UNIVERSIDADE FEDERAL DE ALAGOAS

INSTITUTO DE MATEMÁTICA

PROGRAMA DE PÓS GRADUAÇÃO EM MATEMÁTICA

JOHANN FELIPE VOIGT

APRENDIZAGEM PROFUNDA PARA RECONHECIMENTO DE GESTOS DA MÃO USANDO IMAGENS E ESQUELETOS COM APLICAÇÕES EM LIBRAS

MACEIÓ - AL

DEZEMBRO DE 2018

JOHANN FELIPE VOIGT

APRENDIZAGEM PROFUNDA PARA RECONHECIMENTO DE GESTOS DA MÃO USANDO IMAGENS E ESQUELETOS COM APLICAÇÕES EM LIBRAS

Dissertação de Mestrado submetida em Dezembro de 2018 à Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Orientador: Prof. Dr. Thales Miranda de Almeida Vieira

MACEIÓ - AL

DEZEMBRO DE 2018

Catalogação na fonte
Universidade Federal de Alagoas
Biblioteca Central

Bibliotecária Responsável: Helena Cristina Pimentel do Vale – CRB4 - 661

V892a Voigt, Johann Felipe.

Aprendizagem profunda para reconhecimento de gestos da mão usando imagens e esqueletos com aplicações em libras / Johann Felipe Voigt. – 2018.

91f. : il.

Orientador: Thales Miranda de Almeida Vieira.

Dissertação (mestrado em Matemática) – Universidade Federal de Alagoas. Instituto de Matemática. Programa de Pós-Graduação em Matemática. Maceió, 2018.

Bibliografia: f. 89-91.

1. Matemática – Estudo e ensino. 2. Língua brasileira de sinais. 3. Leap motion. 4. Redes neurais. 5. Reconhecimento de gestos. I. Título.

CDU: 51:004.9

JOHANN FELIPE VOIGT

APRENDIZAGEM PROFUNDA PARA RECONHECIMENTO DE GESTOS DA MÃO USANDO IMAGENS E ESQUELETOS COM APLICAÇÕES EM LIBRAS

Dissertação de Mestrado submetida em Dezembro de 2018 à Banca Examinadora, designada pelo Colegiado do Programa de Pós-Graduação em Matemática da Universidade Federal de Alagoas, como parte dos requisitos necessários à obtenção do grau de Mestre em Matemática.

Banca examinadora:

Prof. Dr. Adelailson Peixoto da Silva
UFAL

Prof. Dr. Dimas Martinez Morera
UFAM

Prof. Dr. Tiago Figueiredo Vieira
UFAL

Prof. Dr. Thales Miranda de Almeida Vieira
Orientador
UFAL

Agradecimentos

Ao longo de meus trinta anos de vida, um incontável número de pessoas dividiu este palco comigo. Alguns trouxeram flores, outros vieram com espinhos, e alguns outros carregaram grandes buquês.

Todos, sem exceção, me ensinaram algo precioso e importante, seja me indicando qual caminho seguir, seja fazendo exatamente o contrário, me mostrando como eu mesmo não gostaria de ser.

À todos aqueles que não se importaram realmente comigo, por tudo que me ensinaram, lhes entrego aqui meu humilde agradecimento.

Aos demais, lembrem-se que não precisam de nada disso. Vocês já tem meu amor.

Resumo

Neste trabalho apresentamos metodologias baseadas em Aprendizagem Profunda (Deep Learning) para reconhecimento de gestos estáticos e dinâmicos da mão, com aplicações em sinais de Libras. Através de dados capturados pelo dispositivo Leap Motion, incluindo tanto imagens quanto esqueletos da palma da mão, avaliamos diversas arquiteturas de Redes Neurais para reconhecer gestos, com ênfase em sinais de Libras. As metodologias podem ser descritas em três etapas. Na primeira, buscamos reconhecer os gestos estáticos (poses) usando redes perceptron multicamadas (MLP) para os dados do esqueleto, redes convolucionais (CNN) para as imagens, e redes de múltiplas entradas, utilizando ambos os tipos de informação. Na segunda, classificamos individualmente gestos que incluam movimento, e para tanto incluímos camadas recorrentes Long Short-Term Memory (LSTM) em nossas arquiteturas. Para tornar o processo ainda mais preciso, aplicamos Transferência de Aprendizado nos blocos convolucionais, trazendo os parâmetros já treinados com as poses estáticas para dentro da rede projetada para os gestos dinâmicos, e avaliamos o resultado com e sem a transferência. Por fim, apresentamos um novo algoritmo que nos permita reconhecer online os mesmos gestos dinâmicos da etapa anterior, mas executados de forma sequencial, sem pausas, e sem ter informação sobre o início e final da execução de cada gesto.

Palavras-chaves: língua brasileira de sinais, leap motion, aprendizagem profunda, reconhecimento de gestos, redes neurais convolucionais, redes neurais recorrentes.

Abstract

In this work we present methodologies based on Deep Learning for the recognition of static and dynamic gestures of the hand, with applications in signs of Libras (Brazilian Sign Language). Through data captured by the Leap Motion device, including both images and skeletons of the palm, we evaluated several architectures of Neural Networks to recognize gestures, with emphasis on signs of Libras. The methodologies can be described in three stages. In the first one, we sought to recognize static gestures (poses) using multilayer perceptron networks (MLP) for skeletal data, convolutional networks (CNN) for images, and multiple input networks using both types of information. In the second, we individually classify gestures that include motion, and for this we include recurrent Long Short-Term Memory (LSTM) layers in our architectures. To make the process even more precise, we apply Learning Transfer to the convolutional blocks, bringing the previously trained parameters with the static poses into the network designed for the dynamic gestures, and evaluate the result with and without the transfer. Finally, we present a new algorithm that allows us to recognize online the same dynamic gestures from the previous step, but executed sequentially, without pauses, and without having information about the beginning and end of the execution of each gesture.

Keywords: brazilian sign language, leap motion, deep learning, hand gesture, convolutional neural networks, recurrent neural networks.

Lista de ilustrações

Lista de tabelas

Sumário

1. INTRODUÇÃO

Desde o início da raça humana, nós nos caracterizamos pela grande capacidade de desenvolver novas tecnologias, e junto delas, novas ferramentas que facilitem tarefas do nosso cotidiano. Por muitos anos, a maioria dessas ferramentas era de uso simples e direto, como martelos ou copos, garantindo uma também simplificada interação entre usuário e ferramenta.

O desenvolvimento tecnológico culminou na criação e popularização de microcomputadores, ferramentas formidáveis capazes de nos auxiliar em várias tarefas diferentes, com finalidades diferentes. Entretanto, uma ferramenta tão complexa tem potencial para exigir uma interação usuário-ferramenta igualmente complexa.

Se inicialmente a única forma de enviar comandos para os computadores era através de cartões perfurados, o advento de novos dispositivos como teclados, mouses, e então telas sensíveis a toque tornaram a interação com máquinas mais simples, precisa e acessível. Hoje, comandos de voz já são realidade em todo *smartphone*, e se comandos visuais, através de detecções de características do rosto ou da mão, ainda não são igualmente acessíveis e utilizados, existem trabalhos e pesquisas nessa área buscando ampliar suas aplicações.

Da mesma forma que comandos de voz permitem interação com computadores enquanto nos concentramos em outra tarefa, uma interação visual, sobretudo utilizando nossas mãos, permitiria um controle rápido e preciso sobre várias tarefas, além de propiciar que usuários com deficiência auditiva ou de fala tenham acesso à estas mesmas facilidades, ou até mesmo garantindo um meio de comunicação entre estes usuários e outros que não compreendem sua linguagem de sinais.

Uma grande dificuldade é, claro, a detecção precisa de tal interação. Nos últimos anos diversas novas tecnologias têm sido desenvolvidas e popularizadas nesse sentido, especialmente na captura de esqueletos, como o *Microsoft Kinect*[1], o *Intel RealSense*

3D[2] ou o *Leap Motion*[3], mas mesmo com essas tecnologias, o reconhecimento de sinais ou movimentos específicos é complexo, pois é necessário tolerar uma pequena variação em cada caso, já que duas pessoas diferentes vão inevitavelmente executar o mesmo movimento com alguma diferença, e mesmo duas execuções da mesma pessoa raramente são idênticas.

E naturalmente, esses dispositivos de captura ainda apresentam suas próprias adversidades, como a dificuldade em rastrear corretamente partes do corpo humano quando são obstruídas por algum objeto, incluindo outras partes do corpo que está sendo rastreado. Assim, o movimento de um braço nas costas de um usuário que esteja de frente para o Kinect acaba sendo questionável.

Enquanto isso, outras tecnologias também sofreram grandes avanços nos anos recentes: em particular, a explosão de capacidade computacional tornou o uso de Redes Neurais Profundas uma realidade, mesmo as mais complexas. Redes Neurais Convolucionais são hoje capazes de categorizar imagens com altíssima acurácia, diferenciando objetos ou cenários distintos com muita segurança. Ainda, Redes Neurais Recorrentes tornaram a avaliação de sequências de dados, com tamanhos arbitrários, mais uma realidade.

Ambas as abordagens, i.e., usar dados paramétricos (esqueletos) ou imagens, são promissoras, mas ambas possuem pontos frágeis. Com a finalidade de que os pontos fortes de uma sejam capazes de compensar os pontos fracos de outra, experimentaremos a utilização destas redes neurais juntamente com o uso dos esqueletos previamente mencionados, para diferenciar diferentes gestos e movimentos que possam ser úteis na interação entre homem e computadores. Neste trabalho, avaliamos os 26 sinais da Linguagem Brasileira de Sinais (Libras), mas os resultados poderão então serem ampliados para qualquer gesto ou movimento realizado apenas com uma mão.

Certamente, mesmo que esses gestos possam ser reconhecidos de forma precisa, eles terão pouca utilidade em utilizações práticas, a menos que o usuário possa executar tais gestos, de forma natural, em frente a uma máquina. É com isso em mente que buscaremos não apenas o reconhecimento dos sinais de Libras já mencionados, mas ainda que eles sejam reconhecidos sem pausas, enquanto o usuário os executa em sequência, uma após a outra.

No capítulo 2 a seguir, descreveremos trabalhos anteriores nesta linha de pesquisa, de como alguns influenciaram a nossa abordagem, e de como nos baseamos diretamente

de alguns deles para esse trabalho. No capítulo 3 abordaremos conceitos importantes para compreender os processos desse trabalho, sobretudo acerca de Redes Neurais e Processamento de Imagens. No capítulo 4 explicaremos em detalhes o que é e como funciona o sensor óptico que utilizamos, quais pré-processamentos foram necessários sobre a informação capturada por ele, quais estruturas optamos por utilizar em nossas Redes Neurais, e como funciona o algoritmo que propomos para o processamento de gestos em sequência. Por fim, o capítulo 5 inclui os resultados diretos de cada um dos testes que efetuamos, e algumas observações e argumentações que pudemos traçar sobre tais resultados.

2. TRABALHOS RELACIONADOS

Em se tratando de trabalhos de reconhecimento de gestos ou poses, grande parte dos trabalhos que serão apresentados aqui abordam tanto o caso de reconhecimento de gestos estáticos quanto o de dinâmicos. De fato, mesmo o que em geral o objetivo final dos trabalhos seja o segundo caso, as soluções para o primeiro muitas vezes podem ser estendidas para o segundo, e em outras vezes o caso dinâmico pode ser visto como a aplicação sequencial de vários casos estáticos.

O reconhecimento de poses (e, mais tarde, especificamente o de gestos de mão) tem evoluído e acompanhado o desenvolvimento de novas tecnologias, sobretudo no que diz respeito aos sensores usados na captura dos movimentos, sejam eles ativos ou passivos [4], passando por câmeras simples ou webcams [5] ou luvas de dados [6], chegando até os mais recentes sensores de profundidade [7] ou de movimento [8].

Em 2014, Marin et al.[9] utilizou o *Leap Motion* em conjunto com uma das mais populares câmeras de profundidade atualmente, o *Kinect*, com um classificador SVM (*Support Vector Machine*) de multi-classes para reconhecer gestos de mão. Dois anos depois, Lu et al.[10] alcançou bons resultados utilizando apenas o *Leap Motion*.

Schmidt et al.[11] também utilizou apenas o *Leap Motion*, com tecnologias de SVM e Florestas Aleatórias para reconhecer sinais, e o conjunto de descritores que ele extraiu do esqueleto fornecido pelo *Leap Motion* mais tarde inspirou o trabalho de Silva[8], que seria uma base importante para nosso trabalho atual. Entretanto, nenhum dos trabalhos mencionados até agora utilizou Redes Neurais Profundas.

A publicação da camada LSTM (*Long-short Term Memory*) por Hochreiter e Schmidhuber[12] resolveu o principal problema que afligia camadas recorrentes naquela época, e desde então ela tem demonstrado enorme capacidade de aprendizado em diversos campos, como reconhecimento de fala [13] ou traduções entre idiomas [14, 15].

Tendo em vista esses bons desempenhos, Donahue et al.[16] propôs um sistema de categorização de vídeos baseado em uma estrutura que chamou de LRCN (*Long-term Recorrent Convolutional Network*), que se trata de camadas convolucionais processando separadamente cada frame de entrada do vídeo, e repassando seus resultados para uma camada LSTM.

Essa mesma estrutura foi utilizada por John et al.[17] para o reconhecimento de comandos com a mão, para uma interface sem toque em veículos automotivos, apenas com pequenas variações, como efetuar um pré-processamento para estipular quais frames serão repassados à camada LSTM, ao invés de enviar todos.

Logo, quando tratamos do reconhecimento de gestos de mão, por um lado temos bons desempenhos utilizando o esqueleto fornecido pelo *Leap Motion* e classificadores SVM, e por outro temos bons desempenhos utilizando redes neurais profundas, com camadas CNN e LSTM processando uma sequência de imagens. Buscaremos, nesse trabalho, reunir ambos os elementos em uma mesma estrutura, e avaliar se o uso de ambos é ou não positivo para a solução desse problema.

Por fim, para possibilitar a utilização dessa técnica em aplicações práticas, há uma clara necessidade de que todo esse processo seja executado em tempo real e de forma não segmentada. É importante destacar que poucos trabalhos na literatura focaram no problema de reconhecimento online, como [18] e [19], onde algoritmos para reconhecimento online em Florestas de Decisão e Grafos de Ação foram apresentados. Porém, não é de conhecimento do autor nenhum trabalho que trate de reconhecimento de gestos online usando aprendizagem profunda. Apresentamos neste trabalho uma abordagem inédita nesta linha.

3. PRELIMINARES

Ao longo desse trabalho utilizaremos conceitos relacionados às áreas de Aprendizagem de Máquina, e Processamento de Imagens. Quanto ao primeiro, abordaremos sobretudo conceitos referentes à Redes Neurais, uma metodologia de Aprendizagem Supervisionada. Quanto ao segundo, especialmente conceitos referentes ao tratamento de imagens através de filtros convolucionais. Nas subseções seguintes abordaremos as propriedades mais essenciais para o nosso trabalho acerca desses temas.

3.1 Aprendizagem supervisionada

Este ramo da Aprendizagem de Máquina consiste em estudar e otimizar a tarefa de definir uma função $f : \hat{\mathcal{X}} \to \mathcal{Y}$, que mapeie adequadamente determinadas entradas (exemplos) para determinadas saídas (a resposta correta àquele exemplo), baseado em exemplos de um conjunto de treinamento $\mathcal{X} \subset \hat{\mathcal{X}}$ previamente fornecido, antes de ser generalizado para o conjunto $\hat{\mathcal{X}}$. Mais precisamente, cada exemplo é um par consistindo em um elemento $x \in \mathcal{X}$, o dado de entrada; e um valor de saída correspondente, $y \in \mathcal{Y}$.

Um algoritmo analisa os dados de treinamento de $\mathcal{X}$ e cria uma função f que aproxima os resultados de saída a partir das entradas, e que pode então ser usada em novos exemplos $x \in \hat{\mathcal{X}}$, mesmo os que não foram usados no treinamento da função, ou seja, $x \notin \mathcal{X}$. Mais especificamente, exemplos que ainda não foram categorizados ou analisados por um humano poderão ser classificados pelo algoritmo.

O cenário ideal culmina em uma função capaz de classificar corretamente não apenas todos os exemplos do treinamento, ou seja $\forall x_k \in \mathcal{X}, f(x_k) = y_k$, mas ainda os exemplos novos, não utilizados no treinamento, isto é, $\forall \hat{x} \in \hat{\mathcal{X}}, f(\hat{x}) = y$. Isso significa que a função foi capaz de generalizar o conjunto de treinamento para englobar situações

inesperadas.

O processo de Aprendizagem Supervisionada é bastante natural ao ser humano, pois grande parte de nosso próprio aprendizado é feito da mesma forma, tanto quando aprendemos diretamente de um professor quanto quando usamos livros escritos por especialistas, e principalmente durante nossos primeiros anos de vida.

Muito antes de aprendermos as diferenças essenciais entre cães e gatos, nossos familiares e pessoas ao nosso redor nos ensinam quais são cães, e quais são gatos. Apenas com essa informação, somos capazes de "criar uma função f" capaz de diferenciá-los, mesmo que a maioria de nós sequer estude com profundidade as diferenças morfológicas entre canídeos e felinos.

Contrariamente a essa linha de pensamento, existe o Aprendizado não-supervisionado, que é feito sem auxílio, isto é, se baseia unicamente no conjunto $\mathcal{X}$, sem o auxílio do conjunto Y para guiar sua função f. As vantagens dessa metodologia é autoexplicativa: sem a necessidade de definir explicitamente um y_k para cada $x_k \in \mathcal{X}$, podemos contar com conjuntos de treinamento mais amplos, gerados com menos esforço.

Naturalmente, sem um guia através do qual indicarmos o que buscamos extrair do conjunto de treinamento $\mathcal{X}$, a Aprendizagem Não Supervisionada pode encontrar características dos elementos $x \in \mathcal{X}$ que não são necessariamente as que nós desejamos. Por exemplo, fornecido um conjunto de imagens de cães ou gatos, o sistema pode separar as imagens por "animais com pelo claro" e "animais com pelo escuro", ao invés de separá-los entre "cães" e "gatos".

Quanto aos problemas abordados por Aprendizagem Supervisionada, podemos diferenciá-los entre problemas de Classificação, que se preocupam em separar os exemplos dados em grupos diferentes, com características semelhantes; e problemas de Regressão, que se preocupam em prever uma resposta numérica real.

Por exemplo, prever o valor de venda de uma casa baseado em seu tamanho e localização, ou prever a quantidade de álbuns vendida de uma banda, baseado na popularidade de seus shows e nas vendas anteriores, são problemas de regressão, que oferecem como valores de $f(x) = y$ qualquer valor numérico válido dentre um certo alcance, a depender do problema.

Por outro lado, prever em qual bairro uma casa se localiza, baseado em seu tamanho

e preço de venda, ou prever se os valores de ações de uma empresa irão aumentar ou diminuir no dia seguinte, são problemas de classificação, que possuem respostas $f(x) = y$ como valores discretos, como “Bairro A” ou “Bairro C”, ou “sim” ou “não”.

O objetivo deste trabalho é criar um sistema capaz de determinar qual sinal foi feito diante da câmera, sendo um problema claramente discreto. Portanto, de agora em diante nosso foco será em problemas de Classificação.

Existem vários algoritmos disponíveis que seguem esses conceitos, cada um com suas vantagens e desvantagens, desde a simples Regressão Linear, até casos mais complexos como a Máquina de Vetores de Suporte (em inglês, SVM), Aprendizagem baseada em Árvores de Decisão, ou Redes Neurais Artificiais. Neste trabalho, iremos focar nas últimas.

3.2 Redes Neurais Artificiais (e deep learning)

Redes Neurais Artificiais são sistemas computacionais vagamente inspirados na estrutura das Redes Neurais Biológicas de animais. Na maioria dos casos é um sistema de aprendizagem supervisionada, capaz de aprender determinada tarefa mesmo sem que sejam programados detalhes específicos referentes à ela.

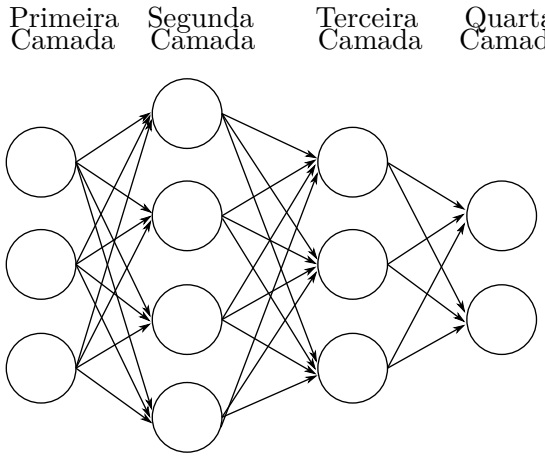
Por exemplo, a partir de um banco de dados de imagens categorizadas como “com avião” e “sem avião”, uma Rede Neural Artificial vai ser capaz de classificar corretamente imagens novas de aviões na categoria “com avião”, mesmo que em momento nenhum lhe informarmos que um avião possui asas e hélices, ou o formato específico do mesmo. A partir dos exemplos fornecidos, algumas características são aprendidas automaticamente pela máquina para resolver o problema de classificação.

Uma Rede Neural Artificial é formada por um conjunto de unidades, representando neurônios, que possuem conexões uns com os outros, representando sinapses. Cada neurônio então recebe um determinado sinal como entrada, efetua um determinado processamento no mesmo, e o repassa para os neurônios adiante.

Em geral esse sinal é representado por um número real, e cada conexão entre dois neurônios por um coeficiente chamado de peso (normalmente representado pela letra w, e com índices indicando quais neurônios ele conecta), que fortalece ou enfraquece o sinal durante a transmissão. O processo de treinamento consiste em determinar valores para

estes pesos, de forma a otimizar a classificação dos exemplos.

Figura 1 – Grafo representativo de uma Rede Neural



Fonte – Autor, 2018.

Os neurônios são tipicamente organizados em camadas, como mostra a Figura 1, e cada camada costuma efetuar diferentes tipos de transformações em seus sinais, enquanto os mesmos viajam da primeira camada (entrada) até a última (saída).

As primeiras ideias conceituando Redes Neurais surgiram ainda na década de 40, mas ainda levariam duas décadas até o sistema se tornar reconhecível com o que temos atualmente. No final da década de 60, observou-se que a falta de capacidade computacional da época tornaria o sistema impossível de ser implementado.

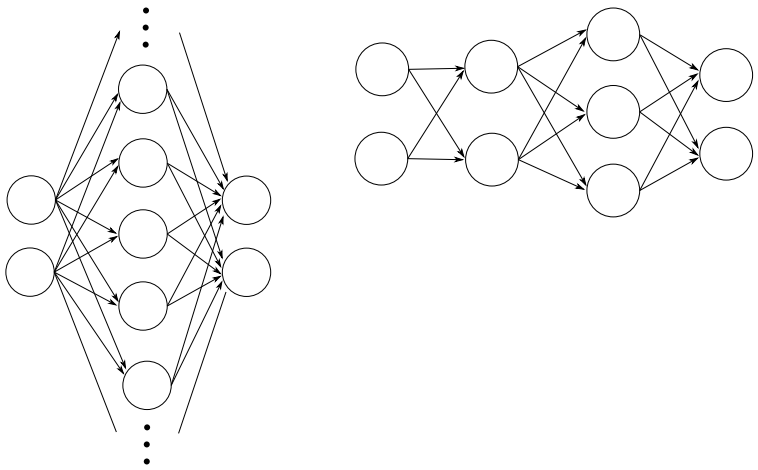
Apenas com o desenvolvimento do algoritmo de Retropropagação em 1975, e o uso de processamento distribuído paralelamente na década de 80, as Redes Neurais voltaram a ser foco constante de pesquisa e desenvolvimento. Também nessa época, as primeiras ideias de Redes Neurais Profundas foram conceituadas.

