UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LUCAS DA SILVA BIFF

Um Método de Ataque Adversarial a Redes Neurais Convolutivas para Reconhecimento Facial

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Bruno Castro da Silva

Porto Alegre
2020

AGRADECIMENTOS

Primeiramente, gostaria de agradecer aos meus pais, que sempre me ensinaram tudo e são os responsáveis pela minha formação enquanto pessoa. Amo vocês.

Ao André Palmeira que, ao longos destes 4 anos e meio, esteve sempre ao meu lado e observou toda minha transformação enquanto futuro cientista. Não há, atualmente, melhor pessoa que mais me faça feliz quanto você.

Ao meu grande amigo, Keoma Pacheco, pela amizade que se estende por mais de uma década. És uma das pessoas que mais me rende gargalhadas, mesmo nos momentos ruins. Aguarde minha visita.

À minha amiga, Luísa Cornetet, a quem admiro muito. Não é com qualquer pessoa que se conversa sobre ciência da computação. Foi de grande ajuda revisando boa parte do texto e dando dicas. Também a visitarei em breve.

Aos meus colegas e ex-colegas de trabalho, Rafael Santana, Marcia Santos e Paola Macalão. Obrigado pelo companheirismo que recebo até hoje e saibam que podem sempre contar comigo. Para qualquer coisa.

Ao meu colega e dupla de graduação, Eduardo Gonçalves Pereira, que foi um dos maiores presentes que recebi da UFRGS. Continuarei me esforçando para manter essa amizade maravilhosa.

Ao senhores Athos Lagemman e Gabriel Conte que foram meus companheiros nestes últimos semestres. Ainda quero dividir muitas mesas de bares com vocês.

A todos, sem exceção, que junto comigo ingressaram no curso de Ciência da Computação em 2013/1. Se não fossem por vocês, certamente não teria chegado tão longe. Vocês são incríveis e espero acompanhá-los, mesmo que de longe, para sempre.

E ao meu orientador Professor Bruno Castro da Silva que me apresentou tudo o que precisava para realização deste trabalho, dando ideias e apresentando recursos. Ficou comigo até o fim e sou eternamente grato por isso.

RESUMO

Atualmente, as tecnologias de redes neurais permitem realizar atividades de reconhecimento facial com uma precisão nunca vista antes. Reconhecimento facial está sendo usado para fins de mídia social, para, por exemplo, reconhecimento de pessoas que estão presentes em uma foto. No entanto, também está sendo usado por departamentos de polícia para identificar participantes de protestos. Isso tem causando uma discussão quanto a invasão de privacidade e existe um movimento de pessoas que tentam desenvolver maneiras de evitar esse sistema de vigilância contante. Muitos estudos surgiram após a descoberta de que redes neurais são facilmente enganadas com pequenas alterações não vistas a olho nu. Nesse trabalho, investigamos uma variante de ataque que consiste no uso de uma abordagem de otimização para descobrir um conjunto geométrico simples, de diferentes escalas, posições e cores que, quando colocadas sobre o rosto da pessoa, faz com que uma rede neural não consiga realizar classificações corretamente. Métodos de estado da arte têm um percentual de sucesso de ataques em aproximadamente, no mínimo, 88%. Entretanto, esses métodos atingem esse percentual utilizando padrões geométricos extremamente complexos e envolvendo mudanças substanciais no rosto de uma pessoa, o que nem sempre pode ser factível de ser implementado na vida real. Por outro lado, investigamos neste trabalho ataques com padrões geométricos significativamente mais simples do que aqueles investigado pelos métodos de estado da arte. Com a nossa abordagem, nós mostramos que ainda é possível atacar essas redes, embora com um percentual de sucesso menor (34%), mas com a vantagem que o processo de otimização se torna mais fácil. Os padrões geométricos descobertos são compostos por formas geométricas simples e visíveis ao olho nu, ao contrário do que acontece no estado da arte quando frequentemente pixels individuais precisam ter suas cores alteradas para valores específicos. Nós demonstramos os resultados empiricamente atacando uma rede neural de convolução treinada no conjunto de dados VGGFace2, o qual envolve aproximadamente 9000 classes. Embora esse trabalho seja apenas uma investigação preliminar, é direcionado que abordagens complementares, com aquela já existentes na literatura, podem ser bem-sucedidas mesmo que envolvam padrões de ataque mais simples que os tipicamente investigados.

Palavras-chave: Redes neurais. Ataques adversariais. Reconhecimento facial.

An Adversarial Attack Method on Convolutional Neural Networks for Facial Recognition

ABSTRACT

Nowadays neural network technologies allow for facial recognition activities with a performance never seen before. Face recognition is being used for social media purposes, for example, for recognition of people who are present in a photograph. However, it is also being used by police departments to identify protesters. This has been causing a debate as invasion of privacy issues and there is a growing movement of people trying to develop ways to avoid being subject to a constant surveillance system. Many studies have emerged after the discovery that neural networks are easily fooled by minor changes not seen with the naked eye. In this paper we investigate a variant of attack that consists of using an optimization approach to discover a simple geometric set of forms of different scales, positions and colors that, when placed on the person's face, causes the neural network to misclassify its input. State-of-the-art methods have a success rate of attacks of at least 88%. However, these methods achieve this percentage using extremely complex geometric patterns and involving substantial changes in a person's face which may not always be feasible to implement in real life. On the other hand we investigated in this work attacks with significantly simpler geometric patterns than those investigated by state of the art methods. With our approach we have shown that it is still possible to attack these networks even with a lower percentage of failure (34%) but with the advantage that the optimization process becomes easier and that the discovered geometric patterns, which are capable of attacking the networks, are composed of simple geometric shapes visible to the naked eye as opposed to what happens in the state of the art when often individual pixels need to have their colors changed to specific values. We demonstrate the results empirically by attacking a trained convolution neural network in the VGGFace2 dataset which has approximately 9000 classes. Although this work is only a preliminary investigation, it is intended that complementary approaches such as those already available in the literature may be successful even if they involve simpler attack patterns than those typically investigated.

Keywords: Neural Networks. Adversarial Attacks, Face Recognition.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

RNC	Rede Neural Convolutiva
RN	Rede Neural
CTC	Camada totalmente conectada
DE	*Differencial Evolution*
MNIST	*Modified National Institute of Standards and Technology*

SUMÁRIO

1 INTRODUÇÃO

As atividades envolvendo aprendizado de máquina sofreram um enorme impulso nos últimos tempos, destacando-se em áreas como reconhecimento de fala, jogos, visão computacional e processamento de imagens. Em algumas dessas aplicações, sua performance pode ser superior a de um ser humano. Na área de processamento de imagens, as redes neurais convolutivas (RNC) têm se mostrado bem-sucedidas, sendo utilizadas como reconhecedores de espécies de plantas e animais, por exemplo (CHEN et al., 2014).

Apesar dos benefícios do uso de RNCs para classificação de imagens, descobriu-se casos em que pequenas perturbações nas imagens podem fazer classificadores efetuarem predições errôneas (DALVI et al., 2004). Tais fenômenos podem ser visíveis ou não ao olho humano: perturbações visíveis são alterações que podem ser percebidas a olho nu e as não-visíveis são os casos em que a alteração nos pixels da imagem não causa mudança observável em uma fotografia.

