Detecção de CAPTCHA com Aprendizado de Máquina e Visão Computacional

CHRISTIAN MAEKAWA - RA: 231867 - E-MAIL: c231867@DAC.UNICAMP.BR,*
GIOVANE DE MORAIS - RA: 192683 - E-MAIL: g192683@DAC.UNICAMP.BR, *
MAÍSA SILVA - RA: 181831 - E-MAIL: m181831@DAC.UNICAMP.BR,*
MATTEUS VARGAS - RA: 262885 - E-MAIL: m262885@DAC.UNICAMP.BR,*
STÉFANI FERNANDES - RA: 147939 - E-MAIL: s147939@DAC.UNICAMP.BR*

***Resumo* – CAPTCHA é um teste centrado no ser humano para distinguir um operador humano de *bots*, programas de ataque ou outro agente computadorizado que tenta imitar a inteligência humana. Esta pesquisa visa desenvolver um reconhecedor de CAPTCHAs, a fim de detectar suas fraquezas, vulnerabilidades e, possivelmente, forças; Foi criado, via *script, datasets* de treino e teste com duas metodologias diferentes de montagem de *string*, aleatório e por permutação, para avaliar se isso gera impacto. Os *datasets* de treino possuem de 90 à 100 mil imagens. A solução proposta é capaz de investigar CAPTCHAs alfanuméricos. Para treinar e testar foi desenvolvido uma Rede Neural, baseada em *Convolutional Neural Networks* (CNN), com *dropout*, e o seu desempenho, em acurácia, foi medido. Variamos parâmetros de treinamento e da rede neural a fim de determinar os cenários com melhores resultados e o papel que cada parâmetro desempenhou no processo. Nos melhor cenário, a solução alcançou 90% e 85% de acurácia para o método aleatório e por permutação, respectivamente, sendo que o *dropout* resolveu questões que poderiam ser problemáticas. Em termos de segurança, a permutação é mais desafiadora para ser quebrada. Ainda assim, deve ser necessário desenvolver pesquisas para aumentar a robustez dos CAPTCHAS.**

***Palavras-chave* – *Convolutional Neural Networks*, CAPTCHA, Visão Computacional**

I. INTRODUÇÃO

CAPTCHA, abreviatura de *Completely Automated Public Turing test to tell Computers and Humans Apart*, é um teste de computador para distinguir entre humanos e robôs [1]. O CAPTCHA pode ser usado para prevenir diferentes tipos de ataques de segurança cibernética. Esses ataques geralmente levam a situações em que programas de computador substituem humanos e tentam automatizar serviços [2], ou trazer sobrecargas no tráfego, o *distributed denial-of-service* (DDOS) [3]. Uma solução é a utilização de um sistema CAPTCHA. Assim, é possível distinguir um atacante se é uma máquina ou não. Geralmente, o CAPTCHA oferece problemas que humanos podem responder rapidamente, mas as máquinas podem achar difícil [4]. Os CAPTCHAs podem ser strings numéricas ou alfanuméricas, voz ou imagens.

Existem formas de tornar os CAPTCHAS mais complexos, como adição de ruído, linhas cruzadas sobre as letras, diferentes tipos de fonte, entre outros métodos. Atualmente, os algoritmos baseados em visão computacional são mais precisos e inteligentes para superar tais barreiras. Neste projeto, vamos nos concentrar em CAPTCHAs baseados em texto, por serem mais comuns, devido ao seu menor custo computacional. Esta pesquisa visa desenvolver um reconhecedor de CAPTCHAs, a fim de detectar suas fraquezas, vulnerabilidades e, possivelmente, forças.

Para isso, criamos, via *script*, quatro *datasets* (dois para treinamento e dois para teste) alfanuméricos com duas metodologias diferentes na sua seleção de caracteres e montagem da *string*, a fim de avaliar se isso poderia impactar de alguma forma a solução. Posteriormente, desenvolvemos uma Rede Neural, baseada em *Convolutional Neural Networks* (CNN), com *dropout*, que foi treinada e testada com tais *datasets*, após a etapa intermediária de pré-processamento dos dados, e o seu desempenho, em acurácia, foi medido. Testamos a solução variando parâmetros de treinamento e da rede neural selecionados a fim de determinar qual configuração atinge os melhores resultados, além de avaliar o papel que cada parâmetro desempenhou.

As diferenças encontradas foram confrontadas e discutidas. Por fim, como conclusão, fica determinado qual *dataset* se saiu melhor com a nossa solução e, em termos de segurança, qual seria a melhor abordagem a ser aplicada e sugestão de modificações, além de confrontar este projeto com trabalhos da literatura a fim de destacar suas contribuições.

O artigo está organizado da seguinte forma: Seção II, com conceitos teóricos necessários para total compreensão do trabalho; Seção III, a descrição, em detalhes, do que foi desenvolvido e como; Seção IV, os testes realizados e os seus resultados; Seção V, discussão dos resultados; Seção VI, conclusão geral acerca do trabalho e o seu posicionamento perante a literatura.

II. REFERENCIAL TEÓRICO

Nesta seção, serão apresentados os conceitos e ferramentas aplicados neste trabalho, a fim assegurar a compreensão por completo do projeto.

A. CAPTCHA

Desenvolvido em meados de 1997 para o site Alta Vista e, posteriormente para o *Yahoo*, o *CAPTCHA* (*Completely Automated Public Turing test to tell Computers and Humans Apart*) é utilizado como uma ferramenta de segurança anti-spam cuja premissa é utilizar uma mensagem distorcida como desafio para comprovar que o acessante é um ser humano,

em vez de um robô/computador. Trata-se de um problema de *Visão Computacional*, resolvido por meio de *Aprendizado de Máquina Supervisionado* e *Redes Neurais Convolucionais*.

O Teste de Turing (postulado por Alan Turing, em 1950) é uma forma de determinar se as atividades de máquina inteligentes podem ser assimiladas ao comportamento humano. Qualquer sistema (incluindo o CAPTCHA) se converge em gerar questões nas quais apenas seres humanos e não os computadores, por mais inteligentes que sejam, não conseguiriam resolver. No caso do CAPTCHA, a pergunta desafio pode ser de referência à interpretação de mensagens alfanuméricas distorcidas em imagens, interpretação de objetos e cliques em partes específicas de uma imagem; partindo da premissa de que máquinas com inteligência artificial, por mais aperfeiçoadas que sejam, não conseguiriam interpretar de forma razoável.

B. Aprendizado de Máquina

Atualmente a produção de dados (imagens, textos, sons, vídeos, dentre outras fontes) está em crescimento. A maior distinção neste cenário é que as informações criadas/processadas ao redor do globo não são produzidas apenas por seres humanos, mas sim, dispositivos eletrônicos e aplicações.

Devido a imensa quantidade de dados gerados, os seres humanos, incapacitados de interpretar tamanhas quantidades, terceirizam cada vez mais esta função para dispositivos dotados de inteligência artificial através do conceito de aprendizado de máquina que, por sua vez, utiliza-se de ferramentas e tecnologia que buscam responder perguntas e gerar *insights* através do consumo dados[5].