Um grande atrativo das Redes Neurais é a capacidade de reduzir problemas não-lineares para outros problemas menores e mais simplificados, e então resolvê-los de forma mais eficiente. Em geral, uma maior complexidade vai exigir uma quantidade maior de conexões, e portanto, de neurônios; inevitavelmente, chegaremos no ponto onde precisaremos optar por redes amplas ou profundas.

Quando possuem poucas camadas, mas cada camada contém muitos neurônios, classificamos a Rede Neural como Ampla (Figura 2a). Ao contrário, quando ela possui várias camadas, mas cada uma com menos neurônios, a classificamos como Profunda (Figura 2b).

Figura 2 – Comparativo entre tipos diferentes de Redes Neurais

(a) Rede Neural Ampla (b) Rede Neural Profunda



Testes empíricos sugerem que uma rede ampla precisa de dez vezes mais conexões para obter o mesmo resultado de uma rede profunda, ou seja, as profundas alcançam os mesmos resultados com menos conexões, e são mais computacionalmente eficientes. Por esse motivo as Redes Neurais Profundas tem tido maior foco de pesquisas nos anos recentes.

3.2.1 Backpropagation e Método do Gradiente Descendente

Assim como outras técnicas de Aprendizagem Supervisionada, as Redes Neurais tratam de definir uma função $f : \mathcal{X} \to \mathcal{Y}$ capaz de mapear adequadamente os exemplos do banco de dados, definidos como elementos do conjunto $\mathcal{X}$ nos resultados esperados, definidos como elementos do conjunto $\mathcal{Y}$. Essa função é explicitamente definida através de um número específico de coeficientes $\mathcal{C} = \{c_1, c_2, ..., c_p\}$, cuja ordinalidade p depende diretamente da estrutura da rede.

Posteriormente, ainda desejaremos estender a f para um conjunto $\hat{\mathcal{X}}$, com $\mathcal{X} \subset \hat{\mathcal{X}}$ e $\hat{\mathcal{X}}$ sendo não apenas os exemplos encontrados no treinamento, mas também casos novos, que precisam ser adequadamente generalizados.

De forma geral, iniciamos os coeficientes c_i com valores arbitrários, e em seguida entra o processo de treinamento da função, que é feito em etapas. Sinalizaremos então como $c_i^{(0)}$ o i-ésimo coeficientes inicial, e $c_i^{(j)}$ o i-ésimo coeficiente da j-ésima etapa, assim como usaremos $f^{(0)}$ para a função inicial e $f^{(j)}$ para a função definida com os coeficientes

da j-ésima etapa.

Etapa após etapa, modificamos os coeficientes $c_i^{(j)}$de modo a aproximar a função $f^{(j)}$ de nosso ideal f, que é capaz de levar todos os exemplos $x \in \mathcal{X}$ em suas respostas corretas $y = f(x) \in \mathcal{Y}$, e ainda generalizá-los para todo $x \in \hat{\mathcal{X}}$.

Para isso, inicialmente estabelecemos uma função de erro (por vezes também chamada de função Objetivo, ou Função de Perda), $r : \mathcal{W} \times \mathcal{C} \to \mathbb{R}^+$, que mapeia uma base de treinamento $\mathcal{X} \in \mathcal{W}$ e certa função $f^{(j)}$, aqui representada pelo conjunto de seus coeficientes $c_i^{(j)}$, sobre um valor não negativo que é nulo quando os coeficientes correspondem a uma função ideal f, e aumenta quanto mais distante desse ideal ela está.

Ou seja, em primeiro lugar fixamos uma base de exemplos $\mathcal{X}$, definimos uma função inicial $f^{(0)}$, e avaliamos o valor da função de erro $r(\mathcal{X}, f^{(0)})$. Desse ponto em diante, trata-se de um problema de otimização, onde desejamos encontrar o ponto mínimo de r em relação a sua segunda variável.

Uma função de erro frequentemente utilizada por sua simplicidade é a distância euclidiana entre o resultado calculado de $f^{(j)}(x_k) = y_k^{(j)}$ e o resultado informado de y_k, onde $x_k \in \mathcal{X}$ é o k-ésimo exemplo do conjunto de treinamento $\mathcal{X}$: $d(y_k^{(j)}, y_k) = |y_k^{(j)} - y_k|^2$.

Tendo em vista a simplificação dos processos seguintes, podemos adicionar um fator de $\frac{1}{2}$ em cada distância, e então tomamos a nossa função r como a média das distâncias de cada um dos exemplos no conjunto de treinamento. Ou seja, admitindo que nosso conjunto $\mathcal{X}$ possua n elementos, uma definição comum de função de erro é expressa na Equação 3.1:

$$r(\mathcal{X}, f^{(j)}) = \frac{1}{2n} \sum_{x \in \mathcal{X}} |f^{(j)}(x) - y|^2 \tag{3.1}$$

É interessante ressaltar que Redes Neurais com objetivos distintos podem se beneficiar de funções de erro distintas. Enquanto a distância euclidiana já mencionada é de simples compreensão, ela não lida adequadamente com alguns tipos de dados, ou melhor dizendo, outras funções de erro lidam de forma mais eficiente. Por exemplo, Redes Neurais focadas em categorização de elementos frequentemente utilizam uma função de erro baseada em Entropia Cruzada, como abordado por Nielsen[20].

3.2.1.1 Gradiente Descendente

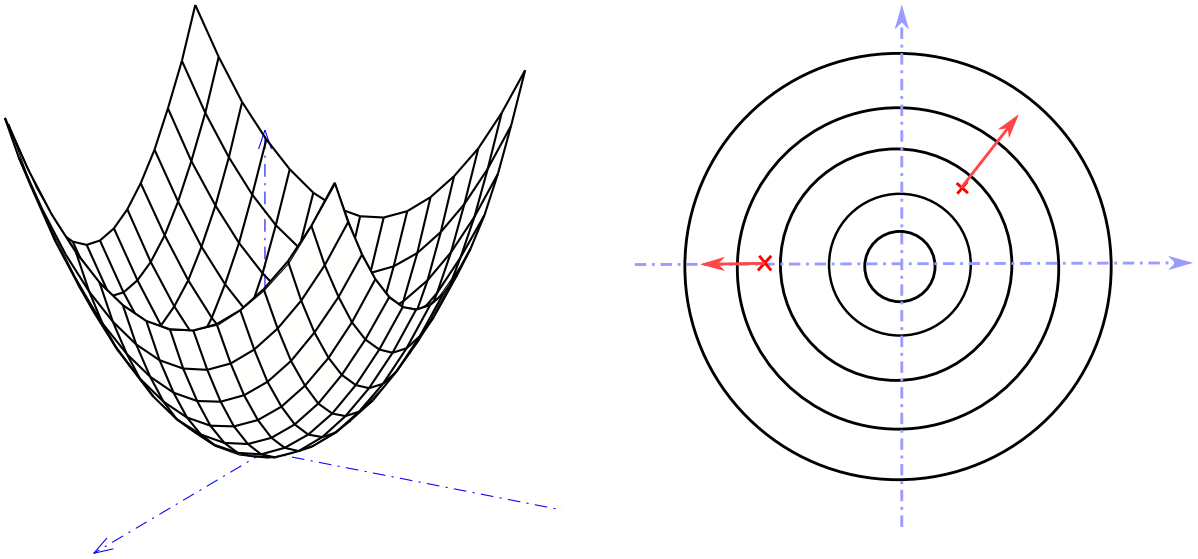
Para minimizar a função $r(\mathcal{X}, f^{(j)})$, utilizaremos a estratégia do Gradiente Descendente. O Gradiente de uma aplicação $\phi : \mathcal{U} \subset \mathbb{R}^m \to \mathbb{R}$ qualquer é um vetor $v \in \mathbb{R}^m$ onde cada coordenada corresponde a uma das derivadas parciais da aplicação em questão: $\nabla\phi(x) = (\frac{\partial\phi}{\partial x_1}(x), ..., \frac{\partial\phi}{\partial x_m}(x))$.

Uma de suas características mais interessantes é a sua interpretação geométrica: Quando calculado sobre um ponto qualquer do domínio da aplicação, a direção do Gradiente indica em qual direção a imagem da aplicação cresce com maior intensidade, como ilustrado na Figura 3.

Figura 3 – Análise do gradiente da aplicação $z^2 = x^2 + y^2$

(a) Gráfico da Parabolóide

(b) Curvas de nível da Parabolóide, com o gradiente calculado em dois pontos



Fonte – Autor, 2018.

Mas em nosso caso particular, desejamos encontrar o ponto mínimo da função $r(\mathcal{X}, f^{(j)})$, e a para isso a direção onde ela mais cresce é pouco interessante; de fato, queremos exatamente o contrário disso. Essa estratégia consiste em tomar a direção do gradiente v, e caminhar na direção oposta a ele , $-v$.

O processo completo consiste nos seguintes passos:

1. Calcular as derivadas parciais de $r(\mathcal{X}, f^{(j)})$ em relação a cada uma das coordenadas de sua segunda variável, assim definindo $v = \nabla r(\mathcal{X}, f^{(j)})$;

2. Atualizar cada uma das coordenadas de $f^{(j)}$ de acordo com $c_i^{j+1} = c_i^j - \alpha v$, onde α é chamado de Fator de Aprendizagem, efetivamente definindo os coeficientes da função $f^{(j+1)}$;

3. Avaliar $r(\mathcal{X}, f^{(j+1)})$ e verificar se o erro está dentro do aceitável;

4. Se necessário, repetir o processo para $r(\mathcal{X}, f^{(j+1)})$.

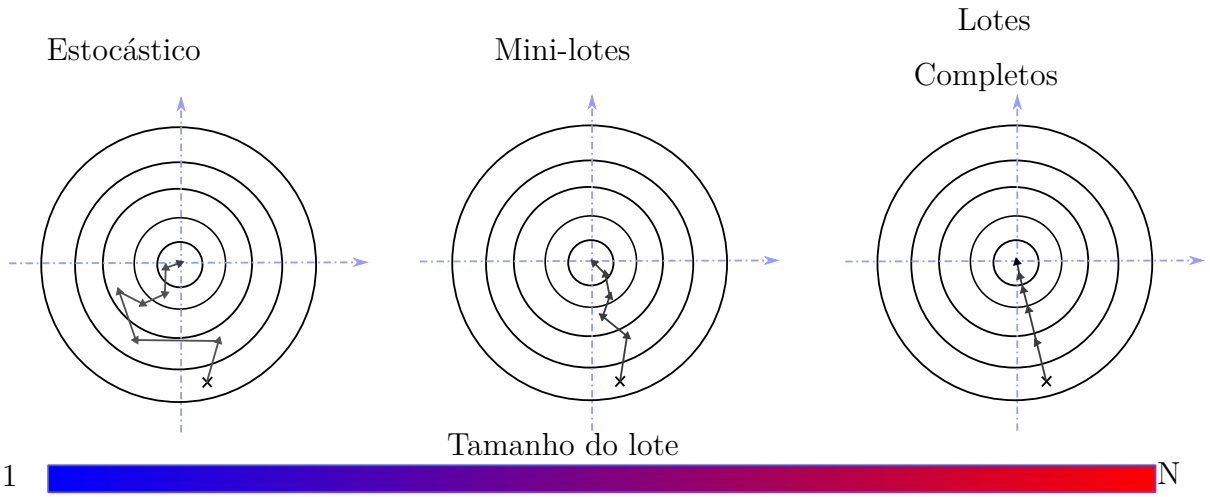
Observe que neste processo, para cada uma das etapas, é necessário calcular $f^{(j)}(x), \forall x \in \mathcal{X}$, o que pode se tornar computacionalmente exaustivo para um conjunto de treinamento $\mathcal{X}$ muito grande, especialmente porque o processo tende a ser repetido por um longo número de etapas.

Assim, uma solução natural para esse problema foi realizar os processos não sobre todo o conjunto de treinamento $\mathcal{X}$, mas sobre apenas um exemplo $x \in \mathcal{X}$, escolhido arbitrariamente. A cada etapa seguinte, um novo exemplo, $x^{(j)} \in \mathcal{X}$, é escolhido e utilizado, e em geral, repetem-se etapas até que todos os exemplos de $\mathcal{X}$ tenham sido utilizados.

Essa variação em particular é conhecida como Gradiente Descendente Estocástico (ou na sigla em inglês SGD), e apesar de ter a melhor eficiência computacional possível, ao levarmos em conta apenas um exemplo para o cálculo do gradiente, estamos nos tornando mais vulneráveis a ruído, e um exemplo particularmente ruim pode resultar em um gradiente que não representa o melhor caminho em direção ao ponto mínimo. Em geral, esse problema é corrigido nas etapas seguintes, e mesmo que em passos menos consistentes, o processo leva seguramente ao destino final, ainda em menos tempo do que se utilizássemos todo os exemplos de $\mathcal{X}$ para calcular o gradiente, que nos daria um gradiente perfeito, porém, muito demorado de se calcular.

Porém, podemos combinar o melhor de cada uma das duas abordagens utilizando o Gradiente Descendente com mini-lotes, que agrupa exemplos do conjunto $\mathcal{X}$ em blocos de um número flexível de exemplos arbitrários, com por exemplo, 8, 16 ou 32 exemplos, denominado lote. A cada etapa do Gradiente Descendente, o cálculo do gradiente é feito sobre a média dos exemplos deste lote. Assim o custo computacional é tremendamente reduzido, mas ainda possuímos vários exemplos para normalizar o gradiente calculado, tornando-o menos vulnerável a ruído de exemplos ruins, como mostra a Figura 4.

Figura 4 – Comparativo do comportamento de várias abordagens diferentes do Método do Gradiente Descendente



Fonte – Autor, 2018.

3.2.1.2 Retropropagação

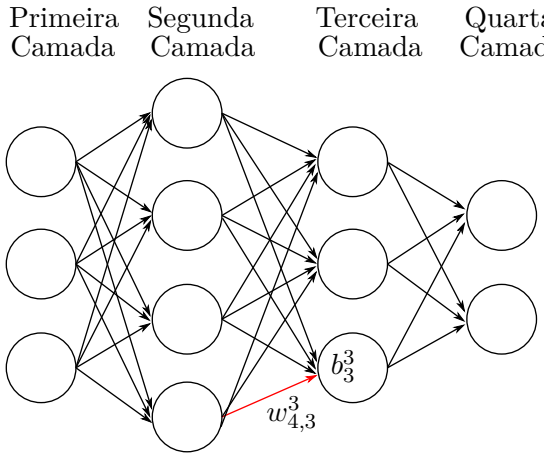
Com o aumento da complexidade da estrutura da Rede Neural, a função f se torna proporcionalmente complexa, e junto com ela, a Função de Erro r, até o ponto onde calcular o gradiente de r torna-se um desafio. Para lidar com isso, utilizamos o algorítimo de Retropropagação de Erros, uma implementação da Regra da Cadeia específica para Redes Neurais.

Para compreender o funcionamento do algoritmo, é conveniente diferenciar os coeficientes c_i da função f em dois tipos: os pesos w, representando conexões entre dois neurônios, e sendo um fator multiplicativo que potencializa ou enfraquece o uso do neurônio que está recebendo a conexão; e os *biases* b, parcelas aditivas que somam ou subtraem valor diretamente de cada neurônio.

Denominaremos $w^l_{j,k}$ o peso referente à conexão entre o k-ésimo neurônio da l-ésima camada, e o j-ésimo neurônio da $(l-1)$-ésima camada. De forma análoga, chamaremos de b^l_j o *bias* referente ao j-ésimo neurônio da l-ésima camada. A Figura 5 tem uma indicação visual dessa notação.

Podemos agora caracterizar a saída o^l_j de cada j-ésimo neurônio da l-ésima camada, como sendo a soma dos neurônios da camada anterior, cada um devidamente processado pelo peso correspondente entre os neurônios, somado com o *bias* daquele neurônio em

Figura 5 – Notação padrão para indicar Pesos e Bias

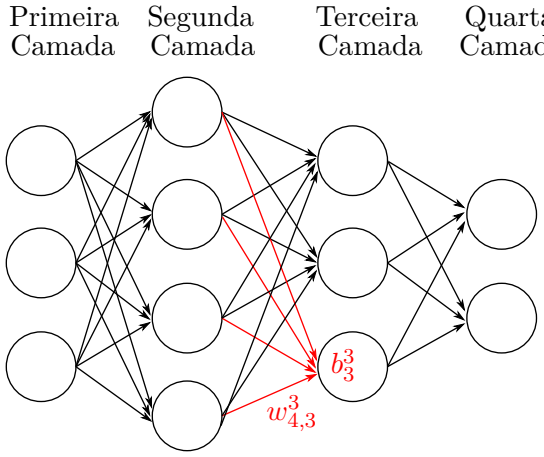


Fonte – Autor, 2018.

si, e ainda encapsulado pela função de ativação σ escolhida (veja a Equação 3.2 e sua representação visual na Figura 6).

$$o_j^l = \sigma\left(\sum_k w_{k,j}^l o_k^{l-1} + b_j^l\right) \tag{3.2}$$

Figura 6 – Fatores que influenciam no cálculo do Output do terceiro neurônio da terceira camada



Fonte – Autor, 2018.

Para simplificar a notação, iremos denominar de z_j^l a Soma Ponderada das Entradas do j-ésimo neurônio da l-ésima camada como sendo $z_j^l = \sum_k w_{k,j}^l o_k^{l-1} + b_j^l$, exatamente o argumento da função de ativação na Equação 3.2. Portanto, podemos reescrever a saída de um neurônio como sendo $o_j^l = \sigma\left(z_j^l\right)$.

E por fim, definiremos ainda o valor δ_j^l, representando o quanto o erro total varia quando o z_j^l varia, indicando o quanto cada neurônio está contribuindo para o erro total da rede neural. Formalmente, teremos a Equação 3.3:

$$\delta_j^l = \frac{\partial r}{\partial z_j^l} \tag{3.3}$$

Entretanto, descobrir qual exatamente é esse valor não é tão óbvio, e o objetivo do algoritmo de Retropropagação é calcular todos os δ_j^l e o_j^l, e a partir deles definir as derivadas parciais $\frac{\partial r}{\partial w_{j,k}^l}$ e $\frac{\partial r}{\partial b_j^l}$, tendo assim definido o gradiente.

Mais uma vez com objetivo de simplificar a notação, utilizaremos o^l e δ^l como os vetores $o^l = (o_1^l, o_2^l, ..., o_J^l)$ e $\delta^l = (\delta_1^l, \delta_2^l, ..., \delta_J^l)$, representando assim os vetores J-dimensionais de saídas e de erros da l-ésima camada, onde J é a quantidade de neurônios da l-ésima camada, valor que pode variar de camada a camada.

Tendo essa notação em mente, observamos que a função f, que nossa rede neural está buscando definir, pode ser vista como a saída da última camada da rede, ou seja, o^L, e daí temos $f(x) = o^L = \sigma(z^L)$. Agora, relembrando que nossa função de erro r é escrita sobre a função f, podemos reescrever r como uma função de o^L. Voltando ao exemplo anterior, onde usamos a metade da distância euclidiana como função de erro (Equação 3.1), teríamos na Equação 3.4 o erro calculado sobre um único exemplo:

$$r(o^L) = \frac{1}{2}|o^L - y|^2 \tag{3.4}$$

Lembramos ainda da definição de δ_j^l na Equação 3.3. A partir dela, aplicaremos a regra da cadeia, e teremos:

$$\delta_j^l = \frac{\partial r}{\partial z_j^l} = \sum_k \frac{\partial r}{\partial o_k^l} \frac{\partial o_k^l}{\partial z_j^l}$$

Veja agora que $\frac{\partial o_k^l}{\partial z_j^l}$ será nula em todas as parcelas, exceto quando $j = k$, pois a variação da Soma Ponderada do j-ésimo neurônio só irá afetar a saída do mesmo j-ésimo neurônio. Logo, reduzimos a expressão para:

$$\delta_j^l = \frac{\partial r}{\partial o_j^l}\frac{\partial o_j^l}{\partial z_j^l}$$

E como temos que, por definição, $o_j^l = \sigma(z_j^l)$, a segunda derivada parcial pode ser reduzida à $\sigma'(z_j^l)$, e então:

$$\delta_j^l = \frac{\partial r}{\partial o_j^l}\sigma'(z_j^l) \tag{3.5}$$

Tendo isso em mente, começaremos a descrever os passos do algoritmo, que serão ilustrados na Figura 7: O primeiro é o passo de ida, onde tendo escolhido um exemplo arbitrário $x \in \mathcal{X}$, seguimos camada a camada calculando as saídas de cada neurônio, até alcançarmos a última camada, que iremos chamar de camada L, e calcularmos o vetor o^L a partir do z^L.

O segundo passo é calcular δ^L, e temos todo o necessário para calculá-lo, já que tanto nossa Função de Ativação σ quanto a Função de Erro r são conhecidas e facilmente diferenciáveis, e o valor z^L acabou de ser calculado no fim do passo anterior. Basta calcular cada uma das coordenadas δ_j^L com a Equação 3.5.

O terceiro é o passo de volta, onde começando da última camada, e retornando até a primeira, iremos camada a camada calcular o seu δ^l. Para isso, veremos que podemos escrever δ^l em termos de δ^{l+1}. Partiremos, mais uma vez, da definição de δ_j^l (na Equação 3.3), mas agora as derivadas parciais serão referentes à z_k^{l+1}:

$$\begin{aligned}
\delta_j^l &= \frac{\partial r}{\partial z_j^l} \\
&= \sum_k \frac{\partial r}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} && (3.6)\\
&= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l}\delta_k^{l+1} && (3.7)
\end{aligned}$$

Observe que na Equação 3.6, $\frac{\partial r}{\partial z_k^{l+1}}$, coincide exatamente com a definição da erro do k-ésimo elemento da camada seguinte, e assim o substituímos por δ_k^{l+1} na Equação 3.7.

Resta calcularmos a parcial seguinte, e para que este processo fique claro, iremos expandir a expressão z_k^{l+1}:

$$z_k^{l+1} = \sum_j w_{k,j}^{l+1} o_j^l + b_j^{l+1} = \sum_j w_{k,j}^{l+1} \sigma(z_j^l) + b_j^{l+1}$$

Portanto, a relação entre z_k^{l+1} e z_j^l é clara, e podemos derivar para termos:

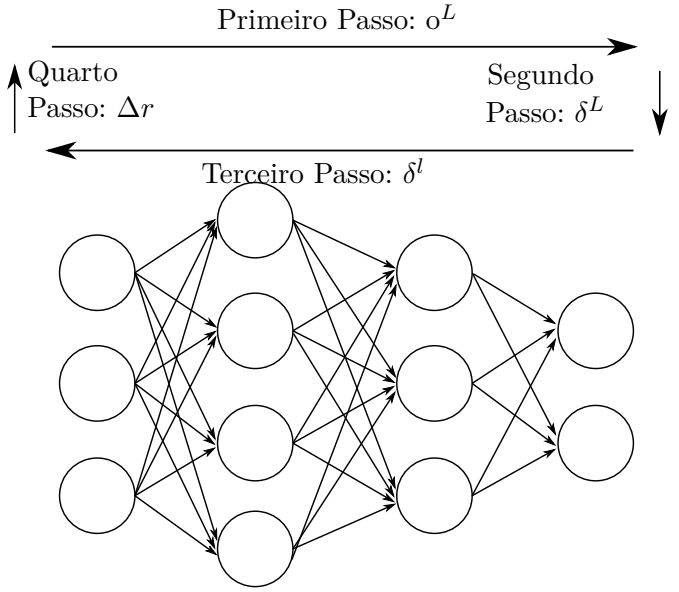
$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{k,j}^{l+1} \sigma'(z_j^l) \tag{3.8}$$

Portanto, retornando à Equação 3.7 e combinando com o resultado da 3.8:

$$\delta_j^l = \sum_k w_{k,j}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$$

Agora, tendo δ^L, podemos calcular δ^{L-1}, e então δ^{L-2}, e então todo o caminho pela rede até δ^1.