Estudos recentes mostraram que é possível, por exemplo, utilizar uma técnica de *poisoning* em um conjunto de dados (GU et al., 2017). O objetivo desta técnica é adicionar aos dados de treinamento um padrão de pixels alterados manualmente de modo que, durante a validação, o classificador faça predições arbitrárias escolhidas pelo atacante. Outro estudo demonstra um caso em que um simples papel, se afixado a uma placa de trânsito, faz com que um carro autônomo classificasse inadequadamente uma placa de pare como sendo uma placa de velocidade máxima permitida (GU et al., 2017). Para ataques físicos, isto é, envolvendo não alterações nos pixels de imagem, e, sim, no objeto sendo fotografado, um estudo demonstrou a possibilidade de enganar uma rede neural ao inserir perturbações em óculos desenhado em uma imagem. A Figura 1.1 mostra o caso em que, ao adicionar um padrão geométrico e de cores cuidadosamente otimizado um óculos na imagem de Kaylee Defer, a rede neural a classifica como sendo Nancy Travis. (SHARIF et al., 2016)

Dado este cenário, o presente trabalho tem como objetivo explorar possíveis vulnerabilidades em uma RNC de reconhecimento facial pré-treinada com o conjunto de dados VGGFace2 (CAO et al., 2018). Para isso, estudamos em particular perturbações simples que podem ser vistas a olho nu e replicadas no mundo real. Acreditamos que este problema é relevante, pois métodos alternativos de ataques a redes neurais frequentemente assumem que é possível criar perturbações sofisticadas de pixels com

Figura 1.1: Perturbação inserida apenas na região dos óculos.

Fonte: (SHARIF et al., 2016)

cores arbitrárias dentro de uma determinada região. Embora isso possa parecer possível quando se tem acesso a imagem digital, que será usada no ataque, nem sempre é possível executar esse tipo de ataque, caso, por exemplo, seja necessário imprimir o padrão geométrico do ataque e vesti-lo fisicamente no rosto.

Neste trabalho, vamos investigar até que ponto é possível criar ataques bem-sucedidos a essa rede, sob a suposição de que os padrões geométricos desenvolvidos no ataque vão ser compostos por um pequeno conjunto de formas geométricas visíveis ao olho nu ao invés de grandes padrões compostos por um número elevados de pixels que podem ter seus parâmetros ajustados de forma arbitrária.

O objetivo é encontrar uma série de perturbações em que, ao inserir em uma imagem de entrada, o classificador não faça predições corretamente. As perturbações utilizadas são formas geométricas comuns como quadrados, círculos, triângulos e dois desenhos de um coração e uma estrela. Configurações como cores, posição na imagem e fatores de escala também podem ser alterados e testados. O propósito é encontrar um padrão que faça a probabilidade da classe real diminuir de forma que o classificador não consiga reconhecê-la mais. Os parâmetros destas configurações são alterados por um método de otimização que se encarregará de procurar um conjunto que minimize a classificação correta dadas imagens de uma classe.

Ao encontrar padrões geométricos que enganem as predições da RNC, é possível reproduzir, no mundo real, os mesmos padrões manualmente. A finalidade disto é, por exemplo, usar um destes padrões geométricos para que não seja possível ser identificado por alguma aplicação de reconhecimento facial de maneira confiável. Uma possibilidade seria usar uma padrão destes para evitar ser reconhecido em eventos como protestos ou simplesmente não ser reconhecido por câmeras públicas por questões de privacidade.

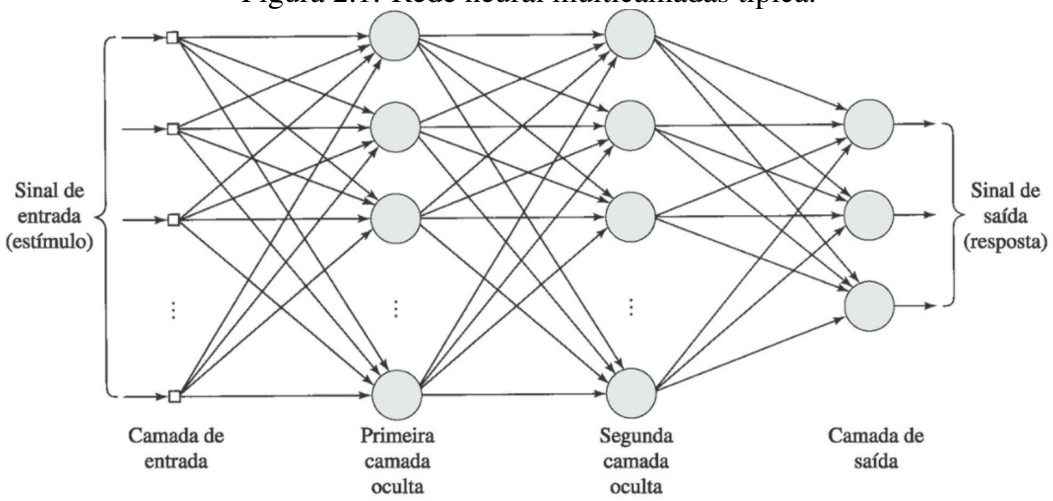
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, veremos os principais tópicos sobre os algoritmos utilizados neste trabalho. Faremos uma revisão sobre conceitos de redes neurais e redes neurais de convolução. Discutiremos também sobre ataques adversariais a algoritmos de aprendizado de máquina e, por fim, será apresentado brevemente o método de otimização proposto por Powell, que utilizaremos para otimizar nossos ataques adversariais.

2.1 Redes Neurais

Redes neurais artificiais são modelos matemáticos que procuram mimetizar o funcionamento de um cérebro humano (MCCULLOCH; PITTS, 1943). Uma rede neural é representada por um grupo de nodos interconectados da mesma forma como em um grafo direcionado. Esses nodos são denominados neurônios e suas conexões, chamadas de sinapses, possuem um valor associado chamado de peso. O valor de saída de um neurônio é determinado pela aplicação de uma função de ativação na soma dos produtos de seus neurônios conectados pelos valores das sinapses correspondentes. Uma função de ativação típica é, por exemplo, a função sigmóide $f(z) = \frac{1}{1+e^{(-z)}}$. A Figura 2.1 representa a arquitetura de uma rede neural artificial de exemplo. Neurônios são agrupados em camadas e se interconectam através das sinapses.

Figura 2.1: Rede neural multicamadas típica.



Fonte: (HAYKIN, 2007)

Uma rede neural comumente utilizada é a *Feed Forward Network*. Neste modelo, as informações contidas nos neurônios na camada de entrada se movimentam em direção

a camada de saída, passando por camadas intermediárias, ou camadas escondidas (GOODFELLOW et al., 2016). Sendo um modelo de aprendizado supervisionado, uma rede neural precisa ser treinada para encontrar os melhores pesos capazes de minimizar o erro dos neurônios de saída dado um determinado conjunto de entrada. Dentre várias técnicas para encontrar os pesos ótimos, uma bastante empregada é o *Backpropagation*.

