

Resolvendo Captchas
usando aprendizado parcialmente
supervisionado

Julio Adolfo Zucon Trecenti

TESE APRESENTADA AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA UNIVERSIDADE DE SÃO PAULO
PARA OBTENÇÃO DO TÍTULO DE
DOUTOR EM CIÊNCIAS

Programa: Estatística
Orientador: Prof. Dr. Victor Fossaluza

Janeiro de 2023

Resolvendo Captchas
usando aprendizado parcialmente
supervisionado

Julio Adolfo Zucon Trecenti

Esta é a versão original da tese elaborada
pelo candidato Julio Adolfo Zucon Trecenti,
tal como submetida à Comissão Julgadora.

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)*

Aos meus pais,

Agradecimientos

Gratitude is not just a feeling, but a choice. We can choose to focus on the things we are grateful for, even in the midst of difficult circumstances. When we do this, we can experience a sense of peace and contentment that can help us to weather any storm.

— ChatGPT

Agradecer a todo mundo.

Lista de abreviaturas

IME Instituto de Matemática e Estatística
USP Universidade de São Paulo

Lista de símbolos

Lista de figuras

1.1	Explicação de von Ahn sobre o funcionamento do reCaptcha	8
1.2	Exemplo do reCaptcha v2 com a imagens de perus	9
2.1	Fluxo do Captcha	17
2.2	Exemplo de Captcha no TJMG.	18
2.3	Exemplo de captcha gerado pela função <code>captcha::captcha_generate()</code>	19
2.4	Esquema mostrando o funcionamento do oráculo.	20
2.5	Diagrama representando o modelo utilizado de forma genérica, com todos os componentes e subcomponentes apresentados de forma esquemática. As partes de fora dos componentes são entradas de dados ou decisões de parada do ajuste.	23
2.6	Imagem de Captcha utilizado em exemplos anteriores.	23
2.7	Aplicação de uma convolução com kernel horizontal.	24
2.8	Aplicação de uma convolução com kernel horizontal.	24
2.9	Aplicação de uma convolução com kernel horizontal.	25
2.10	Resultado da aplicação da primeira convolução à imagem.	26
2.11	Exemplo de MNIST-Captcha	37
2.12	Exemplo de MNIST-Captcha	37
2.13	Ciclo da raspagem de dados. Fonte: curso de Web Scraping da Curso-R.	38
2.14	Página de busca de CNPJ da RFB.	38
2.15	Página de busca de CNPJ da RFB, com Captcha de texto.	39
2.16	Resultado da busca por CNPJ, mostrando a aba Rede.	39
3.1	Ganho percentual ao utilizar a técnica do oráculo, dividido por quantidade de tentativas.	48
3.2	Ganhos absolutos ao utilizar a técnica do oráculo, dividido por quantidade de tentativas.	48
3.3	Ganho percentual ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.	49

3.4	Ganho absoluto ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.	49
3.5	Ganho absoluto ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.	50
3.6	Resultados da simulação por captcha, quantidade de tentativas e modelo inicial.	51
3.7	Resultados da aplicação iterada da técnica.	52
A.1	Exemplo de aplicação da função <code>plot</code> a um objeto <code>captcha</code>	66
A.2	Exemplo de aplicação da função <code>plot</code> a um objeto <code>captcha</code> com várias imagens.	67
A.3	Exemplo de aplicação da função <code>plot</code> a um objeto <code>captcha</code> com várias imagens, disponibilizadas em duas colunas.	67
A.4	Demonstração da função <code>plot()</code> com muitas imagens.	68
A.5	Demonstração das funções de <code>subset</code> e <code>length</code> aplicadas a um objeto do tipo <code>captcha</code>	69
A.6	Demonstração da função <code>plot()</code> quando o <code>Captcha</code> possui um rótulo . .	69
A.7	Exemplo de aplicação da função <code>classify</code> . O rótulo <code>bhusp5</code> foi inserido manualmente.	70
A.8	Diagrama das funções básicas do pacote <code>{captcha}</code>	71
A.9	Exemplo de criação de um novo projeto de <code>Captcha</code> utilizando o <code>RStudio</code> . .	73
A.10	Exemplo de <code>Captcha</code> baixado diretamente do <code>TRF5</code>	80

Lista de tabelas

2.1	Lista de captchas analisados.	36
2.2	Lista de captchas analisados e suas características.	43

Sumário

Sobre este documento	1
1 Introdução	3
1.1 Captchas em serviços públicos	4
1.2 Uma luta entre geradores e resolvedores	7
1.3 Oráculo	11
1.4 Objetivo	12
1.5 Justificativa	12
1.6 Hipóteses	13
1.7 Organização do trabalho	13
2 Metodologia	15
2.1 Definição do problema	16
2.1.1 Captcha	17
2.1.2 Redes neurais	21
2.1.3 Aprendizado estatístico	30
2.2 Método WAWL	32
2.3 Dados	34
2.3.1 Escolha dos Captchas analisados	35
2.3.2 Construção dos dados	37
2.4 Simulações	43
2.4.1 Primeiro passo: modelo inicial	43
2.4.2 Segundo passo: dados	44
2.4.3 Terceiro passo: modelo final	46
3 Resultados	47
3.1 Discussão	52
4 Conclusões	55

Bibliografia	59
Apêndices	62
A Pacotes	63
A.1 Pacote captcha	64
A.1.1 Uso básico	65
A.1.2 Modelagem	71
A.1.3 Resolvendo um novo Captcha do zero	73
A.2 Pacote captchaDownload	79
A.3 Pacote captchaOracle	83

Sobre este documento

Este documento foi construído utilizando o [Quarto](#), um sistema de publicação científica de código aberto desenvolvido com base no [Pandoc](#). Os códigos foram escritos com o [software estatístico R](#) na [versão 4.2.2](#).

O documento foi escrito em duas versões: PDF e HTML. A versão em HTML foi construída com [Quarto Books](#). A versão em PDF incorpora o [template do IME-USP](#), mantido pelo [Centro de Competência de Software Livre](#).

Todos os códigos e textos da tese podem ser acessados publicamente [neste link](#). Boa leitura!

Capítulo 1

Introdução

Why do we need to prove we're not robots to a robot? Isn't that a robot's job?

— ChatGPT, a robot

Captcha (*Completely Automated Public Turing test to tell Computers and Humans Apart*) é um desafio utilizado para identificar se o acesso à uma página na internet é realizada por uma pessoa ou uma máquina. O desafio é projetado para ser fácil de resolver por humanos, mas difícil de resolver por máquinas. Outro nome para os Captchas é *Human Interaction Proofs*, ou HIPs (CHELLAPILLA et al., 2005).

Um Captcha pode ser classificado como uma variação do teste de Turing (TURING, 2009). A diferença no caso do Captcha é que a avaliação da humanidade do agente é feita por uma máquina ao invés de uma pessoa – por isso o termo *automated*. Captchas podem ser classificados em algumas situações com os chamados **testes de Turing reversos**, apesar dos autores originais afastarem essa caracterização (AHN; BLUM; LANGFORD, 2002).

Captchas estão presentes em toda a internet. Inicialmente criados para prevenir *spam* (*Sending and Posting Advertisement in Mass*), os desafios se tornaram populares rapidamente (AHN et al., 2008), sendo utilizados como forma de evitar evitar o uso indevido de aplicações da *web*. Algumas ações que os desafios podem ajudar a evitar são:

- Criação de contas falsas nos sites.
- Envio automático de mensagens, via *email* ou formulários de contato.
- Operações automatizadas, como compra de ingressos para eventos e voto automático em sites de votação.
- Voto automático em ferramentas de votação na internet.
- Consulta automatizada em sites para obtenção de dados.

O uso de Captchas tem, por princípio, o objetivo de aumentar a segurança das pessoas que acessam a internet e proteger os sistemas *web* de uso abusivo. Para pessoas que acessam os sites pontualmente, a presença de Captchas representa um mero dissabor; para quem realiza acessos massivos, uma grande dificuldade.

No entanto, o uso de Captchas não é adequado em todas as situações. Um exemplo

são os sites de vendas: o uso dos desafios pode aborrecer o usuário, alterando a qualidade da sua experiência. Os sites devem levar esse fator de aborrecimento em conta para não reduzir a taxa de conversão. Em alguns casos, pode fazer mais sentido abandonar os Captchas e utilizar outros mecanismos de prevenção à fraude, como monitoramento da sessão do usuário («Inaccessibility of CAPTCHA», [s.d.]).

Também existem casos em que o uso de Captchas é prejudicial. Sua utilização em serviços públicos do Brasil, por exemplo, é problemática, como discutido na próxima seção.

1.1 Captchas em serviços públicos

A Constituição Federal de 1988 (CF), em seu inciso XXXIII do art. 5º, prevê que “todos têm direito a receber dos órgãos públicos informações de seu interesse particular, ou de interesse coletivo ou geral, que serão prestadas no prazo da lei, sob pena de responsabilidade, ressalvadas aquelas cujo sigilo seja imprescindível à segurança da sociedade e do Estado;”. Essa previsão é implementada pela Lei de Acesso à Informação (Lei 12.527/2011; LAI), que se aplica “aos órgãos públicos integrantes da administração direta dos Poderes Executivo, Legislativo, incluindo as Cortes de Contas, e Judiciário e do Ministério Público”, bem como “as autarquias, as fundações públicas, as empresas públicas, as sociedades de economia mista e demais entidades controladas direta ou indiretamente pela União, Estados, Distrito Federal e Municípios” (Art. 1º).

A LAI, apesar de trazer diversos benefícios à sociedade, tem dois problemas. A primeira é o **esforço**: tanto a pessoa/órgão que solicita os dados, quanto o órgão que retorna os dados precisam trabalhar para disponibilizar as informações, sendo necessário deslocar equipes para realizar os levantamentos pedidos. A segunda é o **formato**: os dados enviados como resultado de pedidos de LAI podem chegar em formatos inadequados para consumo da solicitante, muitas vezes em *Portable Document Format* (PDF), que dificulta a leitura e análise dos dados (MICHENER; MONCAU; VELASCO, 2015, pág. 55); além disso, como o levantamento é realizado de forma individualizada, o mesmo pedido feito em diferentes períodos (e.g. uma atualização mensal dos dados) pode vir em formatos diferentes, dificultando a leitura e arrumação dos dados.

Uma forma de evitar os problemas de esforço e formato em pedidos de LAI é disponibilizar os dados de **forma aberta**. Como definido pela *Open Knowledge Foundation* (OKFN), a base de dados “deve ser fornecida em uma forma conveniente e modificável isento de obstáculos tecnológicos desnecessários para a realização dos direitos licenciados. Especificamente, os dados devem ser legíveis-por-máquina, disponíveis todo o seu volume, e fornecidos em um formato aberto (ou seja, um formato com sua especificação livremente disponível, e publicada sem qualquer restrições, monetárias ou não, da sua utilização) ou, no mínimo, podem ser processados com pelo menos uma ferramenta de software livre e gratuita.”¹

As vantagens ao disponibilizar dados públicos de forma aberta para a sociedade é um tema pacífico na comunidade científica (MURRAY-RUST, 2008). Infelizmente, existem

¹ Link: <https://okfn.org/opendata/>. Último acesso em 01/11/2022.

diversos dados públicos que não estão disponíveis de forma aberta, em plataformas como o dados.gov.br.

A dificuldade de acesso é particularmente evidente no Poder Judiciário, que além de não disponibilizar um portal de dados abertos, impõe barreiras aos pedidos de acesso à informação por utilizar diversos sistemas para armazenar os dados. Por exemplo, para pedir uma lista de todos os processos judiciais relacionados a recuperação judicial de empresas, as únicas alternativas são i) pedir os dados ao Conselho Nacional de Justiça (CNJ), que não possui informações suficientes para obter a lista² ou ii) expedir ofícios aos 27 Tribunais Estaduais. Cada tribunal apresentaria diferentes opções e critérios de acesso aos dados, diferentes prazos para atendimento e diferentes formatos, podendo, inclusive, negar o pedido de acesso.

A dificuldade para acessar os dados do judiciário é a principal barreira para realização de pesquisas pela Associação Brasileira de Jurimetria (ABJ), empresa na qual o autor desta tese trabalha. A entidade tem como missão principal a realização de estudos para implementar políticas públicas utilizando dados do judiciário.

Dos 16 projetos disponibilizados na [página de pesquisas no site da ABJ](#), pelo menos 12 (75%) apresentaram dificuldades na obtenção dos dados via pedidos de acesso aos órgãos. Três exemplos icônicos são o da pesquisa sobre Tempo dos processos relacionados à adoção no Brasil («Tempo dos processos relacionados à adoção», [s.d.]), o Observatório da Insolvência: Rio de Janeiro («Observatório da insolvência», [s.d.]) e o Diagnóstico do Contencioso Tributário Administrativo («Diagnóstico do Contencioso Tributário Administrativo», [s.d.]). No primeiro caso, dois tribunais enviaram os dados em arquivos em papel, sendo que um deles ultrapassou mil páginas com números de processos impressos. No segundo caso, o pedido foi respondido com uma planilha de contagens ao invés da lista de processos. No último caso, até mesmo órgãos que faziam parte do grupo de trabalho da pesquisa negaram pedido de acesso a dados de processos tributários em primeira instância, com argumentos que variavam desde a dificuldade técnica de levantar os dados até a Lei Geral de Proteção de Dados (LGPD).

Em muitas situações, portanto, a única alternativa para realizar as pesquisas é acessando os dados via coleta automatizada nos sites. Todos os tribunais possuem ferramentas de consulta individualizadas de processos, por conta do que está previsto na CF. A solução, portanto, passa a ser construir uma ferramenta que obtém todos os dados automaticamente. Tal conceito é conhecido como *raspagem de dados* (ZHAO, 2017a) e será desenvolvido com maiores detalhes no Capítulo Capítulo 2.

Os Captchas se tornam prejudiciais à sociedade quando o acesso automatizado é necessário para realizar pesquisas científicas. Infelizmente, vários tribunais utilizam a barreira do Captcha. Alguns tribunais, inclusive, têm o entendimento de que o acesso automatizado é prejudicial, como o TJRS, que emitiu um [comunicado](#) sobre o tema.

Uma justificativa comum para implementar Captchas em consultas públicas é a estabilidade dos sistemas. Ao realizar muitas consultas de forma automática, um robô pode

² O CNJ só consegue listar os processos relacionados a um tema a partir da definição de Classes e Assuntos, disponíveis nas [Sistema de Gestão de Tabelas \(SGT\) do CNJ](#). Processos relacionados a recuperação judicial, no entanto, não respeitam a taxonomia do SGT («Observatório da insolvência», [s.d.]).

tornar o sistema instável e, em algumas situações, até mesmo derrubar o servidor que disponibiliza as consultas.

O problema é que utilizar Captchas não impede o acesso automatizado. As empresas que fazem acesso automatizado em tribunais podem construir ferramentas ou utilizar serviços externos de resolução de Captchas. Ou seja, ao utilizar Captchas, o acesso não é impedido, e sim especializado.

Além disso, utilizar Captchas é uma solução ineficiente. Do ponto de vista técnico, a solução mais eficiente para disponibilizar os dados é através de ferramentas de dados abertos como o *Comprehensive Knowledge Archive Network* (CKAN). Ao disponibilizar os dados de forma aberta, as consultas automatizadas ficariam isoladas dos sites de consulta pública, o que garantiria o acesso das pessoas sem problemas de indisponibilidade.

Não é só para as pessoas que fazem pesquisa com dados públicos que o uso de Captchas pode ser prejudicial. No mercado, existem serviços de resolução de Captcha que utilizam mão de obra humana, em regimes que pagam muito menos do que um salário mínimo a 8 horas de trabalho. Um exemplo é o 2Captcha³, que funciona como um Uber dos Captchas: o algoritmo automatizado envia o Captcha para a plataforma, que é acessado e resolvido por uma pessoa, retornando a solução para o algoritmo. O 2Captcha é operado pela AL-TWEB LLC-FZ, uma empresa com base em Dubai⁴.

Segundo o site, o valor pago pelo 2Captcha é de US\$ 0.5 para 1 a 2 horas de trabalho. No regime da Consolidação das Leis do Trabalho (Decreto-Lei 5.452/1943, CLT) as horas mensais de trabalho são 220. Trabalhando continuamente no 2Captcha, isso daria um salário de 55 a 110 dólares por mês, valor bem abaixo do salário mínimo, que no ano de 2022 era de R\$ 1.100,00⁵. Ou seja, os serviços públicos acabam, indiretamente, incentivando um mercado que paga abaixo do salário mínimo. Luis von Ahn, um dos criadores dos Captchas, define o 2Captcha como um *sweatshop*, um termo utilizado para caracterizar empresas que têm condições de trabalho inaceitáveis.

A solução definitiva para os problemas gerado pelos Captchas é a disponibilização dos dados públicos de forma aberta. Na ausência dessa solução, seja por falta de interesse ou iniciativa dos órgãos públicos, a alternativa é desenvolver uma solução para resolver Captchas que seja gratuita e aberta. Tal solução desincentivaria economicamente o uso de sistemas como o 2Captcha, protegendo as pessoas que fazem as resoluções e permitindo que as pessoas pesquisadoras realizem seus levantamentos.

O presente trabalho busca avançar nesse sentido. A solução desenvolvida envolve um modelo que resolve alguns Captchas automaticamente, reduzindo significativamente a necessidade de rotulação manual.

Para compreender completamente o avanço que a tese representa, no entanto, é necessário apresentar o histórico de desenvolvimento dos Captchas. O histórico é apresentado na próxima Seção, através da luta de geradores e resolvidores de Captchas, que pode ser dada como encerrada no ano de 2018, com o advento do *reCaptcha v3*.

³ [Link do 2Captcha](#). Último acesso em 01/11/2022.

⁴ [Link dos termos de serviço do 2Captcha](#). Último acesso em 01/11/2022.

⁵ Fonte: IPEA. Último acesso em 01/11/2022.

1.2 Uma luta entre geradores e resolvidores

O primeiro texto técnico sobre Captchas foi publicado por AHN; BLUM; LANGFORD (2002). O texto apresenta o Captcha e seu significado através do problema de geração de *emails* automáticos no Yahoo. Em seguida, apresenta alguns exemplos de candidatos a Captcha, com tarefas de reconhecimento de padrões ou textos. Uma característica interessante que os autores colocam sobre o Captcha é que suas imagens devem ser disponíveis publicamente. O texto também faz a conexão entre a tarefa dos Captchas e os desafios da inteligência artificial. Um ponto a destacar é que os autores defendem que os Captchas devem ser resolvidos, pois isso implica em avanços na inteligência artificial. O site original do projeto, *The Captcha Project*, foi lançado em 2000.

O relatório técnico de AHN; BLUM; LANGFORD (2002) não foi o primeiro a apresentar o nome Captcha, nem suas aplicações. RESHEF; RAANAN; SOLAN (2005) foi o primeiro registro de patente com o termo e LILLIBRIDGE et al. (2001) foi o primeiro registro de patente que implementou uma solução aos sistemas de Captchas. No entanto, o relatório de 2002 é o primeiro que reconhecidamente trata do tema como um problema de inteligência artificial como um teste de Turing automatizado.

Os artigos mais conhecidos de introdução aos Captchas são VON AHN et al. (2003) e VON AHN; BLUM; LANGFORD (2004). O conteúdo dos trabalhos é o mesmo, sendo o primeiro deles na forma de apresentação e o segundo na forma de relatório. A apresentação é a primeira a mostrar o projeto reCaptcha, que será comentado em uma subseção própria. Um detalhe interessante é a ênfase dos autores no termo *Public* dos Captchas, mostrando a preocupação em manter os códigos públicos.

Os autores também defendem que o Captcha é uma forma de fazer com que pessoas mal intencionadas contribuam com os avanços da inteligência artificial. Se uma pessoa (ainda que mal intencionada) resolve um Captcha e publica essa solução, isso significa que a comunidade científica avançou na área de inteligência artificial.

Não demorou para surgirem os primeiros resolvidores de Captchas⁶. MORI; MALIK (2003) foi um dos primeiros trabalhos publicados sobre o tema e utiliza diversas técnicas de processamento de imagens para obter os rótulos corretos. Também não demorou para a comunidade científica perceber que redes neurais eram úteis nesse contexto (CHELLAPILLA; SIMARD, 2004). No artigo de 2004, Chellapilla e Simard desenvolvem um algoritmo baseado em heurísticas para segmentar a imagem e redes neurais para identificar as imagens individuais.

A partir desse ponto, foi iniciada uma luta entre geradores e resolvidores de Captchas. Do lado dos geradores, as pessoas envolvidas foram tanto acadêmicos tentando desenvolver desafios cada vez mais difíceis para avançar na pesquisa em inteligência artificial, quanto empresas de tecnologia tentando se proteger contra programas avançados. Do lado dos resolvidores, as pessoas envolvidas foram tanto acadêmicos tentando desenvolver novas técnicas para avançar nos modelos de reconhecimento de imagens, quanto *spammers* buscando novas formas de realizar ataques cibernéticos.

⁶ Outro termo para *resolver* Captchas é *quebrar* Captchas. Nesta tese, optou-se por utilizar o termo *resolver*, para enfatizar a interpretação do Captcha como um desafio, não como um problema de criptografia.

Um dos geradores de Captchas mais conhecidos é Luis von Ahn, que também é um dos criadores do Captcha. Um pedaço da história está disponível nos primeiros cinco minutos da entrevista com Luis Von Ahn em um programa chamado *Spark*⁷. Na entrevista, Von Ahn conta um pouco da origem dos Captchas em Carnegie Mellon, contando que ficou frustrado com o fato das pessoas perderem tempo de inteligência humana ao resolver Captchas, o que deu origem ao reCaptcha. Outro vídeo instrutivo é uma palestra de Von Ahn na *Thinking Digital Conference* sobre a história do reCaptcha⁸. Segundo ele, a *startup* foi criada em maio de 2007⁹, depois de von Ahn perceber que aproximadamente 200 milhões de Captchas eram resolvidos diariamente.

O reCaptcha v1 foi uma solução de aproveitar o tempo das pessoas que resolvem Captchas para digitalizar livros (AHN et al., 2008). A ideia do reCaptcha, como demonstrada na Figura 1.1, foi apresentar duas palavras distorcidas para a pessoa: a primeira seria utilizada para verificar se a pessoa era um humano, e a segunda seria utilizada para decifrar um texto que os robôs na época não conseguiam ler. Em 2009, a empresa foi comprada pela Google, que utilizou o reCaptcha para digitalizar os Google Books.

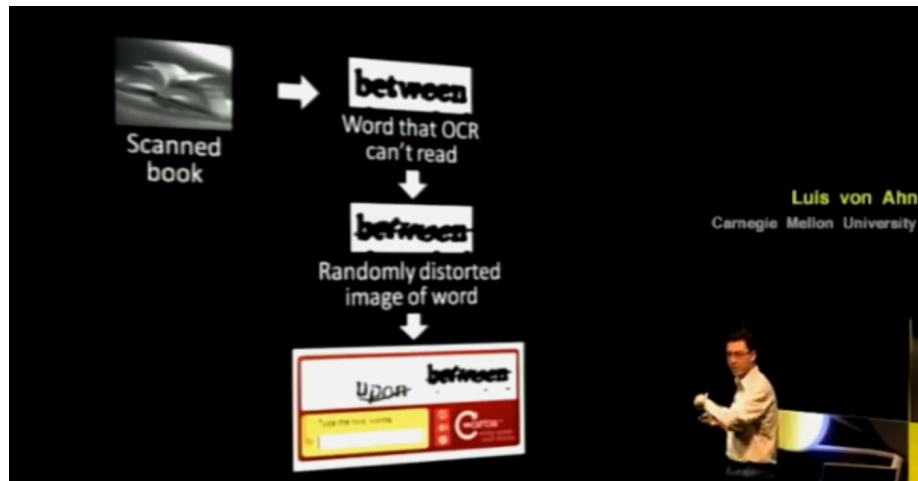


Figura 1.1: Explicação de von Ahn sobre o funcionamento do reCaptcha

Curiosamente, foi com a própria Google que os resolvedores ficaram em vantagem na luta. Os modelos de inteligência artificial continuaram avançando, notadamente com o avanço dos modelos de redes neurais profundas (LECUN; BENGIO; HINTON, 2015a). No trabalho de GOODFELLOW et al. (2013), todos funcionários da Google na época, foi apresentado um modelo de redes neurais convolucionais que resolvia o reCaptcha v1 com 99.8% de acurácia. No ano seguinte, em 2014, a Google descontinuou o reCaptcha v1, lançando o reCaptcha v2.¹⁰

⁷ Spark, 2011. [Link no Web Archive](#). Último acesso em 01/11/2022.

⁸ [Link do vídeo no YouTube](#). Último acesso em 01/11/2022.

⁹ Segundo o [texto da Wired](#): “So he’s fighting back. In late May, von Ahn launched reCaptcha, a service that he believes is the toughest Captcha yet devised. ReCaptcha presents users with two stretched and skewed words, each bisected by a diagonal line”. Último acesso em 01/11/2022.

¹⁰ *Are you a robot? Introducing No Captcha reCaptcha*. Acessível no [blog da Google](#). Último acesso em 01/11/2022.

O reCaptcha v2 apresentou duas inovações importantes. O primeiro foi o botão “*I’m not a robot*”, um verificador automático do navegador que utiliza heurísticas para detectar se o padrão de acesso ao site se assemelha com um robô ou humano. O segundo foi a mudança no tipo de tarefa: ao invés de rotular um texto distorcido, o desafio passou a ser identificar objetos e animais, como na Figura 1.2.

A mudança do tipo de tarefa de visão computacional mudança foi importante para o sucesso do reCaptcha v2. Isso ocorre porque o desafio é mais difícil, já que existem muito mais objetos e imagens do que letras e números, aumentando significativamente o suporte da variável resposta. Por exemplo, um modelo para identificar um reCaptcha de gatos pode ser facilmente desenvolvido a partir de uma base pré-classificada, potencialmente custosa para ser construída. O reCaptcha v2, no entanto, pode facilmente mudar a tarefa para identificar leões, cães, hidrantes e semáforos, inutilizando o modelo criado para classificar gatos. O reCaptcha v2 também foi um sucesso para treinar os modelos desenvolvidos pela própria Google: usando o mesmo princípio do reCaptcha v1, a humanidade era verificada com apenas uma parte das imagens. As outras imagens eram utilizadas para rotular imagens, utilizadas para aprimorar os modelos utilizados nos projetos de carros autônomos, Google Street View e outras iniciativas da empresa.

