



UFOP

Universidade Federal
de Ouro Preto

**Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas**

**Avaliando a eficiência de CAPTCHAs
baseados em texto usando redes
neurais convolucionais**

Saymon Junio Matozinhos

**TRABALHO DE
CONCLUSÃO DE CURSO**

ORIENTAÇÃO:
Prof. Dr. Mateus Ferreira Satler

**Abril, 2021
João Monlevade–MG**

Saymon Junio Matozinhos

Avaliando a eficiência de CAPTCHAs baseados em texto usando redes neurais convolucionais

Orientador: Prof. Dr. Mateus Ferreira Satler

Monografia apresentada ao curso de Engenharia de Computação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina “Trabalho de Conclusão de Curso II”.

Universidade Federal de Ouro Preto

João Monlevade

Abril de 2021

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

M433a Matozinhos, Saymon Junio .

Avaliando a eficiência de CAPTCHAs baseados em texto usando redes neurais convolucionais. [manuscrito] / Saymon Junio Matozinhos. - 2021. 44 f.: il.: color., gráf., tab..

Orientador: Prof. Dr. Mateus Ferreira Satler.

Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Engenharia de Computação .

1. Aprendizado do computador . 2. CAPTCHA (teste de resposta ao desafio). 3. Inteligência artificial. 4. Redes neurais (Computação) . 5. Turing, Teste de. I. Satler, Mateus Ferreira. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.8

Bibliotecário(a) Responsável: Flavia Reis - CRB6-2431



FOLHA DE APROVAÇÃO

Saymon Junio Matozinhos

Avaliando a Eficiência de CAPTCHAs Baseados em Texto usando Redes Neurais Convolucionais

Monografia apresentada ao Curso de Engenharia da Computação da Universidade Federal de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Engenharia da Computação

Aprovada em 29 de abril de 2021

Membros da banca

Doutor - Mateus Ferreira Satler - Orientador (Universidade Federal de Ouro Preto)
Mestre - Bruno Cerqueira Hott - (Universidade Federal de Ouro Preto)
Doutora - Lucineia Souza Maia - (Universidade Federal de Ouro Preto)
Mestre - Victor Hugo Cunha de Melo - (Universidade Federal de Minas Gerais)

Mateus Ferreira Satler, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 01/06/2021



Documento assinado eletronicamente por **Mateus Ferreira Satler, PROFESSOR DE MAGISTERIO SUPERIOR**, em 01/06/2021, às 10:24, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do [Decreto nº 8.539, de 8 de outubro de 2015](#).



A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0164785** e o código CRC **EB3631F7**.

Aos meus pais, que sempre me incentivaram nos momentos mais difíceis.

Agradecimentos

Aos meus pais, por todo apoio e conselhos.

Ao meu orientador Mateus Ferreira Satler, que aceitou me orientar e me auxiliou durante o desenvolvimento deste trabalho.

Aos meus amigos que sempre estiveram ao meu lado, tornando essa caminhada mais prazerosa.

“A verdadeira viagem de descobrimento não consiste em procurar novas paisagens, mas em ter novos olhos.”

— Marcel Proust (1871 – 1922)

Resumo

Captchas baseados em texto são frequentemente utilizados em websites como um mecanismo de proteção contra ataques automatizados. Diversos métodos têm sido propostos para solucioná-los, mas muitos com limitada aplicabilidade. Neste trabalho é proposto um modelo de rede neural convolucional que é simples de implementar e tem baixo custo computacional, para solucionar e avaliar a eficiência de captchas baseados em texto. O modelo foi testado em 7 esquemas de captchas, incluindo os gerados por duas bibliotecas populares e foi capaz de solucionar todos os esquemas com acurácia variando de 86,2% a 100%. O modelo proposto pode ser treinado sem alterar o processo ou os parâmetros de treinamento, e diferentemente de outras abordagens utilizando redes neurais convolucionais, é capaz de solucionar captchas com texto de tamanho variável sem depender de técnicas de segmentação de imagens ou de ser combinado com uma rede neural recorrente. Considerando-se a aplicabilidade do modelo proposto, os resultados que foram obtidos reforçam a ideia de que esse tipo de captcha pode não ser um mecanismo de proteção adequado.

Palavras-chaves: CAPTCHA. Eficiência. Redes Neurais Convolucionais.

Abstract

Text-based captchas are often used on websites as a protection mechanism against automated attacks. Several methods have been proposed to solve them, but many with limited applicability. This work proposes a convolutional neural network model that is simple to implement and has low computational cost, to solve and evaluate the efficiency of text-based captchas. The model was tested on 7 captcha schemes, including those generated by two popular libraries and was able to solve all schemes with accuracy ranging from 86,2% to 100%. The proposed model can be trained without changing the training process or parameters, and unlike other approaches using convolutional neural networks, it is capable of solving captchas with variable text length without relying on image segmentation techniques or being combined with a recurrent neural network. Considering the applicability of the proposed model, the results that have been obtained reinforce the idea that this type of captcha may not be an adequate protection mechanism.

Key-words: CAPTCHA. Efficiency. Convolutional Neural Network.

Lista de ilustrações

Figura 1 – Arquitetura do modelo Deep-CAPTCHA	16
Figura 2 – Interpretação geométrica do SGD	20
Figura 3 – Arquitetura típica de uma CNN	21
Figura 4 – Operação de convolução com múltiplos filtros	22
Figura 5 – Operação de convolução com preenchimento de 1 <i>pixel</i>	23
Figura 6 – A operação de max pooling	24
Figura 7 – (a) Arquitetura do modelo. (b) Um bloco residual. (c) Um bloco residual com subamostragem.	27
Figura 8 – Exemplo de como codificar o texto “47” usando um alfabeto numérico e texto entre 1 e 3 caracteres	28
Figura 9 – Exemplos de captchas gerados pela biblioteca 1 (Python).	33
Figura 10 – Exemplos de captchas gerados pela biblioteca 2 (PHP).	33
Figura 11 – Exemplos de captchas coletados dos websites	34
Figura 12 – Acurácia - Biblioteca 1	36
Figura 13 – Acurácia - Biblioteca 2	36
Figura 14 – Acurácia - Website 1	36
Figura 15 – Acurácia - Website 2	36
Figura 16 – Acurácia - Website 3	36
Figura 17 – Acurácia - Website 4	36

Lista de tabelas

Tabela 1 – Arquitetura do modelo	28
Tabela 2 – Processo de treinamento	35
Tabela 3 – Acurácia do modelo	35
Tabela 4 – Exemplos de captchas do website 1 solucionados incorretamente	37
Tabela 5 – Comparação da acurácia com outros trabalhos	37

Sumário

1	INTRODUÇÃO	13
1.1	O problema	13
1.2	Objetivos	14
1.3	Organização do trabalho	14
2	REVISÃO BIBLIOGRÁFICA	15
2.1	Trabalhos relacionados	15
3	FUNDAMENTAÇÃO TEÓRICA	17
3.1	Redes neurais artificiais	17
3.1.1	Algoritmos de aprendizagem	18
3.1.1.1	Gradiente descendente estocástico	19
3.1.1.2	Adam	20
3.2	Redes neurais convolucionais	21
3.2.1	Camada de convolução	22
3.2.2	Camada totalmente conectada	23
3.2.3	Funções de ativação	23
3.2.4	Max pooling	24
3.2.5	Normalização de lote	25
4	O MODELO	26
4.1	Visão geral	26
4.2	Arquitetura do modelo	26
5	DETALHES DE IMPLEMENTAÇÃO	30
5.1	Tecnologias utilizadas	30
5.1.1	Google Colab	30
5.1.2	Pillow	30
5.1.3	PyTorch	30
5.2	Processo de desenvolvimento	31
6	TESTES E RESULTADOS	33
6.1	Base de dados	33
6.1.1	Bibliotecas	33
6.1.2	Websites	34
6.2	Treinamento	34
6.3	Acurácia do modelo	34

6.4	Análise dos resultados	35
6.5	Comparação com outros trabalhos	37
7	CONCLUSÃO	39
	REFERÊNCIAS	40
	APÊNDICES	42
	APÊNDICE A – IMPLEMENTAÇÃO DE UM BLOCO RESIDUAL EM PYTORCH	43

1 Introdução

Uma necessidade comum em websites é a de garantir que determinadas ações sejam executadas exclusivamente por humanos e não por robôs de forma automatizada. Essa necessidade pode surgir em diversas situações, por exemplo: um website que permite aos seus usuários criar contas de email ou participar de alguma votação *online*, provavelmente deseja que esses usuários sejam humanos, e uma forma de tentar garantir isso é através do uso de CAPTCHAs¹ (acrônimo para “*Completely Automated Public Turing test to tell Computers and Humans Apart*”). Um captcha é um tipo de teste automatizado, utilizado para distinguir humanos de robôs, e embora não exista um teste padrão, todos eles tem a mesma premissa: o teste deve ser simples de ser realizado por humanos, mas difícil de ser realizado por robôs.

