: largura, altura e profundidade. Observe que a profundidade da palavra aqui se refere à terceira dimensão de um volume de ativação, não à profundidade de uma Rede Neural completa, que pode se referir ao número total de camadas em uma rede. Por exemplo, as imagens de entrada em CIFAR-10 [17], são um volume de entrada de ativações e o volume tem dimensões $32x32x3$, largura, altura e profundidade, respectivamente. Como veremos em breve, os neurônios em uma camada só serão conectados a uma pequena região da camada

antes dela, em vez de todos os neurônios de uma maneira totalmente conectada. Além disso, a camada de saída final para CIFAR-10 [17] teria dimensões $1x1x10$, porque até o final da arquitetura ConvNet reduziremos a imagem completa em um único vetor de pontuações de classe, organizadas ao longo da dimensão de profundidade.

As redes convolucionais são inspiradas nos processos biológicos [18]. Neles, o padrão de conectividade entre os neurônios é inspirado na organização do córtex visual dos animais. Neurônios corticais individuais respondem a estímulos apenas em regiões restritas do campo de visão conhecidas como campos receptivos. Os campos receptivos de diferentes neurônios se sobrepõem parcialmente para cobrir todo o campo de visão.

Uma rede neural convolucional tende a exigir um nível mínimo de pré-processamento quando comparado a outros algoritmos de classificação de imagens [19]. Isso significa que a rede "aprende"os filtros que, em um algoritmo tradicional, precisariam ser implementados manualmente. Essa independência de um conhecimento a priori e do esforço humano no desenvolvimento de sua funcionalidade básica pode ser considerada a maior vantagem de sua aplicação.

Este tipo de rede é usado principalmente em reconhecimento de imagem e processamento de vídeo 2.6, embora já tenha sido aplicado com sucesso em experimentos envolvendo processamento de voz e linguagem natural.

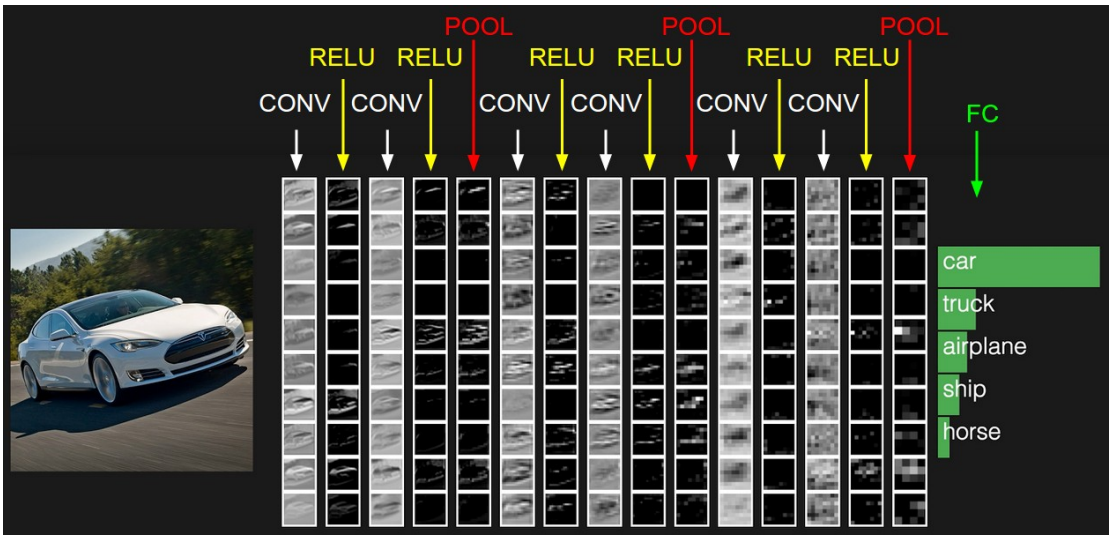

Figura 2.6: Um exemplo de arquitetura ConvNet usada para classificar objetos. O volume inicial armazena os pixels da imagem bruta, a esquerda, e o último volume armazena as pontuações da classe, a direita)

Existem várias arquiteturas no campo de redes neurais convolucionais que possuem nomes específicos. Os mais comuns são:

- LeNet - As primeiras aplicações bem sucedidas de CNNs foram desenvolvidas por Yann LeCun. Destes, o mais conhecido é a arquitetura LeNet que foi usada para ler códigos postais, dígitos e classificar objetos [20].

- AlexNet - O primeiro trabalho que popularizou as CNNs na Computer Vision, desenvolvido por Alex Krizhevsky [21]. O AlexNet foi submetido ao desafio ImageNet ILSVRC em 2012 e superou significativamente o segundo colocado (top 5 error of 16% comparado ao vice-campeão com 26% de erro). A rede tinha uma arquitetura muito semelhante à LeNet, mas era mais profunda, maior e apresentava camadas convolucionais empilhadas umas sobre as outras.

- ZF Net - Foi uma melhoria na AlexNet, aprimorando os hiperparâmetros de arquitetura, em particular expandindo o tamanho das camadas convolucionais intermediárias e tornando o tamanho da passada e do filtro na primeira camada menor [22].

- GoogLeNet - Sua principal contribuição foi o desenvolvimento de um Módulo de Iniciação que reduziu drasticamente o número de parâmetros na rede 4M,

comparado com o AlexNet com 60M [23].

- VGGNet - Sua principal contribuição foi mostrar que a profundidade da rede é um componente crítico para um bom desempenho. Sua melhor rede final contém 16 camadas CONV/FC e, de forma atraente, apresenta uma arquitetura extremamente homogênea que executa apenas convoluções 3x3 e agrupamento 2x2 do começo ao fim. Uma desvantagem do VGGNet é que é mais caro avaliar e usar muito mais memória e parâmetros, 140M [24]. Essa arquitetura foi usada neste trabalho.

- ResNet - Possui conexões de salto especiais e um uso intenso de normalização de lotes. A arquitetura também está faltando camadas totalmente conectadas no final da rede [25].

Camadas de uma ConvNet

Uma arquitetura de uma rede neural convolucional é formada por uma pilha de camadas distintas que transformam o volume de entrada em um volume de saída, por exemplo, mantendo as pontuações da classe, através de uma função diferenciável. Alguns tipos distintos de camadas são comumente usados.

Camada Convolucional

A camada convolucional é o núcleo do bloco de construção de uma rede neural convolucional. Os parâmetros da camada consistem em um conjunto de filtros, ou núcleos, que podem ser aprendidos, que possuem um pequeno campo receptivo, mas se estendem por toda a profundidade do volume de entrada. Durante o passo para frente, cada filtro é convolvido através da largura e altura do volume de entrada, computando o produto escalar entre as entradas do filtro e a entrada e produzindo um mapa de ativação bidimensional desse filtro. Como resultado, a rede aprende filtros que são ativados quando detecta algum tipo específico de recurso em alguma posição espacial na entrada.

O empilhamento dos mapas de ativação para todos os filtros ao longo da dimensão de profundidade forma o volume total de saída da camada de convolução. Cada entrada no volume de saída também pode ser interpretada como uma saída de um neurônio que olha para uma pequena região na entrada e compartilha parâmetros

com neurônios no mesmo mapa de ativação.

Camada de Agrupamento

Outro conceito importante de redes neural convolucional é o pooling, ou agrupamento, que é uma forma de down-sampling não linear. Existem várias funções não lineares para implementar o agrupamento entre as quais o agrupamento máximo é o mais comum. Ele particiona a imagem de entrada em um conjunto de sub-imagens retângulares não sobrepostas. Para cada sub-imagem, verifica o pixel de maior valor. A intuição é que a localização exata de um recurso é menos importante do que sua localização aproximada em relação a outros recursos. A camada de pooling serve para reduzir progressivamente o tamanho espacial da representação, reduzir o número de parâmetros e a quantidade de computação na rede e, portanto, também controlar o overfitting. É comum inserir periodicamente uma camada de agrupamento entre camadas convolucionais sucessivas em uma arquitetura CNN. A operação de pooling fornece outra forma de invariância de conversão.

A camada de pooling opera independentemente em cada fatia de profundidade da entrada e a redimensiona espacialmente. A forma mais comum é uma camada de agrupamento com filtros de tamanho $2x2$ aplicados com um passo de 2 amostras em cada fatia de profundidade na entrada por 2 ao longo de largura e altura, descartando 75% das ativações. Neste caso, cada operação máxima é superior a 4 números. A dimensão da profundidade permanece inalterada.

Além do *pool* máximo, as unidades de *pool* podem usar outras funções, como *pool* médio ou *pool* de norma L2. O *pool* médio foi frequentemente usado historicamente, mas recentemente caiu em desuso, dando lugar ao *pool* máximo, que funciona melhor na prática.

Devido à redução significativa no tamanho da representação, a tendência é usar filtros menores ou descartar completamente a camada de agrupamento.

Camada ReLU

ReLU é a abreviação de unidades lineares retificadas. Esta camada aplica a função de ativação sem saturação $f(x) = max(0, x)$. Ito aumenta as propriedades não-lineares da função de decisão e da rede global sem afetar os campos receptivos da camada de convolução.

Outras funções também são usadas para aumentar a não-linearidade, por exem-

plo, a tangente hiperbólica de saturação $f(x) = \tanh x$ e $f(x) = \tanh x$, e a funçã sigmóide $f(x) = \frac{1}{1+e^{-x}}$. O ReLU é frequentemente preferido à outras funções, porque ele treina a rede neural bem mais rápido, sem uma penalidade significativa para a precisão da generalização.

Camada Totalmente Conectada

Finalmente, após várias camadas convolutionais e max *pooling*, o raciocínio de alto nível na rede neural é feito através de uma camada totalmente conectada. Neurônios em uma camada totalmente conectada têm conexões com todas as ativações das camadas anterior, como visto em redes neurais regulares. Suas ativações podem, portanto, ser calculadas com uma multiplicação de matrizes seguida por um deslocamento de polarização.

