UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Diogo Felipe Félix de Melo

Aprendizado profundo com capacidade computacional reduzida: uma aplicação à quebra de CAPTCHAs.

Recife

Julho de 2018

Diogo Felipe Félix de Melo

Aprendizado profundo com capacidade computacional reduzida: uma aplicação à quebra de CAPTCHAs.

Monografia apresentada ao Curso de Bacharelado em Ciências da Camputação da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Bacharel em Ciências da Camputação.

Universidade Federal Rural de Pernambuco – UFRPE

Departamento de Computação

Bacharelado em Ciências da Camputação

Orientador: Pablo de Azevedo Sampaio

Recife

Julho de 2018

Agradecimentos

Meus pais, familiares e amigos.

Ao meu orientador, por toda a paciência e dedicação.

Resumo

Na última década, Redes Neurais Profundas tem se mostrado uma poderosa técnica de aprendizado de máquina. Em geral, essas técnicas demandam alto poder computacional e grandes volumes de dados para obter resultados expressivos. Entretanto, o projeto cuidadoso da arquitetura e do treino podem ajudar a reduzir estes requisitos. Neste trabalho apresentamos uma abordagem comparativa para a aplicação de redes neurais profundas à quebra de CAPTCHAs de texto. Estudamos modelos capazes de aprender a segmentar e identificar o texto contido em imagens baseando-se apenas em exemplos. A partir da experimentação de diferentes hiper-parâmetros e arquiteturas, fomos capazes de obter um modelo final com acurácia de 96.06% de acerto por token em aproximadamente 3 horas de treino executado em um simples computador pessoal.

Palavras-chave: Aprendizado de Maquina, Aprendizado Profundo, CAPTCHA.

Abstract

During the last decade, Deep Neural Networks has been shown to be a powerfull machine learn technique. Generally, to obtain relevant results, these techniques require high computacional power and large volumes of data. Neverthless, a careful project of trainig and archtecture may help to reduce these requirements. In the this work we present a comparative approach to the application of deep neural networks to text based CAPTCHAs. We studied models that are capable of learn to segment and identify the text content of images, only based on examples. By experimentation of different hiper-parameters and architectures, we were capable to obtain a final model with 96.06% of token prediction accuracy in approximately 3 hours of training in a simple personal computer.

Keywords: Machine Learning, Deep Learning, CAPTCHA.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CAPTCHA	Completely Automated Public Turing tests to tell Computers and Humans Apart. Testes automatizados de Turing para diferenciar humanos e computadores.
OCR	Optical Character Recognition. Reconhecimento do caractere em uma imagem.
RGB	Red-Green-Blue. Canais de codificação de cores.
RAM	Random Access Memory. Memória de Acesso Aleatório.
HIP	Human Interaction Proofs. Provas de interação humana.

Lista de símbolos

i, j, k, ...	Índices.
I, J, K, ...	Conjunto de todos os valores dos índices i, j, k, ... $\kappa = (I, J, K, \ldots)$ é uma coleção de índices.
$\mathbf{x}$	Um vetor.
x_i	Coordenada i do vetor $\mathbf{x}$.
$\mathbf{T}^{\kappa}$	Um tensor com índices na coleção κ. O tamanho da coleção define o ranque do tensor. Sendo ranque 1 vetores, 2 matrizes, etc. Quando definido no contexto, omitiremos κ.
$T_{i,j,k,\ldots}$	O elemento $i, j, k, \ldots$ do tensor $\mathbf{T}$.
$\{\ldots\}$	Conjunto de todas as possibilidades de um variável. Usualmente um espaço vetorial.
$\langle\ldots\rangle_D$	Valor esperado de uma variável no conjunto D.
$\lvert D\rvert$	Número de elementos no conjunto D.
$\lvert\mathbf{T}\rvert_p$	Norma p do tensor $\mathbf{T}$. Quando omitido, $p = 2$ (norma euclidiana).
$\nabla_{\mathbf{T}}$	Operador gradiente com respeito ao tensor $\mathbf{T}$. Isto é, as derivadas em cada elemento de $\mathbf{T}$.
$\Re$	Conjunto dos números reais.
$\Re^{\kappa}$	Espaço dos tensores de ranque $\mathbf{T}^{\kappa}$ sob o conjunto dos números reais. Isto é, $T_{i,j,k,\ldots} \in \Re$.
$\Re^{\kappa}[0, 1]$	Compacto dos tensores $\mathbf{T}^{\kappa}$ sob o conjunto $[a, b]$, com $a, b \in \Re$.
2^D	Conjunto de todos os subconjuntos de D.
$\mathtt{x}$	Variável aleatória.
$P(\mathtt{x} = A)$	Probabilidade do evento $\mathtt{x} = a$ ocorrer. Quando estiver claro no contexto, utilizamos apenas $P(a)$

Sumário

1 Introdução

Algoritmos de aprendizado baseados em neurologia são conhecidos desde meados do século passado (1). Das proposições iniciais até os dias de hoje, essa classe de modelos tem evoluído em complexidade e técnicas de forma contínua, culminando em um alto poder de expressividade e níveis cada vez mais abstratos de representação (ver (2) ou (3) para uma breve revisão histórica). Os poucos resultados teóricos disponíveis demonstram que redes neurais possuem um alto poder de generalização, sendo capaz de, sob certas circunstâncias, codificar diversas classes de funções (4, 5). Apesar dos avanços na área, foi apenas recentemente que modelos neurais começaram a redefinir o estado da arte, superando outras classes de algoritmos de aprendizado de máquina (6) e até mesmo alcançando performances sobre humanas (7). Tais avanços foram possíveis devido a três fatores chaves: a viabilização de bases de dados cada vez maiores, o aumento do poder computacional e o desenvolvimento de novas arquiteturas e técnicas de treino.

A crescente melhoria de performance dos modelos neurais de aprendizado profundo tem motivado estudos em áreas onde se é preciso distinguir computadores e humanos. Dentre essas áreas temos os CAPTCHAs (8) (do inglês Completely Automated Public Turing tests to tell Computers and Humans Apart) ou HIPs (10) (do inglês Human Interaction Proofs), que definem uma coleção de técnicas que tem como objetivo bloquear a ação de agentes autônomos na rede mundial de computadores. Um dos subconjuntos mais conhecidos dessas técnicas talvez seja o de CAPTCHAs baseados em texto (9). Nesse tipo de desafio, uma imagem contendo uma sequência de caracteres é exibida e a validação é feita pela comparação entre o texto informado pelo usuário e a resposta correta. Formulado como um problema de aprendizado de máquina, desejamos descobrir de forma automatizada um mapa entre a imagem e o texto codificado. Na versão informada do problema, um ser humano escolhe previamente técnicas de preprocessamento (filtros, segmentação de caracteres, etc.) antes que o aprendizado propriamente dito ocorra. Ajudados por humanos, redes neurais simples e com poucos exemplos conseguem resultados satisfatórios nesse tipo de desafio (10). De fato, mesmo técnicas ingênuas como contagem de pixels podem obter bons resultados quando o preprocessamento correto é fornecido (11). Na versão não informada, entretanto, encontrar mapas imagem-texto de forma automatizada é usualmente muito mais desafiador. Em trabalhos recentes, foram relatados modelos baseados em redes neurais capazes de burlar esse tipo de desafio com acurácias de acerto próximos à humana em sequências sorteadas a partir de um repositório (12) e modelos com alta eficiência de dados (13). Para o problema geral de quebrar CAPTCHAs baseados em texto, entretanto, modelos de aprendizado profundo ainda mostram desempenho inferior ao humano. Contudo, pesquisas recentes apontam para avanços claros nos próximos anos (14). Em comum, esses

modelos possuem a necessidade de clusters e/ou sistemas de computação sob demanda para treinamento, com hardware de alto poder de processamento e/ou paralelização, como GPUs e TPUs. Adicionalmente, as bases de treinamento necessárias comumente alcançam alguns terabytes e envolvem grandes operações de aquisição e/ou geração.

Neste trabalho propomos uma abordagem comparativa entre diferentes arquiteturas de redes neurais para a solução de CAPTCHAs baseados em texto sem informação humana, nos restringindo, entretanto, à um ambiente com poder computacional reduzido. Pretendemos mostrar que é possível fazer uso dessas técnicas em um mero computador pessoal (na contra-mão dos trabalhos usualmente encontrados na literatura) e ainda obter resultados próximos ao estado da arte. Este trabalho se encontra organizado como segue. No capítulo 2 apresentamos uma breve introdução à diferentes tipos de CAPTCHAs, com ênfase em desafios baseados em texto. Sequencialmente, no capítulo 3, arquiteturas e técnicas de projeto e treino de redes neurais comuns na literatura são abordados. Os principais resultados do uso dessas técnicas em CAPTCHAs de texto explorados e nossas considerações iniciais sobre essa aplicação apresentadas. No capítulo 4 uma descrição das arquiteturas dos modelos usados neste estudo é feita em conjunto com uma breve fundamentação para as escolhas. No capítulo 5, detalhes dos experimentos realizados são formalizados. Por fim, no capítulo 6 os resultados dos experimentos são comparados e no capítulo 7 nossas conclusões e considerações finais apresentadas.

2 CAPTCHAs

Neste capítulo abordaremos como funcionam os principais tipos de CAPTCHA conhecidos. Daremos um enfoque especial aos CAPTCHAs baseados em texto, objeto de estudo do presente trabalho. Na secção 2.2 darem uma definição mais precisa para este HIP em particular.

2.1 Introdução

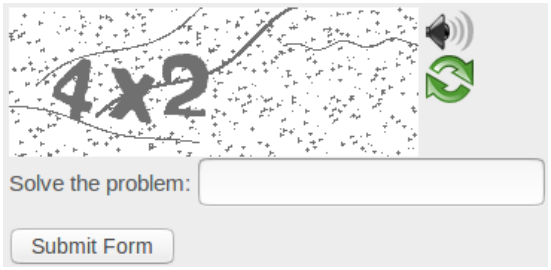
CAPTCHAs (8) ou HIP (10), são um conjunto de técnicas que tem como objetivo discernir a ação automatizada de robôs da ação de seres humanos na Rede Mundial de Computadores. Esses filtros tem sido usados de forma efetiva na defesa de diversos tipos de ataque: proteger informações sensíveis, como *e-mail* e dados pessoais; impedir tentativas de *login* automatizados; coibir acesso massivo à sistemas de bases de dados; entre outros. Entretanto, desde as primeiras aplicações até os dias de hoje, existe uma corrida co-evolucionária entre atacantes e defensores. Por um lado, algoritmos de '*quebra*' de CAPTCHA se tornam cada vez mais sofisticados e precisos. Por outro lado, filtros mais complexos são desenvolvidos. Contudo, como explicado por Chellapilla *Et al.* (10), existe um balanço entre complexidade e factibilidade que os defensores devem buscar, explorando habilidades em que humanos ainda não foram ultrapassados por máquinas. O autor também estima que os requisitos mínimos de um HIP para ser considerado efetivo é ser solúvel por humanos 90% das vezes e é considerado '*quebrado*' se puder ser enganado por robôs em mais do que 1% das tentativas, sendo esses valores dependentes do custo de repetição do ataque.

De forma geral, esses filtros podem ser formulados como um desafio sobre um conjunto de domínio cuja a resposta é um token. O domínio pode ser um trecho de áudio, uma sequencia de imagens ou até mesmo o histórico de navegação do desafiado. O token pode ser constituído de um conjunto de ações, o texto extraído de um áudio ou imagem, ou possuir um histórico de navegação confiável. Podem ainda ser constituídos de uma única etapa ou de varias. Uma categorização dos tipos de CAPTCHA pode ser encontrado em (15). Na figura 1 podemos ver diferentes tipos de CAPTCHAs gerados com a biblioteca de código aberto *Securimage* (16).

O processo de '*quebra*' envolve duas ideias principais: explorar vulnerabilidades e/ou uso de algoritmos inteligentes. Por serem testes automatizados, CAPTCHAs geralmente apresentam algum padrão de comportamento ou falha de projeto. A padronização na geração de desafios pode permitir que um atacante desenvolva heurísticas de ataque. Imagens com mesmo espaçamento de caracteres ou padrões repetitivos podem ser explora-

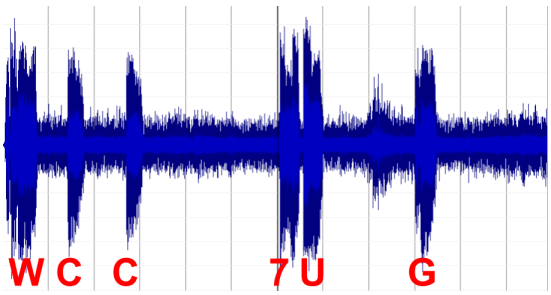
Figura 1 – Diferentes tipos de CAPTCHAs

(a) Letras



(b) Desafio matemático

(c) Duas palavras



(d) Áudio

(a) O desafiado deve reconhecer corretamente um texto formado por caracteres não correlacionados. (b) O desafio consiste em extrair e resolver corretamente um problema simples de álgebra. (c) Identificar palavras sorteadas de um repositório. (d) O desafio consiste com extrair corretamente um conjunto de símbolos codificados em áudio. Todos os exemplos foram gerados pelo autor utilizando a biblioteca Securimage. O espectro de intensidades em (d) foi gerado pelo autor a partir do áudio de um CAPTCHA gerado pela biblioteca.

dos e facilitar a segmentação da imagem, como foi explorado por (11). Falhas ou vieses no projeto desses métodos também podem expor brechas. Quanto ao uso de algoritmos inteligentes, redes neurais recebem um lugar de destaque devido a flexibilidade e capacidade de generalização que esses modelos conseguem alcançar. O uso dessa classe de algoritmos foi a abordagem escolhida por (12) e (10), por exemplo. Redes neurais são exploradas no capítulo 3.

Ao longo dos anos os domínio e os desafios tem evoluído em complexidade, na medida que os atacantes evoluem em técnicas de ataque. Um exemplo da co-evolução desses sistemas é a forma como o sistema reCaptcha (17) tem mudado ao longo dos anos. Quando proposto, o sistema era baseado em trechos de livros e jornais antigos em que os melhores algoritmos conhecidos à época falharam em uma tarefa de OCR (do inglês optical character recognition). Trechos que não haviam sido corretamente identificados (teste) eram separados e exibidos para humanos juntamente com imagens de trechos onde o reconhecimento havia falhado (controle). Teste e controle eram comparados para certificar a atuação humana. Quando proposto, os autores alegaram que humanos seriam capazes de resolver o desafio em quase todos os casos e que comutadores teriam chance

quase nula de passarem desapercebidos, já que o repositório de exemplos era composta por imagens em que as técnicas vigentes à época haviam falhado. Porém, poucos anos depois, avanços em técnicas de OCR baseadas em redes neurais obtiveram 99% de precisão na base de palavras utilizada por esse sistema (12), inviabilizando o uso dessa técnica e forçando o reCaptcha a evoluir para uma segunda versão. Nessa nova abordagem, os dados de navegação dos usuários são analisados por um algoritmo de risco. Caso uma pontuação mínima seja atingida, o usuário é considerado humano, caso contrário, é exposto a problemas conhecidamente difíceis para computadores, como reconhecimento de objetos, contextualização de imagens e busca de similaridades, combinados com diferentes ações que devem ser performadas pelo usuário. Entretanto, Sivakorn *Et al.* (18) mostraram que explorando-se os critérios de avaliação de risco, seria possível enganar o sistema em 85% das vezes, forçando a constante atualização dos desafios.