Dado um conjunto de treinamento, o *Backpropagation* tem como procedimento realizar um passo de *Feed Forward* e comparar a saída da rede neural com o valor esperado, calculando o erro entre esses valores. Este erro é repassado da camada de saída para as camadas anteriores com o objetivo de ajustar os pesos dos neurônios. O processo é repetido até que algum critério de parada previamente definido seja atingido (BRYSON et al., 1963).

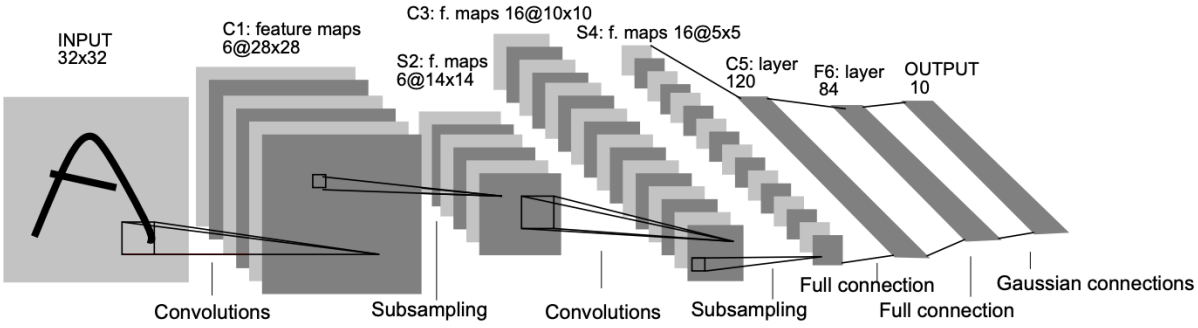
2.2 Redes Neurais Convolutivas

Redes neurais convolutivas (RNCs) são usadas em inúmeras aplicações como classificações de imagem, reconhecimento de vídeo e processamento de linguagem natural. São como as redes neurais clássicas, no entanto, são adicionadas camadas de convoluções (GOODFELLOW et al., 2016).

Tais camadas têm como objetivo identificar padrões de importância de alto nível, como bordas em uma imagem, por exemplo. Os filtros (ou *kernels*) apropriados para utilização são aprendidos pela RNC. Isso acaba se tornando uma vantagem, pois não é necessário informar esta informação previamente, de forma manual. Normalmente, após a camada de convolução, há uma camada de *Pooling*, que é responsável por fazer a redução espacial dos dados de entrada, diminuindo o poder computacional exigido para o processamento e auxiliando em selecionar dados dominantes da entrada após a convolução (GOODFELLOW et al., 2016). Não há limites ou um número fixo sobre quantas destas camadas são necessárias para otimizar a performance de uma CNN.

Ao final da arquitetura, há uma ou mais camadas totalmente conectadas (CTC), as quais recebem o resultado deste pré-processamento de extração de padrões, por parte das camadas de convolucões. Uma CTC possui a mesma arquitetura de uma rede neural multicamadas típica e a mesma finalidade: propagar seus dados até a camada de saída e produzir uma classificação. A Figura 2.1 demonstra um exemplo de uma RNC.

Figura 2.2: Arquitetura de uma RNC, Lenet-5, para reconhecimento de dígitos.



Fonte: (LECUN et al., 1998)

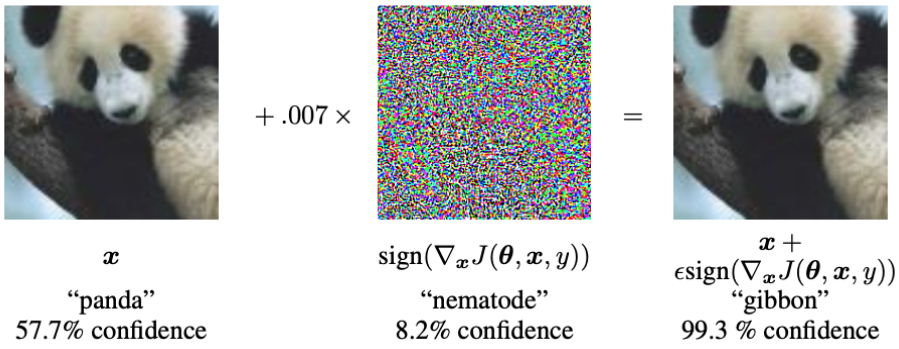
2.3 Ataques Adversariais a Algoritmos de Aprendizado de Máquina

Algoritmos de aprendizado de máquina são vulneráveis a ataques de manipulação de dados de entrada com o objetivo de gerar classificações incorretas. RNCs, como classificadores de imagens, podem ser vítimas deste tipo de ataque ao adicionarmos pequenas perturbações às imagens de entrada, mesmo que tais alterações sejam imperceptíveis (SZEGEDY et al., 2013).

O problema é descrito como segue: seja M um modelo de aprendizado de máquina e $Input$ um exemplo de entrada legítimo. $Input$ é classificado corretamente por M, tal que $M(Input) = y$, onde y é classificação correta associada ao $Input$ no modelo M. Um exemplo adversarial Adv é construído a partir de $Input$ adicionando perturbações E, de forma que $Adv = Input + E$. Visualmente, pode não haver diferenças entre $Input$ e Adv, no entanto, $M(Adv) \neq y$, ou seja, mesmo Adv sendo igual a $Input$ ao olho humano, M classifica Adv incorretamente (KURAKIN et al., 2016). Além disso, a mesma perturbação em $Input$ pode fazer com que outro classificador, treinado com outros conjuntos de dados, também faça classificação incorreta (SZEGEDY et al., 2013). A Figura 2.3 apresenta um exemplo de perturbações não vistas a olho humano.

No exemplo discutido acima são possíveis dois objetivos, que chamaremos, neste trabalho, de ataques direcionados e não-direcionados. Um ataque não-direcionado consiste na otimização de uma perturbação E, que faz com que a rede neural classifique incorretamente a imagem, independente de qual classe ela iria produzir, ou seja, basta que $M(x + E) \neq y$. Um ataque direcionado, por outro lado, corresponde ao problema de encontrar uma perturbação E que faz com que a rede não apenas erre a classificação, mas que seja capaz de garantir que a rede gere uma saída y' específica determinada pelo atacante, ou seja, $M(x + E) = y'$.

Figura 2.3: Adicionando pequenos ruídas a imagem do panda, a rede neural o classfica como um gibão.



Fonte: (GOODFELLOW et al., 2014)

A possibilidade de ataques desse tipo pode gerar impacto na segurança de aplicações que utilizam algoritmos de aprendizado de máquina. A propriedade de transferibilidade evidencia que é possível criar exemplos maliciosos e realizar classificações incorretamente, sem necessitar de acesso ao modelo ou informações do classificador (SZEGEDY et al., 2013).

2.4 Métodos de Otimização de Busca Local

Nesta seção, nós discutiremos um método de otimização em particular, chamado método de Powell, que pode ser usado para maximizar ou minimizar funções. No contexto do nosso trabalho, esse tipo de método de otimização é importante, dado que implementa um tipo de algoritmo que pode ser utilizado para otimizar os padrões envolvidos nos ataques. Embora muitas outras técnicas de otimização diferentes possam ser usadas para encontrar os parâmetros da perturbação envolvidas no ataque, nesse trabalho específico nós utilizamos esse método.