Camada de Perda

A camada de perda especifica como o treinamento penaliza o desvio entre os rótulos previstos e verdadeiros e normalmente é a camada final. Várias funções de perda apropriadas para diferentes tarefas podem ser usadas. A perda do Softmax é usada para prever uma única classe de K classes mutuamente exclusivas. A perda de entropia cruzada sigmóide é usada para prever valores de probabilidade independentes de K em $[0, 1]$. A perda euclidiana é usada para regredir para rótulos com valor real $(\infty, -\infty)$.

Escolha de Hiperparâmetros

Uma rede neural convolucional possi mais hiperparâmetros do que uma rede neural padrão. Embora as regras usuais para as taxas de aprendizado e as constantes de regularização ainda se apliquem, convém ter em mente o seguinte ao otimizar.

Número de Filtros

Como o tamanho do mapa de recurso diminui com a profundidade, as camadas próximas à camada de entrada tendem a ter menos filtros, enquanto as camadas superiores podem ter mais. Para equalizar a computação em cada camada, o recurso **x** posição do pixel é mantido constante entre as camadas. A preservação de mais informações sobre a entrada exigiria manter o número total de ativações, número de mapas de recursos multiplicando o número de posições de pixels, não diminuindo de uma camada para a próxima.

O número de mapas de recursos controla diretamente a capacidade e depende do número de exemplos disponíveis e da complexidade da tarefa.

Formas de Filtro

Formas de filtros comuns, encontradas na literatura, variam muito e são geralmente escolhidas com base no conjunto de dados.

O desafio é encontrar o nível certo de granularidade, de modo a criar abstrações na escala adequada, dado um determinado conjunto de dados.

Max Pooling Shape

Max pooling, ou o agrupamento máximo, é um processo de discretização baseado em amostra. O objetivo é reduzir a amostragem de uma representação de entrada, imagem, matriz de saída de camada oculta, etc, reduzindo sua dimensionalidade e permitindo suposições sobre os recursos contidos nas sub-regiões categorizadas.

Valores típicos são $2x2$. Volumes de entrada muito grandes podem justificar o agrupamento de $4x4$ nas camadas inferiores. No entanto, a escolha de formas maiores reduzirá drasticamente a dimensão do sinal e poderá resultar em perda excessiva de informações. Geralmente, as janelas de *pool* não sobrepostas têm melhor desempenho.

Overfitting e Métodos de Regularização

Nas estatísticas, o overfitting é a produção de uma análise que corresponde muito de perto ou exatamente a um conjunto particular de dados e, portanto, pode falhar em ajustar dados adicionais ou prever observações futuras de forma confiável. Um modelo super adaptado é um modelo estatístico que contém mais parâmetros que podem ser justificados pelos dados. A essência do overfitting é ter extraído inconscientemente alguma da variação residual, ou seja o ruído, como se essa variação representasse a estrutura do modelo subjacente.

Em machine learning, temos a seguinte situação: O treinamento acusou uma taxa de 99% de acerto, entretanto, usando o dataset de testes, teve uma taxa de 50%. Essa situação é configurada como um overfiting. Agora, será abordado a forma de contornar este problema. Chama-se métodos de regularização.

A regularização é um processo de introdução de informações adicionais para impedir o overfitting. Redes neural convolucional usam vários tipos de regularização.

Abaixo será apresentado alguns métodos:

Dropout

Como uma camada totalmente conectada ocupa a maioria dos parâmetros, é propensa a overfitting. Um método para reduzir o overfitting é o dropout. Em cada estágio de treinamento, nós individuais são "descartados"da rede com probabilidade $1-p$ ou mantida com probabilidade p, de modo que uma rede reduzida é deixada; as bordas de entrada e saída para um nó descartado também são removidas. Somente a rede reduzida é treinada nos dados nesse estágio. Os nós removidos são então reinseridos na rede com seus pesos originais.

Nos estágios de treinamento, a probabilidade de que um nó oculto seja descartado é geralmente $0,5$; para nós de entrada, isso deve ser muito menor, intuitivamente, porque as informações são perdidas diretamente quando os nós de entrada são ignorados.

No tempo de teste após o término do treinamento, nós idealmente gostaríamos de encontrar uma média de amostra de todas as redes removíveis 2^n; infelizmente isto é inviável para grandes valores de n. No entanto, podemos encontrar uma aproximação usando a rede completa com a saída de cada nó ponderada por um fator de p, então o valor esperado da saída de qualquer nó é o mesmo que nos estágios de treinamento. Esta é a maior contribuição do método de dropout: embora ele efetivamente gere redes neurais 2^n, e como tal permite a combinação de modelos, no tempo de teste apenas uma única rede precisa ser testado.

Evitando treinar todos os nós em todos os dados de treinamento, o dropout diminui o overfitting. O método também melhora significativamente a velocidade de treinamento. Isso torna a combinação de modelos prática, mesmo para redes neurais profundas. A técnica parece reduzir as interações dos nós, levando-os a aprender recursos mais robustos que melhor se generalizam para novos dados.

DropConnect

DropConnect é a generalização do dropout em que cada conexão, ao invés de cada unidade de saída, pode ser descartada com probabilidade $1-p$. Cada unidade recebe entrada de um subconjunto aleatório de unidades na camada anterior.

O DropConnect é semelhante ao dropout, pois introduz a dispersão dinâmica dentro do modelo, mas difere na medida em que a dispersão está nos pesos, e não

nos vetores de saída de uma camada. Em outras palavras, a camada totalmente conectada com o DropConnect se torna uma camada esparsamente conectada na qual as conexões são escolhidas aleatoriamente durante o estágio de treinamento.

Pooling Estocástico

Uma grande desvantagem do Dropout é que ele não tem os mesmos benefícios para camadas convolucionais, onde os neurônios não estão totalmente conectados.

Em pooling estocástico, as operações de agrupamento determinístico convencional são substituídas por um procedimento estocástico, onde a ativação dentro de cada região de agrupamento é escolhida aleatoriamente de acordo com uma distribuição multinomial, dada pelas atividades dentro da região de agrupamento. A abordagem é livre de hiperparâmetros e pode ser combinada com outras abordagens de regularização, como o abandono e o aumento de dados.

Uma visão alternativa do pool estocástico é que ele é equivalente ao pool máximo padrão, mas com muitas cópias de uma imagem de entrada, cada uma com pequenas deformações locais. Isso é semelhante às deformações elásticas explícitas das imagens de entrada, que fornecem excelente desempenho do MNIST. Usar o conjunto estocástico em um modelo multicamadas fornece um número exponencial de deformações, uma vez que as seleções em camadas superiores são independentes daquelas abaixo.

Dado Artificial

Também conhecido como data augmentation, aumento de dados. Uma vez que o grau de overfitting do modelo é determinado tanto pelo seu poder quanto pela quantidade de treinamento que recebe, fornecer uma rede convolucional com mais exemplos de treinamento pode reduzir o overfitting. Como essas redes são geralmente treinadas com todos os dados disponíveis, uma abordagem é gerar novos dados do zero (se possível) ou perturbar os dados existentes para criar novos dados. Por exemplo, as imagens de entrada podem ser cortadas assimetricamente em alguns por cento para criar novos exemplos com o mesmo rótulo do original.

Parada Antecipada

Um dos métodos mais simples para evitar o overfitting de uma rede é simplesmente interromper o treinamento antes que o overfitting tenha ocorrido. Vem com a desvantagem de que o processo de aprendizagem é interrompido.

Número de Parâmetros

Outra maneira simples de evitar o overfitting é limitar o número de parâmetros, normalmente limitando o número de unidades ocultas em cada camada ou limitando a profundidade da rede. Para redes convolucionais, o tamanho do filtro também afeta o número de parâmetros. Limitar o número de parâmetros restringe o poder preditivo da rede diretamente, reduzindo a complexidade da função que pode executar nos dados e, assim, limita a quantidade de overfitting.

Decaimento de Peso

Uma forma simples de regularizador adicionado é o decaimento de peso, que simplesmente adiciona um erro adicional, proporcional à soma dos pesos, norma L1, ou magnitude quadrada, norma L2, do vetor de peso, ao erro em cada nó. O nível de complexidade do modelo aceitável pode ser reduzido aumentando a constante de proporcionalidade, aumentando assim a penalidade para vetores de grande peso.

A regularização L2 é a forma mais comum de regularização. Ele pode ser implementado penalizando a magnitude quadrada de todos os parâmetros diretamente no objetivo. A regularização L2 tem a interpretação intuitiva de penalizar fortemente vetores de peso de pico e preferindo vetores de peso difuso. Devido às interações multiplicativas entre pesos e insumos, isso tem a propriedade útil de encorajar a rede a usar todos os seus insumos um pouco, em vez de alguns de seus insumos, muito.

A regularização L1 é outra forma comum. É possível combinar L1 com regularização L2 (isso é chamado de regularização da rede elástica). A regularização L1 leva os vetores de peso a se tornarem escassos durante a otimização. Em outras palavras, os neurônios com regularização L1 acabam usando apenas um subconjunto esparso de suas entradas mais importantes e se tornam quase invariantes aos insumos ruidosos.

Principais Dataset

Existem alguns dataset de livre acesso para auxiliar o desenvolvimento do seu algortimo de machine learning. Entre eles citamos o **CIFAR-10**, o **CIFAR-100** e o **MNIST**.

O conjunto de dados CIFAR-10 consiste em 60.000 imagens coloridas de $32x32$

em 10 classes, com 6.000 imagens por classe. Existem 50.000 imagens de treinamento e 10.000 imagens de teste.