Figura 7 – Etapas do processo de Backpropagation: 1) Segue da primeira até a última camada calculando o^l; 2) usar o^L para calcular δ^L; 3) seguir da última até a primeira camada calculando δ^l; 4) usar cada δ^l para calcular ∇r



Fonte – Autor, 2018.

O último passo é utilizar os valores de δ_j^l para determinar o gradiente de nossa Função de Erro r. Isso é feito através das seguintes equações, que relacionam cada derivada parcial de r com os erros δ_j^l:

$$\frac{\partial r}{\partial w_{j,k}^{l}} = o_k^{l-1}\delta_j^l$$
$$\frac{\partial r}{\partial b_j^l} = \delta_j^l$$

Agora estamos de posse do gradiente da função de erro, podemos concluir a implementação do Gradiente Descendente: tomamos uma fração de seu oposto, e multiplicamos cada coeficiente da equação, seja ele peso ou bias, por este valor. Assim, definimos um novo conjunto de coeficientes, e damos início à etapa seguinte, repetindo o processo de Retropropagação para um novo exemplo, enquanto o erro total esteja fora do tolerado, ou alguma outra condição for determinada.

3.2.2 Redes Neurais Convolucionais (CNN)

Conforme mencionado anteriormente, uma aplicação muito interessante das redes neurais é na solução de problemas não-lineares; e um leque muito grande de problemas não-lineares envolve reconhecimento de linguagem natural por voz, e problemas relacionados à visão.

Mencionamos ainda que Redes Neurais alcançam ótimos resultados em problemas de reconhecimento e classificação de imagens, e agora iremos detalhar esse processo. Para tanto, precisamos ter claro em mente os conceitos da operação matemática de Convolução e dos Filtros Convolucionais usados no tratamento de imagens.

3.2.2.1 Convolução e Convolução Discreta

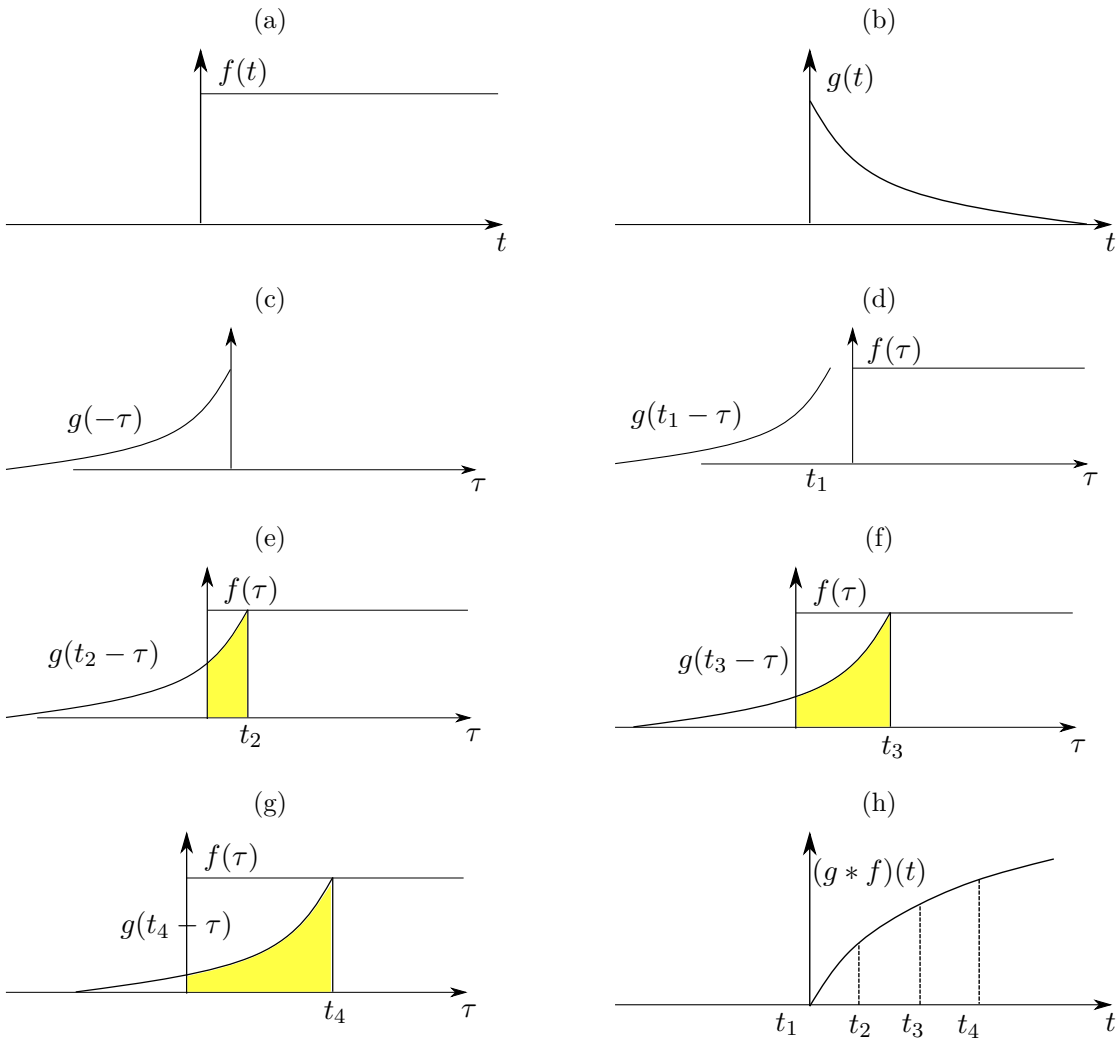
Chamamos de Convolução a operação comutativa entre duas funções f e g, representada pelo operador $*$, produzindo uma terceira função $f * g$, que representa a soma dos produtos entre as funções ao longo da região formada pela superposição delas. Este resultado, além da operação em si, também é denominado Convolução.

A operação é definida formalmente como a integral sobre o produto de uma das funções com uma cópia deslocada e espelhada da outra, podendo ser percebida como a média ponderada da primeira função, com os pesos dados pela segunda função, depois de deslocada e espelhada:

$$(f * g)(t) = \int_{-\infty}^{+\infty} f(\tau) \cdot g(t - \tau) d\tau$$
$$= \int_{-\infty}^{+\infty} g(\tau) \cdot f(t - \tau) d\tau = (g * f)(t)$$

Podemos enxergar a Convolução graficamente de forma bastante intuitiva, como segue ilustrado na Figura 8.

Figura 8 – Passo a passo da operação de convolução em funções contínuas: estabelecimento de f (em a) e g (em b), inversão de g (em c), e a translação de g sobre f, calculando em cada instante t_i a área abaixo das duas funções, aqui representada em amarelo (entre d e g), e por fim, os valores da área como um gráfico próprio (em h).



Fonte – Autor, 2018.

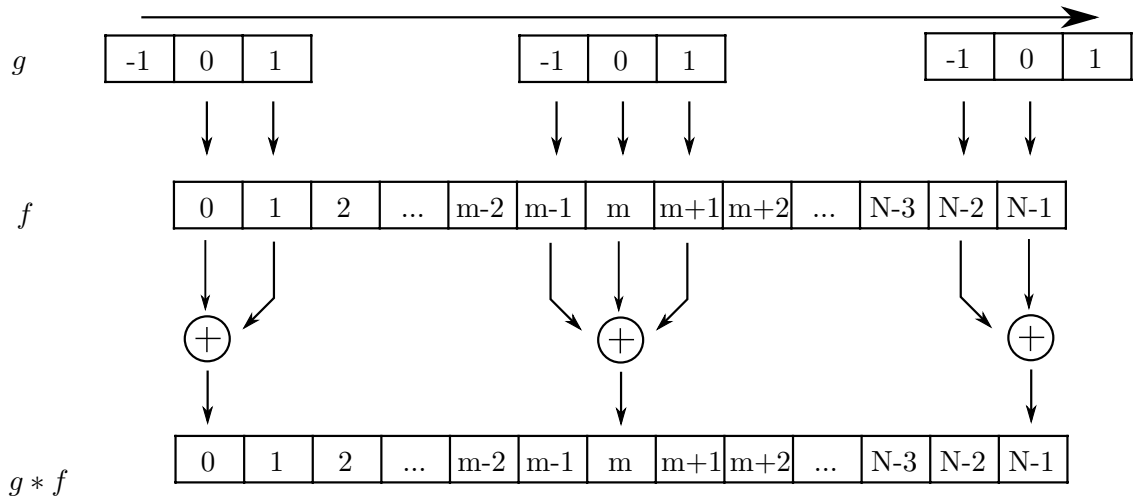
Essa operação apresenta uso em diversas áreas de pesquisa, como Probabilidade,

Estatística, Equações Diferenciais e Processamento de Sinais. Entretanto, para sua utilização num sistema computacional, precisamos antes convertê-la para o universo discreto, com a seguinte simplificação:

$$\begin{aligned}(f * g)(n) &= \sum_{j=-k}^{k} f(j) \cdot g(n-j) \\ &= \sum_{j=-k}^{k} g(j) \cdot f(n-j) = (g * f)(n)\end{aligned} \tag{3.9}$$

Cujo processo, mais uma vez análogo ao caso contínuo, também é bastante intuitivo, como visto na Figura 9.

Figura 9 – Descrição do funcionamento da operação de convolução discreta



Fonte – Autor, 2018.

3.2.2.2 Filtros Convolucionais

Para simplificar a notação, adicionaremos uma razoável condição a respeito da função g: a de que ela seja simétrica. Nesse caso teremos $g(j) = g(-j)$, e poderemos seguir o seguinte raciocínio, partindo da Equação 3.9:

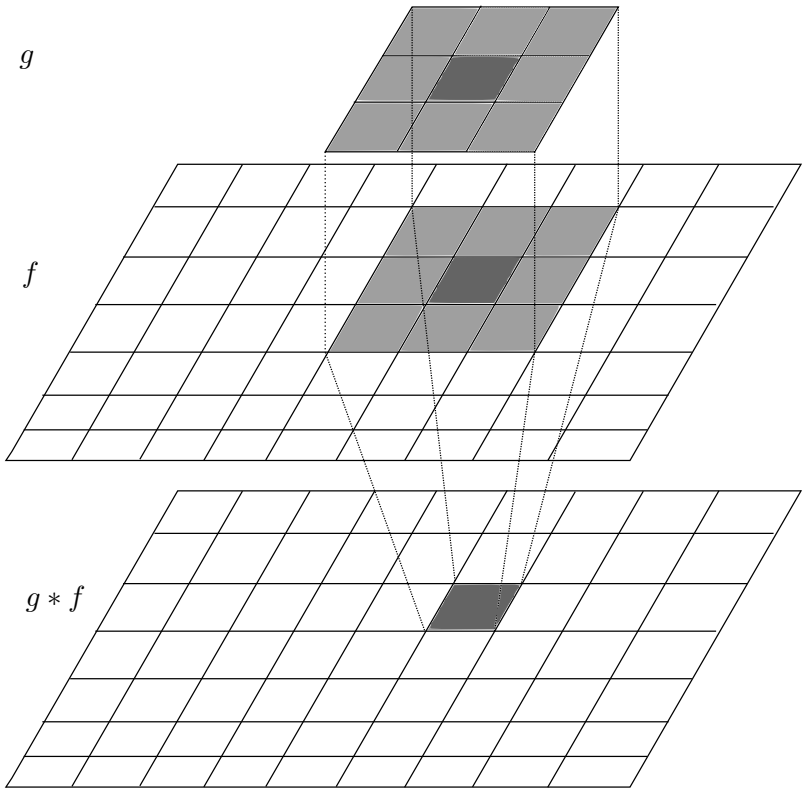
$$
\begin{aligned}
(g * f)(n) &= \sum_{j=-k}^{k} g(j) \cdot f(n-j) \\
&= \sum_{j=-k}^{k} g(-j) \cdot f(n+j) \\
&= \sum_{j=-k}^{k} g(j) \cdot f(n+j) \qquad (3.10)
\end{aligned}
$$

Nesse momento, nosso objetivo é aplicar a operação de Convolução sobre imagens, e para isso será necessário generalizá-la para um domínio bidimensional. Esse é um processo bastante natural a partir da Equação 3.10, como se segue:

$$
(g * f)(m,n) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} g(i,j) \cdot f(m+i, n+j) \qquad (3.11)
$$

E o funcionamento dessa operação é mais uma vez ilustrada na Figura 10.

Figura 10 – Descrição do funcionamento da Convolução discreta bidimensional, usada em imagens



Fonte – Autor, 2018.

A função g neste caso é conhecida como núcleo ou máscara da convolução (nos termos em inglês, *kernel* ou *mask*). Em geral, o núcleo pode ser representado por uma

pequena matriz quadrada, de dimensões ímpares, como 3×3 ou 5×5, e a aplicação de diferentes núcleos sobre uma imagem causa efeitos igualmente variados. Por exemplo, ruídos podem ser removidos com o filtro gaussiano (como visto na Figura 11b), enquanto outras máscaras reforçam os contornos (Figura 11c), ou ainda, removem da imagem tudo que não é um contorno (Figura 11d).

Figura 11 – Exemplo de diversos filtros aplicados sobre uma imagem

(a) Original

(b) Gaussiano

(c) Reforço de contorno

(d) Apenas contornos

Fonte – Autor, 2018.

3.2.2.3 Camadas Convolucionais

Mais uma vez com o objetivo de simplificar a explicação, vamos iniciar nossos exemplos com imagens de um único canal, como imagens em escala de cinza.

Cada pixel da imagem corresponde a um valor entre 0 e 255 (ou, quando normalizados, entre 0 e 1), representando a luminosidade daquele pixel. Em se tratando de redes neurais com o objetivo de processar essas imagens, a abordagem natural é tomar cada um desses valores como entrada de nossa rede neural.

Isto gera uma enorme quantidade de neurônios e conexões, e consequentemente uma grande quantidade de pesos, que acarreta em uma etapa de treinamento mais demorada e num processamento menos eficiente de forma geral. O primeiro desafio neste caso é reduzir a quantidade de dados sem descartar características importantes para o reconhecimento das imagens.

Com esse objetivo em mente, podemos iniciar o processo aplicando certos filtros sobre a imagem, com a ideia de diferenciar informações mais relevantes de informações menos relevantes, e poder descartar o segundo grupo.

Isso seria, a princípio, uma etapa de pré-processamento dos exemplos. A rigor, precisaríamos decidir quais filtros geram resultados interessantes, que facilitassem o processo de classificação de uma rede neural.

Entretanto, uma Camada Convolucional permite que a própria Rede Neural aprenda quais filtros devem ser aplicados sobre as imagens. Tudo que fazemos é determinar para a camada quantos filtros ela deve calcular, e qual o tamanho do núcleo de cada um desses filtros.

Assim, por exemplo, se configurarmos uma Camada Convolucional para encontrar 10 filtros, cada um com núcleo de dimensões 3×3, o processo de treinamento automaticamente vai encontrar 10 filtros diferentes que destaquem características úteis para conseguir classificar com sucesso as imagens. Se a Rede Neural observar que suavizar a imagem tem resultados positivos, um desses filtros provavelmente se aproximará de um filtro Gaussiano. Se a detecção de bordas for mais importante, provavelmente será usado um núcleo como o da Figura 11d. Mesmo núcleos que não sejam clássicos e que não sejam comumente usados nessas aplicações podem ser encontrados durante o processo de treinamento.

A implementação de filtros é uma rotina muito útil no reconhecimento de imagens, pois permite selecionar características que serão relevantes exatamente para o problema que está sendo abordado, enquanto demanda uma quantidade pequena de recursos (veja que cada filtro 3×3 implica em, inicialmente, apenas 9 pesos).

Por outro lado, se a camada convolucional processou 10 filtros, a saída desta camada será a imagem de entrada processada a partir de cada um desses filtros. Ou seja, se a entrada foi uma imagem em tons de cinza de dimensões 100×100, a saída será 10 dessas imagens, o equivalente a uma imagem 100×100 com 10 canais, ou ainda, uma matriz

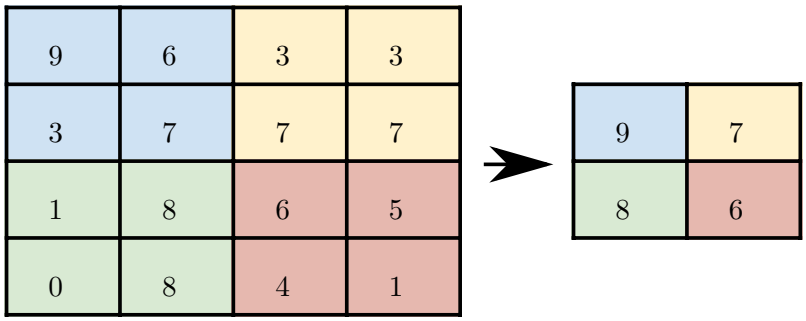
$100 \times 100 \times 10$.

Aumentamos a quantidade de informação disponível, já que cada uma dessas imagens resultantes de filtros carrega uma característica diferente, como luminosidade ou contornos, mas estamos com ainda mais dados para alimentar nossa rede, e isso gerará ainda mais conexões, e consequentemente, mais pesos.

Mesmo uma imagem pequena e simples, com dimensões 100×100 e em tons de cinza, geraria 10^4 neurônios apenas na primeira camada, e uma quantidade exponencialmente maior de pesos entre essa e a camada seguinte. E como vimos, o uso de filtros convolucionais amplia esse volume de dados ainda mais.

A solução desse problema surge através de camadas de agrupamento, cuja função é apenas reduzir o volume de dados. Dois parâmetros importantes precisam ser definidos nessa camada: o tamanho do núcleo, bastante semelhante ao núcleo dos filtros mencionados anteriormente. Um núcleo de dimensões 2×2 vai reduzir cada uma das dimensões das imagens à metade, enquanto um de tamanho 3×3 vai reduzir a um terço, por exemplo.

Figura 12 – Funcionamento de uma Camada de Agrupamento Máximo, reduzindo uma imagem de 16 pixels para uma de 4



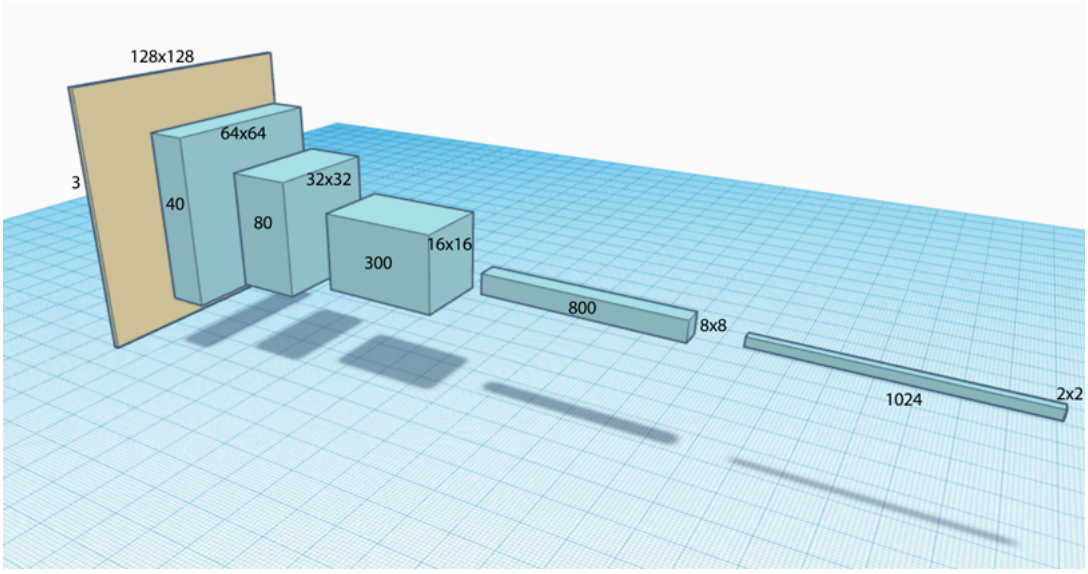
Fonte – Autor, 2018.

Outro parâmetro importante é qual função será usada dentro de cada núcleo. Opções comuns incluem funções de $min()$, $avg()$ e $max()$, sendo que essa última costuma apresentar melhores resultados e é mais frequentemente utilizada.

O processo de aplicação de filtros convolucionais seguido de camada de agrupamento pode ser compreendido da seguinte forma: Primeiro, os filtros separam características relevantes da imagem. Em seguida, a camada de agrupamento máximo vai dividir a imagem em setores, e selecionar a característica mais relevante daquele setor. Essa, e apenas essa, será passada para a camada seguinte, como ilustrado na Figura 12.

Na prática, uma entrada de tons de cinza com dimensões 100×100 é uma matriz $100 \times 100 \times 1$. Após a camada convolucional que descrevemos mais cedo, ela se tornaria uma matriz $100 \times 100 \times 10$, e após a camada de agrupamento, $50 \times 50 \times 10$, resultando em 25000 neurônios na camada de entrada. Normalmente, esse processo é repetido algumas vezes, com outra camada de convolução procurando características interessantes nos dados, e outra camada de agrupamento reduzindo o volume e passando adiante apenas dados relevantes. Como, de forma geral, reduzimos as dimensões espaciais enquanto aumentamos a dimensão de profundidade, a representação visual da rede toma a forma de uma pirâmide, como pode ser visto na Figura 13.

Figura 13 – Pirâmide Convolucional



Fonte – Hui, 2017.

Uma vez que o conjunto de dados esteja suficientemente pequeno, os dados passam por uma camada de Achatamento, que transforma uma imagem de, digamos, $6 \times 6 \times 10$ em um vetor de características de dimensão 360. Com uma quantidade relativamente pequena de processamento computacional, reduzimos nossa entrada inicial, que consistia em 10^4 dados, em mais de 95%, não só utilizando apenas os 5% mais relevantes da imagem, mas encontrando potencialmente novos dados que estavam ocultos na imagem original.

Tudo isso fez as Camadas Convolucionais alcançarem grande sucesso na classificação de imagens. Um bloco inicial composto de camadas de Convolução e Agrupamento Máximo, intercaladas, seguidas de uma camada de achatamento, que fornece um vetor de características para um bloco de camadas regulares, com neurônios completamente

conectados uns aos outros, é a estrutura básica das chamadas Redes Neurais Convolucionais (da sigla em inglês CNN).

3.2.3 Redes Neurais Recorrentes (RNN) e Long Short-Term Memory (LSTM)

Por muito tempo, uma grande limitação das Redes Neurais foi a necessidade de que os dados de entrada fossem uniformemente formatados, no sentido de terem a mesma dimensão.

Uma determinada rede poderia receber de entrada uma imagem de dimensões 40×40, ou um vetor consistindo de 14 ângulos; mas essa mesma rede não está preparada para uma imagem de 40×39, ou um vetor com 20 ângulos.