2.2 Captchas de texto

Nos CAPTCHAs baseados em texto, uma imagem contendo uma sequência de caracteres é exibida ao desafiado. O teste consiste em conseguir recuperar corretamente o texto codificado na imagem. Matematicamente, uma imagem com altura A, largura L e C canais pode ser representada como um tensor $\mathbf{X} \in \Re^{A,L,C}[0,1]$, de modo que X_{ijk} representa a intensidade do pixel na posição (i,j) canal k. Um token é uma sequencia u sob um alfabeto de símbolos Σ, podendo o comprimento da sequência ser limitado ou não. Assim, '*quebrar*' um CAPTCHA de texo é encontrar um mapa $f(\mathbf{X}) \rightarrow u$ que obtenha uma taxa de acerto maior do que a definida por (10). Um sistema de CAPTCHA torna-se completamente inútil se não conseguir diferenciar humanos de robores.

Quanto à imagem, usualmente são utilizadas adição de ruído, linhas e/ou grades para dificultar o processo de segmentação de caracteres, bem como efeitos de distorção, corrosão e/ou dilatação, e transformações geométricas como rotação e translação, entre outros. Em desafios mais simplórios, o número de canais pode ser reduzido, podendo a imagem ser representada como em tons de cinza. Em desafios mais complexos, o espaços de canais é explorado, aumentando algebricamente as possibilidades representação, mantendo, todavia, a factibilidade do desafio para seres humanos, dada nossa facilidade em distinguir cores[1]. É possível explorar imagens sintéticas, geradas de forma automatizada por computadores, ou imagens naturais, como paisagens ou textos em fotos reais.

Quanto ao texto, diferentes complexidades podem ser adicionadas. Fixadas as demais variáveis, a dificuldade do desafio é usualmente proporcional ao tamanho de

[1] Nem sempre esta afirmação é verdadeira. De fato, pessoas que sofrem da Síndrome de Dalton, o Daltoninsmo, podem ter dificuldades em resolver desafios que explorem diferentes cores. O balanço entre complexidade e inclusão de pessoas com necessidades especiais ainda é uma questão em aberto no projeto de CAPTCHAs.

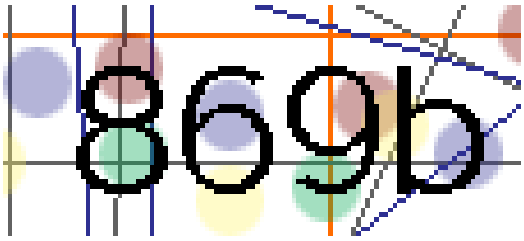
alfabeto escolhido. Dentre os mais simples, o conjunto dos números $\Sigma = 0123456789$ possui apenas dez símbolos. No outro extremo, os logogramas chineses constituem alfabetos com um número indeterminado de símbolos, com as representações mais usuais se limitando a algo em torno de 3000 caracteres. A tipografia também interfere na dificuldade de resolver um CAPTCHA. Para o mesmo alfabeto Σ existem diferentes possibilidades de representação gráfica dos seus caracteres. Fontes regulares tendem a fornecer representações mais simples, com a desvantagem de serem facilmente reconhecíveis por OCR. No outro extremo, fontes simbólicas e caracteres escritos à mão podem representar um grande desafio para máquinas. Quando os símbolos são sorteados de forma aleatória, humanos e máquinas, tipicamente, presenciam maiores dificuldades, dado que cada entrada deve ser reconhecida separadamente (errar o reconhecimento um símbolo significa uma falha completa). Quando sorteamos palavras a partir de um repositório, o desafio se torna mais amigável para humanos, primeiro por termos uma facilidade maior em reconhecer padrões do que especifidades e segundo por nos permitir o uso indireto de outras fontes de conhecimento para a solução do problema. Se soubermos que os textos estão em Português, por exemplo, podemos esperar que nenhuma palavra conterá uma subsequência *tt* (gramaticalmente incorreto) ou que uma vogal será, provavelmente, seguida por uma consoante ou por uma vogal diferente (padrão mais frequente na língua). Entretanto, quando o repositório de palavras é exposto, atacantes podem desenvolver heurísticas para explorar o problema, como um dicionário de palavras ou correção gramatical para aprimorar a eficiência das técnicas. Além disso, repositórios de palavras induzem correlações entre os caracteres, o que limita bastante o espaço de possíveis tokens. Ver (19) para um apanhado de CAPTCHAs de texto utilizados pelo mundo e (9) para uma a evolução desses sistemas ao longo dos anos. Na figura 2 temos alguns exemplos utilizados por instituições brasileiras.

Figura 2 – Diferentes CPATCHAs de texto em uso por instituições brasileiras.

(a) Banco Central do Brasil

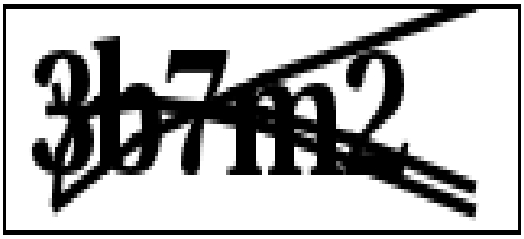
(b) Instituto Nacional do Seguro Social (INSS)

(c) Banco Itaú



(d) Ministério da Educação (MEC)

(e) Ministério Público do Estado do Rio de Janeiro (MPRJ)



(f) Tribunal de Justiça do Estado de Pernambuco (TJPE)

(a) Disponível em: <https://www3.bcb.gov.br/faleconosco/#/login/consultar>. Acesso em: 23/06/201. (b) Disponível em: <https://cadsenha.inss.gov.br/cadsenha-internet-web/faces/pages/segurado/entradaNitNb.xhtml>. Acesso em: 23/06/201. (c) Disponível em: <https://bankline.itau.com.br/boletos-novosite/simulador_etapas_boletos_02.htm>. Acesso em: 23/06/201. (d) Disponível em: <http://painel.mec.gov.br/painel/captcha>. Acesso em: 23/06/201. (e) Disponível em: <http://www.mprj.mp.br/comunicacao/ouvidoria/consulte-a-sua-comunicacao>. Acesso em: 23/06/201. (f) Disponível em: <https://srv01.tjpe.jus.br/consultaprocessualunificada/>. Acesso em: 23/06/201.

3 Redes Neurais

Neste capítulo introduziremos redes neurais como uma forma genérica para escrever famílias de funções. Os conceitos de composição, tipos de camadas e arquiteturas são introduzidos. Daremos um enfoque aos dois tipos de camadas que foram usados no presente trabalho. Posteriormente, detalhes técnicos do treino de redes neurais são explorados. Por fim, apresentamos trabalhos selecionados da literatura que utilizaram redes neurais para a quebra de CAPTCHAs de texto.

3.1 Introdução

Dentro do campo de Inteligência Artificial, aprendizado de máquina é uma abordagem algorítmica para inferir regras em um problema a partir de exemplos. Uma categoria especial é a de aprendizado supervisionado, onde os exemplos constituem-se de um *elemento* em um domínio conhecido e um *rótulo* associado. O objetivo é deduzir abstrações que permitam relacionar elementos e rótulos. Nesse contexto, uma rede neural[1] é, em última análise, uma forma genérica de escrever funções[2] sobre relações elemento-rótulo. Dada uma métrica para a estimativa dos erros cometidos por uma relação inferida, o desafio de inferir regras de associação ótimas se torna um problema de encontrar uma função que minimiza nossa estimativa de erro. O adjetivo '*neural*' advém da inspiração em funções biológicas que historicamente influenciaram e ainda influenciam essa forma de exprimir relações. Em resumo, redes neurais são um conjunto de técnicas inspiradas em processos cognitivos desempenhados pelo sistema nervoso que fornecem uma maneira genérica de descrever famílias de funções.

De forma mais específica, dado um conjunto de exemplos $D = \{(x, y)\}$, onde x pertence algum domínio conhecido e y um rótulo associado, desejamos encontrar a função $\hat{y} = f(x)$, de tal modo que $\hat{y}$ seja o mais similar possível à y. Por '*mais similar o possível*' entende-se que conhecemos uma estimativa para os erros, também referida como função custo, que é tão menor quanto melhor for a aproximação dada por $f(x)$, e é normalmente representada como $J(y, \hat{y})$. Formalmente, desejamos encontrar f^* tal que

$$f^* = \arg\min_{\{f\}} J^{(D)} = \arg\min_{\{f\}} \langle J(y, f(x)) \rangle_D, \tag{3.1}$$

onde $\langle \ldots \rangle_D$ representa o valor esperado no conjunto D.

[1] Apesar de introduzidas aqui como um algoritmo supervisionado, redes neurais podem ser aplicadas de forma efetiva à problemas não supervisionados.

[2] Funções estão sendo usadas aqui com um sentido mais relaxado do que o usualmente utilizado na matemática.

Dada uma família de funções $f^{\mathbf{\Theta}} : x \to y$ definida por uma rede neural e parametrizadas por $\mathbf{\Theta}$, podemos vasculhar o espaço de busca induzido, $\{\mathbf{\Theta}\}$, para encontrar um função que satisfaça alguma propriedade de interesse. Em particular, no caso de aprendizado de máquina, estamos interessados em encontrar o parâmetro $\mathbf{\Theta}^*$ tal que:

$$\mathbf{\Theta}^* = \arg\min_{\{\mathbf{\Theta}\}} \langle J(y, f^{\mathbf{\Theta}}(x)) \rangle_D. \tag{3.2}$$

A versatilidade desse formalismo nos permite explorar diferentes tipos de estruturas relacionais. Em particular, relações hierárquicas podem ser emuladas utilizando-se composição de funções. Essa abordagem nos permite construir redes neurais mais complexas e expressivas a partir de unidades básicas mais simples. Neste caso, podemos reescrever a função $f^{\mathbf{\Theta}}$ como:

$$f^{\mathbf{\Theta}}(x) = f_1^{\mathbf{\Theta}^{(1)}}(f_2^{\mathbf{\Theta}^{(2)}}(f_3^{\mathbf{\Theta}^{(3)}}(\dots(x)))), \tag{3.3}$$

sendo $\mathbf{\Theta} = (\mathbf{\Theta}^{(1)}, \mathbf{\Theta}^{(2)}, \mathbf{\Theta}^{(3)}, \dots)$ o parâmetro composto por cada um dos parâmetros individuais. Quando compomos funções desta forma, é comum nos referirmos à cada função $f_i^{\mathbf{\Theta}^{(i)}}$ como sendo a *i-ésima camada* da rede neural, sendo as camadas para além da mais externa também conhecidas como *camadas escondias* (em inglês *hidden layers*). A *profundidade* da rede é uma referência à quantidade de funções internas usadas na composição. Diferentes tipos de funções definem diferentes tipos de transformações, as quais nos referimos como *tipo da camada*. A especificação de todas as camadas em uma rede neural é o que chamamos de *arquitetura* da rede. Não existe um consenso sobre o que seria exatamente a profundidade de uma arquitetura, ou a partir de que ponto podemos chamá-la de profunda (2). Neste trabalho vamos adotar a convenção de que a profundidade de um modelo é dado pelo número de camadas e que uma arquitetura é profunda se possuir mais de uma camada escondida. A seguir vamos explorar dois tipos de camadas que foram utilizados no presente estudo.

3.2 Camadas Densas

Camadas *densas* ou totalmente conectadas definem uma transformação afim entre o conjunto de entradas e saídas. Tipicamente, após a transformação afim, segue-se a aplicação de uma função não linear elemento-à-elemento, conhecida como *função de ativação*, permitindo a expressão de relações mais complexas. As camadas densas são biologicamente inspiradas no mecanismo de comunicação dos neurônios, onde a diferença de potencial elétrico exprimida nas sinapses do axônio é proporcional, em alguma medida, à soma das diferenças de potenciais experimentadas nos dendritos (20). As conexões de entrada em um neurônio e sua regra de composição dos sinais definem sua regra de ativação, ou seja, quais outros neurônios devem estar ativos (e em que intensidade) para que haja uma transição de estado. Como veremos mais adiante, camadas totalmente conectadas tentam emular o comportamento de vários neurônios simultaneamente.

De maneira mais formal, seja $\mathbf{x}$ um vetor no conjunto de entrada, a relação expressa por uma camada densa é dada por:

$$f^{\mathbf{W},\mathbf{b}}(\mathbf{x}) = act(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}) \tag{3.4}$$

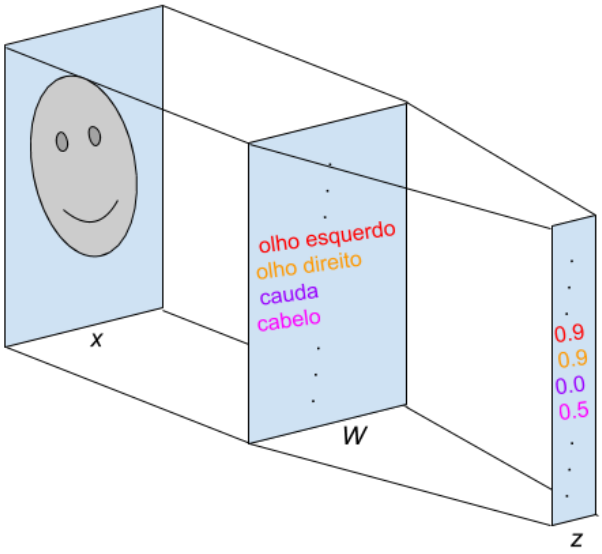
onde $\mathbf{b}$ é referido como *viés* (ou, no inglês, *bias*), $\mathbf{W}$ uma matriz de transformação, *act* uma função de ativação e $\cdot$ é a operação usual de multiplicação de matrizes, definida elemento-à-elemento como $[\mathbf{W} \cdot \mathbf{x}]_i = \sum_k W_{ik} x_k$. Diferentes funções de ativação expressam relações distintas sob um vetor de entrada $\mathbf{z}$. Dentre os exemplos mais conhecidos na literatura, ressaltamos a função sigmoide, $\sigma(\mathbf{z})_i = \frac{1}{1+\exp(-z_i)}$, que mapeia os elementos de saída no intervalo $\Re[0, 1]$, a função de retificação linear (relu), $relu(\mathbf{z})_i = max(0, z_i)$, e a função máximo atenuado (softmax), $\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$, que possui a interessante propriedade $\sum_i \sigma(\mathbf{z})_i = 1$, sendo usualmente utilizada para expressar distribuições de probabilidade.