O conjunto de dados é dividido em cinco lotes de treinamento e um lote de teste, cada um com 10.000 imagens. O lote de teste contém exatamente 1.000 imagens selecionadas aleatoriamente de cada classe. Os lotes de treinamento contêm as imagens restantes em ordem aleatória, mas alguns lotes de treinamento podem conter mais imagens de uma classe do que outra. Entre eles, os lotes de treinamento contêm exatamente 5.000 imagens de cada classe. A figura 2.7 ilustra esse banco de dados com algumas imagens aleatórias.

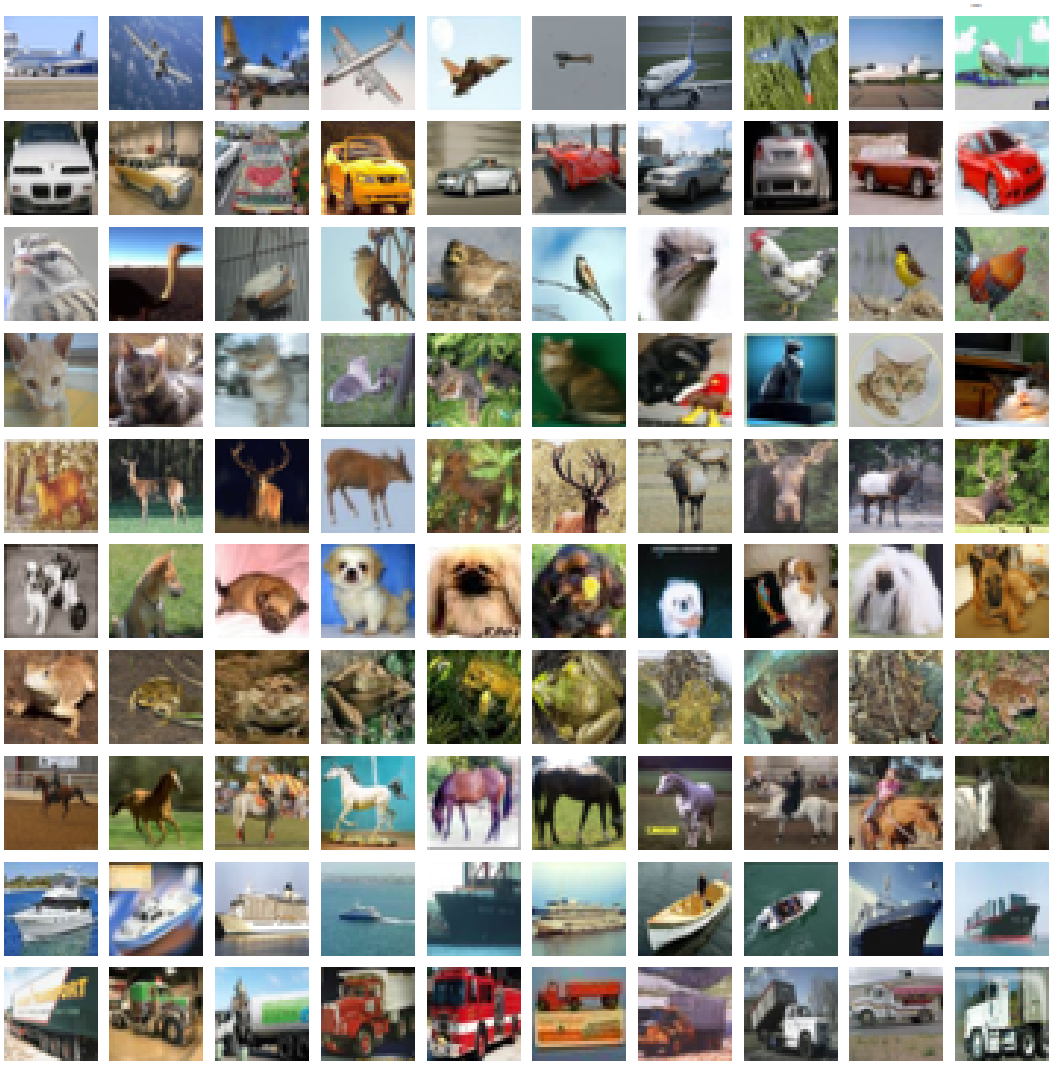

Figura 2.7: Uma amostra das imagens encontradas no dataset CIFAR-10

O conjunto CIFAR-100 de dados é parecido com o CIFAR-10, exceto que tem 100 classes contendo 600 imagens cada. Existem 500 imagens de treinamento e 100

imagens de teste por turma. As 100 classes no CIFAR-100 são agrupadas em 20 superclasses. Cada imagem vem com uma etiqueta "fina", a classe à qual pertence, e uma etiqueta "grossa", a superclasse à qual ela pertence.

O banco de dados MNIST de dígitos manuscritos 2.8, disponível nesta página, possui um conjunto de treinamento de 60.000 exemplos e um conjunto de teste de 10.000 imagens. É um subconjunto de um conjunto maior disponível no NIST. Os dígitos foram normalizados por tamanho e centralizados em uma imagem de tamanho fixo. É um bom banco de dados para pessoas que querem experimentar técnicas de aprendizado e métodos de reconhecimento de padrões em dados do mundo real, enquanto gastam esforços mínimos em pré-processamento e formatação.

Figura 2.8: Uma amostra das imagens encontradas no dataset MNIST

Existem outros Datasets, mas a sua maioria são proprietários e tem um custo para uso. Em Inteligência Artificial, muitas vezes o Dataset é mais importante que o próprio algortimo ou as estratégias para reduzir erros.

2.2 Linguagem de programação C++ e Python

C++ é uma das linguagens mais populares utilizadas principalmente com software de sistema/aplicativo, drivers, aplicativos cliente-servidor e firmware embarcado.

O principal destaque do C++ é uma coleção de classes predefinidas, que são tipos de dados que podem ser instanciados várias vezes. A linguagem também facilita a declaração de classes definidas pelo usuário. As classes podem acomodar ainda mais funções-membro para implementar funcionalidades específicas. Vários objetos de uma determinada classe podem ser definidos para implementar as funções dentro da classe. Objetos podem ser definidos como instâncias criadas em tempo de execução. Essas classes também podem ser herdadas por outras novas classes que aceitam as funcionalidades públicas e protegidas por padrão.

C++ inclui vários operadores, como comparação, aritmética, manipulação de bits e operadores lógicos. Um dos recursos mais atraentes do C++ é que ele permite a sobrecarga de determinados operadores, como a adição.

Alguns dos conceitos essenciais dentro da linguagem de programação C ++ incluem polimorfismo, funções virtuais e de amigos, modelos, namespaces e ponteiros.

O principal motivo de utilizar C++ nesse trabalho é sua eficiência e o suporte à alguns frameworks de machine learning.

Python é uma linguagem de programação interpretada de alto nível para programação de propósito geral. Criado por Guido van Rossum e lançado pela primeira vez em 1991, o Python tem uma filosofia de design que enfatiza a legibilidade do código, notavelmente usando espaço em branco significativo. Ele fornece construções que permitem programação clara em escalas pequenas e grandes. Em julho de 2018, o criador Van Rossum deixou o cargo de líder na comunidade linguística após 30 anos.

O Python possui um sistema de tipo dinâmico e gerenciamento automático de memória. Suporta múltiplos paradigmas de programação, incluindo orientados a objetos, imperativos, funcionais e processuais, e possui uma biblioteca padrão ampla e abrangente.

Os intérpretes de Python estão disponíveis para muitos sistemas operacionais. O CPython, a implementação de referência do Python, é um software de código aberto e tem um modelo de desenvolvimento baseado na comunidade, assim como quase

todas as outras implementações do Python. Python e CPython são gerenciados pela Python Software Foundation sem fins lucrativos.

O uso de Python neste trabalho se deve a sua fácil prototipação e pelo forte apelo da comunidade de inteligência artificial em usar a ferramenta, com criação de algumas bibliotecas de machine learning.

2.3 Framewors de desenvolvimento

2.3.1 OpenCV

O OpenCV, Open Source Computer Vision Library, está sob uma licença BSD e, portanto, é gratuito para uso acadêmico e comercial. Possui interfaces C++, Python e Java e suporta Windows, Linux, Mac OS, iOS e Android. O OpenCV foi projetado para eficiência computacional e com um forte foco em aplicativos em tempo real. Escrito em C/C++ otimizado, a biblioteca pode aproveitar o processamento de vários núcleos. Habilitado com o OpenCL, ele pode aproveitar a aceleração de hardware da plataforma de computação heterogênea subjacente [26] .

Adotado em todo o mundo, o OpenCV tem mais de 47 mil pessoas da comunidade de usuários e um número estimado de downloads que ultrapassam 14 milhões. O uso varia de arte interativa a inspeção de minas, costura de mapas na web ou através de robótica avançada.

2.3.2 Keras

Keras é uma API de redes neurais de alto nível, escrita em Python e capaz de rodar em cima do TensorFlow, CNTK ou Theano. Foi desenvolvido com foco de permitir a experimentação rápida. Ser capaz de ir da ideia ao resultado no menor tempo possível, mostrando como um forte aliado para se fazer pesquisas.

Keras é uma biblioteca de aprendizado profunda que:

- Permite a criação de protótipos fácil e rápida por meio de facilidade de uso, modularidade e extensibilidade.
- Suporta redes convolucionais e redes recorrentes, bem como combinações das duas.

- Funciona perfeitamente na CPU e GPU.

Keras é uma API projetada para seres humanos, não para máquinas. Coloca a experiência do usuário na frente e no centro. A Keras segue as práticas recomendadas para reduzir a carga cognitiva: ela oferece APIs consistentes e simples, minimiza o número de ações do usuário necessárias para casos de uso comuns e fornece um feedback claro e acionável sobre o erro do usuário.