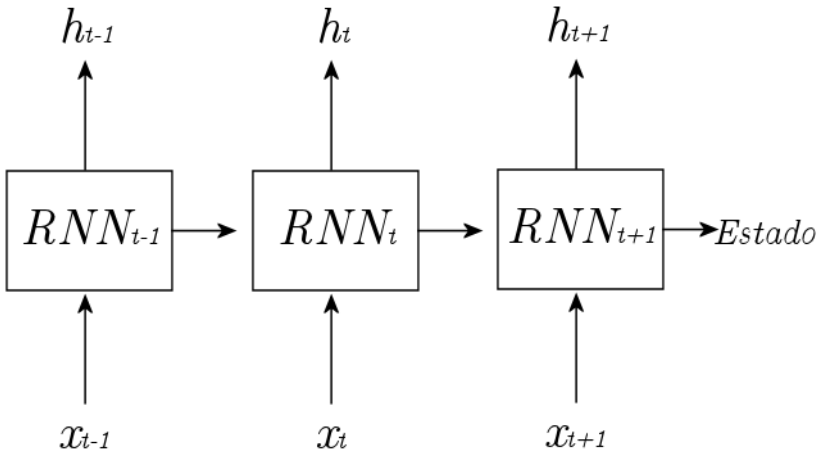
Essa limitação foi particularmente impactante quando foram iniciados testes de redes neurais em um tipo particular de entrada de dados: frases. Desde o início foram tomadas medidas para converter palavras em números, e então utilizá-las como sinais em uma rede neural. Mas a necessidade de que toda frase tivesse a mesma quantidade de palavras, ou então a necessidade de incluir palavras "vazias" ou de "preenchimento" prejudicava o desempenho das redes.

O conceito de recursividade surge para trabalhar essa problema. As palavras de uma frase são fornecidas para uma Camada Recursiva individualmente, uma após a outra. Essa camada, ilustrada na Figura 14, possui não apenas um, mas dois resultados de saída: Primeiro, a saída regular, que faz a predição normal, como se a palavra que acabou de ser processada fosse a última. E segundo, uma saída de Estado, que armazena e resume todas as palavras que já foram fornecidas nesta frase, e é fornecida diretamente para a próxima etapa da mesma camada.

O grande diferencial é que não apenas cada palavra nova fornecida pela frase é usada para calcular a saída regular; também é utilizada essa segunda saída, o Estado da camada. O estado, depois de ajudar a calcular a saída regular, é também atualizado levando em conta a palavra atual, e se prepara para ser repassado para a próxima etapa do processo recursivo.

Uma rede que empregue o uso de Camadas Recursivas é chamada de Rede Neural

Figura 14 – Estrutura Básica de uma Camada Recorrente



Fonte – Autor, 2018.

Recursiva, ou RNN, da sigla em inglês. Além de flexibilizar o tamanho das entradas, essa abordagem garante um certo nível de controle de contexto às frases, já que palavras podem possuir significados diferentes quando colocadas juntas com outras palavras, e isso é levado em conta neste processo.

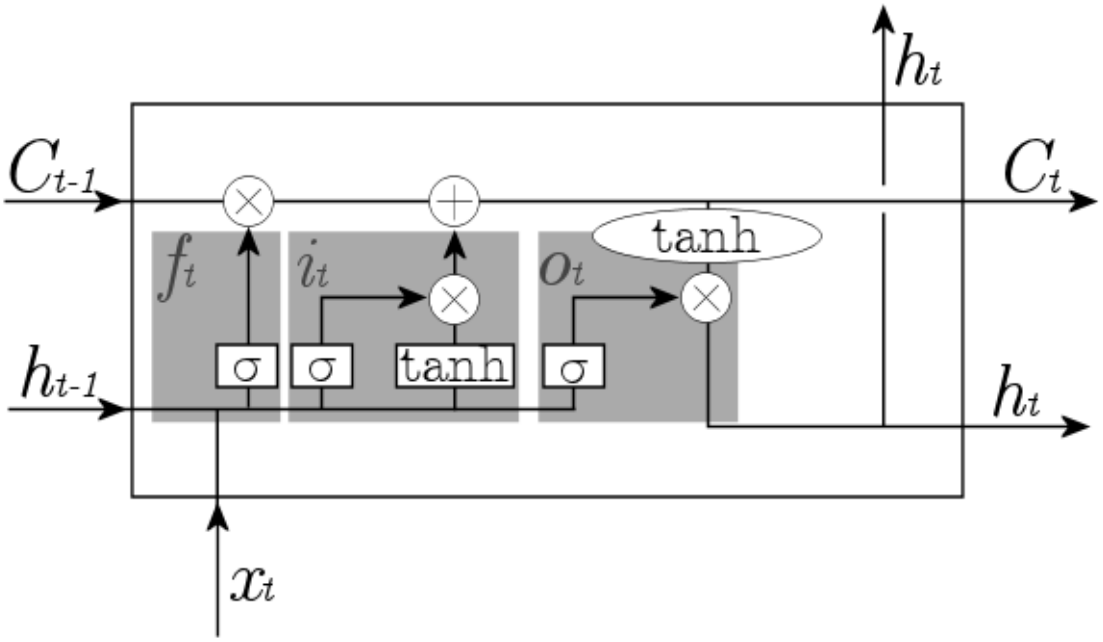
Entretanto, essa noção de contexto é ainda muito rudimentar, e testes com cadeias maiores de entradas, por exemplo, frases com grande quantidade de palavras, demonstraram que grande relevância era dada para as entradas mais recentes, e a cada etapa da recursão as entradas mais antigas tinham pouca influência no valor da saída de Estado.

Mas essa maior relevância é injustificada. Voltando ao exemplo de frases, encontramos muitas situações onde uma palavra de grande importância para a compreensão do contexto é dita logo no seu início. Mostrou-se necessário adaptar o processo para refletir esse tipo de situação, e é nesse sentido surge o conceito de memória.

Uma camada de Long Short-term Memory (ilustrada na Figura 15, e descrita pela primeira vez por [12]) possui não apenas uma saída de estado, mas duas, denominadas Estado Oculto, que é basicamente o Estado observado nas RNN; e o Estado da Célula, que aqui cumprirá a função de "memória" que acabamos de comentar. E tão importantes quanto os Estados são os três portões que controlam essa memória e a interação dela com a nova entrada: Portão do Esquecimento, da Entrada e da Saída.

A cada etapa do processo recursivo, a entrada atual é combinada com a saída da etapa anterior (que é o Estado Oculto, h_{t-1}), e essas duas informações juntas (note que

Figura 15 – Estrutura Básica de uma Célula LSTM



Fonte – Autor, 2018.

x_t e h_{t-1} são concatenados logo após a entrada na célula) definem, através do Portão do Esquecimento, quais informações do Estado da Célula são relevantes e quais já podem ser esquecidos.

Formalmente, o Portão do Esquecimento é representado por f_t (do inglês *forget gate*), e é uma camada simples de Rede Neural, com função de ativação sigmoide, definida pela Equação 3.12.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.12}$$

O resultado desse portão vai ser um grande vetor com valores entre 0 e 1, que vai ser multiplicado, elemento a elemento, com os valores do Estado da Célula que acabou de ser recebida da célula anterior, C_{t-1}, definindo quais deles são relevantes e devem continuar armazenados (que irão multiplicar valores próximos a 1) e quais são irrelevantes e devem ser esquecidos (que irão multiplicar valores próximos a 0).

Em seguida, através do Portão da Entrada, os dados da entrada são avaliados e sua própria relevância é determinada. Em um processo muito semelhante ao anterior, uma camada sigmoide vai definir quais valores do vetor concatenado $[h_{t-1}, x_t]$ devem ser preservadas e quais devem ser esquecidas. O resultado dessa camada sigmoide, outro vetor

onde os elementos variam entre 0 e 1, vai multiplicar o próprio vetor $[h_{t-1}, x_t]$ após ser processado por uma outra camada, com função de ativação customizável (uma opção comum é a tangente hiperbólica).

Esse portão é representado por i_t (do inglês *input gate*), e o resultado processado do vetor de entrada por $\tilde{C}_t$, e ambos são definidos como nas Equações 3.13 e 3.14.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.13}$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \tag{3.14}$$

Em seguida, caso relevante, essa informação é apropriadamente agregada ao Estado da Célula. Isso significa que i_t e $\tilde{C}_t$ são multiplicados elemento a elemento, e o resultado desse produto é somado ao vetor C_t. Esse vetor, definido dessa forma, na Equação 3.15, será repassado para a próxima célula.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{3.15}$$

Por fim, o Portão da Saída trata de avaliar o conteúdo do Estado da Célula e gerar uma Saída apropriada com ela. Isso é feito levando em conta tanto o Estado da Célula, quanto a Entrada atual, e ainda a Saída do estado anterior. A Saída atual é, então, repassada como Estado Oculto para a próxima etapa do processo.

Formalmente, temos o Portão de Saída representado por o_t (do inglês *output gate*), é outra camada sigmoide que irá usar a Entrada atual e o Estado Oculto anterior para definir quais dados do Estado da Célula serão usados na saída (Equação 3.16), e da mesma forma que as primeiras RNN que vimos anteriormente, o valor de saída é repassado como Estado Oculto para a próxima célula (Equação 3.17).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.16}$$

$$h_t = o_t * \tanh(C_t) \tag{3.17}$$

Como resultado desse processo, uma rede LSTM pode avaliar corretamente uma entrada como “A cor do céu, ao contrário da cor do deserto, é _____”, prevendo

adequadamente que nesse caso, o substantivo “céu” possui mais relevância que o substantivo “deserto”.

Apesar de predominantemente termos utilizado exemplos com frases e palavras, o conceito de rede LSTM pode ser aplicado sobre qualquer entrada que consista de uma sequência de elementos que precisam ser avaliados dentro do contexto dessa sequência, como por exemplo, cada uma das imagens que constituem um vídeo, como já abordado por Donahue et al.[16].

4. METODOLOGIA

Podemos organizar esse trabalho em três etapas distintas, cada uma bem representada por um módulo de grande importância.

Em primeiro lugar, a etapa de captura de exemplos, onde um módulo exibe a imagem capturada do usuário, e ele realiza um gesto específico, indica qual a classe do gesto que ele está fazendo, e informa o início e o final do gesto. Todos os frames capturados entre esses dois instantes são armazenados, tanto as imagens quanto as informações de esqueleto, e irão compor o banco de dados de treinamento $\mathcal{X}$.

Após a etapa de aquisição do conjunto de treinamento $\mathcal{X}$, inicia-se a etapa de treinamento da Rede Neural. Outro módulo localiza os exemplos, efetua pré-processamentos no mesmo, define a arquitetura da rede neural, e inicia o processo de treinamento em si, que irá definir os coeficientes da função f pelo algorítmo de retropropagação. Uma vez finalizado, armazena os coeficientes que definem f em um arquivo local, tornando a rede neural já treinada pronta para uso.

Por fim, outro módulo lê o arquivo salvo na etapa anterior e carrega a função f já devidamente treinada. Em seguida, com uma interface semelhante à do primeiro módulo, que exibe na tela em tempo real as imagens capturadas dos gestos que o usuário está fazendo nesse momento, o sistema se prepara para predizer os gestos do usuário de duas formas diferentes.

Na primeira, denominada Reconhecimento *Offline*, o usuário informa os tempos de início e fim do gesto que está efetuando. O módulo seleciona os frames capturados entre estes dois pontos, e repassa à Rede Neural, que vai usar os pesos treinados anteriormente para realizar uma predição, e o resultado é exibido ao usuário.

Na segunda, denominada Reconhecimento *Online*, o usuário apenas realiza gesto após gesto, e o módulo se encarrega de reconhecer qual está sendo realizado.

4.1 Aquisição e Pré-processamento dos Dados

Os dados que serviram de entrada para as redes neurais neste trabalho, tanto as imagens quanto as informações do esqueleto, foram capturados com o dispositivo Leap Motion. As imagens são simples projeções da palma da mão do usuário, em tons de cinza, e em teoria poderiam ser capturadas com outros sensores, desde que fossem pré-processadas de forma semelhante.

Entretanto, além do óbvio uso dos dados do esqueleto em mais da metade de nossas arquiteturas, também usamos dados do esqueleto, fornecidos pelo *Leap Motion* durante o pré-processamento das imagens, o que dificulta o uso de outros sensores. Em nossos experimentos, avaliamos o uso de imagens, esqueletos, e imagens junto com esqueletos.

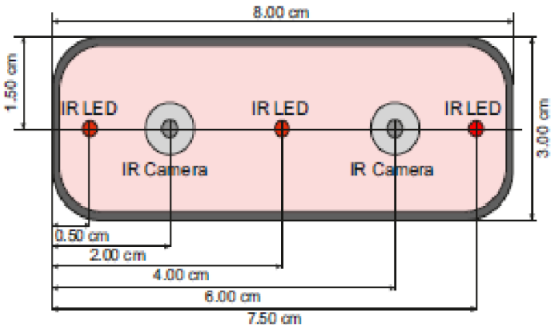
4.1.1 Leap Motion

O *Leap Motion* é um sensor óptico de movimento, otimizado para rastrear objetos como mãos e pontas de dedos ou de canetas. Seu Hardware consiste em um periférico de oito centímetros de comprimento e três centímetros de largura, que conecta-se ao computador por cabo USB, e pode ser utilizado de dois modos diferentes: em repouso, deitado sobre a mesa, com os sensores voltados para cima, onde as mãos são capturadas por baixo; ou acoplado a um óculos de Realidade Virtual, com o sensor acompanhando o campo de visão do usuário, onde as mãos são capturadas por trás.

O dispositivo consiste em três emissores de luz infra-vermelha, distribuídos uniformemente ao longo do seu comprimento, pelo que é definido como Eixo X pela sua API (*Application Programming Interface*). Entre os primeiro e segundo, e segundo e terceiro emissor, existem duas câmeras infra-vermelha, responsáveis pela captação das imagens, como ilustrado na Figura 16.

Assim, o Leap Motion se caracteriza como um sensor de Visão Computacional Estéreo, capturando duas imagens a partir de posições levemente diferentes (aqui, separadas por quatro centímetros de distância) e apontadas em direções paralelas, e utilizando ambas para calcular as posições tridimensionais dos objetos retratados. Em particular, o dispositivo busca por mãos e dedos, e o cálculo usado para determinar a posição dos objetos identificados não é publicamente divulgado.

Figura 16 – Visão esquematizada do Leap Motion, descrevendo suas dimensões e o posicionamento dos sensores de Infra-Vermelho



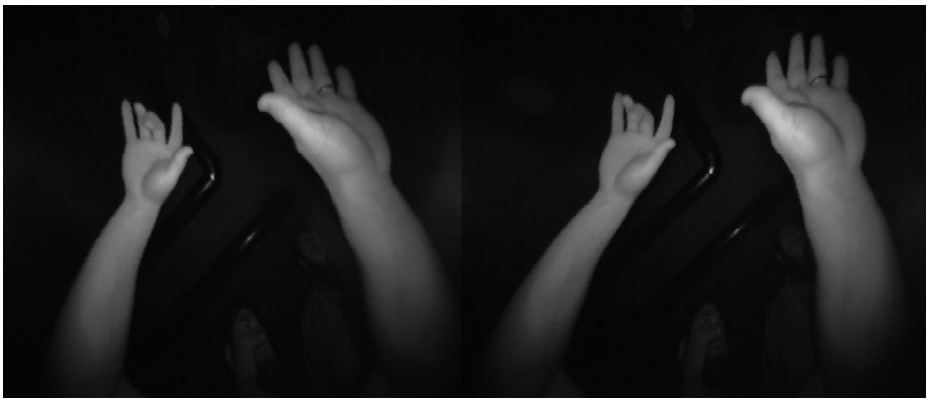
Fonte – Vani e Reddy, 2015.

O dispositivo tem um alcance espacial de 25mm a 600mm de altura, uma amplitude de 150°, e é capaz de gerar até 200 quadros por segundo. Cada quadro (por vezes chamado pelo termo em inglês *frame*) é composto de dois tipos de informação: visual, com as imagens capturadas pelas duas câmeras de infra-vermelho; e espacial, contendo detalhes do posicionamento das mãos rastreadas, calculadas com base nas imagens.

4.1.1.1 Informação Visual: Câmeras de infra-vermelho

Cada uma das duas câmeras de infra-vermelho oferecem uma imagem a cada quadro, e devido à natureza estéreo do dispositivo, elas são virtualmente idênticas, se diferenciando entre si apenas pela pequena mudança de posicionamento de suas câmeras.

Figura 17 – Uma mesma cena capturada por cada uma das duas câmeras de Infra-vermelho do Leap Motion



Fonte – Colgan, 2014.

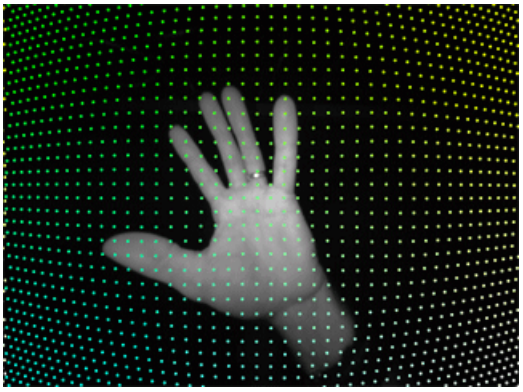
A captura é organizada em uma matriz onde cada posição representa um pixel de uma imagem, e seu valor corresponde à luminosidade medida pelo sensor naquele ponto.

Podemos então visualizar essa matriz como uma imagem em tons de cinza, onde os pixels mais claros representam pontos mais luminosos, como a Figura 17.

Portanto, na imagem final, pontos mais próximos aos sensores refletem a luz infravermelha com mais intensidade, e portanto são mais claros. Vemos ainda que pontos emissores de luz, como lâmpadas, também são representados por pixels claros. Mas de forma geral, pontos mais distantes são representados por pixels mais escuros.

Ainda, devido à natureza da lente das câmeras do dispositivo, ambas imagens são gravadas com uma pequena distorção. Enquanto a API não disponibiliza uma correção automática da imagem, nós fazemos tais correções a partir de informações oferecidas pela API, calculando o ângulo de incidência de cada raio de luz, ou seja, de cada pixel na imagem, como na Figura 18.

Figura 18 – Grade de distorção sobre imagem capturada pela câmera, indicando a significante distorção durante a captura



Fonte – Motion, 2017.

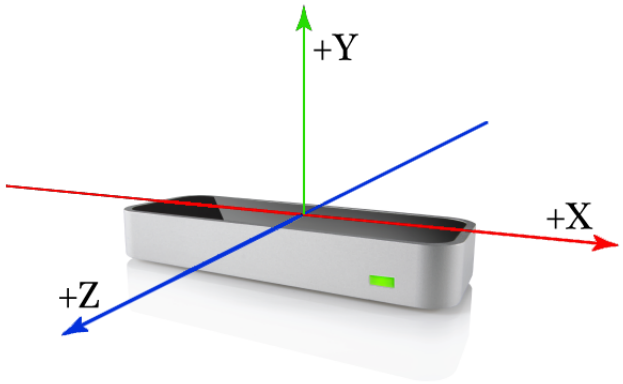
Por outro lado, a orientação da imagem é corrigida automaticamente, fazendo com que seja irrelevante a orientação horizontal do dispositivo. Na prática, temos que as mãos são sempre representadas saindo da parte inferior da imagem, mesmo que o aparelho esteja de “cabeça para baixo”.

4.1.1.2 Informação Espacial: Esqueleto

As imagens estéreo do dispositivo são apenas o recurso utilizado para calcular as informações espaciais do esqueleto das mãos detectadas no campo de visão do aparelho. Portanto, estes são os principais dados fornecidos pelo sensor.

A princípio, os eixos que formam o espaço tridimensional gerado pelo dispositivo são distribuídos da seguinte forma: O Eixo X, como já mencionado, é aquele onde os sensores se dispõe linearmente. Tal como a reta dos números reais, esse eixo é crescente à direita. O Eixo Y é ortogonal ao Eixo X, e perpendicular à base do aparelho, apontando diretamente para cima, e sendo crescente nessa direção. Por fim, o último é o Eixo Z, ortogonal aos dois e crescente na direção do usuário.

Figura 19 – Posição dos eixos coordenados sobre o Leap Motion



Fonte – Motion, 2017.

Como já mencionado, uma vez que mãos forem detectadas no campo de visão do aparelho, ele irá, se necessário, girar o espaço para que o Eixo Z sempre seja crescente em direção ao usuário. Caso a posição do aparelho ou do usuário seja alterada ao longo do uso, as coordenadas são prontamente atualizadas para refletir isso.

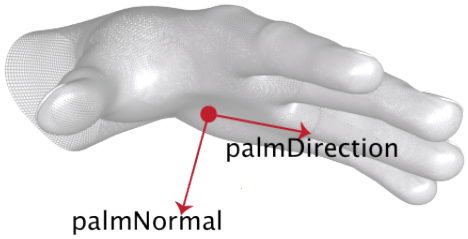
A partir disso, uma série de informações poderá ser obtida a partir da API do dispositivo. A cada quadro, podemos acessar listas de mãos e dedos rastreados, respectivamente, nas classes *Hand* e *Finger*.

Cada objeto da classe *Hand* possui atributos interessantes, dentre as quais podemos mencionar:

- Direção: um vetor tridimensional unitário partindo da palma da mão em direção aos dedos (Figura 20);
- Lista de dedos: do polegar ao dedo mínimo, todos os dedos relacionados a esta mão;
- Esquerda/Direita: informa a posição da mão em relação ao corpo;
- Posição da palma: posição espacial do centro da palma, em milímetros;

- Normal da palma: vetor tridimensional unitário partindo da palma da mão na direção de onde a palma está virada (Figura 20);
- Tamanho da palma: estimativa em milímetros do tamanho da palma, quando aberta.

Figura 20 – Posição dos vetores de Direção e Normal da palma da mão

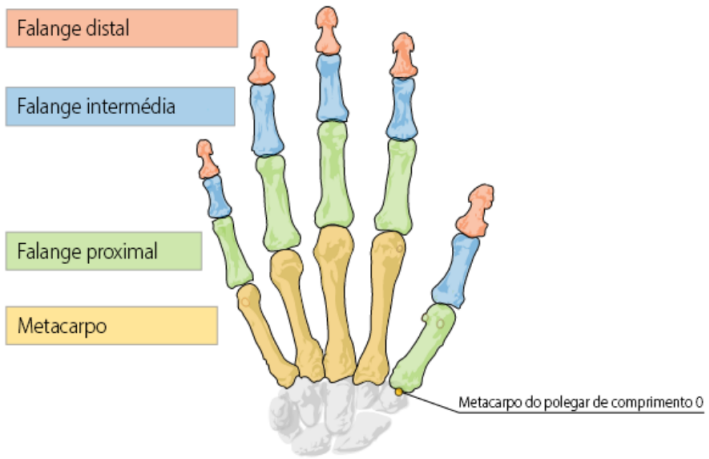


Fonte – Motion, 2017.

Da mesma forma, cada objeto da classe *Finger* possui atributos como:

- Lista de ossos: do metacarpo à falange distal (como mostra a Figura 21), todos os ossos relacionados a este dedo, e suas posições em coordenadas cartesianas;
- Esticado: informa se o dedo está ou não esticado;
- Tamanho: estimativa em milímetros do tamanho do dedo;
- Posição da ponta: posição espacial da extremidade do dedo, em milímetros.

Figura 21 – Esqueleto de uma mão humana, com a identificação de cada osso.



Fonte – Motion, 2017.

4.1.2 Pré-processamentos

O pré-processamento dos dados vindos do *Leap Motion* podem ser organizados em quatro etapas distintas; três delas correspondendo ao processamento de dados da imagem, e a última aos dados do esqueleto:

4.1.2.1 Pré-processamentos na Imagem

4.1.2.1.1 Correção da distorção da lente

Como mencionamos anteriormente, a natureza das lentes das câmeras causa uma pequena distorção na imagem recebida, e a API do *Leap Motion* nos oferece um Mapa de Distorção para fazermos as correções manuais.