Podemos interpretar a transformação expressa na equação 3.4 como uma projeção do exemplo $\mathbf{x}$ sob um espaço de características que descrevem uma relação que desejamos expressar. De fato, se reescrevermos a *i-ésima* linha de $\mathbf{W}$ como $\alpha_i \hat{\mathbf{w}}_i$, onde $\hat{\mathbf{w}}_i$ é um vetor normalizado e α_i a constante de normalização, podemos interpretar cada uma das saídas de uma camada densa (antes da função de ativação) como uma série de projeções balanceadas $\alpha_i \hat{\mathbf{w}}_i \cdot \mathbf{x} + b_i$ ($\cdot$ aqui é o produto interno usual). Em outras palavras, a contribuição da *i-ésima* saída da camada é dada pela importância α_i dessa característica multiplicada por o quanto do exemplo é composto por essa característica, $\hat{\mathbf{w}}_i \cdot \mathbf{x}$, adicionado de um viés b_i que independe do exemplo em questão. Assim, para camadas densas, determinar os parâmetros ótimos pode ser entendido como encontrar projeções do espaço de entrada em um conjunto de características que sejam pertinentes ao problema. Neste sentido, cada saída de uma camada densa pode ser interpretada como um neurônio, que exprime em sua saída uma soma ponderada dos estímulos de entrada. Quando compomos camadas densas expressamos características que dependem de outras características. Durante o aprendizado, esperamos que essas projeções sejam descobertas de forma automática pelo modelo. Um desenho esquemático de como essas projeções funcionam pode ser visto na figura 3. O número de parâmetros em uma camada densa é dado pelo produto entre tamanho do exemplo de entrada e a quantidade de características.

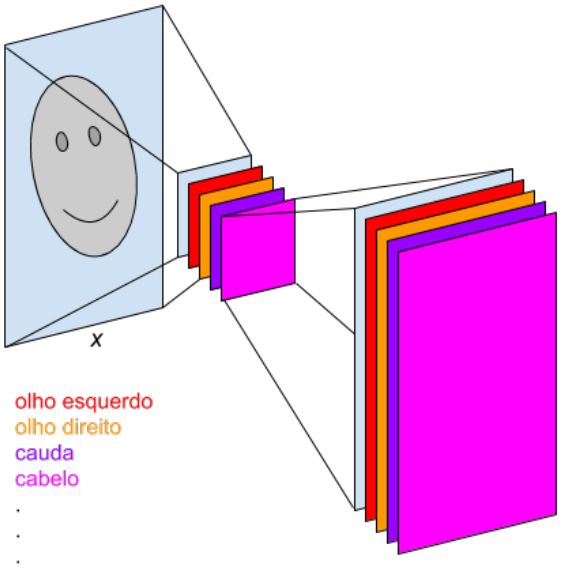
3.3 Camadas Convolucionais

Camadas *convolucionais* (21, 22) possuem similares com as densas: definem uma transformação afim representada por uma matriz, seguida, possivelmente, de uma função de ativação. Diferem, entretanto, no tipo de operação matricial utilizada, trocando-se o produto usual de matrizes por uma operação convolucional. Como vimos na seção anterior, as saídas de uma camada densa podem ser interpretadas como uma coleção de projeções

Figura 3 – Imagem ilustrativa das possíveis projeções aprendidas em uma camada (a) densa e (b) convolucional em um problema de reconhecimento de faces.



(a)



(b)

Em (a) as projeções atuam sobre todo o exemplo de entrada, mapeando-os em um espaço de importâncias de cada característica. Note que as projeções densas são fixas, no sentido que expressamos '*olho direito*' em uma posição específica. Em (b) expressamos projeções menores que '*varrem*' o exemplo de entrada buscando por uma característica em particular, projetando o exemplo em canais de características. Note que, neste caso, procuramos por uma característica que esteja presente em qualquer lugar da imagem, com os sinais de resposta registrados no canal respectivo. (Fonte: elaborado pelo autor.)

independentes do exemplo de entrada sob um espaço de características. Em camadas convolucionais, o mesmo operador projeção é reutilizado em diferentes subespaços do conjunto de domínio, varrendo o mesmo exemplo em busca de padrões específicos em diferentes localizações. Esta peculiaridade é usualmente referida como *compartilhamento de parâmetros*. O reuso promove a vantagem de reduzir consideravelmente o espaço de busca $\{\boldsymbol{\Theta}\}$ (Uma imagem ilustrativa do funcionamento dessas camadas pode ser visto na figura 3). O número de parâmetros nesse caso é dado pelo produto entre o tamanho da projeção e o número características. Camadas convolucionais são especialmente efetivas quando os elementos do domínio possuem alta localidade, como é o caso de imagens (cada pixel é altamente correlacionado com seus vizinhos em imagens naturais) e séries temporais periódicas (o valor da série do tempo t é correlacionado com o valor da série em $t + \tau$, onde τ é o período da série). Tipicamente, mais de uma projeção é encapsulada em um mesmo tensor de transformação, sendo a matriz de cada projeção referida como núcleo. O resultado da convolução do núcleo com o exemplo de entrada é tipicamente referida como *canal*.

Matematicamente, dado um tensor $\mathbf{x}$, a atuação da camada de convolução pode ser escrita como:

$$f^{\mathbf{W},\mathbf{b}}(x) = act(\mathbf{W} \otimes \mathbf{x} + \mathbf{b}) \tag{3.5}$$

onde $\mathbf{W}$ é um tensor de transformação que encapsula a informação de diferentes projeções, $\mathbf{b}$ é o vetor de viés, de dimensão igual ao número de canais de saída, *act* uma função de ativação e $\otimes$ é a operação de convolução, definida elemento-à-elemento por[3]

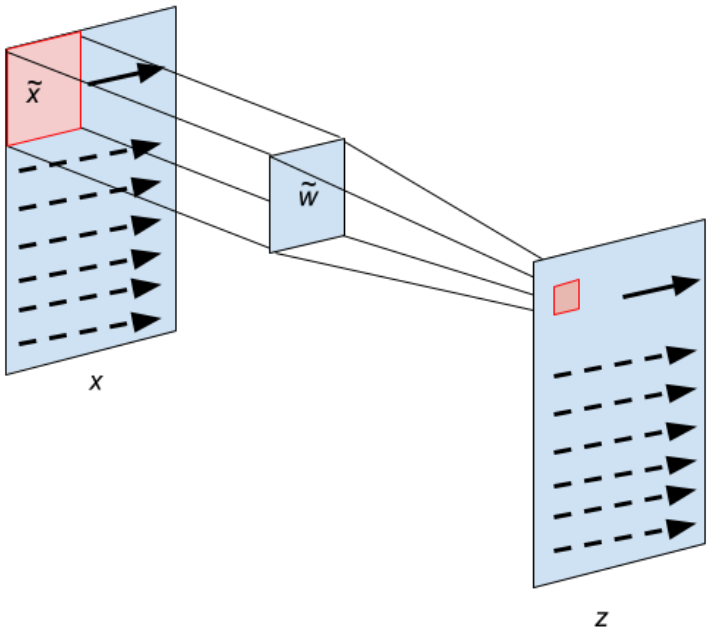
$$[\mathbf{W} \otimes \mathbf{x}]_{ijd} = \sum_{c} \sum_{m=i'-k_i}^{i'+k_i} \sum_{n=j'-k_j}^{j'+k_j} W_{m,n,c,d} x_{m,n,c} \tag{3.6}$$

onde c (d) se estende por todos os canais de entrada (saída), $2k_i + 1$ ($2k_j + 1$) é o tamanho do núcleo da projeção na direção i (j) e a relação $s_i = i'/i$ ($s_j = j'/j$) define o passo da transformação (em geral, $k_i = k_j = k$ e $s_i = s_j = s$). Uma forma intuitiva de visualizar uma operação convolucional é imaginar que a cada etapa (i, j), cada operador de projeção $\tilde{\mathbf{w}}$ no tensor $\mathbf{W}$ define, por canal, uma região $\tilde{\mathbf{x}}$ no tensor de entrada centrada em (i, j). O resultado da operação de convolução é dado pelo produto interno usual entre $\tilde{\mathbf{w}}$ e $\tilde{\mathbf{x}}$. No passo seguinte a operação é repetida em outra região da entrada até que todos os (i, j)'s de entrada sejam cobertos. Um esquema ilustrativo pode ser visto na figura 4. A contribuição total para cada canal de saída é dado pela soma das convoluções nos canais de entrada.

Como visto, a operação de convolução atua mapeando uma região (do tamanho do núcleo) no canal de entrada em um único pixel no canal de saída. Dada a propriedade de localidade, essas representações podem adicionar muita redundância de informação no canal

[3] Por simplicidade, estamos definindo aqui a operação de convolução para imagens coloridas, representadas por tensores de ranque 3. Entretanto, operações de convolução podem ser definidas para outros domínios, como séries temporais (ranque 1), imagens em tons de cinza (ranque 2) e vídeo (ranque 4).

Figura 4 – Exemplo esquemático da execução da operação de convolução em um canal do tensor de entrada x.



No passo i, j, cada operador de projeção $\tilde{\mathbf{w}}$ define um campo de atuação $\tilde{\mathbf{x}}$. Para o canal $\mathbf{z}$, $z_{i,j} = \tilde{\mathbf{w}} \cdot \tilde{\mathbf{x}}$. O produto interno é realizado transformando $\tilde{\mathbf{w}}$ em um vetor linha e $\tilde{\mathbf{x}}$ em um vetor coluna. (Fonte: elaborado pelo autor.)

de saída. Uma forma de contornar o problema é adicionando uma operação de agrupamento (em inglês *pooling*) após as ativações da camada. Uma forma simples de realizar um agrupamento é aplicar um mapa entre um conjunto de pixels de entrada em apenas um pixel de saída, reduzindo o tamanho da representação e forçando a rede a encontrar representações mais eficientes entre os níveis de abstração. Escolhas usuais são a média, o valor máximo (*maxpooling*) ou uma posição fixa das ativações de entrada. As operações de agrupamento podem ser vistas como um tipo especial de convolução (possivelmente não linear) com parâmetros fixos. De fato, uma forma eficiente de implementar agrupamento com um valor fixo é escolher um passo maior que 1 na operação de convolução.

Camadas de convolução são inspiradas na interpretação vigente do funcionamento do córtex visual de mamíferos. Diferentes camadas hierárquicas são sensíveis à níveis distintos de abstração na representação de imagens: as iniciais apreendem informações de traços simples, posteriormente composições de traços, formas geométricas, objetos complexos e assim por diante. De fato, estudos mais detalhados das projeções hierárquicas aprendidas após o treino por redes convolucionais profundas mostraram similaridades com o seu análogo biológico (23) e crescentes níveis de abstração na representação das imagens (24, 25), iniciando com traços simples e texturas nas camadas iniciais e culminando em representações abstratas para objetos nas finais.

Frisamos que esta não é uma lista exaustiva dos diferentes tipos de camada que podem ser encontrados na literatura. Não listadas aqui, mas que merecem destaque, podemos citar redes de capsula (26), que desempenham operações de convolução e posterior alinhamento das ativações dos filtros, e redes recorrentes (e suas variações), que adicionam correlação temporal entre os exemplos através de mecanismos de memória. Uma revisão histórica da evolução dessas camadas pode ser encontrado em (3). Para uma descrição mais detalhada das arquiteturas descritas nesse capítulo, incluindo limitações, aplicabilidade e detalhes técnicos, ver (2).

3.4 Aprendizado

Definida a arquitetura da rede neural, isto é, a sequencia das camadas e respectivas funções de ativação, procedemos para o *treino* da rede, que consiste em encontrar os parâmetros ótimos
$mathbfTheta^*$ como definido na equação 3.2. A forma mais ingênua para procurar o valor ótimo seria utilizar o *método do gradiente* (ou método do máximo declive), que consiste em atualizar iterativamente $\mathbf{\Theta}$ na direção contrária à do gradiente da função custo segundo a regra

$$\mathbf{\Theta} \leftarrow \mathbf{\Theta} - l_r \nabla_{\mathbf{\Theta}} J^{(D)}, \tag{3.7}$$

onde l_r é o *hiper-parâmetro de aprendizado* ou taxa de aprendizado, D o conjunto onde a atualização é realizada e $\nabla_{\mathbf{\Theta}}$ o operador gradiente com respeito aos parâmetros $\mathbf{\Theta}$. A evolução de $\mathbf{\Theta}$ ao longo do treino é referida como *dinâmica do aprendizado*.

O método do gradiente nos garante que, uma vez atingido o mínimo global, o gradiente da função custo é nulo ($\nabla_{\mathbf{\Theta}} J^{(D)} = 0$), e nenhuma atualização posterior se fará necessária. Entretanto, não nos é garantido o recíproco: uma vez que $\nabla_{\mathbf{\Theta}} J^{(D)} = 0$, não podemos afirmar se encontramos um mínimo local, um ponto de sela ou até mesmo um ponto de máximo. Também não é garantido se eventualmente um ponto de gradiente nulo será encontrado. Adicionalmente, a escolha do hiper-parâmetro l_r e do conjunto de atualização D não são especificados pelo método.

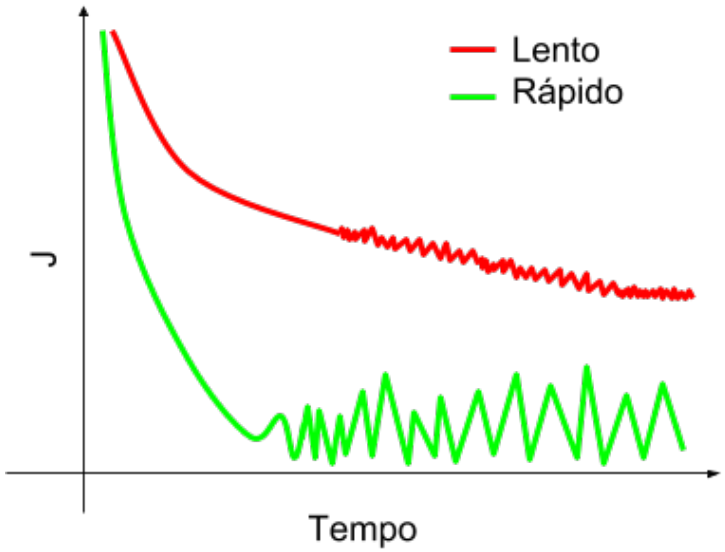
Quanto à escolha do conjunto de atualização, podemos escolher dentre atualizar os parâmetros para cada exemplo, para um subconjunto ou para todos os exemplos disponíveis por vez. No primeiro caso, conhecido como *gradiente estocástico*, temos a vantagem de atualizar $\mathbf{\Theta}$ sempre que a rede é exposta à um exemplo, o que poderia levar a uma convergência mais rápida. Essa escolha é comumente referida como aprendizado *online*. Entretanto, as flutuações estatísticas de cada exemplo podem gerar uma condição de difícil convergência, com o gradiente da função custo mudando drasticamente de direção a cada atualização. No outro extremo, conhecido como *gradiente em lote*, D constitui todo o conjunto disponível de treino. Nessa escolha, o gradiente calculado para cada exemplo

é acumulado e a atualização é feita pela média dos gradientes em todos os exemplos de treino. Apesar da vantagem da estabilidade estatística, corremos dois riscos: média sobre o conjunto de treino pode anular ou diminuir particularidades de subconjuntos específicos ou podemos demorar demais para atualizar $\boldsymbol{\Theta}$, o que torna o aprendizado mais lento. Uma terceira opção consiste em juntar o melhor das duas escolhas anteriores: promover a atualização dos parâmetros à cada N_B exemplos. Tal técnica é denominada atualização em mini-lote ou *gradiente em mini-lote* e foi a técnica escolhida no presente trabalho.