1.1 O problema

Entre os vários tipos de captchas, os baseados em texto são um dos mais utilizados (Wang et al., 2020) e seu funcionamento é relativamente simples. Em geral, é apresentado ao usuário uma imagem com algum texto que deve ser reconhecido. E para evitar que o texto seja reconhecido com facilidade, são empregados alguns mecanismos de segurança, como rotação e distorção de caracteres, fonte de texto de tamanho variável e etc. Em seguida, o usuário deve reescrever os caracteres do captcha em uma caixa de texto, finalizando assim o processo. Dada a popularidade dos captchas baseados em texto, é importante saber se eles são realmente eficiente contra ataques automatizados. Para definir o conceito de eficiência do captcha, adotou-se a definição de Bursztein, Martin e Mitchell (2011), que determina que um captcha é eficiente quando não for possível solucioná-lo de forma automatizada, com acurácia maior que 1%. Para os captchas baseados em texto, existem métodos que demonstram que é possível solucioná-los com acurácia maior que 1%, porém, a maioria desses métodos são específicos e funcionam somente para o esquema² de captcha para o qual foram criados. Além disso, os métodos mais recentes que utilizam redes neurais e funcionam com qualquer esquema de captcha, tem limitada aplicabilidade devido a algumas limitações práticas. Como a necessidade de solucionar uma grande quantidade de captchas manualmente ou por terem um custo computacional alto. A limitada aplicabilidade desses métodos, pode ser um dos motivos desse tipo de captcha ainda ser tão popular.

¹ O acrônimo será utilizado em letras minúsculas por questões de legibilidade.

² Esquema é o conjunto de captchas gerados por uma mesma fonte, seja essa fonte uma biblioteca que gera captchas ou uma página de algum website.

1.2 Objetivos

Diante desse cenário, este trabalho propõe um método para solucionar e avaliar a eficiência de captchas baseados em texto. O método proposto pretende ser de simples implementação e de maior aplicabilidade se comparado com abordagens tradicionais e pretende reforçar a ideia de que esse tipo de captcha pode não ser adequado. O método proposto, consiste em construir e treinar um modelo de Rede Neural Convolutiva (CNN - *Convolutional Neural Network*) (LeCun et al., 1989), capaz de solucionar captchas baseados em texto sem utilizar quaisquer outras técnicas de processamento de imagens, como por exemplo segmentação de caracteres.

1.3 Organização do trabalho

O restante deste trabalho é organizado como se segue. No Capítulo 2 são apresentados os principais trabalhos relacionados a solução de captchas. No Capítulo 3 são apresentados os fundamentos teóricos necessários para entender o modelo desenvolvido. O Capítulo 4 apresenta as metodologias empregadas para o desenvolvimento do modelo. Logo após, no Capítulo 5, os detalhes de implementação são explicados, incluindo as tecnologias utilizadas e o processo de desenvolvimento. No Capítulo 6 são apresentados os testes realizados, junto com os resultados e análise dos mesmos. Por fim, o Capítulo 7, contém a conclusão sobre o trabalho.

2 Revisão bibliográfica

Neste capítulo são discutidos alguns métodos para solucionar captchas e suas limitações.

2.1 Trabalhos relacionados

Os primeiros métodos criados para solucionar captchas eram direcionados a esquemas de captchas específicos e, uma vez que um desses esquemas era burlado, era relativamente simples trocar o esquema utilizado. Um dos primeiros métodos foi desenvolvido por [Mori e Malik \(2003\)](#) para solucionar o EZ-Gimpy, um popular esquema de captcha usado pelo Yahoo. A proposta conseguiu solucionar o EZ-Gimpy com uma acurácia de 92%, mas as técnicas que foram utilizadas, entre elas um ataque baseado em dicionário, não podiam ser reaproveitadas com facilidade. Outros métodos que surgiram posteriormente também apresentaram bons valores de acurácia para determinados esquemas, mas não podiam ser facilmente adaptados para outros.

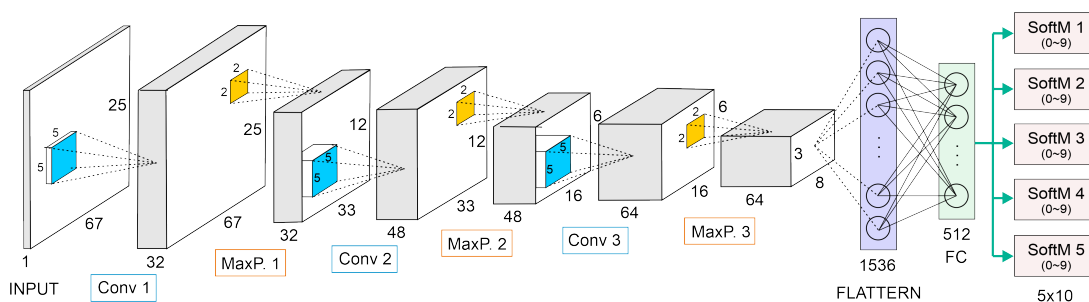
[Bursztein et al. \(2014\)](#) foram um dos primeiros autores a desenvolver um método de solução genérico. Os autores perceberam que muitos dos métodos anteriores usavam uma abordagem semelhante para solucionar captchas, que consistia em duas etapas principais: 1) segmentação da imagem em caracteres e 2) classificação dos caracteres individuais. Usando essa abordagem, a maior dificuldade era criar um algoritmo genérico para realizar a segmentação. Visando superar esse problema, foi desenvolvido um método onde uma rede neural é treinada usando aprendizagem por reforço, para realizar as etapas de segmentação e classificação de forma simultânea. Embora o método seja genérico, ele tem várias limitações práticas. Como a complexidade de implementação, e o custo computacional que aumenta exponencialmente com o comprimento do texto no captcha.

Em [George et al. \(2017\)](#) foi desenvolvido um modelo hierárquico chamado de *Recursive Cortical Network*, que tem a vantagem de não precisar de uma grande quantidade de captchas solucionados, normalmente necessários para treinar uma rede neural. No entanto, o modelo precisa ser treinado em caracteres individuais, gerados por fontes semelhantes às usadas nos captchas. Além disso, os captchas precisam ser pré-processados utilizando um procedimento que necessita ser ajustado para cada esquema de captcha.

Em [Noury e Rezaei \(2020\)](#), assim como no presente trabalho, foi desenvolvido um modelo de CNN para avaliar a eficiência dos captchas gerados por uma biblioteca Python. Esse modelo, chamado de Deep-CAPTCHA, é composto por 5 camadas, sendo três camadas de convolução com *max pooling* e duas camadas totalmente conectadas. Os

captchas processados devem ter dimensões de 67×25 *pixels* na escala de cinza e texto com tamanho fixo de 5 caracteres. Esse modelo foi capaz de obter uma acurácia de 98,94% e 98,31% para captchas utilizando um alfabeto numérico e alfanumérico respectivamente. Mas para alcançar essa acurácia, Deep-CAPTCHA precisou ser treinado com 500.000 captchas, uma quantidade que não é viável de obter em todas as situações.

Figura 1 – Arquitetura do modelo Deep-CAPTCHA



Fonte: Noury e Rezaei (2020)

Wang et al. (2020) resolveram os dois problemas principais nos trabalhos citados anteriormente: 1) o uso de técnicas complexas de processamento de imagens e/ou 2) a necessidade de obter uma grande quantidade de captchas solucionados. Na abordagem utilizada pelos autores, um modelo de rede neural é primeiro treinado em um conjunto de captchas qualquer, e posteriormente retreinado utilizando transferência de aprendizado, para solucionar esquemas de captchas específicos. Apesar dessa ser uma abordagem eficaz, ela ainda tem algumas desvantagens em relação a complexidade de implementação e custo computacional. O modelo utilizado pelos autores é a ResNet-101, que é uma CNN com 101 camadas que tem um custo computacional relativamente alto, principalmente quanto ao uso de memória RAM. Além disso, esse modelo é combinado com uma rede neural recorrente (*Recurrent Neural Network* - RNN) (Hochreiter; Schmidhuber, 1997) para lidar com captchas com texto de tamanho variável, o que torna a implementação mais complexa.

A abordagem empregada nesse trabalho pode ser vista como uma simplificação da utilizada por Wang et al. (2020), por remover a necessidade de utilizar uma RNN e por reduzir o tamanho do modelo de CNN utilizado.

3 Fundamentação teórica

Neste capítulo são apresentados os conceitos teóricos necessários para entender o funcionamento do modelo de rede neural desenvolvido neste trabalho. A Seção 3.1 introduz as redes neurais e discute o seu funcionamento de forma geral. Já a Seção 3.2 apresenta as redes neurais convolucionais e os seus componentes que foram utilizados no modelo desenvolvido.

3.1 Redes neurais artificiais

Tradicionalmente, programas de computadores são escritos por humanos como uma sequência bem definida de passos lógicos e com o objetivo de solucionar determinados problemas. No entanto, para certos tipos de problemas, essa não é uma opção viável. Isso acontece, geralmente, com problemas que humanos conseguem solucionar usando apenas a intuição, porque converter um procedimento intuitivo para um algoritmo não é fácil. Por exemplo, reconhecer texto em captchas é normalmente uma tarefa simples para seres humanos, mas fazer isso de forma consciente, pensado em termos de *pixels* individuais é muito mais difícil.

Redes neurais artificiais ou simplesmente redes neurais são uma das ferramentas na área de aprendizagem de máquina que tem sido utilizada para solucionar problemas dessa natureza. Diferentemente da abordagem tradicional, onde é necessário que um humano descreva de forma explícita quais passos um algoritmo deve seguir, uma rede neural utiliza um conjunto de exemplos (também chamado de base de dados de treinamento), que demonstram como instâncias do problema podem ser solucionadas, e através de um algoritmo de aprendizagem, aprende como solucionar novas instâncias do problema. Essa capacidade de aprender regras gerais de um conjunto limitado de dados é chamada de generalização.

Existem diversos tipos de tarefas que podem ser solucionadas por uma rede neural, sendo uma delas a de classificação. Nesse tipo de tarefa, o objetivo é construir um modelo que recebe uma entrada e retorna como saída as probabilidades dessa entrada pertencer a um conjunto de possíveis categorias. Por exemplo, a entrada pode ser uma imagem, e as categorias os caracteres do alfabeto.