Iniciamos o processo tendo em mãos a imagem original, de dimensões 640×240, e uma imagem em branco, com as dimensões desejadas de 400×400. Então criamos uma correspondência direta entre alguns pixels da imagem em branco com as posições do Mapa de Distorção, que é uma matriz de dimensão 64×64, fornecida juntamente com a imagem original.

Cada uma das 4096 posições do Mapa de Distorção contém uma coordenada na imagem original, e é dessa coordenada que iremos buscar o brilho correspondente para preencher a posição atual da imagem nova.

Para todos os pixels da imagem nova que não foram diretamente cobertos pela grade do Mapa de Distorção, não temos nenhuma informação sobre seus valores. Nesse caso, nossa melhor opção é aproximá-lo através de uma interpolação bilinear, utilizando os quatro pontos definidos mais próximos. A Figura 22 ilustra o processo de definir o valor no ponto $P = (x, y)$, conhecendo a luminosidade L nos pontos $X1 = (x_1, y_2)$, $X2 = (x_2, y_2)$, $X3 = (x_1, y_1)$ e $X4 = (x_2, y_1)$.

Inicialmente, definimos o valor da luminosidade L nos pontos (x, y_1) e (x, y_2) usando interpolações lineares simples entre os pontos $X1$ e $X3$, e depois entre os pontos $X2$ e $X4$ (pois estes pontos compartilham a mesma primeira coordenada), da forma como mostram as equações 4.1 e 4.2.

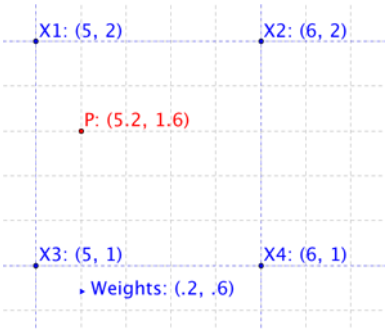
$$L(x, y_1) = \frac{x_2 - x}{x_2 - x_1} L(x_1, y_1) + \frac{x - x_1}{x_2 - x_1} L(x_1, y_2) \tag{4.1}$$

$$L(x, y_2) = \frac{x_2 - x}{x_2 - x_1} L(x_2, y_1) + \frac{x - x_1}{x_2 - x_1} L(x_2, y_2) \tag{4.2}$$

Uma vez que os valores dos pontos (x, y_1) e (x, y_2) estejam definidos, seguimos com outra interpolação linear simples para computar o ponto (x, y), nosso objetivo final, como formaliza a equação 4.3.

$$L(x, y) = \frac{y_2 - y}{y_2 - y_1} L(x, y_1) + \frac{y - y_1}{y_2 - y_1} L(x, y_2) \tag{4.3}$$

Figura 22 – Processo de interpolação bilinear. O valor do ponto P será calculado com base nos pontos $X1$, $X2$, $X3$ e $X4$



Fonte – Motion, 2017.

4.1.2.1.2 Recorte da região da imagem correspondente à mão

O próximo passo é reconhecer qual a posição da palma da mão na imagem capturada e recortar apenas a região imediatamente ao redor da mão. Para isso, utilizamos informações do esqueleto fornecidas pelo *Leap Motion*, em particular o ponto correspondente ao centro da palma da mão.

A API do *Leap Motion* fornece um ponto tridimensional com as coordenadas cartesianas do centro da palma, e então é necessário projetar essa posição sobre a imagem bidimensional que temos. Para isso, mais uma vez usamos uma função fornecida pela API, cuja entrada é um ponto tridimensional, e saída é a coordenada do pixel correspondente à sua projeção na imagem.

Naturalmente, essa coordenada corresponde à imagem original, ainda com a distorção de lente que processamos anteriormente, e portanto, também precisam ser retificadas para corresponder à imagem corrigida.

Mas observe que a abordagem tomada no problema anterior não nos permite partir de um pixel na imagem original e encontrar seu correspondente na imagem corrigida: todo o processo foi feito tomando cada um dos pixels da segunda, e então encontrando seu correspondente na primeira.

Como essa nova situação diz respeito a uma quantidade bastante reduzida de pixels, no sentido de que precisamos remapear apenas um pixel, correspondendo ao centro da palma, ao invés de todos os 400×400 pixels da imagem completa, optamos por implementar diretamente uma interpolação bilinear, ao invés de utilizarmos uma implementação já otimizada do algoritmo, como feito anteriormente.

Assim, no primeiro quadro recebido pelo nosso sistema a partir do *Leap Motion*, calculamos e armazenamos um Mapa de Distorção separado, e ao longo dos frames seguintes, sempre que precisarmos remapear qualquer pixel da imagem original para a imagem corrigida, poderemos utilizar esse mapa para encontrar as coordenadas corretas. Com isso, não apenas o centro da palma, mas outros pontos interessantes, como as pontas dos dedos, podem ser localizados de forma eficiente.

Para representar o recorte da palma da mão, por exemplo, em uma abordagem inicial optamos por capturar os pontos correspondentes ao centro da palma, à posição do punho, e às pontas dos cinco dedos, e com eles formar um fecho convexo. Entretanto, isso se provou ineficiente em algumas situações, como quando a mão está fechada, com as cinco pontas dos dedos muito próximos ao centro da palma.

Nesse caso, o fecho convexo deixa de fora parte dos dedos, e seria necessário um preenchimento arbitrário para aumentar o tamanho do fecho, e englobar com totalidade a mão do usuário. Entretanto, esse mesmo preenchimento arbitrário gera muito espaço vazio caso o usuário esteja com a mão completamente aberta.

Portanto, optamos por definir um raio, também em pixels, e a partir apenas do centro da palma, definir uma área quadrada circunscrita ao círculo formado por tal raio, que fosse capaz de englobar a mão com totalidade. É fácil observar que, quanto mais próxima do sensor, ou seja, mais baixa, maior a mão é projetada na imagem, e quanto

mais distante do sensor, ou seja, mais alta, menor ela é projetada, como o diagrama na Figura 23 representa, e como é uma consequência esperada da projeção perspectiva do *Leap Motion*.

Em particular as Figuras 23b e 23c mostram que, nesse modelo, o mesmo objeto, dependendo de sua distância D (maior em 23b, e menor em 23c) em relação à câmera, possui representações de tamanhos r diferentes. Entretanto, como é o mesmo objeto, seu tamanho real R não é alterado, e da mesma forma, a distância d entre a câmera e a projeção também se mantém a mesma.

Agora, se tomarmos dois triângulos retângulos, um formado pelos catetos D e $R/2$, e outro pelos catetos d e $r/2$, veremos que eles são semelhantes entre si, e então temos:

$$\frac{r}{d/2} = \frac{R}{D/2}$$

Agruparemos os coeficientes d e R, por serem constantes:

$$r = \frac{R \cdot d/2}{D/2}$$

E então podemos simplificar os fatores em comum:

$$r = \frac{R \cdot d}{D}$$

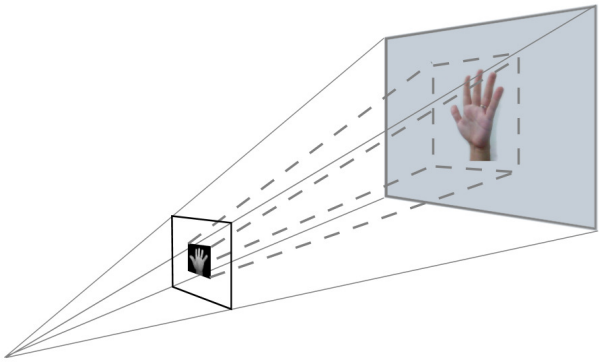
Veja que, neste modelo, D corresponde exatamente à distância entre a palma da mão à câmera, dado que é fornecido pelo *Leap Motion* em precisão milimétrica, r corresponde ao tamanho do raio suficientemente grande para englobar toda a projeção da mão na imagem, em pixels, e $R \cdot d$ é um valor constante que pode ser determinado empiricamente.

Assim, modelamos o raio na imagem como uma função racional da distância da mão à câmera, satisfazendo nossa observação inicial, de que raio e distância eram inversamente proporcionais. Posteriormente o coeficiente $R \cdot d$ foi definido de forma empírica com o valor de 6250.

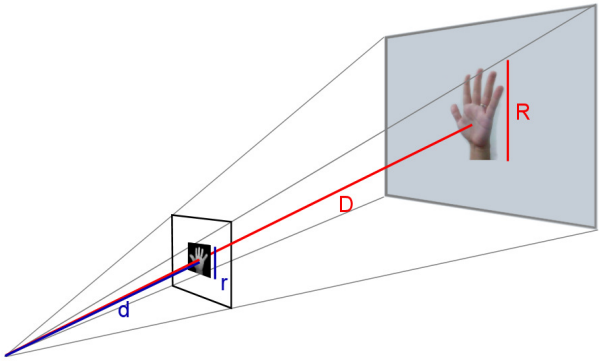
Tendo o valor do raio, e usando o centro da palma como ponto central, é simples criarmos uma caixa capaz de envolver completamente a mão. E sabendo que posterior-

Figura 23 – Modelo de Câmera Virtual projetando uma mão real sobre a imagem gerada e o comportamento das variáveis envolvidas de acordo com a posição da mão em relação à câmera

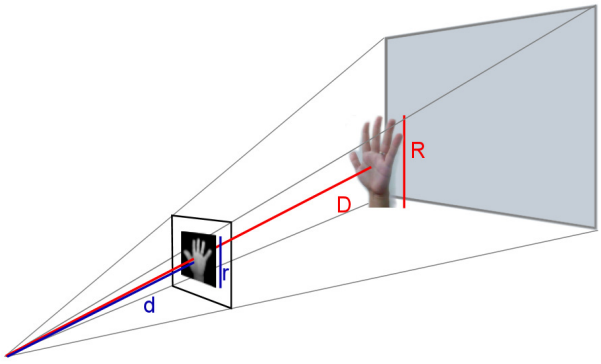
(a) Modelo de Câmera Virtual projetando uma mão real sobre a imagem gerada



(b) Indicadores das variáveis envolvidas quando a mão está distante da câmera



(c) Indicadores das variáveis envolvidas quando a mão está próxima da câmera



Fonte – Autor, 2018.

mente a uniformidade das dimensões dos exemplos será importante, desde já tratamos de redimensionar essa caixa para um tamanho pré-definido de 40×40, utilizando interpolação *spline*.

4.1.2.1.3 Limpeza de ruídos da imagem

Em seguida, para reduzir os ruídos da imagem, a ideia natural seria binarizá-la. Mas acreditando que alguns detalhes no espectro superior do brilho (entre 150 e 250, por exemplo) poderiam ser relevantes para o reconhecimento da pose, optamos apenas por filtrar pixels com baixo brilho (definindo como 0 todo pixel com valor abaixo de 50).

Isso é o suficiente para remover detalhes indesejados, como manchas correspondentes ao rosto ou corpo do usuário, que eventualmente caem dentro do recorte da palma da mão, enquanto mantém o máximo de detalhes possíveis da mão em si, que está bem próxima do sensor e possui níveis de brilho bem superiores a 50.

Com isso, temos uma imagem em tons de cinza, de 40×40, limpa de ruídos, contendo completamente a mão do usuário, e com a mão do usuário ocupando completamente a imagem. Esse é o exemplo ideal para nossas redes neurais.

4.1.2.2 Pré-processamentos no Esqueleto

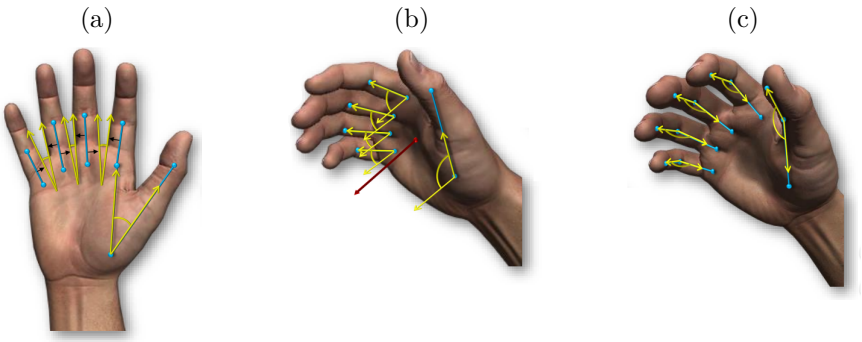
Agora, juntamente com cada imagem salva, iremos guardar também um arquivo de texto contendo informações úteis acerca do esqueleto. Os dados que nós buscamos não são fornecidos diretamente pela API do *Leap Motion*, então precisaremos calculá-los a partir do que ele nos oferece.

A principal informação que usaremos para diferenciar os gestos são os ângulos entre os ossos da mão, por serem invariantes por translação e conseguirem representar de forma compacta qualquer pose das quais estamos interessados em classificar.

Utilizaremos os mesmos ângulos que já foram propostos por Silva[8]:

- 4 ângulos entre as falanges proximais, como descrito na Figura 24a
- 5 ângulos entre a falange proximal e o vetor normal da palma, como na Figura 24b
- 5 ângulos entre cada falange intermediária e falange proximal, como na Figura 24c

Figura 24 – Descrição visual dos ângulos usados como descritores



Fonte – Silva, 2017.

Em adição a estes ângulos, também armazenaremos dois importantes vetores de direção, indicando os sentidos da palma e dos dedos da mão, como vistos na Figura 20

Por fim, outros dois vetores informarão a posição e a velocidade da mão. Apesar da inclusão da posição tridimensional do centro da palma da mão tornar os exemplos variantes por translação, essa informação é importante para alguns gestos em particular, e portanto optamos por incluí-la neste momento. O vetor velocidade é calculado de forma simples, apenas com a diferença entre a posição atual do centro da palma e a posição capturada no frame imediatamente anterior.

Dessa forma, temos um total de 14 ângulos e quatro vetores tridimensionais, totalizando 26 números reais, que são gravados como um arquivo de texto, de forma pareada com cada uma das imagens.

4.2 Arquiteturas adotadas

A partir de agora, detalharemos quais hiperparâmetros foram utilizados em cada uma das etapas do desenvolvimento deste trabalho. A maioria deles é comum a todas as etapas, e algumas partes da arquitetura foram propositalmente reutilizadas em etapas seguintes, mas ainda assim, diferenças relevantes poderão ser observadas ao longo delas.

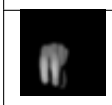
Mantivemos constantes a taxa de aprendizagem de $5 \cdot 10^{-4}$. Todas as camadas foram configuradas com a função de ativação $tanh()$, exceto a última camada densa de cada rede, onde foi usada a $softmax$ para assegurar a saída como um vetor de probabilidades consistente. Por fim, o método otimizador do gradiente descendente escolhido foi o RMSprop

[25], e a função de erro a de Categorical Cross-entropy [20].

4.2.1 Reconhecimento de Poses

A mais simples das etapas foi a de reconhecimento de poses estáticas. Para esse objetivo, desenvolvemos uma rede neural convolucional para classificar os sinais do alfabeto da Língua Brasileiras de Sinais (Libras) que sejam estáticos, ou seja, dependam apenas do correto posicionamento dos dedos para serem reconhecidos, como mostra a Tabela 1.

Tabela 1 – Poses de Libras que foram trabalhadas na etapa de poses estáticas

A	B	C	D	E	F	G	I	L	M
									
N	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	
									

Fonte – Autor, 2018.

Para isso, iniciamos a rede com um bloco de camadas convolucionais, que aplicarão filtros sobre as imagens de entrada, buscando ressaltar características interessantes e que possibilitem uma distinção entre os exemplos fornecidos.

Neste caso em particular, foram utilizadas três camadas convolucionais, todas aplicando filtros de dimensões 3×3, e cada uma seguida de uma camada de agrupamento máximo, responsável por reduzir as dimensões da imagem pela metade, simplificando os dados e mantendo apenas as características mais relevantes.

As duas primeiras camadas aplicam cada uma 16 filtros, enquanto a última aplica 32. Dessa forma, as dimensões dos dados de entrada vão se transformando ao longo das camadas, como mostra a Figura 25: Iniciando com uma imagem monocanal de dimensões 40×40, após a primeira camada convolucional ela se torna $38 \times 38 \times 16$, e após ser agrupada, $19 \times 19 \times 16$. A segunda camada de filtros reduz as dimensões para $17 \times 17 \times 16$, e a segunda camada de agrupamento para $8 \times 8 \times 16$. Por fim, os últimos filtros são aplicados, e os dados se transformam em $6 \times 6 \times 32$, que por fim são reduzidos pela última camada de agrupamento para $3 \times 3 \times 32$.

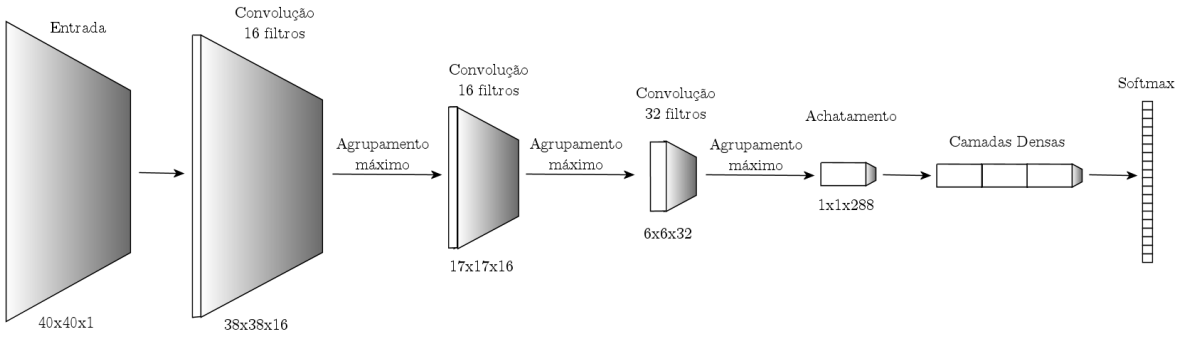
Concluindo este bloco, aplicamos uma camada de achatamento, que irá transformar esses dados em um único vetor de características, com dimensão $3 \cdot 3 \cdot 32 = 288$.

Esse vetor irá então servir de entrada para o segundo bloco da rede neural, formado por camadas densas, com cada neurônio completamente interligado com os das camadas anterior e posteriores.

Mais uma vez, esse bloco é composto por três camadas, com respectivamente 400, 200 e 200 neurônios; e cada uma seguida de uma camada de *Dropout*, configurada para abandonar aleatoriamente 15% dos pesos da camada anterior, com o objetivo de evitar *overfitting*[26].

Por fim, aplicamos uma última camada densa, com função de ativação $softmax$ e quantidade de neurônios equivalente à quantidade de classes sendo avaliadas (neste caso, 19).

Figura 25 – Representação visual da estrutura da rede utilizada para classificação de poses



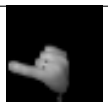
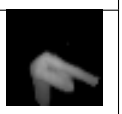
Fonte – Autor, 2018.

4.2.2 Reconhecimento de Gestos Offline

Em seguida, partimos para o reconhecimento individual de gestos dinâmicos. Agora, estruturaremos uma rede neural capaz de classificar os sinais do alfabeto de Libras que incluam não apenas a posição adequada dos dedos, mas também um movimento específico, seja dos dedos ou da palma da mão, como destaca a Tabela 2.

Para isto, optamos por uma abordagem semelhante à da rede anterior, e iniciamos a arquitetura com um bloco de camadas convolucionais, que aplicarão uma série de filtros

Tabela 2 – Primeiro e último frame de cada uma das poses de Libras que foram trabalhadas nas etapas de gestos dinâmicos

H	J	K	W	X	Y	Z
						
						

Fonte – Autor, 2018.

para ressaltar características interessantes das imagens fornecidas.

Nosso bloco convolucional mais uma vez consiste de três camadas, aplicando respectivamente 16, 16 e 32 filtros de dimensão 3×3, cada camada seguida por uma de agrupamento máximo que reduz os dados pela metade, ressaltando apenas os filtros mais relevantes.

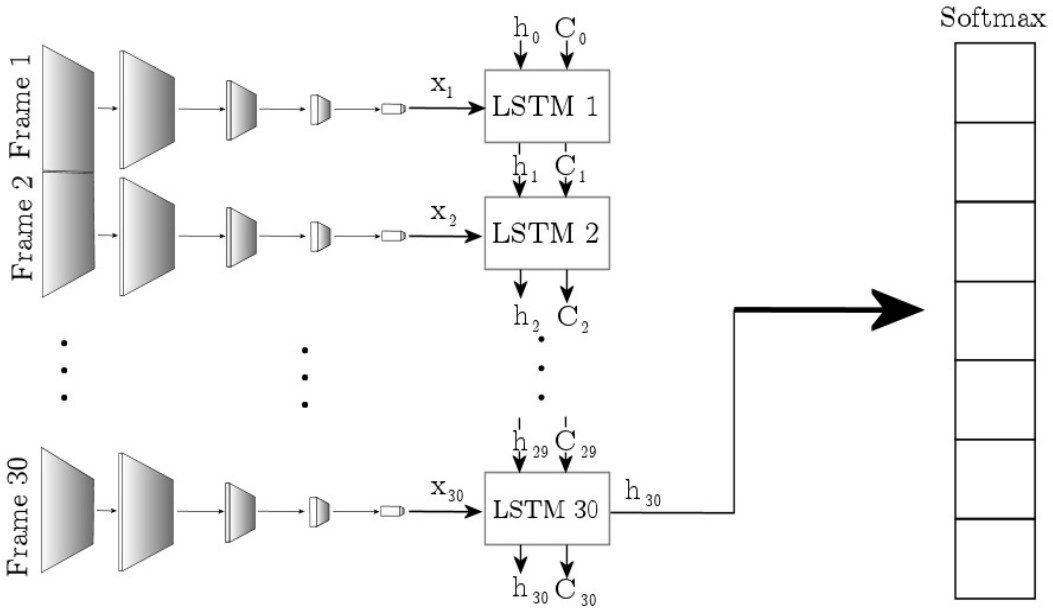
Entretanto, como ilustra a Figura 26, a diferença fundamental entre as duas redes é a existência de uma dimensão temporal nesta etapa. Os dados de entrada não são simples imagens de dimensão $40 \times 40 \times 1$, mas sim uma sequência composta de 30 dessas imagens. O que precisamos é aplicar o modelo de rede convolucional comentado anteriormente para cada uma dessas imagens, sem que cada instante de tempo interfira nos dados dos demais instantes.

Assim, na prática, para cada exemplo da segunda etapa, este bloco convolucional já discutido é aplicado independentemente 30 vezes, e o resultado é uma sequência de 30 vetores de características de dimensão 288.

Entretanto, a partir de agora, não queremos que os dados sejam avaliados independentemente, instante após instante. Por isso, ao invés de seguir apenas aplicando o modelo da etapa anterior, iremos substituir o segundo bloco, que era uma série de camadas densas, por uma camada LSTM (*Long Short-term Memory*) composta de 150 unidades, após o processamento da última imagem.

Apenas o resultado da última etapa da camada LSTM, onde as características de cada imagem foram dados de entrada, na ordem cronológica de execução, é então usado em uma única camada densa, com ativação $softmax$ e quantidade de neurônios igual à quantidade de classes avaliadas (neste caso, 7).

Figura 26 – Representação visual da estrutura da rede utilizada para classificação de gestos (pelo método Offline)



Fonte – Autor, 2018.