Quanto à escolha de l_r temos que buscar um compromisso entre dois extremos. Valores pequenos produzem dinâmicas lentas, onde as atualizações do parâmetros são pequenas e o tempo de treino se estende por diversas iterações. Em alguns casos, a dinâmica pode estagnar em regiões onde $\nabla_{\boldsymbol{\Theta}} J^{(D)}$ é pequeno mas o valor do custo ainda é alto. No outro extremo, temos uma dinâmica mais rápida, porem mais instável. Pontos de mínimo podem ser completamente ignorados (fenômeno referido como *overshoot*) e a função de erro tende a exibir flutuações que dificultam a convergência. Para valores extremamente altos, a dinâmica pode superestimar os erros cometidos e/ou amplificar demais as flutuações, fazendo o valor do custo aumentar ao invés de diminuir. Uma maneira de contornar o problema é utilizar uma abordagem adaptativa. Uma solução simples é diminuir o valor da taxa de aprendizado ao longo do treino, sendo decaimento linear ou exponencial duas escolhas bastante comuns. Dessa forma, temos um balanço entre rápido aprendizado no início da dinâmica e maior estabilidade próximo ao fim. Outras técnicas incluem estabelecer regras de atualização baseadas no gradiente da função de custo e no momento (taxa de atualização do gradiente). A descrição detalhada dessas técnicas fogem ao escopo deste trabalho. Nos experimentos utilizamos a técnica conhecida como Estimativa Adaptativa do Momento (em inglês, *Adaptive Moment Estimation*), também referida como Adam (27) que foi especialmente desenvolvida para atualizações em mini-lote e utiliza o momento dos gradientes do custo para estabilizar o aprendizado e decaimento da taxa de treino. Um exemplo esquemático de uma dinâmica lenta (l_r pequeno) e rápida (l_r grande) pode ser vista na figura 5.

Para redes compostas por várias camadas, como descrito na equação 3.3, a atualização dos parâmetros torna-se complicada e potencialmente suscetível a erros numéricos. Uma forma eficiente de contornar o problema é utilizar a técnica de propagação reversa (em inglês *back-propagation*). Nesta técnica, a computação realizada pela rede é mapeada em um grafo, sendo cada operação realizada na computação expresso como um nó e o fluxo de dados entre duas operações representado por uma aresta direcionada. Durante a fase direta (*feed-foward*) a computação é executada nó-a-nó e os resultados parciais de cada operação armazenados. Durante a fase reversa, as arestas são invertidas e o erro propagado entre os vértices do grafo reverso utilizando-se os valores previamente armazenados e a regra da cadeia para expressar a dependência entre os erros. Durante essa etapa, cada nó que possui um parâmetro a ser aprendido é atualizado. Ver (3) para uma revisão do

Figura 5 – Exemplo de comportamento característico da função de custo.



Em vermelho o aprendizado é lento e apresenta flutuações menores. Em verde o custo decai rapidamente, mas apresenta flutuações mais drásticas. (Fonte: elaborado pelo autor.)

método.

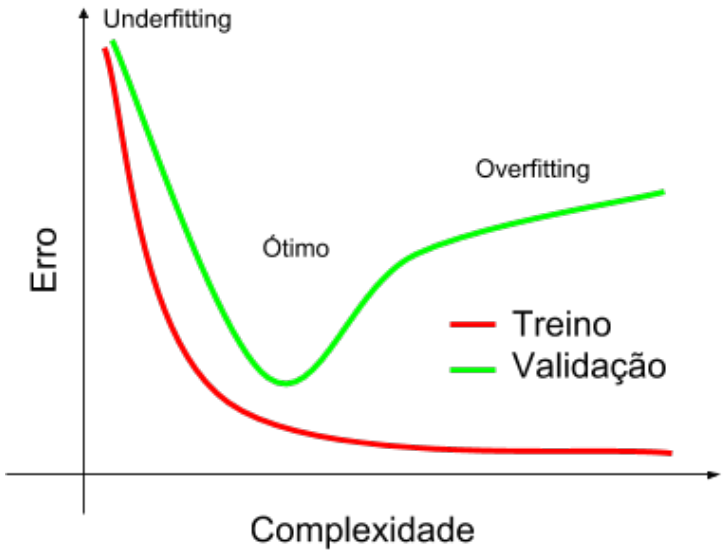
3.5 Regularização

Como visto da seção 3.4, durante a evolução da dinâmica de aprendizado, os parâmetros da rede evoluem de forma a reduzir o valor esperado da função de custo no conjunto de treino. Entretanto, diminuir o erro nos exemplos vistos não garante que o modelo apresentará a mesma qualidade em novos elementos ausentes durante essa fase. De forma mais precisa, seja U o conjunto de todos elemento-rótulo possíveis, tipicamente temos a disposição um subconjunto $D \subset U$ para otimizar a rede, e utilizamos $J^{(D)}$ como um estimador estatístico para $J^{(U)}$, isto é, supondo que D tenha sido formado por instâncias aleatórias de U, teremos $J^{(D)} \to J^{(U)}$ quando $|D| \to |U|$. Tipicamente, entretanto, temos a situação em que $|D| \lll |U|$. Como o modelo foi otimizado neste subconjunto, o valor esperado da função custo em D se torna um estimador enviesado para o erro esperado em exemplos não vistos. Uma maneira simples e eficiente de estimar a capacidade de generalização modelo é criar a cobertura exata (D_{tr}, D_{val}) para D, treinar o modelo em D_{tr} e acessar sua qualidade em D_{val}. Esta técnica é conhecida como *hold-out*, e $J^{(D_{val})}$ uma estimativa para o poder de generalização do modelo (28).

Um outro problema recorrente no treino de modelos de aprendizado de máquina é a adaptação do modelo ao conjunto de treino. Em um extremo, modelos de baixa expressividade podem não conseguir capturar a complexidade das relações exemplo-rótulo.

Esse fenômeno é conhecido como *underfitting*. No outro limite, a medida que aumentamos a expressividade (mais camadas, mais projeções, etc.), aumenta também capacidade de apreender relações mais complexas entre exemplos e rótulos. Entretanto, modelos com maior expressividade tendem a ser mais propensos a '*decorar*' exceções e superestimar particularidades do conjunto de treino. Tal fenômeno é conhecido como coadaptação ou *overfitting*. Outra situação em que podemos incorrer em *overfitting* é quando o modelo é excessivamente exposto aos mesmos exemplos durante uma seção de treino. O *underfitting* pode ser detectado diretamente da qualidade do modelo em D_{tr}, quando seu desempenho se mostra aquém do esperado. O segundo pode ser estimado utilizando a decomposição *viés-variância* da estimativa do erro, onde entendemos que parte do erro cometido é devido a baixa expressividade e parte por superestimar particularidades do conjunto de treino. Para tal, a qualidade do modelo em D_{tr} e sua capacidade de generalização (estimada em D_{val}) são comparadas e a diferença entre as duas servem para estimar a relação entre viés e variância. Um exemplo ilustrativo do comportamento do erro nesses dois conjuntos pode ser visto na figura 6.

Figura 6 – Exemplo ilustrativo de *overfitting* e *underfitting*.



Em verde vemos que o erro estimado no conjunto de treino se comporta como uma função decrescente da complexidade do modelo. Em vermelho, o erro estimado o conjunto de validação mostra que existe um valor ótimo para o poder de generalização. Fixada a complexidade do modelo, experimenta-se um comportamento similar quando aumentamos a exposição aos exemplos no conjunto de treino. (Fonte: elaborado pelo autor.)

Para minimizar o efeito de coadaptação dos parâmetros, podemos impor condições extras durante o processo de aprendizado. Uma das condições mais usuais é aplicar ao vetor de parâmetros $\boldsymbol{\Theta}$ alguma restrição. Podemos, por exemplo, adicionar à função custo

um termo proporcional à norma do vetor $\boldsymbol{\Theta}$, na forma

$$\tilde{J} = J + \lambda \left|\boldsymbol{\Theta}\right|_p , \tag{3.8}$$

onde $\tilde{J}$ é o custo utilizado durante a minimização, $\left|\boldsymbol{\Theta}\right|_p = \sqrt[p]{\boldsymbol{\Theta}^p}$ é a norma p e λ o hiper-parâmetro de regularização ou constante de normalização. Desta forma penalizamos parâmetros que '*aprendam*' demais uma determinada característica. Apesar de efetivo, o uso de regularização baseado em normas adiciona dois novos hiper-parâmetros: a constante de normalização e o tipo de norma que devemos usar. Outra forma de regularização comum na literatura é o *dropout* (29). Nesta técnica, a saída i de uma camada é removida da computação (atribuída o valor zero) com uma probabilidade p, hiper-parâmetro da técnica. Intuitivamente, o uso do *dropout* durante o treino força a rede a encontrar representações redundantes e mais robustas à detalhes, dado que a cada nova iteração uma característica aprendida anteriormente pode estar ausente.

3.6 Redes Neurais e CAPTCHAs

Como visto na secção 2.2, CAPTCHAs de texto podem ser vistos como um problema de extração de texto em imagens, sendo assim uma generalização para o problema de OCR e um subproblema de localização de texto em imagens (identificar, se existir, a localização de todos os textos em uma imagem). É preciso ressaltar, entretanto, que nesses HIPs as imagens são especialmente desenvolvidas para serem de difícil solução para computadores e preferencialmente fáceis para seres humanos. Assim, algoritmos usuais de OCR tendem a demonstrar baixo desempenho na solução desses desafios. Vamos separar as abordagens para o problema em dois grandes grupos: abordagens *informadas* e não *informadas*.

Em abordagens informadas o autor do ataque estuda os efeitos e o alfabeto utilizado na composição do desafio e confecciona um pipeline para preprocessamento e segmentação dos caracteres componentes do token. Esta etapa envolve o uso de heurísticas e é extremamente dependente da regularidade das imagens[4] e da habilidade do atacante para que bons resultados sejam alcançados. Após a segmentação, o desafio se resume à reconhecer corretamente cada símbolo. Um dos trabalhos pioneiros na quebra de CAPTCHAs de texto foi conduzido por (10) e utiliza essa abordagem. Inicialmente as imagens foram preprocessadas e segmentadas. Para o reconhecimento foram usadas redes convolucionais de duas camadas, alcançando acurácias entre 10% e 50% (fortemente dependente da qualidade do preprocessamento). Bursztein *et al.* formalizaram ainda mais o pipeline de processamento, lançando a ferramenta *Decaptcha* (19), que permite explorar e testar etapas do processo de quebra especificando-se técnicas de preprocessamento, segmentação, pos-segmentação, reconhecimento e aprimoramento pós reconhecimento. Com o auxilio

[4] Isso é tipicamente verdade quando a imagem é gerada automaticamente por um computador. Neste caso, técnicas de engenharia reversa podem guiar o desenvolvimento de heurísticas.

dessa ferramenta, os autores relataram modelos com acurácias mais baixas em trono de 20%, chegando 100% de acertos em desafios mais simples. Entretanto, como sugerido pelos próprios autores, na fase de reconhecimento a ferramente utiliza algoritmos com baixo poder de expressividade (k-vizinhos mais próximos e máquinas de vetores de suporte) se comparados com técnicas mais atuais. Se olharmos apenas para o desafio pós segmentação, a extração de texto de CAPTCHAs se resume a um problema de reconhecimento de caracteres, e nesse campo o estado da arte é dominado por redes neurais. De fato, no repositório do MNIST[5] (30), por exemplo, redes convolucionais pontuam entre as melhores taxas de acerto, sendo o melhor modelo registrado o desenvolvido por Cireşan *et al.*, que alcançou 99.77% de acerto e é baseado na média do voto de 35 redes convolucionais independentes (31). Para CAPTCHAs de texto formados por imagens reais, as diversas condições de aquisição (iluminação, ângulo, escala, etc.) degradam as vantagens de um pipeline único de preprocessamento. Mesmo neste caso, redes neurais ainda são capazes de demonstrar bom desempenho. Netzer *et al.* propuseram o repositório SVHN[6] como benchmark para esse tipo de desafio (32). Partindo das imagens já segmentadas, os autores foram capazes de obter precisões em torno de 90% utilizando-se abordagens não supervisionadas de redes neurais. O resultado foi posteriormente refinado utilizando-se redes convolucionais profundas e treino supervisionado, alcançando 94.5% de precisão (33).

Quanto às abordagens não informadas, o modelo deve aprender de forma autônoma como localizar, segmentar e reconhecer os caracteres em uma imagem, minimizando a interferência direta humana. Entretanto, inferir esse nível de abstração apenas baseado em exemplos pode requerer quantidades massivas de dados anotados. Utilizando uma abordagem não informada, Goodfellow *et al.* foram capazes de obter 99.8% de acerto nos textos da primeira versão do reCAPTCHA (12). Os autores ainda conduziram um estudo no repositório SVHN, obtendo acertos de 97.84% por caractere e 96% para o token completo. Entretanto, a base de treino utilizada era formada por 600 mil imagens para o SVHN (toda a base de teino) e mais de um milhão de imagens para o reCaptcha. Mais recentemente, foi investigada uma alternativa para diminuir volume de dados necessário para esse tipo de abordagem através de uma nova arquitetura denominada Redes Corticais Recorrentes (do inglês, Recurrent Cortical Network) (13). Utilizando esta nova arquitetura, os autores do estudo relataram uma alta eficiência de dados, obtendo resultados satisfatórios com algumas milhares ou até mesmo centenas de exemplos. O diferencial dessa arquitetura

[5] O MNIST (Modified National Institute of Standards and Technology database) é um repositório aberto contendo 70000 imagens de dígitos escritos à mão e usualmente usado como benchmark para técnicas de OCR. A tarefa consiste em reconhecer corretamente o número codificado na imagem. No sítio do repositório existe uma tabela comparativa da precisão de diversos estudos publicados utilizando-se diferentes técnicas ao longo dos anos.