1) Treinamento: A premissa básica para todo aprendizado de máquina é o ”treinamento”desta onde se usa dados previamente selecionados para criação e parametrização do determinado modelo de predição, que é usado para gerar previsões de resultados futuros de acordo com os estímulos que o agente inteligente receber [6]. Contudo, para que o próprio treinamento seja satisfatório, os dados devem ser tratados previamente de modo a selecionar quais dados são importantes de fato para o aprendizado. Fato este que requer muito esforço [6].

2) Aplicações e Tipos: O aprendizado de máquina é amplamente utilizado em aplicações a fim de personalizar os seus produtos de forma automatizada para cada usuário, como em recomendação de vídeos, identificação fácil, sistemas de saúde, segurança, entre outras aplicações. Há três grandes categorias de aprendizado de máquina: supervisionado, não-supervisionado e reforço [6]. Nesta obra foi usado o aprendizado supervisionado.

3) Aprendizado de Máquina Supervisionado: Neste tipo de aprendizado, os dados são rotulados, então eles são corretos para realizar o treinamento do modelo. Esse método é deveras eficiente, já que o sistema pode trabalhar com informações corretas [6].

A rotulação dos dados é o cerne deste tipo de aprendizado, pois quando os rótulos são contínuos, então o problema é de regressão e, quando discretos, o problema é de classificação [6].

1) Regressão — Dada uma imagem de homem/mulher, tentar prever sua idade com base em dados da imagem.
2) Classificação — Dada um exemplo de tumor cancerígeno, tentar prever se ele é benigno ou maligno através do seu tamanho e idade do paciente.

O aprendizado supervisionado é aplicável quando o sistema já entende quais entradas estão associadas com determinadas saídas e deve aprender um método de montar essa associação. Usam a compreensão de padrões para tecer previsões [6].

C. Visão Computacional

Visão Computacional é a capacidade responsável pela visão de uma máquina, pela maneira como o computador vislumbra o ambiente à sua volta, angariando informações advindas de imagens capturadas por diversos dispositivos [7]. Isso permite o reconhecimento, manipulação e pensamento sobre objetos que compõem uma imagem.

Sistemas de visão computacional usualmente são especialistas, ou seja, necessitam de conhecimento específico para solucionar determinado(s) problema(s). Portanto, há muitas formas de se implementar um sistema de visão computacional. De acordo com Milano e Honorato (2014), há, minimamente,uma organização de processos comuns nesses tipos de sistemas, na ordem que se segue:

1) Aquisição de Imagem
2) Pré-processamento: a aplicar métodos específicos que facilitem a identificação de um objeto.
3) Extração de características: Extração de características matemáticas que compõem uma imagem.
4) Detecção e segmentação: Processo realizado para destacar regiões relevantes da imagem.
5) Processamento de alto nível: Processo que inclui validação da satisfação dos dados obtidos e classificação dos objetos obtidos em diferentes categorias.

D. Redes Neurais Convolucionais (CNN)

As Redes Neurais Convolucionais (*Convolutional Neural Networks*, CNNs) são Redes Neurais compostas por neurônios com seus respectivos *pesos* e *bias* que requerem treinamento [8]. Cada neurônio é alimentado por algumas entradas. Em seguida é aplicado a ele o produto escalar das entradas e pesos, e uma função não-linear. A CNN é composta pelas seguintes camadas, que também podem ser conferidas graficamente na Figura 1:

1) Input: Representa a entrada, sendo esta uma imagem com suas características de largura, altura e profundidade [8].

As *CNNs* são arquitetadas estritamente para reconhecimento de formas bidimensionais com um alto grau de invariância quanto a translação, escalonamento, inclinação e outras formas de distorção [8].

2) Camada Convolucional: Essa camada é composta por um conjunto de filtros, que são pequenas matrizes de valores reais capazes de aprender de acordo com o treinamento. Nesse momento há operações de convolução para reconhecer a imagem de entrada e criar um mapeamento de características [5]. Isso significa que, se o conjunto de valores de um filtro

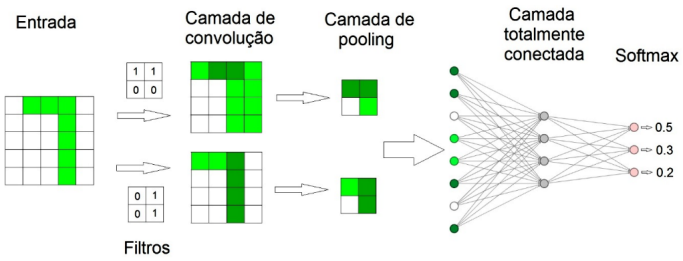


Figura 1: Arquitetura de uma CNN [9]

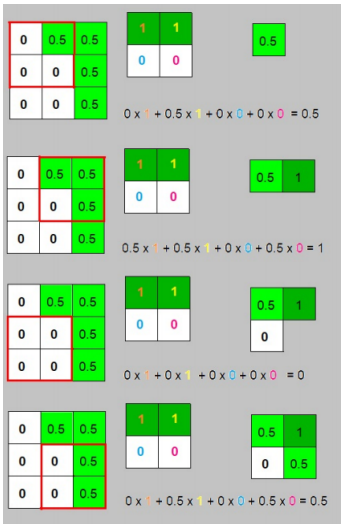


Figura 2: Operação de convolução e *strides* em matrizes de entrada [9]

for capaz de retratar uma região da imagem, a rede como um todo compreende que estes mesmos valores poderão ser usados para identificar outra região da entrada se posto em outro filtro [8]. Mesmo assim, os valores dos filtros variam ao longo do treinamento a fim de tecer reparos, adequações e executar melhores extrações de características de uma imagem de entrada.

Aqui ocorre um fenômeno de deslizamento de quadro que seleciona uma quantidade de valores na imagem. Isso é orientado pelo parâmetro *stride*. Às vezes o tamanho do filtro e do *stride* (Figura 2) não se adequa a imagem, tendo então que acionar o *padding*. Isso não é nada mais do que colocar zeros na borda da imagem para que o deslizamento ocorra [8].

3) Camada de Pooling: É escolhida uma unidade de área para transitar por toda saída da camada anterior. Isso serve para resumir a informação de determinada área em um valor apenas. Por exemplo, se a saída da camada anterior for 24x24 e a unidade de área definida for 2x2, então a saída é uma matriz 12x12. O meio mais empregado para isso é o *maxpooling* (Figura 3), aonde apenas o maior número da unidade é enviado à saída. Isso serve para decrescer a quantidade de pesos a serem aprendidos e também evitar o *overfitting*.

4) Camada Totalmente Conectada: É a última camada de uma CNN. Essa camada se caracteriza pela completude de conexões com a camada antecessora, onde sua entrada é a

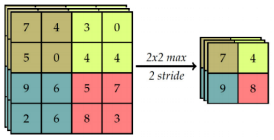


Figura 3: *Pooling* com *maxpooling* [5]

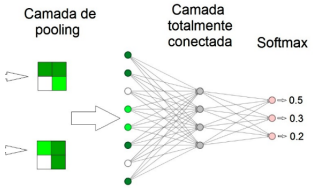


Figura 4: Camada totalmente conectada, com *softmax* [9]

saída da camada anterior e sua saída são N neurônios, com N sendo a quantidade de classes do modelo para finalizar a classificação (Figura 4). É possível então adicionar as técnicas de melhoramento de desempenho, como o *dropout*, por exemplo [9].