2.4 Segmentação de Imagem e Thresholding

Em visão computacional, a segmentação de imagens é o processo de particionar uma imagem digital em múltiplos segmentos, conjuntos de pixels, também conhecidos como super-pixels. O objetivo da segmentação é simplificar e/ou alterar a representação de uma imagem em algo que seja mais significativo e mais fácil de analisar. A segmentação de imagem é normalmente usada para localizar objetos e limites, como linhas, curvas, etc, em imagens. Mais precisamente, a segmentação de imagens é o processo de atribuir um rótulo a cada pixel de uma imagem, de modo que os pixels com o mesmo rótulo compartilhem determinadas características.

O resultado da segmentação de imagens é um conjunto de segmentos que cobrem coletivamente a imagem inteira ou um conjunto de contornos extraídos da imagem. Cada um dos pixels em uma região é semelhante em relação a alguma propriedade característica ou computada, como cor, intensidade ou textura. As regiões adjacentes são significativamente diferentes em relação às mesmas características. Quando aplicado a uma pilha de imagens, típica em imagens médicas, os contornos resultantes da segmentação da imagem podem ser usados para criar reconstruções em 3D com a ajuda de algoritmos de interpolação.

Thresholding é o método mais simples de segmentação de imagens. A partir de uma imagem em escala de cinza, o limite pode ser usado para criar imagens binárias. Os métodos de thresholding mais simples substituem cada pixel de uma imagem por um pixel preto se a intensidade da imagem $I_{i,j}$ é menor que alguma constante fixa T, ou seja $I_{i,j} < T$, ou um pixel branco se a intensidade da imagem for maior que essa constante. A figura 2.9 ilustra este método.

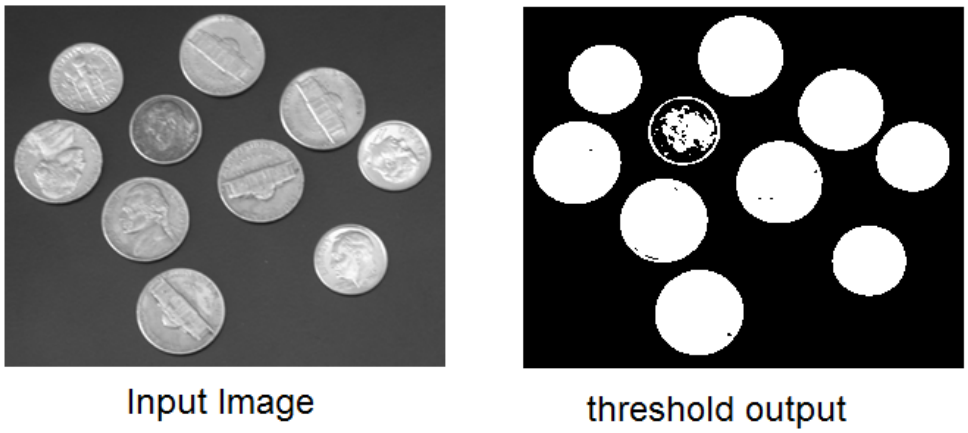

Figura 2.9: Exemplo de aplicação do algoritmo de thresholding.

Imagens coloridas também podem ser segmentadas. Uma abordagem é designar um limite separado para cada um dos componentes RGB da imagem e, em seguida, combiná-los com uma operação AND. Isso reflete a maneira como a câmera digital funciona e como os dados são armazenados no computador, mas não corresponde ao modo como as pessoas reconhecem as cores. Para representação de cores, como percebidas por humanos, outros modelos de cores são utilizados, como HSL e HSV.

Capítulo 3

Método Proposto

> In fact the (analytical) engine may be described as being the material expression of any indefinite function of any degree of generality and complexity...
>
> Ada Lovelace. Notes (1842)

Este capítulo apresenta o algoritmo de segmentação da imagem de uma amostra, a construção do dataset de imagens que é utilizado no trabalho e a implementação de cada método de classificação.

3.1 Construção do Banco de Dados de Imagens

O problema a ser resolvido neste trabalho é fazer uma fotografia microscópica, veja figura 3.1, para poder contar a incidência de um determinado microrganismo. Uma amostra pode gerar dezenas de imagens, e cada imagem contém centenas de imagens de microorganismos. Até agora esse trabalho de segmentação era feito manualmente o que demandava muito tempo para ser concluido. O desafio é tornar essa tarefa simples para o computador, de maneira robusta e precisa. Para identificar as imagens manualmente *Chattonella*, utilizou-se um humano bem treinado para fazer esse tipo de trabalho. No caso foi o aluno Rafael Menezes, aluno de mestrado do Laboratório de Fitoplâncton Marinho.

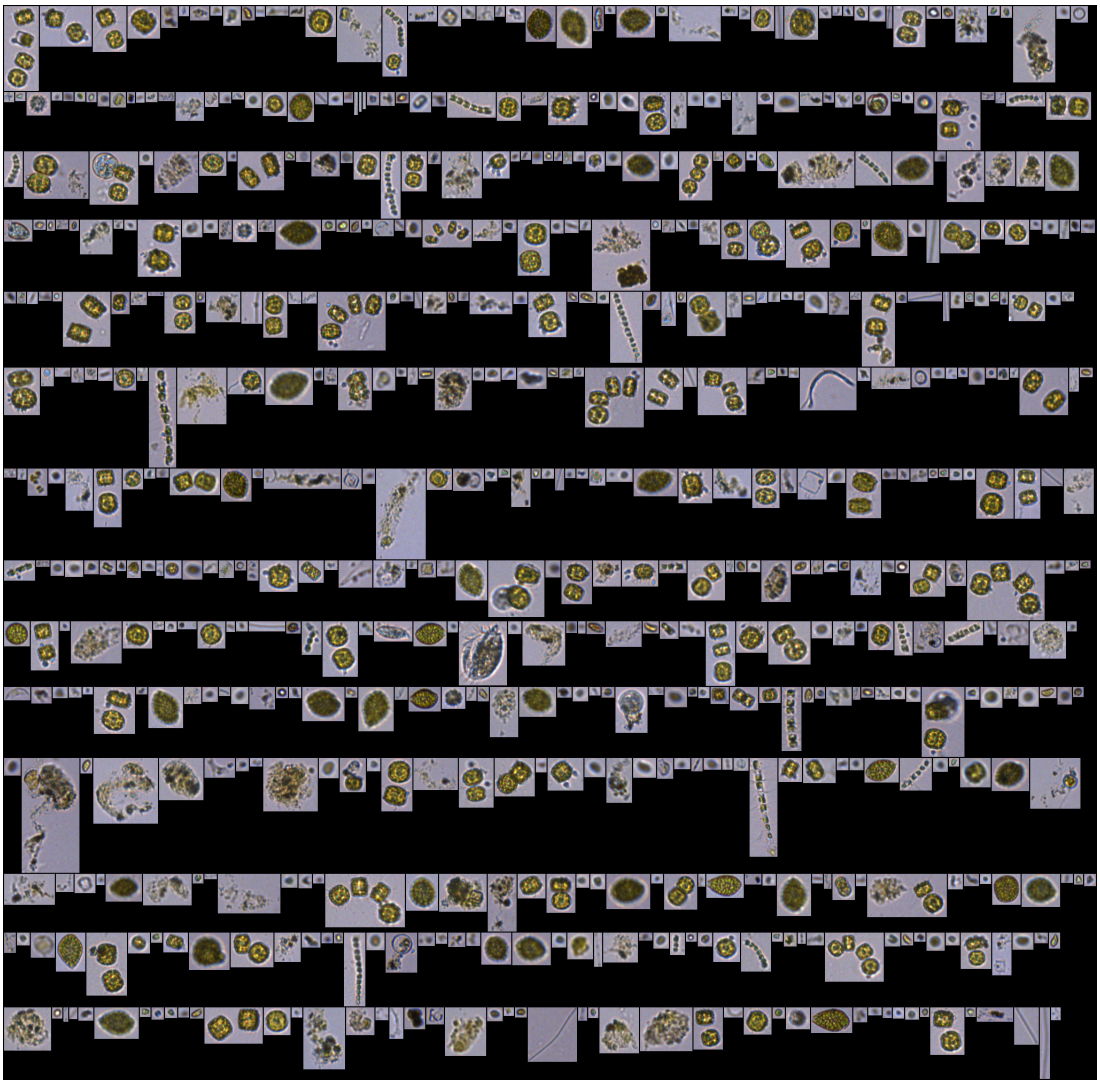

Figura 3.1: Uma fotografia microscópica.

A primeira parte do trabalho foi construir a Dataset. Para isso, um algoritmo de segmentação foi desenvolvido para separar as sub-imagens em arquivos independentes.

O método de segmentação usado foi *thresholding* [27]. Este método é muito simples e pode ser explicado pela equação 3.1.

$$g(x,y) = \begin{cases} 0, & \text{if} f(x,y) < T \\ 1, & \text{if} f(x,y) \geqslant T \end{cases} \tag{3.1}$$

O valor limite T do pixel usado foi 0x000000. Assim, o algoritmo desenvolvido atravessa a imagem em busca de um pixel com esse valor e, a partir daí, remove a região de interesse, neste caso, é a imagem de um único microorganismo. O

algoritmo é mostrado em 3.1, onde a operação deve ficar mais claramente.

Algorithm 3.1 Algoritmo adaptado para segmentação da imagem.

```
T ← 0x000000 {threshold value}
para x := 0 to width faça
  para y := 0 to height faça
    se pixel[x * height + y] = T então
      continue
    se-não
      imageW ← 0 {new image width}
      imageH ← 1 {new image height}
      repetir
        pushPixel(pixel[x * height + y + imageW])
        imageW ← imageW + 1
      até-que pixel[x * height + y + imageW] = T
      repetir
        k ← y
        para k := y to y + imageW faça
          pushPixel(pixel[(x + imageH) * height + k])
        fim para
        imageH ← imageH + 1
      até-que pixel[(x + imageH) * height + k] = T
    fim se
    resizeImage()
    saveImage()
  fim para
fim para=0
```

A função *pushPixel*() é usada para salvar os pixels de cada sub-imagem na nova imagem salva em arquivos independentes, referente a cada microorganismo presente na imagem microscópica. você deve criar todas as imagens com a mesma quantidade de pixels e também armazenar essa imagem (*resizeImage*()) no sistema de arquivos para uso posterior (*saveImage*()).