[6] SVHN (Street View House Numbers), composto por mais de 600000 fotos de fachadas de construções contendo números. O desafio consistem em reconhecer corretamente os algarismos codificados na imagem. O repositório possui dois formatos para os mesmas imagens: a) Caracteres segmentados; e b) Apenas a região contendo os números.

são conexões extras adicionadas entre as camadas que permite um fluxo mais complexo da informação. Aplicada à quebra de CAPTCHA, os autores treinaram esse tipo de rede em imagens contendo diferentes fontes tipográficas para o mesmo alfabeto. Durante a validação, a arquitetura mostrou capacidade de generalizar e abstrair as transformações aplicadas em CAPTCHAs sintéticos, sendo capaz de reconhecer os símbolos mesmo após as deformações. Um ponto negativo, entretanto, é o tempo de treino dessa nova arquitetura. As mensagens trocadas entres as camadas e o algoritmo proposto para supressão de característica limitam a capacidade de paralelização e tornam a execução mais lenta. De fato, há uma nota no repositório dos autores informando que o treino em 1000 imagens do desafio MNIST (onde o algoritmo se aproxima do estado da arte nesse desafio) pode levar horas em múltiplas CPUs[7]. Adicionalmente, os autores alegam que fizeram ajustes nos filtros convolucionais e nas fontes de treino para cada aplicação, o que pode ser interpretado com uma forma de adicionar viés ao modelo (algo como uma abordagem semi-informada). Em um estudo similar ao nosso, Pinto alcançou $76,6\%$ de precisão na quebra CAPTCHAs de texto utilizando redes neurais profundas. Contudo, foram necessários uma base de treino com 180 mil imagens e placas de processamento gráfico de última geração, sendo o treino realizado em sistemas de computação sob demanda (34). Sendo este o trabalho mais recente diretamente relacionado ao nosso encontrado na literatura, o definimos como trabalho de referência para comparações com os resultados obtidos em nosso trabalho.

Diante disso, apesar da qualidade dos resultados encontrados na literatura, os requisitos desses sistemas os tornam proibitivos para serem treinados em computadores comuns. Na tabela 1 apresentamos um breve comparativo dos requisitos de alguns dos estudos selecionados. Essas restrições limitam o acesso dessa tecnologia à ambientes com menor poder de processamento ou restrições orçamentárias. É particularmente proibitivo para o estudante médio brasileiro, o que pode gerar uma defasagem de aprendizado tecnológico, e para pequenas empresas experimentarem soluções inovadoras baseadas nessas descobertas. Assim, o objetivo deste trabalho é propor um abordagem construtiva para a experimentação de redes neurais profundas em computadores comuns focada no problema de **quebra não informada de CAPTCHAs de texto**. Pretendemos mostrar que é possível alcançar resultados próximos ao estado da arte a partir da investigação cautelosa do comportamento dos modelos durante a dinâmica de treino, com bases de treinos menores e arquiteturas mais simples.

[7] Fonte: <https://github.com/vicariousinc/science_rcn>. Acesso em: 23/06/2018.

Tabela 1 – Comparação entre os requisitos dos modelos encontrados na literatura.

Referência	Limitação
Netzer *et al.* (32)	600 mil imagens, 500 filtros convolucionais 8x8.
Sermanet *et al.* (33)	16 filtros 5x5, 512 filtros 7x7, 2 camadas densas de 4000 entradas.
Goodfellow *et al.* (12)	De 9 a 11 camadas convolucionais (8-192 filtros), camada densa com mais de 3 mil entradas, mais de 500 mil exemplos.
Pinto (34)	180 mil exemplos, quatro camadas convolucionais com 64, 128, 256 e 512 canais, duas camadas densas com mais de 4000 entradas, totalizando mais de $6\,10^7$ parâmetros.
Dileep *et al.* (13)	Treinameto de difícil paralelização. Muitas horas de treino para alcançar resultados satisfatórios. Necessidade de acompanhamento por humanos.

4 Modelagem

Neste capítulo apresentamos uma justificativa para a abordagem comparativa proposta no presente estudo. Em seguida descrevemos as camadas e arquiteturas que foram utilizadas. Adicionalmente, definimos a nomenclatura adotada de forma a simplificar a transmissão de nossos resultados.

4.1 Abordagem Comparativa

O projeto da arquitetura é um ponto crucial para a obtenção de resultados satisfatórios. Infelizmente, até o presente momento, não existe nenhum estudo que demonstre de forma precisa como as camadas interagem entre si ou sobre como estimar o impacto de cada componente no modelo final. Encontramos um problema similar na escolha dos hiper-parâmetros. Em sua grande maioria, não existe uma metodologia definida de como escolher os valores que apresentarão os melhores resultados, sendo nosso conhecimento usualmente limitado a ideias gerais. De fato, se imaginarmos que um hiper-parâmetro é um valor que altera o funcionamento de um método, mas que precisa ser escolhido para cada caso, a própria arquitetura da rede pode ser vista como uma espécie de hiper-parâmetro que precisa ser definido. Assim, deste ponto em diante, iremos nos referir à escolha da arquitetura e dos hiper-parâmetros simplesmente como configuração. A falta de um suporte teórico para as decisões à serem tomadas no projeto de uma configuração, obrigam o projetista a basear suas escolhas em critérios empíricos.

Uma forma de contornar o problema seria realizar uma busca em todas as possíveis combinações e escolher a melhor dentre elas. Entretanto, basta notar que as possibilidades de configurações aumentam de forma algébrica com cada possibilidade para se convencer de que esta é uma solução computacionalmente intratável. Mesmo se encontrarmos uma forma de simplificar a busca por soluções, impondo limites às configurações válidas e/ou utilizando alguma algoritmo inteligente (busca heurística, algoritmos evolucionários, etc.), ainda teríamos de realizar um treino completo de cada configuração para poder estimar sua performance[1]. O problema de escolher uma boa configuração é ainda mais grave em ambientes restritivos, onde adicionalmente temos que nos limitar aos recursos disponíveis.

Podemos, entretanto, lançar mão de alguns pressupostos para realizar uma escolha eficiente de configurações para experimentação:

[1] O problema de encontrar configurações de algoritmos de aprendizado de máquina de forma automática é referido na literatura como automl (do ingles, auto machine learning). Mesmo para modelos mais simples ainda é um problema de intensa pesquisa.

1. O início da dinâmica de uma configuração fornece informações importantes sobre o comportamento seu ao longo do resto treino.

2. Configurações similares tendem a produzir resultados similares.

3. Uma experimentação mais consistente resulta em conclusões mais consistentes.

Neste trabalho, propomos a realização de experimentos comparativos entre diferentes configurações como uma forma de ajudar o processo do projeto da arquitetura e escolha de hiper-parâmetros satisfazendo os pressupostos da seguinte forma:

1. Várias configurações são treinadas durante um tempo reduzido e sua performance avaliada.

2. As arquiteturas à serem estudadas são substancialmente diferentes entre sí. Esse princípio também é aplicado à parâmetros contínuos como a taxa de aprendizado, por exemplo.

3. Um conjunto de hiper-parâmetros sempre será mantido fixo enquanto os demais são explorados de forma mais minuciosa, permitindo uma comparação direta do impacto de cada escolha. De outra forma, é preferível fixar um configuração e variar apenas um parâmetro do que ter diferentes configurações que não podem ser diretamente comparadas.

Este método nos permite, ao mesmo tempo, satisfazer as restrições e executar treinos de forma mais eficiente. Tipicamente, cada etapa de treino consiste dos mesmos passos, assim, se calcularmos o tempo médio de duração de cada etapa no inicio da dinâmica, podemos estimar o tempo máximo do treino (restrição de tempo). Tipicamente, fenômenos como coadaptação e divergência podem ser detectados nas primeiras etapas de treino, poupando-nos de prosseguir com a experimentação. Adicionalmente, durante o tempo que seria usado em um treino completo, podemos realizar vários treinos mais rápidos. Como projetamos cada uma das arquiteturas, podemos nos restringir àquelas que obedecem ao limite disponível de recursos (restrição de memória e processamento) e ainda obter uma variabilidade de experimentação. A escolha consistente dos hiper-parâmetros nos permite determinar de forma mais precisa sua influência na performance e decidir de forma consciente qual valor usar. Realizados os experimentos, podemos então escolher uma configuração que apresente boa performance e obedeça os critérios impostos e então realizar um treino completo.

4.2 Camadas

Nesta secção descrevemos as camadas utilizadas no presente trabalho. De acordo com a metodologia proposta, manteremos fixas um conjunto escolhas e variamos as demais. Desta forma, cada camada é descrita explicitando o que será mantido fixo e o que será experimentado.

Camadas densas ($\mathbf{Fl}_O$) possuem O neurônios ou projeções como visto no capítulo 3. Estas camadas mapeiam uma soma balanceada dos I sinais de entrada em O sinais de saída, tendo $I \times O$ parâmetros. Quando presente, a ativação *relu* é aplicada aos sinais de saída. Camadas multi-caracteres ($\mathbf{M}$) mapeiam os sinais de entrada em uma distribuição de probabilidades para cada caractere no token. Mais especificamente, esta camada é formada por N (sendo $N = 5$, o tamanho fixo do token) classificadores independentes, cada um formado por uma camada densa seguida de uma ativação softmax. Os parâmetros de cada classificador não são compartilhados entre si. A camada multi-caracteres está presente em todas as arquiteturas, sendo sempre a última. Sendo o sinal de entrada um vetor de tamanho I e o de saída de tamanho O (onde $O = 36$ é o tamanho fixo do alfabeto), o número de parâmetros da camada $\mathbf{M}$ é dado por $N \times I \times O$ ($= 180 \times I$).

Camadas convolucionais $\mathbf{C}_O$ possuem núcleos de tamanho fixo $k = 5$ em cada uma das direções, O canais de saída e passo $s = 1$ ou $s = 2$ dependendo da arquitetura. Se a camada convolucional for seguida por uma agregação de *maxpooling*, ela sempre terá passo 1, caso contrário, o passo é 2, exceto quando houver mais de uma camada convolucional. Neste caso, a primeira tem passo 1 e as demais 2. O tamanho de um camada convolucional com I canais de entrada e O canas de saída é dado por $k^2 \times I \times O$ ($= 25 \times I \times O$). Seja $\mathbf{X}^{H,W,I}$ o tensor e entrada, a saída é um tensor $\mathbf{Z}^{(H-k+1)/s,(W-k+1)/s,O}$. Após cada camada convolucional é aplicada uma ativação *relu*.

A operação de linearização transforma o tensor de entrada em vetor de saída, através da reordenação dos índices. Seja o tensor de entrada $X_{H,W,I}$, a saída desta camada é um vetor $x'_{H\times W\times I}$. A linearização está sempre presente antes da primeira camada densa da arquitetura. As operações de *dropout* ($\mathbf{D}$) e de *maxpooling* ($\mathbf{Max}$) podem estar presentes ou não, dependendo da arquitetura. Quando presente, o *dropout* atuará em cada uma das camadas ocultas da rede, anulando cada um dos sinais de saída da camada de forma independente com probabilidade de $p_{drop} = 30\%$. A única exceção é nas arquiteturas com apenas a camada multi-caractere. Neste caso, o *dropout* é aplicado diretamente nos sinais de entrada da rede, isto é, diretamente na imagem. A operação de *maxpooling*, quando presente, atua depois de cada camada convolucional, com passo fixo em 2 em ambas as direções. Ou seja, apenas o maior valor dos quatro pixels de entrada (2 na direção i e 2 na direção j) estará presente no canal de saída. A operação agregação só é aplicada ao fim de camadas convolucionais que teriam passo 2 (vide descrição no parágrafo anterior). Caso presente, as camadas convolucionais passam a ter passo 1. O tensor de saída tem

as dimensões transformadas de forma similar à uma operação convolucional com $k = 2$ e $s = 2$. Nenhuma dessas operações (linearização, agrupamento ou *dropout*) adicionam parâmetros à arquitetura, tendo tamanho 0.

4.3 Arquiteturas

As arquiteturas estudadas são formadas pela composição das camadas descritas na seção 4.2. De modo a facilitar a referência posterior, os nome dado a cada uma é definido pela concatenação dos nomes de cada camada, com a indicação das operações facultativas quando presentes. Nas tabelas 2-7 a seguir as arquiteturas são descritas por camada, estando as dimensões de entrada e saída e número de parâmetros explicitados. A presença da regularização é indicada entre colchetes. O fluxo de dados durante a computação ocorre da primeira para a última linha na tabela. O detalhamento das arquiteturas usando agregação *maxout* pode ser facilmente obtidos a partir das descrições apresentadas, sendo, portanto, omitidas.

No próximo capítulo descrevemos os detalhes do treino e validação das arquiteturas aqui descritas, completando a definição de uma configuração e de como foram executadas as experimentações. Para os hiper-parâmetros foram utilizados os mesmos princípios explicitados neste capítulo. No capítulo de resultados, as métricas de interesses paras essas arquiteturas são comparadas e um modelo escolhido à luz da metodologia proposta.

Tabela 2 – Arquitetura $M[D]$

Camada	Descrição	Entrada	Saída	Parâmetros
Lin	[Dropout]	(50,200,3)	(30000)	0
M	5 classificadores.	(30000)	(5,36)	5400000
total				5400000

Tabela 3 – Arquitetura $C_6M[D]$

Camada	Descrição	Entrada	Saída	Parâmetros
C_6	Convolucional com 3 canais de entrada e 6 de saída. Passo da convolução 2. [Dropout]	(50,200,3)	(23,98,6)	450
Lin	-	(23,98,6)	(13524)	0
M	5 classificadores.	(13524)	(5,36)	2434320
total				2434770

Tabela 4 – Arquitetura $C_6C_{12}M[D]$

Camada	Descrição	Entrada	Saída	Parâmetros
C_6	Convolucional com 3 canais de entrada e 6 de saída. Passo da convolução 1. [Dropout]	(50,200,3)	(46,196,6)	450
C_{12}	Convolucional com 6 canais de entrada e 12 de saída. Passo da convolução 2. [Dropout]	(46,196,6)	(21,96,12)	1800
Lin	-	(21,96,12)	(24192)	0
M	5 classificadores.	(24192)	(5,36)	4354560
total				4356810

Tabela 5 – Arquitetura $C_6C_{12}Fl_{100}M[D]$

Camada	Descrição	Entrada	Saída	Parâmetros
C_6	Convolucional com 3 canais de entrada e 6 de saída. Passo da convolução 1. [Dropout]	(50,200,3)	(46,196,6)	450
C_{12}	Convolucional com 6 canais de entrada e 12 de saída. Passo da convolução 2. [Dropout]	(46,196,6)	(21,96,12)	1800
Lin	-	(21,96,12)	(24192)	0
Fl_{100}	Camada densa com 24192 sinais de entrada e 100 sinais de saída. [Dropout]	(24192)	(100)	2419200
M	5 classificadores.	(100)	(5,36)	18000
total				2439450

Tabela 6 – Arquitetura $C_6C_{12}C_{36}C_{36}M[D]$

Camada	Descrição	Entrada	Saída	Parâmetros
C_6	Convolucional com 3 canais de entrada e 6 de saída. Passo da convolução 1. [Dropout]	(50,200,3)	(46,196,6)	450
C_{12}	Convolucional com 6 canais de entrada e 12 de saída. Passo da convolução 2. [Dropout]	(46,196,6)	(21,96,12)	1800
C_{36}	Convolucional com 12 canais de entrada e 36 de saída. Passo da convolução 2. [Dropout]	(21,96,12)	(9,46,36)	10800
C_{36}	Convolucional com 36 canais de entrada e 36 de saída. Passo da convolução 2. [Dropout]	(9,46,36)	(3,21,36)	32400
Lin	-	(3,21,36)	(2268)	0
M	5 classificadores.	(2268)	(5,36)	408240
total				453690