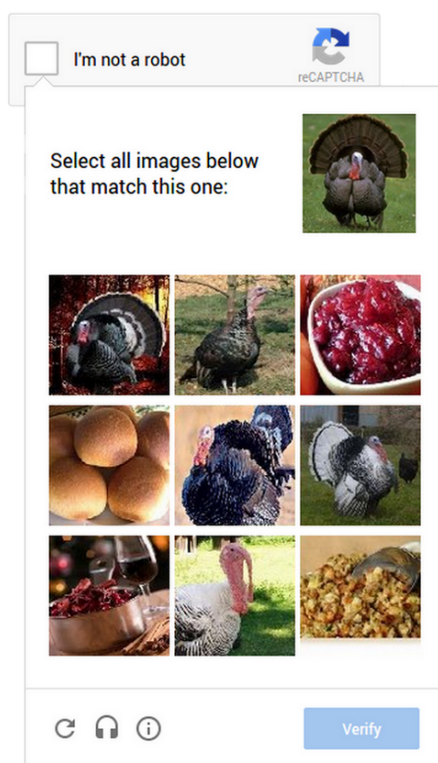


Figura 1.2: Exemplo do reCaptcha v2 com a imagens de perus

Com o advento do reCaptcha v2, a pergunta dos resolvedores de Captcha passou a ser: como criar modelos que funcionam razoavelmente bem nos Captchas sem a necessidade de rotular vários casos? Se esse desafio fosse resolvido, dois avanços aconteceriam: i) um grande avanço na inteligência artificial, especificamente na área de visão computacional, e ii) uma nova forma de vencer a luta.

Até o momento de escrita da tese, não existia um modelo geral que resolvesse com alta acurácia todos os desafios colocados pelo reCaptcha v2 e seus concorrentes. No entanto, vários avanços apareceram no sentido de reduzir a quantidade de imagens rotuladas para criar candidatos a resolvidores. Dentre eles, o mais significativos são os baseados nas redes generativas adversariais (*Generative Adversarial Networks*, ou GANs), propostas no famoso trabalho de GOODFELLOW et al. ([s.d.]). O primeiro trabalho que utiliza modelos generativos no contexto de Captchas mostrou uma redução de 300x na quantidade de dados classificados necessária para resolver um Captcha (GEORGE et al. (2017); nesse caso, os autores propõem uma rede diferente do GAN, chamada *Recursive Cortical Network*, ou RCN). Outros trabalhos mais recentes (WANG et al., 2021; YE et al., 2018a) avançam ainda mais na pesquisa, reduzindo o trabalho de classificação para um novo Captcha de texto para aproximadamente 2 horas.

Mas foi em 2018, com o reCaptcha v3, que a Google deu a palavra final. Com a nova versão, as verificações de navegador passaram a ser muito mais poderosas, sendo raros os casos em que o site fica em dúvidas se a pessoa é ou não um robô. Versões mais recentes, como o reCaptcha *Enterprise*, de 2020, ainda permitem que as pessoas que mantêm os sites façam o ajuste de modelos de detecção de robôs. Os desafios de reconhecimento de texto, objetos ou imagens perderam a importância.

Então, no final, quem venceu a luta de geradores e resolvidores? Na verdade, nenhuma das duas! O que ocorreu com o reCaptcha v3 e seus sucessores foi, no fundo, uma mudança de perspectiva: o Captcha deixou de ser um sistema **passivo** e passou a ser um sistema **ativo** de verificação. Ao invés de criar uma tarefa difícil de resolver por robôs e fácil de resolver por pessoas, os sistemas criaram uma camada de verificação da sessão de acesso do usuário, incluindo análises do navegador, dos *cookies* e dos padrões de cliques. Antes mesmo de chegar no desafio de reconhecimento, o algoritmo de acesso precisa enganar os verificadores. Essa tarefa é muito mais parecida com um problema de *cyberataque* do que uma tarefa de inteligência artificial.

A guerra entre sites e *spammers* continua, mas não é mais uma luta entre geradores e resolvidores. Por conta disso, os desafios dos Captchas, sejam de texto ou de imagem, são hoje muito mais uma questão acadêmica do que uma questão de segurança. A pesquisa sobre Captchas ainda é promissora e pode gerar muitos resultados importantes para a área de inteligência artificial.

Infelizmente, Captchas de textos em imagens continuam sendo populares na *internet*. Isso é especialmente evidente nos serviços públicos – objeto deste trabalho –, já que os serviços raramente são atualizados com ferramentas mais recentes. Desenvolver uma ferramenta que facilita a resolução de Captchas em sites públicos é uma forma de incentivar os sites a serem atualizados. Se o Captcha inseguro parar de fazer efeito prático, os sites terão de atualizar a tecnologia, seja colocando Captchas mais poderosos, que é o resultado indesejado, ou disponibilizando os dados públicos de forma aberta, que é o resultado desejado.

Essa é a lacuna identificada a partir da observação do estado atual dos serviços públicos e dos trabalhos acadêmicos analisados. Nesta pesquisa, será apresentado um fluxo de trabalho que pode ser facilmente aplicado a diferentes modelos de resolução de Captchas, incluindo arquiteturas que ainda não foram desenvolvidas. O fluxo de trabalho funcionará

como um acelerador do aprendizado, possibilitando a criação de modelos que não precisam de intervenção humana. O resultado será encontrado explorando o potencial de uso do **oráculo**, apresentado na próxima seção.

1.3 Oráculo

Modelos de aprendizagem profunda usuais, quando ajustados sem cuidado, são muito sensíveis a perturbações pequenas nas imagens (YUAN et al., 2019). Por isso, se o site de um tribunal, por exemplo, mudar o Captcha utilizado, seria necessário baixar e anotar uma nova base de dados para treinar o modelo. Os avanços em técnicas de regularização fazem com que o modelo seja menos afetado por mudanças nos desafios gerados. Uma técnica de regularização que ajuda na capacidade de generalização é a aumento de dados com adição de ruídos (NOH et al., 2017). No entanto, nenhuma técnica garante que o modelo terá excelentes resultados em novos desafios.

Outra alternativa é desenvolver modelos que aprendem com poucos dados anotados. Como comentado anteriormente, GANs e modelos relacionados podem apresentar bons resultados na resolução de tarefas de imagens, mesmo com uma base de dados anotada. Nesse sentido, ainda que um site mude seu Captcha, é possível ajustar um modelo que resolve esse Captcha sem a necessidade de anotar muitos exemplos para construir uma nova base de treino.

Nessa tese, apresenta-se uma nova técnica para resolver Captchas com poucos dados anotados, chamada de *oráculo*. A técnica consiste em aproveitar o fato de que o Captcha aplica um teste de Turing automático para gerar bases de dados automaticamente. Ou seja, a técnica resolve o problema não com modelos mais sofisticados, mas com a utilização eficiente dos recursos disponíveis. Qualquer modelo pode se aproveitar dessa característica dos Captchas, incluindo arquiteturas que ainda não foram desenvolvidas.

Oráculo é a resposta do site pesquisado, afirmando se o rótulo enviado está correto. Eles estão disponíveis em todos os sites com Captchas, já que, por definição, o Captcha precisa apresentar o resultado do teste para o usuário. O nome “oráculo” foi inspirado na mitologia grega, partindo do fato de que o site já possui a informação correta, como um deus. O site, no entanto, se comunica com o usuário através de um intermediário (o oráculo) que apresenta a resposta de forma limitada.

Oráculos se manifestam de diversas formas nos sites com Captchas. Por exemplo, pode dar a possibilidade de realizar apenas um teste por imagem, vários testes por imagem, ou ainda retornar informações ruidosas. Um exemplo de oráculo ruidoso é o reCaptcha v1, que pode retornar com um “bom o suficiente” quando o rótulo não está totalmente correto (AHN et al., 2008).

O oráculo é uma forma de obter uma base virtualmente infinita. Do ponto de vista de modelagem, é similar a um problema de aprendizado por reforço (SUTTON; BARTO, 2018), mas com uma resposta binária (acertou ou errou) no lugar de um score.

A técnica consiste em utilizar um modelo inicial, possivelmente fraco. Em seguida, o site é acessado múltiplas vezes, gerando uma nova base de dados virtualmente infinita, que é completamente anotada nos casos de acerto e que apresenta o histórico de erros no

caso de erro. O modelo inicial poderia ser ajustado com as técnicas usuais de modelagem, ou utilizando um modelo mais sofisticado como GAN.

Infelizmente, utilizar somente os casos classificados corretamente, obtidos dos acertos no teste do oráculo, induz viés de seleção na amostra (NA et al., 2020). Como o modelo só tem acesso aos casos em que já funciona bem, a informação obtida não é tão relevante. O desafio de modelagem da tese reside em como considerar a informação fornecida pelo oráculo nos casos em que o modelo inicial erra.

Do ponto de vista estatístico, a informação produzida pelo oráculo pode ser entendida como uma informação censurada (COLOSIMO; GIOLO, 2006). Isso acontece pois a informação existe e é correta, mas não está completa. No entanto, como a informação é resultado do teste de um rótulo produzido por um modelo, faz sentido afirmar que a censura não é gerada por acaso.

Na área de aprendizado de máquinas, um modelo apresenta resposta censurada ou incompleta é colocado na classe de problemas do aprendizado fracamente supervisionado (ZHOU, 2018). Trata-se de uma área ainda pouco investigada estudada na literatura, mas bastante ampla, englobando não só os métodos supervisionados como também os métodos semi-supervisionados. A tese apresentará os conceitos de aprendizado fracamente supervisionado, com foco na classe de problemas que a modelagem utilizando Captchas representa.

Para criar uma base de dados usando o oráculo, é necessário utilizar técnicas de raspagem de dados, imitando repetidamente o que um usuário faria para acessar o site. Por isso, a raspagem de dados se torna fundamental para a construção do modelo, tornando-se um dos objetivos da pesquisa.

1.4 Objetivo

O objetivo geral da tese é desenvolver uma solução inovadora, chamada WAWL (*Web Automatic Weak Learning*) para resolver Captchas, misturando técnicas de aprendizado profundo com raspagem de dados e aproveitando os dados fornecidos pelo oráculo.

Especificamente, a pesquisa tem como objetivos:

1. Descrever o modelo proposto e estudar suas propriedades.
2. Construir e disponibilizar um repositório de dados para realização de mais pesquisas no ramo.
3. Ajustar e testar a eficácia do modelo proposto.
4. Disponibilizar um pacote computacional aberto, possibilitando a resolução de Captchas presentes em serviços públicos.

1.5 Justificativa

O presente trabalho é relevante para a ciência por conta de sua importância prática, importância teórica e viabilidade técnica. Os pontos são explicados abaixo.

Captchas em serviços públicos causam desequilíbrio de mercado e incentivam o uso de serviços com formas de remuneração duvidosas. O objetivo 4 vai de encontro direto com esse problema, disponibilizando uma ferramenta gratuita e aberta para resolução de Captchas que, pelo menos até a escrita da tese, pode ser utilizada em dezenas de serviços públicos.

Do ponto de vista teórico, a tese é importante por apresentar uma aplicação muito elegante do aprendizado fracamente supervisionado. No caso do Captcha, como a base de dados fracamente supervisionada é virtualmente infinita, trata-se de uma excelente oportunidade para testar novas técnicas e como elas se comportam empiricamente. Os objetivos 1 e 2 estão relacionadas a essa justificativa.

Finalmente, com relação à viabilidade técnica, o trabalho parte de uma lista de Captchas que já foram resolvidos utilizando aprendizado profundo. Como os Captchas já estão resolvidos, mesmo que a WAWL não apresentasse bons resultados – e apresenta – o projeto ainda teria como subprodutos as bases de dados e o pacote computacional disponibilizados abertamente. O objetivo 3 é o que torna a proposta tecnicamente viável.

1.6 Hipóteses

O projeto foi desenvolvido em torno de duas hipóteses principais. As hipóteses têm origem tanto do levantamento bibliográfico realizado para desenvolver a pesquisa, quanto da experiência pessoal do autor em projetos de pesquisa aplicados.

1. A utilização de aprendizado fracamente supervisionado permite a criação de modelos que resolvem Captchas de textos em imagens sem a necessidade de criar grandes bases anotadas.
2. É possível aliar a área de raspagem de dados com a área de modelagem estatística.

1.7 Organização do trabalho

O segundo capítulo, “metodologia”, contém todos os passos dados para construção da tese, tanto do ponto de vista teórico como prático. Parte-se da definição técnica dos Captchas, chegando até as redes neurais e a classe problema trabalhada de forma ampla. Em seguida, apresenta-se o método WAWL e suas características. Depois, a base de dados é descrita, mostrando as fontes de dados consideradas e as técnicas de raspagem de dados utilizadas. Por último, descreve-se, com detalhes, as simulações realizadas para obtenção dos resultados empíricos.

O terceiro capítulo, “resultados”, apresenta os resultados da pesquisa. Do ponto de vista teórico, são apresentadas as propriedades do modelo proposto. Do ponto de vista empírico, são apresentados os resultados das simulações e outros experimentos realizados com a técnica WAWL. O capítulo também apresenta uma breve discussão dos resultados obtidos.

No quarto e último capítulo, “conclusão”, a pesquisa é concluída, com apresentação das considerações finais e próximos passos. Também foi incluído um apêndice descrevendo e

documentando o pacote {captcha}, criado atingir o objetivo 4 da pesquisa.

Capítulo 2

Metodologia

*If machine learning can decipher captchas,
what's next? Toilets that can read our minds?*
— ChatGPT

O capítulo está organizado em três seções: definição do problema, dados e simulações. A primeira mostra a base matemática do problema estudado e as escolhas de sintaxe e terminologias. A segunda descreve as fontes de dados e o processo de coleta, já que a base foi construída totalmente do zero. A terceira mostra como foram planejadas as simulações para verificar se a solução proposta funciona bem empiricamente.

Na parte de redes neurais, optou-se por trabalhar nas pontes entre os modelos clássicos de estatística e os modelos de redes neurais, mas sem tecer todos os detalhes técnicos que podem ser encontrados em livros didáticos. Antes de 2015, a pesquisa em redes neurais nos departamentos de estatística era uma novidade, sofrendo até certo preconceito por ser uma classe de modelos “caixa-preta”. Com o passar do tempo, no entanto, a área está ficando cada vez mais popular, envolvendo até mesmo trabalhos de iniciação científica no tema. Por isso, optou-se por trazer poucos detalhes da área, focando nas pontes entre os modelos clássicos e as redes neurais. Espera-se que o conteúdo possa ser aproveitado por pessoas interessadas em ensinar redes neurais para estudantes de estatística.

Na seção de dados, procurou-se apresentar a metodologia de coleta em detalhes. Isso foi feito porque a área de raspagem de dados não é comum para estudantes de estatística, existindo até uma percepção entre acadêmicos de que trata-se de uma área separada da estatística. Uma das hipóteses de pesquisa, bem como a solução técnica apresentada neste trabalho é justamente uma ponte entre as duas áreas, justificando um detalhamento maior dos conceitos.

Implementações de raspagem de dados, no entanto, são inconstantes. Um site de interesse pode mudar sua estrutura ou simplesmente trocar o Captcha para um reCaptcha da noite para o dia, alterando completamente o fluxo de coleta. Isso aconteceu, inclusive, com um dos sites mais importantes dentro do contexto da jurimetria: em 2018, o Tribunal de Justiça de São Paulo (TJSP) passou a utilizar o reCaptcha para bloquear ferramentas automatizadas. Qualquer código ou fluxo para acessar as fontes de dados consideradas no

trabalho, portanto, estariam datadas no momento da publicação. Por isso, optou-se por apresentar essa parte de forma genérica e deixar as atualizações para os códigos, que estão disponíveis publicamente e serão mantidos com contribuições da comunidade.

Na seção de simulações, procurou-se descrever os passos dados em detalhe. Nesse caso, a escolha do detalhamento se deu por motivos puramente científicos, para que qualquer pessoa possa reproduzir os passos dados. Dessa forma, pessoas interessadas na área podem replicar os resultados em outros exemplos com alterações mínimas no fluxo, além de sugerir melhorias.

2.1 Definição do problema

O problema a ser trabalhado é um caso de *aprendizado fracamente supervisionado* (ZHOU, 2018). Trata-se de uma generalização do aprendizado supervisionado e também do aprendizado *semi-supervisionado*. Usualmente, a área de aprendizado estatístico (ou aprendizado de máquinas) se concentra em dois tipos de problemas principais: o aprendizado supervisionado e o aprendizado não supervisionado. Isso ocorre principalmente por fins didáticos, pois é mais fácil passar os modelos que fazem parte de cada área.

No entanto, a estatística evolui com os problemas que ocorrem no mundo. E, no mundo, os problemas nem sempre recaem em uma ou outra categoria. O que temos, na verdade, é que os problemas não supervisionados e supervisionados estão conectados, desde que o objetivo de uma pesquisa seja o de prever valores (regressão) ou categorias (classificação).

Nesse sentido, uma área que ficou popular nos últimos anos, até por conta dos avanços na área de Deep Learning, é o *aprendizado semi-supervisionado* (ZHU, 2005). Trata-se de uma classe de problemas contendo uma amostra completamente anotada e uma amostra sem anotações. A amostra sem anotações é usada para compreender como os dados foram gerados, e os parâmetros podem ser compartilhados com a parte supervisionada do modelo. Isso poderia indicar que existem três classes de problemas: o não supervisionado, o supervisionado e o semi-supervisionado.

Mas isso também não representa todas as classes de problemas. Em muitas aplicações reais, obter uma anotação completa e correta pode ser custoso ou até impraticável. Além disso por envolver trabalho humano, é comum que classificações contenham erros. Para lidar com esses casos existe uma área, que generaliza as anteriores, que é o aprendizado fracamente supervisionado.

Aprendizado fracamente supervisionado é um termo guarda-chuva. Dentro da área existem diversos tipos de problemas, como aprendizado semi-supervisionado, aprendizado de múltiplas instâncias (BLUM; KALAI, 1998) e o aprendizado com rótulos incorretos ou incompletos (ZHOU, 2018). O caso dos Captchas com o uso do oráculo será apresentado como outra classe de problemas, chamada ***aprendizado com rótulos parciais*** (*partial label learning*, PLL, (JIN; GHAFRMANI, 2002)). Essa classe apresenta uma especialização ainda mais próxima do problema estudado, chamado ***aprendizado com rótulos complementares*** (*complementary label learning*, (ISHIDA et al., 2017a)).

A interpretação do Captcha como um problema de PLL será apresentada no final do

capítulo. A jornada começa de onde deve começar, com aqueles que são objeto de análise deste trabalho: os Captchas.

2.1.1 Captcha

Captcha é um *desafio* do tipo *desafio-resposta* usado para determinar se a usuário do sistema é um humano. Existem diversos tipos de Captcha diferentes, que envolvem desde identificar textos em imagens até resolver expressões matemáticas complexas.

O foco deste trabalho reside nos Captchas baseados em imagens rotuladas, que é o tipo mais comum. Em seguida, a menos que se mencione ao contrário, todos os Captchas apresentados são desse tipo.

O fluxo completo de um Captcha envolve cinco componentes: um *rótulo*, um *gerador*, uma *imagem*, um agente e um *oráculo*. Um ciclo do Captcha é completado ao seguir os passos:

1. O rótulo é definido, usualmente com algum procedimento aleatório, ocultado do agente.
2. A imagem é gerada a partir do rótulo e apresentada para o agente.
3. O agente preenche sua resposta a partir da imagem (que pode estar certa ou errada)
4. O oráculo verifica se a resposta está correta.
5. Dependendo da resposta, o agente é direcionado para a página autenticada ou para uma página de erro.

A Figura 2.1 esquematiza o fluxo do Captcha.

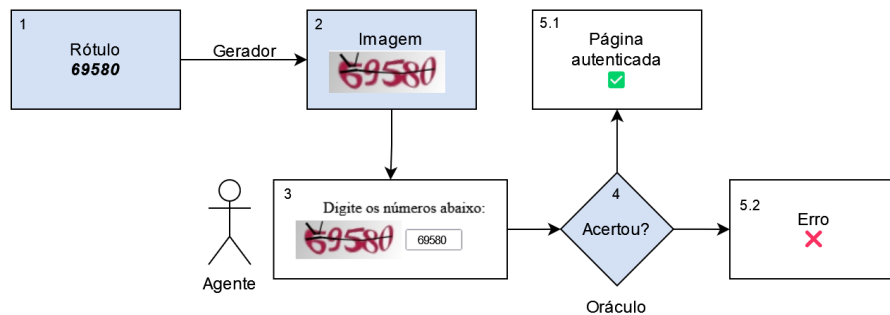


Figura 2.1: Fluxo do Captcha

Imagem, rótulo e variável resposta

A imagem é uma matriz $x = \{x_{nmr} \in [0, 1]\}_{N \times M \times R}$, contendo padrões que, a partir da análise humana, levam ao rótulo do Captcha. O *rótulo* é dado por um vetor de caracteres $c = [c_1, \dots, c_L]^T$. O comprimento L pode ser fixo ou variável, ou seja, duas imagens criadas pelo mesmo gerador podem vir com comprimentos diferentes. Nas definições que seguem considerou-se L como fixo, por simplicidade.

Captchas costumam ter dimensões relativamente pequenas, com a altura N variando entre 30 e 200 *pixels* e a largura M variando entre 100 e 300 *pixels*. As imagens costumam

ser retangulares para comportar várias letras lado a lado, ou seja, geralmente $M > N$. O valor de R é 1 para imagens em escala de cinza e 3 para imagens coloridas.

Os elementos do vetor c fazem parte de um alfabeto \mathcal{A} , com cardinalidade $|\mathcal{A}|$, finito e conhecido. O alfabeto contém todos os possíveis caracteres que podem aparecer na imagem. Na maioria dos casos, \mathcal{A} corresponde a uma combinação de algarismos arábicos (0-9) e letras do alfabeto latino (a-z), podendo diferenciar ou não as letras maiúsculas e minúsculas¹.

O elemento da matriz x_{nm} é denominado *pixel*. Um pixel representa a menor unidade possível da imagem. Em uma imagem colorida, por exemplo, $R = 3$. Nesse caso, um pixel é um vetor de três dimensões com valores entre zero e um, representando a intensidade de vermelho, verde e azul da coordenada (n, m) da imagem. Em imagens em escala de cinza, $R = 1$ e o pixel, de uma dimensão, representa a intensidade do cinza, sendo 1 o equivalente da cor branca e 0 da cor preta.

A Figura 2.2 mostra um exemplo Captcha do Tribunal de Justiça de Minas Gerais (TJMG). Nesse caso, tem-se $L = 5$ e $|\mathcal{A}| = 10$, apenas os dez algarismos arábicos. A imagem tem dimensões $N = 110$, $M = 40$ e $R = 3$. O rótulo da imagem é $[5, 2, 4, 3, 2]^T$.



Figura 2.2: Exemplo de Captcha no TJMG.

A **variável resposta** é uma matriz binária $y_{L \times |\mathcal{A}|}$, em que cada linha representa um dos valores do vetor c , enquanto as colunas possuem um representante para cada elemento de \mathcal{A} . Um elemento y_{ij} vale 1 se o elemento i do rótulo c corresponde ao elemento j do alfabeto \mathcal{A} , valendo zero caso contrário. A variável resposta pode ser pensada também como o *one-hot encoding* do rótulo.

Uma maneira alternativa de definir a variável resposta seria com um vetor de índices representando cada elemento do alfabeto em um vetor. O problema de trabalhar dessa forma é que a variável resposta y tem um número exponencial de combinações: o rótulo possui L caracteres, sendo que cada caractere pode ter $|\mathcal{A}|$ valores, totalizando $|\mathcal{A}|^L$ combinações.

Por exemplo, um Captcha com $L = 6$ letras e $|\mathcal{A}| = 36$ possibilidades em cada letra (26 letras do alfabeto latino e 10 algarismos arábicos), possui um total de 2.176.782.336 (> 2 bilhões) combinações. Por isso, modelar as imagens diretamente através de uma única variável resposta categórica é tecnicamente inviável.

A forma *one-hot* da resposta pode ser entendida como uma **multinomial multivariada** (LI; TSUNG; ZOU, 2014). A resposta é multivariada porque temos L caracteres na imagem e multinomial porque temos $|\mathcal{A}|$ possíveis caracteres em cada posição. Dessa forma,

¹ existem exemplos de Captchas baseados em imagens que não são limitados a letras e números para constituir o rótulo (KAUR; BEHAL, 2014). Como esses casos não aparecem nas aplicações práticas de interesse, estão fora do escopo do trabalho.

podemos pensar que um modelo que resolve o Captcha envolve L classificadores com resposta multinomial, cada um dando conta de um dos caracteres. Os classificadores podem ser independentes e podem até contar com etapas de pré-processamento separadas.

Seguindo o exemplo da Figura 2.2, é possível representar o rótulo da seguinte forma:

$$c = \begin{bmatrix} 5 \\ 2 \\ 4 \\ 3 \\ 2 \end{bmatrix} \rightarrow y = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A forma *dummy* da resposta facilita os trabalhos que seguem. Como será visto mais adiante, o modelo de rede neural gerará uma matriz de probabilidades que somam 1 em cada linha, com as probabilidades de cada caractere em cada posição.

Gerador

O **gerador** é uma função g que recebe um rótulo como entrada e devolve uma imagem como saída. Um bom gerador é aquele que é capaz de gerar uma imagem fácil de interpretar por humanos, mas difícil de se resolver por máquinas.

Um exemplo de gerador é a função `captcha_generate()` criada no pacote `{captcha}`, como descrito no Apêndice A. A função foi criada para realizar simulações do sistema de resolução proposto na tese, a partir do pacote `{magick}` (OOMS, 2021), que utiliza o software *ImageMagick*. A função aplica uma série de distorções e efeitos comuns no contexto de Captchas, gerando imagens como a da Figura 2.3.

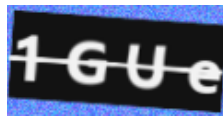


Figura 2.3: Exemplo de captcha gerado pela função `captcha::captcha_generate()`

O gerador segue os passos abaixo, a partir do momento em que um rótulo c existe:

1. É criada uma matriz $N \times M \times R$, com valores entre zero e um gerados por simulações de uma $\mathcal{U}(0, 1)$.
2. É adicionada uma cor base ao ruído, definida de forma aleatória.
3. A matriz é transformada em um objeto do tipo `magick-image`.
4. A imagem é preenchida com o valor do rótulo, adicionando-se efeitos como rotação, uma linha unindo as letras e variação de cores.
5. A imagem recebe outros tipos de distorções, como adição de ruído, alteração de cores e outros efeitos.

No final, o gerador retorna a imagem, que é a única informação enviada ao agente. O rótulo fica escondido para verificação do oráculo.

Oráculo

Para definir o oráculo, utilizou-se uma terminologia que é facilmente encaixada com a teoria de aprendizado fracamente supervisionado. Seja g um classificador utilizado para prever o rótulo de uma imagem e seja X_{n+1} uma nova imagem que é observada, com sua resposta Y_{n+1} , desconhecida. A operação $g(X_{n+1}) = \hat{Y}_{n+1}$ retorna um candidato para Y_{n+1} , que pode estar correto ou errado.