A motivação para escolha deste método se dá pela simples utilização e porque não exige conhecimento do gradiente da função sendo otimizada, que é o que normalmente acontece quando se tenta atacar essas redes neurais.

Assuma, por exemplo, que iremos tentar atacar uma rede neural adicionando a uma determinada imagem um pequeno círculo de raio, cor e posição configuráveis. O método de otimização pode ser utilizado para testar diferentes configurações deste círculo para cada um obtendo uma medida de performance, por exemplo, uma medida de erro de classificação cometido pela rede quando apresentada essa imagem perturbada.

O método de Powell (POWELL, 1964) é um algoritmo de otimização que não requer restrições ou informações como gradientes da função de minimização. Esta técnica procura minimizar a função objetivo f localmente aproximando por uma função quadrática. Dado uma solução candidata inicial X_0 composta por N parâmetros, o algoritmo determina um conjunto de vetores base com a orientação de onde um mínimo está. Desta forma, o método cria uma sequência $X_0 = P_1, P_2, P_3, ..., P_N = X_1$, em que P_i são os pontos entre X_0 e X_1 é o mínimo desta direção partindo de X_0. O processo é repetido em X_1 até encontrar uma solução candidata X_k em que não exista mais possibilidade de realizar movimentos (MATHEWS et al., 2004).

3 TRABALHOS RELACIONADOS

Neste capítulo, serão apresentados alguns artigos que estão relacionados com este trabalho.

3.1 Ataques Adversariais em um Único Pixel

No trabalho proposto por Su et al. (SU et al., 2019), é estudada uma forma de alterar imagens causando perturbações em apenas um pixel, de modo a impedir a correta classificação feita por uma RNC, em uma forma de ataque não-direcionado.

O ataque é descrito como um problema de otimização com restrições. Dada uma imagem x e um conjunto de pixels $x = (x_1, x_2, ...x_n)$, $f_t(x)$ é a probabilidade de x ser da classe correta t. Seja $e(x) = (e_1, e_2, ..., e_n)$ perturbações adicionadas em x, L é a perturbação máxima de $e(x)$ em x, tal que, neste trabalho, $L = 1$. Desta forma, dado que $f_{adv}(x)$ é a probabilidade de x ser de uma classe incorreta, o objetivo é encontrar um $e(x)$ em que $f_{adv}(x + e(x))$ e $f_t(x) \neq f_{adv}(x)$ (SU et al., 2019).

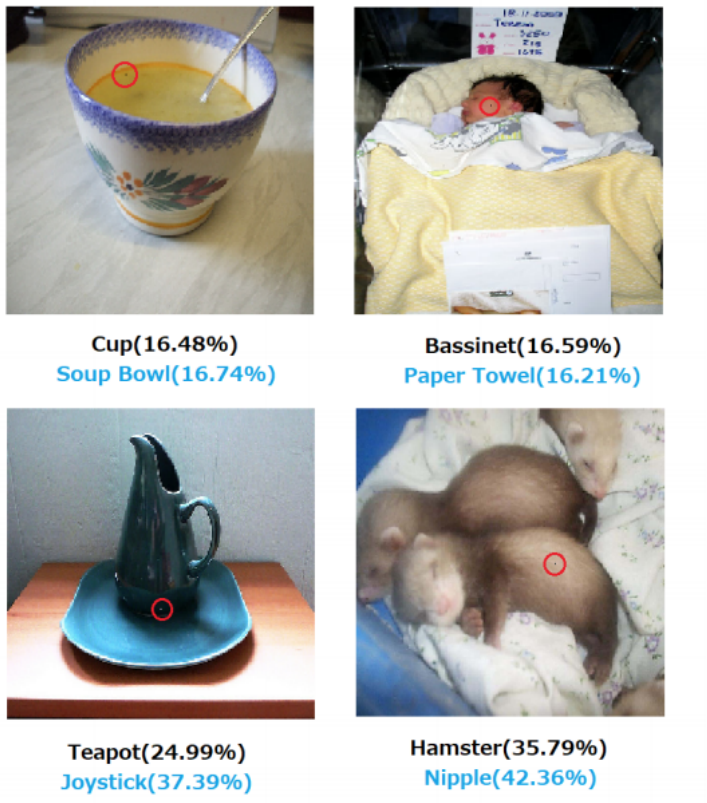
O método de otimização para encontrar perturbações candidatas utilizado foi o DE, *Differential Evolution*, que se caracteriza por elementos como: alta probabilidade de encontrar um ótimo global; simplicidade, uma vez que se faz necessário somente as probabilidades da classificação para o funcionamento do algoritmo, e menor necessidade de informações do sistema que se quer atacar, pois o DE não requer derivadas como os métodos de gradiente descendente ou quasi-newton (SU et al., 2019).

A estrutura da solução baseia-se em um vetor de 5 posições: coordenadas (x, y) e o valor RGB da perturbação. Na população inicial, são geradas 400 soluções candidatas, e, usando o algoritmo usual do DE, são gerados novos 400 indivíduos que competirão com os pais, e a perturbação com melhor *fitness* passará para a nova geração. O método foi utilizado com os conjuntos de dados CIFAR-10 e ImageNet. A Figura 3.1 demonstra ataques bem-sucedidos do *one pixel attack* (SU et al., 2019).

O sucesso do ataque se dá quando, ao adicionar a perturbação à imagem, a classificação do modelo prediz uma classe qualquer que não seja a correta (o caso de ataques não-direcionados), ou prediz a classificação que o atacante desejava no ataque, no caso de ataques direcionados (SU et al., 2019).

A vantagem deste método é que dada a perturbação mínima deste ataque, dificilmente será possível perceber visualmente que a imagem foi atacada, especialmente

Figura 3.1: Ataques a imagens com perturbações em apenas um pixel.



Fonte: (SU et al., 2019)

em imagens com resoluções grandes. A desvantagem é que uma mudança deste nível só é viável possuindo acesso físico ao arquivo de imagem que será enviada para a RNC, pois é praticamente inviável uma alteração manual em nível de pixels. Outra desvantagem é que a performance do ataque cai conforme as dimensões das imagens de entrada do modelo atacado aumentam.

3.2 Ataques Contra Transferência de Aprendizado

O trabalho de Gu et al. (GU et al., 2017) explora o conceito de redes neurais com um *backdoor* incluído da fase de treinamento. O objetivo é envenenar um modelo com amostras de imagens com perturbações não visíveis ao olho humano para que o modelo faça classificações de acordo com o padrão de ataque otimizado por esse *backdoor*.