Tabela 7 – Arquitetura $C_6C_{12}C_{36}C_{36}Fl_{100}M[D]$

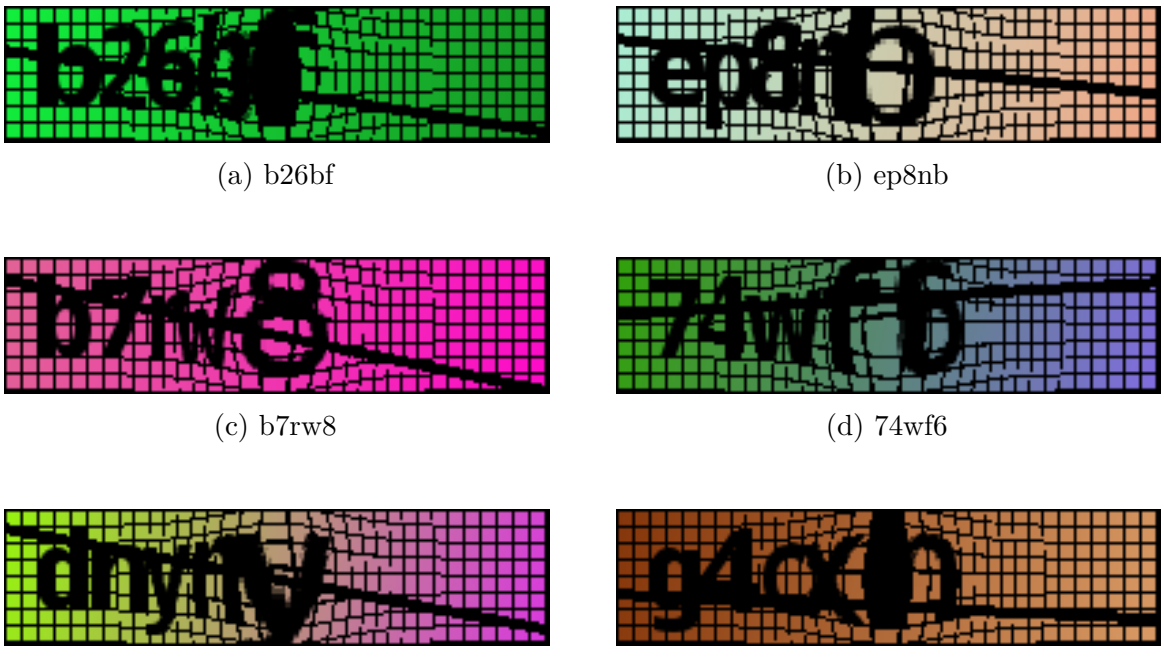
Camada	Descrição	Entrada	Saída	Parâmetros
C_6	Convolucional com 3 canais de entrada e 6 de saída. Passo da convolução 1. [Dropout]	(50,200,3)	(46,196,6)	450
C_{12}	Convolucional com 6 canais de entrada e 12 de saída. Passo da convolução 2. [Dropout]	(46,196,6)	(21,96,12)	1800
C_{36}	Convolucional com 12 canais de entrada e 36 de saída. Passo da convolução 2. [Dropout]	(21,96,12)	(9,46,36)	10800
C_{36}	Convolucional com 36 canais de entrada e 36 de saída. Passo da convolução 2. [Dropout]	(9,46,36)	(3,21,36)	32400
Lin	-	(3,21,36)	(2268)	0
Fl_{100}	Camada densa com 2268 sinais de entrada e 100 sinais de saída. [Dropout]	(2268)	(100)	226800
M	5 classificadores.	(100)	(5,36)	18000
total				290250

5 Metodologia

Neste capítulo os detalhes envolvidos na geração das imagens de CAPTCHAs são expostos. Em seguida, definimos as grandezas de interesse que nos permitem acessar a qualidade dos modelos treinados. Por fim, as etapas de treino e validação são formalizadas.

5.1 Geração dos CAPTCHAs

Todos os exemplos foram gerados utilizando a biblioteca SimpleCaptcha(35). Ao total, foram gerados 30000 pares imagem-token. As sequências de texto possuem comprimento fixo em 5 e os caracteres foram sorteados de forma independente a partir do alfabeto ordenado $\Sigma = \{0123456789abcdefghijklmnopqrstuvwxyz\}$ de 36 símbolos. Dentre os efeitos escolhidos para as imagens, enfatizamos as variações nas cores de fundo, desenho de grades, adição de linhas aleatórias e deformação em explosão, que são técnicas efetivas para construir desafios fáceis para humanos e difíceis para computadores, de acordo com estudo conduzido por (10). Uma pequena amostra dos pares imagen-token gerados pode ser vista na Fig.7.



(a) b26bf (b) ep8nb

(c) b7rw8 (d) 74wf6

(e) dnyny (f) g4cxh

Figura 7 – Exemplos de CAPTCHAs gerados e seus respectivos tokens.

Considere $D = (x, y)$ o conjunto formado por todos os pares de imagem-token gerados. Durante o treino, cada exemplo de entrada é formado por um tensor imagem

x e uma matriz y representando o token correto u, de dimensões $(200, 50, 3)$ e $(5, 36)$, respectivamente. Ao realizar uma predição, apenas a entrada x é fornecida à rede. Cada $x_{ijk} \in \Re[0, 1]$ representa a intensidade do pixel localizado na posição (i, j) e canal k, enquanto $y_{ij} \in \Re[0, 1]$ foi codificado utilizando-se a técnica *one-hot encoding*, onde i representa a posição na sequência u e j o índice no vocabulário do caractere nessa posição, de modo que

$$y_{ij} = \begin{cases} 1, & \text{se } u_i = \Sigma_j \\ 0, & \text{caso contrário,} \end{cases} \tag{5.1}$$

ou, de forma mais compacta, $y_{ij} = \delta_{u_i,\Sigma_j}$, onde $\delta_{m,n}$ é o delta de Kronecker. Essa codificação nos permite interpretar y como sendo uma distribuição de probabilidade. Se imaginarmos z como uma variável aleatória descrevendo a i-*ésima* entrada na sequência, e $p_i(z|x)$ como a probabilidade de na posição i da sequência termos o caractere z dada a imagem x, para os exemplos gerados, o conhecimento da imagem define automaticamente qual o caractere em cada posição com 100% de certeza, ou seja, se c é o caractere de fato na sequência $(c = u_i)$, teremos $p_i(z = c|x) = p_i(z = u_i|x) = 1$ e, caso contrário, $p_i(z \neq c|x) = p_i(z \neq u_i|x) = 0$. Ou, utilizando o delta de Kronecker, $p_i(z|x) = \delta_{z,u_i}$. Definindo $ord(c)$ como o índice do caractere c no alfabeto Σ (isto é, $c = \Sigma_{ord(c)}$), da equação 5.1 vem que, $p_i(z|x) = \delta_{z,u_i} = \delta_{u_i,z} = \delta_{u_i,\Sigma_{ord(z)}} = \delta_{u_i,\Sigma_j} = y_{ij}$.

A saída da rede neural é a matriz $\hat{y} = f^{\Theta}(x)$, a distribuição de probabilidade inferida pela rede. O caractere predito pela rede na posição i é dado pelo caractere mais provável nessa distribuição nesta posição, isto é, $\Sigma_{\arg\max_j \hat{y}_{ij}}$, sendo o token predito formado pelos caracteres mais prováveis em cada posição. Nosso objetivo é encontrar $\hat{y}$ como uma aproximação para y.

Para a preparação da base, o D foi reordenando de forma aleatória e separado em dois subconjuntos: o conjunto de treino, D_{tr}, com $\frac{2}{3}$ do total de pares, e o conjunto de validação, D_{val}, com os demais exemplos. Devido à natureza combinatória do espaço de imagens possíveis (36^5 tokens $\times$ 255^3 cores de fundo $\times$ espaço de todas as pertubações possíveis), D é muito menor do que o conjunto de todas as possíveis composições imagem-token e, supondo que seus elementos tenham sido construídos ao acaso, é virtualmente improvável que existam exemplos repetidos nesse conjunto.

5.2 Treino e Validação

Uma época de aprendizado consiste em duas etapas: treino e validação. Durante o treino, um subconjunto $D_{batch} \subset D_{tr}$ é sorteado ao acaso. Os parâmetros da rede são atualizados utilizando o algoritmo adaptativo Adam com os parâmetros sugeridos em(27) ($\beta_1 = 0.9$, $\beta_2 = 0.999$ e $\epsilon = 10^{-8}$) e taxa de aprendizado l_r de forma a minimizar o erro nesse subconjunto. A subetapa de treino (treino dentro de uma época) se encerra após

$|D_{tr}|/|D_{batch}|$ atualizações. Na etapa de validação, as grandezas de interesse são calculadas para D_{tr} e D_{val} e salvas para posterior análise. Em todos os experimentos fixamos $N_B = 10$ exemplos.

As redes foram inicializadas segundo a heurística proposta em (36) e as épocas de aprendizado se sucedem até que o critério de parada ser alcançado. Escolhemos como critério de parada uma heurística semelhante às definidas por (37). A dinâmica de aprendizado leva no mínimo T^{min} e no máximo T^{max} épocas. Após a etapa de validação o aprendizado é interrompido prematuramente se um dos dois critérios forem verificados: o custo calculado em D_{val} na época atual ultrapasse em mais de 10% o menor valor de $J^{(D_{val})}$ nas épocas anteriores; o valor de $J^{(D_{tr})}$ atual seja maior do que 97% da média dos cinco últimos custos nesse conjunto. Ou seja, o treinamento é parado prematuramente se for detectado *overfitting* ou se não houver melhora significativa em relação aos últimos valores.

Para selecionar o valor do hiper-parâmetro l_r, foram realizados experimentos durante as 10 épocas e $l_r = 10^{-\eta}$, com $\eta = 1, 2, 3, 4, 5$, para cada arquitetura. A partir dos experimentos, selecionamos manualmente os limites inferior e superior, (l_r^-, l_r^+), que apresentam o melhor compromisso entre velocidade de aprendizado e estabilidade (vide seções 3.4 do capítulo 3 e 4.1 do capítulo 4). Os experimentos posteriores foram realizados com uma taxa de aprendizado segundo a equação:

$$l_r(t) = l_r^+ + (l_r^- - l_r^+) * \frac{t}{T_{max} - 1}, \tag{5.2}$$

onde $t = 0, 1, 2, \ldots, T_{max} - 1$, onde t é a época atual.

Todos os experimentos realizados nesse trabalho foram executados ema máquina com processador Intel® Core™i5-6200U, 8gb de RAM e placa de aceleração gráfica NVIDIA® 920M com 2 gb de memória, utilizando a biblioteca de código aberto Tensorflow (38).

5.3 Métricas

No capítulo de fundamentação teórica de redes neurais (capítulo 3), vimos que cada arquitetura é parametrizada Θ. Mais especificamente, cada uma das arquiteturas utilizadas neste trabalho possui como parâmetros um conjuntos de números reais. Assim, definimos o **tamanho do modelo**, para fins de comparação, como a soma da quantidade de parâmetros de cada camada da arquitetura. Para treinar e avaliar a qualidade dos modelos, consideramos as grandezas definidas à seguir. Em todas as definições, considere D um conjunto de exemplos, $(x, y) \in D$ e $\hat{y} = f^{\Theta}(x)$ a distribuição de probabilidade inferida pelo modelo (eq. 5.1).

Para avaliar o erro cometido pelos classificadores, podemos utilizar a **entropia cruzada** (no inglês *cross entropy*), que pode ser interpretada como uma medida de

divergência entre duas distribuições de probabilidade. Assim, o custo associado ao inferir $\hat{y}$ quando a verdadeira distribuição deveria ser y, por caractere, é dado por

$$H_i(y, \hat{y}) = -\sum_j y_{ij} \log_2 \hat{y}_{ij} \tag{5.3}$$

$$= -\sum_j \delta_{u_i,\Sigma_j} \log_2 \hat{y}_{ij} \tag{5.4}$$

$$= -\log_2 \hat{y}_{i\, ord(u_i)} \tag{5.5}$$

onde utilizamos o fato de $\delta_{u_i,\Sigma_j} = 0$ exceto em $j = ord(u_i)$. Em outras palavras, a entropia associada ao classificador da posição i é o logaritmo da probabilidade predita para o caractere correto nessa posição. Definimos o **custo esperado por caractere** do classificador i em D como

$$J_i^{(D)} = \frac{1}{|D|} \sum_{(x,y)\in D} H_i(y, \hat{y}) \tag{5.6}$$

e **custo esperado por token** como a média dos erros em cada posição, ou seja:

$$J^{(D)} = \langle J_i^{(D)} \rangle_{\{i\}}. \tag{5.7}$$

Durante o treino tentaremos minimizar a 5.6 para cada classificador, e o custo total associado aos erros cometidos pelo modelo é calculado pela equação 5.7.

Uma estimativa da probabilidade de acerto por caractere é dada pela **acurácia de cada classificador**, isto é, o número de acertos do caractere i no conjunto D, representado por N_i, normalizado pelo tamanho do conjunto D:

$$\hat{p}_i^{(D)} = acc_i^{(D)} = \frac{N_i}{|D|}. \tag{5.8}$$

Estimamos a probabilidade de acerto do token através da **acurácia do modelo** pela equação:

$$\hat{p}_u^{(D)} = \frac{N_u}{|D|}, \tag{5.9}$$

onde N_u é o número de tokens preditos corretamente.

Quanto ao tempo da dinâmica de aprendizado, definimos, para cada época t, o **tempo de treino por época**, $\tilde{\tau}$, como sendo o tempo gasto durante a fase de treino nessa época e o **tempo total de uma época**, τ, com a soma dos tempos gastos com treino e validação. Adicionalmente, definimos o **tempo de convergência**, T, como o tempo decorrido até que o critério de parada tenha sido alcançado.

6 Resultados e Discussão

Os resultados análise são apresentados no inicio deste capítulo. Posteriormente, o comportamento das redes treinadas por mais épocas é explicitado. Ao fim, apresentamos nossas conclusões sobre o estudo.

6.1 Experimentos Preliminares

As arquiteturas descritas no capitulo 4 foram treinadas durante as 10 primeiras épocas para os valores de $l_r = 10^{-\eta}$, com $\eta = 1, 2, 3, 4, 5$. Na tabela 8 vemos o tamanho, o tempo médio na fase de treino e o tempo médio total por época de arquiteturas selecionadas. O desvio padrão relativo é menor do que 0.1% em todas as médias. Em geral, a etapa de treino constitui de 70% à 85% do tempo total de uma época.