O oráculo é uma função $\mathcal{O} : \mathcal{Y} \rightarrow 2^{\mathcal{Y}}$, ou seja, uma função que recebe um elemento do domínio da resposta \mathcal{Y} (ou seja, do conjunto de todas as combinações de rótulos) para o conjunto de subconjuntos (as partes) de \mathcal{Y} . Na prática, a função retorna uma lista de possíveis valores de Y_{n+1} , da seguinte forma:

$$\mathcal{O}(\hat{Y}_{n+1}) = \begin{cases} \{Y_{n+1}\}, & \text{se } Y_{n+1} = \hat{Y}_{n+1} \\ \mathcal{Y} \setminus \{\hat{Y}_{n+1}\}, & \text{se } Y_{n+1} \neq \hat{Y}_{n+1} \end{cases}$$

Quando o classificador g acerta o rótulo, o oráculo retorna uma lista que contém apenas um elemento: o próprio rótulo. Para simplificar, também é possível utilizar a notação de *rótulo complementar* $Y_{n+1} \neq \hat{Y}_{n+1} = \bar{Y}$. Quando o classificador g retorna o rótulo errado, o oráculo retorna uma lista com todos os outros possíveis rótulos do rótulo, o que inclui o verdadeiro valor Y_{n+1} .

A Figura 2.4 mostra o funcionamento do oráculo no exemplo do TJMG. Quando a predição é igual ao rótulo, o resultado apresentado é o valor um, indicando que o rótulo está correto. Quando a predição é diferente do rótulo, o resultado apresentado é o valor zero, indicando que o valor testado está incorreto e que, portanto, o rótulo real é um dentre todos os outros possíveis rótulos.

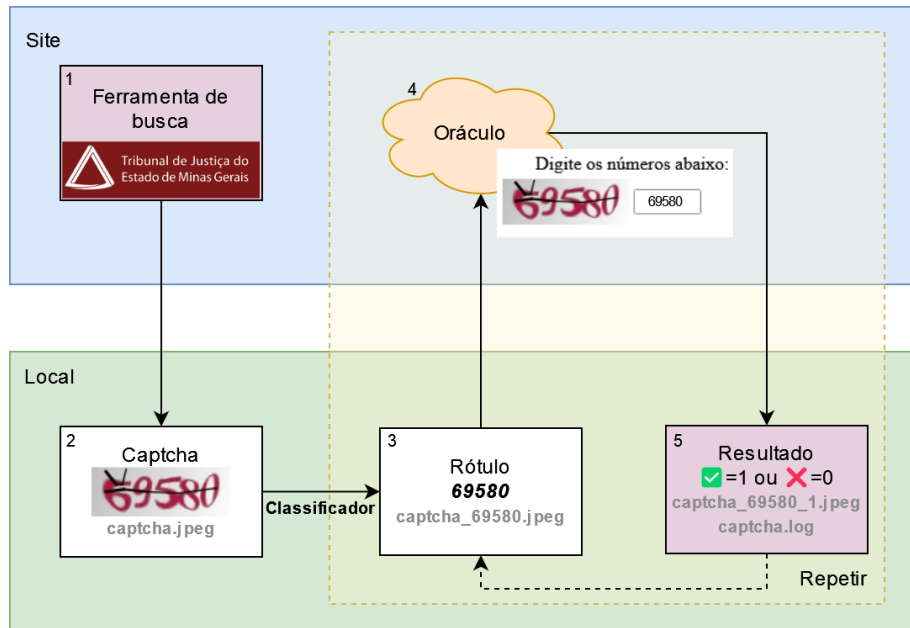


Figura 2.4: Esquema mostrando o funcionamento do oráculo.

É possível generalizar naturalmente o oráculo para múltiplos chutes mudando a

definição da função que faz previsões. Seja h uma função que retorna um conjunto de k respostas possíveis, $k \in \mathbb{N}$, $k \geq 1$, com x_{n+1} e y_{n+1} iguais aos definidos anteriormente. Então o oráculo tem o funcionamento definido abaixo:

$$\mathcal{O}(h(x_{n+1})) = \begin{cases} \{y_{n+1}\}, & \text{se } y_{n+1} \in h(x_{n+1}) \\ y \in h(x_{n+1}), & \text{se } y_{n+1} \notin h(x_{n+1}) \end{cases}.$$

Nesse caso, o oráculo também retorna uma lista com a resposta y_{n+1} . A única diferença é que, quando o Captcha aceita múltiplos chutes, a lista retornada em caso de erro tem um comprimento menor.

O oráculo tem um papel fundamental na solução proposta. O fato do oráculo sempre retornar a resposta correta na lista de opções faz com que ela necessariamente reduza o espaço de respostas a serem buscadas em uma tentativa futura. Esse fato será explorado a partir de um método iterativo para encontrar o valor real do rótulo.

Fatos estilizados

Historicamente, uma alternativa para resolver Captchas é separando o problema em duas tarefas: segmentar e classificar. A tarefa de segmentação consiste em receber uma imagem com várias letras e detectar pontos de corte, separando-a em várias imagens de uma letra. Já a classificação consiste em receber uma imagem com uma letra e identificar o caractere correspondente. Nesse caso, a resposta é reduzida para $|\mathcal{A}|$ categorias, que cresce linearmente e, portanto, tratável.

A tarefa de resolver Captchas também poderia ser vista como um problema de reconhecimento óptico de caracteres (*Optical Character Recognition*, OCR). No entanto, as distorções encontradas em Captchas são bem diferentes das distorções encontradas em textos escaneados, que são o objeto de aplicação de ferramentas de OCR. Por esse motivo, as ferramentas usuais de OCR apresentam resultados pouco satisfatórios em vários Captchas.

As distorções encontradas em Captchas podem ser agrupadas em distorções para dificultar a segmentação e distorções para dificultar a classificação. Na parte de classificação, as principais formas de dificultar o trabalho dos modelos são i) mudar as fontes (serifa ou sem serifa ou negrito/itálico, por exemplo), ii) mudar letras minúsculas para maiúsculas e iii) adicionar distorções nos caracteres. Já na parte de segmentação, as principais formas são i) colar os caracteres e ii) adicionar linhas ligando os dígitos. Essas técnicas são combinadas com a adição de ruído e distorção nas imagens completas para compor a imagem final.

2.1.2 Redes neurais

A abordagem discutida ao longo da tese utiliza redes neurais convolucionais. Para explicar o funcionamento dessa técnica, apresenta-se as definições para redes neurais e para a operação de convolução no contexto de Captchas, construindo o modelo utilizado nas simulações do modelo proposto.

A ideia abaixo é apresentar como funcionam as redes neurais no contexto de Captchas. O modelo apresentado é o que foi utilizado nas simulações, que é um modelo de redes neurais convolucionais simples, similar ao LeNet, com três camadas convolucionais e duas camadas densas (LECUN et al., 1998).

A técnica proposta pela tese pode utilizar diversas arquiteturas de redes neurais. A escolha de uma arquitetura mais simples foi feita para demonstrar a eficácia do procedimento de forma mais contundente. Outras arquiteturas mais rebuscadas, como as apresentadas no referencial teórico (GEORGE et al., 2017; YE et al., 2018b) podem melhorar a aplicação do modelo. A única restrição é que ela possa receber uma função de perda modificada, como será mostrado a seguir.

É possível organizar a estrutura de uma rede neural em três componentes: a **arquitetura da rede**, a **função de perda** e o **otimizador**. Os componentes são detalhados nas próximas subseções.

Como uma rede neural possui muitos componentes e subcomponentes, é usual apresentar sua estrutura na forma de um diagrama. Redes neurais costumam ser fáceis de representar através de grafos, que podem ser utilizados de forma mais ou menos detalhada, dependendo do interesse.

A Figura 2.5 mostra, de forma esquemática, os componentes (retângulos tracejados) e subcomponentes (partes internas dos componentes) do modelo utilizado.

Arquitetura da rede

A arquitetura da rede é uma função que leva os dados de entrada na estrutura de dados da variável resposta. A arquitetura tem papel similar ao exercido pelo componente sistemático em um modelo linear generalizado (NELDER; WEDDERBURN, 1972). Trata-se da parte mais complexa da rede neural, carregando todos os parâmetros que serão otimizados.

A arquitetura da rede possui três componentes principais, separados em dois itens cada:

- as camadas ocultas: camadas **convolucionais** e camadas **densas**;
- as técnicas de regularização: **normalização em lote** (*batch normalization*), **dropout** e junção de pixels (*max pooling*);
- as funções de ativação: função de ativação linear retificada (*rectified linear unit*, ReLU) e a função de normalização exponencial (*softmax*).

Abaixo, apresenta-se as definições seguindo-se a ordem de aplicação das operações na arquitetura da rede neural: camada convolucional, ReLU, *max pooling*, *batch normalization*, *dropout*, camada densa e *softmax*.

A **convolução** é uma operação linear que recebe como entrada uma matriz e retorna outra matriz. Ela é diferente de uma operação usual de multiplicação de matrizes vista no contexto de modelos lineares generalizados, por envolver uma operação nos elementos na vizinhança de cada pixel.

Uma forma organizada de fazer essa soma ponderada é criando uma matriz de pesos.

2.1 | DEFINIÇÃO DO PROBLEMA

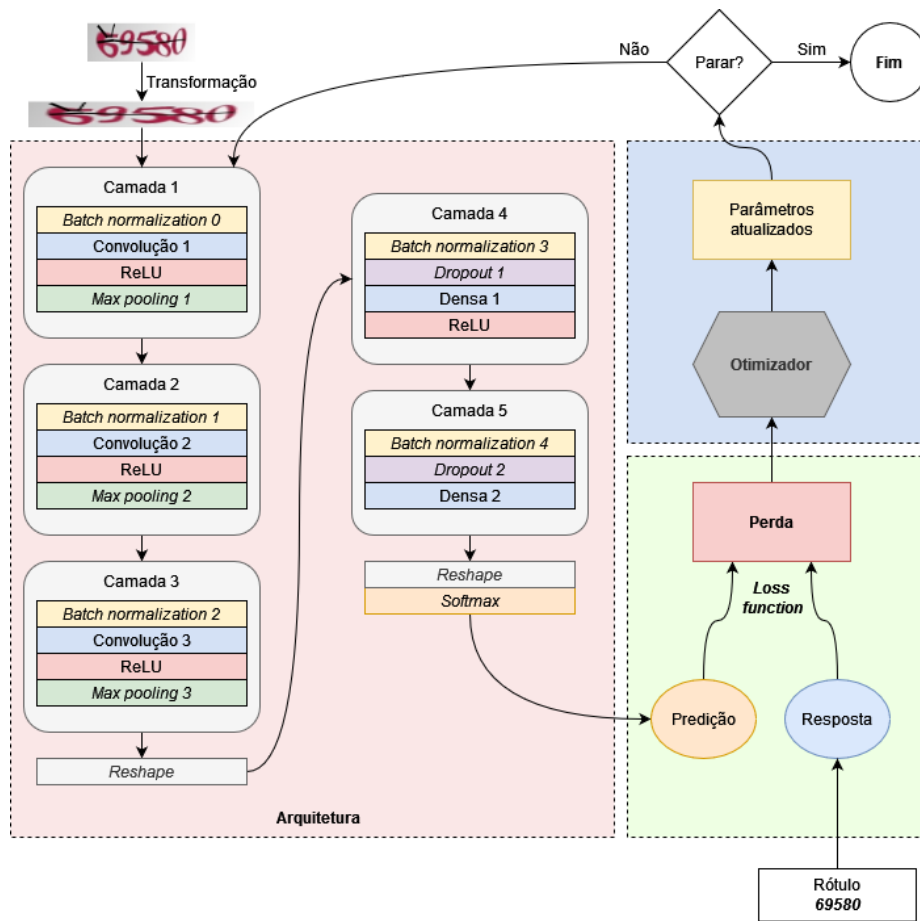


Figura 2.5: Diagrama representando o modelo utilizado de forma genérica, com todos os componentes e subcomponentes apresentados de forma esquemática. As partes de fora dos componentes são entradas de dados ou decisões de parada do ajuste.

Com ela, não é necessário procurar os pontos da vizinhança. Para cada ponto (i, j) , obtém-se a matriz de vizinhança, multiplica-se pontualmente pela matriz de pesos e soma-se os valores resultantes. A matriz de pesos é chamada de núcleo, ou *kernel*.

Considere

$$K = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

e a imagem da Figura 2.6. Como visto anteriormente, trata-se de uma matriz de dimensão $40 \times 110 \times 3$.



Figura 2.6: Imagem de Captcha utilizado em exemplos anteriores.

Tome por exemplo a primeira dimensão do pixel $(i, j, k) = (12, 16, 1)$. A vizinhança

3x3 em torno desse ponto é dada por

$$P_{i,j,k} = \begin{bmatrix} 0.094 & 0.412 & 0.686 \\ 0.051 & 0.063 & 0.529 \\ 0.071 & 0.000 & 0.086 \end{bmatrix}$$

A operação de convolução é feita da seguinte forma:

$$\begin{aligned} (P_{12,16,1} * K)_{12,16,1} = & k_{1,1}p_{11,15,1} + k_{1,2}p_{11,16,1} + k_{1,3}p_{11,17,1} + \\ & + k_{2,1}p_{12,15,1} + k_{2,2}p_{12,16,1} + k_{2,3}p_{12,17,1} + \\ & + k_{3,1}p_{13,15,1} + k_{3,2}p_{13,16,1} + k_{3,3}p_{13,17,1} \end{aligned}$$

Esse é o valor a ser colocado no ponto (i, j, k) . Isso funciona em todos os pontos que não estão na borda da imagem.

Existem duas formas de trabalhar com as bordas da imagem. A primeira é preenchendo as bordas com zeros, de forma a considerar apenas os pontos da imagem. A segunda é descartar os pontos da borda e retornar uma imagem menor, contendo somente os pixels em que foi possível aplicar todo o *kernel*.

No caso do exemplo, o resultado da convolução fica como na Figura 2.7. A matriz não foi escolhida por acaso: ela serve para destacar padrões horizontais da imagem. Como a primeira linha é formada por -1 e a última é formada por 1 , a matriz fica com valor alto se a parte de cima do pixel for preta e a parte de baixo for branca (grande $* 1 +$ pequeno $* (-1)$). A parte destacada da imagem acabou sendo a parte de baixo dos números e, principalmente, a linha que une os números.



Figura 2.7: Aplicação de uma convolução com kernel horizontal.

Aplicando o kernel vertical abaixo

$$K = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix},$$

as partes destacadas são as laterais dos números, conforme Figura 2.8.



Figura 2.8: Aplicação de uma convolução com kernel horizontal.

O resultado da convolução pode ter números negativos ou maiores que um. Para que seja possível visualizar, as imagens mostradas acima foram normalizadas.

Uma característica das imagens mostradas acima é que elas ficaram escuras, ou seja, com muitos valores próximos de zero. Uma técnica para modificar a imagem é adicionar uma constante numérica ao resultado da convolução. Esse é o chamado **viés** (*bias*) da convolução.

A Figura 2.9 mostra o efeito de adicionar um viés de 0.6 após aplicação da convolução com kernel vertical. É possível identificar claramente a diferença entre os números (mais suaves) e as curvas usadas para conectar os números (mais proeminentes).



Figura 2.9: Aplicação de uma convolução com kernel horizontal.

Uma **camada convolucional** envolve a aplicação de convoluções com d kernels em uma matriz, além da adição do *bias*. O resultado da aplicação de uma camada convolucional com preenchimento das bordas é uma matriz com as mesmas dimensões N e M da matriz de entrada, mas com d entradas na dimensão das cores. Como o valor de d pode ser diferente de 1 ou 3, não faz mais sentido tratar essa dimensão como cores, por isso essa dimensão é chamada de **canais** da imagem resultante.

É importante notar que, nos exemplos apresentados anteriormente, a convolução foi aplicada a apenas um dos canais da imagem: o primeiro. Quando a imagem de entrada possui vários canais, camada convolucional aplica cada *kernel* em cada canal da imagem e, depois, faz a soma dos valores resultantes.

A Figura 2.10 mostra um exemplo de aplicação de camada convolucional para a imagem utilizada nos exemplos anteriores. Os *kernels* foram escolhidos com base em um modelo que já foi ajustado para o Captcha. Note que os canais capturam a informação dos números e dos ruídos, focando em detalhes diferentes.

Antes da aplicação da camada convolucional, a operação de **batch normalization** foi aplicada. Essa operação normaliza os números da matriz de entrada antes da aplicação da convolução, retirando a média e dividindo pelo desvio padrão.

$$x_z = \left(\frac{x - \bar{x}}{\sqrt{\sigma_x^2 + \epsilon}} \right) \gamma + \beta$$

O valor ϵ , geralmente um valor pequeno, é adicionado para evitar problemas numéricos quando a variância é muito baixa. Os parâmetros γ e β podem ser adicionados no passo da normalização, fazendo parte do fluxo de aprendizagem do modelo. Apesar de não ser uma teoria fechada, alguns resultados indicam que o uso de *batch normalization* reduz o tempo de aprendizado dos modelos (IOFFE; SZEGEDY, 2015). O passo foi adicionado nos modelos por apresentar bons resultados nas simulações.

Após a aplicação da convolução, também é aplicada a função não linear **ReLU**. A transformação ReLU é a mais simples das funções de ativação, sendo igual à função identidade quando a entrada é positiva e zero caso contrário:



Figura 2.10: Resultado da aplicação da primeira convolução à imagem.

$$\text{ReLU}(x) = x \mathbb{I}_{(x>0)}.$$

A função ReLU serve para tornar a arquitetura do modelo uma operação não linear. Qualquer operação não linear poderia ser utilizada, mas a mais simples e mais popular é a ReLU.

Em seguida, aplica-se uma operação para reduzir a dimensão da imagem, chamada **max pooling**. Trata-se de uma operação que recebe a imagem e um *kernel*, retornando, para cada janela, o maior valor dos pixels. Usualmente, a técnica também utiliza *strides* fazendo com que cada pixel seja avaliado apenas uma vez. Por exemplo, para uma matriz com dimensões $M_{10 \times 10}$ e *kernel* com dimensões 2×2 , o resultado é uma matriz $M_{5 \times 5}^p$ onde cada elemento é o valor máximo da janela correspondente ao pixel.

A operação *max pooling* é muito comum no contexto de redes neurais convolucionais. Sua aplicação é importante para que os *kernels* sejam aplicados em diferentes níveis da imagem de entrada.

A aplicação das camadas convolucionais é repetida três vezes. Ou seja, as seguintes operações são aplicadas a partir da imagem original:

1. *batch normalization*: 6 parâmetros
2. camada convolucional: 896 parâmetros
3. ReLU
4. *max pooling*
5. *batch normalization*: 64 parâmetros
6. camada convolucional: 18.496 parâmetros
7. ReLU

8. *max pooling*
9. *batch normalization*: 128 parâmetros
10. camada convolucional: 36.928 parâmetros
11. ReLU
12. *max pooling*
13. *batch normalization*: 128 parâmetros

A dimensão da imagem de entrada, bem como quantidade de canais gerados por cada camada convolucional foram fixadas. Tais números podem ser considerados como hiperparâmetros do modelo, mas foram fixados para facilitar as simulações, que já contam com diversos hiperparâmetros.

A imagem de entrada foi fixada na dimensão 32×192 . O valor foi definido dessa forma porque um dos Captchas de referência, da Receita Federal do Brasil (RFB), possui 6 letras e $32 * 6 = 192$. Ou seja, é como se a imagem fosse a colagem lado a lado de 6 imagens 32×32 .

A quantidade de canais gerados pelas camadas convolucionais foram fixadas em 32, 64 e 64. A utilização de números crescentes de canais nas camadas convolucionais é comum (LECUN et al., 1998), bem como a utilização de números que são potências de 2 (LECUN; BENGIO; HINTON, 2015b). Nesse sentido, um possível valor para a terceira camada era de 128 canais, mas optou-se por 64 canais para que a quantidade de parâmetros não ficasse grande demais, já que isso exigiria mais tempo de computação e computadores mais poderosos.

O total de parâmetros que podem ser otimizados até o final das camadas convolucionais é 56.646. Esse número pode parecer grande no contexto de modelos estatísticos tradicionais como uma regressão linear, que teria, considerando cada pixel como uma covariável, 4.401 parâmetros (40×110 e o intercepto). No entanto, é uma quantidade relativamente pequena no contexto de redes neurais. Redes neurais recentes aplicadas a imagens, como o DALL-E 2 possui 3,5 bilhões de parâmetros (RAMESH et al., [s.d.]).

Em seguida, o resultado é transformado para um formato retangular, similar ao que se encontra em modelos de regressão. Aqui, as dimensões da imagem não são mais importantes e os pixels de cada canal são tratados como variáveis preditoras. Esse passo pode ser interpretado da seguinte forma: as camadas convolucionais funcionam como um pré-processamento aplicado às imagens, como uma engenharia de variáveis (KUHN; JOHNSON, 2019) otimizada, já que os parâmetros são ajustados no modelo.

Uma vez obtidas as variáveis preditoras com o pré-processamento, é a hora de aplicar as camadas densas. Tais camadas são as mais comuns no contexto de redes neurais. Nesse caso, a operação linear aplicada é uma multiplicação de matrizes, similar ao que é feito em um modelo linear generalizado. Na verdade, o componente sistemático de um modelo linear generalizado é equivalente a uma camada densa com a aplicação de viés, com a função de ativação da fazendo o papel da função de ligação.

Assim como existem os canais das camadas convolucionais, existem os filtros das camadas densas. A quantidade de filtros define a dimensão do vetor de saída. O número de parâmetros da camada densa é igual ao número de itens no vetor de entrada multiplicado pelo número de filtros, somado à quantidade de filtros novamente, por conta do *bias*. No

caso do exemplo, a saída das camadas convolucionais tem dimensão $2 \times 22 \times 64$, ou seja, 64 canais de imagens 2×22 . Com a transformação em vetor, a quantidade de colunas da base passa a ser a multiplicação das dimensões, ou 2.816. No modelo ajustado que foi utilizado como exemplo, aplicou-se 200 filtros na camada densa, totalizando 563.400 parâmetros. Nas simulações, a quantidade de filtros foi variada para produzir modelos com menor ou maior capacidade.

É no contexto da grande quantidade de parâmetros que entra o conceito do *dropout* (BALDI; SADOWSKI, 2013). Trata-se de uma regra de regularização muito simples de implementar, mas que possui grande impacto no ajuste dos modelos. A técnica consiste em selecionar uma amostra dos parâmetros em uma das camadas e apagá-los, forçando que os valores sejam fixados em zero. Na prática, essa técnica obriga o modelo a ser ajustado de forma que amostras aleatórias dos parâmetros sejam boas para prever a variável resposta. Quando o modelo ajustado é usado para inferências, o *dropout* é desativado e o modelo pode utilizar todos os parâmetros, obtendo-se, na prática, uma média ponderada das predições de cada sub-modelo. Dessa forma, o dropout tem um efeito similar à aplicação da técnica de *bagging* (GALAR et al., 2011), muito utilizada na área de árvores de decisão.

O *dropout* é aplicado após a finalização das camadas convolucionais. Em seguida, vem a primeira camada densa, um ReLU e um *batch normalization*. Depois, é aplicada mais um *dropout* e mais uma camada densa. Com isso, a aplicação de operações é finalizada. O total de parâmetros na configuração do modelo apresentado foi de 630.496. Os modelos mais simples utilizados nas simulações, com 100 filtros na camada densa, têm 343.696. Os mais complexos, com 300 filtros na camada densa, têm 917.396 parâmetros.

Para finalizar a arquitetura do modelo, as quantidades resultantes devem ser ajustadas ao formato da variável resposta. O número de filtros da segunda camada densa precisa ser escolhido cuidadosamente, pois deve ser igual à multiplicação das dimensões da variável resposta. No caso do TJMG, os rótulos têm comprimento igual a 5 e vocabulário de comprimento 10 (algarismos arábicos), organizados em uma matriz 5×10 , com 50 entradas. Por isso, a quantidade de filtros da última camada densa também é 50, e o vetor de saída é formatado para uma matriz de dimensão 5×10 .

No final, o resultado precisa ser normalizado para que fique no mesmo escopo de variação da resposta. A resposta possui apenas zeros e uns, sendo que cada linha da matriz tem somente um número “1”, correspondendo ao índice do rótulo no alfabeto e, nas outras entradas, o valor zero. A saída do modelo deve, portanto, apresentar números entre zero e um que somam 1 em cada linha.

Isso é feito através da função *softmax*, aplicada a cada linha da matriz de saída. A função softmax é uma normalização que utiliza a função exponencial no denominador, forçando que a soma dos valores do vetor seja um.

$$\text{soft max}(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{|A|} e^{y_j}}$$

No exemplo, a saída do modelo é a matriz abaixo:

$$\hat{z} = \begin{bmatrix} -17.5 & -13.5 & -15.4 & -6.6 & -9.9 & 9.9 & -11.4 & -10.9 & -11.8 & -9.3 \\ -10.9 & -15.6 & 8.3 & -6.5 & -11.0 & -10.3 & -10.0 & -5.8 & -11.4 & -15.1 \\ -10.5 & -13.6 & -9.6 & -11.4 & 11.2 & -14.3 & -9.9 & -11.3 & -9.9 & -10.0 \\ -18.1 & -9.6 & -10.9 & 5.3 & -10.1 & -6.6 & -15.5 & -13.3 & -6.8 & -10.8 \\ -11.3 & -8.7 & 6.4 & -7.0 & -6.1 & -9.2 & -18.9 & -10.3 & -16.1 & -9.6 \end{bmatrix}.$$

Note que a matriz apresenta valores negativos e positivos. Na primeira linha, por exemplo, o valor positivo está na sexta coluna, correspondendo ao algarismo “5”. De fato, esse é o valor do primeiro elemento do rótulo para esta imagem. Após a aplicação do *softmax*, a matriz de previsões obtida é a matriz abaixo. O modelo de exemplo aparenta ter confiança nas respostas, já que dá probabilidades bem altas para alguns valores e quase zero para outros valores.

$$\hat{y} \times 1000 = \begin{bmatrix} 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1000.0 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 1000.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 1000.0 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 999.99 & 0.00 & 0.01 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 999.99 & 0.00 & 0.00 & 0.00 & 0.00 & 0.01 & 0.00 & 0.00 \end{bmatrix}.$$

Vale notar que, dependendo da implementação, nem sempre é necessário aplicar a função *softmax*. Em alguns pacotes computacionais como o *torch*², utilizado nesta tese, a normalização pode ser feita diretamente na função de perda, que aproveita a expressão completa para realizar algumas simplificações matemáticas e, com isso, melhorar a precisão das computações. O uso da função de perda ficará claro na próxima subseção.

Perda

A função de perda utilizada em um problema de classificação deve levar em conta as probabilidades (ou log-probabilidades) associadas aos rótulos. A perda deve ser pequena se a probabilidade associada ao rótulo correto for alta e a perda deve ser grande se a probabilidade associada ao rótulo correto for baixa.