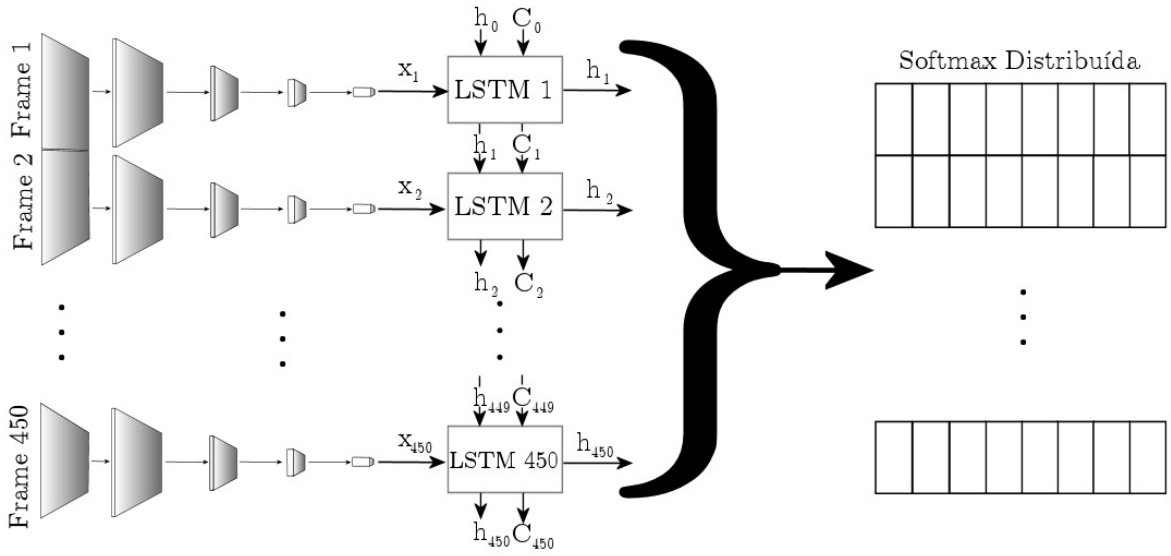
4.2.3 Reconhecimento de Gestos Online

Para o caso Online, as diferenças estruturais na arquitetura são minúsculas: Utilizaremos a mesma estrutura apresentada na Figura 26, exceto por três pontos:

- Os dados de entrada serão sequências de 450 quadros de imagens;
- A quantidade de unidades na camada LSTM, que aqui terá 250;
- O fato de que a camada LSTM não retornará apenas o último resultado processado, mas toda a sequência de resultados de cada frame, desde h_1 até o último, h_{450}.

Por fim, a última camada, que contém a ativação $softmax$ e é responsável por gerar o vetor de probabilidades, será calculada a cada um dos instantes da dimensão de tempo da saída. Isso significa que ao invés de termos um vetor de probabilidades definindo a classe de toda a sequência, teremos 450 vetores probabilidade, cada um definindo a classe correspondente de um frame do exemplo, como ilustra a Figura 27.

Figura 27 – Representação visual da estrutura da rede utilizada para classificação de gestos (pelo método Online)



Fonte – Autor, 2018.

A maior diferença entre os casos Online e Offline não se dá na arquitetura da rede, mas sim no treinamento e na manipulação dos exemplos de predição em tempo real, e tais processos serão descritos na sessão 4.3.

4.2.4 Transferência de Aprendizagem entre Poses e Gestos

Apesar das poses que estamos considerando em cada um dos casos anteriormente abordados serem diferentes (as do primeiro caso estáticas, e as do segundo caso, dinâmicas) todas as poses possuem a mesma natureza (são compostas de palma e dedos) e passaram pelo mesmo pré-processamento, e então presumimos que as características ressaltadas pela primeira rede (sinais estáticos) serão úteis para categorizar as imagens da segunda e terceira rede (sinais dinâmicos).

Assim, ao definirmos o bloco convolucional da segunda e terceira rede, utilizamos exatamente as mesmas camadas e filtros do bloco correspondente da rede anterior, com o objetivo de aplicarmos uma técnica de Transferência de Aprendizagem, resgatando e fixando os parâmetros já bem treinados anteriormente para esta nova rede.

A Transferência de Aprendizagem é muitas vezes utilizada com o objetivo de otimizar o treinamento, reduzindo tempo de processamento e aumentando a eficiência computacional.

No nosso caso, a transferência foi realizada sobre cerca de sete mil parâmetros, que correspondiam ao bloco convolucional. Esse número é pálido em comparação à quantidade total de parâmetros da rede, e portanto, o ganho em eficiência computacional foi pouco significativo.

Entretanto, acreditamos que a estrutura mais simples da primeira rede, que trata de exemplos sem dimensão temporal e possui menos variáveis para controlar, aliada à maior variedade de exemplos de treinamento (com 19 classes ao contrário de 7) resultará numa calibração de parâmetros mais precisa durante o treinamento, e consequentemente este bloco convolucional será mais competente em extrair características relevantes de cada imagem, mesmo que elas estejam em meio a uma sequência de outras imagens, como são os exemplos usados na segunda rede.

Portanto, nosso propósito com esta Transferência de Aprendizagem não é melhorar a eficiência do treinamento da rede, mas sim tornar a predição mais precisa, através de uma melhor extração de características das imagens.

4.2.5 Multi-entradas com dados do Esqueleto

Parte relevante da nossa pesquisa é estudar qual o impacto da utilização de dados do esqueleto junto com Redes Neurais Convolucionais, que foram detalhadas anteriormente. Estes dados se caracterizam por um vetor unidimensional de 14 (ou 26) posições de números flutuantes, correspondendo à ângulos entre os ossos dos dedos da mão (e à vetores de direção, posição e velocidade da mão), e a integração deles aos modelos já descritos é dada através de uma rede híbrida com múltiplas entradas. Denominaremos esse modelo de rede mista, em contra-partida às redes que usam exclusivamente imagens e esqueleto.

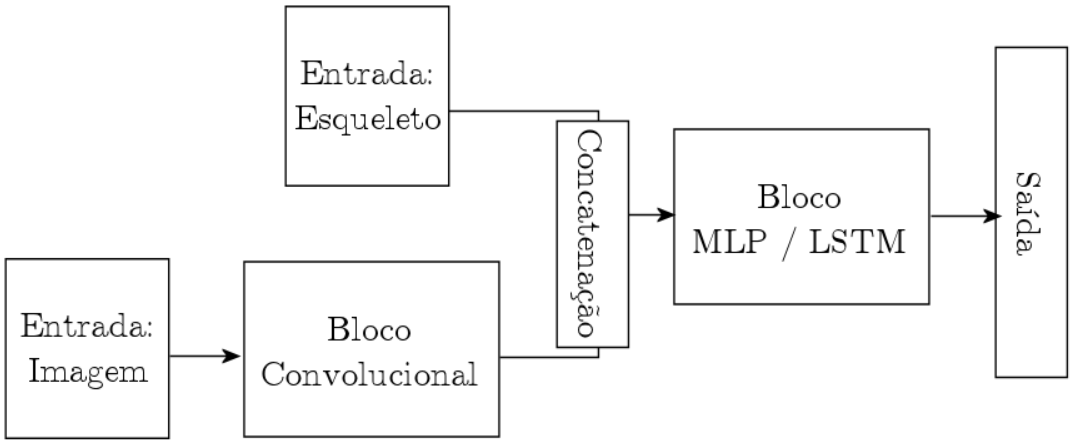
À estrutura criada para reconhecer poses estáticas, basta incluir uma simples entrada extra com os dados do esqueleto, que então será diretamente concatenada com o vetor de características, resultante do bloco de camadas convolucionais, que é um vetor unidimensional de dimensão 288. É este vetor resultante, com dimensão 302, que será repassado às camadas densas seguintes.

O modelo para os casos dinâmicos, que incluem dimensão temporal, seguirá a mesma ideia, com única alteração que a entrada não se trata de um vetor unidimensional de 26 posições, mas sim de uma sequência de 30 destes vetores, cada um correspondendo a

um instante do movimento do sinal. Tais dados são lidos e logo em seguida concatenados com a sequência de vetores característica resultados do bloco de camadas convolucionais. Por fim, esse conjunto de dados de dimensão 30×288 é passado à camada de LSTM e processado da mesma forma daquele ponto em diante.

Uma representação genérica desses casos pode ser vista na Figura 28. A única diferença sensível entre os casos estáticos e dinâmicos é o fato de, no segundo, o bloco convolucional estar encapsulado para ser processado separadamente para cada instante de tempo recebido pela entrada de imagens.

Figura 28 – Representação visual da estrutura básica de redes com múltiplas entradas



Fonte – Autor, 2018.

4.3 Algorítmo para o Reconhecimento Online

A última das etapas trata do reconhecimento de gestos dinâmicos Online, onde o usuário realiza gestos continuamente, e a máquina não sabe a priori quando cada gesto começa e termina. Essa tarefa apresenta novas dificuldades, e para solucioná-las, mais do que modificar a arquitetura de rede, optamos por modificar o processo de captura e predição, e a forma como abordamos a classificação de sinais.

Ao fazer vários gestos diferentes em sequência, de frente à câmera, além da dificuldade em determinar automaticamente quando um gesto terminou e o subsequente iniciou, naturalmente observamos um intervalo entre eles, preenchido com a movimentação da mão de um gesto para o outro, que essencialmente não é nem o primeiro, nem o segundo gesto.

Com isso em mente, buscamos uma rede que fosse capaz de treinar não apenas as classes de imagens, mas também esse intervalo entre cada gesto, e preparamos uma rede neural que além das 7 classes de gestos que realmente deve reconhecer, treinasse novas classes, caracterizadas como “lixo”, que iriam incluir tudo que não fosse um dos 7 gestos anteriores.

Observamos então que, considerando nossas 7 categorias de gestos, podemos ter 49 transições diferentes entre eles. Tivemos que optar então por uma dentre duas abordagens naturais: Criar 49 novas categorias, e colocar cada transição distinta em sua própria classe; ou criar apenas 1 nova categoria, e agrupar todas as transições, mesmo que sejam distintas entre si, na mesma classe.

Para explicar precisamente qual optamos, precisamos antes explicar a respeito da segunda mudança importante na arquitetura: o fato de determinarmos a qual classe pertence cada quadro de nosso exemplo, ao contrário de apenas determinar uma classe para todo o exemplo. Com isso podemos determinar a cada instante do exemplo se o movimento que está sendo feito é correspondente a algum gesto real ou se é um momento intermediário entre dois gestos.

Assim sendo, ambas abordagens mencionadas anteriormente possuem problemas próprios. Adicionar muitas classes tornaria obrigatório o treinamento intensivo de cada uma delas, o que tornaria a criação do banco de treinamento muito mais complexa. Além disso, testes iniciais apontaram uma queda no desempenho da rede ao adicionarmos muitas classes de lixo, e por fim, precisamos considerar a possibilidade do usuário executar um movimento incorreto, que não seja exatamente uma transição entre dois sinais dinâmicos (acidentalmente abrir completamente a mão, por exemplo).

Por outro lado, manter uma única classe tornaria ela muito ampla, com exemplos muitos distintos entre si, o que dificultaria para a rede encontrar características em comum para a classificação. Ainda mais, a proporção entre a quantidade de quadros de lixo e quadros de cada classe específica ficaria muito desequilibrada. Em nossos experimentos, encontramos mais de cinco vezes a quantidade de exemplos de lixo do que a de qualquer um das outras classes.

Tendo em vista isso, optamos não pela criação de uma nova classe, mas numa leve modificação no processo de treinamento de cada quadro. Em todas as redes neurais desse

trabalho, associamos a cada exemplo de classe i um vetor em código *one-hot*, que consiste num vetor de n posições (onde n é o total de classes trabalhadas), todas com valor nulo, exceto na i-ésima posição, que possui valor 1.

Ou seja, um frame que fosse associado ao sinal W não seria representado pelo número 3, mas sim pelo vetor $(0, 0, 0, 1, 0, 0, 0)$. A modificação que fizemos especificamente para o processo Online foi que, quando um frame estivesse associado à classe de lixo, ou seja, fosse um momento entre dois sinais bem estabelecidos, ao invés de associá-lo à uma nova classe, o fizemos ao vetor $(\frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n})$, onde, nesse caso, $n = 7$ e portanto $\frac{1}{n} \approx 0.14$.

Assim, garantimos que a rede será treinada para que frames de lixo ou ambíguos tendessem para a saída $(\frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n}, \frac{1}{n})$, que é um vetor de probabilidade válido (a soma dos elementos totaliza 1), portanto é uma saída razoável para a camada de $softmax$ no final da rede.

Como o resultado de todas nossas redes neurais é uma distribuição de probabilidade, mesmo durante a etapa de poses ou a Offline, um problema com o qual precisamos lidar foi o de definir quando uma predição é clara o suficiente para definirmos a execução de um sinal.

A abordagem natural dessa questão é observar qual das classes possui probabilidade maior que as demais, e defini-la como a predição atual. Em casos ideais, onde a maior das predições é suficientemente alta, como 90%, essa abordagem não apresenta problemas. Mas em casos particulares, mesmo uma probabilidade baixa como 41% poderia ser o suficiente para caracterizar um gesto, mesmo que houvesse outra classe com uma probabilidade igualmente alta, de, por exemplo, 40% (enquanto as demais classes dividiriam os 19% restantes).

Avaliamos que, em casos como esse, a rede deveria oferecer um retorno igualmente ambíguo, já que não foi capaz de definir com precisão qual dos gestos foi realizado. Por fim, a abordagem que optamos por utilizar é a de comparar as duas maiores probabilidades dentre a distribuição resultante da rede neural: se a primeira delas for pelo menos 1.5 vezes superior à segunda, como por exemplo, 24% e 15%, o resultado será definido como o correspondente à primeira classe. Caso contrário, o resultado será classificado como "indefinido", já que a rede neural não obteve o grau de precisão desejado.

Nosso objetivo é, em tempo real, recolher os últimas 100 frames capturados pelo *Leap Motion*, e passá-los à rede Online, que vai definir a categoria de cada frame individualmente. Então avaliaremos esse resultado, a partir do frame mais recente, em direção aos frames mais antigas, procurando por momentos onde o movimento realizado deixou de ser um movimento intermediário para se tornar um gesto reconhecido, e quando ele deixou de se tornar um gesto reconhecido para se tornar um movimento intermediário. Podemos seguramente definir estes momentos, respectivamente, como o final e o início de um gesto.

Verificamos ainda se a rede predisse a mesma classe por um período longo de frames (maior que uma quantidade pré-determinada). Em caso positivo, entendemos que o sinal correspondente aquela classe foi efetuado e finalizado, e indicamos ao sistema que tal gesto ocorreu.

Mudanças na predição por uma quantidade de frames menor que a mínima são descartadas, sendo tratadas apenas como uma pequena flutuação nas probabilidades. Naturalmente, por exemplo, é impossível executar um sinal complexo como os que estamos considerando em menos do que 5 frames, que correspondem a 15 centésimos de segundo.

Por fim, caso um sinal seja detectado por um período longo de frames, mas o sistema não seja capaz de determinar com precisão qual sinal foi realizado, estando em dúvida entre duas ou mais categorias, utilizaremos a informação de início e fim do gesto, adquirida anteriormente, para prosseguir com a análise.

Ignoraremos os demais frames e focaremos nossa análise apenas neste pequeno sub-exemplo. Se entre o início e o fim desse gesto, os resultados da rede neural forem incontestáveis, ou seja, a grande maioria dos frames pertencer à mesma classe, podemos definir o gesto feito entre esses dois momentos como pertencente àquela classe. Uma situação semelhante à essa é retratada na Tabela 3a.

Eventualmente podemos encontrar casos onde a conclusão não é tão simples, como o movimento do gesto H, que tem seus primeiros frames muito semelhantes ao movimento do gesto K (veja na Tabela 2). Nesse caso, é compreensível e até mesmo esperado que a rede encontre dificuldades e seu resultado seja inconsistente ao longo do sub-exemplo, e tenhamos uma situação como a descrita na Tabela 3b.

Mas mesmo nessa situação fomos capazes de definir com certa precisão o início e o final do gesto, e com essa informação, podemos repassar esse sub-exemplo diretamente

para a rede anterior, a de reconhecimento de gestos offline, que tendo o gesto completo para avaliar, poderá predizer o gesto feito com baixa margem de erro.

Tabela 3 – Exemplos de possíveis predições feitas pelo algoritmo Online

(a) Predição consistente: Entre os quadros 3 e 7, o gesto da classe 1 foi feito

Quadro	Classe
1	X
2	X
3	1
4	1
5	1
6	1
7	1
8	X
9	X
10	X

(b) Predição inconsistente: Entre os quadros 3 e 8, algum gesto foi feito

Quadro	Classe
1	X
2	X
3	1
4	1
5	3
6	3
7	3
8	1
9	X
10	X

Fonte – Autor, 2018.

Dessa forma, a cada quadro coletado pelo *Leap Motion*, apenas uma ou duas predições são necessárias para determinar com precisão qual o último gesto realizado, tornando o processo plenamente capaz de ser executado em tempo real.

Portanto, o uso de duas redes neurais distintas, uma preparada especificamente para o processo online, e outra apta a reconhecer os gestos offline, para quando a primeiro se mostrar não confiável, garante um método robusto e eficiente de predição de gestos.

5. EXPERIMENTOS

A seguir iremos detalhar os experimentos realizados em cada uma das fases da pesquisa. Em cada uma delas, iniciaremos com particularidades do banco de dados, apresentaremos alguns resultados da rede neural, eventualmente apresentando algumas variações estruturais da mesma, para propósitos de ilustração, e por fim adicionaremos alguns comentários a respeito do processo de predição que faz uso da rede neural já treinada.

5.1 Reconhecimento de gestos estáticos

A primeira etapa trata do treinamento e reconhecimento de gestos estáticos. Naturalmente ela é a mais simples das etapas, o que nos permitiu criar um banco de dados relativamente extenso e alcançar os melhores resultados tanto observando o treinamento em si quanto no módulo final de predição.

5.1.1 Banco de dados

O banco de dados dessa etapa consistiu de 80 exemplos de cada uma das 19 classes, totalizando 1520 capturados. Cada um consiste de 2 arquivos:

- Uma imagem no formato PNG (*Portable Network Graphics*), em tons de cinza, de 40 pixels quadrados.
- Um arquivo de texto plano contendo 14 valores numéricos correspondentes a ângulos entre ossos específicos da mão, conforme já especificamos na Figura 24

Os exemplos foram capturados de usuários diferentes, onde cada usuário contribuiu com 20 exemplos para cada classe. A captura dos exemplos foi feita a partir de um módulo minimal, que exibia a imagem capturada em tempo real, permitia ao usuário indicar a qual classe correspondia o sinal que ele estava fazendo naquele momento, a com um comando do usuário, capturava tanto a imagem quanto os dados de ângulos do esqueleto, conforme detalhamos na sessão 4.1.2.2

Distinguiremos a base de treinamento da base de validação da seguinte forma: Cada exemplo está contido em uma pasta cujo nome é o identificador da classe, numerais de 0 (representando a classe A) até 18 (a classe V), e estas pastas estão divididas em outras duas pastas, uma representando o conjunto de treinamento (com 60% dos exemplos) e outra o conjunto de validação (com os 40% restantes).

Para garantir que os exemplos de treinamento e validação não fossem sempre os mesmos, foi criada uma rotina que, imediatamente antes de todo treinamento da rede neural, automaticamente reunia todos os exemplos e aleatoriamente selecionasse 60% de todo o banco para o conjunto de treinamento, e colocasse o restante no conjunto de validação, enquanto garantia que cada imagem fosse devidamente acompanhada de seu arquivo de texto correspondente.

Por fim, com o objetivo de aumentar o banco de treinamento, trabalhamos com pré-processamento (*data-augmentation*) nas imagens do mesmo, através da própria biblioteca de Redes Neurais. Optamos por rotacionar os exemplos em, no máximo, 15° para a esquerda ou para a direita, de forma que ainda mantivessem sua integridade como sinal (já que alguns sinais distintos são representados pela mesma pose, apenas rotacionada). O mesmo processo não é feito nos dados vindos do esqueleto, que consistem unicamente de ângulos invariantes por rotações.

5.1.2 Resultados

Os testes foram efetuados sobre a mesma base de dados, ou seja, a mesma divisão de bancos entre treinamento e validação, utilizando os mesmos parâmetros em comum e ao longo das mesmas 30 épocas de treinamento. As matrizes de confusão correspondentes aos testes podem ser analisadas na Tabela 4 (referente aos testes feitos apenas com as imagens), Tabela 5 (apenas com os esqueletos) e Tabela 6 (com ambos os tipos de dados

de entrada).

5.1.2.1 Apenas imagens como entrada

A rede neural que utilizou apenas imagens como dados de entrada alcançou acurácia de 99% na base de treinamento, e de 92% na base de validação (como detalha a Tabela 4). Vemos que os erros mais comuns ocorrem entre os sinais C, F, R, T e V.

Em alguns desses erros podemos ver limitações do uso de apenas imagens para essa tarefa. Por exemplo, mesmo que os sinais C e V sejam bastante distintos entre si, algumas de suas imagens podem ser interpretadas como "punho fechado e duas extremidades", dependendo do quão "curvos" ficam os dedos no sinal C, e quão "inclicado" fica o punho no sinal V, como fica claro observando a Tabela 1.

Por outro lado, os sinais F e T possuem diferenças sutis entre eles, e esperávamos que fossem difíceis de se diferenciarem. Mas enquanto eles alcançaram acurácias abaixo da média, eles não foram frequentemente confundidos um com o outro, mas com outros sinais com menos semelhança com eles, como P e C (veja na Tabela 1). De maneira similar, o sinal R foi frequentemente confundido com L, e facilmente distinguido do sinal U, que possui um gesto muito mais próximo a ele.

5.1.2.2 Apenas esqueleto como entrada

A rede neural que utilizou apenas ângulos do esqueleto da mão como dados de entrada alcançou acurácia de 83% na base de treinamento, e de 78% na base de validação (como detalha a Tabela 5). Os erros aqui se tornam mais frequentes, mas os sinais com pior desempenho são C, F, O, Q e V

Apesar dos resultados inferiores, podemos mais uma vez observar as limitações do uso de apenas dados do esqueleto para essa tarefa. Levando em conta apenas os ângulos entre as falanges, a diferença entre um sinal C e um sinal O é bastante discreta, apesar de que com o recurso visual, conseguimos distinguí-los facilmente se observamos se as pontas dos dedos se tocam ou não. De forma semelhante, a distinção entre os sinais D e O, do ponto de vista do esqueleto, é feita através de um único ângulo (o entre a primeira falange do indicador e a palma da mão), enquanto visualmente temos imagens com contornos mais distintos.

Tabela 4 – Matriz de confusão da base de validação, para poses estáticas, utilizando como entrada apenas as imagens capturadas

	A	B	C	D	E	F	G	I	L	M	N	O	P	Q	R	S	T	U	V
A	32																		
B		31												1					
C			27		1	1													3
D				32															
E					32														
F			3		1	27			1										
G							32												
I								30						1					1
L			2						29						1				
M							1			31									
N											32								
O			1	1								29							1
P													31				1		
Q								2						30					
R							2		8						22				
S							2									30			
T													5				26		1
U																		32	
V			5	1															26

Fonte – Autor, 2018.

De todos os casos avaliados, o erro mais frequente é a confusão entre os sinais D e Q, que visualmente são muito diferentes. Entretanto, observando unicamente seus ângulos, vemos que eles se distinguem por uma pequena diferença em três ângulos, entre a primeira falange e a palma da mão dos dedos médio, anelar e mínimo.

5.1.2.3 Imagens e esqueleto como entrada

Por fim, a rede neural que utilizou ambos dados de entrada, imagens e ângulos de esqueleto, alcançou acurácia de 99% na base de treinamento, e de 94% na base de validação (como detalha a Tabela 6). Vemos que a melhora em relação à rede que usa apenas imagens é modesta, entretanto, em relação à rede que usa apenas esqueleto, é bastante expressiva.