Na saída (elemento mais ao extremo) é aplicada uma função (no caso deste projeto, a *sigmoid* [5]) para se ter a probabilidade de dada entrada pertencer a uma certa classe. Aqui é realizado o algoritmo de treinamento supervisionado *backpropagation*. O erro obtido nesta camada é propagado para que os pesos dos filtros das camadas convolucionais sejam ajustados. Assim, os valores dos pesos compartilhados são aprendidos ao longo do treinamento.

E. Funções de Ativação

O funcionamento de um neurônio artificial baseia-se em tecer um cálculo, que é a soma ponderada das entradas com um bias, para decidir sobre seu disparo [5]. Esta saída é então encaminhada como entrada para a próxima camada de neurônios [8]:

$$w * x + b = y$$

Onde x é o conjunto de entradas, w os pesos, b é o *bias* (elemento que usado para aumentar o grau de liberdade dos ajustes dos pesos [8]) e y é a pontuação para cada classe.

1) Função Unidade Linear Retificada (ReLU) [6]: A função de ativação ReLU resolve $f(x) = max(0,x)$, portanto a ativação se dá por um limiar em zero [5]. Ele acelera a convergência do Gradiente descendente quando se comparado às funções *sigmoid* e *tanh*, além de ser implementada com funções menos custosas que a sigmoide e a *tanh*.

2) Função SoftMax: De acordo com Ebermam e Krohling (2018), essa função é aplicada em redes neurais de classificação, sendo a última camada e responsável por apresentar resultados. Essa função estimula a saída da rede a representar a chance dos dados serem de uma determinada classe. É muito útil quando tem que se lidar com várias classes. Essa função transforma as saídas para cada classe em valores entre 0 e 1 e também divide pela soma das saídas. Isso essencialmente dá a probabilidade de a entrada estar em uma

determinada classe. A quantidade de neurônios nela é definido pelo número de classes do problema. Pode ser definida pela equação:

$$y_j = \frac{e^{y(j)}}{\sum_{i=1}^{n} e^{y(i)}}$$

A saída y do neurônio j passa a ser o valor da mesma dividido pelo somatório de todas as saídas dos neurônios dessa camada.

3) Função Sigmoid: É uma função não-linear que resolve:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

A função essencialmente tenta empurrar os valores de Y para os extremos entre $[0, 1]$. O uso de *Sigmoid* é útil para produzir probabilidades já que seus resultados dentro desse limite, podem ser interpretados como a probabilidade de determinado caractere pertencer ou não a determinada classe [5].

F. Função de otimização Adam

O Adam é um meio para otimização estocástica eficiente que requer gradientes de primeira ordem com pouco consumo de memória [10]. Ele calcula as taxas de aprendizagem para diferentes parâmetros a partir de estimativas do primeiro e segundo *momentum* dos gradientes. Por sua vez, *Momentum* [5] é um termo na função de otimização que assume um valor entre 0 e 1 e tem como finalidade aumentar o tamanho das iterações que buscam o mínimo erro.

G. Inicializador de Kernel Glorot Uniform

É um inicializador de pesos de cada camada. Ele calcula o limite $[-limit, limit]$ em que:

$$limit = \sqrt{6/(unidades\_entrada + unidades\_saida)}$$

Em seguida, os pesos são escolhidos aleatoriamente no limite $[-limit, limit]$.

H. Overfitting

Overfitting é quando a diferença entre o valor de perda para o conjunto de treinamento e o valor de perda do conjunto de teste é muito grande [5]. Isso quer dizer que, é notável que a rede memoriza os padrões do treinamento, mas não consegue generalizar, não podendo então prever as saídas corretamente.

I. Dropout

Dropout é a remoção aleatória de neurônios no processo de aprendizagem, a fim de se evitar o *overfitting*. Em outras palavras, ele elimina neurônios individuais a cada etapa do treinamento, tornando a rede neural mais reduzida, com menos conexões e parâmetros, com o objetivo final de melhorar o desempenho da rede.

J. Matriz de Confusão

De acordo com Souza (2019), matriz de confusão é uma tabela que mostra as frequências de classificação para cada classe do modelo. Ela vai nos mostrar as frequências:

- Verdadeiro positivo (*true positive — TP*): quando a classe que estamos buscando foi prevista corretamente.
- Falso positivo (*false positive — FP*): quando a classe que estamos buscando prever foi prevista incorretamente.
- Falso verdadeiro (*true negative — TN*): quando a classe que não estamos buscando prever foi prevista corretamente.
- Falso negativo (*false negative — FN*): quando a classe que não estamos buscando prever foi prevista incorretamente.

As matrizes de confusão são inerentemente conectadas aos seguintes conceitos:

- Acurácia: Diz quanto o meu modelo acertou das previsões possíveis;
- Recall: O quão bom o modelo é para prever positivos, sendo positivo entendido como a classe que se quer prever;
- Precisão: a proporção de identificações positivas;
- f-score: o balanço entre a precisão e o recall de nosso modelo;

K. Ferramentas

Há várias ferramentas disponíveis para implementação de CNNs e demais aplicações envolvendo Aprendizado de Máquina. Aqui foram usados:

- *Tensorflow* [12]: biblioteca de aprendizado, utilizado por meio do Python;
- *Keras* [13]: API de alto nível para construção de redes neurais sobre o TensorFlow;
- *OpenCV (Open Source Computer Vision)* [14]: é uma biblioteca voltada para visão computacional usada para vários tipos de análise em imagens e vídeos;
- *Scikit-learn* [15]: é uma biblioteca Python para trabalhar com Aprendizado de Máquina, com ela já estão implementados diversos métodos, algoritmos e técnicas;

III. Metodologia

Para solucionar o problema CAPTCHA, a abordagem escolhida foi o uso de técnicas associadas à leitura e identificação de *pixels* na imagem com CNN. O processo aqui segue o raciocínio básico de uma CNN: extração de *features* pelas camadas convolucionais (alteram a representação dos dados e aprende os filtros) e classificação pelas camadas finais *Pooling* (reduz a escala para a próxima camada) e *Fully-Connected (Dense)*. Sendo assim cada filtro aprendido é um extrator de *features* e cada imagem resultante de um filtro é um mapa de características. As subseções a seguir irão descrever o processo em detalhes: definição do(s) *dataset*(s), pré-processamento dos dados e estruturação da rede neural. As etapas de treinamento, teste e resultados serão abordadas na Seção IV.