Uma função de perda natural e popular nesse sentido é a de entropia cruzada, ou *cross-entropy*. Trata-se de uma perda com a formulação

$$\ell(g(x), y) = - \sum_{i=1}^c \mathbb{I}(y = i) \log(g_i(x)),$$

em que $g_i(x)$ é a probabilidade dada ao rótulo i pela função g . Se o rótulo i é diferente do rótulo correto y , a função de perda vale zero por conta da função indicadora. Quando $i = y$, a perda é igual ao oposto do logaritmo da probabilidade associada ao rótulo i . Quanto menor a probabilidade, maior o valor da perda.

² Mais sobre o (py)torch: <https://pytorch.org/>. Último acesso em 22 de novembro de 2022.

Ao trabalhar com o oráculo, a entropia cruzada passa a não fazer sentido nos casos em que o modelo inicial erra. Por isso, a função de perda terá de ser adaptada no método WAWL.

Otimizador

O otimizador utilizado para os modelos ajustados na tese foi o ADAM (KINGMA; BA, [s.d.]). A sigla significa *Adaptive Moment Estimator* e funciona como uma extensão da descida de gradiente estocástica (LECUN et al., 2012), atualizando os parâmetros da seguinte forma:

$$\begin{aligned} m_{\theta}^{(t+1)} &\leftarrow \beta_1 m_{\theta}^{(t)} + (1 - \beta_1) \nabla_{\theta} L^{(t)} \\ v_{\theta}^{(t+1)} &\leftarrow \beta_2 v_{\theta}^{(t)} + (1 - \beta_2) (\nabla_{\theta} L^{(t)})^2 \\ \hat{m}_{\theta} &= \frac{m_{\theta}^{(t+1)}}{1 - \beta_1^t} \\ \hat{v}_{\theta} &= \frac{v_{\theta}^{(t+1)}}{1 - \beta_2^t} \\ \theta^{(t+1)} &\leftarrow \theta^{(t)} - \eta \frac{\hat{m}_{\theta}}{\sqrt{\hat{v}_{\theta} + \epsilon}}, \end{aligned}$$

onde m e v são médias moveis para atualização dos parâmetros, ponderando a perda e a perda ao quadrado com o passo anterior usando pesos β_1 e β_2 , respectivamente. Nessa notação η é a taxa de aprendizado, um hiperparâmetro a ser ajustado. Por último, o valor de ϵ é uma constante, usualmente pequena, para evitar divisão por zero.

2.1.3 Aprendizado estatístico

Apresentados o objeto de estudo, as redes neurais utilizadas e a proposta da pesquisa, passa-se a discutir o significado disso tudo no contexto de aprendizado estatístico. Essa parte foi escrita para proporcionar a base teórica e a notação para apresentar as propriedades do modelo WAWL.

O aprendizado fracamente supervisionado pode ser dividido em três tipos principais. A supervisão com erros, a supervisão com rótulos incompletos e a supervisão de grupos de observações. O caso do Captcha pode ser entendido como uma sub-área do aprendizado fracamente supervisionado com rótulos incompletos chamada aprendizado com dados parcialmente rotulados (*partial label learning*, PLL), já que uma parte da base pode ser anotada sem erros e uma parte da base é a resposta do oráculo indicando uma lista de rótulos possíveis incluindo o correto.

A área de PLL não é nova (GRANDVALET, 2002) e aparece com outros nomes, como aprendizado com rótulos ambíguos (HÜLLERMEIER; BERINGER, 2006) e aprendizado de rótulos em superconjuntos (*superset-label learning*) (LIU; DIETTERICH, 2012). Um caso particular de PLL, aplicável ao tema do Captcha são rótulos complementares (ISHIDA et al., 2017b), que considera os chutes errados na notação do problema.

As definições seguem uma terminologia adaptada a partir da leitura de JIN; GHAHRA-MANI (2002), COUR; SAPP; TASKAR (2011) e FENG et al. (2020a). Sempre que possível,

os casos são adaptados para o problema do Captcha diretamente. Quando necessário, apresenta-se primeiro a definição genérica e depois a formulação para o Captcha.

Em um problema de aprendizado supervisionado tradicional, tem-se um conjunto de casos rotulados $S = \{(x_i, y_i), i = 1, \dots, m\}$ com uma distribuição $p(X, Y)$ desconhecida, onde $X \in \mathcal{X}$ é uma imagem e Y é o rótulo, que possui $|A|^L$ possíveis valores. O objetivo é obter um classificador g que leva um valor de x para o rótulo correto y .

Para delimitar se o resultado da aplicação do classificador está bom ou ruim, utiliza-se uma função de perda. No caso do Captcha, como o interesse é simplesmente acertar o rótulo inteiro (não importa se o classificador acerta só uma parte do rótulo), utiliza-se uma função chamada 0-1:

$$\mathcal{L}(g(x), y) = \mathbb{I}(g(x) \neq y), \quad (2.1)$$

em que $\mathbb{I}(\cdot)$ é uma função indicadora. Como a função de perda é aplicada a apenas um par (x, y) , define-se formalmente que o objetivo do problema de aprendizado é minimizar o *risco*, que é o valor esperado da função de perda:

$$\mathcal{R}(g) = \mathbb{E}_{p(X, Y)}[\mathcal{L}(g(X), Y)]. \quad (2.2)$$

A função de risco, no entanto, não é observada, já que depende da distribuição desconhecida de $p(X, Y)$. Para lidar com esse problema, usualmente é utilizado um estimador do risco, calculado tanto em bases usadas na validação cruzada quanto na base de teste.

$$\hat{\mathcal{R}}(g) = \sum_{i=1}^n \ell(g(x_i), y_i)$$

Na base de teste, utilizada para estimar o risco, a função de perda 0-1 é apropriada. Na etapa de validação cruzada de um modelo de aprendizado profundo, é útil considerar uma aproximação da função de perda que seja contínua e derivável, funcionando como uma versão suavizada da perda 0-1. A partir de um vetor de parâmetros θ originados da arquitetura do modelo, uma escolha de função de perda é a entropia cruzada, como mostrado anteriormente. Os parâmetros são estimados a partir de um otimizador, como o ADAM, apresentado na Seção 2.1.2.

As definições começam a precisar de ajustes quando y deixa de ser um rótulo fixado. Como descrito na Seção 2.1.1, a base de dados observada contém tanto rótulos observados de forma exata quanto rótulos apenas parcialmente informados. Nesse caso, os dados são gerados por uma distribuição

$$p(X, Y, \bar{Y}) = p(X, Y)p(\bar{Y}|X, Y),$$

em que \bar{Y} é um conjunto de rótulos *incorretos*. Nesse caso observam-se, além das instâncias (x_i, y_i) quando o modelo inicial acerta, as instâncias (x_j, \bar{y}_j) quando o modelo inicial erra. Supondo que \bar{Y} é condicionalmente independente de Y dado X , temos

que

$$p(X, Y, \bar{Y}) = p(X, Y)p(\bar{Y}|Y).$$

No caso dos Captchas, essa suposição é verificada. A probabilidade do modelo inicial errar depende apenas do rótulo e não das distorções realizadas pela imagem gerada a partir do rótulo. Além disso, a partir do modelo inicial, é possível estimar os valores de $p(\bar{Y}|Y)$ a partir da base de teste utilizada para medir a acurácia do modelo.

Nos casos em que $|\hat{Y}| = 1$, as probabilidades $p(\bar{Y}|Y)$ podem ser organizadas em uma matriz de transição Q , contendo as probabilidades de se obter um rótulo incorreto para cada possível valor do rótulo. Isso acontece nos Captchas em que não é possível realizar múltiplos chutes. Para resolver problemas desse tipo, é possível realizar um ajuste na função de predição que a torna a função de perda consistente e com taxa de convergência conhecida (YU et al., 2018):

$$f_{\text{adj}}(X) = Q^\top f(X)$$

O tipo de problema apresentado acima é conhecido como *biased complementary label*, ou seja, rótulo complementar com viés. Também é possível considerar um caso sem viés, ou seja, quando $p(\bar{Y}|Y) = \frac{1}{c-1}$ para todos os valores de Y . Esse caso também foi resolvido do ponto de vista teórico (ISHIDA et al., 2017a). As conclusões são parecidas, ou seja, é possível encontrar taxas de convergência para que o problema com rótulos complementares se aproxime de um problema com observações completas.

Quando os rótulos complementares não apresentam viés, existe ainda uma extensão para rótulos complementares múltiplos (FENG et al., 2020b). Neste caso, é possível derivar uma função de risco empírica que, novamente, converge para a função de risco do problema completamente supervisionado, além de apresentar taxas de convergência para essa função de risco.

O caso do oráculo e dos Captchas é um problema com múltiplos rótulos complementares e com viés. Até o momento, não existe uma solução geral para este tipo de problema. No entanto, espera-se que as soluções para problemas desse tipo tenham taxas de convergência mais estreitas do que o caso de rótulos complementares, com ou sem viés, já que rótulos complementares múltiplos trazem mais informação do que rótulos complementares simples.

2.2 Método WAWL

O método WAWL (*Web Automatic Weak Learning*) é a solução proposta na pesquisa. Trata-se da técnica baixar dados da web para compor parte da amostra que é utilizada no ajuste do modelo.

O método WAWL é inovador por dois motivos. Primeiro, porque o método faz a ponte entre áreas que até o momento eram partes separadas do ciclo da ciência de dados: a ras-

pagem de dados e o aprendizado estatístico. Além disso, o método é uma nova alternativa para resolver Captchas com pouca ou nenhuma intervenção humana.

Existem duas formas principais de aplicar o método WAWL. A primeira criando novas bases de treino a partir de um modelo inicial e atualizando os modelos com os dados baixados. A segunda é baixando os dados dentro do próprio ciclo de ajuste do modelo, acessando a web no momento de construção de um *minibatch*.

A arquitetura do modelo WAWL pode ser a mesma de um modelo ajustado com uma base completamente anotada. O modelo pode, inclusive, aproveitar os parâmetros já ajustados em uma eventual versão inicial do modelo para acelerar o aprendizado. Nada impede, no entanto, que uma arquitetura diferente seja utilizada, desde que a entrada seja uma imagem e a saída seja uma matriz com as dimensões da variável resposta. O WAWL é agnóstico à arquitetura do modelo.

A função de perda deve ser adaptada para considerar a informação limitada fornecida pelo oráculo. Quando o rótulo fornecido pelo modelo está correto, a informação é considerada normalmente, através da função de perda da regressão multinomial multivariada. Já quando o rótulo fornecido pelo modelo é incorreto, a função de perda é calculada com base na probabilidade do rótulo estar incorreto:

$$1 - p(y|\theta),$$

Considerando o rótulo complementar \bar{y} e a função \hat{f} dada pela rede neural, a fórmula para descrever a função de perda é descrita da seguinte forma:

$$l(\bar{y}, \hat{f}(x)) = -\log \left[1 - \sum_y \hat{f}_y(x) \mathbb{I}(y = \bar{y}) \right]$$

A função de perda proposta pode ser explicada de maneira intuitiva através de um exemplo. Considere um problema com apenas c possíveis valores para o rótulo (ou seja, uma resposta multinomial, sem ser multivariada). Considere também que a rede neural retorna uma alta probabilidade, por exemplo, 0.99, para o valor i , que o oráculo identificou como incorreta. Nesse caso, a função de perda é dada por

$$l(i, \hat{f}(x)) = -\log [1 - \hat{f}_i(x)] = -\log [1 - 0.99] = 4.61$$

Como é possível ver no exemplo, quanto maior a probabilidade dada a um rótulo identificado como incorreto pelo oráculo, mais a função de perda penaliza essa predição. Dessa forma, a função de perda consegue incorporar completamente a informação dada pelo oráculo.

Quando o Captcha aceita múltiplos chutes, a mesma conta é válida, bastando subtrair as probabilidades de todos os rótulos incorretos:

$$l(\bar{y}, \hat{f}(x)) = -\log \left[1 - \sum_y \hat{f}_y(x) \mathbb{I}(y \in \bar{y}) \right]$$

No final, o valor que é passado para a função de perda é a soma das perdas para todas as observações do *minibatch*. A soma considera tanto as perdas calculadas com base nos rótulos corretos quanto as perdas calculadas com base nos rótulos incorretos.

O otimizador que obtém novas estimativas dos parâmetros também não precisa ser modificado. Basta aplicar a mesma técnica utilizada na modelagem usual, como descida de gradiente estocástica ou métodos adaptativos, como *RMSProp* ou *Adam*.

Um detalhe importante sobre o método é sobre a implementação. Com a utilização de ferramentas que fazem diferenciação automática como o *torch* e o *TensorFlow*³, basta implementar a parte da arquitetura, a função de perda e especificar o otimizador, já que o processo de atualização dos parâmetros é feito automaticamente. No entanto, dependendo da implementação, não é possível fazer a atualização dos parâmetros usando o componente de computação gráfica, que potencialmente acelera o ajuste dos modelos de forma significativa. Na implementação atual, a função de perda apresentada não permite utilização desse componente, sendo uma melhoria possível em futuros trabalhos.

O ajuste dos modelos, tanto para simulações quanto para construção dos modelos finais, utilizou o pacote `{torch}` (FALBEL; LURASCHI, 2022a), que é uma implementação do PyTorch para a linguagem de programação R (R CORE TEAM, 2021). O pacote `{luz}` (FALBEL, 2022a) foi utilizado para organizar as funções de perda e hiperparâmetros, enquanto o pacote `{torchvision}` (FALBEL, 2022b) foi utilizado para utilidades no tratamento de imagens.

2.3 Dados

Nesta seção, descreve-se em detalhes como foi a obtenção dos dados para realizar a pesquisa. Como comentado anteriormente, a base foi construída do zero para os fins do projeto, sendo uma parte significativa dos esforços para chegar nos resultados.

No total, foram construídas bases de dados de dez Captchas que estavam disponíveis publicamente no período de realização da pesquisa. Os Captchas foram revisados pela última vez no dia 14/09/2022, para verificar se ainda estavam ativos. Além disso, foram construídas duas bases de dados de Captchas desenvolvidos internamente para fins de teste.

Parte dos dados foram obtidos como um passo intermediário das simulações. A presente seção descreve como os robôs de coleta foram construídos, bem como a metodologia para obter rótulos via classificação manual. Na subseção de dados da seção de simulação, é possível acessar informações sobre os dados baixados para realizar as simulações.

³ Mais detalhes em <https://www.tensorflow.org/>. Último acesso em 22 de novembro de 2022.

2.3.1 Escolha dos Captchas analisados

Para selecionar os Captchas, foram adotados alguns critérios objetivos. Os critérios foram:


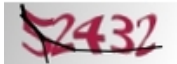








1. O site acessado é de um serviço público (governo federal, tribunal, etc).
2. O Captcha contém letras (A a Z) e números (0 a 9) em uma imagem com extensão jpeg ou png.
3. O comprimento do Captcha é fixo, ou seja, dois Captchas da mesma origem devem ter sempre o mesmo comprimento.

A primeira restrição para escolha dos Captchas é de ordem principiológica. Um serviço público não deveria restringir o acesso aos dados para robôs. Como já discutido anteriormente, nesses casos, a existência do Captcha não tem como finalidade dar maior segurança ao serviço prestado, mas sim limitar o acesso aos servidores por robôs.

As restrições 2 e 3 foram escolhidas com o objetivo de facilitar as simulações para obtenção dos resultados. Em princípio, nada impede que os modelos desenvolvidos trabalhem com outros tipos de rótulos, desde que exista uma lista prévia de rótulos. Além disso, é possível realizar adaptações no pré-processamento base de dados para lidar com diferentes comprimentos de Captchas.

A Tabela 2.1 mostra os Captchas trabalhados. Dos 10 exemplos trabalhados, 6 têm origem em tribunais, que são conhecidos por não disponibilizarem os dados de forma aberta.

Tabela 2.1: *Lista de captchas analisados.*

Captcha	Exemplo	Descrição
trf5		Tribunal Regional Federal 5
tjmg		Tribunal de Justiça de Minas Gerais
trt		Tribunal Regional do Trabalho 3
esaj		Tribunal de Justiça da Bahia
jucesp		Junta Comercial de São Paulo
tjpe		Tribunal de Justiça de Pernambuco
tjrs		Tribunal de Justiça do Rio Grande do Sul
cadesp		Centro de Apoio ao Desenvolvimento da Saúde Pública
sei		Sistema Eletrônico de Informações - ME
rfb		Receita Federal

Além dos Captchas de sites, também foram consideradas imagens geradas artificialmente. O motivo de criar Captchas artificiais é a facilidade de rodar modelos e simulações, já que nos casos reais é necessário ter acesso à internet e também construir bases de dados de cada Captcha.

Foram gerados dois tipos de Captchas artificiais. O primeiro, chamado **MNIST-Captcha**, é simplesmente uma adaptação da conhecida base MNIST para ficar no formato de um Captcha. A partir da escolha do comprimento e dos caracteres que fazem parte da imagem, o gerador simplesmente faz uma amostra aleatória da base do MNIST e compõe as imagens horizontalmente.

A Figura 2.11 mostra um exemplo do Captcha gerado a partir da base MNIST. No exemplo, o comprimento escolhido para o Captcha foi de 4 valores.



Figura 2.11: *Exemplo de MNIST-Captcha*

O problema do MNIST-Captcha é que a base de dados original é finita. Apesar de possuir por volta de 60 mil observações e de um Captcha crescer em ordem exponencial, o MNIST-Captcha pode gerar Captchas repetidos. Além disso, é necessário tomar cuidado com as bases de treino e teste, já que os elementos de teste não poderiam fazer parte de nenhuma observação de treino.

Pelos motivos supracitados, também foi criado um Captcha gerado inteiramente por programação, chamado **R-Captcha**. O Captcha é gerado utilizando a ferramenta ImageMagick, com a possibilidade de customizar diversos parâmetros, como

- Quais caracteres usar na imagem
- O comprimento do Captcha
- Dimensões da imagem
- Probabilidade de rotação da imagem
- Probabilidade de adicionar um risco entre as letras
- Probabilidade de adicionar uma borda nas letras
- Probabilidade de adicionar uma caixa (retângulo) em torno das letras
- Probabilidade de adicionar um ruído branco no fundo da imagem
- Probabilidade de adicionar efeitos de tinta óleo e implosão

A Figura 2.12 mostra um exemplo de R-Captcha. O exemplo apresenta uma linha ligando as letras, comprimento 4, dígitos maiúsculos e minúsculos e distorções.

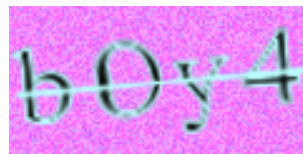


Figura 2.12: *Exemplo de MNIST-Captcha*

Por ser uma versão mais flexível e completa, optou-se por trabalhar principalmente com o R-Captcha nas simulações. O MNIST-Captcha foi implementado mas não foi utilizado nas simulações.

2.3.2 Construção dos dados

Para obter os dados da pesquisa, foram utilizadas técnicas de raspagem de dados (ZHAO, 2017b). A raspagem de dados é uma área da ciência da computação responsável por criar rotinas que automatizam a coleta de dados provenientes da web. Trata-se de uma atividade muito comum em pesquisas aplicadas, especialmente as que envolvem análise de dados públicos que não estão disponíveis de forma aberta, como os dados do Judiciário.

Dentro do ciclo da ciência de dados, pode-se considerar que a raspagem de dados está

inserida nas tarefas de coleta e arrumação de dados. De certa forma, é possível comparar a raspagem com uma consulta a um banco de dados remoto, ou mesmo à obtenção de informações através de uma *Application Programming Interface* (API).

Para raspar uma página da web, usualmente se segue o fluxo descrito na Figura 2.13. Nem todos os passos foram seguidos na obtenção dos dados necessários para realizar as simulações, mas é importante conhecê-los para compreender bem a origem da ideia de utilizar raspagem em conjunto com métodos de aprendizado de máquinas. O exemplo da RFB foi utilizado para dar contexto aos passos.

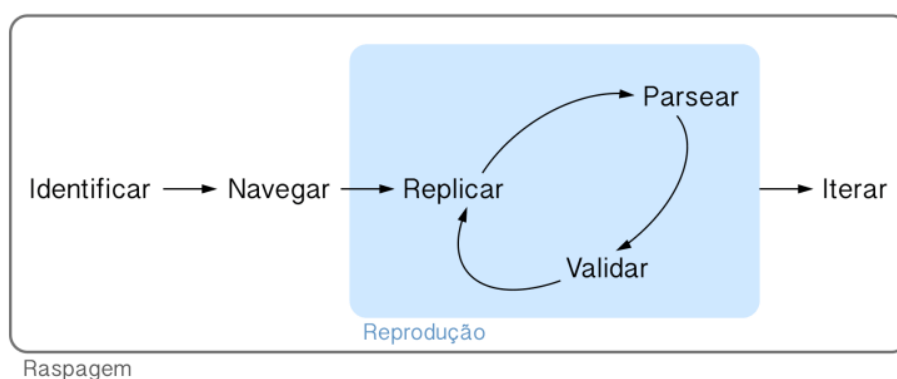


Figura 2.13: Ciclo da raspagem de dados. Fonte: *curso de Web Scraping da Curso-R*.

No caso da RFB, o trabalho é iniciado acessando-se a **página inicial de busca de CNPJ**, como mostrado na Figura 2.14. É possível notar que o desafio disponível é do tipo *hCaptcha*, que não é o Captcha de interesse da pesquisa. No entanto, ao clicar em “Captcha Sonoro”, é possível acessar o Captcha de interesse, como mostrado na Figura 2.15. O motivo pelo qual o Captcha de texto em imagem foi mantido após a implementação do *hCaptcha* é desconhecido pelo autor.

Figura 2.14: Página de busca de CNPJ da RFB.

A segunda tarefa é a de navegar pelo site, registrando as requisições realizadas pelo navegador para realizar a consulta. Isso envolve abrir o inspetor de elementos do navegador, na aba Rede (ou *Network*, em inglês), anotando as requisições que são realizadas.

Emissão de Comprovante de Inscrição e de Situação Cadastral

Cidadão,

Esta página tem como objetivo permitir a emissão do Comprovante de Inscrição e de Situação Cadastral de Pessoa Jurídica pela Internet em consonância com a [Instrução Normativa RFB nº 1.863, de 27 de dezembro de 2018](#).

Digite o número de CNPJ da empresa e clique em "Consultar".

CNPJ:

7hkhze
 Digite os caracteres acima:

Figura 2.15: Página de busca de CNPJ da RFB, com Captcha de texto.

No exemplo, testamos o CNPJ 13.612.840/0001-57, da Associação Brasileira de Jurimetria. Ao preencher o CNPJ e o rótulo do Captcha, algumas requisições aparecem na aba “Rede”, como mostrado na Figura 2.16. A primeira requisição é do tipo POST⁴, responsável por enviar os dados de CNPJ e do rótulo da imagem para o servidor, que retorna com os dados da empresa.

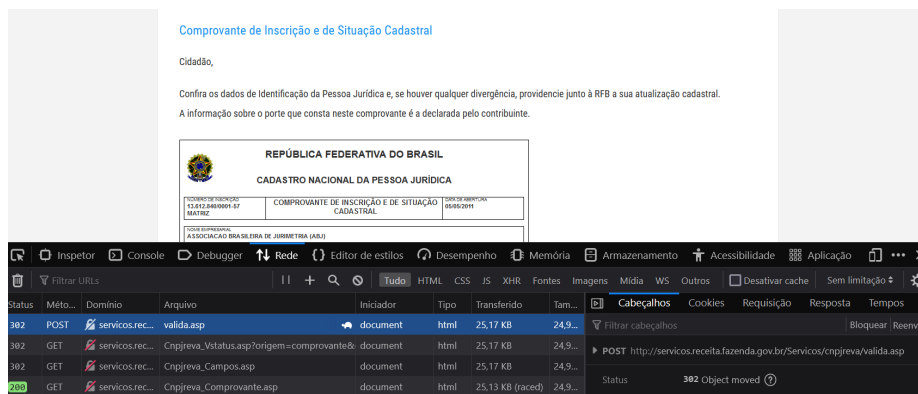


Figura 2.16: Resultado da busca por CNPJ, mostrando a aba Rede.

Investigando a requisição POST, na sub-aba “Requisição”, é possível observar os dados da consulta. Trata-se de um conjunto de parâmetros enviados na forma de lista, com as informações abaixo. Para replicar a requisição na linguagem de programação, estes são os dados enviados.

```
{
  "origem": "comprovante",
  "cnpj": "13.612.840/0001-57",
  "txtTexto_captcha_serpro_gov_br": "7hkhze",
  "search_type": "cnpj"
}
```

⁴ Existem dois tipos principais de requisição HTTP. A requisição GET serve para capturar uma página da internet, enquanto a requisição POST serve para enviar dados para o servidor como, um login e uma senha. A lista completa de requisições está disponível na [documentação da Internet Engineering Task Force \(IETF\)](#).

As etapas de replicar, parsear e validar envolvem baixar e processar os dados na linguagem de programação. No caso do Captcha da RFB, essa tarefa envolve os passos abaixo.

1. Acessar a página inicial de **busca com Captcha sonoro**, através de uma requisição GET.
2. Baixar a imagem do Captcha com uma requisição GET, usando o **link gerado** ao clicar no botão de atualizar o Captcha.
3. Obter o rótulo a partir da imagem do Captcha.
4. Realizar a requisição POST com os dados do exemplo e o rótulo correto da imagem, baixando arquivo resultante em um HTML.
5. Utilizar técnicas de raspagem de arquivos HTML para obter os dados de interesse (como, por exemplo, a razão social da empresa) e validar os resultados, verificando, por exemplo, se o resultado estava completo e disponível.

Todos os passos descritos acima devem ser realizados em uma sessão persistente. Isso significa que a biblioteca utilizada para realizar as requisições deve ser capaz de guardar os *cookies* entre a requisição GET do primeiro passo e a requisição POST do quarto passo, de forma que as requisições sejam interligadas.

O quinto passo da lista acima descreve a parte de *parsear*, que é a responsável pelo nome “raspagem” nessa área do conhecimento. O nome é adequado porque usualmente os arquivos baixados estão em um formato bruto, inadequado para realização de análises. Os dados precisam ser então extraídos – raspados – do arquivo HTML, através de ferramentas de transformação de arquivos como a *libxml2* (WICKHAM; HESTER; OOMS, 2021), técnicas para acessar pedaços do documento, como o XPath (WICKHAM, 2022a) e técnicas de manipulação de textos, como expressões regulares (WICKHAM, 2022b).

A iteração encerra o fluxo da raspagem de dados. Nessa etapa, as operações de replicar, parsear e validar o resultado são reaplicadas iterativamente, com o fim de baixar dados para compor uma base maior. No exemplo da RFB, isso significaria montar uma base de dados a partir de uma lista de CNPJs.

No contexto dos Captchas, o interesse está nos passos de Replicar e Validar. Estes são os passos em que a imagem é baixado e o rótulo é anotado e testado no servidor. Esses são os passos relacionados à classificação manual, e também à implementação do oráculo.