De maneira formal, uma rede neural é uma função $f(x; \theta) = y$ que transforma uma entrada x para uma saída y , onde θ é o conjunto de parâmetros da função. A base de dados de treinamento é um conjunto de exemplos formado por pares na forma $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$, onde $y^{(i)}$ é a saída esperada (algumas vezes também chamada

de rótulo) para a entrada $x^{(i)}$. E treinar uma rede neural, significa utilizar um algoritmo de aprendizagem para encontrar o conjunto de parâmetros θ , que façam o modelo atender alguma métrica de desempenho. Como o objetivo é alcançar a generalização, o desempenho é avaliado em um conjunto de dados (base de dados de teste) diferente do utilizado durante o treinamento e a métrica utilizada depende do problema sendo solucionado. Em problemas de classificação, é comum usar a **acurácia**, que é a proporção de exemplos para os quais o modelo retorna a previsão correta.

A forma exata que a função representando a rede neural irá assumir, vai depender do tipo de rede neural, mas em geral, $f(x; \theta)$ será composta de outras funções formando uma hierarquia $f(x; \theta) = f_L(\dots f_2(f_1(x_1; \theta_1); \theta_2)\dots); \theta_3$). Nesse contexto, cada função é referida como uma camada, sendo a última chamada de **camada de saída** e todas as outras tem o nome de **camadas ocultas** (Goodfellow; Bengio; Courville, 2016, p. 168–169). Uma rede neural com mais do que uma camada é comumente chamada de rede neural profunda. Cada camada em uma rede neural possui uma entrada, uma saída e o seu próprio conjunto de parâmetros treináveis. Além dos parâmetros treináveis, uma camada pode possuir parâmetros que devem ser escolhidos, geralmente através de tentativa e erro, também chamados de **hiperparâmetros**.

Construir e treinar um modelo de rede neural envolve basicamente coletar uma base de dados, definir a arquitetura do modelo, ou seja, quais camadas serão utilizadas e como serão configuradas, escolher uma métrica de desempenho e um algoritmo de aprendizagem.

3.1.1 Algoritmos de aprendizagem

Uma rede neural é treinada com o objetivo de encontrar o conjunto de parâmetros que resultam no seu melhor desempenho, quando avaliado em dados não utilizados durante o treinamento. De outra forma, o objetivo é reduzir o erro de generalização. No entanto, isso é realizado de forma indireta, usando a base de dados de treinamento $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$. Uma forma de medir o erro de generalização é através de uma função de erro ou função de custo (Goodfellow; Bengio; Courville, 2016, p. 275-276) que penaliza o modelo por cada previsão errada. Então o custo na base de dados pode ser dado por:

$$C(\theta) = \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}; \theta), y^{(i)}), \quad (3.1)$$

onde L é a função de custo, $f(x^{(i)}; \theta)$ e $y^{(i)}$ são respectivamente a saída prevista e a saída correta quando $x^{(i)}$ é a entrada. Idealmente, a função de custo L deve ser igual a 0 quando a saída prevista e a saída correta são iguais e deve aumentar se a diferença entre os valores de saída aumentam. Dessa forma, o objetivo do algoritmo de aprendizagem pode ser

formulado como um problema de otimização: encontrar o conjunto de parâmetros θ^* que minimizam o valor da função de custo

$$\theta^* = \arg \min \frac{1}{n} \sum_{i=1}^n L(f(x^{(i)}; \theta), y^{(i)}). \quad (3.2)$$

Existem diversas escolhas possíveis para a função L e uma das mais utilizadas é a **entropia cruzada**. Minimizar a entropia cruzada é o equivalente a minimizar a diferença entre a distribuição de probabilidade real, definida pelo conjunto de treinamento e a distribuição de probabilidade retornada pelo modelo (Goodfellow; Bengio; Courville, 2016, p. 132). Seja $\hat{y} = f(x; \theta)$ o vetor representando a distribuição de probabilidade prevista pelo modelo e $y \in \mathfrak{R}^k$ a distribuição de probabilidade correta, a função de entropia cruzada é definida da seguinte forma:

$$L(y, \hat{y}) = -\frac{1}{k} \sum_{i=1}^k y_i \times \ln(\hat{y}_i) + (1 - y_i) \times \ln(1 - \hat{y}_i). \quad (3.3)$$

3.1.1.1 Gradiente descendente estocástico

O Gradiente Descendente Estocástico (*Stochastic gradient descent* - SGD) (Robbins; Monro, 1951) é um algoritmo de otimização iterativo utilizado para encontrar o ponto mínimo de uma função, usando informações do seu gradiente. Considere a função custo $C(\theta)$ e o seu gradiente $\nabla_{\theta} C(\theta)$. É possível afirmar que $C(\theta - \epsilon \nabla_{\theta} C(\theta)) < C(\theta)$ para um valor suficientemente pequeno de ϵ . Assim pode-se reduzir $C(\theta)$ por mover θ em pequenos passos na direção negativa do gradiente.

Computar o gradiente $\nabla_{\theta} C(\theta)$ em todo o conjunto de treinamento pode ser um processo computacionalmente caro, por isso, o SGD usa uma estimativa do gradiente calculada em uma amostra (*minibatch*) aleatória da base de dados de treinamento $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$. Então em cada iteração (**época**) do algoritmo essa estimativa é dada por

$$g = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \quad (3.4)$$

e o valor de θ é atualizado para

$$\theta' = \theta - \epsilon g \quad (3.5)$$

onde $0 < \epsilon < \infty$ é a **taxa de aprendizado**. Uma taxa de aprendizado maior pode fazer o algoritmo convergir mais rápido, caso as direções indicadas pelo gradiente não mudem muito. O SGD e suas variações não são garantidos para encontrar o ponto mínimo de uma

Algorithm 1 Gradient Descendente Estocástico

```

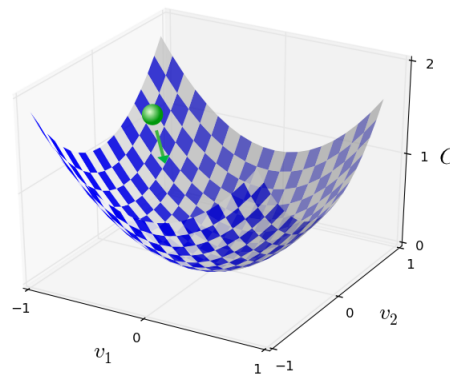
1: input Taxa de aprendizado  $\epsilon$ .
2: input Valor inicial de  $\theta$ .
3: while critério de parada não foi alcançado do
4:   Amostra um minibatch de  $m$  exemplos do conjunto de treinamento
      $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$ .
5:   Calcula uma estimativa do gradiente:  $g \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)})$ 
6:   Atualiza o valor do conjunto de parâmetros:  $\theta \leftarrow \theta - \epsilon g$ 
7: end while

```

função, então é necessário definir algum critério de parada, como um número máximo de épocas ou uma taxa de erro mínima. O pseudocódigo do SGD é mostrado em algoritmo 1.

Intuitivamente, é possível visualizar o comportamento do SGD como o de um objeto sem massa, movendo-se numa superfície na direção mais íngreme de sua vinhança, sempre a passos fixos, com o objetivo de encontrar o seu ponto mais baixo. Um dos problemas com essa abordagem, é que dependendo da curvatura dessa superfície a convergência do algoritmo pode ser muito lenta.

Figura 2 – Interpretação geométrica do SGD



Fonte: Nielsen (2015)

3.1.1.2 Adam

O algoritmo Adam (Kingma; Ba, 2014), mostrado em algoritmo 2, é um algoritmo semelhante ao SGD, no sentido que também usa informações do gradiente. O grande diferencial desse algoritmo é o uso de uma taxa de aprendizado por parâmetro, ajustada de forma automática durante o treinamento. Atualmente, o Adam é o algoritmo recomendado para treinar redes neurais, porque geralmente converge mais rápido e é mais robusto a escolha de hiperparâmetros.

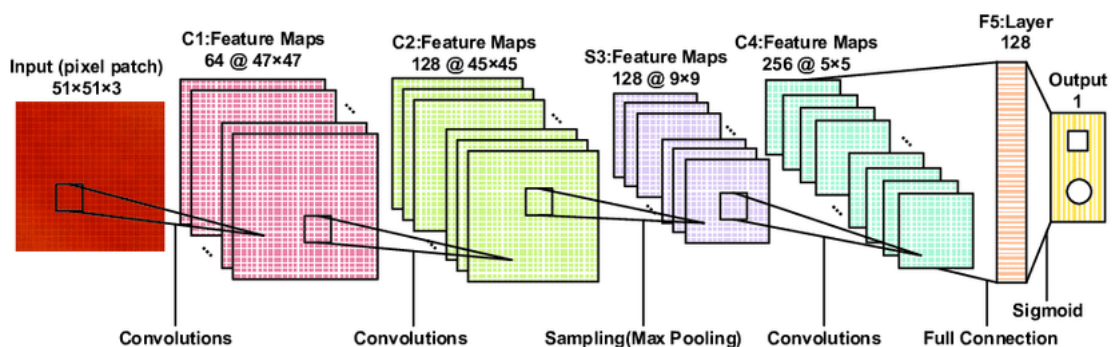
Algorithm 2 Algoritmo Adam

- 1: **input** Taxa de aprendizado ϵ .
- 2: **input** Taxas de decaimento exponencial β_1 e β_2 no intervalo $[0, 1)$.
- 3: **input** Pequena constante δ usada para evitar divisão por 0.
- 4: **input** Valor inicial de θ .
- 5: $m_0 \leftarrow 0$
- 6: $v_0 \leftarrow 0$
- 7: $t \leftarrow 0$
- 8: **while** critério de parada não foi alcançado **do**
- 9: Amostra um *minibatch* de m exemplos do conjunto de treinamento $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$.
- 10: Calcula uma estimativa do gradiente: $g_t \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)})$
- 11: $t \leftarrow t + 1$
- 12: $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$
- 13: $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$
- 14: $\hat{m}_t \leftarrow \frac{m_t}{1 - \beta_1^t}$
- 15: $\hat{v}_t \leftarrow \frac{v_t}{1 - \beta_2^t}$
- 16: Atualiza o valor do conjunto de parâmetros: $\theta_t \leftarrow \theta_{t-1} - \epsilon \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \delta}}$
- 17: **end while**