A. Datasets

O objetivo deste projeto é aplicar CNN na leitura e identificação de imagens de *CAPTCHA*. Deve-se adiantar que o *dataset* impacta no desempenho da aplicação, algo comum em Aprendizado de Máquina. No caso os *datasets* foram elaborados pela equipe especificamente para este trabalho por meio do *script* **Gerador_Captcha.py**. Esse gerador produz

Figura 5: CAPTCHA antes e depois do pré-processamento.

imagens *CAPTCHA* de dimensões 60x160 com cinco caracteres salvos como **string_gerada.png**. Ao todo, três *datasets* foram gerados:

1) Baseado em treino aleatório: nesse *dataset* os cinco caracteres que formam a palavra são escolhidos aleatoriamente a partir das letras do alfabeto de dígitos. Possui um total de 99910 imagens;
2) Baseado em permutação: nesse *dataset*, cinco caracteres são escolhidos aleatoriamente, no início, e as imagens seguintes são todas as permutações possíveis desses caracteres iniciais. Na sequência são escolhidos outros cinco caracteres que não haviam sido escolhidos antecipadamente e o processo se repete até atingir um tamanho aceitável para o *dataset*. Possui um tamanho total de 100029 imagens;
3) Teste: foram geradas imagens com cinco caracteres escolhidos aleatoriamente para testar a solução. Possui um total de 19976 imagens;

B. Pré-Processamento

O pré-processamento das imagens se caracteriza pelo redimensionamento das mesmas para 30x80 e transformando-as para escala de cinza com a biblioteca *OpenCV*. Isso é necessário para melhorar o desempenho da aplicação e a conversão de espaço de cores em espaço em cinza é usado para reduzir o tamanho dos dados, mantendo o mesmo nível de precisão de detecção (veja a Figura 5).

Poderíamos reduzir ainda mais a quantidade de dados redundantes e facilitar o processo de treinamento e previsão. A conversão de uma imagem RGB de três canais para uma imagem em escala de cinza não afeta os resultados, pois a cor não é crucial nos sistemas *CAPTCHA* baseados em texto[1].

Após isso, os arquivos são lidos em lote a partir do diretório dos mesmos por meio da função **read_dataset_batch** e guardado em suas respectivas listas (permutação, randômico e testes). A cada arquivo, um *reshape* é feito na imagem para a metade do seu tamanho da altura e comprimento, reduzindo-a em 1/4 do tamanho original. Isso retorna o *dataset* na forma de **matrix[x][y]**. No *plot*, as imagens ficam como na Figura 5.

C. Rede Neural

A rede neural é construída com o auxílio das bibliotecas *Keras* e *Tensor Flow*, que trazem estruturas de dados complexas e parâmetros para os mesmos. A sua entrada são as imagens devidamente pré-processadas. A partir desse ponto, as camadas são instanciadas, na seguinte ordem:

- *Input*: entrada de dados, serve como instanciação, usando como *shape* o mesmo da imagem;
- Três camadas convolucionais: construído com a *Conv2D*, sem preenchimento de *padding*, usando bias, mas sem inicializador do mesmo, ativado por *'ReLU'*, com inicializador de *kernel 'Glorot Uniform'*. A primeira camada utiliza 16 filtros com *kernel size* de (3x3). As duas outras camadas utilizam 32 filtros.
- Três camadas de *pooling*, com *MaxPooling2D*: localizadas após a primeira e após a segunda camada convolucional, e depois do *Batch normalization*, o objetivo é fazer uma amostragem reduzida da representação da entrada, reduzindo sua dimensionalidade e permitindo suposições sobre os recursos contidos nas sub-regiões classificadas. Ele adiciona um *padding* nas bordas da matriz.
- Camada de *Dropout*: aplicado após os dois primeiros *maxpooling* e na camada totalmente conectada. O primeiro *dropout* possui um valor de 0.1, o segundo 0.2 e o último de 0.5. Isso permite espalhar o conhecimento do neurônio excluído, aumentando a robustez dos demais.
- Camada de normalização (*Batch Normalization*): localizada após a terceira camada de convolução, ela normaliza as ativações da camada anterior em cada lote, ou seja, aplica uma transformação que mantém a ativação média próxima a 0 e o desvio padrão da ativação próximo a 1.
- Camada de saída: realiza uma operação de *MaxPooling2D*. Ao final é aplicado o *Flatten* para nivelar a entrada para próxima camada. O *Flatten* opera uma transformação na matriz da imagem, alterando seu formato para um *array*.
- Camada totalmente conectada (*Dense*): usa como entrada os resultados da camada anterior com cinco saídas, cada uma representando um caractere da imagem do CAPTCHA. É aplicada a função *sigmoid* para se ter a probabilidade da entrada pertencer a uma certa classe. A essa camada também é aplicado o *Dropout*.
- Compilação: o modelo finalmente é compilado utilizando como otimizador o Adam [10] e como métrica a **acurácia**. A **perda** (*loss*), foi definida como sendo o ***Categorical Crossentropy*** [12] que é uma função de perda usada em tarefas de classificação de várias classes, onde um exemplo pode pertencer apenas a uma dentre muitas categorias possíveis, e o modelo deve decidir qual delas [13].

Com a rede neural devidamente montada, é necessário treinar e testar. O treinamento é um para cada *dataset* de treino citado na subseção III-A. A representação gráfica desse esquema pode ser visto na Figura 6.

D. Treinamento da Rede Neural

Há duas abordagens aqui: o *dataset* montado no método de permutação e aquele montado com o método da aleatoriedade (subseção III-A). Em uma rápida observação, é fácil presumir que haverão diferenças no resultado final, mas é preciso comprovar. Para comprovar, é realizado o treinamento da rede neural, um para cada *dataset*. O treinamento é definido pelos seguintes hiper parâmetros [12]:

- *Epochs*: Número de épocas para treinar o modelo. Uma época é uma iteração através de todo o x e y dados fornecidos. Aqui, foi definido um valor de 500;

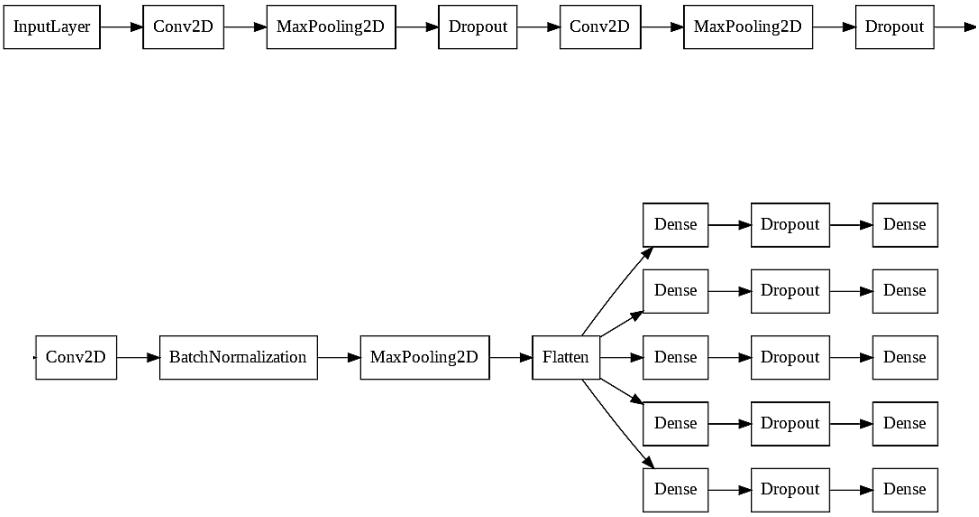


Figura 6: Esquema da Rede Neural proposta para solucionar o CAPTCHA

- *Batch Size*: Número de amostras por atualização gradiente. O valor definido aqui é 1000;
- *Validation Split*: *Float* entre 0 e 1. Fração dos dados de treinamento para ser usado como dados de validação. O valor definido aqui é 0.2;
- *Callbacks* [13]: objeto que pode executar ações em vários estágios do treinamento. Aqui, ele interrompe o treinamento em função do número de *epochs* sem melhora, no caso 200. Outro *callback* é o salvamento do modelo a cada 100 *epochs*;