A classificação manual dos Captchas envolve o trabalho de baixar, anotar (manualmente) e verificar se a anotação está correta. Trata-se de um trabalho repetitivo e dispendioso, utilizado para gerar as simulações do trabalho.

O oráculo envolve a possibilidade de checar, de forma automática, se uma predição do rótulo de uma imagem está correta. Por ser um teste de Turing inverso, o Captcha é obrigado a mencionar se uma predição está correta: se a predição foi correta, a página de interesse é acessada; se a predição está incorreta, o site envia uma mensagem de erro. As etapas de replicar, parsear e validar para qualquer site de interesse envolvem os passos a seguir.

1. Acessar a página do site de interesse.

2. Preencher o formulário de pesquisa com a informação a ser consultada. Por exemplo, no site da RFB, a informação é o CNPJ da empresa a ser consultada. Em um site de tribunal, a informação é um número identificador de processo.
3. Baixar a imagem do Captcha da busca.
4. Obter o rótulo da imagem, aplicando um modelo na imagem baixada ou classificado manualmente.
5. Submeter a consulta no site, informando o rótulo.
6. Verificar o resultado. Se acessou a página desejada, o rótulo está correto. Caso contrário, o rótulo está incorreto.

O procedimento descrito pode ser reproduzindo indefinidamente. Isso significa que é possível criar uma base de dados virtualmente infinita de imagens rotuladas, com a informação adicional do rótulo estar correto ou incorreto. Isso foi feito para gerar os dados utilizados na simulação.

O problema do uso de oráculos é que a informação adicional recebida quando o modelo erra é **incompleta**. A única informação nova disponível é que o rótulo testado está incorreto, dentre todos os rótulos possíveis daquela imagem. Como existe uma grande quantidade de rótulos possíveis em um Captcha, muitas vezes na ordem de milhões, a informação que o oráculo fornece é fraca.

Uma possível abordagem para lidar com o segundo problema seria simplesmente descartar os Captchas classificados incorretamente. É possível criar uma base de dados (virtualmente infinita) somente com os rótulos corretos e ajustar um novo modelo. Essa abordagem, no entanto, tem sérios problemas, já que considera somente os casos em que o classificador já funciona bem. O trabalho realizado na tese incorpora a informação fornecida pelo oráculo quando o modelo erra.

Outra oportunidade que o oráculo oferece em parte dos casos é a possibilidade de testar mais de uma predição. Sites com essa característica permitem que a pessoa ou robô teste mais de uma predição caso o Captcha tenha fracassado. Como é possível observar na Tabela 2.1, dos 10 Captchas trabalhados, 7 permitem a realização desses testes.

Neste momento, cabe uma observação sobre oráculos e força bruta. O poder de testar vários rótulos para o mesmo Captcha implica na possibilidade teórica de resolver um Captcha por força bruta. Bastaria testar todos os rótulos possíveis para acessar a página de interesse. Na prática, no entanto, essa estratégia não funciona, já que a quantidade de rótulos possíveis é muito grande para testar no site, seja por demorar muito tempo ou pelo site forçar a troca do desafio após a passagem de determinado tempo ou quantidade de tentativas.

Voltando ao ciclo da raspagem, ao longo do procedimento de baixar imagens de Captchas e aplicar o oráculo, pelo menos duas funções devem ser criadas: **acesso** e **teste**. A operação de acesso é responsável por preencher o formulário de busca e baixar o Captcha (passos 1 a 3 da lista acima). A operação de teste é responsável por submeter um rótulo do Captcha e verificar retornar se o rótulo está correto ou incorreto (passos 4 a 6 da lista acima). Em alguns casos, as funções de acesso e teste precisam compartilhar parâmetros que contêm a sessão do usuário, para garantir que o teste envolva o mesmo Captcha da etapa de acesso.

Os Captchas foram anotados manualmente com o procedimento chamado de semi-automático, definido a seguir. No pacote `{captchaDownload}` (ver Apêndice A.2), foram desenvolvidas ferramentas para baixar e organizar cada Captcha, utilizando o oráculo para garantir que as imagens eram corretamente classificadas.

Cada Captcha teve as primeiras 100 observações classificadas manualmente. Isso foi feito a partir do próprio RStudio, utilizando a ferramenta de classificação manual do pacote `{captcha}`.

A partir das classificações iniciais, um modelo foi ajustado com a quantidade de observações disponível. Esse passo também foi feito a partir do pacote `{captcha}`, que cria um projeto de classificação para um Captcha específico.

O modelo, então, foi utilizado como uma ferramenta para otimizar a classificação manual, funcionando da seguinte forma. Primeiro, o modelo tenta realizar a predição automaticamente e o oráculo avisa se a predição está correta ou não. Se estiver incorreto e o site aceitar várias tentativas, o modelo tenta novamente, mas com uma segunda alternativa de predição. Caso o site não aceite várias tentativas ou o modelo não consiga acertar o Captcha em N tentativas (abitrado como dez), a imagem do Captcha aparece para classificação manual.

Com o procedimento destacado acima, é criada uma nova base de dados, que por sua vez é utilizada para ajustar um novo modelo. O modelo, atualizado, é utilizado para classificar novos Captchas, e assim por diante, até que o modelo ajustado alcance uma acurácia razoável, que foi arbitrada em 80%. Com isso o procedimento de anotação é finalizado.

O único problema do procedimento de classificação diz respeito aos Captchas que não aceitam várias tentativas. Nesses casos, não é possível verificar com certeza absoluta que um caso classificado manualmente (após a tentativa do modelo) foi classificado corretamente, já que a classificação manual seria a segunda tentativa. No entanto, esse problema aparece somente em três Captchas (`cadesp`, `jucesp` e `trf5`). A classificação manual dos 100 primeiros Captchas, no entanto, mostrou que pelo menos 95% dos Captchas foram classificados corretamente quando classificados manualmente. A proporção máxima de 5% de erro é negligenciável considerando que a maior parte das bases de dados foi construída com verificação do oráculo.

Em alguns casos, os rótulos dos Captchas podem ser obtidos sem intervenção humana, utilizando técnicas de raspagem de dados e processamento de sinais. Um exemplo é o Captcha do SEI, que mostra informações suficientes para resolver o Captcha na própria URL que gera a imagem. Outro exemplo é o TJMG, que libera, além da imagem, um áudio contendo o mesmo rótulo da imagem, sem a adição de ruídos. Como o áudio não tem ruídos, basta ler o áudio, separar os áudios de cada caractere e calcular uma estatística simples (como a soma das amplitudes, ao quadrado). Essa estatística é utilizada para associar um pedaço de áudio a um caractere.

A Tabela 2.2 caracteriza os Captchas anotados. Todos os Captchas possuem comprimento entre 4 e 6 dígitos e, com exceção do SEI, não são sensíveis a maiúsculas e minúsculas.

Tabela 2.2: *Lista de captchas analisados e suas características.*

Captcha	Vários chutes	Caracteres	Comprimento	Colorido	# Rótulos anotados
trf5	Não	0-9	6	não	1000
tjmg	Sim	0-9	5	sim	1000
trt	Sim	a-z0-9	6	não	1500
esaj	Sim	a-z	5	sim	3000
jucesp	Não	a-z0-9	5	não	4000
tjpe	Sim	a-z0-9	5	não	4000
tjrs	Sim	0-9	4	sim	2000
cadesp	Não	a-z	4	sim	3000
sei	Sim	a-zA-Z0-9	4	sim	10000
rfb	Sim	a-z0-9	6	não	4000

As bases de dados com imagens anotadas foram disponibilizadas na aba de lançamentos (*releases*) do [repositório principal do projeto de pesquisa](#). As bases com imagens e modelos ajustados estão disponíveis para quem tiver interesse em fazer novas pesquisas e utilizar os resultados em suas aplicações, sem restrições de uso.

2.4 Simulações

Para verificar o poder do uso do oráculo para o aprendizado do modelo, uma série de simulações foi desenvolvida. As simulações foram organizadas em três passos: modelo inicial, dados e modelo final. Os passos foram descritos em maior detalhe a seguir.

2.4.1 Primeiro passo: modelo inicial

A simulação do modelo inicial teve como objetivo obter modelos preditivos de Captchas com acurácias distintas. O modelo inicial seria usado, então, para baixar dados diretamente do site usando o oráculo e, por fim, ajustar um modelo final com os novos dados provenientes do oráculo.

Os modelos iniciais foram construídos em dois passos. O primeiro foi montar a base de dados completa, suficiente para ajustar um modelo com alta acurácia, que arbitrados em 80%, como descrito anteriormente. Depois, montou-se 10 amostras de dados com subconjuntos das bases completas, cada uma contendo 10%, 20%, e assim por diante, até a base completa. Por exemplo: no Captcha da Jucesp, construiu-se um modelo com acurácia maior que 80% com 4000 Captchas. A partir disso, foi feita uma partição dos dados com

400 imagens (10% do total), 800 imagens (20% do total) e assim por diante, até o modelo com 4000 Captchas.

Para cada tamanho de amostra A , aplicou-se uma bateria de 27 modelos. Isso foi feito porque para diferentes quantidades de amostra, a configuração dos hiperparâmetros que resulta no melhor modelo pode ser diferente. Os modelos seguiram uma grade de hiperparâmetros considerando três informações:

- A quantidade de unidades computacionais na primeira camada densa após as camadas convolucionais, com os valores considerados: 100, 200 e 300.
- O valor do *dropout* aplicado às camadas densas, com os valores considerados: 10%, 30% e 50%.
- O fator de decaimento na taxa de aprendizado a cada época, com os valores considerados: 1%, 2% e 3%.

Combinando os três valores dos três hiperparâmetros, tem-se um total de $27 = 3^3$ hiperparâmetros. Com isso, foi possível identificar, para cada tamanho de amostra A , o classificador C_A com a melhor acurácia dentre os modelos ajustados.

No final do primeiro passo, portanto, considera-se apenas o melhor modelo para cada tamanho de amostra, dentre os 27 ajustados. É claro que os modelos encontrados por essa técnica não são, necessariamente, os melhores modelos possíveis. No entanto, como a técnica é a mesma para todos os Captchas, é possível fazer comparações através de uma metodologia mais transparente.

2.4.2 Segundo passo: dados

O segundo passo teve como objetivo construir as bases de dados utilizando o oráculo. Primeiro, foi necessário decidir quais modelos, dentre os 10 ajustados para cada Captcha, seriam utilizados para construir novas bases. Não faria sentido, por exemplo, considerar um modelo com acurácia de 0%, já que ele não produziria nenhuma observação comparado com um modelo que chuta aleatoriamente. Também não faria sentido considerar um classificador com acurácia de 100%, já que nesse caso não há o que testar com a técnica do oráculo.

Decidiu-se que seria necessário considerar somente os modelos que resultaram em acurácias maiores de 1% e menores de 50%. O valor máximo foi decidido após realizar alguns testes empíricos e verificar, informalmente, que a técnica do oráculo realmente resultava em ganhos expressivos, mesmo com modelos de baixa acurácia. Concluiu-se então que não seria necessário testar a eficácia da técnica para classificadores com alta acurácia. Já o valor mínimo foi decidido de forma arbitrária, retirando-se os classificadores com acurácia muito baixa.

A segunda decisão a ser tomada para construção dos dados foi a quantidade de imagens que seria baixada para cada Captcha. Como são Captchas de diferentes dificuldades, a quantidade de dados seria diferente. Optou-se por baixar a quantidade de dados de forma a montar uma base de treino que contém a quantidade de observações necessária para obter o melhor modelo daquele Captcha. Por exemplo, no TJRS, um modelo com acurácia próxima de 100% foi identificado com 2000 observações. O melhor modelo com 300

imagens (240 para treino, 60 para teste) resultou em uma acurácia de 35%. Foram, então, baixadas 1760 observações para compor o total de 2000 na base de treino. As imagens de teste do modelo inicial poderiam até ser utilizadas, mas optamos por descartar para garantir que o modelo não ficasse sobreajustado para a primeira base.

O motivo de baixar a mesma quantidade de observações que o melhor modelo inicial foi feita por dois motivos. O primeiro é que existem evidências de que é possível construir um bom modelo com essa quantidade de imagens, ainda que em um caso as informações são completas e, no outro, incompletas. O segundo é que isso permite a comparação do resultado do modelo completamente anotado contra o modelo que é parcialmente anotado e com anotações incompletas provenientes do oráculo.

A terceira e última decisão tomada para baixar os dados foi a quantidade de chutes que o modelo poderia fazer, nos casos em que isso é permitido pelo site. Optou-se, de forma arbitrária, por três valores: 1, que é equivalente a um site que não permite múltiplos chutes, 5 chutes e 10 chutes.

Portanto, o procedimento de coleta dos dados foi feito, para cada Captcha, da seguinte forma:

1. Listou-se todos os melhores modelos ajustados para cada tamanho de amostra.
2. Filtrou-se os modelos para os que apresentavam acurácia de 5% até 50%
3. Definiu-se o tamanho da base a ser obtida, com base no tamanho da base de treino utilizada no modelo e a quantidade total que se objetivou obter.
4. Para cada quantidade de tentativas disponível (1, 5 e 10), baixou-se as imagens, anotando com o valor “1” se o rótulo de alguma das tentativas estivesse correto e com o valor “0” caso contrário.
5. Nos casos com erros, armazenou-se um arquivo de log para cada Captcha com o histórico de tentativas incorretas, que é a informação mais importante a ser passada para o modelo final.

No final, obteve-se bases de dados de treino para todos os Captchas analisados, com quantidades de imagens variadas de acordo com os parâmetros definidos anteriormente, variando também pela quantidade de tentativas. A quantidade total de bases de dados geradas foi 65.

Além das bases de treino, foi construída uma base de teste para cada Captcha. As bases de teste foram construídas completamente do zero, sem utilizar informações de bases anteriores. Para construir as bases, utilizou-se a mesma técnica semi-automática definida anteriormente, usando o melhor modelo disponível para classificar a maioria das imagens e classificando manualmente em caso de falha. Em alguns casos, como TJMG e TJRS, a classificação humana quase não foi necessária, pois os classificadores obtidos apresentaram acurácia próxima de 100%.

Como o único objetivo da base de teste foi o de estimar a acurácia dos modelos finais, a quantidade de observações poderia ser arbitrada. O tamanho das bases de teste foi, então, arbitrado em 1000 imagens para cada Captcha.

2.4.3 Terceiro passo: modelo final

O modelo final foi ajustado para cada uma das 65 bases de treino disponíveis após a realização dos passos 1 e 2. Nesse caso, utilizou-se o modelo proposto na Seção 2.2. Caso a imagem tenha sido corretamente classificada, a função de perda é calculada normalmente. Caso ela tenha sido classificada incorretamente, considera-se a probabilidade de não observar nenhum dos chutes.

Além de modificar a forma de calcular a função de perda do modelo, foi necessário realizar uma nova busca de hiperparâmetros. Optou-se por utilizar os mesmos hiperparâmetros dos modelos iniciais para manter a consistência. O único detalhe nesse ponto é que, como os parâmetros de partida são os do modelo inicial, optou-se por não modificar a quantidade de unidades na camada densa, variando somente os valores de *dropout* e de decaimento na taxa de aprendizado. Portanto, ajustou-se 9 e não 27 modelos para cada base de dados.

No final, assim como no primeiro passo, os classificadores com melhor acurácia foram selecionados para cada modelo. Obteve-se, então, com 65 modelos no final para comparar com os modelos iniciais e estimar a efetividade do oráculo. As comparações foram feitas através de gráficos de barras, explorando o efeito do uso do oráculo para diferentes Captchas, diferentes modelos iniciais e diferentes quantidades de chutes, além de um gráfico de dispersão para relacionar as acurácias iniciais e finais.

Além do terceiro passo, outros experimentos foram realizados para verificar se, ao aplicar a técnica do oráculo iterativamente, os resultados continuariam melhorando. Ou seja, é possível considerar os modelos obtidos no passo 3 como os modelos iniciais do passo 1, aplicar novamente o passo 2 (baixar dados) e o passo 3 (rodar modelo com os novos dados). Isso foi feito para apenas um conjunto selecionado de Captchas para verificar essa possibilidade, não fazendo parte das simulações principais do estudo.

As bases de dados das simulações também foram disponibilizadas na aba de lançamentos (*releases*) do **repositório principal do projeto de pesquisa**. As bases podem ser utilizadas para aumentar as bases de treino e para testar outras arquiteturas de redes neurais ao tema dos Captchas com uso de aprendizado fracamente supervisionado.

Capítulo 3

Resultados

*I'm not a robot, I'm a human. But I'm pretty
sure the robot is better at this than I am.*
— ChatGPT

Neste capítulo, discute-se os resultados da metodologia WAWL. Para isso, são apresentadas resultados empíricos que demonstram que os a proposta possui bons resultados.

Os resultados foram obtidos a partir das simulações com diversos Captchas. Foram realizadas 65 simulações no total, variando no tipo de Captcha, a acurácia do modelo inicial e a quantidade de tentativas no oráculo.

Para realizar os cálculos, montou-se uma base de dados com os resultados das simulações. A base está disponível publicamente no [repositório da tese](#) e contém informações do Captcha ajustado (`captcha`), da quantidade de observações do modelo inicial (`n`), da quantidade de tentativas do oráculo (`ntry`), da etapa de simulação (`fase`, inicial ou WAWL), do caminho do modelo ajustado (`model`) e da acurácia obtida (`acc`).

Os resultados gerais mostram um ganho de 333% na acurácia após a aplicação da metodologia WAWL. Ou seja, em média, a acurácia do modelo no terceiro passo da simulação (ver Seção 2.4.3) foi de mais de **três vezes** a acurácia do modelo inicial. Em termos absolutos (diferença entre as acurácias), o ganho foi de 33%, ou seja, após o terceiro passo modelos ganharam, em média, 33% de acurácia.

As Figuras 3.1 e 3.2 mostram os ganhos relativos e absolutos, separando os resultados gerais por quantidade de tentativas. Cada ponto é o resultado de uma simulação e o ponto em destaque é o valor médio, acompanhado de intervalo $m \mp 2 * s / \sqrt{(n)}$, com m sendo a média, s o desvio padrão e n a quantidade de dados. A linha pontilhada indica se a acurácia aumentou ou diminuiu após a aplicação da técnica.

Na Figura 3.1, é possível notar que os ganhos em acurácia apresentam alta variabilidade, mas que apresentam uma tendência positiva com relação ao número de tentativas. O ganho entre aplicar 5 e 10 tentativas é menos expressivo do que o ganho entre aplicar 1 e 5 tentativas, indicando que a oportunidade oferecida por sites que aceitam vários chutes é relevante e que não há necessidade de realizar tantos chutes para aproveitar essa oportu-

tunidade.

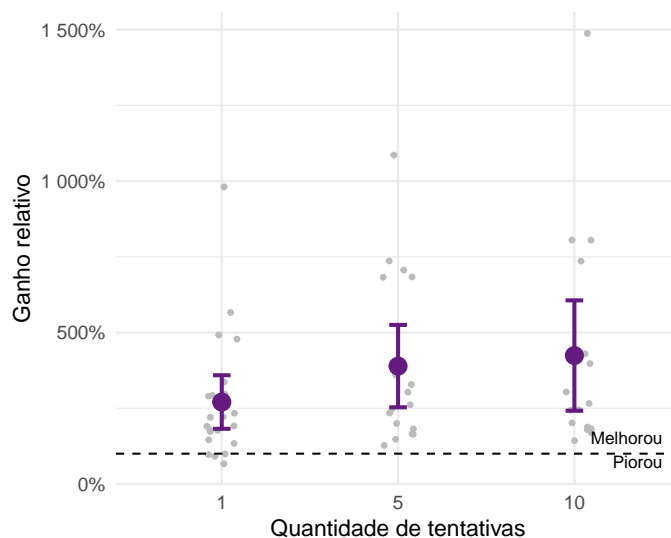


Figura 3.1: Ganho percentual ao utilizar a técnica do oráculo, dividido por quantidade de tentativas.

A Figura 3.2, com os ganhos absolutos, mostra a mesma informação mas em quantidades mais fáceis de interpretar. O ganho médio absoluto em sites que permitem mais de um chute ficou em torno de 40%, enquanto que o ganho com apenas um chute ficou um pouco acima de 25%. Importante notar também que o uso do oráculo só piorou a acurácia do modelo em casos que com apenas um chute, mostrando que a técnica é efetiva de forma consistente.

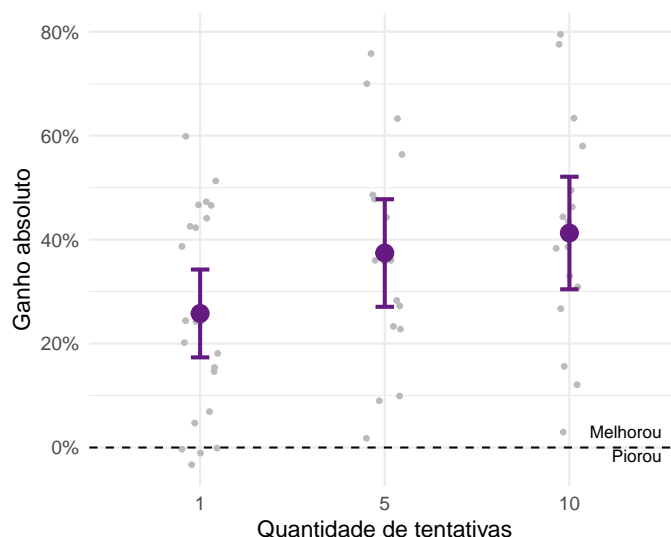


Figura 3.2: Ganhos absolutos ao utilizar a técnica do oráculo, dividido por quantidade de tentativas.

As Figuras 3.3 e 3.4 apresentam os resultados gerais separando por acurácia inicial do modelo. A estrutura do gráfico é similar às visualizações anteriores, que separaram os resultados por quantidade de tentativas. As categorias escolhidas foram: até 10%, mais

de 10% até 35% e mais de 35% de acurácia no modelo inicial. A escolha dos intervalos se deram pela quantidade de observações em cada categoria.

A Figura 3.3 mostra os ganhos relativos. É possível notar uma tendência de queda no ganho de acurácia com uso do oráculo conforme aumenta a acurácia do modelo inicial. Esse resultado é esperado, pois, como a acurácia é um número entre zero e um, um modelo que já possui alta acurácia não tem a possibilidade de aumentar muito de forma absoluta.

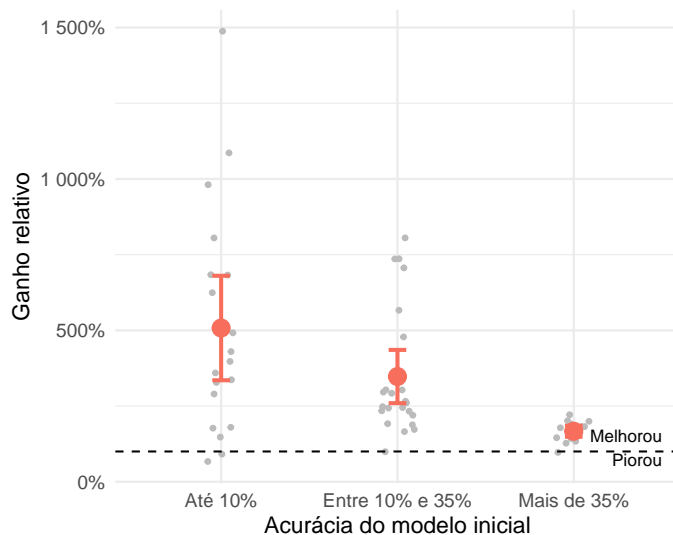


Figura 3.3: Ganho percentual ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.

A Figura 3.4 mostra os ganhos absolutos. O gráfico apresenta o mesmo problema que o anterior, já que o ganho máximo depende da acurácia inicial do modelo. Ainda assim, é possível notar que, em termos absolutos, modelos com acurácia inicial entre 10% e 35% apresentaram um ganho maior que modelos com acurácia inicial de até 10%.

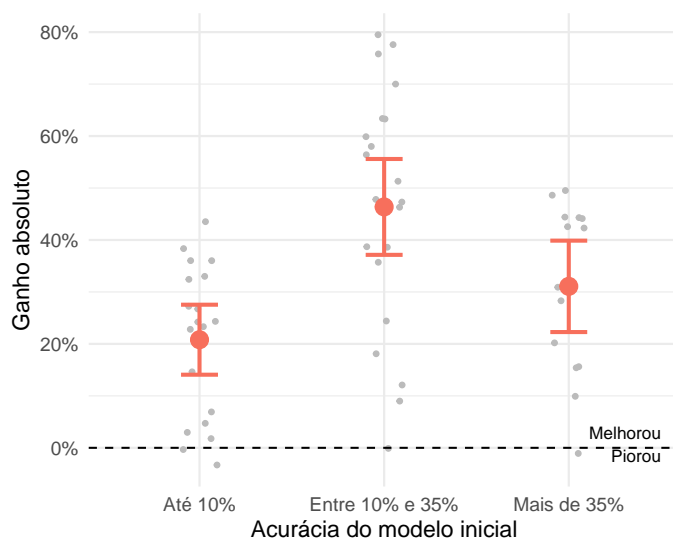


Figura 3.4: Ganho absoluto ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.

Para lidar com o fato da acurácia ser um número limitado, fizemos o mesmo gráficos de antes, mas ajustado pelo máximo possível que a técnica do oráculo poderia proporcionar. O ganho absoluto ajustado de uma simulação é dado por

$$\text{ganho} = \frac{\text{oráculo} - \text{inicial}}{1 - \text{inicial}}.$$

A Figura 3.5 mostra os ganhos ajustados. Pelo gráfico, é possível notar que existe um ganho expressivo da técnica WAWL do oráculo para modelos iniciais com mais de 10% de acurácia com relação a modelos iniciais com até 10% de acurácia. Ou seja, quando o modelo inicial é fraco, o ganho ao usar a técnica é menor. É importante notar, no entanto, que as simulações mostram a aplicação da técnica apenas uma vez – é possível baixar mais dados e atualizar o modelo indefinidamente. O menor efeito da técnica para modelos iniciais fracos não significa, portanto, que a técnica não funciona para modelos iniciais fracos; pelo contrário: ela ajuda o modelo a sair do estado inicial e o leva para um estado com acurácia maior, de onde seria possível aplicar a técnica novamente para obter resultados mais expressivos.