3.2 Redes neurais convolucionais

Uma rede neural convolucional (CNN - *Convolutional Neural Network*) (LeCun et al., 1989) é um tipo de rede neural com uma arquitetura especializada no processamento de imagens. Uma CNN é caracterizada pelo uso de **camadas de convolução** e adicionalmente pelo uso de **camadas de pooling**. Essas camadas realizam operações que exploram certas propriedades estatísticas encontradas em imagens, para realizar algumas operações de forma mais eficiente. Esta abordagem de utilizar camadas especializadas é bastante eficaz, e atualmente, CNNs são utilizadas para solucionar diversos problemas na área de visão computacional, como classificação, segmentação e detecção de objetos em imagens (He et al., 2017).

Figura 3 – Arquitetura típica de uma CNN



Fonte: Guo et al. (2016)

3.2.1 Camada de convolução

Uma camada de convolução é formada por um conjunto de filtros, sendo que cada filtro é utilizado para detectar quais características estão presentes nos dados de entrada, através da operação de convolução. As características que um filtro detecta são aprendidas no processo de treinamento da rede neural, e nas primeiras camadas de uma CNN, os filtros geralmente aprendem a detectar padrões simples, como bordas (Krizhevsky; Sutskever; Hinton, 2012). A saída de uma camada de convolução com um único filtro é chamada de mapa de características ou mapa de ativação e é dada por:

$$S(i, j) = \sigma((I \star K)(i, j) + b), \tag{3.6}$$

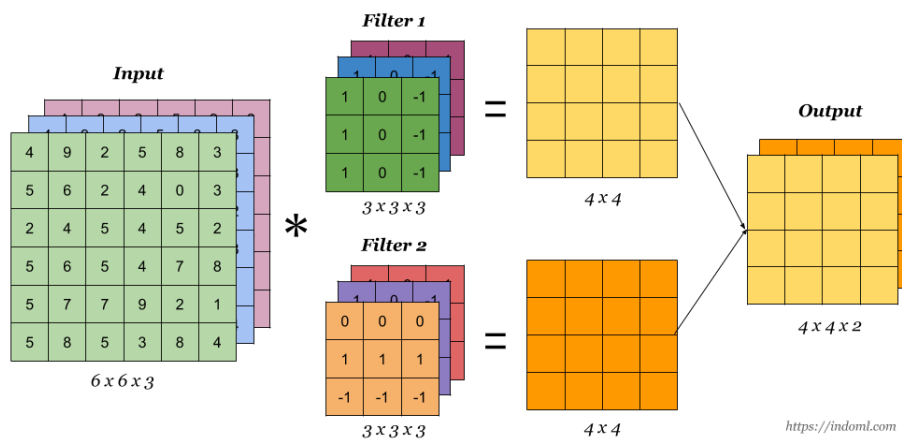
onde σ é uma função de ativação, $I \in \mathbb{R}^{h \times w \times c}$ é o tensor¹ de entrada, $K \in \mathbb{R}^{r \times s \times c}$ é o kernel ou filtro, $b \in R$ o viés e $I \star K$ é a operação de convolução definida assim:

$$(I \star K)(i, j) = \sum_{c=1}^c \sum_{m=1}^r \sum_{n=1}^s I(i + m - 1, j + n - 1, c)K(m, n, c). \tag{3.7}$$

Observe que c , o número de canais do filtro K , deve ser igual ao número de canais do tensor I .

A Equação 3.6 é o exemplo mais simples de uma camada de convolução, porque é utilizado um único filtro. Quando mais de um filtro é utilizado, todos devem ter as mesmas dimensões espaciais. E a saída vai ser um tensor 3D, resultado de empilhar os mapas de características gerados por cada filtro. Dessa forma, para o caso mais geral, a saída da camada de convolução será um tensor $S \in \mathbb{R}^{h_1 \times w_1 \times f}$, considerando-se o uso de f filtros $K^{r \times s \times c}$.

Figura 4 – Operação de convolução com múltiplos filtros

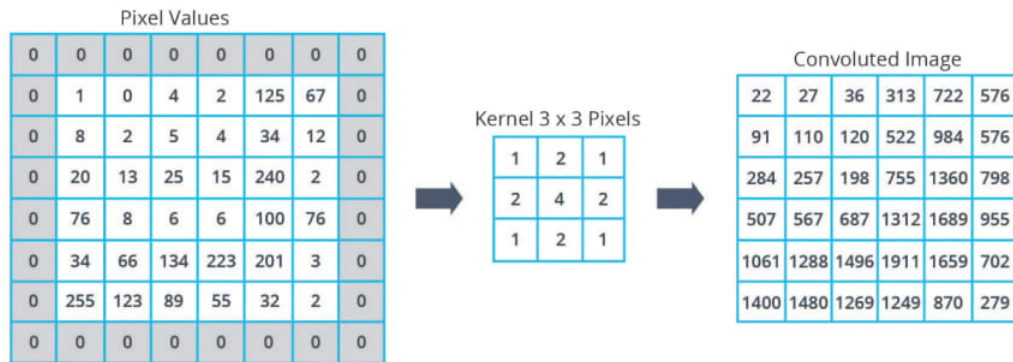


Fonte: IndoML (2018)

¹ Um tensor é um objeto matemático que generaliza vetores e matrizes para um número arbitrário de dimensões. Um vetor é um tensor 1D e uma matriz é um tensor 2D.

A operação de convolução reduz as dimensões do tensor de entrada I , de modo que $w_1 = w - r + 1$ e $h_1 = h - s + 1$. Algumas vezes essa redução de dimensão pode não ser desejável, pois ela limita o número de camadas de convolução que podem ser utilizadas sucessivamente. Para contornar esse problema, utiliza-se a operação de convolução com preenchimento, basicamente, para um preenchimento de tamanho p pixels, as dimensões da entrada se tornam $w + p$ e $h + p$, sendo os elementos adicionados iguais a zero.

Figura 5 – Operação de convolução com preenchimento de 1 pixel



Fonte: Sandeep Balachandran (2020)

3.2.2 Camada totalmente conectada

Uma camada totalmente conectada é formada por um conjunto de unidades chamadas de neurônios, sendo que todos os neurônios estão “conectados” a todos os elementos da entrada. Conectado, nesse contexto, significa que para cada neurônio i , o seu valor de saída $y_i = \sigma(x \cdot w^{(i)} + b_i)$ depende de todos os elementos da entrada x . A saída de uma camada totalmente conectada é a saída combinada de todos os neurônios, dada por:

$$y = \sigma(xW + b), \tag{3.8}$$

onde σ é uma função de ativação, $x \in \mathbb{R}^{1 \times p}$ é a entrada, $W \in \mathbb{R}^{p \times q}$ e $b \in \mathbb{R}^{1 \times q}$ são os parâmetros da camada, sendo a i -ésima coluna de W o vetor $(w^{(i)})^T$ e o i -ésimo elemento de b o escalar b_i .

Esse tipo de camada é geralmente a última em uma CNN, porque quando combinada com uma função de ativação adequada, pode ser usada para representar em sua saída uma distribuição de probabilidade.

3.2.3 Funções de ativação

Uma rede neural composta somente de camadas de convolução ou camadas totalmente conectadas, é capaz de aproximar uma grande variedade de funções realísticas,

incluindo todas as funções contínuas, desde que seja utilizada uma função de ativação adequada (Leshno et al., 1993). Uma dessas funções é a unidade linear retificada (ReLU - *Rectified Linear Unit*), normalmente utilizada em camadas ocultas.

$$\text{ReLU}(x) = \max\{0, x\} \quad (3.9)$$

Uma outra função de ativação bastante utilizada é a sigmoide, essa função é útil quando deseja-se prever valores independentes de probabilidade na camada de saída de uma rede neural.

$$\text{sigmoide}(x) = \frac{1}{1 + \exp(-x)} \quad (3.10)$$

Nas duas equações acima, x é um tensor e a função de ativação é aplicada elemento a elemento.

3.2.4 Max pooling

A camada de *max pooling* é utilizada após uma camada de convolução e tem dois propósitos: reduzir as dimensões da saída da camada de convolução e indicar que a camada de convolução deve aprender uma função invariante a determinados tipos de distorções geométricas (Goodfellow; Bengio; Courville, 2016, p. 342). A operação de *max pooling* faz uma subamostragem de cada mapa de ativação selecionando o elemento de maior valor dentro de uma região retangular (filtro). Essa camada tem como único hiperparâmetro as dimensões do filtro e não tem parâmetros treináveis. A Figura 6 mostra o resultado da operação de *max pooling* em um mapa de ativação com dimensões 4×4 usando um filtro com dimensões 2×2 .