As variáveis supracitadas são implantadas como parâmetros na função **fit** (treinamento). Essa função recebe como entrada o conjunto de dados do *dataset*, tendo como resultado alvo os caracteres identificados pela rede neural ($y[0]$ à $y[4]$). Isso pode ser notado no esquema da Figura 6, onde a primeira camada, *Input*, corresponde a entrada de dados de imagens e, ao final, há uma cinco *Denses*, uma para identificar cada caractere do CAPTCHA.

Com essa configuração, a rede foi colocada à treino, um para cada *dataset*. O resultado das Figuras 7 e 8 demonstra que, indo para 500 *epochs* ele tende a convergir. A etapa final é a execução dos testes e discussão dos resultados.

IV. Testes

A ideia geral é que dado qualquer CAPTCHA após o treinamento, o modelo consiga interpretá-lo. Para tanto, a acurácia deve ser alta. Aqui, foi dado foco na medida de **acurácia**. O ambiente de execução escolhido foi o *Google Colaboratory*, que é um ambiente de pesquisa para criar protótipos de modelos de aprendizado de máquina em poderosas opções de *hardware*, como GPUs (*Graphics Processing Unit*) e TPUs (*Tensor Processing Unit*) [16], além de oferecer até $12.72GB$ de Memória RAM e $107.77GB$ de Disco, o que, neste projeto, é o suficiente para executar a solução e armazenar o *dataset*.

Para cada *dataset*, foi avaliada a acurácia da solução como um todo e de caractere por caractere. O número de imagens para cada *dataset* de treinamento e de teste pode ser verificado na subseção III-A. Aqui serão demonstrados os testes e aquilo que fora avaliado. Sua discussão se dará na seção V. A variação de cenários se inicia pelas seguintes indagações:

- Se há cenários melhores;
- Se variar hiperparâmetros influencia no resultado final;
- Se a constituição do *dataset* influencia, de alguma forma, o resultado final;
- Avaliar se houve *overfitting* e, consequentemente, o papel do *Dropout*;
- Determinar a melhor combinação e seu respectivo cenário;

Nos focamos então mais na parte do *Dense* para frente, na nossa rede neural. O primeiro teste executado é o *baseline*, ou seja, a base, ou simplesmente, *baseline*. Desse cenário foram escolhidos os atributos que deveriam ser explorados nos testes deste projeto, que são:

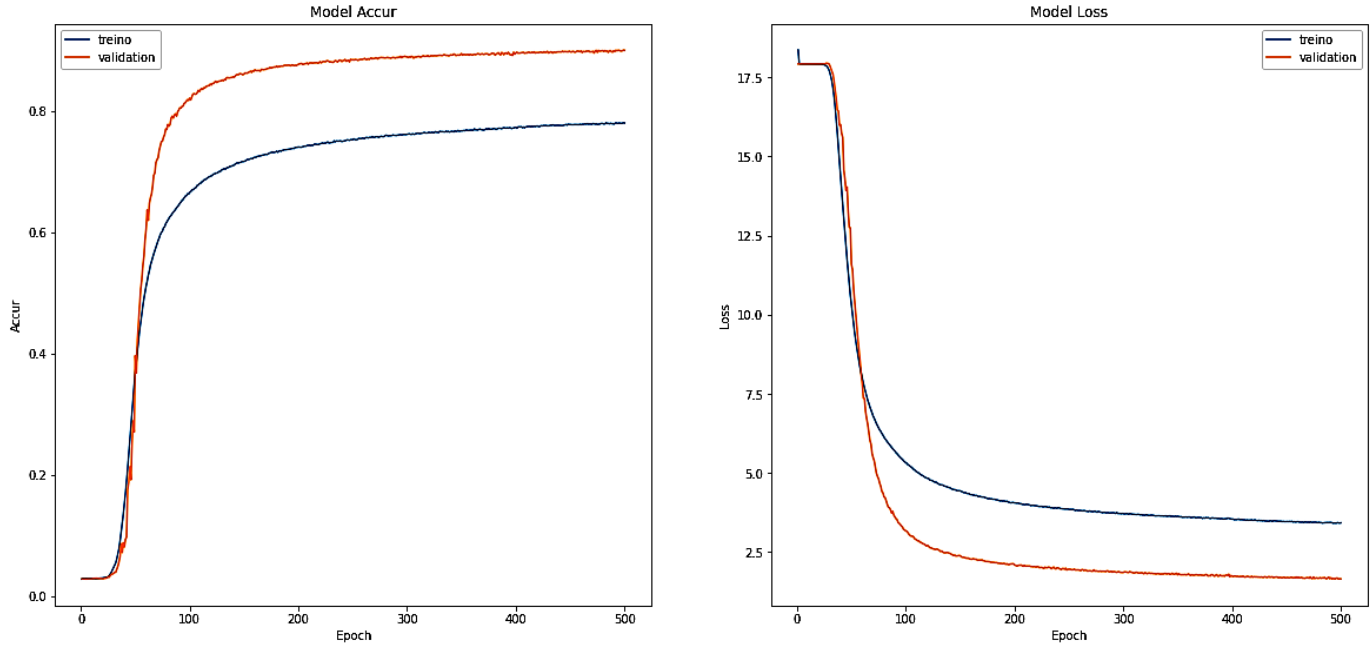
Atributos de treinamento

- Número de *Epochs* (padrão 500);