Neste ataque, o modelo deve manter a performance usual, seja com os dados originais ou com os que tiveram o *backdoor* inserido. Os dados envenenados devem manter algum segredo escolhido pelo atacante, o qual é chamado de *backdoor trigger*. (GU et al., 2017)

Primeiramente, foi feito um experimento com o conjunto de dados MNIST, conjunto de imagens com dígitos de 0 a 9 manuscritos. Dois tipos de *backdoor* foram testados: *single pixel backdoor* e *pattern backdoor*. Para o *single pixel backdoor*, foi determinado apenas pixel como *backdoor trigger*, enquanto o *pattern backdoor* determinava um conjunto de pixels com a mesma finalidade. Em cada tipo de *backdoor*, dois tipos de ataques foram testados: *single target attack* e o *all-to-all attack*. O *single target attack* consistiu em classificar uma imagem que possuía um número i como sendo um número j, tal que $i \neq j$ e $i, j \in [0, 9]$. No *all-to-all attack*, o *backdoor trigger* faz com que a imagem de um determinando dígito i seja classificado como $i + 1$. Foi ressaltado que, a medida que novos exemplos com *backdoor* são adicionados aos dados de treinamento, a frequência de erros nas imagens originais aumentava enquanto os de imagem com *backdoor* diminuía (GU et al., 2017). A Figura 3.2 exemplifica como os *backdoor triggers* são inseridos.

Figura 3.2: Exemplo de *backdoors*. À esquerda, a imagem original, no centro, imagem com um *single pixel backdoor* e, à direita, um exemplo de *pattern backdoor*.

Fonte: (SU et al., 2019)

Adicionalmente, um estudo de caso foi realizado com um conjunto de placas de trânsito. No modelo criado, os autores consideraram três classes: placas de pare, de velocidade máxima permitida, e de advertência. Os *backdoor triggers* deste sistema foram o desenho de um quadrado amarelo, de uma flor, de uma bomba, e foram incluídos visivelmente nas imagens das placas. A Figura 3.3 exemplica os *backdoors* inseridos nas imagens de entrada. Os métodos de ataque utilizados foram *single target attack*, que implica fazer imagens de pare serem classificadas como placas de velocidade máxima, e *random target attack*, cujo objetivo é selecionar uma classe qualquer que não seja a correta (GU et al., 2017).

A vantagem desse método é tornar possível a realização de ataques sem que a acurácia do modelo seja severamente alterada, ou seja, a confiabilidade permanece semelhante. No entanto, é necessário possuir acesso aos dados de treinamento para inserir as amostras com o *backdoor* desejado.

Figura 3.3: Placas de trânsito com imagens responsáveis por agirem como *backdoor triggers*. Respectivamente: imagem original, quadrado amarelo, bomba e flor.

Fonte: (SU et al., 2019)

3.3 Ataques a Redes Neurais Fisicamente Viáveis

Diferentemente dos trabalhos citados anteriormente, o trabalho de Sharif et al. (SHARIF et al., 2016) procura realizar ataques que sejam possíveis de serem aplicados fisicamente. Os ataques são de maneira direcionada e não-direcionada. Para este problema, os autores tomaram os seguintes passos para construção do algoritmo:

- Utilizar acessórios faciais (contendo os padrões envolvidos no ataque) que sejam facilmente impresso em impressoras 3D ou 2D.
- Possibilidade de encontrar uma fórmula matemática que encontre perturbações que sejam robustas a pequenas variações, sejam suaves ao olho humano, e que sejam reprodutíveis utilizando impressoras convencionais.

O acessório escolhido para conter o padrão geométrico do ataque foi óculos, uma vez que é um acessório comum que pessoas vestem. Os óculos selecionados representavam cerca de 6.5% de todos os pixels das imagens de rosto (224 x 224), ou seja, a perturbação total do ataque seria de no máximo 6.5% das imagens (SHARIF et al., 2016).

A técnica para encontrar as perturbações consistia em inicialmente pintar os óculos de cor amarela e utilizar um algoritmo de gradiente descendente para encontrar as cores adversariais. Para obter robusteza a pequenos movimentos, durante cada iteração, os pixels do acessório eram aleatoriamente movidos vertical, horizontal e rotacionalmente. As perturbações criadas também eram generalizadas, ou seja, provocavam equívoco do classificador em uma quantidade de imagens do atacante, e não apenas em uma imagem particular sendo atacada.

A robusteza a pequenos movimentos é relevante, pois, se a pessoa que estiver

vestindo os óculos do ataque se movimentar, isso alterará a imagem que será fornecida para rede neural. Se o padrão geométrico for robusto a movimentações, mesmo que a imagem seja capturada de uma forma ou a partir de um ângulo um pouco diferente do original, ainda assim ela vai resultar num ataque bem-sucedido. Isto é importante, pois nem sempre se garante que a pessoa que vai vestir o padrão de ataque vai ser fotografada sobre as mesmas condições (posição, iluminação, etc.).

As redes neurais utilizadas foram o VGGFace (PARKHI et al., 2015) e outros dois modelos treinados via aprendizado por transferência. Por se tratar de um ataque físico, a primeira rede neural citada não era ideal para o experimento, uma vez que não era possível realizar testes com pessoas presentes no conjunto de dados, pois os autores não poderiam solicitar que eles vestissem os óculos de ataque. Portanto, as outras redes foram treinadas para incluir pessoas que fossem disponíveis para tais testes. Os autores calcularam as perturbações necessárias para os óculos e então os vestiram. A Figura 3.4 demonstra como os ataques foram realizados.

Figura 3.4: Ao vestir os óculos, o classificador é enganado. Fig. (a) ataques não-direcionados. Fig. (b), Fig. (c) e Fig. (d) ataques direcionados, em que o rosto do topo é classificado como sendo o rosto de baixo.

(a) (b) (c) (d)

(SHARIF et al., 2016)

As vantagens deste método são a robusteza do ataque a pequenas variações de movimentos e a simplicidade em vestir um acessório facial comum, ao invés de assumir acesso à imagem digital sendo atacada. No entanto, as perturbações não são facilmente replicáveis, sendo necessário equipamentos específicos que consigam imprimir exatamente aquele padrão estabelecido pelo algoritmo.

3.4 Discussão

O método que propomos no presente trabalho tem um objetivo bastante similar discutido nesta subseção. Em particular, gostaríamos de identificar padrões que sejam visíveis ao olho nu e que possam ser aplicados em ataques factíveis da vida real. Ao contrário do método proposto (SHARIF et al., 2016), nosso objetivo é que os padrões geométricos não sejam complexos. Como pode ser visto na Figura 3.4, frequentemente o padrão geométrico que o método otimiza, para ser impresso nos óculos, consiste em um padrão complexo de pixels com cores arbitrariamente diferentes entre si. Isso dificulta a aplicação desse tipo de ataque em situações reais, aonde nem sempre um atacante possui acesso a recursos como impressoras de alta resolução, e aonde nem sempre se pode assumir que a câmera que captura as imagens fornecidas para as redes também sejam câmeras de com tal resolução, capazes de capturar todos os detalhes do padrão de ataque. Para lidar com essas suposições, o presente método, como será discutido no capitulo seguinte, assume que os padrões otimizados de ataque serão compostos, não por um grande conjunto de pixels arbitrariamente complexos e, sim, por um pequeno conjunto de formas geométricas simples, que poderão ser movidas em uma pequena região do rosto e terem suas cores, posições e escalas alteradas.

No capítulo de experimentos, iremos quantificar o quão bem-sucedidos os ataques a rede neurais conseguem ser em função de qual simples ou complexas envolvidas e iremos demonstrar, em particular, que utilizando um pequeno conjunto de formas simples e visíveis a olho nu, ainda assim é possível construir ataques que gerem classificações incorretas.