Após a segmentação e a geração de milhares de images de microrganismos, nós ainda precisamos aplicar um método de aumento de dados [28] [29], para expandir o tamanho do banco de dados. Isso se faz necessário para termos mais imagens para treinamento das RN. Para isso, aplicamos algoritmos aleatórios de rotação e translação nas imagens geradas. Com essas técnicas, o banco de dados tem um aumento consideravel, dobrando ou mesmo triplicando a quantidade de imagens. A figura 3.2 ilustra um exemplo de tal procedimento. O aumento de dados é uma maneira de combater o overfitting, porém não é suficiente, já que nossas amostras

aumentadas ainda são altamente correlacionadas. A capacidade entropica do modelo implementado é o que se apresenta como principal característica para combater o overfitting. Um modelo que pode armazenar muitas informações tem o potencial de ser mais preciso aproveitando mais recursos, mas também corre o risco de começar a armazenar recursos irrelevantes. Enquanto isso, um modelo que pode armazenar apenas alguns recursos terá que se concentrar nos recursos mais significativos encontrados nos dados, e esses são mais propensos a serem realmente relevantes e a generalizar melhor.

Existem diferentes maneiras de modular a capacidade entrópica de um modelo. A principal é a escolha do número de parâmetros em seu modelo, ou seja, o número de camadas e o tamanho de cada camada. Outra forma é o uso da regularização de peso, como a regularização $L1$ ou $L2$, que consiste em controlar os pesos do modelo para obter valores menores. *Dropout* [30] também ajuda a reduzir o overfitting, impedindo que uma camada veja duas vezes o mesmo padrão, agindo de maneira análoga ao aumento de dados. Pode se dizer que tanto o *dropout* quanto o aumento de dados tendem a romper correlações aleatórias que possam ocorrer nos seus dados.

Figura 3.2: Um exemplo de aumento de dados. Usando a mesma imagem com transformações lineares aplicadas: rotações, inversões, cortes, etc.

Com isso, foi criado o banco de dados para a aplicação de algoritmos de Inteligência Artificial. É importante notar que esta etapa do trabalho é muito importante, já que algoritmos de inteligência artificial precisa de dados confiáveis. Na próxima sessão será explicada a implementação de cada algoritmo, com as informações de ajuste de cada parâmetro utilizado.

3.2 Implementação dos Algoritmos de Inteligência Artificial

3.2.1 K-Vizinhos mais Próximos

Este algoritmo foi implementado com a linguagem de programação C++ e o uso de OpenCV para manipulação da imagem. No caso esta implementação é composta dos seguintes passos:

- Lê os conjuntos de dados de treinamento e teste;
- Separa parte das imagens do conjunto de treinamento para validação;
- Utiliza o conjunto de validação para escolher o melhor K, número de vizinhos a considerar;
- Calcula a taxa de acerto para diferentes valores de K.

Para ser aplicado em imagens, o algoritmo k-vizinhos mais próximos foi implementado da seguinte maneira:

1. Leitura do valor do peso do pixel lido, o peso referente ao histograma;
2. Observa qual grupo melhor o representa;
3. Conclui-se a qual classe o pixel pertence.

Os valores aplicados para K variou entre 3 e 9, o que será explicado na parte dos resultados. O algoritmo 3.2 mostra essa implementação em psedocódigo.

3.2.2 Rede Neural Sem Peso

Este algoritmo também foi implementado com a linguagem de programação C++ e o uso de OpenCV para manipulação da imagem. Antes de classificar as imagens, foi realizado um pré-processamento de binarização das imagens, visto que o algoritmo de Rede Neural Sem Peso é necessário utilizar imagens em preto e branco. Segue os passos realizados para implementação.

Algorithm 3.2 Algoritmo k-vizinhos mais próximos

para t = 1 to T **faça** $S_t \leftarrow S_{t-1}$

  para $s_q \in S_t$ **faça** $N_q \leftarrow$ k nearest neighbors

    of s_q using $D(s_q, s_i)$

    label(s_q)= $argmax \sum_{s_i \in N_q} D(s_q, s_i)$;

    se label(s_q)$\neq y_q$ **então**

      para $s_i \in N_i$ **faça** $y_i \neq y_q$

        $w_i^t \leftarrow w_i^t - \lambda / d(x_q, x_i)$;

        $w_i^t \leftarrow w_i^t + \lambda / d(x_q, x_i)$;

      fim para

    fim se

  fim para

  se label(s_q)= $y_q \forall_{s_q}$ **então**

    break

  fim se

fim para

- Aplicata-se um filtro de binarização, com um valor definido de threashould para as imagens de microorganismos;
- Redimensionamento das imagens;
- Escolhe aleatoriamente imagens de *Chattonella* e não *Chattonella* para ser o discriminador. Houve multiplos discriminadores;
- Escolhe aleatoriamente tuplas de entrada para que seja montada a rede, criando assim N RAM's;
- Para cada RAM, tem-se o resultado binário, que é utilizado para calcular o nível de confiança.

No caso a rede montada utilizou-se de 8 descriminadores diferentes, 4 *Chattonella* e 4 não *Chattonella*. O motivo desta escolha foi para se fazer um balanceamento mais consistente dos resultados.

3.2.3 Rede Neural Convolucional

Este algoritmo foi implementado com a linguagem Python, usando o framework keras. Este framework simplificou o trabalho, pois abstrai todos os algoritmos básicos para a confecção de uma Rede Neural Convolucional. No caso usou-se uma arquitetura já estabelecida, a VGG16. A figura 3.3 apresenta a configuração desta arquitetura.

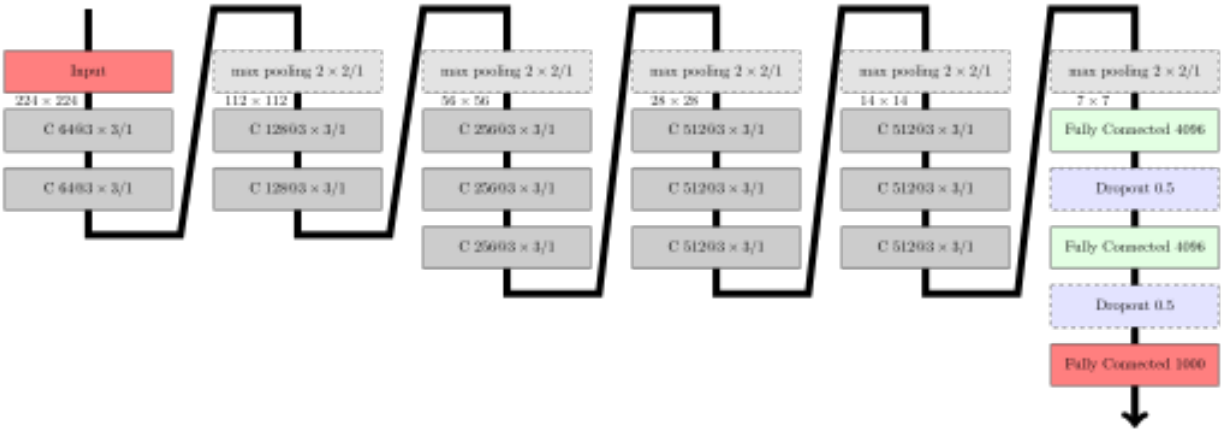

Figura 3.3: Arquitetura VGG16

Neste caso, o unico tratamento realizado nas imagens de entrada foi fazer um redimensionamento, todas devem ter o mesmo tamanho. O motivo de usarmos essa arquitetura foram:

- Ela é ótima para classificação de imagens;
- Consegue-se trabalhar com um banco de dados pequeno, até 1000 imagens;
- Consegue-se trabalhar com imagens de tamanho até $224 * 224 * 3$ pixels, RGB.

Implementamos esta rede com um total 16 camadas, sendo sua construção descrita abaixo:

- Convolução usando 64 filtros;
- Convolução usando 64 filtros + Max pooling;
- Convolução usando 128 filtros;
- Convolução usando 128 filtros + Max pooling;

- Convolução usando 256 filtros;
- Convolução usando 256 filtros;
- Convolução usando 256 filtros + Max pooling;
- Convolução usando 512 filtros;
- Convolução usando 512 filtros;
- Convolução usando 512 filtros + Max pooling;
- Convolução usando 512 filtros;
- Convolução usando 512 filtros;
- Convolução usando 512 filtros + Max pooling;
- Conexão completa com 4096 nós;
- Conexão completa com 4096 nós;
- Camada de saída com ativação Softmax usando 1000 nós.

A figura 3.4 mostra o que acontece com os pixels dessa imagem de uma imagem ao avançar pelas camadas da rede.