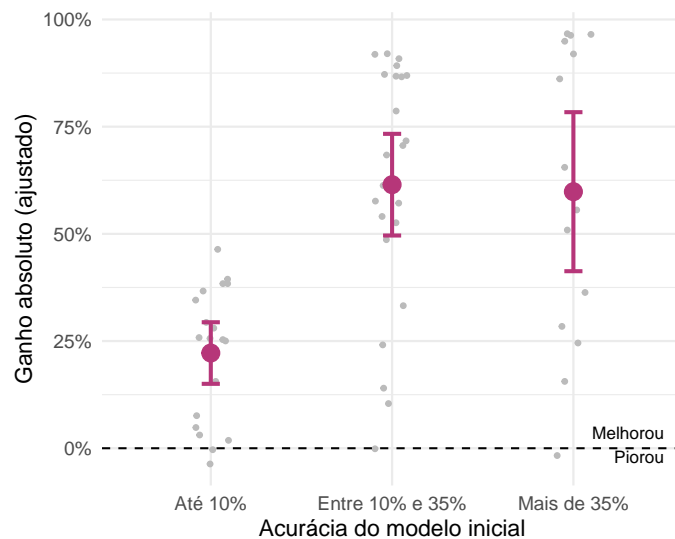


Figura 3.5: Ganho absoluto ao utilizar a técnica do oráculo, dividido por acurácia do modelo inicial.

Na Figura 3.6, são apresentados os resultados separando por Captcha. Cada linha é uma combinação de Captcha, quantidade de tentativas e acurácia modelo inicial, classificados nas três categorias mostradas anteriormente. As linhas pontilhadas indicam modelos ajustados com mais de uma tentativa, enquanto as linhas contínuas mostram modelos ajustados com apenas uma tentativa. A primeira extremidade de cada linha, do lado esquerdo, indica a acurácia do modelo inicial e a segunda extremidade, do lado direito, a acurácia do modelo usando o método WAWL.

Pelo gráfico, é possível identificar duas informações relevantes. Como já verificado anteriormente, os modelos ajustados com mais de uma tentativa apresentam maiores ganhos do que os modelos ajustados com apenas uma tentativa. Verifica-se também que modelos com acurácia inicial de até 10% só apresentam ganhos menores que os modelos com acurácia inicial maior que 10% nos casos em que apenas uma tentativa é definida. Ou seja, existe

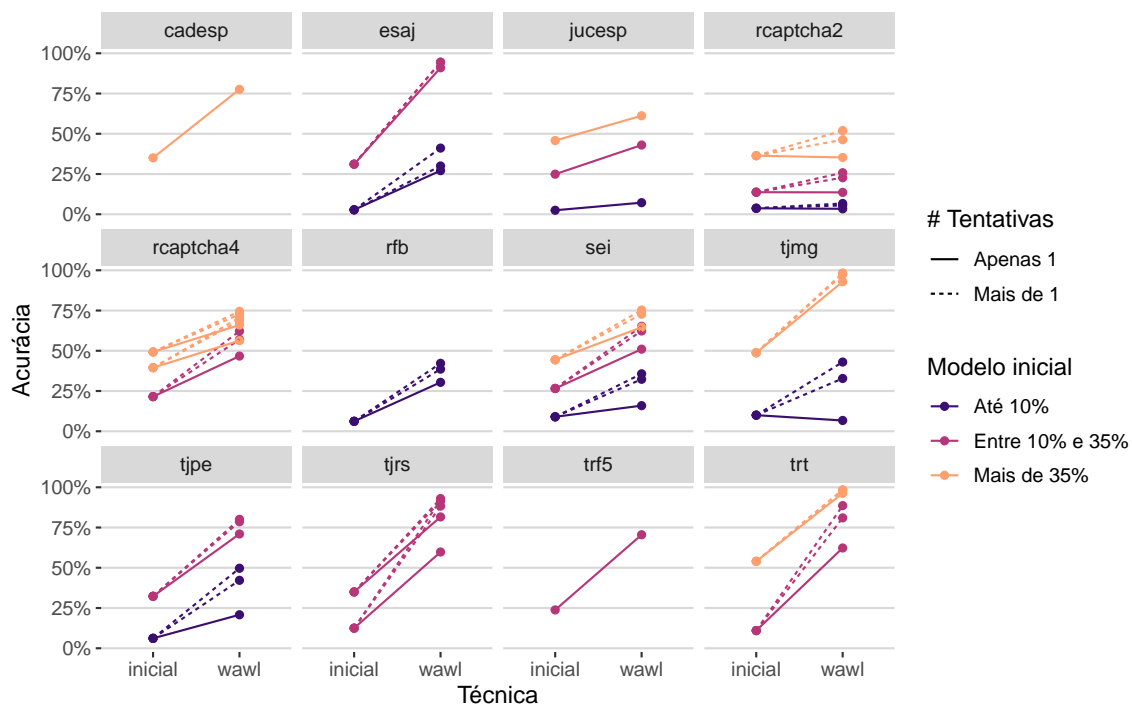


Figura 3.6: Resultados da simulação por captcha, quantidade de tentativas e modelo inicial.

interação entre a quantidade de tentativas e a acurácia do modelo inicial ao avaliar o impacto nos ganhos empíricos do método WAWL.

Pelos resultados das simulações, é possível concluir que o método WAWL foi bem sucedido. Primeiro, o método apresenta resultados expressivos e de forma consistente, sem realizar novas anotações manuais. Além disso, a técnica aproveita a oportunidade oferecida pelos sites de obter o *feedback* oráculo múltiplas vezes na mesma imagem. Finalmente, o método apresenta, em média, resultados positivos mesmo para modelos iniciais muito fracos (com acurácia de até 10%), indicando que sua aplicação é possível para qualquer modelo inicial, o que é bastante factível de atingir com bases pequenas ou com modelos generalistas para resolver Captchas.

Um possível problema em aplicar o WAWL é que a técnica poderia introduzir viés de seleção no modelo, impedindo-o de ser aprimorado indefinidamente. Mesmo que os resultados teóricos dêem uma boa base para acreditar que isso não seja verdade, foi feito um experimento adicional, com apenas um dos Captchas, para verificar se a aplicação da técnica múltiplas vezes apresenta bons resultados.

O Captcha escolhido para a simulação foi o *trf5*, por ser um Captcha que não aceita múltiplos chutes, em uma tentativa de obter um pior caso. Para esse Captcha, o melhor modelo obtido com a técnica do oráculo foi considerado como modelo inicial, sendo usado para baixar novos dados do site do Tribunal. Os novos dados foram adicionados à base de treino, ajustando-se um novo modelo.

A Figura 3.7 mostra os resultados da aplicação iterada. A utilização da técnica não só funcionou como levou o modelo a uma acurácia de 100%.

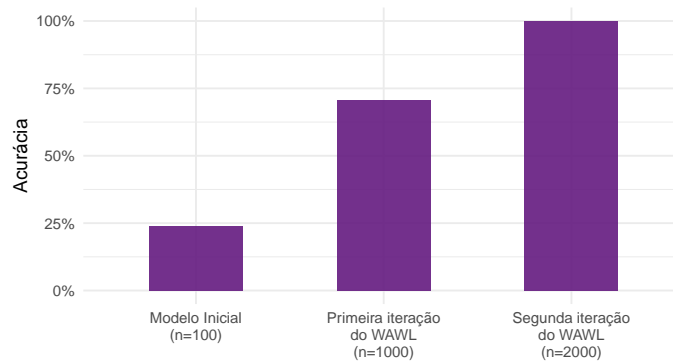


Figura 3.7: Resultados da aplicação iterada da técnica.

O resultado sugere que a técnica pode ser utilizada em várias iterações para auxiliar no aprendizado do modelo. Ela sugere, ainda, que uma técnica de aprendizado ativo com *feedback* automático do oráculo pode dar bons resultados, já que a forma de obter os dados não introduz viés de seleção no ajuste do modelo.

3.1 Discussão

Os resultados apresentados nas seções anteriores mostram que o método WAWL possui boas propriedades matemáticas e bons resultados empíricos. Nesta seção, os resultados foram confrontados com as hipóteses de pesquisa definidos na Seção 1.6 de forma crítica.

A primeira hipótese de pesquisa diz respeito à pertinência de utilizar do aprendizado fracamente supervisionado como forma de ajustar modelos para resolver Captchas. A hipótese foi verificada, tanto do ponto de vista teórico quanto do ponto de vista prático.

Na parte teórica, várias pesquisas já apontam que o aprendizado com rótulos parciais ou rótulos complementares têm boas propriedades. Por isso, já seria esperado que uma nova função de perda, desde que pensada com cuidado, traria resultados positivos. O resultado foi verificado a partir da obtenção das propriedades da função de perda proposta, mostrando que a função de risco baseada nessa perda se aproxima da função de risco de um problema completamente supervisionado.

Na parte prática, até o momento não existiam evidências de que a utilização de rótulos parciais ou rótulos complementares teriam bons resultados empíricos em Captchas. Isso foi verificado em todos os 12 Captchas estudados, sendo 10 obtidos do mundo real. Em todos os casos, a função de perda proposta funcionou bem e trouxe ganhos significativos na acurácia do modelo, tanto em termos relativos quanto absolutos. Isso demonstra que a escolha do método se alia bem ao problema que deu origem à pesquisa, que são os Captchas.

Sobre a parte de aplicação iterada do WAWL, cabe um comentário. O resultado encontrado, com 100% de acurácia, pode sugerir que o método WAWL sempre chegará em um

resultado de 100% para qualquer Captcha que surgir. No entanto, pode ser que exista uma limitação na capacidade do modelo, que é habilidade do modelo para se ajustar aos dados a partir dos parâmetros. Pode ser que a arquitetura de rede neural escolhida para resolver o Captcha não seja capaz de chegar a um modelo com 100% de acurácia, independente da quantidade de imagens observadas. É importante olhar o resultado apresentado de forma crítica e compreender que os resultados finais podem ser limitados, já que a arquitetura da rede neural não é parte do método WAWL.

A segunda hipótese de pesquisa é a possibilidade de aliar a área de raspagem de dados com a área de modelagem estatística. Essa parte está bem mais relacionada com a parte prática da pesquisa, já que os conceitos de raspagem de dados não são utilizados para estudar a propriedade dos modelos. A hipótese também foi verificada, já que o método WAWL, que utiliza técnicas de raspagem de dados, apresentou bons resultados empíricos.

Neste momento, cabe um comentário sobre o ineditismo da ponte entre raspagem de dados e estatística. É verdade que existem muitas pesquisas que são possibilitadas por conta dos dados obtidos via raspagem de dados: as pesquisas da ABJ, mencionadas na Seção 1.1 são alguns exemplos. Também existem soluções que utilizam dados provenientes de raspagem de dados para construção de modelos: por exemplo, o DALL-E-2, que é parte de uma base de dados construída utilizando imagens baixadas da internet (MURRAY; MARCHESOTTI; PERRONNIN, 2012; RAMESH et al., [s.d.]). No entanto, até o momento da realização da pesquisa, não foi encontrado nenhum trabalho que utiliza a raspagem de dados como parte do processo de aprendizado estatístico. O método WAWL conecta as áreas de forma intrínseca, podendo ser entendida como uma nova variação de aumento de dados aplicada a redes neurais convolucionais.

O fato da raspagem de dados ser relevante para o ajuste de um modelo estatístico pode levar a algumas discussões sobre o ensino da estatística. Primeiro, é importante mencionar que:

1. Raspagem de dados não faz parte dos currículos de Bacharelado em Estatística das principais universidades do país¹. Logo, pode-se argumentar que raspagem de dados não é uma área de interesse da estatística.
2. Raspagem de dados não é uma área de conhecimento bem definida, como álgebra ou análise de sobrevivência. A área é melhor desenvolvida através de aplicações práticas e utilização de ferramentas (como R ou python) do que através de aulas teóricas.

Os resultados levam, então, a um problema de balanceamento entre pertinência e oportunidade. De um lado, a área de raspagem não se encaixa muito bem no currículo de estatística. Por outro lado, a área expande as possibilidades de atuação de uma profissional da estatística.

Para aliar a pertinência e a oportunidade, uma opção seria oferecer disciplinas operativas de raspagem de dados nos cursos de estatística. Para aumentar a quantidade de potenciais ministrantes, a disciplina poderia ser oferecida em parceria com outros cursos, como ciência da computação, matemática aplicada e engenharias. Dessa forma, as pessoas

¹ Universidades consultadas: USP Butantã (IME), UFSCar, UNESP, Unicamp, USP São Carlos (ICMC), UFBA, UFPR, UFRGS, UFPE, UFAM, UFRN, UFF, ENCE, UFRJ, UFMG, UnB e UFG.

interessadas teriam a oportunidade de aprender um pouco sobre as técnicas principais, conectando a raspagem de dados com as áreas de conhecimento específicas, como é o caso do Captcha, que alia raspagem de dados com estatística e inteligência artificial.

No final, as duas hipóteses de pesquisa foram verificadas. No processo de obtenção dos resultados, no entanto, um terceiro avanço muito importante foi realizado na parte computacional. O pacote `{captcha}` e os pacotes auxiliares `{captchaDownload}` e `{captchaOracle}` são frutos desse trabalho. Pela primeira vez, foi construída uma ferramenta aberta contendo um fluxo de trabalho adaptado para trabalhar com Captchas. Além disso, trata-se de uma das primeiras aplicações completas dos pacotes `{torch}` e `{luz}`, que têm potencial de revolucionar a forma em que os modelos estatísticos são desenvolvidos por pessoas que fazem pesquisa em estatística. Os pacotes foram descritos em detalhes no `sec-pacote`.

Por fim, todos os modelos construídos foram disponibilizados no pacote `{captcha}`. Os códigos, dados e resultados das simulações estão disponíveis no pacote `{captchaOracle}`. Os dados utilizados para elaboração da tese estão disponíveis no [repositório da tese no GitHub](#). Dessa forma, a pesquisa pode ser considerada como reproduzível, podendo servir como base para pesquisas futuras.

Capítulo 4

Conclusões

Even robots need a break from the daily grind of solving captchas.

— ChatGPT

Este trabalho de doutorado teve como objeto de estudo os Captchas, que são desafios utilizados para identificar se o acesso à uma página na internet é realizada por uma pessoa ou uma máquina. A pesquisa apresentou um breve histórico dos Captchas, os problemas de sua utilização em serviços públicos e as abordagens existentes para resolução automática de Captchas. Em seguida, apresentou-se como uma nova abordagem, o WAWL, poderia aliar técnicas de raspagem de dados e aprendizado estatístico com rótulos parciais para obter modelos poderosos de resolução de Captchas sem a necessidade de anotar vários casos manualmente. Por fim, foram apresentadas as propriedades do modelo proposto e os resultados empíricos através de uma série de simulações.

Os resultados da pesquisa foram positivos. Na parte teórica, várias pesquisas já apontavam que o aprendizado com rótulos parciais ou rótulos complementares possuíam boas propriedades. O resultado foi verificado a estudando-se as propriedades da função de perda proposta. Na parte prática, o trabalho mostrou que a técnica apresenta bons resultados, aumentando a acurácia dos modelos iniciais em mais de 3 vezes, sem a necessidade de anotar novos dados. Além disso, foram encontradas evidências de que o método pode ser aplicado iterativamente, resultado em modelos com poder preditivo ainda maior.

As contribuições do estudo podem ser organizadas em três tipos: contribuições para a sociedade em geral, contribuições para a pesquisa acadêmica e contribuições para a comunidade de programação. Os próximos parágrafos descrevem esses avanços.

A contribuição para a sociedade em geral está na quebra de um mecanismo de incentivo nefasto, gerado pela utilização de Captchas em sites serviços públicos. Como comentado na introdução, o uso de Captchas gera um incentivo para que pessoas e empresas que fazem raspagem de dados utilizem serviços que se aproveitam de mão de obra humana com baixíssima remuneração. Ao disponibilizar os modelos para resolução de Captchas publicamente e uma ferramenta que facilita seu uso, as pessoas e empresas interessadas podem resolver Captchas gratuitamente, afastando a necessidade de utilizar esses serviços. Dessa forma, espera-se que o trabalho possa ter um impacto positivo, ainda que pequeno,

na qualidade das relações de trabalho na sociedade.

A contribuição para a pesquisa acadêmica pode ser separada em duas partes. A primeira é que a função de perda proposta apresenta boas propriedades matemáticas, mostrando que pode ser um bom ponto de partida para quem deseja trabalhar com aprendizado com rótulos parcialmente anotados. A segunda é relacionada ao uso da raspagem de dados como passo intermediário na construção de modelos estatísticos, que pode ser a base para o desenvolvimento de um novo campo de pesquisa. Espera-se que os resultados obtidos sirvam como incentivo para que as técnicas de raspagem de dados sejam ensinadas como disciplinas optativas em cursos de estatística e similares.

A contribuição para a comunidade de programadoras e programadores está no pacote `{captcha}`. O pacote é uma caixa de ferramentas completa para quem tiver interesse em resolver Captchas, além de ser uma das primeiras aplicações que utilizam os pacotes `{torch}` e `{luz}` como motor computacional. O pacote possui uma interface que permite o compartilhamento de códigos, bases de dados e modelos de forma distribuída, possibilitando a criação de soluções que vão muito além do próprio pacote.

Com isso, pode-se concluir que os quatro objetivos descritos na Seção 1.4 foram atendidos. O modelo proposto foi descrito e suas propriedades estudadas. Um repositório de dados completo foi construído e disponibilizado no repositório do pacote `{captcha}`, contendo dados e modelos ajustados. O método foi utilizado e testado para diferentes Captchas e diferentes situações, com sucesso. Finalmente, foi disponibilizado um pacote computacional aberto, possibilitando a resolução de novos Captchas que aparecerem em serviços públicos.

É evidente, no entanto, que o trabalho apresenta algumas limitações. Na parte teórica, os resultados matemáticos podem ser desenvolvidos com maior detalhamento. Especificamente, outras propostas de função de perda podem ser apresentadas, bem como os testes empíricos com essas funções de perda. A escolha de apenas uma alternativa foi feita por conta do foco em resolver o problema de pesquisa (os Captchas) em detrimento da discussão detalhada do aprendizado com rótulos parciais, bem como da necessidade de fazer muito mais simulações se outras funções de perda fossem propostas.

Outra limitação importante do estudo está na aplicação iterada. A pesquisa apresentou essa parte como um resultado adicional, mas os limites da aplicação iterada ainda não foram estudados de forma completa. Essa limitação pode ser entendida também como um próximo passo, que seria uma solução de *online learning*.

Uma extensão possível desta pesquisa é a criação de um modelo que aprende diretamente da *web*, sem a separação de passos descrita pelo WAWL. A técnica consiste em inserir as funções de acesso e teste do Captcha como método de obtenção de amostras do *dataset* do Captcha (mais detalhes na Seção A.1.3). Dessa forma, o modelo poderia obter novos dados em cada *minibatch*, indefinidamente. A vantagem dessa abordagem é que ela seria menos burocrática que o WAWL, que precisa de uma pessoa para escrever as aplicações iteradas. A desvantagem é que o ajuste do modelo dependeria de uma conexão com a internet e seria mais suscetível a problemas de conexão, que precisariam ser tratados com cuidado.

Outra extensão oportuna seria a criação de um modelo geral de resolução de Captchas,

desenvolvido a partir das bases que foram construídas durante a pesquisa. Esse modelo poderia ser utilizado como modelo inicial para a resolução de novos Captchas e aprimorado com o oráculo, podendo até afastar completamente a necessidade de anotação manual. No momento, não é possível saber que esse modelo realmente funcionaria na prática, já que i) nada garante que ele tenha uma acurácia maior que 10% para um novo Captcha, mesmo se construído com base em Captchas de várias origens e ii) foram encontradas evidências de que os modelos com acurácia menor de 10% têm mais dificuldades em melhorar com o método WAWL.

A presente tese foi fruto de um longo processo de investigação e desenvolvimento, investigando os aspectos relevantes dos Captchas de textos em imagens. Espera-se que a metodologia proposta, os resultados obtidos e os pacotes computacionais desenvolvidos sejam úteis na luta pela abertura dos dados públicos, especialmente no judiciário. O uso de Captchas em sites de serviços públicos deve acabar.

Bibliografia

AHN, L. VON et al. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. **Science**, v. 321, n. 5895, p. 1465–1468, 12 set. 2008. Disponível em: <<https://www.science.org/doi/10.1126/science.1160379>>.

AHN, L. VON; BLUM, M.; LANGFORD, J. **Telling humans and computers apart automatically or how lazy cryptographers do AI** (Tech. Rep. No. CMU-CS-02-117). Disponível em: <<http://reports-archive.adm.cs.cmu.edu/anon/2002/CMU-CS-02-117.pdf>>.

ALLAIRE, J.; TANG, Y. tensorflow: R Interface to 'TensorFlow'. 2022. Disponível em: <<https://CRAN.R-project.org/package=tensorflow>>.

BALDI, P.; SADOWSKI, P. J. Understanding dropout. **Advances in neural information processing systems**, v. 26, 2013.

BLUM, A.; KALAI, A. A note on learning from multiple-instance examples. **Machine learning**, v. 30, n. 1, p. 2329, 1998.

BOETTIGER, C.; HO, T. piggyback: Managing Larger Data on a GitHub Repository. 2022. Disponível em: <<https://CRAN.R-project.org/package=piggyback>>.

CHELLAPILLA, K. et al. **Designing human friendly human interaction proofs (HIPs)**. : CHI '05. New York, NY, USA: Association for Computing Machinery, 2 abr. 2005. Disponível em: <<https://doi.org/10.1145/1054972.1055070>>.

CHELLAPILLA, K.; SIMARD, P. Using machine learning to break visual human interaction proofs (HIPs). **Advances in neural information processing systems**, v. 17, 2004.

COLOSIMO, E. A.; GIOLO, S. R. **Análise de sobrevivência aplicada**. Editora Blucher, 2006.

COUR, T.; SAPP, B.; TASKAR, B. Learning from partial labels. **The Journal of Machine Learning Research**, v. 12, p. 15011536, 2011.

Diagnóstico do Contencioso Tributário Administrativo., [s.d.]. Disponível em: <<https://abj.org.br/pesquisas/bid-tributario/>>.

FALBEL, D. luz: Higher Level 'API' for 'torch'. a2022. Disponível em: <<https://CRAN.R-project.org/package=luz>>.

FALBEL, D. torchvision: Models, Datasets and Transformations for Images. b2022. Disponível em: <<https://CRAN.R-project.org/package=torchvision>>.

- FALBEL, D.; LURASCHI, J. torch: Tensors and Neural Networks with 'GPU' Acceleration. a2022. Disponível em: <<https://CRAN.R-project.org/package=torch>>.
- FALBEL, D.; LURASCHI, J. torch: Tensors and Neural Networks with 'GPU' Acceleration. b2022. Disponível em: <<https://CRAN.R-project.org/package=torch>>.
- FENG, L. et al. Provably consistent partial-label learning. **Advances in Neural Information Processing Systems**, v. 33, p. 1094810960, a2020.
- FENG, L. et al. **Learning with multiple complementary labels**. PMLR, b2020.
- GALAR, M. et al. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. **IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)**, v. 42, n. 4, p. 463484, 2011.
- GEORGE, D. et al. A generative vision model that trains with high data efficiency and breaks text-based CAPTCHAs. **Science**, v. 358, n. 6368, p. eaag2612, 2017.
- GOODFELLOW, I. J. et al. Multi-digit number recognition from street view imagery using deep convolutional neural networks. **arXiv preprint arXiv:1312.6082**, 2013.
- GOODFELLOW, I. J. et al. **Generative Adversarial Networks**. [s.d.].
- GRANDVALET, Y. **Logistic regression for partial labels**. 2002.
- HÜLLERMEIER, E.; BERINGER, J. Learning from ambiguously labeled examples. **Intelligent Data Analysis**, v. 10, n. 5, p. 419439, 2006.
- Inaccessibility of CAPTCHA.**, [s.d.]. Disponível em: <<https://www.w3.org/TR/turingtest/>>.
- IOFFE, S.; SZEGEDY, C. **Batch normalization: Accelerating deep network training by reducing internal covariate shift**. PMLR, 2015.
- ISHIDA, T. et al. Learning from complementary labels. **Advances in neural information processing systems**, v. 30, b2017.
- ISHIDA, T. et al. Learning from complementary labels. **Advances in neural information processing systems**, v. 30, a2017.
- JIN, R.; GHAHRAMANI, Z. Learning with multiple labels. **Advances in neural information processing systems**, v. 15, 2002.
- KAUR, K.; BEHAL, S. Captcha and Its Techniques: A Review. **International Journal of Computer Science and Information Technologies**, v. 5, 1 jan. 2014.
- KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. [s.d.].
- KUHN, M.; JOHNSON, K. **Feature engineering and selection: A practical approach for predictive models**. CRC Press, 2019.
- LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 22782324, 1998.
- LECUN, Y. A. et al. Efficient backprop. Em: Springer, 2012. p. 948.

BIBLIOGRAFIA

- LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, v. 521, n. 7553, p. 436444, b2015.
- LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, v. 521, n. 7553, p. 436444, a2015.
- LI, J.; TSUNG, F.; ZOU, C. Multivariate binomial/multinomial control chart. **IIE Transactions**, v. 46, n. 5, p. 526542, 2014.
- LILLIBRIDGE, M. D. et al. **Method for Selectively Restricting Access to Computer Systems.**, fev. 2001.
- LIU, L.; DIETTERICH, T. A conditional multinomial mixture model for superset label learning. **Advances in neural information processing systems**, v. 25, 2012.
- MICHENER, G.; MONCAU, L. F.; VELASCO, R. B. **Estado brasileiro e transparência avaliando a aplicação da Lei de Acesso à Informação.**
- MORI, G.; MALIK, J. **Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA.** IEEE, 2003.
- MURRAY, N.; MARCHESOTTI, L.; PERRONNIN, F. **AVA: A large-scale database for aesthetic visual analysis.** IEEE, 2012.
- MURRAY-RUST, P. Open data in science. **Nature Precedings**, p. 11, 2008.
- NA, B. et al. **Deep Generative Positive-Unlabeled Learning under Selection Bias.** : CIKM '20. New York, NY, USA: Association for Computing Machinery, 19 out. 2020. Disponível em: <<https://doi.org/10.1145/3340531.3411971>>.
- NELDER, J. A.; WEDDERBURN, R. W. Generalized linear models. **Journal of the Royal Statistical Society: Series A (General)**, v. 135, n. 3, p. 370384, 1972.
- NOH, H. et al. Regularizing deep neural networks by noise: Its interpretation and optimization. **Advances in Neural Information Processing Systems**, v. 30, 2017.
- Observatório da insolvência: Rio de Janeiro.**, [s.d.]. Disponível em: <<https://abj.org.br/pesquisas/obsrjrj/>>.
- OOMS, J. magick: Advanced Graphics and Image-Processing in R. 2021. Disponível em: <<https://CRAN.R-project.org/package=magick>>.
- R CORE TEAM. **R: A Language and Environment for Statistical Computing.** Vienna, Austria: R Foundation for Statistical Computing, 2021. Disponível em: <<https://www.R-project.org/>>.
- RAMESH, A. et al. **Hierarchical Text-Conditional Image Generation with CLIP Latents.** [s.d.].
- RESHEF, E.; RAANAN, G.; SOLAN, E. **Method and System for Discriminating a Human Action from a Computerized Action.**, 2005.
- SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction.** MIT press, 2018.