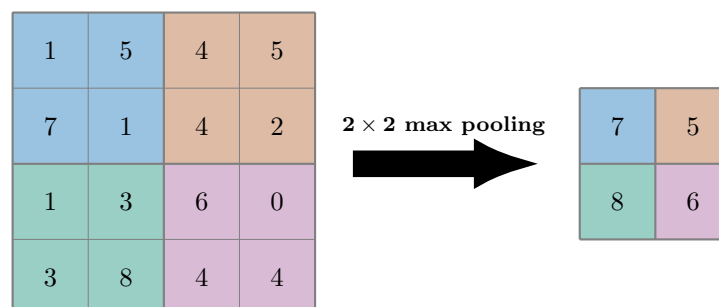


Figura 6 – A operação de max pooling

3.2.5 Normalização de lote

Em uma rede neural a entrada de uma camada depende dos parâmetros de todas as camadas anteriores, por isso, durante o treinamento, pequenas variações nesses parâmetros podem causar perturbações significativas nas entradas das camadas seguintes, tornando mais difícil o treinamento de modelos profundos. A camada de normalização de lote (Ioffe; Szegedy, 2015) pode ser utilizada para minimizar esse problema. Deixe $H \in \mathfrak{R}^{m \times h \times w \times c}$ ser um *minibatch* de m ativações que deve ser normalizado. A transformação definida pela camada de normalização de lote é dada por:

$$Y = H'\gamma + \beta, \quad (3.11)$$

onde $\gamma \in \mathfrak{R}^c$ e $\beta \in \mathfrak{R}^c$ são parâmetros que devem ser aprendidos e são inicializados com uns e zeros respectivamente. Y é o resultado de multiplicar e somar cada canal c de H' elemento a elemento por y_c e β_c , sendo

$$H' = \frac{H - \mu}{\sqrt{\sigma^2}} \quad (3.12)$$

o *minibatch* transformado de modo que todos os canais tenham média 0 e variância 1, onde $\mu \in \mathfrak{R}^c$ é um vetor de médias por canal, definido assim:

$$\mu = \frac{1}{m} \sum_{i=1}^m H_i \quad (3.13)$$

e $\sigma^2 \in \mathfrak{R}^c$ é um vetor de variâncias por canal, definido assim:

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (H_i - \mu)^2 \quad (3.14)$$

Uma vez que o modelo foi treinado μ e σ^2 são substituídos pela média desses valores calculadas durante o treinamento do modelo.

4 O modelo

Este capítulo apresenta as metodologias empregadas para o desenvolvimento do modelo de rede neural convolucional que foi utilizado para solucionar e avaliar os captchas baseados em texto. A Seção 4.1 descreve o funcionamento do modelo de forma geral e a Seção 4.2 descreve a arquitetura do modelo e o seu funcionamento de forma detalhada.

4.1 Visão geral

Conceitualmente, o modelo desenvolvido é simples: ele recebe como entrada os *pixels* de uma imagem RGB e retorna como saída os caracteres na imagem. Esse modelo é um método genérico para solucionar captchas, no sentido que pode ser treinado para solucionar diferentes esquemas de captchas, sem a necessidade de alterar o processo ou os parâmetros de treinamento. Treinar o modelo requer então apenas um conjunto pequeno de captchas solucionados, devido ao uso da técnica de transferência de aprendizado.

O modelo foi desenvolvido para solucionar captchas com dimensões de 160 x 64 *pixels* e com texto de tamanho variável entre 4 e 6 caracteres, sendo reconhecidos 62 caracteres diferentes (26 letras maiúsculas, 26 letras minúsculas e 10 dígitos numéricos). Além disso o modelo pode ser treinado para solucionar captchas que empregam diferentes mecanismos de segurança, como imagens com ruídos, caracteres distorcidos e etc.

4.2 Arquitetura do modelo

Desenvolver um modelo de rede neural pode ser difícil porque geralmente não se sabe que tipo de arquitetura irá funcionar. Por isso, o processo de desenvolvimento consiste de tentativa e erro, onde a intuição é utilizada para decidir quais camadas e hiperparâmetros poderiam funcionar melhor, e então, o modelo deve ser treinado para saber se o seu desempenho será satisfatório. Neste trabalho esse procedimento foi seguido e foram testadas diversas arquiteturas resultando na que é mostrada na Figura 7.

O componente básico utilizado na construção do modelo é o bloco residual (He et al., 2015). Um bloco residual possui dois “ramos” que processam uma determinada entrada de forma independente produzindo duas saídas intermediárias que depois são somadas resultando na saída final. Esse tipo de bloco ajuda a mitigar um problema que ocorre no treinamento de modelos profundos, conhecido como “*vanishing gradient problem*” (Bengio; Simard; Frasconi, 1994). Dois tipos de blocos residuais foram utilizados, blocos residuais com e sem subamostragem, sendo a única diferença entre eles o uso da operação de *max*

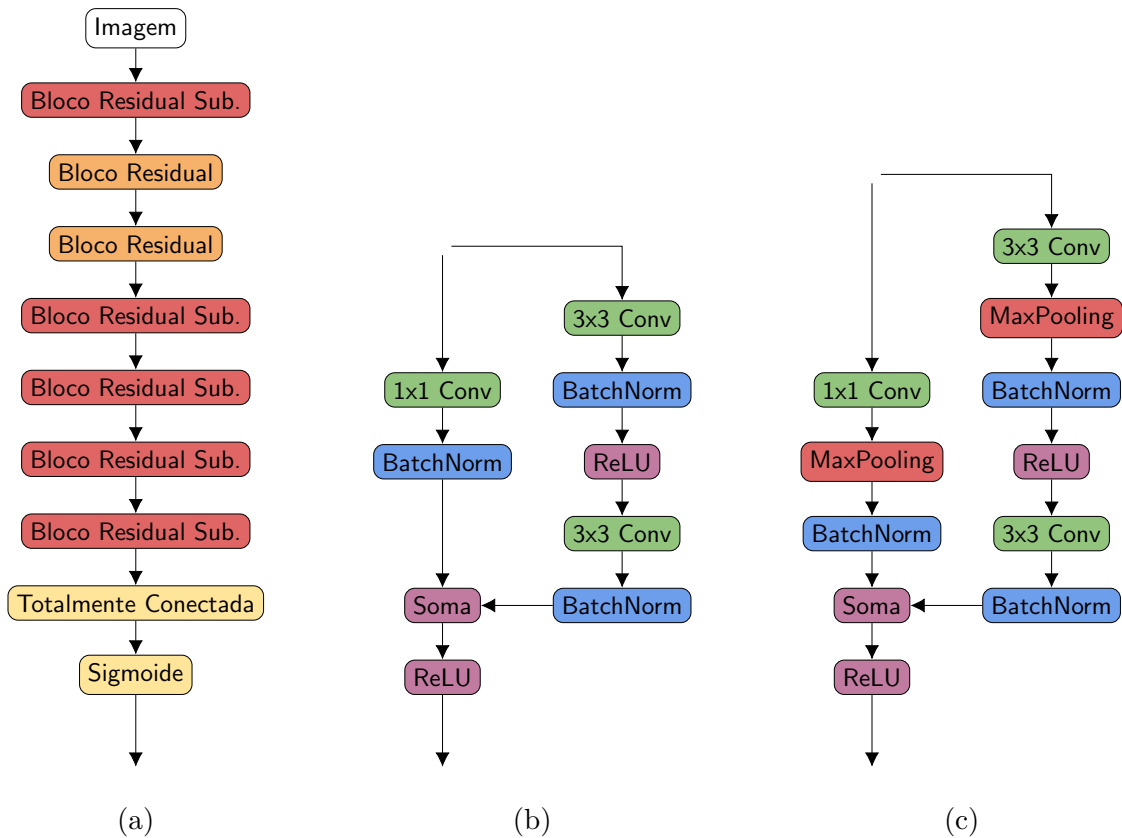


Figura 7 – (a) Arquitetura do modelo. (b) Um bloco residual. (c) Um bloco residual com subamostragem.

pooling. Em um bloco residual com subamostragem a operação de *max pooling* reduz as dimensões da entrada pela metade, enquanto em um bloco residual sem subamostragem as dimensões da entrada não são alteradas.

Nos blocos residuais utilizados, as camadas de convolução com filtro 3x3 têm preenchimento de 1 *pixel* e todas as três camadas utilizam o mesmo número de filtros, o número exato é mostrado na Tabela 1. Também é interessante observar nessa tabela a quantidade de parâmetros em cada camada. O número de parâmetros em cada bloco residual é a soma dos parâmetros das três camadas de convolução, e apesar disso, somente o bloco 6 tem mais parâmetros do que a camada totalmente conectada. Inicialmente, foram utilizados no bloco 7, 128 filtros de convolução, resultando em um desempenho em média 2% melhor, mas posteriormente optou-se por utilizar somente 64 filtros, para reduzir o número de parâmetros do modelo.