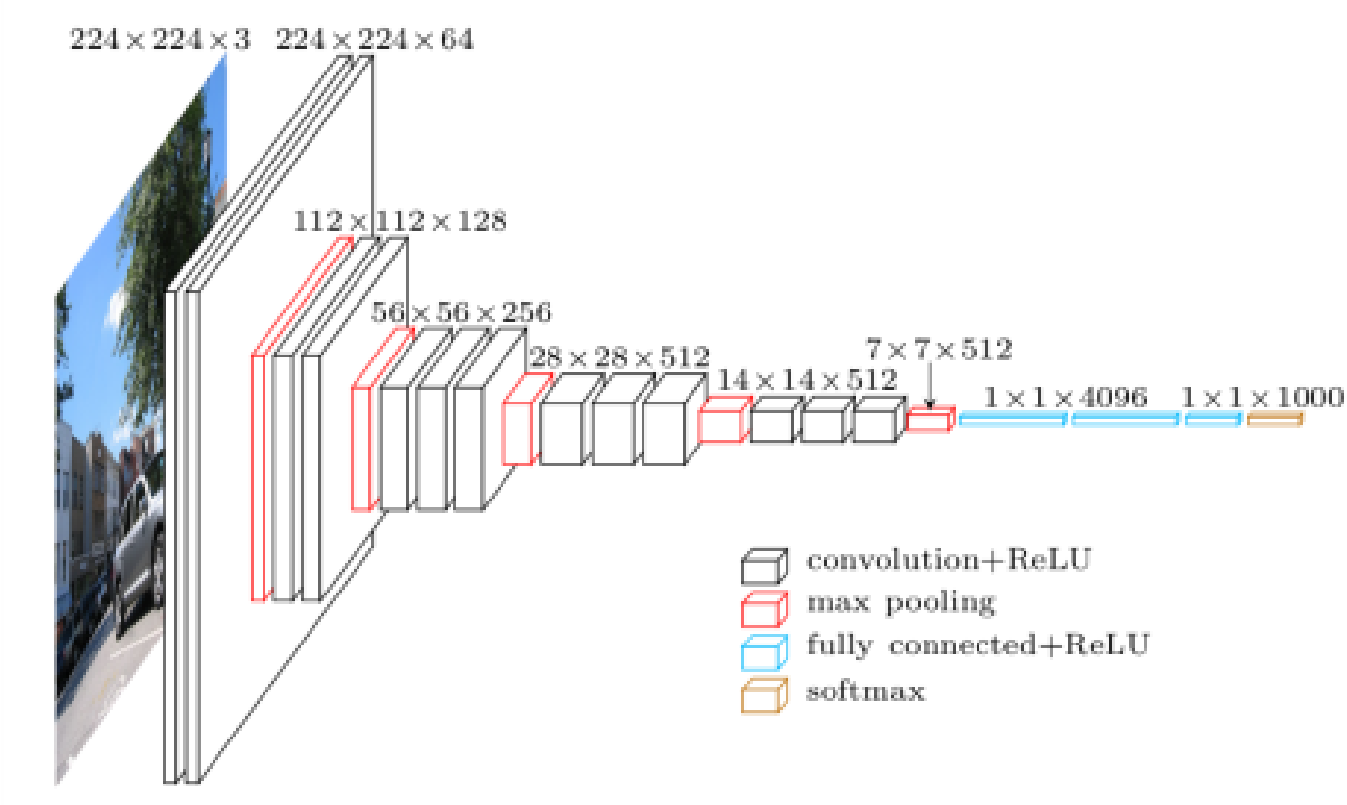

Figura 3.4: Exemplo de uma visão geral da transformação da Imagem em uma CNN com arquitetura VGG16

Capítulo 4

Resultados

4.1 Metodologia para avaliação do Método

Neste trabalho nós comparamos as taxas de acerto de cada algoritmo entre si e com o método manual (padrão ouro), de tal modo que deixou-se de lado a relação entre tempo de processamento e otimização dos algoritmos implementados. A arquitetura que foi utilizada é descrita na tabela 4.1.

Os algoritmos relacionados aos k-vizinhos mais próximos, rede neural sem peso e de segmentação, foram implementados usando a linguagem de programação C++. O algoritmo da rede neural convolucional foi implementado usando a linguagem de programação Python. Para facilitar o desenvolvimento, utilizou-se a biblioteca OpenCV 3.0, para a segmentação e a comparação dos resultados do KNN, e o Keras, para a implementação da rede neural convolucional, reduzindo o tempo de desenvolvimento consideravelmente.

Tabela 4.1: Configuração do computador utilizado para os testes.

Computador de teste		
Modelo:	HP G4-340BR	(laptop)
Processador:	Intel Core i5	2.40 GHZ
Núcleos:	4	
Mem. RAM:	6GB	
Sist. Oper.:	Linux	Ubuntu 16.04
Ling. de prog.:	gcc v5.4.0 e python v2.7	linux/amd64

4.1.1 K-Vizinhos Mais Próximos

O KNN foi desenvolvido em C++. Através dos pesos de cada pixel. Os valores de K usados para o experimento foram 1, 3, 5, 7 e 9. A implementação teve o uso da biblioteca OpenCV [26] para extrair os pixels de cada imagem. O método da distância euclidiana foi usado para definir cada resultado. A Tabela 4.2 mostra a taxa de acertos para os diversos valores de K.

K	Hit Rate (%)
1	42.3
3	45.7
5	58.1
7	54.7
9	51.6

Tabela 4.2: Taxa de acerto usando o algoritmo de KNN

Os experimentos iniciais confirmaram a baixa precisão e exatidão do método KNN aplicado à este tipo de problema. Como apontado anteriormente, este método é simples de implementar, porém para este tipo de problema não tem um resultado aceitável.

4.1.2 Rede Neural sem Peso

Na segunda séria de experimentos medimos o desempenho do método da rede neural sem peso. Neste caso, na implementação utilizamos 8 memórias de 16 bits para cada discriminador. Esta configuração gera 1 discriminadores de RAM com 64 entradas cada. Neste caso, as imagens foram redimensionadas para 64 pixels, numa matriz $8x8$, convertidas em tons de cinza e depois binarizadas, figura 4.1, sendo considerados três valores de threshold: 40%, 60% e 80%. O gráfico 4.2 mostra a taxa de acertos em relação aos valores de *thresholds* utilizados.

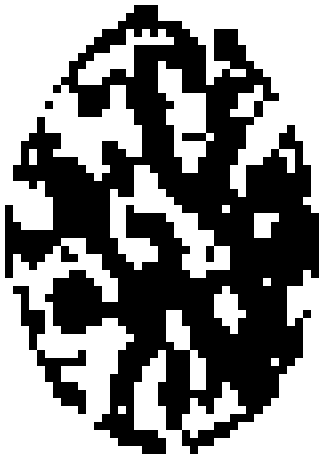

Figura 4.1: Exemplo de discriminador utilizado na implementação deste algoritmo.

Como observado no grafico 4.2, para um limite de 40%, temos uma taxa de acerto de 55.7%. Para um valor de 60%, temos um desempenho ligeiramente melhor, com uma taxa de $65,3\%$. Finalmente, considerando uma distância de 80%, temos uma taxa de acerto de $58,5\%$. Embora esta metodologia tenha uma taxa de sucesso um pouco melhor que o anterior, e seja de fácil implementação, o resultado obtido ainda não é satisfatório para este problema.

4.1.3 Rede Neural Convolucional

Nosso último experimento foi medir o desempenho do método da rede neural convolucional. Neste caso, usamos uma arquitetura VGG16, conforme ilustrado na figura 4.3. Neste caso, esta rede foi treinada usando 50 épocas com um tamanho de lote de 2.000 imagens.

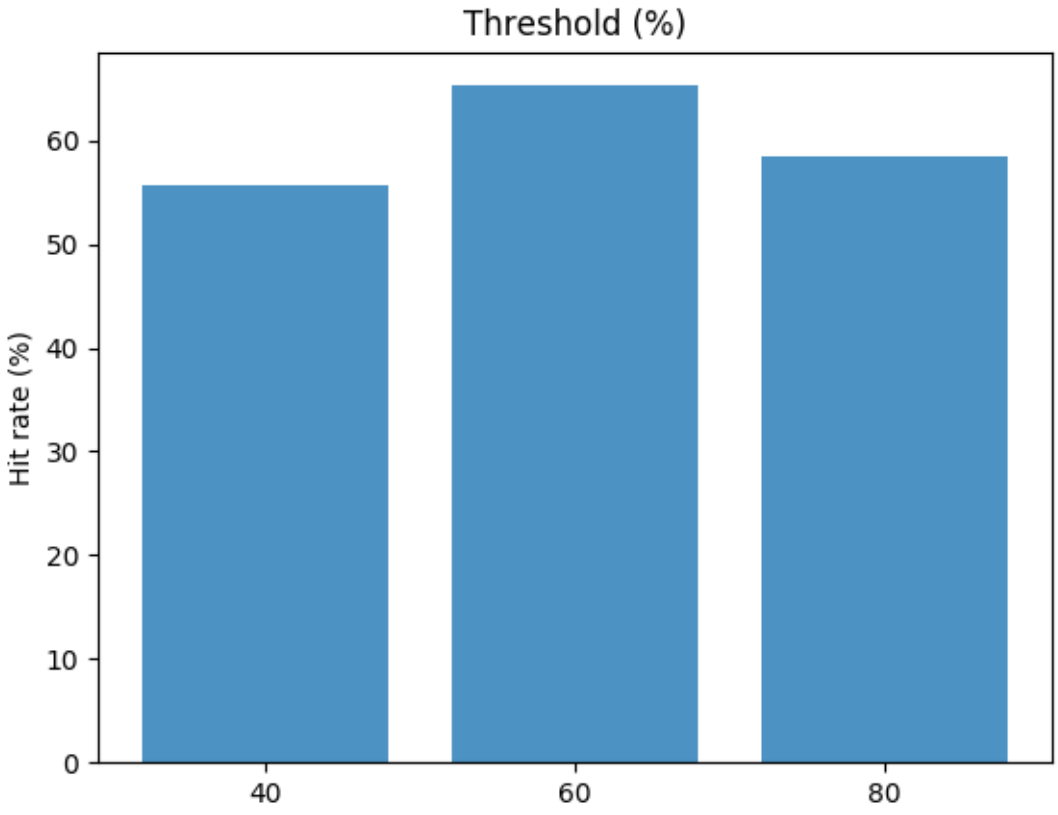

Figura 4.2: Resultado com três diferentes valores de threshold.