Atributos da Rede Neural

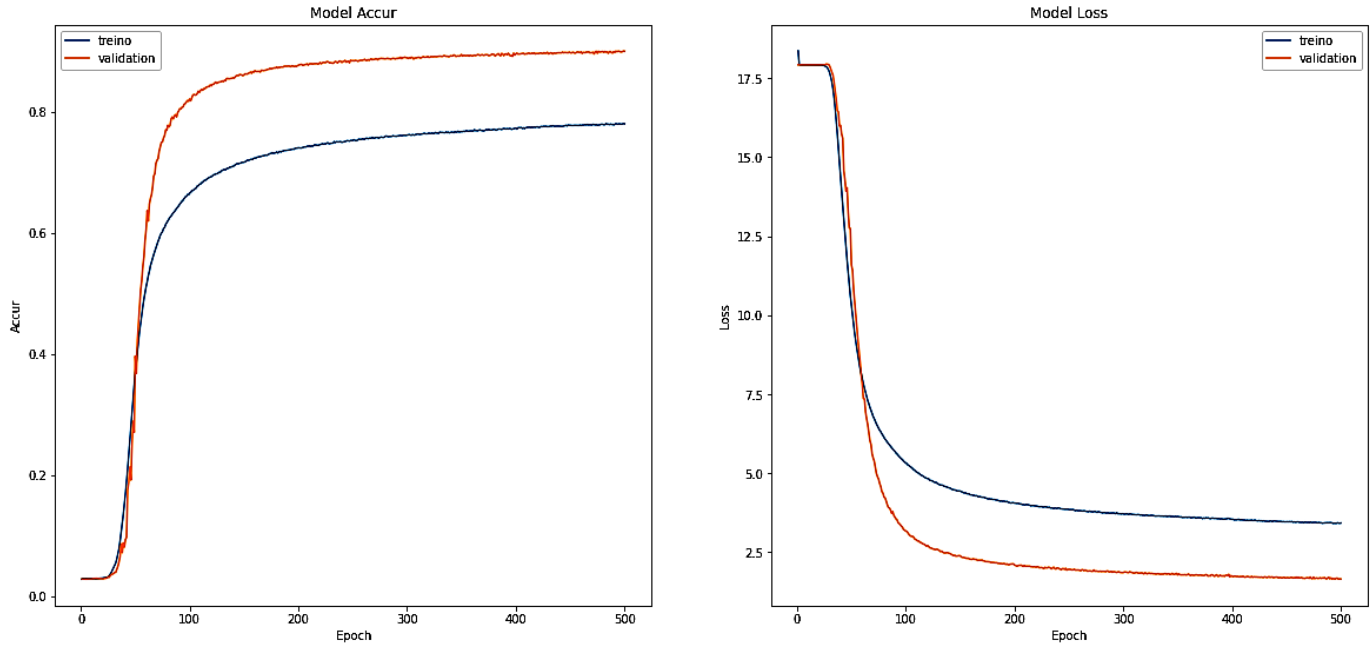
- Função de ativação *Sigmoid* (camada totalmente conectada, *Dense*);
- Função de otimização *Adam*;
- Uso de *Dropout*;

Os resultados dessa combinação estão sumarizados na Tabela I. Três características aqui ficam bem claras: a precisão



(a)

Figura 7: Treinamento da Rede Neural com *dataset* aleatório



(a)

Figura 8: Treinamento da Rede Neural com *dataset* de permutação

Dígito	Aleatório	Permutação
1	0.94	0.88
2	0.88	0.80
3	0.86	0.77
4	0.88	0.81
5	0.94	0.90
Total	**0.90**	**0.83**

Tabela I: Resultados da acurácia dos dígitos nos testes do *baseline*.

Dígito	Aleatório	Permutação
1	0.05	0.03
2	0.05	0.03
3	0.05	0.03
4	0.05	0.03
5	0.06	0.03
Total	**0.05**	**0.03**

Tabela II: Resultados da acurácia dos dígitos nos testes do cenário Softmax-SGD.

retornada é muito boa, os dígitos mais extremos possuem uma acurácia mais alta do que os dígitos do meio. Por fim, os resultados do *dataset* de permutação são inferiores ao do *dataset* aleatório, contudo o resultado da predição com algumas amostras se mostrou favorável, como demonstra a Figura 9.

Como a configuração original demonstra bons resultados, os próximos testes tem como foco a busca cenários melhores e esclarecer a função de cada atributo. A primeira mudança a ser feita são nos hiperparâmetros da rede neural, mais especificamente a função de ativação na camada *Dense* e a função de otimização, mas mantendo os mesmos atributos de treinamento. As alternativas encontradas foram a *Softmax (Sigmoid)* e *SGD (Adam)*, como sugere o trabalho de [1]. Portanto, as mudanças são:

- *Softmax-SGD* (Tabela II);
- *Sigmoid-SGD* (Tabela III);
- *Softmax-Adam* (Tabela IV);

Dígito	Aleatório	Permutação
1	0.05	0.03
2	0.04	0.04
3	0.04	0.03
4	0.04	0.03
5	0.05	0.03
Total	**0.04**	**0.03**

Tabela III: Resultados da acurácia dos dígitos nos testes do cenário Sigmoid-SGD.

Dígito	Aleatório	Permutação
1	0.93	0.82
2	0.87	0.71
3	0.85	0.67
4	0.88	0.71
5	0.93	0.84
Total	**0.89**	**0.75**

Tabela IV: Resultados da acurácia dos dígitos nos testes do cenário Softmax-Adam.

Dígito	Aleatório	Permutação
1	0.94	0.90
2	0.88	0.82
3	0.86	0.80
4	0.89	0.83
5	0.94	0.91
Total	**0.90**	**0.85**

Tabela V: Resultados da acurácia dos dígitos nos testes do cenário Sigmoid-Adam, 1000 epochs.

Dígito	Aleatório	Permutação
1	0.93	0.89
2	0.80	0.72
3	0.74	0.67
4	0.83	0.75
5	0.94	0.91
Total	**0.85**	**0.79**

Tabela VI: Resultados da acurácia dos dígitos nos testes do *baseline*, sem *Dropout*.

Essa alteração não conseguiu gerar resultados melhores, aliás, todos são piores, mas as diferenças supracitadas entre os *datasets* se mantém, sendo o aleatório superior. É possível perceber que a permanência do *Adam* como otimizador manteve os valores totais de acurácia em níveis satisfatórios, indicando que ele possa vir a ser o ponto de sucesso desse modelo (Figuras 10 e 11).

A próxima série de testes foca em avaliar os **atributos de treinamento**. Haja visto anteriormente que próximo de 500 o modelo tende a convergir, foi decidido por aumentar, não diminuir, o número de *epochs* e avaliar se isso impactaria diretamente no resultado final, como demonstra a Tabela V.

Esse resultado é factível, novamente tendo em vista que ele busca a convergência indo para 500 *epochs*. Curiosamente, essa configuração não gera tanto impacto assim no *dataset* aleatório, apesar de alguns dígitos ganharem um pouco mais de precisão, mas o *dataset* de permutação melhora bem. Ele ganha de 0.02 à 0.03 de ganho por dígito, o que faz com que todos eles cheguem à casa de 0.8, ou 80% de acurácia. O gráfico da Figura 12 compara esse cenário com o seu correspondente, o de 500 *epochs*. Esse percentual é considerado muito bom, e passa da mesma forma nos testes de predição da Figura 9.

Por fim, o último teste executado é específico para o *Dropout*. O teste é básico, executa-se o *baseline* e cataloga os resultados de acurácia, entretanto, agora sem o *dropout*. Não é certo de que haja *overfitting*, o que seria a função principal do *dropout*, mas a inserção desse elemento pode ser muito útil para deixar a execução da rede neural mais precisa. Removendo o *dropout* chega-se ao resultado sumarizado na Tabela VI.