Na última camada do modelo foi utilizada uma camada totalmente conectada com $62 \times 6 + 3 = 375$ neurônios com a função de ativação sigmoide. A saída dessa camada é um vetor \hat{y} contendo a previsão para 6 caracteres e o tamanho do texto. A cada 62 elementos desse vetor está codificada a previsão de um caractere, logo o i -ésimo ($1 < i < 6$) caractere previsto é dado pela posição do elemento com maior valor no intervalo

Tabela 1 – Arquitetura do modelo

#	Camada	Filtros	Entrada	Saída	Parâmetros
1	Bloco Residual Sub.	32	$160 \times 64 \times 3$	$80 \times 32 \times 32$	1.920
2	Bloco Residual	64	$80 \times 32 \times 32$	$80 \times 32 \times 64$	39.104
3	Bloco Residual	64	$80 \times 32 \times 64$	$80 \times 32 \times 64$	78.016
4	Bloco Residual Sub.	64	$80 \times 32 \times 64$	$40 \times 16 \times 64$	78.016
5	Bloco Residual Sub.	128	$40 \times 16 \times 64$	$20 \times 8 \times 128$	156.032
6	Bloco Residual Sub.	128	$20 \times 8 \times 128$	$10 \times 4 \times 128$	311.680
7	Bloco Residual Sub.	64	$10 \times 4 \times 64$	$5 \times 2 \times 64$	78.016
8	Totalmente Conectada	-	640	375	240.000

$[(i - 1) \times 62 + 1, i \times 62]$. O tamanho do texto está codificado nos últimos 3 elementos do vetor e será 4, 5 ou 6 dependendo se o antepenúltimo, penúltimo ou último elemento respectivamente do vetor é o maior. Para alcançar isso, foi atribuído um rótulo na forma de um vetor y para cada captcha na base de dados de treinamento representando a saída esperada do modelo. Na Figura abaixo é mostrado um exemplo simplificado de como essa codificação foi realizada.

1	2	3	4	5	6	7	8	9	10
0	0	0	0	1	0	0	0	0	0
11	12	13	14	15	16	17	18	19	20
0	0	0	0	0	0	0	1	0	0
21	22	23	24	25	26	27	28	29	30
0	0	0	0	0	0	0	0	0	0
31	32	33							
0	1	0							

Figura 8 – Exemplo de como codificar o texto “47” usando um alfabeto numérico e texto entre 1 e 3 caracteres

Para comparar a saída correta, como mostrado na Figura 8, com a saída prevista pelo modelo, foi utilizada a função de custo mostrada abaixo:

$$L(y, \hat{y}) = -\frac{1}{z} \sum_{i=1}^z \begin{cases} 0 & t \times 62 < i \leq 372 \\ y_i \times \ln(\hat{y}_i) + (1 - y_i) \times \ln(1 - \hat{y}_i) & \text{caso contrário} \end{cases} \quad (4.1)$$

onde $z = 375$. Essa função de custo é a entropia cruzada aplicada seletivamente na saída do modelo. Então durante o treinamento essa função foi otimizada utilizando o algoritmo Adam com $\beta_1 = 0,9$, $\beta_2 = 0,999$ e taxa de aprendizado igual a 0,001.

Um ponto positivo da arquitetura mostrada na Figura 7 é a sua simplicidade de implementação, já que todos os componentes utilizados podem ser facilmente encontrados em *frameworks* para aprendizado de máquina.

5 Detalhes de implementação

Neste capítulo são apresentados os detalhes de implementação do modelo. Na Seção 5.1 são apresentadas as tecnologias utilizadas para implementá-lo e na Seção 5.2 o processo de desenvolvimento.

5.1 Tecnologias utilizadas

Para desenvolver o modelo foi utilizado a linguagem Python¹ que é conhecida por sua simplicidade de uso e é suportada pelos principais sistemas operacionais. Essa linguagem foi escolhida por possuir um vasto ecossistema de bibliotecas e *frameworks* para as mais diversas áreas, como a biblioteca para processamento de imagens Pillow² e o *framework* para aprendizado de máquina PyTorch³. Além disso é suportada pelo Google Colab.

5.1.1 Google Colab

Treinar uma rede neural é geralmente um processo computacionalmente caro, que é melhor executado em hardware especializado, como GPUs. Sendo assim, as etapas deste trabalho que precisaram de maiores recursos computacionais foram realizadas na plataforma do Google Colab. Essa plataforma, entre outras funcionalidades, permite executar código Python em um ambiente com uma grande quantidade de memória RAM e com uma GPU de alto desempenho.

5.1.2 Pillow

Pillow é uma biblioteca Python para processamento de imagens. Essa biblioteca foi necessária para realizar duas tarefas simples. Primeiro, foi utilizada para carregar imagens do disco rígido que estavam salvas em formato compactado e convertê-las para um tensor 3D de *pixels*. E segundo, para redimensionar os captchas que foram coletados dos websites, uma vez que o modelo foi construído para imagens com dimensões de 160 x 64 *pixels*.

5.1.3 PyTorch

PyTorch é um *framework* para a linguagem Python que facilita o desenvolvimento de aplicações na área de aprendizado de máquina. A sua API de baixo nível é composta

¹ <https://www.python.org/>

² <https://pypi.org/project/Pillow/>

³ <https://pytorch.org/>

basicamente por um tipo de dados básico chamado de **Tensor** (vetor multidimensional) e sobre operações que podem ser realizadas sobre ele. As operações realizadas em um *Tensor* podem ser executadas na CPU ou GPU e PyTorch fornece uma interface simples para mover dados entre estes dispositivos.

PyTorch também possui uma API de alto nível que fornece todos os componentes necessários para construir e treinar modelos de redes neurais. Entre esses componentes é possível encontrar implementações de camadas de convolução, camadas totalmente conectadas, algoritmos de otimização e funções de custo. Além disso, também fornece abstrações para realizar tarefas comuns, como carregar dados do disco rígido e executar código paralelizado de forma totalmente transparente ao usuário.

5.2 Processo de desenvolvimento

Usando a API de alto nível do *framework* PyTorch, o processo de desenvolvimento consiste basicamente em estender algumas classes abstratas através do mecanismo de herança da linguagem Python. Essas classes e outros componentes da API que foram utilizados são descritos a seguir.

PyTorch tem duas classes principais para manipular dados: *Dataset* e *DataLoader* que podem ser encontradas no módulo *torch.utils.data*. A classe *Dataset* representa uma abstração da base de dados e foi estendida implementando dois métodos: o primeiro, retorna a quantidade de exemplos na base de dados, o segundo recebe como parâmetro um índice e retorna um exemplo específico, como um par de tensores (x, y) , sendo x os *pixels* da imagem e y o rótulo (construído como especificado na Seção 4.2). A classe *DataLoader* encapsula um objeto do tipo *Dataset* em um iterável, permitindo configurar o tamanho m do *minibatch* que será usado ao acessar o *DataLoader* dentro de um laço de repetição. De outra forma, um objeto retornado pelo *DataLoader* dentro de um laço *for*, será um conjunto de m exemplos (x, y) aleatórios da base de dados.

Um modelo ou camadas de um modelo em PyTorch são classes derivadas de *torch.nn.Module*. Instâncias dessa classe comportam-se como funções Python e podem ser chamadas passando um *Tensor* como parâmetro e também irão retornar como saída um *Tensor*. Essa classe foi utilizada para implementar os blocos residuais e o modelo final. Ao estendê-la para criar uma nova camada, deve-se definir no seu construtor quais são as outras camadas que ela irá utilizar (camada de convolução, *max pooling* e etc) e implementar o método *forward* que especifica como os dados de entrada serão processados. No Apêndice A é mostrado como essa classe foi estendida para implementar um bloco residual.

Por fim, para treinar o modelo foi implementado o laço (*loop*) de treinamento. Em cada repetição (época) do laço de treinamento, exemplos da base de dados são carregados

para a memória RAM através do *DataLoader* e depois movidos para a memória da GPU onde as operações são realizadas. Com os dados na GPU o modelo é utilizado para fazer previsões sobre todo o *minibatch* e o erro sobre as previsões é calculado usando a função de custo *torch.nn.functionals.binary_cross_entropy*. Os parâmetros do modelo são então ajustados com o objetivo de reduzir o erro, utilizando o otimizador *torch.optim.Adam*. Ainda dentro do laço de treinamento, a acurácia do modelo é calculada no conjunto de treinamento e teste, e caso essa seja a maior acurácia obtida até o momento, os parâmetros do modelo são salvos em disco. PyTorch fornece as funções *torch.save* e *torch.load* para salvar e carregar instâncias da classe *torch.nn.Module*.

6 Testes e Resultados

Neste capítulo é descrito como os captchas foram coletados, os testes realizados e os resultados obtidos.

6.1 Base de dados

Para construir a base de dados foram utilizadas duas bibliotecas geradoras de captchas (uma¹ escrita em Python e outra² em PHP) e também foram coletados captchas de 4 websites. Utilizar bibliotecas geradoras de captchas, ao invés de apenas coletar captchas de websites, tem as seguintes vantagens:

- Permite gerar uma quantidade arbitrária de captchas para treinar o modelo.
- Facilita reproduzir os resultados deste trabalho.
- Como essas bibliotecas são populares, é provável que sejam utilizadas por vários websites.

6.1.1 Bibliotecas

Para cada biblioteca foram criadas duas bases de dados, uma base de treinamento e uma base de teste, com 150.000 e 1000 captchas respectivamente. Todos os captchas foram gerados com dimensões de 160 x 64 *pixels* e com texto variando em tamanho entre 4 e 6 caracteres, usando um alfabeto contendo letras maiúsculas, minúsculas e números. Algumas bibliotecas permitem configurar alguns parâmetros para gerar os captchas, como nível de ruído e/ou fonte de texto. Foram usados os valores padrões de todos os parâmetros configuráveis. Nas figuras abaixo são mostrados exemplos dos captchas gerados.



Figura 9 – Exemplos de captchas gerados pela biblioteca 1 (Python).



Figura 10 – Exemplos de captchas gerados pela biblioteca 2 (PHP).