4 METODOLOGIA

4.1 Método proposto

Neste trabalho, criamos um ambiente para manipulação de imagens utilizando padrões geométricos simples como forma de ataque a uma rede neural. O propósito disso era encontrar parâmetros de cores, fatores de escala e posições destes padrões que, inseridos na imagem, fizessem com que a rede neural não fosse capaz de classificar a imagem corretamente.

Para isto, foram definidos 10 padrões de ataques compostos por formas geométricas simples. Os parâmetros iniciais, descritos anteriormente, eram gerados aleatoriamente e um otimizador se encarregava de encontrar novos parâmetros que minizassem a probabilidade da classe correta. A finalidade disto era que, ao minimizar a probabilidade da classe da imagem de entrada, outra classe fosse definida pela rede neural como correta. A imagem era submetida a cada um dos 10 padrões de ataque e otimizados com 3 *restarts*, uma vez que o otimizador navegava pelo espaço de soluções a partir da solução inicial gerada.

4.1.1 Criação e Manipulação de Formas Geométricas

Com objetivo de atacar imagens de uma determinada classe, máscaras foram criadas e aplicadas às imagens de entrada do modelo. Uma máscara, neste trabalho, é uma imagem de mesmas dimensões da imagem de entrada que conterá as figuras geométricas e suas respectivas transformações. Uma máscara contém duas áreas: uma nula, com o RGB $(0, 0, 0)$, representando a região da imagem aonde padrões de ataque não podem ser posicionados, e uma não-nula, com RGB $(255, 255, 255)$, representando a região da imagem aonde padrões de ataquem podem ser posicionados. Os padrões geométricos de ataque são desenhados apenas dentro da área não-nula, sendo truncados pela área de cor preta caso alguma das transformações exceda a área de cor branca. Assim, a área não-nula é dimensionada para que coincida com a região em que iremos fazer o ataque. A Figura 4.1 representa o exemplo de imagem utilizada para o ataque. A Figura 4.2 demonstra um exemplo da área de ataque especificada via máscara para a face da Figura 4.1.

Neste trabalho, como mencionado anteriormente, nós investigamos a eficácia de

Figura 4.1: Exemplo de imagem de entrada.

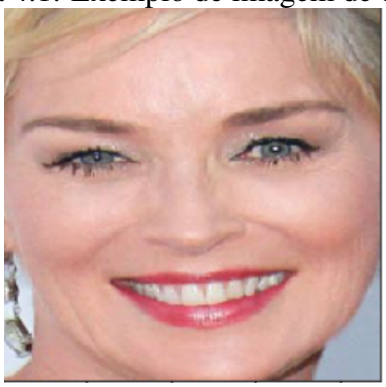
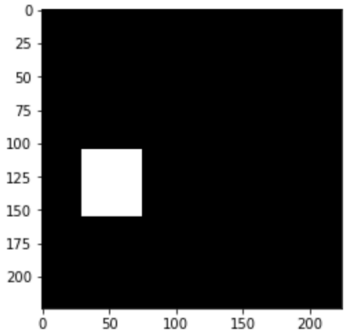


Figura 4.2: Exemplo de máscara para Figura 4.1. Note que a área branca é a região de ataque, neste exemplo, a bochecha.



ataques que consistem na aplicação de um pequeno conjunto de formas geométricas sobre o rosto de uma determinada pessoa. Particularmente, dizemos que um padrão geométrico de ataque é composto por n formas geométricas. As formas geométricas simples que podem ser utilizadas correspondem a pequenos círculos, triângulos, quadrados e desenhos na forma de coração e de estrela. Por simplicidade, consideraremos os desenhos de coração e estrela como padrões geométricos. Cada um deles possui 4 parâmetros: a tripla RGB que contém a cor da forma e o fator de escala. O padrão geométrico de ataque possui 2 parâmetros, as posições x e y da localização de onde forma é desenhada dentro da máscara. Os padrões geométricos simples disponíveis são exemplificados pelas Figuras 4.3 a 4.7.

De maneira padronizada, as formas são criadas com dimensões 12 x 12. Um quadrado é desenhado colorindo todos os pixels de sua área com um RGB. Para o círculo, calcula-se o centro da forma e o raio como a metade da dimensão da forma. Com esses valores ajustados, é percorrido toda a área da imagem e uma função $f(x, y)$ com a equação reduzida da circunferência é utilizada para colorir o círculo. A função 4.1 apresenta a função da equação reduzida da circunferência utilizada. A coordenada (x, y) é o pixel da imagem, a coordenada (x_0, y_0) é centro da circunferência e r é o comprimento do raio do

Figura 4.3: Exemplo de círculo - uma das formas que pode ser utilizada para ocupar a região de ataque.

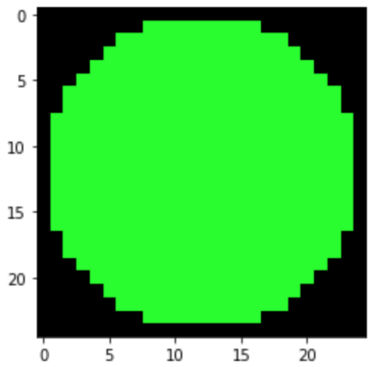


Figura 4.4: Exemplo de triângulo - uma das formas que pode ser utilizada para ocupar a região de ataque.

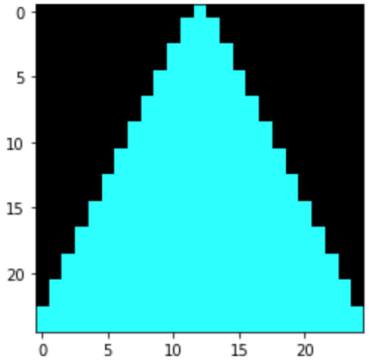


Figura 4.5: Exemplo de quadrado - uma das formas que pode ser utilizada para ocupar a região de ataque.

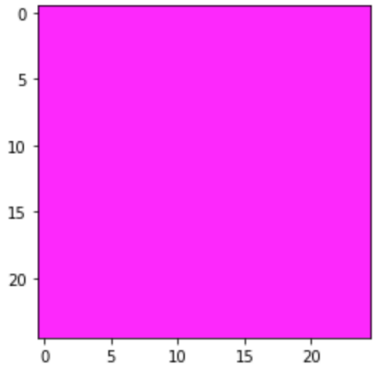


Figura 4.6: Exemplo de coração - uma das formas que pode ser utilizada para ocupar a região de ataque.