Aqui, notamos uma pequena melhora em todos os sinais problemáticos da primeira rede, mas em especial nos sinais R e V. Isso é interessante, pois o sinal R teve resultados medianos em ambos os testes anteriores (atingindo apenas 68% de precisão em cada um), mas ao combinarmos ambos os tipos de entrada, sua acurácia subiu sensivelmente para

Tabela 5 – Matriz de confusão da base de validação, para poses estáticas, utilizando como entrada apenas os esqueletos capturados

	A	**B**	**C**	**D**	**E**	**F**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**
A	29					1		1				1							
B		31										1							
C	2		14	1	1	3	2	1									3		5
D	1			29								1		1					
E					27	2				2						1			
F	3			2	1	17		1							2		5		1
G					4		27			1									
I	1		1					22				2	1	3					2
L									32										
M		1								31									
N											32								
O	3		1	8								18					1		1
P													26			2	3		1
Q				9				2						21					
R	1		2			3		2	2						22				
S																32			
T	1				1								5				24		1
U																1		31	
V	3		2	2		5		6				3					1		10

Fonte – Autor, 2018.

90%. O sinal V é ainda mais impressionante, já que a rede de esqueletos teve muita dificuldade em reconhecê-lo, e sua acurácia nela era de apenas 31%.

Por outro lado, tivemos aqui uma queda na precisão do sinal D, que seguiu um caminho completamente oposto: teve grande índice de acerto nas duas primeiras redes, e nesta foi inferior à ambas.

O sinal F também perdeu um pouco de precisão se comparada à rede que usava imagens, mas até certo ponto esse comportamento é compreensível, tendo em vista a dificuldade que a rede de esqueletos teve para categorizá-lo.

De forma geral, a maioria dos sinais teve apenas uma leve perturbação em suas acurácias, que não representaria uma vantagem sólida entre o uso apenas de imagens, ou de imagens juntamente com o esqueleto. Mas o grande ganho em acurácia de sinais específicos é mais do que suficiente pra compensar as perdas em outros, que são em muito menor magnitude.

Tabela 6 – Matriz de confusão da base de validação, para poses estáticas, utilizando como entrada tanto as imagens quanto esqueletos capturados

	A	**B**	**C**	**D**	**E**	**F**	**G**	**I**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**
A	32																		
B		32																	
C	1		27			2													2
D				28				2				1							1
E					31								1						
F			3		1	26			1										1
G							31			1									
I								32											
L									28						4				
M										32									
N											32								
O								1				31							
P													32						
Q								2						30					
R								1	2						29				
S																32			
T													4				28		
U																2		30	
V			1	1															30

Fonte – Autor, 2018.

5.2 Reconhecimento de gestos dinâmicos (Offline)

A segunda etapa trata do treinamento e reconhecimento individual de gestos dinâmicos. Um nível mais complexo que a tarefa anterior, cada exemplo é dezenas de vezes mais complexo do que no caso anterior, uma vez que onde usávamos 1 única imagem para representar o exemplo, agora cada um é composto de, em média, 30 imagens, acompanhadas das informações de esqueleto correspondentes.

5.2.1 Banco de dados

O banco de dados dessa etapa consistiu de 100 exemplos de cada uma das 7 classes, totalizando 700 exemplos capturados. Cada exemplo consiste de uma pasta, dentro da qual se encontram:

- Uma quantidade arbitrária de imagens no formato PNG (Portable Network Graphics), em tons de cinza, de 40 pixels quadrados, nomeadas com seu índice correspondente

na sequência de frames que constitui o exemplo, de 0 até $n - 1$;

- Um único arquivo de texto plano, nomeado “esqueleto.txt”, contendo $n - 1$ linhas, cada uma correspondendo ao esqueleto do frame de mesmo índice; e em cada uma das linhas, 26 valores numéricos correspondentes à ângulos entre ossos específicos da mão (como mostra a Figura 24), direções (como a Figura 20), posições e velocidade da mão.

Mais uma vez, os exemplos foram capturados de usuários diferentes, onde um contribuiu com 40 exemplos de cada classe, e outros três contribuíram, cada um, com 20. A captura foi feita a partir de uma versão levemente modificada do módulo da etapa anterior, onde o usuário indicava a classe, e depois o início e fim do sinal, enquanto todos os frames recebidos entre esse intervalo eram armazenados a partir do início, e no final eram todos processados e armazenados no banco de dados.

Mais uma vez, a distinção entre conjuntos de treinamento e validação é feita manualmente. Da mesma forma que na etapa anterior, cada exemplo (neste caso, uma pasta) está contido na pasta que corresponde à sua classe, representadas com os numerais de 0 (sinal H) até 6 (sinal Z), e as pastas de classes são divididas entre as pastas de treinamento (60% do total) e validação (40%).

Semelhante à etapa anterior, garantimos a aleatoriedade da distribuição dos exemplos em uma rotina separada, que organiza os exemplos em suas pastas de treinamento e validação logo antes de cada execução de treinamento.

5.2.2 Resultados

Novamente, os testes abaixo foram efetuados sobre a mesma base de dados, ou seja, a mesma divisão de bancos entre treinamento e validação, utilizando os mesmos parâmetros em comum e ao longo das mesmas 30 épocas de treinamento. As matrizes de confusão correspondentes aos testes podem ser analisadas na Tabelas entre 7 e 9, que incluem os resultados dos testes apenas para imagens, apenas para o esqueleto, para ambos os dados; além disso, os testes que incluem imagem serão ainda divididos entre testes com e sem transferência de aprendizado, como já abordamos na sessão 4.2.4.

5.2.2.1 Apenas imagens como entrada

Nosso teste inicial, que treinava todos parâmetros a partir do zero, incluindo os do bloco convolucional, alcançou 83% na base de treinamento, e 79% na base de validação (como detalha a Tabela 7a). O segundo teste, que carrega os parâmetros do bloco convolucional da rede anterior, alcançou acurácias de 97% na base de treinamento, e 92% na de validação (como mostra a Tabela 7b).

Primeiro, podemos observar uma sensível melhora no desempenho da rede, mostrando que uma rede dedicada a localizar características relevantes da imagem atinge seu objetivo em menos tempo que uma rede mais complexa. A rede atual teve as mesmas 30 épocas de treinamento, e esse treinamento foi feito diretamente sobre exemplos relevantes para a os testes posteriores. Ainda assim, a primeira rede, que fez seu treinamento em outra base de dados, foi capaz de extrair características relevantes para este caso.

Vemos que o único erro relevante acontece no sinal X, frequentemente confundido com o sinal J. Do ponto de vista das imagens, o X é um sinal bastante complexo, pois ele se parece com uma grande mancha branca (o punho fechado) que se afasta da câmera, se tornando cada vez menor e mais escuro. O J, por sua vez, é uma mancha branca com uma única ponta, que rotaciona com o passar do tempo, da direita para a esquerda (como podemos ver pelas imagens da Tabela 2).

Percebemos ainda que esse mesmo erro é também o mais comum também na rede que não recebeu transferência de aprendizagem, indicando que não se trata de um problema causado pela falta de uma característica exclusiva do banco de dados de gestos dinâmicos. De fato, algo assim poderia acontecer: a característica que permitiria a classificação desse sinal não é encontrada nos sinais estáticos, e portanto a primeira rede neural a ignorou. Entretanto, o fato de que mesmo treinando diretamente no banco de dados de sinais dinâmicos a rede neural não foi capaz de classificá-lo adequadamente descarta essa hipótese.

Por outro lado, a rede sem a transferência de imagens tinha bastante dificuldade em diferenciar os sinais W e H, o que é bastante surpreendente dado que eles são bastante distintos entre si (veja a Tabela 2). Entretanto, essa precariedade foi suprida ao utilizarmos os parâmetros da rede neural para sinais estáticos.

Tabela 7 – Matrizes de confusão da base de validação, pelo método offline, utilizando como entrada apenas as imagens capturadas

(a) Resultados sem o uso de transferência de aprendizagem

	H	J	K	W	X	Y	Z
H	39					1	
J	2	36			1	1	
K	8	1	30	1			
W	11		3	26			
X	1	17			19		3
Y		1				38	1
Z		5		1			34

(b) Resultados com o uso de transferência de aprendizagem

	H	J	K	W	X	Y	Z
H	36	1	1	1	1		
J	2	35			2	1	
K	1		38	1			
W				40			
X		7			32		1
Y						40	
Z		1		1	1		37

Fonte – Autor, 2018.

5.2.2.2 Apenas esqueleto como entrada

A rede neural que utilizou apenas ângulos do esqueleto da mão como dados de entrada alcançou acurácia de apenas 69% na base de treinamento, e de 55% na base de validação (como detalha a Tabela 8). Assim como o ocorrido na rede neural para poses, os erros aqui são abundantes, mas dentre os sinais com pior desempenhos se destacam K, Y e H

A primeira e mais esperada confusão acontece entre os sinais K e W, que do ponto de vista do esqueleto, são de fato bastante semelhantes, se diferenciando apenas por um ângulo (o entre a primeira falange do dedo anelar e a palma da mão), enquanto reproduzem inclusive o mesmo movimento de translação.

Outras confusões, como entre os sinais Y e J, são mais difíceis de serem explicadas. A posição de seus dedos se difere, também, por apenas um ângulo (entre o dedo polegar e a palma da mão), mas durante o movimento do sinal J, a direção da normal da palma da mão rotaciona de uma posição para seu completo oposto, enquanto no movimento de Y percorre metade dessa distância, e conclui seu movimento numa posição completamente distinta de J em todos os momentos.

Ao mesmo tempo, o sinal J, que possui um movimento bastante único e, portanto, que deveria ser de fácil classificação, é o sinal com a maior quantidade de falsos positivos dessa rede. Uma das possíveis causas na dificuldade em se categorizar o sinal J, e por consequência, a frequente confusão de outros sinais com ele, é o fato de que na segunda

Tabela 8 – Matriz de confusão da base de validação, pelo método offline, utilizando como entrada apenas os esqueletos capturados

	H	**J**	**K**	**W**	**X**	**Y**	**Z**
H	17	8	3		4	3	5
J	2	22	1		1		14
K			14	21	1	3	1
W	1		4	23	4	6	2
X		10			28	1	1
Y		15		7	2	16	
Z	1	4			1		34

Fonte – Autor, 2018.

metade do movimento desse sinal, a posição exata das pontas dos dedos polegar, indicador, médio e anelar ficam ocultas pela palma da mão. Sem poder ler os dados com precisão, o esqueleto fornecido pelo *Leap Motion* torna-se questionável.

Por outro lado, os sinais que mais tiveram sucesso em ser categorizados, mesmo que tenham atingido apenas 70% e 80% de precisão, foram as letras X e Z, que possuem movimentos bastante distintos das outras classes, e cujo movimento é claramente capturado pelo *Leap Motion*.

5.2.2.3 Imagens e esqueleto como entrada

Mais uma vez, a distinção entre os testes com e sem transferência de aprendizagem precisa ser feita: Nos testes que treinavam todos os parâmetros, incluindo o bloco convolucional, a rede alcançou 65% na base de treinamento, e 60% na base de validação (como detalha a Tabela 9a). O segundo teste, que carrega os parâmetros anteriores, alcançou 73% na base de treinamento, e 63% na base de validação (como mostra a Tabela 9b).

Uma comparação rápida com as acurácias e matriz de confusão anteriores revela que a situação é diferente da que tivemos na primeira rede, que tratava de sinais estáticos: Lá, o desempenho da rede com apenas imagens foi superior ao que usava apenas o esqueleto, e a rede que usava ambas foi levemente superior à que usava apenas imagens.

Aqui, manteve-se o desempenho superior da rede de imagens sobre a rede de esqueleto, mas a rede mista, apesar de representar melhoras significativas em relação à rede de esqueleto, é ainda inferior à rede que usa puramente imagens. Portanto, os dados do esqueleto não são suficientes para auxiliar as deficiências dos dados das imagens, e ao

contrário, acabam prejudicando o julgamento da rede, como um todo.

Dito isto, comparando os resultados desse teste, as diferenças entre o uso ou não da transferência de aprendizagem refletem as mesmas mudanças que aconteceram na rede que usava exclusivamente imagens, como a grande melhora na diferenciação dos sinais X e J. A diferenciação dos sinais K e W passa por um fenômeno curioso, onde, com a transferência de aprendizagem, a rede passa de dar preferência ao sinal W para dar preferência ao sinal K, em situações onde está em dúvida entre eles.

As diferenciação de K e W era uma das maiores dificuldades da rede de esqueleto, e como previsto, teve grande melhora com o auxílio das imagens. Entretanto, o uso exclusivo das imagens nos dá resultados indiscutivelmente superiores.

Tabela 9 – Matrizes de confusão da base de validação, pelo método offline, utilizando como entrada tanto as imagens quanto esqueletos capturados

(a) Resultados sem o uso de transferência de aprendizagem

	H	**J**	**K**	**W**	**X**	**Y**	**Z**
H	26	7			1	3	3
J	4	24		1	2	1	8
K			23	15		1	1
W	3		6	28		1	2
X		10		1	25	2	2
Y		8		12	7	12	1
Z	3	3		1	3		30

(b) Resultados com o uso de transferência de aprendizagem

	H	**J**	**K**	**W**	**X**	**Y**	**Z**
H	26	4			4	3	3
J	3	30			1		6
K			31	9			
W	1		21	12		4	2
X		7			29	3	1
Y		7	4	3	4	22	
Z	2	6	5				27

Fonte – Autor, 2018.

5.3 Reconhecimento de gestos dinâmicos (Online)

A última etapa trata do treinamento e reconhecimento de gestos dinâmicos realizados em sequência, e é a mais complexa das tarefas que avaliaremos. Mais uma vez nos restringimos a sinais dinâmicos, então novamente teremos apenas 7 classes distintas a serem categorizadas. Mas nosso maior desafio aqui será reconhecer os gestos sem possuir prévio conhecimento de quando eles começam ou terminam, e como lidar com movimentos intermediários entre dois gestos.

Como já detalhamos anteriormente, para essa tarefa adotamos uma abordagem diferente, onde categorizamos cada frame individual de nossos exemplos. Por conta disso,

mesmo que os sinais que trabalharemos sejam os mesmos da etapa anterior, precisaremos de um banco de dados completamente novo.

5.3.1 Banco de dados

Nossa proposta é criar uma rede capaz de identificar movimentos intermediários entre os sinais que queremos reconhecer, e por esse motivo, fez-se necessário incluir esses movimentos na base de treinamento. Tendo 7 sinais diferentes, naturalmente temos 49 movimentos intermediários entre cada par de sinais, e seriam necessários várias combinações específicas para que englobar todas elas.

Para simplificar o banco de dados, optamos por fazer 12 sequências diferentes dos 7 sinais, buscando cobrir a maior parte das transições. Algumas dessas sequências foram repetidas três vezes, outras duas, e todas foram feitas pela mesma pessoa.

Isso totalizou 29 exemplos, cada um representado por uma pasta, dentro da qual se encontram:

- Uma quantidade arbitrária (em geral ao redor de 300 ou 350) de imagens no formato PNG, em tons de cinza, de 40 pixels quadrados, nomeadas com seu índice correspondente, de 0 até $n - 1$;
- Um arquivo de texto plano contendo $n - 1$ linhas, cada uma correspondendo ao esqueleto do frame de mesmo índice; em cada linha, 26 valores numéricos, exatamente como o arquivo da sessão anterior;
- Um arquivo de texto plano contendo $n - 1$ linhas, cada uma correspondendo a um dos frames, e cada uma contendo um valor inteiro identificando a qual classe corresponde àquele frame (aqui incluindo também uma classe extra, correspondente à classe “lixo”, que engloba as transições entre sinais).

A captura dos exemplos aconteceu em duas etapas diferentes: a primeira, idêntica à captura de exemplos dos sinais dinâmicos, onde o usuário indicava o início e o fim do exemplo, e dentre eles executava todos os sinais que desejava. Em seguida, outro módulo permitia visualizar os exemplos, frame a frame, enquanto o usuário demarcava a qual categoria pertencia cada um dos frames.

A distinção entre os conjuntos de treinamento e de validação, por outro lado, seguiu sendo feita da mesma forma, manualmente, e dividiram os exemplos em pastas de treinamento (60% do total) e validação (40%).

5.3.2 Resultados

Neste momento convém observar que matrizes de confusão falham em demonstrar a acurácia de sucesso desse experimento, devido à abordagem que tivemos de categorizar cada frame ao invés de cada exemplo. Portanto, ao invés disso, iremos selecionar um exemplo arbitrário da base de validação, e dentro dele avaliar, frame a frame, o quão precisos ou não foram as previsões da nossa rede. Uma pequena fração dessa comparação é mostrada na Tabela 10, onde comparamos lado a lado o vetor probabilidade ideal (sobre o qual a rede treinou) e o vetor de probabilidade resultante da rede.

Perceba que, desde antes de efetivamente iniciar o movimento do sinal X, ainda durante os frames de transição, a rede já está esperando que a partir daquele movimento, um sinal X seja realizado em breve, e a probabilidade de X vai, continuamente, aumentando, enquanto a probabilidade dos demais sinais diminui, já que se trata de uma distribuição de probabilidade, cuja soma das partes precisa ser 100%.

Durante a execução do sinal, o vetor de probabilidade atinge a diferença máxima de valores, onde conclui que aquele sinal é indiscutivelmente X, e em seguida, quando percebe que o sinal está sendo finalizado, diminui gradativamente até voltar ao estágio de neutralidade, durante a transição para o próximo sinal.

De forma geral, de todas as formas que a biblioteca que usamos para construir nossas redes neurais (o Keras) tem para calcular acurácia e erro, nenhuma dela realmente reflete nosso objetivo nesse teste. Uma comparação direta, de frame a frame, comparando o valor esperado com o valor calculado, acarreta num grande número de resultados errados, que inevitavelmente diminuem a acurácia e aumentam o erro. Com uma função de erro que não reflete exatamente nosso objetivo, o treinamento em si encontra dificuldades e demora mais para encontrar resultados precisos.

Mas avaliar frame a frame os resultados de um exemplo de mais de 300 frames pode, também, ser pouco intuitivo e visual. Portanto, nesta etapa, estaremos condensando os resultados não em tabelas, mas sim em gráficos que representam as probabilidades

Tabela 10 – Parte da avaliação frame a frame de um exemplo de validação, no método Online

	Resultado Ideal								Resultado Calculado						
Frame	**H**	**J**	**K**	**W**	**X**	**Y**	**Z**		**H**	**J**	**K**	**W**	**X**	**Y**	**Z**
10	14	14	14	14	14	14	14		13	16	12	13	14	15	15
11	14	14	14	14	14	14	14		12	16	11	13	14	14	15
12	14	14	14	14	14	14	14		12	15	11	14	16	13	15
13	14	14	14	14	14	14	14		12	16	12	15	17	12	14
14	14	14	14	14	14	14	14		11	16	12	15	19	11	12
15	14	14	14	14	14	14	14		10	15	12	14	22	11	12
16	14	14	14	14	14	14	14		11	14	11	14	24	10	11
17	14	14	14	14	14	14	14		10	14	11	15	27	10	11
18	14	14	14	14	14	14	14		10	13	10	14	30	9	10
19	14	14	14	14	14	14	14		10	13	10	14	32	9	9
20	14	14	14	14	14	14	14		9	12	10	14	32	9	9
21	14	14	14	14	14	14	14		9	12	10	14	33	9	10
22	14	14	14	14	14	14	14		9	11	9	13	36	9	10
23	0	0	0	0	100	0	0		8	10	8	12	39	9	10
24	0	0	0	0	100	0	0		8	9	8	12	42	8	10
25	0	0	0	0	100	0	0		8	9	7	11	43	9	9
26	0	0	0	0	100	0	0		7	9	7	11	45	8	9
27	0	0	0	0	100	0	0		7	9	7	11	46	8	8
28	0	0	0	0	100	0	0		7	9	7	11	48	7	8
29	0	0	0	0	100	0	0		7	9	7	10	49	7	8
30	0	0	0	0	100	0	0		8	9	6	10	49	7	8
31	0	0	0	0	100	0	0		9	9	6	10	48	7	8
32	0	0	0	0	100	0	0		9	9	6	10	48	7	8
33	0	0	0	0	100	0	0		10	9	6	10	47	7	8
34	0	0	0	0	100	0	0		10	8	6	10	48	7	8
35	0	0	0	0	100	0	0		10	7	6	9	48	8	8
36	0	0	0	0	100	0	0		10	7	6	10	48	7	8
37	0	0	0	0	100	0	0		10	7	7	10	45	8	9
38	0	0	0	0	100	0	0		10	7	8	10	43	9	9
39	0	0	0	0	100	0	0		10	9	9	10	40	11	8
40	0	0	0	0	100	0	0		11	10	9	11	37	11	8
41	0	0	0	0	100	0	0		12	11	9	11	35	12	7
42	0	0	0	0	100	0	0		12	11	9	12	33	12	7
43	0	0	0	0	100	0	0		12	10	9	13	31	13	7
44	0	0	0	0	100	0	0		13	10	9	14	30	13	7
45	14	14	14	14	14	14	14		12	11	10	14	28	14	7
46	14	14	14	14	14	14	14		12	12	11	14	27	14	6
47	14	14	14	14	14	14	14		13	13	11	14	24	15	7
48	14	14	14	14	14	14	14		13	13	11	14	21	17	7
49	14	14	14	14	14	14	14		13	13	11	15	21	17	7

Fonte – Autor, 2018.

calculadas de cada sinal, ao longo dos frames, como pode ser visto nas Figuras 29, 30 e 31.

Por fim, todos os testes a seguir serão realizados exclusivamente com transferência de aprendizagem, tendo em vista os resultados incontestavelmente superiores observados na etapa anterior.

5.3.2.1 Apenas imagens como entrada

Observamos que a rede que utilizou apenas imagens teve resultados muito bons em identificar cada frame como sua classe correspondente, como ilustra a Figura 29. Todos os sinais foram previstos corretamente, em alguns momentos um pouco antes ou depois do início, e em alguns momentos indicando o término do sinal também um pouco antes ou depois do verdadeiro fim, mas sempre dentro de uma margem de erro pequena de, em média, 5 frames.

Tendo em vista que antes de todo sinal executado, existe um movimento intermediário que culmina no início do sinal em questão, mesmo antes do início de um gesto a rede já é capaz de determinar que o usuário está caminhando em direção a determinado sinal, e a probabilidade daquele sinal ser reconhecido passa a aumentar. Quando este sinal se destaca das outras probabilidades o suficiente, o sistema reconhece e associa o frame atual com a classe atual.

Seguindo a execução representada na Figura 29, notamos que o primeiro sinal executado é X, e que ele é identificado sem nenhuma dúvida. Em seguida, o sinal Z é prontamente reconhecido, mas no meio de sua execução, a probabilidade do sinal X aumenta, causando a queda de Z e deixando o sistema indeciso de qual sinal está sendo executado. Essa indecisão dura 6 frames, e logo em seguida Z volta a se destacar, até o final do sinal.

Notamos então que um dos sinais mais problemáticos é o Z. Seu gesto e movimento são bastante distintos de todos os demais sinais, entretanto, a rede tende a categorizá-lo como lixo, indicando a semelhança do sinal com os diversos frames de transição entre dois sinais. Isso é refletido no gráfico, onde vemos que na maioria dos períodos intermediários entre dois sinais, a probabilidade do sinal ser reconhecido como Z sobe. Na maioria dos casos esse crescimento é discreto demais para que o sistema caracterize os frames como Z ao invés de lixo, mas em alguns momentos, como a transição entre Z e Y, o sistema

retorna o falso positivo do sinal Z, por um total de 6 frames.