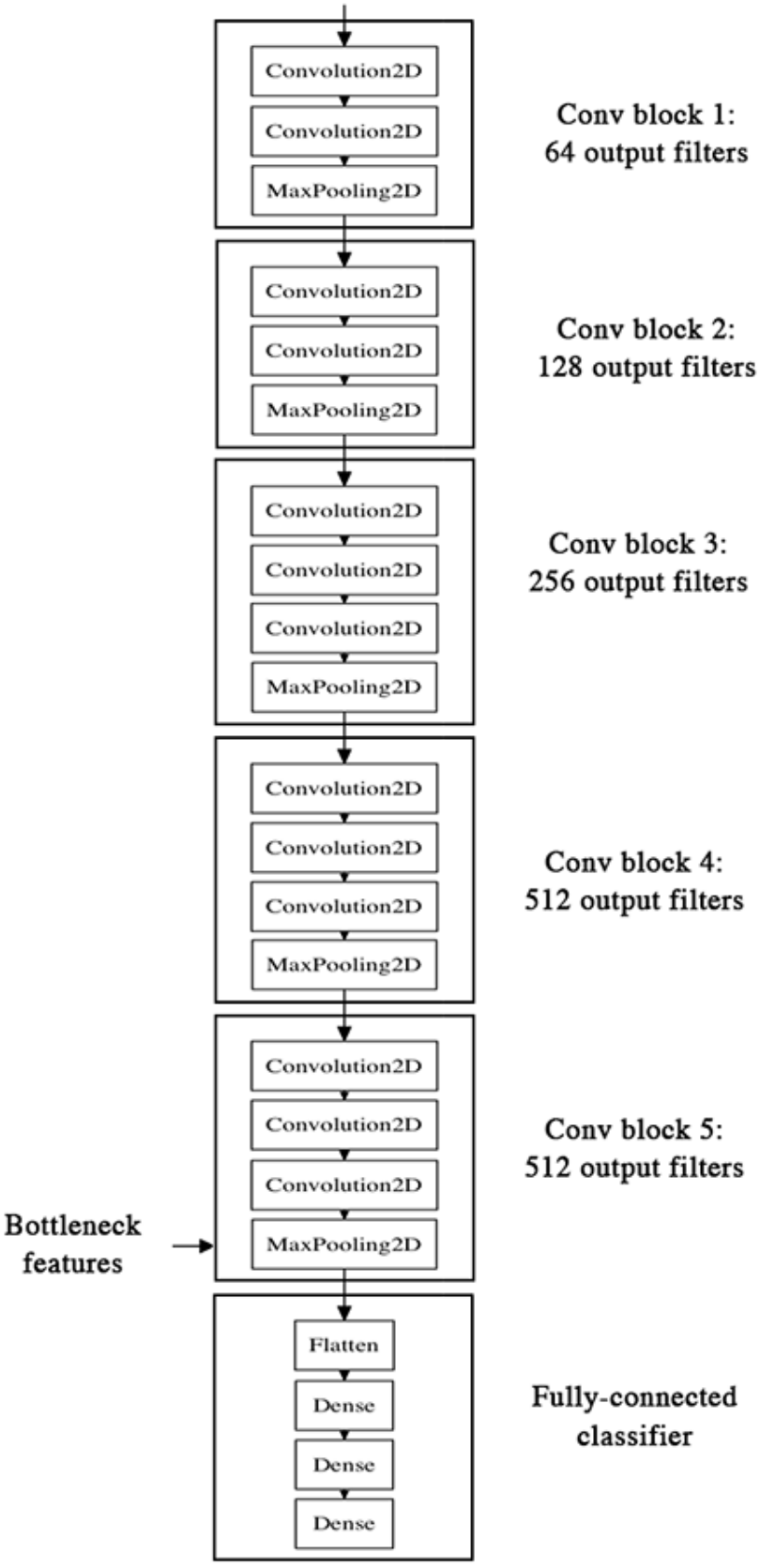

Figura 4.3: CNN com Arquitetura VGG16

Observamos que a implementação utilizando uma rede neural convolucional apresentou um resultado muito melhor, comparado com os anteriores. Como mostrado na figura 4.4, temos uma taxa de acertos acima dos 90%, sendo um ótimo resultado para a aplicação de contagem de microorganismos.

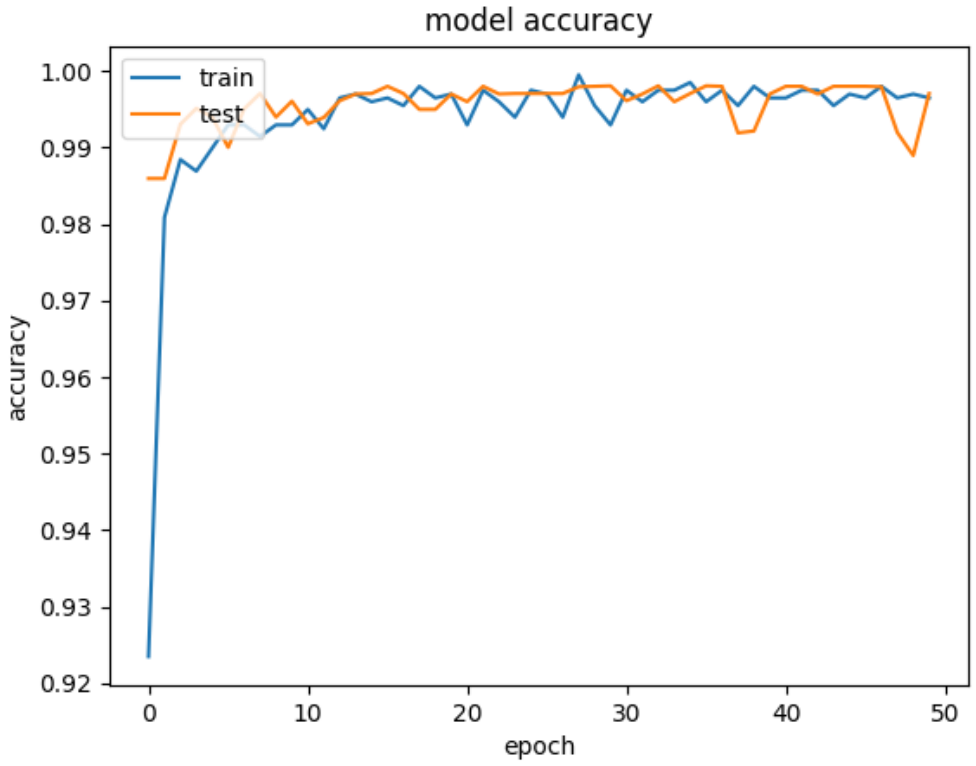

Figura 4.4: Gráfico de acertos e relação entre testes e treinos

Observamos ainda, que a taxa de falsos positivos é aceitavel, variando entre os valores de 1% até 10% dependendo da quantidade de épocas permitidas para o treinamento. A figura 4.5 ilustra a precisão desta rede.

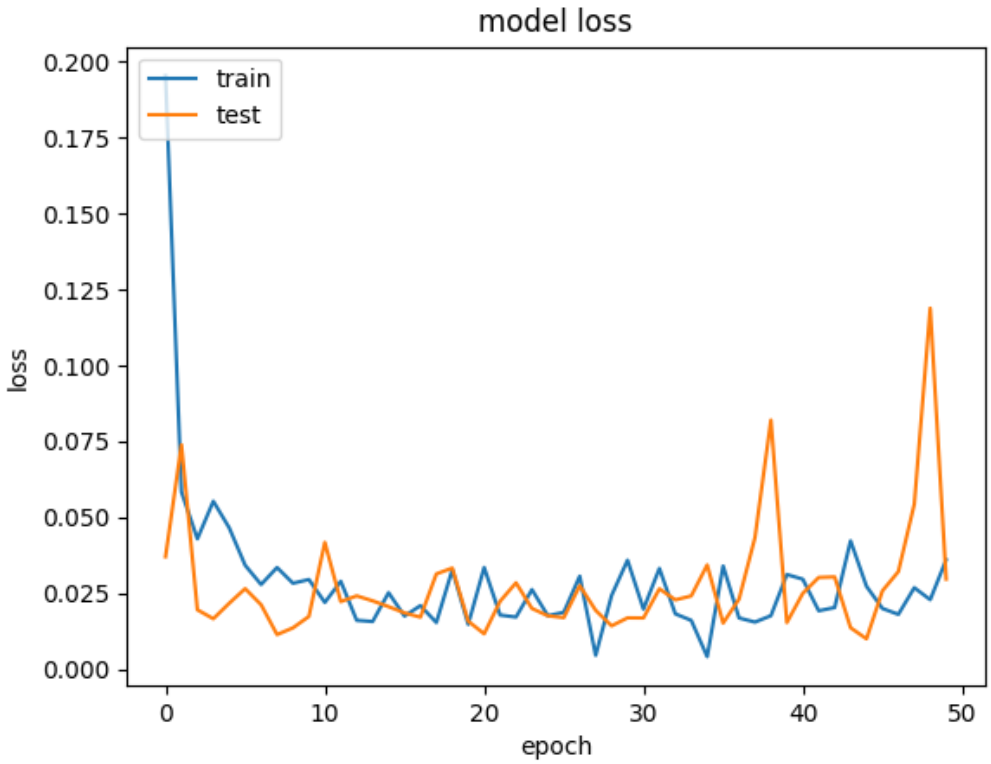

Figura 4.5: Taxa de erro e comparação entre testes e treinos

Capítulo 5

Conclusões

A revisão da literatura de inteligência artificial e realização da implementação e testes empíricos de alguns algoritmos nos permitiu compreender melhor o problema e assimilar algumas noções e padrões dos métodos de classificação automático. Dado a dificuldade do problema proposto e a vastidão de possíveis soluções, a escolha dos métodos comparativos foi de acordo com o nível de popularidade e uso desses no meio acadêmico e na indústria

Nossa implementação da rede neural convolucional proporcionou ganhos significativos no processamento para classificar os microrganismos. Podemos considerar que o uso de uma rede neural convolucional é o estado da arte para problemas de classificação. No entanto, este algoritmo requer um maior poder computacional para reduzir o tempo de treinamento e construção dos pesos da rede.