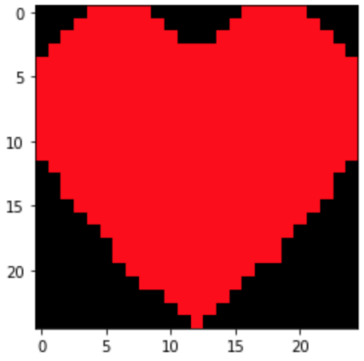
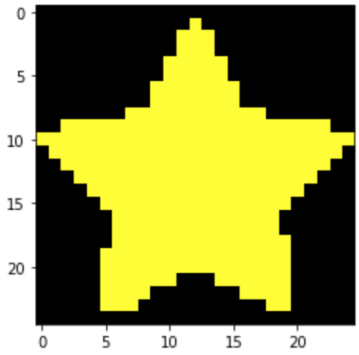


Figura 4.7: Exemplo de estrela - uma das formas que pode ser utilizada para ocupar a região de ataque.



círculo. Se $f(x, y) < 0$, então o pixel é colorido.

$$f(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2 \tag{4.1}$$

Para o triângulo, três pontos são manualmente selecionados, uma função da biblioteca *openCV* faz a ligação entre estes pontos e colore o seu interior. As cores de todas as formas são aleatoriamente geradas para compor o padrão de ataque candidato a ser atingido pelo nosso método. Os três canais assumem valores aleatórios no intervalo de $[1, 254]$.

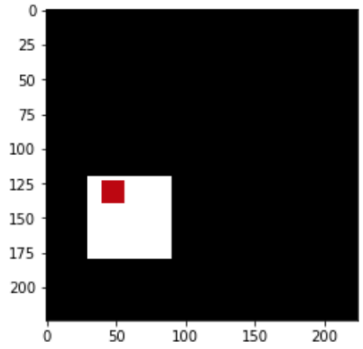
Os padrões no formato de coração e de estrela, por outro lado, são importados de um arquivo de imagem. Ambos padrões podem ter seus parâmetros de cor e escala modificados.

Após a criação de um padrão simples, ocorrem as transformações geométricas. Para escala inicial da forma, na solução de ataque inicial a ser atingida, gera-se um fator de escala randômico entre 0.3 e 1.5. Este intervalo tem os seguintes propósitos: evitar valores pequenos demais, uma vez que queremos que o padrão seja claramente observável, e evitar valores muito grandes, pois o padrão poderia ficar muito grande, cobrindo uma grande região do rosto. Com este fator, as novas dimensões da figura são calculadas e então redimensionadas. O resultado desta etapa é o padrão com o fator de escala aplicado. Algumas formas foram rotacionadas para prover diversidade nos experimentos.

Para o posicionamento inicial da forma, escolhe-se uma coordenada (x, y) aleatória dentro da área não-nula. O processo de cópia da forma sobre a imagem sendo atacada é realizada pixel a pixel. Se o pixel de destino na máscara for um pixel nulo, este não será copiado. A Figura 4.8 demonstra uma máscara pronta para ser utilizada.

No caso do padrão de geométrico de ataque ser composto por várias formas, o processo acima é repetido para cada um delas, a fim de se calcular o padrão final de

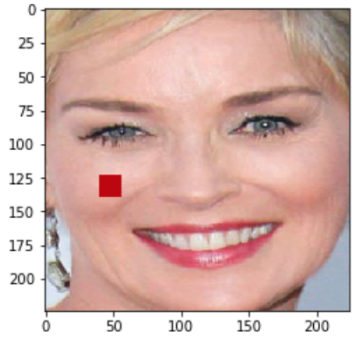
Figura 4.8: Cópia final na máscara. Todo pixel da forma que possuir como destino a região da área não-nula da máscara não será copiado.



ataque a ser posicionada sobre a imagem.

Após a conclusão da máscara, são copiados todos os pixels não-nulos e que não possuem valor $(255, 255, 255)$ diretamente sobre imagem de entrada. A Figura 4.9 mostra um exemplo de uma máscara pronta sendo aplicada sobre uma imagem a ser atacada.

Figura 4.9: Máscara aplicada na imagem de entrada.



Durante a otimização de padrões de ataques, as formas não devem sobrepor em regiões críticas do rosto como olhos, boca e nas narinas. O motivo de evitar essas regiões é pela dificuldade física que o padrão causaria por estar posicionado nestes locais, como dificuldade de enxergar, de respirar ou de falar. Caso a pessoa na imagem esteja utilizando algum acessório, por exemplo, óculos, tal objeto poderá ser atingido pelos padrões geométricos. Para isto, as máscaras deverão conter áreas nulas nas regiões críticas a serem evitadas para que as formas não sejam copiadaa nestes locais. Máscaras foram criadas manualmente com um editor de imagens, em nossos experimentos.

4.1.2 Padrões de Ataques Adversariais

Dado o objetivo deste trabalho de identificar padrões de ataques simples, nós utilizamos a seguinte metodologia: enumeramos manualmente 10 tipos de padrões

geométricos de ataque que podem ser aplicados sobre uma imagem de ataque. Alguns padrões geométricos são mais simples, pois são compostos por apenas uma forma (por exemplo, apenas por um coração), e outros padrões são levemente mais complexos, sendo compostos, por exemplo, por um conjunto de 4 círculos. As combinações foram criadas para serem simples e práticas. Por esta razão, não há formas geométricas diferentes dentro do padrão de ataque. Pequenas variações foram adotadas entre os padrões definidos para avaliar se essas mudanças são relevantes para o nosso ataque. Cada um destes 10 padrões geométricos de ataque tem parâmetros configuráveis. O posicionamento de cada forma dentro do padrão é fixo e predeterminado. Cada forma dentro do padrão geométrico de ataque possui sua cor e fatores de escala regulados como parâmetros a serem otimizados. O padrão geométrico de ataque a ser aplicado na foto tem sua posição (x, y). A Figura 4.10 apresenta os 10 tipos de padrões de ataque que utilizamos em nosso método. O fator escala de todos os padrões geométricos, para esta representação, foi mantido com valor 1.

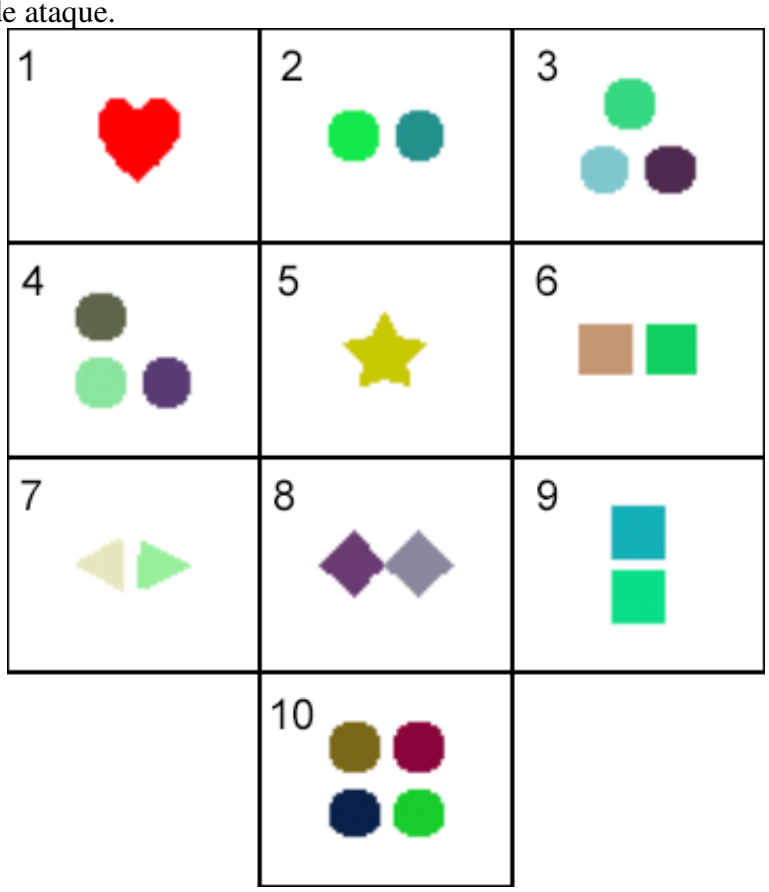
A escolha das formas teve como propósito a reprodutibilidade no mundo real, a facilidade em serem impressas e fisicamente vestidos por uma pessoa e na área total coberta pelo padrão.