Tempo dos processos relacionados à adoção., [s.d.]. Disponível em: <<https://abj.org.br/pesquisas/adocao/>>.

TRECENTI, J. et al. decrypTr: An extensible API for breaking captchas. 2022.

TURING, A. M. Computing machinery and intelligence. Em: Springer, 2009. p. 2365.

USHEY, K.; ALLAIRE, J.; TANG, Y. reticulate: Interface to 'Python'. 2022. Disponível em: <<https://CRAN.R-project.org/package=reticulate>>.

VON AHN, L. et al. **Captcha: Telling Humans and Computers Apart Automatically**. Proceedings of Eurocrypt. **Anais...**2003.

VON AHN, L.; BLUM, M.; LANGFORD, J. Telling Humans and Computers Apart Automatically. **Communications of the ACM**, v. 47, n. 2, p. 56–60, 2004.

WANG, Y. et al. Make complex captchas simple: a fast text captcha solver based on a small number of samples. **Information Sciences**, v. 578, p. 181194, 2021.

WICKHAM, H. stringr: Simple, Consistent Wrappers for Common String Operations. b2022. Disponível em: <<https://CRAN.R-project.org/package=stringr>>.

WICKHAM, H. rvest: Easily Harvest (Scrape) Web Pages. a2022. Disponível em: <<https://CRAN.R-project.org/package=rvest>>.

WICKHAM, H.; BRYAN, J.; BARRETT, M. usethis: Automate Package and Project Setup. 2022. Disponível em: <<https://CRAN.R-project.org/package=usethis>>.

WICKHAM, H.; HESTER, J.; OOMS, J. xml2: Parse XML. 2021. Disponível em: <<https://CRAN.R-project.org/package=xml2>>.

YE, G. et al. **Yet another text captcha solver: A generative adversarial network based approach**. b2018.

YE, G. et al. **Yet another text captcha solver: A generative adversarial network based approach**. a2018.

YU, X. et al. **Learning with biased complementary labels**. 2018.

YUAN, X. et al. Adversarial examples: Attacks and defenses for deep learning. **IEEE transactions on neural networks and learning systems**, v. 30, n. 9, p. 28052824, 2019.

ZHAO, B. Web scraping. **Encyclopedia of big data**, p. 13, a2017. Disponível em: <https://www.researchgate.net/profile/Bo-Zhao-3/publication/317177787_Web_Scraping/links/5c293f85a6fdccfc7073192f/Web-Scraping.pdf>.

ZHAO, B. Web scraping. **Encyclopedia of big data**, p. 13, b2017. Disponível em: <https://www.researchgate.net/profile/Bo-Zhao-3/publication/317177787_Web_Scraping/links/5c293f85a6fdccfc7073192f/Web-Scraping.pdf>.

ZHOU, Z.-H. A brief introduction to weakly supervised learning. **National science review**, v. 5, n. 1, p. 4453, 2018.

ZHU, X. J. Semi-supervised learning literature survey. 2005.

Apêndice A

Pacotes

Este apêndice foi construído para mostrar a estrutura do pacote e suas funcionalidades, mas também um pouco da história. O pacote `{captcha}` é fruto de um trabalho da comunidade e o trabalho de modelagem de Captchas usando técnicas de *deep learning* começou alguns anos antes do início da tese de doutorado.

O trabalho de resolução de Captchas pelo autor da tese surgiu no ano de 2016. Como foi comentado na introdução da tese, é muito comum se deparar com desafios de Captchas ao raspar dados do judiciário, já que os dados não são abertos.

O primeiro Captcha a ser investigado foi o do sistema e-SAJ. O desafio era utilizado no site do TJSP que, depois de alguns anos, passou a utilizar o sistema reCaptcha. O Captcha do SAJ faz parte da tese, mas tem como fonte de dados o TJBA, que continua utilizando o desafio até o momento que os sites foram investigados pela última vez, em setembro de 2022.

A primeira abordagem para resolver o Captcha do e-SAJ foi utilizando heurísticas para separar as letras, em 2016. Infelizmente o pacote original, chamado `{captchasaj}`, foi removido da *internet*, mas um código legado construído para o TJRS está disponível [neste link](#). Nessa abordagem, as letras primeiro são segmentadas, alimentando um modelo de florestas aleatórias que considera os pixels da imagem como variáveis preditoras e a letra como resposta. Esses trabalhos tiveram contribuições importantes de Fernando Corrêa e Athos Damiani.

A segunda abordagem para resolver os Captchas foi utilizando o áudio, também em 2016. O código para resolver o Captcha da RFB utilizando áudio está disponível [neste link](#). A ideia de resolução era parecida, passando pelo procedimento de segmentação e depois de modelagem, mas tinha um passo intermediário de processamento envolvendo engenharia de *features* (KUHN; JOHNSON, 2019). O trabalho teve contribuições importantes de Athos Damiani.

Com o advento da ferramenta TensorFlow para o R (ALLAIRE; TANG, 2022), os modelos passaram a utilizar modelos de redes neurais. No início, por falta de conhecimento da área, a arquitetura das redes era demasiadamente complexa. Depois que os primeiros modelos começaram a funcionar, notou-se que as etapas de pré-processamento com seg-

mentação e algumas camadas das redes eram desnecessárias para ajustar os modelos. Essa parte teve grande contribuição de Daniel Falbel, que foi a pessoa que introduziu o TensorFlow e a área de *deep learning* aos colegas.

Depois de resolver com sucesso alguns Captchas, notou-se que seria possível criar um ambiente completo de modelagem de Captchas. Isso deu origem ao pacote {decryptr} (TRECENTI et al., 2022), que foi construído em 2017 durante uma *datathon* (uma maratona de programação), na casa de amigos. O trabalho teve grandes contribuições de Caio Lente, com participação das colegas de faculdade Milene Farhat e Beatriz Vianna.

Com o passar do tempo, o pacote {decryptr} ficou cada vez mais estável, funcionando como dependência de várias ferramentas utilizadas nos trabalhos de jurimetria. O pacote também ganhou um site: <<https://decryptr.xyz/>> e uma API com acesso gratuito, precisando apenas de uma chave de acesso. A ferramenta ficou bastante popular, com **178 estrelas no GitHub** no mês de dezembro de 2022. Essas ferramentas envolveram contribuições principalmente de Caio Lente e Daniel Falbel.

A construção do pacote {captcha} separada do {decryptr} se deu por dois motivos. Primeiro, o pacote {decryptr}, por ser o primeiro a tratar do assunto, possui muitos códigos legado e dificuldades de instalação por conta da dependência do python, necessário para o funcionamento do TensorFlow, que é chamado através do pacote {reticulate} (USHEY; ALLAIRE; TANG, 2022). Além disso, a implementação das técnicas do oráculo envolviam modificações na função de perda, que são difíceis de implementar no ambiente do {tensorflow}, justamente por conta da necessidade de conhecer o código python que roda por trás dos códigos mais usuais.

Com o advento do pacote {torch} (FALBEL; LURASCHI, 2022b), no entanto, tudo foi facilitado. O pacote não possui dependências com o python, além de ser bastante transparente e flexível na construção da arquitetura do modelo, funções de perda e otimização. O pacote, também construído por Daniel Falbel, é um grande avanço científico e facilitou muito a construção dos códigos desta tese.

O pacote {captcha}, apesar de ter sido construído do zero, foi desenvolvido durante *lives* realizadas na plataforma *Twitch*. A construção em *lives* foi interessante porque era possível obter *feedback* e ideias da comunidade durante a construção da ferramenta, o que acelerou o desenvolvimento e auxiliou na arquitetura do pacote.

Com o desenvolvimento da tese, notou-se a necessidade de construir alguns pacotes adicionais. Os pacotes {captchaDownload} e {captchaOracle} foram desenvolvidos para facilitar a obtenção dos resultados da tese, enquanto o pacote {captcha} pode ser utilizado por qualquer pessoa interessada em visualizar, classificar e resolver Captchas. As próximas subseções do apêndice descrevem os três pacotes.

A.1 Pacote captcha

O pacote {captcha} foi construído para funcionar como uma caixa de ferramentas para pessoas que desejam trabalhar com Captchas. O pacote possui funções de leitura, visualização, classificação, preparação de dados, modelagem, carregamento de modelos

pré-treinados e predição. O pacote também permite a construção de um fluxo de trabalho para resolver um novo Captcha, criando um pacote para orquestrar o passo-a-passo.

A.1.1 Uso básico

A utilização básica do {captcha} envolve as funções `read_captcha()`, `plot()`, `captcha_annotate()`, `captcha_load_model()` e `decrypt()`. As funções são explicadas abaixo.

A função `read_captcha()` lê um vetor de arquivos de imagens e armazenar na memória. Por trás, a função utiliza o pacote {magick} para lidar com os tipos de arquivos que podem aparecer (JPEG, PNG, entre outros).

```
library(captcha)
exemplo <- "assets/img/dados_tjmg.jpeg"
captcha <- read_captcha(exemplo)

captcha
#>   format width height colorspace matte filesize density
#> 1   JPEG   100    50      sRGB FALSE    4530   72x72
```



A função retorna um objeto com a classe `captcha`, que pode ser utilizada por outros métodos.

```
class(captcha)
#> [1] "captcha"
```

O objeto é uma lista com três elementos: `$img`, que contém imagem lida com o pacote {magick}, `$lab`, que contém o rótulo da imagem (por padrão, `NULL`) e `$path`, que contém o caminho da imagem que foi lida.

```
str(captcha)
#> Class 'captcha'  hidden list of 3
#>  $ img :Class 'magick-image' <externalptr>
#>  $ lab : NULL
#>  $ path: chr "assets/img/dados_tjmg.jpeg"
```


A função `read_captcha()` possui um parâmetro `lab_in_path=`, que indica se o rótulo está contido no caminho da imagem. Se `lab_in_path=TRUE`, a função tentará extrair o rótulo do arquivo (obtendo o texto que vem depois do último `_` do caminho) e armazenar no elemento `$lab`.

```
exemplo <- "assets/img/mnist128c49c36e13_6297.png"
captcha <- read_captcha(exemplo, lab_in_path = TRUE)

str(captcha)
#> Class 'captcha'  hidden list of 3
#> $ img :Class 'magick-image' <externalptr>
#> $ lab : chr "6297"
#> $ path: chr "assets/img/mnist128c49c36e13_6297.png"
```

A função `plot()` é um método de classe S3 do R básico. A função foi implementada para facilitar a visualização de Captchas. A função recebe uma lista de imagens obtida pela função `read_captcha()` e mostra o Captcha visualmente, como na Figura A.1.

```
exemplo <- "assets/img/dados_tjmg.jpeg"
captcha <- read_captcha(exemplo)
plot(captcha)
```

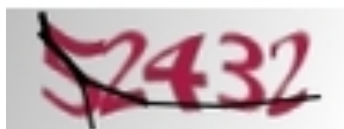


Figura A.1: Exemplo de aplicação da função `plot` a um objeto `captcha`.

Um aspecto interessante da função `plot()` é que ela lida com uma lista de Captchas. Isso é útil quando o interesse é visualizar vários Captchas de uma vez na imagem. A Figura A.2 mostra um exemplo de aplicação

```
exemplos <- paste0("assets/img/", c(
  "dados_tjmg.jpeg",
  "dados_esaj.png",
  "dados_rfb.png",
  "dados_sei.png"
))
captchas <- read_captcha(exemplos)
plot(captchas)
```

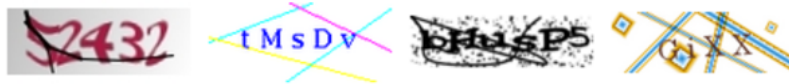



Figura A.2: Exemplo de aplicação da função `plot` a um objeto `captcha` com várias imagens.

Por padrão, a função `plot` dispõe as imagens em quatro colunas. Para mudar o padrão, é possível modificar as opções usando `options(captcha.print.cols = N)`, onde `N` é o número de colunas desejado. A Figura A.3 mostra um exemplo com duas colunas.

```
options(captcha.print.cols = 2)
plot(captchas)
```



Figura A.3: Exemplo de aplicação da função `plot` a um objeto `captcha` com várias imagens, disponibilizadas em duas colunas.

Quando o vetor de `Captchas` é muito grande, a função `plot()` mostra um número máximo de imagens, acompanhado de uma mensagem. Por padrão, esse número é 100, com 25 linhas e 4 colunas. A opção pode ser sobrescrita combinando as opções `captcha.print.cols=` e `captcha.print.rows=`. A Figura A.4 mostra um exemplo do comportamento da função quando o número de imagens excede 100.

```
# mais de 100 imagens:
exemplos <- rep("assets/img/dados_tjmg.jpeg", 110)
```

```

captchas <- read_captcha(exemplos)
plot(captchas)
#> i Too many images, printing first 100. To override, run
#> * options('captcha.print.rows' = MAX_ROWS)
#> * options('captcha.print.cols' = COLUMNS)

```



Figura A.4: Demonstração da função `plot()` com muitas imagens.

Um detalhe interessante é que é possível criar subconjuntos de um objeto de classe `captcha` simplesmente utilizando o operador `[]`. A função `length()` também pode ser utilizada para medir a quantidade de imagens lidas. A Figura A.5 mostra um exemplo dessas operações.

```

captchas_subset <- captchas[1:20]
length(captchas_subset) # 20
#> [1] 20
plot(captchas_subset)

```



Figura A.5: Demonstração das funções de `subset` e `length` aplicadas a um objeto do tipo `captcha`.

Por fim, se a imagem possui um rótulo, por padrão, a função `plot()` mostra o rótulo no canto da imagem. A Figura A.6 mostra um exemplo.

```
exemplo <- "assets/img/mnist128c49c36e13_6297.png"
captcha <- read_captcha(exemplo, lab_in_path = TRUE)
plot(captcha)
```



Figura A.6: Demonstração da função `plot()` quando o `Captcha` possui um rótulo

A função `captcha_annotate()` serve para classificar uma imagem de `Captcha`, manual ou automaticamente. Isso é feito modificando o caminho da imagem, adicionando o texto `_rotulo` ao final do caminho do arquivo. A função possui os parâmetros listados abaixo:

- `files=`: objeto de classe `captcha` lido com a função `read_captcha()` (recomendado) ou vetor de caminhos de arquivos.
- `labels=`: (opcional) vetor com os rótulos das imagens. Deve ter o mesmo `length()` do que `files=`. Por padrão, o valor é `NULL`, indicando que deve ser aberto um prompt para que o usuário insira a resposta manualmente.
- `path=`: (opcional) caminho da pasta onde os arquivos classificados serão salvos. Por padrão, salva os arquivos com nomes modificados na mesma pasta dos arquivos originais.

- `rm_old=`: (opcional) deletar ou não os arquivos originais. Por padrão, é `FALSE`.

A função, depois de aplicada, retorna um vetor com os caminhos dos arquivos modificados. O parâmetro `labels=` é útil para lidar com situações em que sabemos o rótulo do Captcha. Por exemplo, em um fluxo de trabalho que utiliza o oráculo, pode ser que um modelo inicial já forneça o valor correto do rótulo.

Quando não existe um rótulo, a função `captcha_annotate()`, que abre o prompt para classificação e aplica `plot()` para visualizar a imagem. A Figura A.7 mostra um exemplo de aplicação da função `captcha_annotate()` no software **RStudio**.

```
> captcha ← read_captcha("inst/book/assets/img/dados_rfb.png")
> classify(captcha)
Answer: bhusp5
[1] "inst/book/assets/img/dados_rfb_bhusp5.png"
>
```

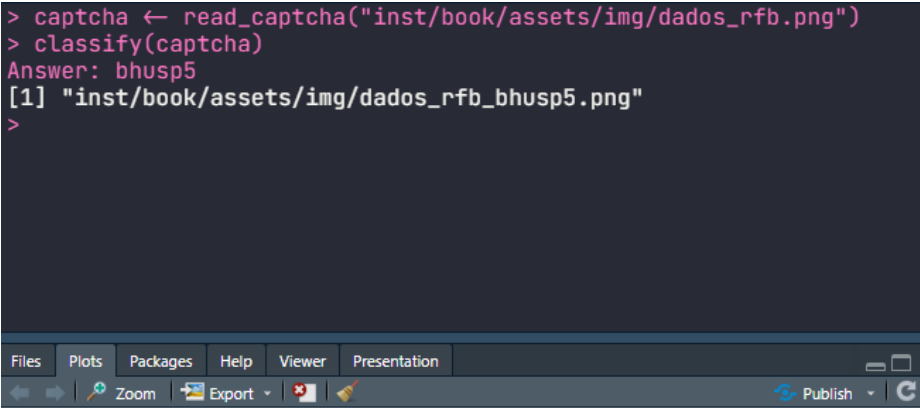



Figura A.7: Exemplo de aplicação da função `classify`. O rótulo `bhusp5` foi inserido manualmente.

Por último, a função `decrypt()` tem o papel de obter o rótulo de uma imagem utilizando um modelo já treinado para aquele tipo de imagem. A função recebe dois argumentos: `file=` que pode ser tanto o caminho do arquivo quanto um objeto de classe `captcha`, e um argumento `model=`, que contém um modelo de classe `luz_module_fitted`, ajustado utilizando as ferramentas que serão apresentadas na próxima subseção.

Para a tese, foram desenvolvidos modelos para vários Captchas diferentes. É possível carregar um modelo já treinado usando a função `captcha_load_model()`, podendo receber em seu único parâmetro `path=` o caminho de um arquivo contendo um modelo ajustado ou uma *string* com o nome de um modelo já treinado, como `"rfb"`, por exemplo. Os modelos treinados são armazenados nos **releases do repositório do pacote captcha**, são baixados e controlados pelo pacote `{piggyback}` (BOETTIGER; HO, 2022) e são lidos utilizando o pacote `{luz}`, que será descrito em maiores detalhes na próxima subseção. No momento de submissão da tese, os Captchas com modelos desenvolvidos eram `trf5`,

tjmg, trt, esaj, jucesp, tjpe, tjrs, cadesp, sei e rfb.

A Figura A.8 resume visualmente as funções apresentadas até o momento. As setas indicam a dependência das funções de objetos gerados por outras funções.

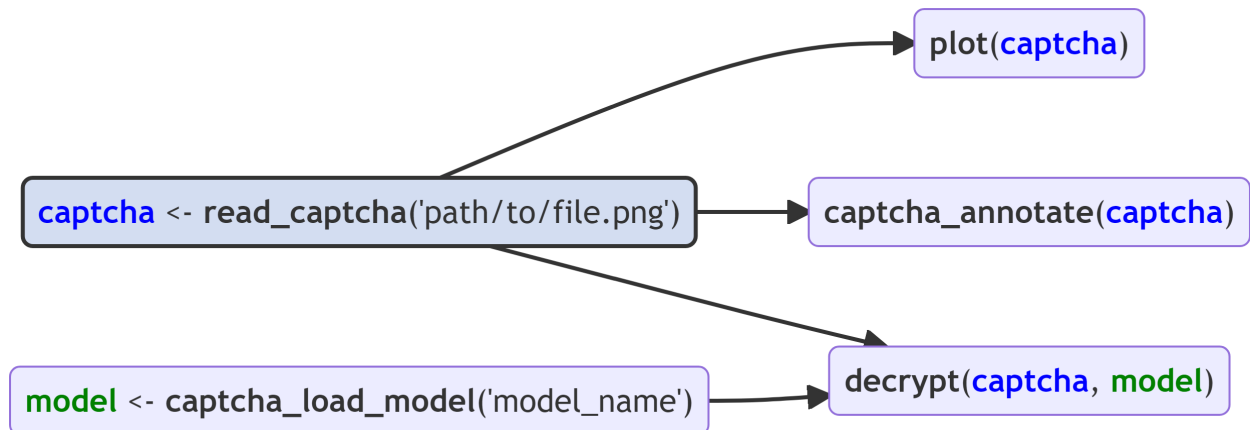


Figura A.8: Diagrama das funções básicas do pacote {captcha}

A.1.2 Modelagem

O pacote {captcha} também fornece uma interface básica para o desenvolvimento de modelos a partir de uma base completamente classificada. A classificação pode ser feita manualmente pela função `captcha_annotate()`, apresentada anteriormente, ou por outro método desenvolvido pelo usuário.

A parte de modelagem parte de algumas premissas sobre a base de dados. As imagens precisam estar em uma pasta e ter o padrão caminho/do/arquivo/<id>_<lab>.<ext>, onde:

- <id>: pode ser qualquer nome, de preferência sem acentuação ou outros caracteres especiais, para evitar problemas de *encoding*. Geralmente, é um *hash* identificando o tipo e id do captcha. **Observação:** ao classificar um caso, é importante que o id seja único, já que dois Captchas podem ter o mesmo rótulo.
- <lab>: é o rótulo do Captcha. Pode ser um conjunto de caracteres entre [a-zA-Z0-9], diferenciando maiúsculas e minúsculas se necessário. No momento, todos os arquivos em uma pasta devem ter a mesma quantidade de caracteres (comprimento homogêneo). Futuramente, o pacote poderá considerar Captchas de comprimento heterogêneo.
- <ext>: extensão do arquivo. Pode ser .png, .jpeg ou .jpg. As operações também funcionam para o formato .svg, mas pode apresentar problemas por conta da transparência da imagem.

Atendidas as premissas da base classificada, é possível ajustar um modelo de redes neurais usando o pacote {captcha}. No entanto, como o ajuste de modelos de redes neurais envolve uma série de nuances e pequenas adaptações, optou-se por exportar funções

em dois níveis de aprofundamento. A primeira é a **automatizada**, utilizando a função `captcha_fit_model()` descrito a seguir, enquanto a segunda é a **procedimental**, utilizando o passo a passo descrito na Subseção [A.1.3](#).

A função `captcha_fit_model()` ajusta um modelo a partir de uma pasta com arquivos classificados. A função recebe os parâmetros: `dir=`, contendo o caminho dos arquivos classificados; `dir_valid=`, (opcional) contendo o caminho dos arquivos classificados para validação; `prop_valid=`, contendo a proporção da base de treino a ser considerada como validação, ignorada quando `dir_valid=` é fornecida (por padrão, considera-se 20% da base para validação).

A função `captcha_fit_model()` também possui alguns parâmetros relacionados à modelagem. São eles: `dropout=`, especificando o percentual de *dropout* aplicado às camadas ocultas da rede (por padrão, 0.25); `dense_units=`, especificando a quantidade de unidades na camada oculta que vem depois das camadas convolucionais (por padrão, 200); `decay=`, especificando o percentual de decaimento da taxa de aprendizado (por padrão, 0.99); `epochs=` número de épocas para ajuste do modelo (por padrão 100). Uma observação importante é que o modelo está configurado para parar o ajuste após 20 iterações sem redução significativa na função de perda (arbitrado em 1%; para mais detalhes ver a Subseção [A.1.3](#)).

No final, a função retorna um modelo ajustado com classe `luz_module_fitted`, que pode ser salvo em disco utilizando-se a função `luz_save()`. O modelo também pode ser serializado para utilização em outros pacotes como `pytorch`. Um tutorial sobre serialização pode ser encontrado na [documentação do pacote torch](#).

O pacote `{captchaOracle}` possui uma interface similar para trabalhar com bases parcialmente classificadas. Como a estrutura de dados nesse caso é mais complexa e pode evoluir no futuro, os códigos foram organizados em outro pacote. Mais detalhes na Seção [A.3](#).

Na documentação do pacote `{captcha}`, foi adicionado um exemplo de aplicação. O exemplo utiliza captchas gerados usando a função `captcha_generate()`, que gera Captchas utilizando o pacote `{magick}`. O Captcha foi criado para a construção da tese, apelidado de R-Captcha, e possui os seguintes parâmetros:

- `write_disk=`: salvar os arquivos em disco? Por padrão, é falso.
- `path=`: Caminho para salvar arquivos em disco, caso o parâmetro anterior seja verdadeiro.
- `chars=`: Quais caracteres usar na imagem.
- `n_chars=`: O comprimento do Captcha.
- `n_rows=`: Altura da imagem, em pixels.
- `n_cols=`: Largura da imagem, em pixels.
- `p_rotate=`: Probabilidade de rotação da imagem.
- `p_line=`: Probabilidade de adicionar um risco entre as letras.
- `p_stroke=`: Probabilidade de adicionar uma borda nas letras.
- `p_box=`: Probabilidade de adicionar uma caixa (retângulo) em torno das letras.
- `p_implode=`: Probabilidade de adicionar efeitos de implosão.
- `p_oilpaint=`: Probabilidade de adicionar efeitos de tinta a óleo.

- `p_noise`=: Probabilidade de adicionar um ruído branco no fundo da imagem.
- `p_lat`=: Probabilidade de aplicar o algoritmo *local adaptive thresholding* à imagem.

A.1.3 Resolvendo um novo Captcha do zero

Em algumas situações, pode ser desejável rodar modelos de forma customizada. Isso acontece pois modelos de aprendizagem profunda costumam precisar de diversos pequenos ajustes, como a taxa de aprendizado, utilização de outras funções de otimização, camadas computacionais e funções de pré-processamento.

A função `captcha_fit_model()`, apresentada na subseção anterior, é engessada. Ela aceita alguns parâmetros para estruturar o modelo, mas não possui elementos suficientes para customização. É para isso que pacotes como `{torch}` e `{luz}` existem, pois criam ambientes de computação mais flexíveis para operar os modelos de aprendizado profundo.

Outra desvantagem da utilização do `captcha_fit_model()` é a disponibilização dos modelos. Um modelo pode ser utilizado localmente, mas a tarefa de disponibilizar as bases de dados e o modelo para outras pessoas não tem um procedimento bem definido.

Para organizar o fluxo de trabalho, implementou-se um fluxo de classificação de Captchas dentro do pacote `{captcha}`. A função que orquestra esse fluxo é a `new_captcha()`. A função possui apenas um parâmetro, `path`=, que é o caminho de uma nova pasta a ser criada.

A função também pode ser chamada criando-se um projeto dentro do próprio RStudio. A Figura A.9 mostra um exemplo de utilização do template dentro do RStudio, após clicar em Novo Projeto > Novo Diretório.