¹ <https://pypi.org/project/captcha/>

² <https://github.com/Gregwar/Captcha>

6.1.2 Websites

De forma semelhante, para cada website foram criadas duas bases de dados, uma base de treinamento e uma base de teste, com 2000 e 1000 captchas respectivamente. Todos os captchas foram redimensionados para ter dimensões de 160 x 64 *pixels*. Os captchas foram solucionados preenchendo os “formulários” em cada website para garantir a solução correta.

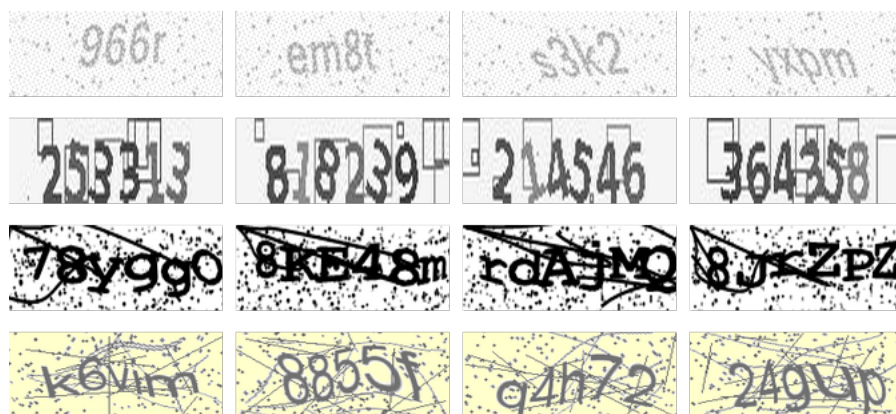


Figura 11 – Exemplos de captchas coletados dos websites

6.2 Treinamento

Os testes realizados consistiram em treinar o modelo para solucionar cada esquema de captcha e avaliar a sua acurácia nas bases de dados de teste. O ambiente de testes foi o Google Colab com uma GPU Tesla P100-16GB.

O processo de treinamento foi dividido em duas etapas. Primeiro, o modelo foi treinado para solucionar os captchas gerados pelas bibliotecas. Essa foi a etapa mais demorada, devido a grande quantidade de captchas utilizados. Na segunda etapa, o modelo foi treinado para solucionar os captchas coletados dos websites, utilizando transferência de aprendizado. As duas possibilidades foram testadas, ou seja, treinou-se o modelo utilizando-se como base ambos os modelos já treinados na etapa 1. Em todas as situações, ao utilizar o modelo treinado para a biblioteca 2, a acurácia alcançada foi maior. A Tabela 2 a seguir sumariza o processo de treinamento.

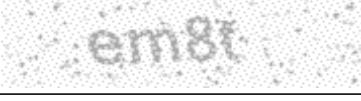
6.3 Acurácia do modelo

A Tabela 3 mostra a acurácia obtida nas bases de dados de teste para cada esquema de captcha. E nas Figuras 12 a 17 são mostrados a acurácia na base de treinamento e teste em cada época. O tempo de inferência, que é o tempo necessário para solucionar um captcha foi medido em um hardware mais modesto. O tempo de inferência em uma GPU GeForce GTX 1050 Ti é de 0,01 segundos e em uma CPU Intel Core i7 é de 0,05 segundos.

Tabela 2 – Processo de treinamento

Esquema	Base de treinamento	Base de teste	Minibatch	Épocas	Tempo
Biblioteca 1	150.000	1000	128	65	4,9 horas
Biblioteca 2	150.000	1000	128	114	8,9 horas
Website 1	2000	1000	8	90	10,8 minutos
Website 2	2000	1000	8	25	3,3 minutos
Website 3	2000	1000	8	60	8 minutos
Website 4	2000	1000	8	60	8 minutos

Tabela 3 – Acurácia do modelo

Esquema	Exemplo	Acurácia
Biblioteca 1		92,1%
Biblioteca 2		90,5%
Website 1		86,2%
Website 2		100%
Website 3		96,2%
Website 4		96,7%

6.4 Análise dos resultados

Nessa seção são levantadas algumas hipóteses para explicar os resultados obtidos. Primeiro, deseja-se explicar porque utilizar o modelo treinado para solucionar a biblioteca 2, como base para a transferência de aprendizado, resultou em uma melhor acurácia em todas as situações. O possível motivo é que os captchas gerados pela biblioteca 2 são mais complexos, no sentido que existe uma maior variação no ruído e no fundo das imagens, como pode ser observado nas Figuras 9 e 10. Além disso, essa biblioteca utiliza 3 fontes de texto diferentes, enquanto a biblioteca 1 somente uma (embora isso possa ser alterado). Dessa forma, o modelo treinado para a biblioteca 2, precisa aprender parâmetros que são naturalmente mais genéricos, tornado mais fácil ajustá-los para outros esquemas de captchas.

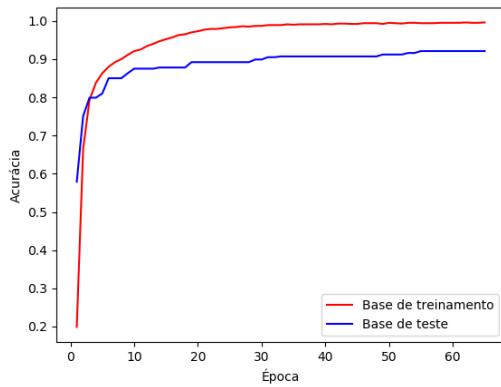


Figura 12 – Acurácia - Biblioteca 1

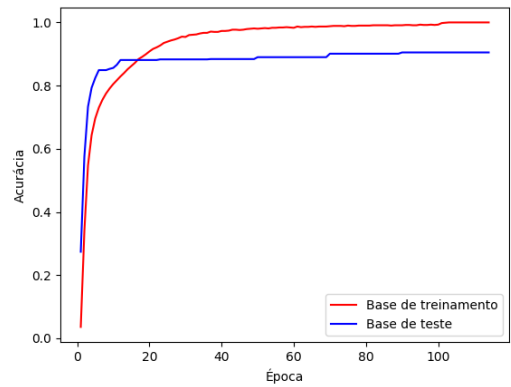


Figura 13 – Acurácia - Biblioteca 2

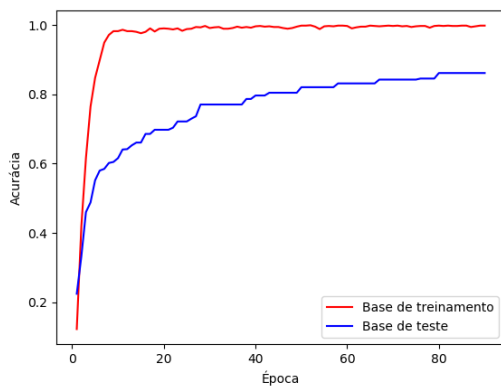


Figura 14 – Acurácia - Website 1

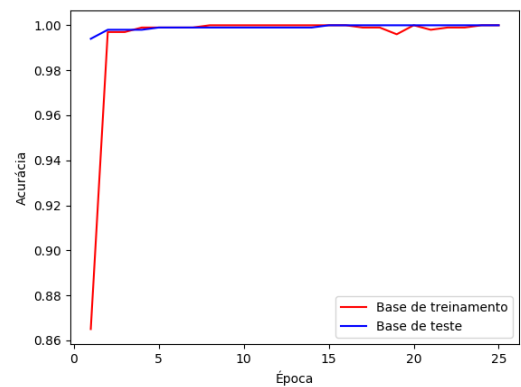


Figura 15 – Acurácia - Website 2

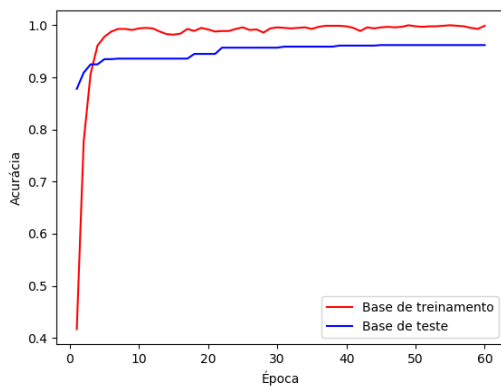


Figura 16 – Acurácia - Website 3

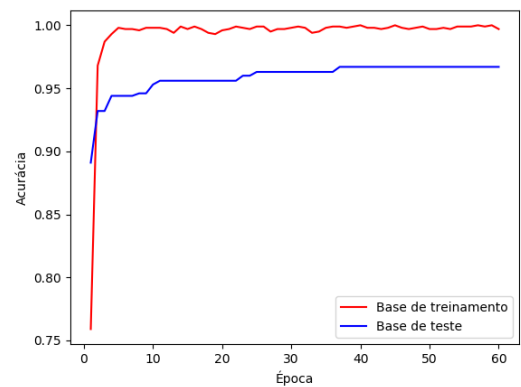


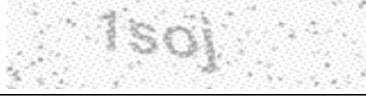



Figura 17 – Acurácia - Website 4

Também é interessante analisar os resultados com maior e menor acurácia. O resultado de maior acurácia, obtido para o esquema de captcha utilizado pelo website 2, não é surpreendente. É mais fácil solucioná-lo porque são utilizados apenas dígitos numéricos. Um pouco mais difícil de explicar é o resultado de menor acurácia (website 1). Ao observar os exemplos mostrados na Figura 11, não há uma razão evidente. Mas

analisando alguns exemplos classificados incorretamente pelo modelo é possível entender o motivo. Na tabela abaixo são mostrados alguns desses exemplos, como pode ser visto, alguns caracteres são muito parecidos, como o “t” e “f” ou “0” e “o”, e isso dificulta o reconhecimento do texto.