Tabela 8 – Comparação do tempo de treino entre as arquiteturas.

modelo	tamanho (parâmetros)	$\tilde{\tau}$ (segundos)	τ (segundos)
(a) M	$5.4\,10^6$	72.63	100.97
(b) MD	$5.4\,10^6$	75.96	108.14
(c) C_6M	$2.4\,10^6$	66.96	91.24
(d) C_6MD	$2.4\,10^6$	71.19	97.57
(e) $C_6C_{12}MD$	$4.4\,10^6$	238.66	291.03
(f) $C_6C_{12}Fl_{100}MD$	$2.4\,10^6$	311.42	357.51
(g) $C_6C_{12}C_{36}C_{36}MD$	$4.5\,10^5$	249.53	302.44
(h) $C_6C_{12}C_{36}C_{36}Fl_{100}MD$	$2.9\,10^5$	256.97	308.77
Pinto(34)	$6.9\ 10^7$	-	-

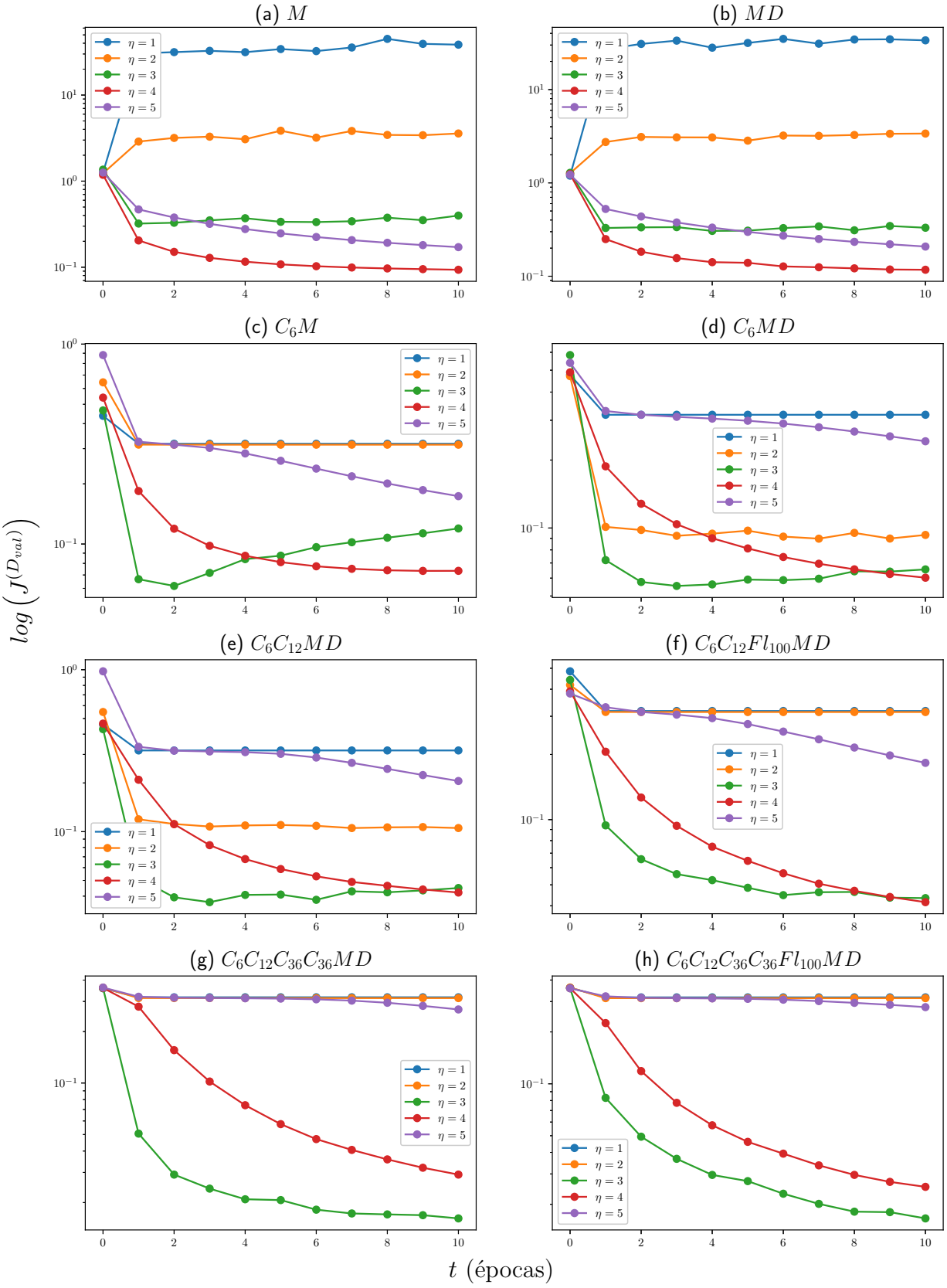
A partir da tabela podemos ver que o tempo consumido depende não apenas da quantidade de parâmetros, mas também da arquitetura. As arquiteturas C_6M e $C_6C_{12}Fl_{100}MD$, por exemplo, possuem aproximadamente o mesmo número de parâmetros, sendo a última, entretanto, pelo menos quatro vezes mais lenta. Em todas as arquiteturas estudadas, a adição do dropout aumentou o ligeiramente o tempo de treino (entre 3% e 10%), mostrando-se uma operação computacionalmente barata. Camadas de maxpooling (omitidas da tabela por brevidade), entretanto, mostraram-se computacionalmente custosas, aumentando a demanda de tempo em pelo menos duas vezes. Chamamos a atenção de que a relação entre as camadas é tão importante para o tamanho e tempo de execução quanto a especificação da camada em sí. No par (d)-(e), por exemplo, a adição de uma camada

convolucional aumentou consideravelmente o tamanho e o tempo de execução. Nesse caso, os canais extras adicionados aumentaram substancialmente a camada densa seguinte. No par (e)-(f) a camada densa reduz drasticamente a quantidade de sinais (de 24192 para 100), reduzindo o tamanho da entrada da camada multi-caractere seguinte, adicionando, entretanto, uma maior de demanda de tempo. Em geral, podemos estabelecer que camadas densas normalmente demandam mais tempo, enquanto camadas convolucionais, usualmente reduzem o número de sinais e tamanho total da arquitetura, mas o impacto exato precisa ser estimado à luz das demais camadas. Para comparação, o modelo no estudo de pinto possui, aproximadamente, 13 vezes o tamanho do maior e 250 vezes o do menor modelo proposto no presente estudo.

Na figura 8 podemos ver a evolução da função custo no conjunto de validação durante as 10 primeiras épocas para diferentes arquiteturas de rede e valores $\eta = 1, 2, 3, 4, 5$. Para $\eta = 1$ e $\eta = 2$, vemos que o custo atinge um platô já durante as primeiras épocas da dinâmica. A estagnação em J é um indicativo de que as correções para o gradiente da função de custo cresce de forma descontrolada. Quando isso ocorre, o algoritmo adaptativo Adam tende a anular as atualizações em subespaços de $\{\Theta\}$, atualizando os parâmetros apenas nas direções de maior gradiente de J. Isto ocorre porque, antes das atualizações, o algoritmo normaliza $\nabla_\Theta J$ pelo módulo do gradiente acumulado nos últimos lotes. Assim, se $|\nabla_\Theta J| \to \infty$, $\Delta\Theta \to 0$, exceto nas regiões onde o gradiente é proporcional acumulado do seu módulo. Em outras palavras, o algoritmo superestima o erro em uma determinada direção um detrimento das demais. Isso leva a rede a cometer os mesmos erros repetidamente nas direções não atualizadas. É importante ressaltar que em algoritmos onde essa normalização não é aplicada (como o *método do gradiente*, por exemplo), quando esse fenômeno ocorre, o valor do custo e os parâmetros também crescem de forma descontrolada. A dinâmica da função de custo nas redes M e MD corroboram para essa interpretação. No gráfico dessas arquiteturas podemos observar um aumento significativo no valor de J antes das atualizações cessarem. Para uma análise mais precisa teríamos de acompanhar a evolução Θ ao longo do treino, o que esta fora do escopo do presente trabalho. No outro extremo, $\eta = 5$, vemos um comportamento similar durante as épocas iniciais para algumas arquiteturas (c-h). Entretanto, a aparente estagnação nesses casos é devida à lenta convergência induzida por esse valor do hiper-parâmetro, observando-se uma melhoria no valor de J na sequência da dinâmica. Assim, esses valores definem os limites práticos para a busca do melhor hiper-parâmetro de aprendizado neste problema.

O valor $\eta = 4$ exibe as dinâmicas mais estáveis, onde a função de custo apresenta um comportamento monotonicamente decrescente durante toda dinâmica, eventualmente estagnando, como visto em (a-d). Para as demais redes (e-i) nota-se que o valor da função de custo ainda apresentaria melhoras caso continuássemos a evolução por mais épocas. Para $\eta = 3$, vemos que a dinâmica é, de fato, mais rápida nas arquiteturas (c-i). Contudo, esse valor leva a instabilidades e, eventualmente, pioras no valor da função de custo (visto

Figura 8 – Dinâmica inicial de arquiteturas selecionadas.



Nos gráficos apresentamos a função custo na escala logarítmica de modo a facilitar a visualização. Os traços são apenas guias visuais. As arquiteturas estão indicadas nos títulos em correspondência com a tabela 8 e os valores do hiper-parâmetro de aprendizado estão indicados na legenda.

de (a) a (f) e mais acentuadamente em (c)). Esse fenômeno de overfitting foi discutido previamente no capítulo 3. Nestes casos, o comportamento pode ser explicado por uma possível memorização dos exemplos de treino, ou seja, esse valor do hiper-parâmetro tende a superestimar os erros cometidos. Buscando o compromisso entre velocidade de aprendizado e estabilidade, os limites para valor de l_r foram fixados em 10^{-3} e 10^{-4}.

Tabela 9 – Comparação entre as arquiteturas na décima época.

modelo	l_r	D_{tr}		D_{val}		$\frac{J^{(D_{val})}}{J^{(D_{tr})}} - 1$
		J	$\hat{p}_u$	J	$\hat{p}_u$	
(a) M	10^{-3}	$1.81\,10^{-1}$	0.37	$3.98\,10^{-1}$	0.21	1.20
	10^{-4}	$4.77\,10^{-2}$	0.53	$9.34\,10^{-2}$	0.30	0.96
(b) MD	10^{-3}	$1.50\,10^{-1}$	0.44	$3.31\,10^{-1}$	0.26	1.20
	10^{-4}	$7.59\,10^{-2}$	0.49	$1.17\,10^{-1}$	0.35	0.55
(c) C_6M	10^{-3}	$2.88\,10^{-3}$	**0.96**	$1.20\,10^{-1}$	0.49	40.49
	10^{-4}	$2.41\,10^{-2}$	0.74	$7.34\,10^{-2}$	0.41	2.05
(d) C_6MD	10^{-3}	$2.80\,10^{-3}$	**0.96**	$6.56\,10^{-2}$	0.56	22.40
	10^{-4}	$3.27\,10^{-2}$	0.72	$6.02\,10^{-2}$	0.51	0.84
(e) $C_6C_{12}MD$	10^{-3}	$\mathbf{2.33\,10^{-3}}$	**0.96**	$4.49\,10^{-2}$	0.68	18.24
	10^{-4}	$2.12\,10^{-2}$	0.81	$4.21\,10^{-2}$	0.65	0.99
(f) $C_6C_{12}Fl_{100}MD$	10^{-3}	$1.99\,10^{-2}$	0.74	$4.34\,10^{-2}$	0.60	1.18
	10^{-4}	$2.10\,10^{-2}$	0.78	$4.16\,10^{-2}$	0.62	0.98
(g) $C_6C_{12}C_{36}C_{36}MD$	10^{-3}	$6.93\,10^{-3}$	0.92	$\mathbf{1.61\,10^{-2}}$	**0.86**	1.33
	10^{-4}	$2.32\,10^{-2}$	0.81	$2.91\,10^{-2}$	0.76	**0.25**
(h) $C_6C_{12}C_{36}C_{36}Fl_{100}MD$	10^{-3}	$1.06\,10^{-2}$	0.86	$1.66\,10^{-2}$	0.81	0.57
	10^{-4}	$1.81\,10^{-2}$	0.81	$2.52\,10^{-2}$	0.75	0.40
Pinto(34)	-	-	0.95	-	0.77	-

A última linha foi reproduzida do trabalho em (34) que utilizou uma rede equivalente à $C_{64}C_{128}C_{256}C_{512}MaxFl_{4096}MD$ e 25 épocas em nossa nomenclatura.

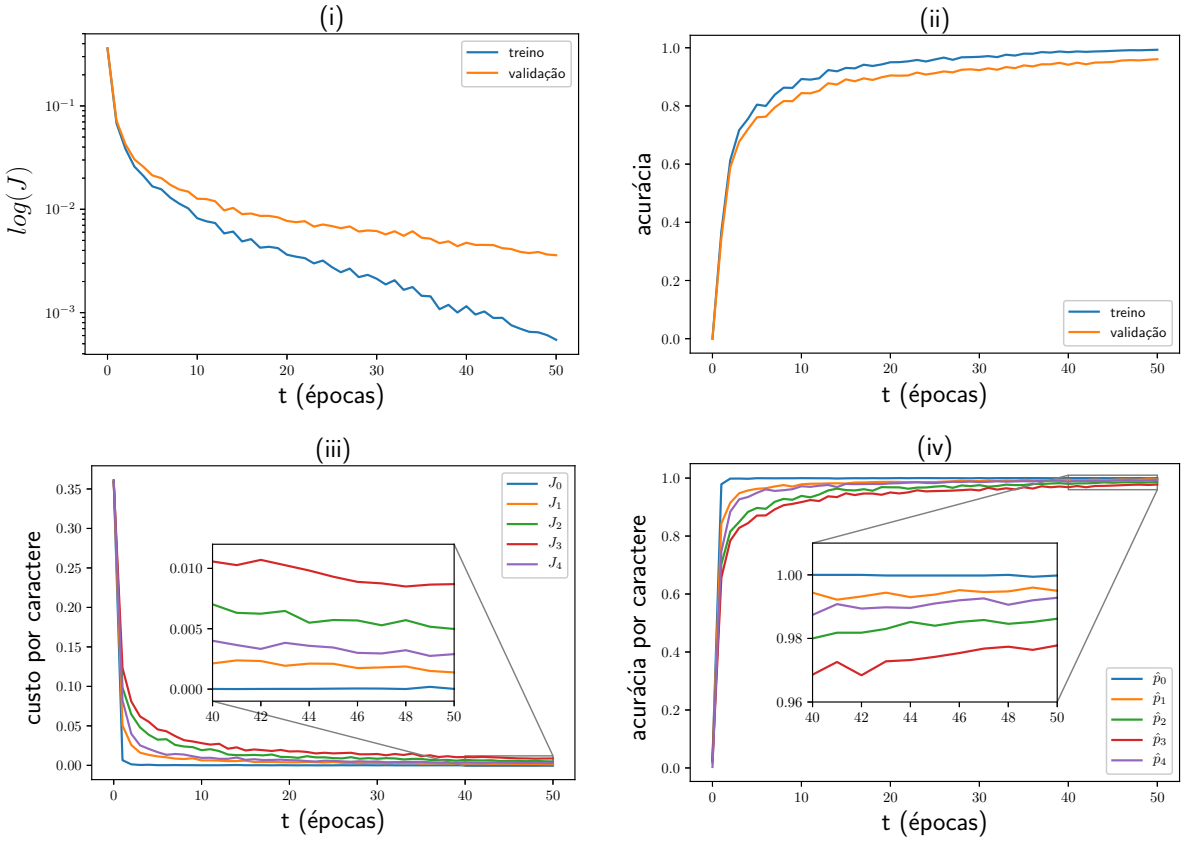
Para melhor entender os modelos treinados com esses valores de l_r fixos, copilamos na tabela 9 o valor da função de custo, a acurácia do modelo para os conjuntos de validação e treino e o valor relativo do custo entre esses dois conjunto, dado por $J^{(D_{val})}/J^{(D_{tr})} - 1$. Todos os valores na tabela foram medidos ná décima época de treino. Na ultima coluna temos o valor reportado por Pinto para comparação. A partir da tabela fica mais claro o overffiting presentes nos modelos (c), (d) e (e). De fato, estes foram os modelos que alcançaram a maior acurácia no conjunto de treino, mas tal performance não se reproduz no conjunto de validação. Comparando os pares (a)-(b) e (c)-(d) podemos ver que a adição de dropout contribuiu de forma significativa para o controle do overfitting. De fato, este comportamento foi observado em todas as comparações feitas entre modelos com e sem essa