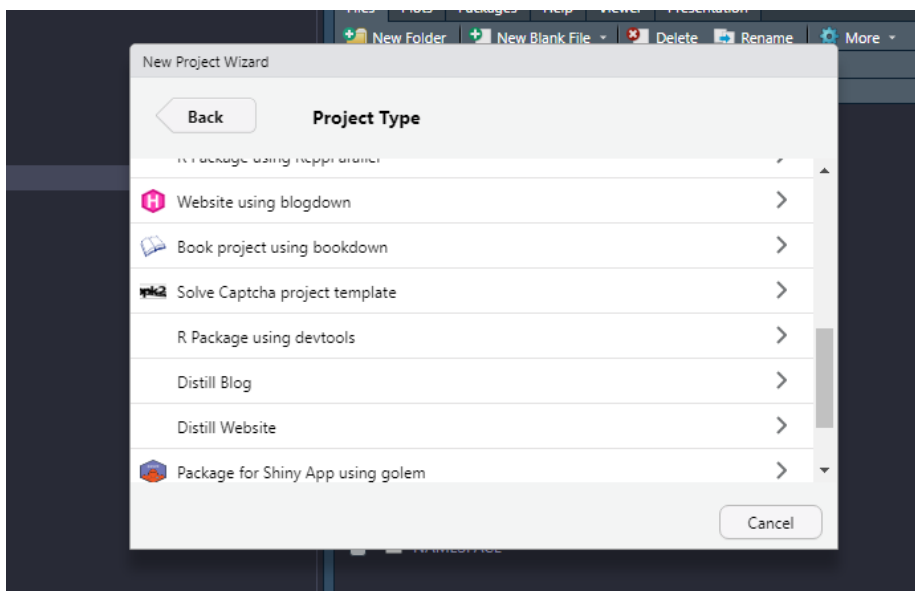


Figura A.9: Exemplo de criação de um novo projeto de Captcha utilizando o RStudio.

Ao criar um novo projeto, pelo comando `new_captcha()` ou pela interface do RStudio, uma nova janela é aberta. O projeto contém quatro arquivos:

- `01_download.R`: Contém algumas funções para auxiliar no desenvolvimento de funções que baixam Captchas em um caso real. Na prática, as funções que baixam Captchas precisam ser adaptadas porque os sites são organizados de formas muito diferentes.
- `02_annotate.R`: Contém um *template* para classificação manual de Captchas. A classificação manual pode tanto ser realizada usando a interface criada pelo pacote `{captcha}` quanto externamente. No final, os arquivos classificados devem ser salvos na pasta `img`, no formato descrito na Subseção A.1.2.
- `03_model.R`: Contém um *template* para modelagem, permitindo a customização completa do procedimento de ajuste. O *script* contém comandos para carregar os dados, especificar o modelo, realizar o ajuste e salvar o modelo ajustado.
- `04_share.R`: Contém funções para criar um repositório *git* da solução e disponibilizar o modelo ajustado. O modelo poderá ser lido e aplicado utilizando-se a função `captcha_load_model()`, para utilização em diferentes contextos, sem a necessidade de copiar arquivos localmente.

Sobre a parte de modelagem, cabe uma descrição mais detalhada. O primeiro passo do *script* é criar objetos do tipo *dataset* (objeto que armazena os dados de forma consistente) e *dataloader* (objeto que obtém amostras do dataset, que são utilizadas como os *minibatches* do modelo), com uma estrutura orquestrada pelo pacote `{torch}`.

A função `captcha_dataset()` cria o *dataset* recebendo como parâmetro uma pasta de arquivos e gera um objeto com classes `my_captcha`, `dataset` e `R6`. A função é, na verdade, um objeto do tipo `dataset_generator`, criada utilizando-se a função `dataset()` do pacote `{torch}`. O objeto é chamado da mesma forma que uma função usual do R, aceitando alguns parâmetros adicionais:

- `transform_image=`: operação de transformação a ser aplicada à imagem. Por padrão, utiliza a função `captcha_transform_image()`, que lê a imagem e redimensiona para ficar com dimensões 32×192. A dimensão foi escolhida para facilitar a implementação das camadas convolucionais e para lidar com o fato de que usualmente os Captchas são imagens retangulares.
- `transform_label=`: operação de transformação para gerar a variável resposta. Por padrão, utiliza a função `captcha_transform_label()`, que recebe um vetor de todos os possíveis caracteres do Captcha e aplica a operação `one_hot()`, obtendo-se a versão matricial da resposta com zeros e uns, como descrito na Seção 2.1.1.
- `augmentation=`: operações para aumento de dados. Por exemplo, pode ser uma função que adiciona um ruído aleatório à imagem original para que, ao gerar uma nova amostra, os dados utilizados sejam sempre diferentes.

A função `captcha_dataset()` deve ser aplicada duas vezes, uma para criar a base de treino e outra para criar a base de validação. A separação de bases de treino e validação deve ser feita de forma manual, copiando parte dos Captchas classificados para uma nova pasta, com aleatorização. É papel do usuário separar as bases em pastas distintas carregá-las em um *dataset*.

Em seguida, os *dataloaders* são criados utilizando-se a função `dataloader()` do pacote `{torch}`. Nessa parte é definido o tamanho do *minibatch*, além de outros possíveis parâmetros disponíveis na função do `{torch}`. Para mais detalhes, o usuário pode [acessar](#)

a [documentação da função neste link](#). Devem ser criados *dataloaders* tanto para a base de treino quanto para a base de validação.

A próxima etapa é a especificação do modelo. No *script* de modelagem, o modelo é fornecido pelo objeto `net_captcha` do pacote `{captcha}`. Assim como no caso do *dataset*, o `net_captcha` é um objeto especial do `{torch}`, com classes `CAPTCHA-CNN`, `nn_module` e `nn_module_generator`. O objeto pode ser utilizado como uma função, gerando um módulo do `torch`, similar a uma função de predição. No entanto, por conta da forma que o objeto é utilizado em passos posteriores pelo pacote `{luz}`, o objeto a ser considerado é mesmo o `nn_module_generator`, como colocado no *script*.

Para customizar o modelo, o usuário deve modificar os métodos `initialize()` e `forward()`, acessados dentro do objeto `net_captcha$public_methods`. O primeiro é responsável pela inicialização do modelo, contendo a descrição das operações que são realizadas, como convoluções. O segundo é a função *feed forward* das redes neurais, que recebe uma imagem e retorna um objeto contendo os *logitos* ou probabilidades, no formato da variável resposta.

Por padrão, o código de inicialização do modelo é o descrito abaixo. Os parâmetros `input_dim=`, `output_ndigits=`, `output_vocab_size=` e `vocab=` descrevem, respectivamente, as dimensões da imagem, o comprimento da resposta, o comprimento do alfabeto e os elementos do alfabeto. Os parâmetros `transform=`, `dropout=` e `dense_units=` controlam, respectivamente, a função de transformação da imagem, os hiperparâmetros de *dropout* e a quantidade de unidades na camada densa. É possível notar que os parâmetros das convoluções são fixos, já preparados para funcionar bem com uma imagem de dimensões 32x192.

```
initialize = function(input_dim,
                      output_ndigits,
                      output_vocab_size,
                      vocab,
                      transform,
                      dropout = c(.25, .25),
                      dense_units = 400) {

  # in_channels, out_channels, kernel_size, stride = 1, padding = 0
  self$batchnorm0 <- torch::nn_batch_norm2d(3)
  self$conv1 <- torch::nn_conv2d(3, 32, 3)
  self$batchnorm1 <- torch::nn_batch_norm2d(32)
  self$conv2 <- torch::nn_conv2d(32, 64, 3)
  self$batchnorm2 <- torch::nn_batch_norm2d(64)
  self$conv3 <- torch::nn_conv2d(64, 64, 3)
  self$batchnorm3 <- torch::nn_batch_norm2d(64)
  self$dropout1 <- torch::nn_dropout2d(dropout[1])
  self$dropout2 <- torch::nn_dropout2d(dropout[2])

  self$fc1 <- torch::nn_linear(
    # must be the same as last convnet
```

```

    in_features = prod(calc_dim_conv(input_dim)) * 64,
    out_features = dense_units
  )
  self$batchnorm_dense <- torch::nn_batch_norm1d(dense_units)
  self$fc2 <- torch::nn_linear(
    in_features = dense_units,
    out_features = output_vocab_size * output_ndigits
  )
  self$output_vocab_size <- output_vocab_size
  self$input_dim <- input_dim
  self$output_ndigits <- output_ndigits
  self$vocab <- vocab
  self$transform <- transform
}

```

A função de *feed forward* foi descrita abaixo. A função aplica o passo-a-passo descrito na Seção 2.1.2, recebendo uma imagem x como entrada e retornando uma matriz de logitos, que dão os pesos do modelo para cada letra da resposta. O modelo retorna os logitos e não as probabilidades porque, no passo seguinte, a função de perda considera como entrada os logitos. Se o usuário decidir modificar o método `forward` para retornar probabilidades, precisará também adaptar a função de perda utilizada.

```

forward = function(x) {

  out <- x |>
  # normalize
  self$batchnorm0() |>
  # layer 1
  self$conv1() |>
  torch::nnf_relu() |>
  torch::nnf_max_pool2d(2) |>
  self$batchnorm1() |>

  # layer 2
  self$conv2() |>
  torch::nnf_relu() |>
  torch::nnf_max_pool2d(2) |>
  self$batchnorm2() |>

  # layer 3
  self$conv3() |>
  torch::nnf_relu() |>
  torch::nnf_max_pool2d(2) |>
  self$batchnorm3() |>

  # dense
  torch::torch_flatten(start_dim = 2) |>

```

```

self$dropout1() |>
self$fc1() |>
torch::nnf_relu() |>
self$batchnorm_dense() |>
self$dropout2() |>
self$fc2()

out$view(c(
  dim(out)[1],
  self$output_ndigits,
  self$output_vocab_size
))
}

```

Definida a arquitetura do modelo, o penúltimo passo é o ajuste. O ajuste do modelo é orquestrado pelo pacote `{luz}`, que facilita a criação do *loop* de ajuste dos parâmetros, desempenhando um papel similar ao que o `keras` realiza para o `tensorflow` puro.

No caso dos Captchas, o código `{luz}` para ajuste do modelo segue quatro passos, encadeados pelo operador *pipe*, ou `|>`:

- `setup()`: serve para determinar a função de perda, o otimizador e as métricas a serem acompanhadas. No *script*, a função de perda utilizada é a `nn_multilabel_soft_margin_loss()` do `{torch}`, o otimizador é o `optim_adam()` do `{torch}` e a métrica é a `captcha_accuracy()`, desenvolvida no pacote `{captcha}` para apresentar a acurácia considerando a imagem completa do Captcha e não a acurácia de cada letra da imagem, que seria o resultado se fosse utilizada a função `luz_metric_accuracy()`, do pacote `{luz}`.
- `set_hparams()`: serve para informar os hiperparâmetros e outras informações do modelo. Os parâmetros colocados dentro dessa função são exatamente os parâmetros do método `initialize()` da rede neural criada no passo anterior.
- `set_opt_hparams()`: serve para informar os hiperparâmetros da otimização. Os parâmetros colocados nessa função são passados para a função de otimização. No *script*, o único parâmetro informado é a taxa de aprendizado, fixada em `0.01`.
- `fit()`: serve para inicializar o *loop* de ajuste do modelo. Aqui, é necessário passar os *dataloaders* de treino e validação, a quantidade de épocas (fixada em 100), e os *callbacks*, que são operações a serem aplicadas em diferentes momentos do ajuste (por exemplo, ao final de cada iteração). Por padrão, os *callbacks* são:
 - O decaimento da taxa de aprendizado utilizando uma taxa multiplicativa. A cada iteração, a taxa de aprendizado decai em um fator determinado pela função definida em `lr_lambda`, que por padrão é `0.99`. Ou seja, em cada época, a taxa de aprendizado fica 1% menor.
 - A parada precoce, ou *early stopping*. Por padrão, está configurado para parar o ajuste do modelo se forem passadas 20 épocas sem que o modelo melhore a acurácia em 1% na base de validação. Por exemplo, se em 20 épocas consecutivas o modelo permanecer com acurácia em 53%, o ajuste será encerrado,

mesmo que não tenha passado pelas 100 épocas.

- O arquivo de log. Por padrão, o modelo guarda o histórico de ajuste em um arquivo do tipo *comma separated values* (CSV), contendo a perda e a acurácia do modelo na base de treino e na base de validação, ao final de cada época. O arquivo de log é importante para acompanhar o ajuste do modelo e verificar sua performance ao longo das épocas, podendo dar *insights* sobre possíveis ajustes nos hiperparâmetros.

No final do fluxo definido pelo pacote `{luz}`, será obtido um modelo ajustado. O modelo possui a classe `luz_module_fitted` e pode ser investigado ao rodar o objeto no console do R. No exemplo do R-Captcha apresentado na subseção anterior, o objeto possui as características abaixo. O objeto contém um relatório conciso e bastante informativo, mostrando o tempo de ajuste, as métricas obtidas no treino e na validação e a arquitetura do modelo ajustado.

```
A `luz_module_fitted`
— Time —————
• Total time: 10m 48.1s
• Avg time per training batch: 415ms
• Avg time per validation batch 217ms

— Results —————
Metrics observed in the last epoch.

☒ Training:
loss: 0.0049
captcha acc: 0.996
☒ Validation:
loss: 0.0356
captcha acc: 0.905

— Model —————
An `nn_module` containing 628,486 parameters.

— Modules —————
• batchnorm0: <nn_batch_norm2d> #6 parameters
• conv1: <nn_conv2d> #896 parameters
• batchnorm1: <nn_batch_norm2d> #64 parameters
• conv2: <nn_conv2d> #18,496 parameters
• batchnorm2: <nn_batch_norm2d> #128 parameters
• conv3: <nn_conv2d> #36,928 parameters
• batchnorm3: <nn_batch_norm2d> #128 parameters
• dropout1: <nn_dropout> #0 parameters
• dropout2: <nn_dropout> #0 parameters
• fc1: <nn_linear> #563,400 parameters
• batchnorm_dense: <nn_batch_norm1d> #400 parameters
• fc2: <nn_linear> #8,040 parameters
```

Por último, o modelo deve ser salvo em um arquivo local. Isso é feito utilizando-se a função `luz_save()` do pacote `{luz}`, guardando um objeto com extensão `.pt`, que será disponibilizado no `04_share.R`.

Cabe também um detalhamento do *script* disponibilizado em `04_share.R`. O script utiliza o pacote `{usethis}` (WICKHAM; BRYAN; BARRETT, 2022) para organizar o repositório, configurando o Git (software de versionamento de códigos) e o GitHub (sistema *web* de organização de repositórios). Além disso, o *script* utiliza o pacote `{piggyback}` (BOETTIGER; HO, 2022) para disponibilizar o modelo ajustado nos *releases* do repositório criado. Opcionalmente, o usuário poderá também disponibilizar a base com os arquivos classificados em um arquivo `.zip`, o que é recomendado, pois permite que outras pessoas possam trabalhar com os mesmos dados e aprimorar os modelos.

Um detalhe importante é sobre a inserção de arquivos pesados no repositório. O *script* utiliza *releases* para disponibilizar as soluções porque não é uma boa prática subir arquivos como modelos ajustados ou arquivos brutos (imagens) diretamente no repositório. Isso acontece porque o repositório pode ficar demasiadamente pesado e o histórico do *Git* fica alterado.

Uma vez compartilhado nos releases do repositório, o modelo poderá ser lido por qualquer pessoa, em outras máquinas utilizando o pacote `{captcha}`. Basta rodar o código abaixo e o modelo será carregado.

Com isso, o trabalho pode ser compartilhado e Captchas podem ser resolvidos de forma colaborativa pelas pessoas interessadas. Utilizando o fluxo do `new_captcha()`, as pessoas têm flexibilidade para construir modelos customizados e utilizá-los de forma eficiente.

A.2 Pacote captchaDownload

O pacote `{captchaDownload}` foi construído para armazenar os códigos de baixar dados de Captchas de forma consistente. O pacote também inclui funções para trabalhar com oráculos.

O pacote não foi criado para ser usado por muitas pessoas. O intuito de criar o pacote foi o de organizar as funções utilizadas para realizar as simulações e obter os resultados empíricos da tese.

As funções do pacote `{captchaDownload}` são organizadas em dois tipos principais. As funções de *acesso*, identificadas pelo termo `_access`, fazem o *download* da imagem do Captcha e retornam todas as informações necessárias para fazer a verificação do oráculo, como, por exemplo, *cookies* e dados da sessão do usuário. Já as funções de *teste*, identificadas pelo termo `_test`, servem para verificar se um rótulo fornecido para o Captcha está correto ou não.

As funções ficam mais claras através de um exemplo. No caso do TRF5, por exemplo, o acesso é feito pela página do **sistema PJe**. A função `captcha_access_trf5()` recebe o parâmetro `path=`, que é a pasta para salvar a imagem, retornando uma lista com o caminho da imagem que foi salva e de componentes da sessão do usuário.

```
acesso <- captchaDownload::captcha_access_trf5("assets/img")
acesso
```

```
$f_captcha
assets/img/trf5ac031dafbd.jpeg
```

```
$j_id
[1] "j_id1"
```

```
$u
[1] "https://pje.trf5.jus.br/pjeconsulta/ConsultaPublica/listView.seam"
```

Em seguida, obtém-se o rótulo do modelo. Isso pode ser feito manualmente ou através de um modelo.

```
library(magrittr) # TODO remove
captcha <- read_captcha(acesso$f_captcha)
plot(captcha)
modelo_trf5 <- captcha_load_model("trf5")
(lab <- decrypt(acesso$f_captcha, modelo_trf5))
#> [1] "969588"
```



Figura A.10: Exemplo de Captcha baixado diretamente do TRF5.

Agora, aplica-se a função `captcha_test_trf5()` para verificar se o rótulo está correto ou incorreto. A verificação é feita de forma automática, diretamente da *internet*, através do oráculo. A função recebe dois parâmetros: `obj=` com as informações obtidas da função de acesso; e `label=`, o rótulo obtido. A função retorna `TRUE` se o rótulo está correto e `FALSE` caso contrário.

```
(acertou <- captchaDownload::captcha_test_trf5(acesso, lab))
```

```
[1] TRUE
```

Cada Captcha possui uma função de acesso e uma função de teste. Na prática, se uma pessoa desejar resolver um novo Captcha usando a técnica do oráculo, são essas funções que ela precisará desenvolver. Todas as outras operações podem ser generalizadas para diferentes casos de uso e estão implementadas nos pacotes `{captchaDownload}` e `{captchaOracle}`. Vale notar que a construção dessas funções geralmente é necessária para a construção de *web scrapers*, ou seja, elas não criam dificuldades adicionais para pessoas interessadas em resolver Captchas para acessar dados da *internet*.

A função principal do pacote `{captchaDownload}` é a `captcha_oracle()`. A função é responsável por realizar a classificação parcial automática dos Captchas utilizando um

modelo inicial e o oráculo. A função possui os seguintes parâmetros:

- `path=`: caminho em que os arquivos serão salvos.
- `model=`: modelo para predizer o rótulo de uma imagem.
- `max_ntry=`: quantidade máxima de chutes até desistir.
- `manual=`: caso o máximo de tentativas seja alcançado, abrir o *prompt* para classificar manualmente? Por padrão, sim.
- `captcha_access=`: função que baixa um Captcha e retorna dados da sessão para validar o Captcha, como mostrada anteriormente.
- `captcha_test=`: função que testa se um Captcha está correto a partir de um rótulo específico, como mostrado anteriormente.

A função amarra todos os conceitos necessários para criar novas bases de dados de forma automática. Primeiro, considera o caminho para salvar os dados. Em seguida, considera o modelo e formas de lidar com o oráculo. Por último, recebe as funções de acesso e de teste do Captcha. A função escreve um arquivo de *log* com os resultados dos testes. O arquivo contém `max_ntry` linhas, podendo ter uma linha adicional se `manual=TRUE`, já que, se o modelo errar todas os chutes, a classificação manual deve ser adicionada.

No exemplo do TRF5, a chamada da função `captcha_oracle()` com um chute ficaria da seguinte forma:

```
modelo_trf5 <- captcha_load_model("trf5")

captchaDownload::captcha_oracle(
  path = "assets/img/",
  model = modelo_trf5,
  max_ntry = 1,
  manual = TRUE,
  captcha_access = captchaDownload::captcha_access_trf5,
  captcha_test = captchaDownload::captcha_test_trf5
)
```

☒ Acertou!!!

No teste do exemplo, a função acertou, salvando o seguinte arquivo de *log*. Espaços foram adicionados para facilitar a visualização.

```
ntry, label , type, result
1,      569328, auto, TRUE
```

Abaixo, foi colocado um modelo ruim para o TRT, para forçar o modelo a errar todos os chutes. O resultado é o log abaixo

```
modelo <- captcha_load_model("assets/modelo_ruim.pt")

captchaDownload::captcha_oracle(
  path = "assets/img/",
  model = modelo,
  max_ntry = 10,
  manual = TRUE,
```

```
captcha_access = captchaDownload::captcha_access_trt,
captcha_test = captchaDownload::captcha_test_trt
)
```

❑ Temos 10 candidatos...

❑ Errou! 0 chute foi: v2su7w

❑ Errou! 0 chute foi: t2su7w

❑ Errou! 0 chute foi: v2su7y

❑ Errou! 0 chute foi: t2su7y

❑ Errou! 0 chute foi: y2su7w

❑ Errou! 0 chute foi: v2su7h

❑ Errou! 0 chute foi: t2su7h

❑ Errou! 0 chute foi: y2su7y

❑ Errou! 0 chute foi: v2wu7w

Label: v2xu7w

No novo exemplo, a função errou todos os dez chutes, salvando o seguinte arquivo de *log*. Espaços foram adicionados para facilitar a visualização. O último valor é um rótulo inserido manualmente.

ntry,	label,	type,	result
1,	v2su7w,	auto,	FALSE
2,	t2su7w,	auto,	FALSE
3,	v2su7y,	auto,	FALSE
4,	t2su7y,	auto,	FALSE
5,	y2su7w,	auto,	FALSE
6,	v2su7h,	auto,	FALSE
7,	t2su7h,	auto,	FALSE
8,	y2su7y,	auto,	FALSE
9,	v2wu7w,	auto,	FALSE
10,	92su7w,	auto,	FALSE
NA,	v2xu7w,	manual,	TRUE

Se o parâmetro `manual=FALSE` e o modelo não consegue acertar o rótulo, a função também adiciona a mensagem:

❑ Errado depois de todas as tentativas...

Em alguns casos, é possível que a função realize menos do que `max_ntry` chutes. Isso acontece quando a probabilidade do melhor rótulo depois do chute errado é muito pequena, segundo o modelo. Isso é feito pela função `captcha_candidates()`, que considera como padrão o corte de 0.01 de probabilidade. Ou seja, na prática, a função testa no máximo os `max_ntry` rótulos com probabilidades maior que 0.01 segundo o modelo.

Em resumo, o pacote `{captchaDownload}` contém toda a parte de *web scraping* utilizada no desenvolvimento da tese. Adicionalmente, o pacote contém funções para orquestrar o *download* automático de Captchas parcialmente rotulados, a partir de um modelo inicial e um oráculo.

Os dados fornecidos pelo pacote ficam tanto na forma de imagens rotuladas quanto na

forma de arquivos de *log*, disponibilizados em arquivos CSV. Para lidar com essa estrutura de dados, mais um pacote foi desenvolvido: o `{captchaOracle}`, definido a seguir.

A.3 Pacote `captchaOracle`

O pacote `{captchaOracle}`, assim como o `{captchaDownload}`, foi desenvolvido para a construção da tese. O pacote, portanto, não apresenta documentação extensiva e suas funções podem não estar com a sintaxe final. Futuramente, o pacote poderá funcionar como novo *backend* para o pacote `{captcha}`, aplicando o WAWL como uma alternativa no fluxo de resolução de Captchas definido na Subseção A.1.3.

O pacote possui quatro funções principais: a `captcha_dataset_oracle()`, a `net_captcha_oracle()`, a `oracle_loss()` e a `captcha_accuracy_oracle()`. Cada função desempenha um papel similar a seus pares do pacote `{captcha}`, mas conseguem lidar com a estrutura de dados fornecida pelo oráculo.

A primeira função a ser utilizada é a `captcha_dataset_oracle()`. Trata-se de uma função similar à `captcha_dataset()` do pacote `{captcha}`, mas com um parâmetro adicional, `path_logs=`, que recebe o caminho dos arquivos de *log*.

A estrutura de dados no caso do oráculo é mais complexa do que no caso canônico. Na resposta, ao invés de guardar uma matriz *one hot* para cada Captcha, é armazenada uma lista com várias matrizes *one hot*, uma para cada tentativa do Captcha. Além disso, é armazenado um vetor `z`, com zeros e uns, informando se algum rótulo está correto ou se todos os rótulos estão incorretos. A variável `z` é construída a partir dos nomes dos arquivos, que contém um `_1` caso o rótulo esteja correto e `_0` caso contrário. Por último, a imagem de entrada é armazenada da mesma forma que na função `captcha_dataset()`.

O módulo `net_captcha_oracle()` faz poucos ajustes à estrutura inicial fornecida pelo módulo `net_captcha()` do pacote `{captcha}`. A única modificação da função é que ela recebe um modelo inicial de entrada, transferindo os pesos ajustados do modelo ao novo módulo. O módulo `net_captcha_oracle()`, inclusive, poderia ser utilizado fora do contexto do WAWL, já que só utiliza os dados de *input*, que não são alterados.

A função `captcha_accuracy_oracle()` é utilizada para estimar a acurácia do modelo. Para isso, a função precisa lidar com o fato de que os dados de validação apresentam uma estrutura diferente dos dados de treino, já que estão completamente classificados. No treino, a acurácia é calculada considerando apenas os casos em que a resposta é conhecida. Na validação, a acurácia é calculada considerando-se todas as observações.

Por último, a função `oracle_loss()` é a que contém a proposta de função de perda do método WAWL. Nos casos corretos, a função de perda é obtida calculando-se uma entropia cruzada simples. Nos casos incorretos, a perda é calculada pela estratégia 1-p, ou seja, considerando o complementar da probabilidade de observar os chutes que foram apresentados segundo o modelo.

Em resumo, o pacote `{captchaOracle}` é o que contém os principais avanços da tese do ponto de vista estatístico. Na prática, é utilizado como *backend* computacional para

ajuste dos modelos que utilizam o oráculo, dentro de um fluxo de trabalho igual ao que é construído para ajuste dos modelos canônicos.

Os códigos para realizar as simulações do modelo foram adicionados na pasta `data-raw` do pacote `{captchaOracle}`. Os códigos foram organizados da seguinte forma:

- `passo_01_*.R`. Contêm os códigos utilizados para ajustar os modelos iniciais. Os códigos são organizados de forma a permitir que vários modelos sejam rodados em paralelo, aproveitando o máximo do poder computacional da máquina utilizada para realizar os ajustes.
- `passo_02_*.R`. Contêm os códigos utilizados para construir as bases de treino e validação para o passo 03. Foi o passo mais demorado da simulação, já que envolveu acessar os sites dos tribunais pela *internet* para obtenção dos Captchas anotados automaticamente. Para realizar a simulação, foram baixados mais de 500.000 Captchas da *internet*.
- `passo_03_*.R`. Contêm os códigos utilizados para ajustar os modelos finais. Os códigos foram organizados de forma similar ao passo 01, mas utilizando as funções desenvolvidas no pacote `{captchaOracle}` para considerar os dados fornecidos pelo oráculo.

Por fim, foi adicionado também um script `report.R`, que monta as bases principais e os resumos dos modelos ajustados. As bases fornecidas pelo último *script* foram adicionadas ao repositório da tese.