Tabela 4 – Exemplos de captchas do website 1 solucionados incorretamente

Exemplo	Texto correto	Texto previsto
	ftmf	ffmf
	oz54	0z54
	1soj	1s0j
	okz7	0kz7

6.5 Comparação com outros trabalhos

Nesta seção são comparados dois métodos recentes com o método deste trabalho. A primeira comparação feita é com o trabalho de [Noury e Rezaei \(2020\)](#), no qual foi avaliada a eficiência dos captchas gerados pela mesma biblioteca Python (biblioteca 1) utilizada neste trabalho. Para fazer a comparação, foram gerados dois novos conjuntos de teste, de forma semelhante aos usados no trabalho de Noury e Rezaei. Um conjunto com captchas contendo apenas caracteres numéricos entre 0 e 9, e o outro com todos os dígitos numéricos e letras minúsculas, exceto pelos caracteres “i”, “l”, “o” e “0”. Ambos os conjuntos contendo 20.000 captchas com tamanho fixo de 5 caracteres. A acurácia do modelo foi novamente calculada nesses conjuntos (o modelo não foi retreinado) e o resultado é mostrado na Tabela 5. O modelo utilizado neste trabalho obteve maior acurácia mesmo utilizando uma quantidade menor de captchas para o treinamento (150.000 *versus* 500.000) e ainda tem a vantagem de poder ser utilizado para solucionar captchas com texto de tamanho variável.

Tabela 5 – Comparação da acurácia com outros trabalhos

Metodologia	Acurácia (numérico)	Acurácia (alfanumérico)
Este trabalho	99,77%	98,78%
Noury e Rezaei (2020)	98,90%	98,30%

Uma comparação de acurácia mais interessante seria com o trabalho de [Wang et al. \(2020\)](#), onde também é utilizado um modelo de CNN (Resnet-101) com blocos residuais. Mas essa comparação não foi feita, porque os captchas utilizados pelos autores não foram disponibilizados, portanto, só seria possível comparar os métodos no conjunto de captchas utilizados neste trabalho, mas é provável que utilizar a ResNet-101 resultará em uma acurácia maior, considerando que esse modelo tem mais camadas.

Considerando as situações onde esse possível ganho de acurácia pode ser desconsiderado ou quando eficiência computacional for mais importante que acurácia, o modelo desenvolvido neste trabalho poderia ser considerado mais adequado. A ResNet-101 tem 101 camadas, enquanto o modelo desenvolvido neste trabalho tem apenas 22, dessa forma, é possível treiná-lo de forma muito mais rápida e utilizando menos memória. Além disso, o mesmo não precisa ser combinado com uma RNN, tornando a sua implementação mais simples, e novamente, necessitando de menos recursos computacionais.

7 Conclusão

Este trabalho apresentou um método genérico, simples e eficiente para avaliar a eficiência de captchas baseados em texto. O método utilizado consistiu em desenvolver e treinar um modelo customizado de CNN utilizando transferência de aprendizado. O método foi testado em 7 esquemas de captchas e foi capaz de solucionar todos os esquemas testados com acurácia entre 86,2% e 100%, que é consideravelmente maior do que os 1% necessários para se considerar um esquema de captcha inseguro.

O modelo desenvolvido é capaz de solucionar captchas sem depender de outras técnicas complexas de processamento de imagens ou de uma grande quantidade de captchas já solucionados. Além disso, o modelo pode ser treinado para solucionar diferentes esquemas de captchas sem a necessidade de alterar o processo ou os parâmetros de treinamento. Quanto a sua eficiência computacional, um captcha pode ser solucionado em apenas 0,01 segundos, e por ser um modelo com poucos parâmetros, pode ser treinado e executado em ambientes com limitados recursos computacionais.

O que diferencia a abordagem deste trabalho de abordagens anteriores utilizando CNNs, é a percepção de que é possível prever em simultâneo os caracteres e o tamanho do texto na saída do modelo, e dessa forma, solucionar captchas com texto de tamanho variável sem utilizar segmentação de caracteres ou uma RNN.

Considerando os resultados obtidos e a facilidade de implementação do método apresentado, conclui-se que captchas baseados em texto talvez não sejam mais adequados e que alternativas consideradas mais seguras deveriam ser utilizadas quando possível. Em trabalhos futuros, propõe-se a análise da eficiência dessas outras alternativas, como o reCAPTCHA¹ e o hCAPTCHA². Outro caminho potencial a ser explorado, é o de investigar como aumentar a segurança dos captchas baseados em texto. Uma possibilidade é a de adicionar nos captchas ruído *adversarial* (Romano et al., 2019) que é imperceptível para humanos, mas torna o reconhecimento mais difícil para redes neurais.

¹ <https://www.google.com/recaptcha/about/>

² <https://www.hcaptcha.com/>

Referências

- Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, v. 5, n. 2, p. 157–166, 1994. Citado na página 26.
- Bursztein, E. et al. The end is nigh: Generic solving of text-based captchas. *8th USENIX Workshop on Offensive Technologies*, 2014. Citado na página 15.
- Bursztein, E.; Martin, M.; Mitchell, J. C. Text-based captcha strengths and weaknesses. *In Proceedings of the 18th ACM conference on Computer and communications security*, 2011. Citado na página 13.
- George, D. et al. A generative vision model that trains with high data efficiency and breaks text-based captchas. *Science*, American Association for the Advancement of Science, v. 358, n. 6368, 2017. ISSN 0036-8075. Disponível em: <<https://science.sciencemag.org/content/358/6368/eaag2612>>. Citado na página 15.
- Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <<http://www.deeplearningbook.org>>. Citado 3 vezes nas páginas 18, 19 e 24.
- Guo, Y. et al. Optic cup segmentation using large pixel patch based CNNs. p. 129–136, 10 2016. Citado na página 21.
- He, K. et al. *Mask R-CNN*. 2017. Citado na página 21.
- He, K. et al. *Deep Residual Learning for Image Recognition*. 2015. Citado na página 26.
- Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural computation*, v. 9, p. 1735–80, 12 1997. Citado na página 16.
- IndoML. *Student Notes: Convolutional Neural Networks (CNN) Introduction*. 2018. Disponível em: <<https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/>>. Citado na página 22.
- Ioffe, S.; Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. Citado na página 25.
- Kingma, D. P.; Ba, J. *Adam: A Method for Stochastic Optimization*. 2014. Citado na página 20.
- Krizhevsky, A.; Sutskever, I.; Hinton, G. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, v. 25, 01 2012. Citado na página 22.
- LeCun, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, v. 1, n. 4, p. 541–551, 1989. Citado 2 vezes nas páginas 14 e 21.

- Leshno, M. et al. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, v. 6, n. 6, p. 861–867, 1993. ISSN 0893-6080. Disponível em: <<https://www.sciencedirect.com/science/article/pii/S0893608005801315>>. Citado na página 24.
- Mori, G.; Malik, J. Recognizing objects in adversarial cultter: Breaking a visual captcha. *In IEEE Computer Society Conferene on Computer Vision and Pattern Recognition*, 2003. Citado na página 15.
- Nielsen, M. A. *Neural Networks and Deep Learning*. [S.l.]: Determination Press, 2015. <<http://neuralnetworksanddeeplearning.com>>. Citado na página 20.
- Noury, Z.; Rezaei, M. *Deep-CAPTCHA: a deep learning based CAPTCHA solver for vulnerability assessment*. 2020. Citado 3 vezes nas páginas 15, 16 e 37.
- Robbins, H.; Monro, S. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, Institute of Mathematical Statistics, v. 22, n. 3, p. 400 – 407, 1951. Citado na página 19.
- Romano, Y. et al. *Adversarial Noise Attacks of Deep Learning Architectures – Stability Analysis via Sparse Modeled Signals*. 2019. Citado na página 39.
- Sandeep Balachandran. *Machine Learning - Convolution with color images*. 2020. Disponível em: <<https://dev.to/sandeepbalachandran/machine-learning-convolution-with-color-images-2p41>>. Citado na página 23.
- Wang, P. et al. Simple and easy: Transfer learning-based attacks to text captcha. *IEEE Access*, v. 8, p. 59044–59058, 2020. Citado 3 vezes nas páginas 13, 16 e 38.

Apêndices

APÊNDICE A – Implementação de um Bloco Residual em PyTorch

```

import torch.nn.functional as F
from torch import nn

"""
Classe que implementa um bloco residual. Se downsample=True as dimensões
da entrada são reduzidas pela metade.
"""

class ResidualBlock3x3(nn.Module):
    def __init__(self, in_dim, out_dim, downsample=False):
        super(ResidualBlock3x3, self).__init__()

        self.downsample = nn.Identity()
        if (downsample):
            self.downsample = nn.MaxPool2d(kernel_size=2)

        self.conv1 = nn.Conv2d(in_dim, out_dim, (3,3), padding=(1,1))
        self.conv1_bn = nn.BatchNorm2d(out_dim)

        self.conv2 = nn.Conv2d(out_dim, out_dim, (3,3), padding=(1,1))
        self.conv2_bn = nn.BatchNorm2d(out_dim)

        self.shortcut = nn.Sequential(
            nn.Conv2d(in_dim, out_dim, (1,1)),
            self.downsample,
            nn.BatchNorm2d(out_dim))

    def forward(self, x):
        s = self.shortcut(x)

        x = self.conv1(x)
        x = self.downsample(x)
        x = self.conv1_bn(x)
        x = F.relu(x)

```

```
x = self.conv2(x)
x = self.conv2_bn(x)
x = F.relu(x + s)

return x
```