Em seguida, o sinal Y é executado, e é identificado de forma limpa, com apenas dois frames adiantados no início, e um frame adiantado no final. O J é reconhecido da mesma forma limpa, com uma margem de erro um pouco maior: ele foi reconhecido 4 frames antes do seu início real, e permaneceu sendo reconhecido por 6 frames após seu término.

Em seguida, temos o interessante caso dos sinais H e K. A posição inicial desses gestos é a mesma, então nos frames iniciais, é muito difícil distinguir qual dos sinais está sendo realizado. Como consequência disso, a probabilidade dos dois começa a se elevar já 10 frames antes do início do sinal, e quando ele reconhece o movimento de translação, a probabilidade de H cai, e K se destaca. Em seguida, a mesma situação acontece, com a probabilidade de ambos se mantendo alta ao longo de 8 frames, e quando o sistema reconhece o movimento de rotação, a probablidade de K cai vertiginosamente, dando destaque e reconhecimento ao H.

Consequentemente, dos quase 20 frames que compõe esse movimento H, 8 deles permanecem indefinidos, e apenas da metade em diante a rede tem dados para definir qual sinal está sendo realizado. Isso é uma consequência direta e inevitável da abordagem que usamos nesse caso, avaliando frame a frame, do início ao final, mas que tem pouco impacto em aplicações que tenham interesse em identificar o sinal após a sua conclusão.

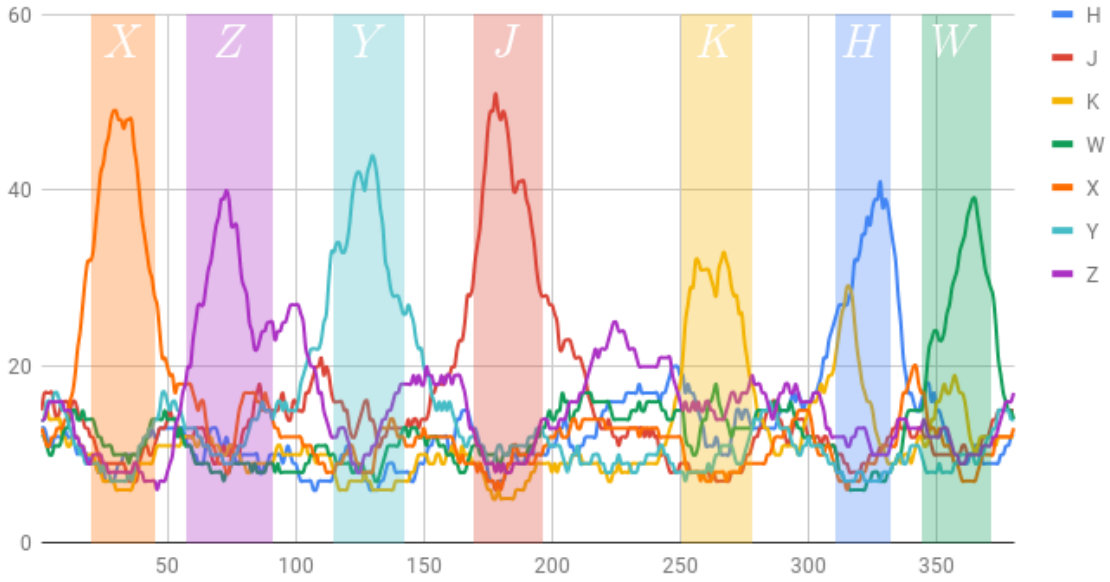
Por fim, o sinal W é identificado com 3 frames de atraso, e passa por um período de falso negativo de 4 frames logo em seu início, quando a probabilidade K também está aumentando. Após isso, ele segue segue corretamente até o final, onde reconhece o sinal por 4 frames a mais do que o caso ideal.

5.3.2.2 Apenas esqueleto como entrada

Observando que a rede neural offline teve muita dificuldade em identificar os sinais baseando-se unicamente pelo esqueleto, podemos esperar o mesmo comportamento nesta rede, que usa a mesma estrutura básica de neurônios. E como ilustra a Figura 30, os resultados são de fato bem inferiores quando comparados ao desempenho da rede que usa exclusivamente imagens.

Notamos, já no início, que o sinal X é confundido de imediato com uma execução de

Figura 29 – Gráfico indicando o desenvolvimento das probabilidades ao longo dos frames, na rede que utiliza apenas imagens, do método online



Fonte – Autor, 2018.

Y, e apenas após a metade da execução a rede reconhece o sinal correto. Mas mesmo esse reconhecimento é repleto de problemas, com vários falsos negativos em meio aos frames reconhecidos como X, e de acordo com os critérios do nosso algorítmo (explanados na sessão 4.3), ele não seria identificado.

O sinal Z, em seguida, é reconhecido com 9 frames de atraso, e sofre de muitos falsos negativos logo em seguida, sendo plenamente identificado apenas faltando 5 frames para a conclusão do sinal. Ainda pior, a probabilidade do sinal não cai durante o período intermediário que se segue, e mesmo quando o sinal executado era Y, por 6 frames a rede ainda o reconheceu como Z, antes de voltar a ficar indeterminada. Por apenas 5 frames a probabilidade de Y se tornou maior que a de Z e X, mas foi uma diferença pequena demais para que a rede considerasse Y como a classe correta.

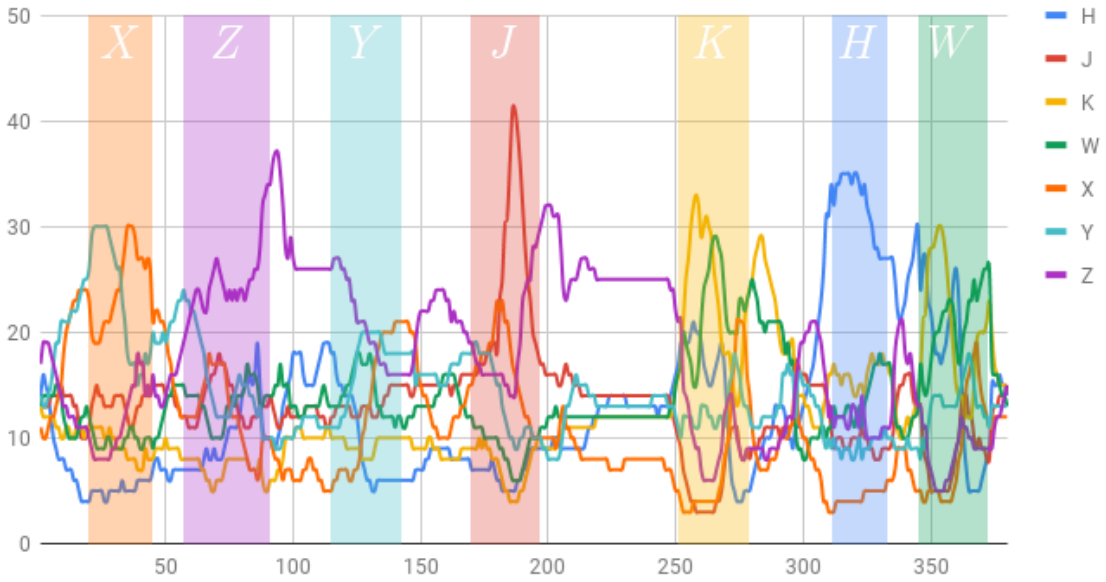
Com um atraso de 14 frames, o sinal J foi corretamente identificado, apenas para logo em seguida perder espaço com o crescimento não justificado do sinal Z, que aqui também ganha relevância em todo momento intermediário entre sinais. De fato, etnre os sinais J e K, a maioria dos frames foi incorretamente classificada como Z.

Nos primeiros frames do sinal K, ele foi corretamente identificado, mas em seguida o crescimento do sinal W voltou a tornar a rede indecisa. Nesse momento, lembramos que

a rede neural offline de esqueletos tinha grande dificuldade em diferenciá-los, e vemos que o mesmo padrão se repete aqui, tanto na execução de K quanto na execução de W.

Por fim, o sinal H foi surpreendentemente bem identificado. Tendo iniciado 4 frames mais cedo, e finalizado 5 frames mais tarde, o sinal foi reconhecido de forma clara, sem falsos negativos ou sequer confusão com o sinal K.

Figura 30 – Gráfico indicando o desenvolvimento das probabilidades ao longo dos frames, na rede que utiliza apenas esqueleto, do método online



Fonte – Autor, 2018.

5.3.2.3 Imagens e esqueleto como entrada

Mais uma vez os testes dessa rede neural são comparáveis com os resultados da rede neural offline, onde os resultados foram superiores ao uso exclusivo de esqueletos, mas ainda longe de serem comparáveis aos da que usa exclusivamente imagens.

Muitos dos erros da rede de esqueletos reaparecem aqui, apenas em menor magnitude. X é identificado apenas nos últimos dois frames de sua execução. Ao invés de reconhecer Y nos primeiros momentos, ele se mantém sempre abaixo do índice de X, mas não baixo o bastante para permitir que X se destaque.

Igualmente, Z é reconhecido apenas no final de sua execução, mas no período intermediário seguinte a rede é mais precisa em classificar mais frames como lixo do que

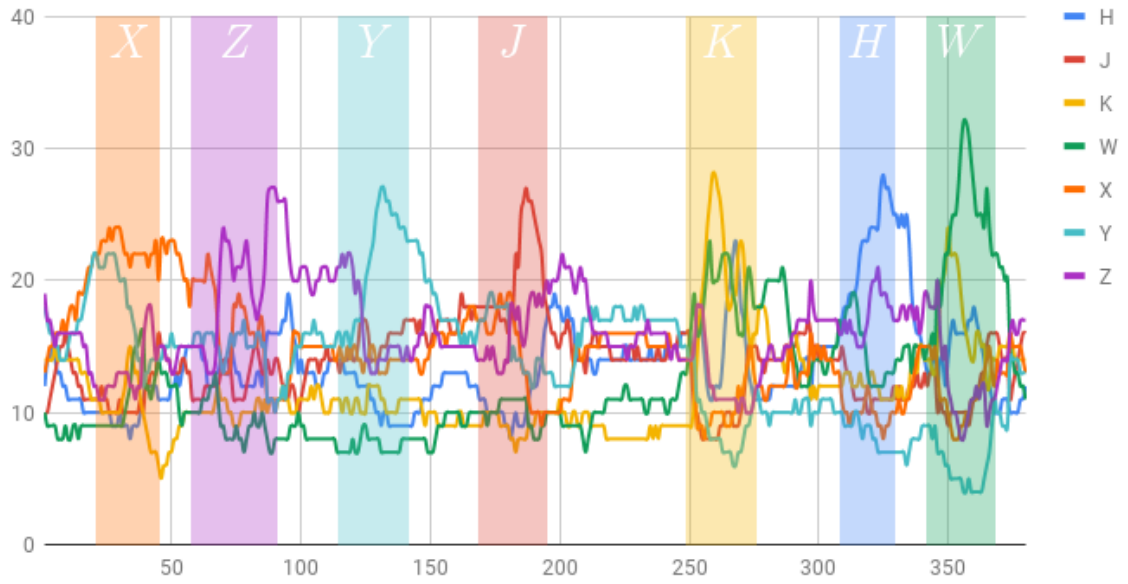
como Z. Y é identificado corretamente, mas apenas depois de 12 frames de atraso no início, e o movimento termina 5 frames antes do ideal.

J mais uma vez demora muitos frames para ser reconhecido, mas ao contrário da rede anterior, não existe confusão com o sinal Z logo após seu término, e no longo período intermediário entre J e K, a rede informa, precisamente, ao invés de Z, como na rede anterior.

Apesar do crescimento sensível de K durante sua execução, ele é seguido por perto pelo sinal W, e não chega a se destacar o suficiente para ser reconhecido, e ao longo de toda a execução o sistema retorna indecisão. O mesmo acontece logo em seguida, com H, que não consegue se diferenciar o suficiente do sinal Z.

Por fim, o sinal W se destaca durante poucos frames, muito menos do que o suficiente para o algorítmo reconhecer o movimento.

Figura 31 – Gráfico indicando o desenvolvimento das probabilidades ao longo dos frames, na rede que utiliza imagens e esqueleto, do método online



Fonte – Autor, 2018.

6. CONCLUSÃO

Apresentamos nesse trabalho metodologias baseadas em Aprendizagem Profunda para reconhecimento de gestos estáticos e dinâmicos. Através do dispositivo *Leap Motion*, capturamos tanto imagens quanto dados do esqueleto da mão, e avaliamos diversas arquiteturas de Redes Neurais para reconhecer gestos, com ênfase em sinais de Libras.

Dividimos as metodologias em três etapas, onde a primeira reconhecia gestos estáticos (poses), a segunda reconheceu individualmente gestos dinâmicos, e a terceira reconheceu os mesmos gestos de forma sequencial, sem pauses ou segmentações.

Analisando os resultados de nossos testes, podemos constatar alguns pontos interessantes acerca de todo nosso trabalho.

Em primeiro lugar, a respeito da motivação inicial de nosso trabalho, vemos que a eficiência do uso de ambos tipos de dados, imagens e esqueleto, está condicionada a alguns fatores. O caso estático alcançou sucesso pleno, atingindo uma maior acurácia do que em ambos casos individuais. Um benefício de 2% pode ser visto como diminuto, mas para avaliar e compreender melhor os impactos, precisamos observar esse impacto individualmente, a cada sinal.

Por outro lado, podemos fazer a análise de como cada classe de gesto foi afetada pelo uso de esqueletos (combinado com as imagens). Nesse sentido, o ganho em acurácia da classe R foi de 68% para 90%, enquanto o da classe T foi de 81% para 87%, e o da classe F caiu de 84% para 81%. A forma mais natural de compreender essas diferenças é que alguns sinais específicos se beneficiam do uso de esqueleto mais do que outros, o que é bastante razoável.

Apesar desses resultados interessantes nas redes estáticas, temos que o desempenho das redes mistas nos casos dinâmicos ficou muito abaixo do esperado, com a rede exclusiva de imagens alcançando quase 30% a mais de acurácia que a rede mista. Uma possível

explicação seria que, durante a realização de um sinal dinâmico, o esqueleto pode não ser bem rastreado em alguns instantes, prejudicando o reconhecimento do sinal inteiro.

Por outro lado, talvez o motivo da diferença de acurácia entre as redes mistas estáticas e dinâmicas não seja especificamente a dimensão temporal, mas sim o conjunto de sinais abordados, que no segundo caso, não apresentam situações onde a deficiência por parte da imagem possa ser compensada pelo esqueleto.

Em segundo lugar, o ganho sensível de acurácia quando utilizamos Transferência de Aprendizado, carregando os parâmetros do bloco convolucional treinado no caso estático, sobre as redes dinâmicas, oferece uma perspectiva interessante no uso dessa técnica. O maior benefício dela tem sido utilizar uma rede que já foi treinada exaustivamente para reconhecer padrões específicos, diminuindo o tempo de treinamento, mas agora também podemos vê-la como uma oportunidade de trazer filtros relevantes de uma rede para outra.

A pergunta natural nesse momento é: como a rede estática foi capaz de treinar um bloco convolucional com tão mais eficiência em diferenciar poses de mão? Um fator importante nesse sentido é o fato das redes dinâmicas serem mais complexas, possuindo um maior número de parâmetros para serem treinados.

Atentamos também ao fato do caso estático possuir uma variedade maior de exemplos para avaliar. São um total de 19 classes, com grandes variações entre cada um deles. No caso dinâmico, possuímos apenas 7, e desses alguns são idênticos, como os primeiros frames de K e H, e outros bastante semelhantes, como K e W. A maior variedade do primeiro caso incita a rede a procurar por uma gama de características mais ampla, que também apresenta bons resultados quando aplicada sobre os gestos dinâmicos, que são diferentes, porém, essencialmente imagens do mesmo tipo.

E em terceiro lugar, o algoritmo proposto para o reconhecimento de gestos online mostrou-se robusto, capaz de detectar o início e o fim da execução de cada gesto com uma margem de erro de, em média, 5 quadros, que correspondem a uma pequena fração de segundo, ao mesmo tempo que apresenta poucas confusões e poucos falsos positivos.

Naturalmente, essa é apenas a primeira versão do algoritmo, que ainda pode ser aprimorado. A maior dificuldade que encontramos nessa etapa foi, talvez, o fato da biblioteca que utilizamos para implementar a rede neural não possuir uma função de erro apta a lidar com o conceito de múltiplas classes associadas à um mesmo exemplo. De fato,

do seu ponto de vista, o conceito de classe é ambíguo nesse caso.

A consequência imediata disso é que durante o treinamento da rede, a função de erro não avalia corretamente o erro dos exemplos. Mais especificamente, poderíamos dizer que a função de erro não reflete o conceito de erro que nós temos em relação aos exemplos. Naturalmente, uma vez que a função de erro não esteja apropriadamente definida, todo o processo de gradiente descendente se torna comprometido.

Uma função de erro adequada garantiria um treinamento mais eficiente, que garantia parâmetros mais calibrados, que tornaria a flutuação das probabilidades no caso online ainda mais precisas.

Por isso, em nossos trabalhos futuros, pretendemos definir o comportamento dessa função de erro, e implementá-la como uma função de erro customizada dentro da biblioteca de redes neurais que usamos atualmente.

Ainda mais, durante o nosso algoritmo de reconhecimento de gestos online, encontramos uma interessante interação entre os classificadores Offline e Online, onde o segundo segmenta um determinado bloco de quadros, e o primeiro efetivamente o classifica. Esse processo apresentou bons resultados em nossos testes iniciais, mas certamente pode ser avaliado mais profundamente.

6. Referências

1 CORPORATION, M. **Meet Kinect for Windows**. 2017. Disponível em: <https://developer.microsoft.com/en-us/windows/kinect>. Citado na página 13.

2 CORPORATION, I. **Intel RealSense Technology**. 2017. Disponível em: <https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html>. Citado na página 14.

3 MOTION, I. L. **Leap Motion For Mac and PC**. 2015. Disponível em: <https://www.leapmotion.com/product/desktop>. Citado na página 14.

4 LAVIOLA, J. J. 3D gestural interaction: The state of the field. **International Scholarly Research Notices**, Hindawi Publishing Corporation, 2013. Disponível em: <https://www.hindawi.com/journals/isrn/2013/514641/>. Citado na página 16.

5 CHEN, Q. et al. Interacting with digital signage using hand gestures. In: SPRINGER. **International Conference Image Analysis and Recognition**. 2009. p. 347–358. Disponível em: <https://link.springer.com/chapter/10.1007/978-3-642-02611-9_35>. Citado na página 16.

6 KUMAR, P.; VERMA, J.; PRASAD, S. Hand data glove: A wearable real-time device for human-computer interaction. **International Journal of Advanced Science and Technology**, v. 43, 2012. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.261.192&rep=rep1&type=pdf>. Citado na página 16.

7 REN, Z. et al. Robust part-based hand gesture recognition using kinect sensor. **IEEE Transactions on Multimedia**, IEEE, v. 15, n. 5, p. 1110–1120, 2013. Disponível em: <https://ieeexplore.ieee.org/document/6470686>. Citado na página 16.

8 SILVA, S. R. **Reconhecimento de gestos customizados da mão em tempo real usando aprendizado de métricas e grafos de ação**. Dissertação (Mestrado) — Universidade Federal de Alagoas, 9 2017. Citado 3 vezes nas páginas 16, 55 e 56.

9 MARIN, G.; DOMINIO, F.; ZANUTTIGH, P. Hand gesture recognition with leap motion and kinect devices. In: IEEE. **2014 IEEE International Conference on Image Processing (ICIP)**. 2014. p. 1565–1569. Disponível em: <https://ieeexplore.ieee.org/document/7025313>. Citado na página 16.

10 LU, W.; TONG, Z.; CHU, J. Dynamic hand gesture recognition with leap motion controller. **IEEE Signal Processing Letters**, IEEE, v. 23, n. 9, p. 1188–1192, 2016. Disponível em: <https://ieeexplore.ieee.org/document/7509645>. Citado na página 16.

11 SCHMIDT, T. et al. Real-time hand gesture recognition based on sparse positional data. In: **Proceedings of WVC 2014, Brazilian Workshop on Computer Vision**. [s.n.], 2014. p. 243–248. Disponível em: <https://homepages.dcc.ufmg.br/~erickson/publications/schmidt_wvc2014.pdf>. Citado na página 16.

12 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Computation**, v. 9, n. 8, p. 1735–1780, 11 1997. Https://dl.acm.org/citation.cfm?id=1246450. Citado 2 vezes nas páginas 16 e 40.

13 GRAVES, A.; JAITLY, N. Towards end-to-end speech recognition with recurrent neural networks. **ICML'14 Proceedings of the 31st International Conference on International Conference on Machine Learning**, v. 32, 6 2014. Https://dl.acm.org/citation.cfm?id=3045089. Citado na página 16.

14 SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. **CoRR**, abs/1409.3215, 2014. Disponível em: <http://arxiv.org/abs/1409.3215>. Citado na página 16.

15 CHO, K. et al. On the properties of neural machine translation: Encoder-decoder approaches. **CoRR**, abs/1409.1259, 2014. Disponível em: <http://arxiv.org/abs/1409.1259>. Citado na página 16.

16 DONAHUE, J. et al. Long-term recurrent convolutional networks for visual recognition and description. **CoRR**, abs/1411.4389, 2014. Disponível em: <http://arxiv.org/abs/1411.4389>. Citado 2 vezes nas páginas 17 e 43.

17 JOHN, V. et al. Real-time hand posture and gesture-based touchless automotive user interface using deep learning. In: IEEE. **Intelligent Vehicles Symposium (IV), 2017 IEEE**. 2017. p. 869–874. Disponível em: <https://ieeexplore.ieee.org/document/7995825>. Citado na página 17.

18 MIRANDA, L. et al. Real-time gesture recognition from depth data through key poses learning and decision forests. In: **Proceedings of the 2012 25th SIBGRAPI Conference on Graphics, Patterns and Images**. Washington, DC, USA: IEEE Computer Society, 2012. (SIBGRAPI '12), p. 268–275. ISBN 978-0-7695-4829-6. Disponível em: <https://ieeexplore.ieee.org/document/6382766>. Citado na página 17.

19 VIEIRA, T. et al. Online human moves recognition through discriminative key poses and speed-aware action graphs. **Machine Vision and Applications**, v. 28, n. 1, p. 185–200, 2017. Disponível em: <https://link.springer.com/article/10.1007/s00138-016-0818-y>. Citado na página 17.

20 NIELSEN, M. **Neural Networks and Deep Learning**. Determination Press, 2015. Disponível em: <http://neuralnetworksanddeeplearning.com/index.html>. Citado 2 vezes nas páginas 23 e 57.

21 HUI, J. **Convolutional neural networks (CNN) tutorial**. 2017. Disponível em: <https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/>. Citado na página 38.

22 VANI, L.; REDDY, R. A. **Robotic Arm Manipulation Using Leap Motion Controller**. 2015. Disponível em: <https://pdfs.semanticscholar.org/19f6/829c3ed3bca54c5ed12ecba7e856080d6364.pdf>. Citado na página 46.

23 COLGAN, A. **How Does the Leap Motion Controller Work?** 2014. Disponível em: <http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/>. Citado na página 46.

24 MOTION, I. L. **API Overview**. 2017. Disponível em: <https://developer-archive.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html>. Citado 4 vezes nas páginas 47, 48, 49 e 51.

25 RUDER, S. An overview of gradient descent optimization algorithms. **CoRR**, abs/1609.04747, 2016. Disponível em: <http://arxiv.org/abs/1609.04747>. Citado na página 57.

26 SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>. Citado na página 58.