regularização (omitido por brevidade). Nas redes treinadas com maxpooling, notamos, em geral, puco impacto ou até mesmo degradação na acurácia dos modelos (também omitido). Nota-se, no geral, a adição de camadas convolucionais melhoraram o desempenho das redes ao mesmo tempo em que limitaram a quantidade de parâmetros a serem optimizados. Nos pares arquiteturas (e)-(f) e (g)-(h), vemos que a adição de uma camada totalmente conectada deteriorou o desempenho. Este é um indicativo de que a simples adição de camadas não necessariamente se traduz em arquiteturas mais expressivas. Na última linha da tabela temos a precisão reportada no estudo de referência. Através da comparação com os parâmetros de rede utilizados no referido trabalho, podemos demonstrar a importância do estudo comparativo do desempenho das arquiteturas de rede no projeto de redes neurais. Fomos capazes de obter resultados muito mais expressivos utilizando menos exemplos e redes mais simples. Em particular, nosso melhor resultados nos experimentos foi obtido em uma rede com 150 vezes menos parâmetros e uma base de treino 9 vezes menor. No estudo de referência estão presentes também fortes indicativos de coadaptação nos parâmetros. A acurácia reportada pelo autor é 23% maior no conjunto de treino do que no de validação, a despeito da aplicação de regularização com norma L_2 e dropout de 50%, sendo esse um forte indicativo de memorização da base de treino.

6.2 Treino completo

O modelo $C_6C_{12}C_{36}C_{36}Fl_{100}MD$ foi treinado até a quinquagésima época e a dinâmica do custo total e da acurácia (total e por caractere) podem ser vistas na figura 9. Esta arquitetura foi selecionada por ter o menor número de parâmetros dentre as estudadas. Como esperado, vemos que o custo, nos intervalo estudado, é uma função decrescente do número de épocas. Esse comportamento sugere que o modelo continuaria a melhorar sua performance caso o treino fosse continuado. Na última época, a acurácia por token foi de 96.06%, sendo o classificador para o primeiro (quarto) caractere o que apresentou melhor (pior) desempenho de acertos, com acurácia final de 99.98% (97.78%). O tempo total de treino for de aproximadamente 4 horas e 16 minutos (3 horas e 33 minutos se descontarmos a fase de validação). Para efeito de comparação, os experimentos do tralho de referência foram realizados em uma máquina com processador Intel® Xeon™E5-2686v4 (*Broadwell*) com 61 gb memória RAM e placa de aceleração gráfica VIDIA® K80 com 12 gb de memória, durando aproximadamente 1 hora 18 minutos.

Figura 9 – Dinâmica da arquitetura $C_6C_{12}C_{36}C_{36}Fl_{100}MD$.



(i) custo (escala logarítmica) e (ii) acurácia ao longo das épocas para o conjunto de treino e validação. (iii) custo e (iv) acurácia por classificador ao longo das épocas para o conjunto de validação.

7 Conclusões

Neste trabalho propomos uma abordagem comparativa entre diferentes arquiteturas de redes neurais para a solução de CAPTCHAs baseados em texto sem informação humana em um ambiente com capacidade computacional inferior às comumente encontradas na literatura. Mostramos que é possível obter performances próximas ao estado da arte a despeito das limitações. O modelo final escolhido obteve uma acuráciafinal de 96.06% de acerto por token, utilizando menos parâmetros, com requisitos de tempo factíveis e treinado em um mero computador pessoal. Seguem nossas observações para que esse tipo de resultado possa ser reproduzido em outras aplicações.

A experimentação dos hiper-parâmetros é um fator chave na obtenção de bons resultados. Como mostrado pelos experimentos, diferentes configurações do hiper-parâmetro de aprendizado podem levar a dinâmicas de treino substancialmente diferentes. Adicionalmente, diferentes arquiteturas possuem diferentes requisitos. É possível projetar arquiteturas que atendam diferentes especificações (tempo de treino, tamanho, acurácia, etc.) e escolher dentre elas a que melhor se aplica ao problema. Para tal, a análise da dinâmica inicial da rede e o uso de validação cruzada são essenciais. Em particular, a avaliação comparativa da evolução da função custo fornece informação crucial para a escolha e projeto de modelos melhores. Escolhido cuidadosamente uma arquitetura que atenda aos requisitos e os valores ideais para os hiper-parâmetros, é possível realizar seções de treino mais longas, com uma maior exposição do modelo à base de treino e refinar ainda mais os resultados.

Arquiteturas convolucionais mostraram-se extremamente eficazes para o problema. De fato, esse tipo de camada tem sido aplicada com sucesso em diversos problemas de processamento de imagem. O compartilhamento de parâmetros ao mesmo tempo reduz o tamanho e adiciona maior poder de expressão aos modelos. O *dropout* mostrou-se uma ferramenta eficaz e computacionalmente barata para combater o *overfitting*, melhorando a performance dos modelos estudados sem degradação preceptiva no tempo de computação. A operação de agrupamento com valor fixo apresentou desempenho superior à de *maxout*, tanto no requisito de tempo quanto na acurácia final dos modelos.

7.1 Trabalhos futuros

Segue uma lista de trabalhos futuros que podem enriquecer a discussão iniciapa pelo presente trabalho.

1. Explorar mais detalhadamente a influência do número de canais e do tamanho

dos núcleos das camadas convolucionais no desempenho das redes, buscando tanto diminuir o número de parâmetros quanto aumentar a acurácia final dos modelos.

2. Aplicar a mesma abordagem à outros problemas de processamento de imagem, como detecção de objetos, por exemplo, ou outras áreas como predição de sequências temporais.

3. Estender os modelos os modelos para CAPTCHAs com palavras de sequência variável e/ou diferentes formatos.

4. Utilizar as mesmas técnicas em CAPTCHAs de texto reais. Em particular, avaliar qual o nível de segurança que esses mecanismos de proteção tem fornecido a instituições no Brasil

Referências

1 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958. Citado na página 10.

2 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 3 vezes nas páginas 10, 18 e 23.

3 SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85–117, 2015. Published online 2014; based on TR arXiv:1404.7828 [cs.NE]. Citado 3 vezes nas páginas 10, 23 e 24.

4 BARRON, A. R. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Information Theory*, v. 39, p. 930–945, 1993. Citado na página 10.

5 ANDONI, A. et al. Learning polynomials with neural networks. v. 5, p. 3955–3963, 01 2014. Citado na página 10.

6 KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012. p. 1097–1105. Disponível em: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>. Citado na página 10.

7 MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, a division of Macmillan Publishers Limited., v. 518, n. 7540, p. 529–533, fev. 2015. ISSN 00280836. Disponível em: <http://dx.doi.org/10.1038/nature14236>. Citado na página 10.

8 AHN, L. von et al. Captcha: using hard ai problems for security. In: *Advances in Cryptology, Eurocrypt*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. v. 2656, p. 294–311. Citado 2 vezes nas páginas 10 e 12.

9 CHOW, Y. W.; SUSILO, W. Text-based captchas over the years. *IOP Conference Series: Materials Science and Engineering*, v. 273, n. 1, p. 012001, 2017. Disponível em: <http://stacks.iop.org/1757-899X/273/i=1/a=012001>. Citado 2 vezes nas páginas 10 e 15.

10 CHELLAPILLA, K. et al. *Building Segmentation Based Human-Friendly Human Interaction Proofs (HIPs)*. [S.l.]: Springer, Berlin, Heidelberg, 2005. v. 3517. 1-26 p. Citado 6 vezes nas páginas 10, 12, 13, 14, 27 e 38.

11 YAN, J.; AHMAD, A. S. E. Breaking visual captchas with naive pattern recognition algorithms. p. 279–291, 01 2008. Citado 2 vezes nas páginas 10 e 13.

12 GOODFELLOW, I. J. et al. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2013. Disponível em: <http://arxiv.org/abs/1312.6082>. Citado 5 vezes nas páginas 10, 13, 14, 28 e 30.

13 GEORGE, D. et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. v. 358, p. eaag2612, 10 2017. Citado 3 vezes nas páginas 10, 28 e 30.

14 BURSZTEIN, E. et al. The end is nigh: Generic solving of text-based captchas. In: *WOOT*. [S.l.: s.n.], 2014. Citado na página 10.

15 SINGH, V. P.; PAL, P. Survey of different types of captcha. *International Journal of Computer Science and Information Technologies*, v. 5, n. 2, p. 2242–2245, 2014. Citado na página 12.

16 SECURIMAGE. *An open-source free PHP CAPTCHA script f.* 2018. Disponível em: <https://www.phpcaptcha.org/>. Acesso em: 23/06/2018. Citado na página 12.

17 AHN, L. von et al. recaptcha: Human-based character recognition via web security measures. v. 321, p. 1465–8, 09 2008. Citado na página 13.

18 SIVAKORN, S.; POLAKIS, I.; KEROMYTIS, A. D. I am robot: (deep) learning to break semantic image captchas. In: . [S.l.]: IEEE, 2016. p. 388–403. Citado na página 14.

19 BURSZTEIN, E.; MARTIN, M.; MITCHELL, J. Text-based captcha strengths and weaknesses. In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. New York, NY, USA: ACM, 2011. (CCS '11), p. 125–138. ISBN 978-1-4503-0948-6. Disponível em: <http://doi.acm.org/10.1145/2046707.2046724>. Citado 2 vezes nas páginas 15 e 27.

20 GERSTNER, W. et al. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014. Disponível em: <http://neuronaldynamics.epfl.ch/>. Acesso em: 23/06/2018. Citado na página 18.

21 LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, MIT Press, Cambridge, MA, USA, v. 1, n. 4, p. 541–551, dez. 1989. ISSN 0899-7667. Disponível em: <http://dx.doi.org/10.1162/neco.1989.1.4.541>. Citado na página 19.

22 LECUN, Y. et al. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1998. v. 86, p. 2278 – 2324. Citado na página 19.

23 LEE, H.; EKANADHAM, C.; NG, A. Y. Sparse deep belief net model for visual area v2. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2008. p. 873–880. Citado na página 22.

24 GATYS, L. A.; ECKER, A. S.; BETHGE, M. Texture synthesis using convolutional neural networks. In: *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 262–270. Disponível em: <http://dl.acm.org/citation.cfm?id=2969239.2969269>. Acesso em: 23/06/2018. Citado na página 22.

25 GATYS, L. A.; ECKER, A. S.; BETHGE, M. Image style transfer using convolutional neural networks. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2016. p. 2414–2423. Citado na página 22.

26 SABOUR, S.; FROSST, N.; HINTON, G. E. Dynamic routing between capsules. In: *Advances in Neural Information Processing Systems*. [S.l.: s.n.], 2017. p. 3856–3866. Citado na página 23.

27 KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. Disponível em: <http://arxiv.org/abs/1412.6980>. Citado 2 vezes nas páginas 24 e 39.

28 FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. [S.l.]: Springer series in statistics New York, NY, USA:, 2001. v. 1. Citado na página 25.

29 HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. Disponível em: <http://arxiv.org/abs/1207.0580>. Citado na página 27.

30 YANN, L.; CORINNA, C.; CHRISTOPHER, J. *The MNIST database of handwritten digits*. 1998. Disponível em: <http://yann.lecun.com/exdb/mnist>. Acesso em: 23/06/2018. Citado na página 28.

31 Cireşan, D.; Meier, U.; Schmidhuber, J. Multi-column Deep Neural Networks for Image Classification. *CoRR*, fev. 2012. Citado na página 28.

32 NETZER, Y. et al. Reading digits in natural images with unsupervised feature learning. In: *NIPS workshop on deep learning and unsupervised feature learning*. [S.l.: s.n.], 2011. v. 2011, n. 2, p. 5. Citado 2 vezes nas páginas 28 e 30.

33 SERMANET, P.; CHINTALA, S.; LECUN, Y. Convolutional neural networks applied to house numbers digit classification. In: IEEE. *Pattern Recognition (ICPR), 2012 21st International Conference on*. [S.l.], 2012. p. 3288–3291. Citado 2 vezes nas páginas 28 e 30.

34 PINTO, V. A. *Redes Neurais Convolucionais de Profundidade Para Reconhecimento de Texto em Imagens de CAPTCHA*. Florianópolis, Santa Catarina.: UNIVERSIDADE FEDERAL DE SANTA CATARINA - DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA, 2016. Disponível em: <https://repositorio.ufsc.br/xmlui/handle/123456789/171436>. Acesso em: 23/06/2018. Citado 4 vezes nas páginas 29, 30, 42 e 45.

35 SIMPLECAPTCHA. *A CAPTCHA Framework for Java*. 2018. Disponível em: <http://simplecaptcha.sourceforge.net/>. Acesso em: 23/06/2018. Citado na página 38.

36 HE, K. et al. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015. Disponível em: <http://arxiv.org/abs/1502.01852>. Citado na página 40.

37 PRECHELT, L. Automatic early stopping using cross validation: Quantifying the criteria. v. 11, p. 761–767, 06 1998. Citado na página 40.

38 ABADI, M. et al. Tensorflow: a system for large-scale machine learning. In: *OSDI*. Savannah, GA, USA: [s.n.], 2016. v. 16, p. 265–283. ISBN 978-1-931971-33-1. Disponível em: <http://download.tensorflow.org/paper/whitepaper2015.pdf>. Acesso em: 23/06/2018. Citado na página 40.