O uso do *Dropout* se mostra muito eficaz para elevar a precisão. Outro fato evidenciado: sem o *dropout* o modelo converge em menor tempo (por volta de 500 *epochs*), mas com valores de acurácia inferiores. As vantagens do *dropout* podem ser vistas na Figura 13. Portanto, como apresentado nos testes, a melhor configuração, sob o ponto de vista dos atributos

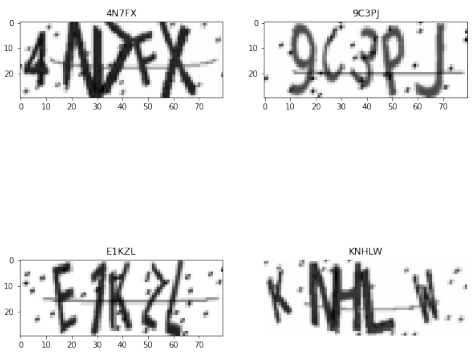


Figura 9: Predição favorável em ambos os *datasets*.

selecionados, é o *baseline* da rede neural, mas utilizando de 1000 *epochs*, algo que atinge diretamente o *dataset* de permutação.

V. Discussão dos resultados

De acordo com os testes realizados, algumas características se destacaram e outros questionamentos surgiram:

1) O *baseline* se mostrou muito bom para o *dataset* aleatório e bom para o de *permutação*;
2) Os dígitos da extremidade possuem valores maiores de acurácia do que os dígitos do meio;
3) Nas alterações feitas entre as funções de ativação e de otimização, fica demonstrado que o otimizador *Adam* mantém as taxas de acurácia altas;
4) No treinamento, o valor de 1000 *epochs* se mostra mais relevante do que o *baseline*;
5) O real papel do *dropout*;

A diferença entre os *datasets* está na sua criação (seção III-A). De acordo com as observações, a hipótese mais plausível é de que ele possa ter alguma dificuldade em generalizar, haja visto que os primeiros cinco caracteres deverão gerar filhos. Já a próxima sequência utilizará outros caracteres diferentes da geração anterior. Por exemplo, se a primeira sequência for: $4MKRA$, os próximos serão derivados disso. Já à próxima geração não poderá usar nenhum desses caracteres. Isso pode ocasionar uma dificuldade de padronização, uma vez que ele não irá ver tantas vezes assim o mesmo caractere e, possivelmente, unido à um conjunto fixo de outros caracteres adjacentes.

Já com o *dataset* aleatório, como o nome diz, qualquer caractere pode aparecer a qualquer momento. Isso deveria gerar mais dificuldade, mas não ocorre. Provavelmente, é o oposto do *dataset* de permutação, ele pode ver mais vezes o mesmo caractere, unido à outros em mais formações.

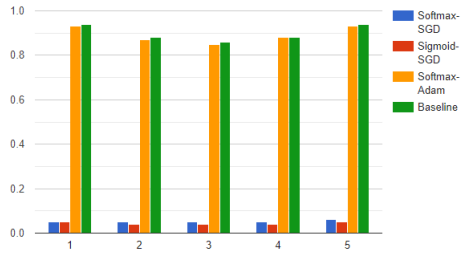
A questão dos dígitos do meio, isso não necessariamente prejudica a predição, mas chama a atenção. Para confirmar isso, mais algumas execuções tiveram que ser feitas e analisadas, bem como verificar como os filtros estão agindo e a constituição dos CAPTCHAS em si. Nas execuções extras, com o *baseline* ele continuou a acertar os caracteres (vide a Figura 9). A análise dos filtros pode sugerir uma resposta. Como mostra a Figura 14, os dígitos do meio aparecem muito juntos e os extremos mais separados, algo que se justifica quando comparado com o seu respectivo CAPTCHA (Figura 14, à direita). Portanto, há um esforço maior em distinguir o que está no meio, por isso os números de acurácia levemente menores.

Sobre a questão dos ativadores-otimizadores, a teoria consegue responder essa questão. A aplicação da função *Sigmoid* adiciona um recurso de não-linearidade aos neurônios, o que melhora o potencial de aprendizado e também a complexidade desses neurônios ao lidar com entradas não lineares. Além disso, estamos lidando com um problema de classificação binária, mesmo não parecendo, mas na verdade ele olha caractere por caractere, então dada a escolha feita pela rede neural por determinada classe, cabe ao *Sigmoid* dizer se é ou não. Então, nesse caso, o *Sigmoid* busca ser mais preciso do que o *Softmax*.

De acordo com os gráficos das Figuras 10 e 11, é o Adam quem eleva os valores de acurácia. O Adam calcula as taxas de aprendizagem adaptativas individuais para cada parâmetro [17], onde o SGD generaliza um pouco mais e calcula o valor do gradiente antes da atualização dos parâmetros [18]. Para o treinamento do SGD funcionar o método é executado dezenas ou centenas de vezes até encontrar os valores mínimos de pesos e bias. O que ocorre aqui então é que o SGD deve ficar preso em um mínimo local e não consegue identificar valores gerais. Vantagem do Adam que faz isso individualmente e, unido ao *Sigmoid* ele consegue inferir valores aos parâmetros e classificar, binariamente, se determinado caractere é da classe que ele presume, ou não.

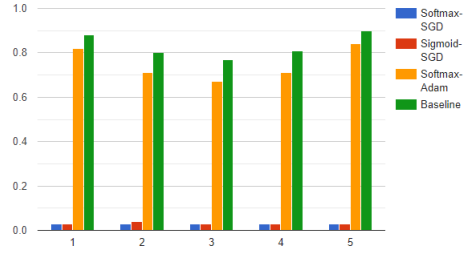
A avaliação do número de *epochs* faz sentido, já que o *epoch* é a quantidade de vezes que o conjunto de dados inteiro passa pela rede neural, e o modelo precisa de mais *epochs* do que o *baseline* para alcançar a total convergência. De acordo com o gráfico da Figura 12, no *dataset* aleatório, isso causa um pouco mais de impacto no dígito 4. Entretanto, é no *dataset* de permutação que ele impacta mais. Isso significa que os dados desse *dataset* precisa passar mais pela rede neural para que todos os dígitos tenham uma acurácia maior ou igual à 0.8, o que, aqui, significa uma acurácia muito boa.

Por último, a análise do *dropout*. Esse método é indicado



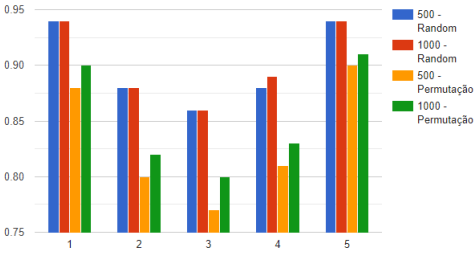
(a)

Figura 10: Comparativo entre ativações-otimizadores no *dataset* aleatório



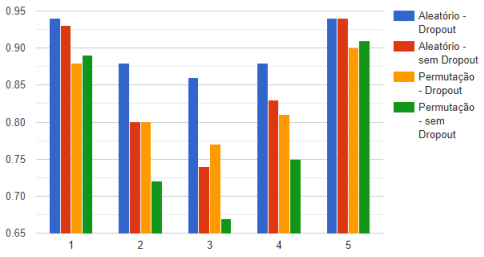
(a)

Figura 11: Comparativo entre ativações-otimizadores no *dataset* de permutação



(a)

Figura 12: Comparativo entre o uso de 500 e 1000 *epochs* em cada um dos *datasets*



(a)

Figura 13: Comparativo entre o uso e a exclusão de *dropout* em cada um dos *datasets*

Figura 14: A forma como a CNN enxerga os caracteres e como o mesmo é constituído. A forma como os caracteres são montados influenciam na acurácia.