Em relação aos demais métodos, não se pode concluir que são péssimas escolhas. Para o problema proposto, eles não funcionaram muito bem, entretanto existem outros problemas que estes métodos empregados tem melhor funcionamento. Algo que não foi levado em consideração é a comparação da eficiência de cada algoritmo em relação ao tempo de processamento, tanto de treino quanto de execução. Neste trabalho, o principal problema era concluir qual o melhor método em relação a taxa de acerto.

Pode-se notar um grande aumento nas publicações relacionadas a Inteligência Artificial nos últimos anos, provavelmente porque empresas e instituições de ensino estão investindo muito neste assunto. Além disso, muitas outras áreas de conhecimento tem usado soluções desse tipo, criando uma multidisciplinalidade sobre o

tema. No caso, esse trabalho é uma interseção entre a biologia e a computação, pois o problema é relacionado a área da biologia, classificação de fitoplânctos, e a computação é a ferramenta para solucionar esse problema. Devido a esta popularidade, o trabalho tem bastante apelo acadêmico e um leque de ideiais para possíveis trabalhos futuros.

Com relação ao uso de redes neurais convolucionais, um assunto relativamente novo e tem sido aplicado com sucesso nas áreas de visão computacional e processamento de imagem. Anteriormente os algoritmos de classificação ou segmentação eram menos robustos pois utilizavam-se apenas de métodos da área de computação gráfica. Atualmente, o estado da arte para isso é usar algoritmos de inteligência artificial, e no caso, a rede neural convolucional é a ferramenta mais usada para esse tipo de problema. Um grande problema para as empresas e instituições de ensino, não é a criação ou otimização dos algoritmos mas a criação de bancos de dados grandes suficientes para se solucionar determinado problema. A aceleração dos algoritmos pode ser realizada com o uso de supercomputadores ou GPU's. Neste caso o importante é como balancear bem a rede criada e reduzir o overfiting que essa rede pode produzir.

Sendo assim, o trabalho desenvolvido atingiu seu objetivo e mostrou-se eficiente. Todos os códigos de implementação estão disponibilizados em https://goo.gl/UkqFSr.

5.1 Trabalhos Futuros

Como trabalhos futuros, consideramos explorar outras alternativas como usar uma rede neural convolucional para realizar a segmentação de imagens. A solução poderia ser aplicada diretamente, sem precisar passar por uma fase de pré-processamente, segmentando cada imagem e fazendo transformações lineares entre elas. Outro objetivo é melhorar o desempenho usando GPU para escrever algoritmo de rede convolucional. No caso seria comparado o ganho de eficiência para o treinamento da rede, e talvez de uma rede mais complexa do que a implementada neste trabalho.

Existe uma necessidade de aplicar este trabalho em um software interativo pra ser usado posteriormente pela biologia. Outro fator importante é disponbilizar de

maneira livre o dataset de imagens, para que outros pesquisadores usem isso em outros problemas. Além de aumentar mais o dataset, com a criação de outras classes, interagindo mais com os pesquisadores da área de biologia. Completando esse dataset, seria de interesse de orgãos governamentais de utilizá-los para trabalhos de preservação do meio ambiente.

Finalmente, o trabalho desenvolvido renderá um artigo que está em fase de escrita. Pelo volume de ideias, tem uma grande possibilidade de aprofundar mais no assunto e ter mais trabalhos produzidos em breve.

Referências Bibliográficas

[1] YAMAGUCHI, M., IMAI, I. "A microfluorometric analysis of nuclear DNA at different stages in the life history of Chattonella antiqua and Chattonella marina (Raphidophyceae)", *Phycologia*, v. 33, n. 3, pp. 163–170, 1994.

[2] ALEKSANDER, I., DE GREGORIO, M., FRANÇA, F. M. G., etãl. "A brief introduction to Weightless Neural Systems." In: *ESANN*, pp. 299–305. Citeseer, 2009.

[3] ALEKSANDER, I., THOMAS, W., BOWDEN, P. "WISARD a radical step forward in image recognition", *Sensor review*, v. 4, n. 3, pp. 120–124, 1984.

[4] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] MENEZES, M., BICUDO, C. E. D. M. "Freshwater Raphidophyceae from the State of Rio de Janeiro, Southeast Brazil", *Biota Neotropica*, v. 10, n. 3, pp. 323–331, 2010.

[6] LU, D., WENG, Q. "A survey of image classification methods and techniques for improving classification performance", *International journal of Remote sensing*, v. 28, n. 5, pp. 823–870, 2007.

[7] BALDI, P., BRUNAK, S. *Bioinformatics: the machine learning approach*. MIT press, 2001.

[8] "Brain Quote". `https://www.brainyquote.com/quotes/`. Accessed: 2018-04-21.

[9] ALTMAN, N. S. "An introduction to kernel and nearest-neighbor nonparametric regression", *The American Statistician*, v. 46, n. 3, pp. 175–185, 1992.

[10] HAMMING, R. W. "Error detecting and error correcting codes", *Bell System technical journal*, v. 29, n. 2, pp. 147–160, 1950.

[11] COOMANS, D., MASSART, D. L. "Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules", *Analytica Chimica Acta*, v. 136, pp. 15–27, 1982.

[12] ULLMANN, J. R. "Experiments with the n-tuple method of pattern recognition", *IEEE Transactions on computers*, v. 100, n. 12, pp. 1135–1137, 1969.

[13] GRIECO, B. P., LIMA, P. M., DE GREGORIO, M., etãl. "Producing pattern examples from "mental" images", *Neurocomputing*, v. 73, n. 7-9, pp. 1057–1064, 2010.

[14] LUDERMIR, T. B., DE OLIVEIRA, W. R. "Weightless neural models", *Computer standards & interfaces*, v. 16, n. 3, pp. 253–263, 1994.

[15] BUGMANN, G., TAYLOR, J. G. "A stochastic short-term memory using a pRAM neuron and its potential applications". In: *Proceedings British Neural Network Society Meeting (BNNS'93)*. Citeseer, 1993.

[16] LAB, L. "Convolutional Neural Networks (LeNet) - DeepLearning 0.1 documentation". Disponível em: <`http://deeplearning.net/tutorial/lenet.html`>.

[17] KRIZHEVSKY, A., NAIR, V., HINTON, G. "The CIFAR-10 dataset", *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[18] MATSUGU, M., MORI, K., MITARI, Y., etãl. "Subject independent facial expression recognition with robust face detection using a convolutional neural network", *Neural Networks*, v. 16, n. 5-6, pp. 555–559, 2003.

[19] YANN, L. "LeNet-5 convolutional neural networks", *URL*, 2015.

[20] BOTTOU, L., CORTES, C., DENKER, J. S., etãl. "Comparison of classifier methods: a case study in handwritten digit recognition". In: *Pattern Recognition, 1994. Vol. 2-Conference B: Computer Vision & Image Processing., Proceedings of the 12th IAPR International. Conference on*, v. 2, pp. 77–82. IEEE, 1994.

[21] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "ImageNet Classification with Deep Convolutional Neural Networks". In: Pereira, F., Burges, C. J. C., Bottou, L., etãl. (Eds.), *Advances in Neural Information Processing Systems 25*, Curran Associates, Inc., pp. 1097–1105, 2012. Disponível em: <`http://papers.nips.cc/paper/`

4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

[22] ZEILER, M. D., FERGUS, R. "Visualizing and Understanding Convolutional Networks", *CoRR*, v. abs/1311.2901, 2013. Disponível em: <http://arxiv.org/abs/1311.2901>.

[23] SZEGEDY, C., LIU, W., JIA, Y., etãl. "Going Deeper with Convolutions", *CoRR*, v. abs/1409.4842, 2014. Disponível em: <http://arxiv.org/abs/1409.4842>.

[24] SIMONYAN, K., ZISSERMAN, A. "Very Deep Convolutional Networks for Large-Scale Image Recognition", *CoRR*, v. abs/1409.1556, 2014.

[25] HE, K., ZHANG, X., REN, S., etãl. "Deep Residual Learning for Image Recognition", *CoRR*, v. abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

[26] BRADSKI, G. "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.

[27] HARALICK, R. M., SHAPIRO, L. G. "Image segmentation techniques", *Computer vision, graphics, and image processing*, v. 29, n. 1, pp. 100–132, 1985.

[28] VAN DYK, D. A., MENG, X.-L. "The art of data augmentation", *Journal of Computational and Graphical Statistics*, v. 10, n. 1, pp. 1–50, 2001.

[29] WANG, J., PEREZ, L. *The effectiveness of data augmentation in image classification using deep learning*. Relatório técnico, Technical report, 2017.

[30] WITTEN, I. H., FRANK, E., HALL, M. A., etãl. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[31] BOIMAN, O., SHECHTMAN, E., IRANI, M. "In defense of nearest-neighbor based image classification". In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.

[32] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[33] ZHANG, W. "Shift-invariant pattern recognition neural network and its optical architecture". In: *Proceedings of annual conference of the Japan Society of Applied Physics*, 1988.

[34] ZHANG, W., ITOH, K., TANIDA, J., etãl. "Parallel distributed processing model with local space-invariant interconnections and its optical architecture", *Applied optics*, v. 29, n. 32, pp. 4790–4797, 1990.

[35] ISO. *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. Geneva, Switzerland, International Organization for Standardization, fev. 2012. Disponível em: <`http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50372`>.

[36] VAN ROSSUM, G., DRAKE, F. L. *The Python Language Reference Manual*. Network Theory Ltd., 2011. ISBN: 1906966141, 9781906966140.