Referência	Acurácia
Wei et al. [21]	98.43%
Noury et al. [1]	98.3%
Goodfellow [22]	97.84%
Yu et al. [23]	92.37%
Proposta	**90% (aleatório)/85%(permutação) - Melhor cenário**

Tabela VII: Comparativo entre a proposta atual, na melhor configuração encontrada, e outras soluções da literatura.

para caso ocorra *overfitting*, mas não é certo se de fato isso ocorre aqui. Removendo essa característica, os resultados de acurácia caem, mas não chegam a ser tão ruins. Isso pode indicar que o *dropout* esteja otimizando e deixando a rede mais *objetiva*, já que cada neurônio restante se torna mais robusto, generalizando melhor, melhorando o seu desempenho. Entretanto, o gráfico da Figura 13 prova que o *dropout* é essencial para este modelo, já que é ele quem eleva a acurácia dos dígitos do meio, uma vez que sem isso os valores caem muito, podendo, nesse caso, ser mesmo um problema. reduz co-adaptações complexas de neurônios. Os trabalhos de [19] e [20] corroboram essa análise.

VI. Conclusão

Projetamos, personalizamos e ajustamos uma rede neural baseada na CNN para detecção CAPTCHA utilizando dois *datasets* alfanumérios, um montado sob a técnica de aleatoriedade dos caracteres e o segundo com permutação de cinco caracteres, a fim de determinar se há alguma diferença entre ambos. A configuração de *baseline*, inicialmente, já demonstrou bons resultados 90% no aleatório e 83% na permutação, com 500 *epochs*, ativado por *sigmoid*, otimizado por *Adam* e com *dropout*.

Nos testes de variação de parâmetros, ficou claro que a melhor configuração é o *baseline* com 1000 *epochs*, elevando o *dataset* de permutação à 85% de acurácia total. O *dataset* aleatório facilmente alcança valores elevados de acurácia, mas o de permutação foi desafiador, haja visto que poucas configurações retornam resultados bons. A questão do embaralhamento dos dígitos centrais foi outro desafio inesperado, mas corrigido com *dropout*. Olhando para os CAPTCHAS pelo viés do seu papel principal que é prover segurança, o *dataset* de permutação é o que mais se aproxima de uma abordagem interessante e desafiadora. Uma solução como esta é interessante para poder mostrar vulnerabilidades desta técnica e buscar maneiras de melhoria a fim de prover mais segurança.

Comparado com a literatura, o presente trabalho se posiciona como demonstrado na Tabela. Ele chega próximo dos valores de outros trabalhos, entretanto os mesmos usam de 500 mil à 1 milhão de imagens, sendo a maioria apenas numéricos ou dígitos. Portanto, o modelo proposto aqui contribui no sentido de ser mais otimizado e com mais relevância no uso do *dropout*. Como futuros trabalhos, a sugestão seria avaliar CAPTCHAS com diferentes tamanhos, tecendo variações em outros parâmetros que não foram explorados aqui e outros métodos de montar um CAPTCHA.

REFERÊNCIAS

[1] Z. Noury and M. Rezaei, "Deep-captcha: a deep learning based captcha solver for vulnerability assessment," *arXiv preprint arXiv:2006.08296*, 2020. 1, 5, 8, 10

[2] O. Bostik and J. Klecka, "Recognition of captcha characters by supervised machine learning algorithms," *IFAC-PapersOnLine*, vol. 51, no. 6, pp. 208–213, 2018. 1

[3] J. Nazario, "Ddos attack evolution," *Network Security*, vol. 2008, no. 7, pp. 7–10, 2008. 1

[4] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *International conference on the theory and applications of cryptographic techniques*. Springer, 2003, pp. 294–311. 1

[5] M. R. M. d. Santos *et al.*, "Identificação de textos em imagens captcha utilizando conceitos de aprendizado de máquina e redes neurais convolucionais," 2018. 2, 3, 4

[6] S. Russell and P. Norvig, "Artificial intelligence: a modern approach," 2010. 2, 3

[7] D. de Milano and L. B. Honorato, "Visão computacional," 2014. 2

[8] S. Haykin, *Redes neurais: princípios e prática*. Bookman Editora, 2007. 2, 3

[9] R. A. K. Elivelto Ebermam, "Uma introdução compreensiva às redes neurais convolucionais: Um estudo de caso para reconhecimento de caracteres alfabéticos," *Revista de Sistemas de Informação da FSMA*, no. 22, pp. 49–59, 2018. 3

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 4, 5

[11] E. G. d. Souza, "Entendendo o que é matriz de confusão com python," Apr 2019. [Online]. Available: https://medium.com/data-hackers/entendendo-o-que-\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{e\global\mathchardef\accent@spacefactor\spacefactor}\accent19e\egroup\spacefactor\accent@spacefactor-matriz-de-confus~ao-com-python-114e683ec509#:~:text=\unhbox\voidb@x\bgroup\let\unhbox\voidb@x\setbox\@tempboxa\hbox{E\global\mathchardef\accent@spacefactor\spacefactor}\accent19E\egroup\spacefactor\accent@spacefactorumtabelaquemostra,estamosbuscandofoiprevistacorretamente. 4

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016. 4, 5

[13] F. Chollet, "keras," https://github.com/fchollet/keras, 2015. 4, 5, 6

[14] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. 4

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. 4

[16] E. Bisong, "Google colaboratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 59–64. 6

[17] S. Silva and A. Serapiao, "Detecção de discurso de ódio em português usando cnn combinada a vetores de palavras," in *Proceedings of KDMILE 2018, Symposium on Knowledge Discovery, Mining and Learning, São Paulo, SP, Brazil*, 2018. 9

[18] V. Jeronymo, "Implementando métodos de otimizaçao para treinamento de redes neurais com pytorch," Ph.D. dissertation, Universidade Estadual de Campinas, 2019. 9

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105. 10

[20] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012. 10

[21] L. Wei, X. Li, T. Cao, Q. Zhang, L. Zhou, and W. Wang, "Research on optimization of captcha recognition algorithm based on svm," in *Proceedings of the 2019 11th International Conference on Machine Learning and Computing*, 2019, pp. 236–240. 10

[22] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv preprint arXiv:1312.6082*, 2013. 10

[23] N. Yu and K. Darling, "A low-cost approach to crack python captchas using ai-based chosen-plaintext attack," *Applied Sciences*, vol. 9, no. 10, p. 2010, 2019. 10