4.1.3 Encontrando Novos Parâmetros Candidatos e Avaliação

Para encontrar conjuntos de parâmetros dos padrões de ataque, foi utilizado um otimizador baseado o algoritmo modificado de Powell (POWELL, 1964). A cada iteração do otimizador, os parâmetros sugeridos pelo método foram utilizados para criar a uma nova máscara candidata e aplicou-se na imagem de entrada. Para verificar a performance daquele conjunto de parâmetros, a imagem atacada com a nova máscara foi enviada ao modelo para realizar a predição. O objetivo do método de otimização era minimizar a probabilidade da predição correta, até o ponto em que o modelo passasse a errar a classificação real.

Iremos executar o otimizador sobre um conjunto de k parâmetros os quais regulam as formas e o padrão geométrico de ataque, em que $k = 4n + 2$ e n é o número de formas contido no padrão geométrico de ataque. Os parâmetros são as cores e os fatores de escala de cada forma e a posição do padrão de ataque. Considerando os 10 tipos de padrões geométricos de ataque, temos que o algoritmo de otimização será executado com os parâmetros $[\theta_1, \theta_2, \theta_3, ..., \theta_{k-1}, \theta_k]$, onde $\theta_{4i-3}, \theta_{4i-2}, \theta_{4i-1}$ para $i \in \{1, 2, ..., n\}$

Figura 4.10: Representação dos 10 padrões base de ataque utilizados por nosso método adversariais de ataque.



representam o RGB da forma i, θ_{4i} representa o fator de escala da forma i e θ_{k-1}, θ_k são respectivamente x e y da posição do padrão geométrico de ataque.

Por ser um método de otimização que não utiliza restrições, parâmetros como os valores RGB das formas poderiam facilmente extrapolar valores válidos de seu domínio. Por este motivo, todas as restrições dos parâmetros de cada padrão geométrico foram manualmente codificadas para evitar valores inválidos.

Notamos, por fim, que a busca feita pelo otimizador por conjunto de parâmetros de alta performance se dá no espaço contínuo de valores. Como os parâmetros de posição e de cor são discretos, eles são arredondados ao serem utilizados para gerar novas formas.

4.2 Ferramentas

Nesta seção, discutiremos brevemente sobre todos os recursos para a implementação do método deste trabalho.

4.2.1 Modelo de RNC e Conjunto de Dados

O conjunto de dados utilizado neste trabalho foi o VGGFace2, utilizado para reconhecer pessoas presentes numa determinada foto dentro o conjunto de 9131 pessoas possíveis. A rede neural foi treinada em média com 362 imagens relativas a cada uma destas pessoas. As imagens desde conjunto foram coletadas no *Google Image Search* e possuem variações como iluminação, poses, idades e profissões. (CAO et al., 2018). A arquitetura da RNC utilizada foi o *resnet50* (HE et al., 2016). A configuração desta arquitetura pode ser vista no Apêndice A.

As amostras de imagens para testes foram selecionadas, por classe, aleatoriamente pelo *Google Image Search*. As imagens, antes de serem utilizadas para treinamento ou teste, são pré-processadas com a finalidade de isolar a face do indivíduo contida na imagem e deixar nas dimensões corretas para o modelo. As Figuras 4.11 e 4.12 apresentam exemplos de amostras coletadas e a extração das faces respectivamente.

Figura 4.11: Exemplo de amostra da uma classe do conjunto VGGFace2.

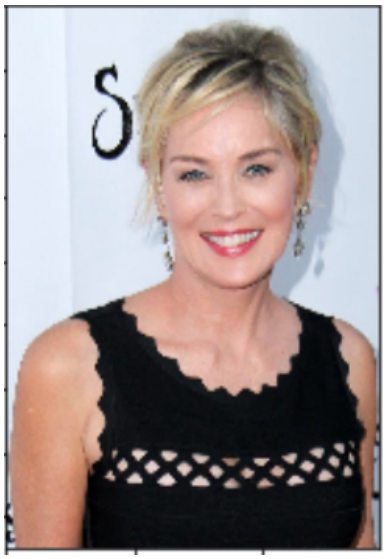


Figura 4.12: Face extraída da Figura 4.11.

4.2.2 Ferramentas de Implementação

A linguagem de programação utilizada foi Python por ser bastante comum na área de aprendizado de máquina. Foram utilizadas as bibliotecas *openCV* (GARCÍA et al., 2015) para desenhar e manipular formas geométricas, *numpy* (WALT et al., 2011) para estrutura de dados, *scikit-learn* (PEDREGOSA et al., 2011) para métodos de otimização e a API *keras* (GULLI; PAL, 2017) para criação, manipulação e treinamento de RNCs.

5 EXPERIMENTOS E RESULTADOS

Para avaliar o método proposto, analisamos a capacidade de cada um dos 10 padrões geométricos de ataque descritos anteriormente (Figura 4.9) de enganar o classificador em 100 imagens do conjunto de dados. Os parâmetros de cada padrão foram alterados pelo otimizador e, a cada iteração, usados para a criação de novas formas de ataque. Todas as 100 imagens foram submetidas previamente ao modelo e possuem probabilidade da classe correta acima de 70%. O objetivo dos experimentos, portanto, é mensurar a capacidade do método de diminuir essa probabilidade suficientemente a ponto de resultar em um erro de classificação.

Cada configuração de parâmetros iniciais dados ao otimizador foi gerada aleatoriamente dentro dos limites descritos previamente. Ao se avaliar o padrão geométrico de ataque 4 (Figura 4.9), por exemplo, se permitiu que o otimizador encontrasse parâmetros para esse padrão em três execuções independentes para mensurar a capacidade média para aquele tipo de padrão quando otimizado de enganar a classificador.

O objetivo do nosso experimento é mensurar o quanto cada um desses tipos de padrões, caso utilizados para compor um ataque, frequentemente conseguem induzir a rede neural ao erro. Particularmente, os experimentos foram conduzidos na seguinte sequência de passos: para cada imagem i, para $i \in \{1, 2, ..., 100\}$ do conjunto de imagens, aplique o padrão de ataque j, para $j \in \{1, 2, ..., 10\}$. Otimize os parâmetros do padrão j na imagem i com o objetivo de minimizar a probabilidade da classe correta i até que o critério de parada do otimizador seja atingido. Repita os passos acima três vezes. O propósito deste processo é verificar três medidas: quantas imagens foram atacadas com sucesso, frequência de ataques bem-sucedidos de cada um dos padrões de ataques e número de iterações do otimizador até o êxito do ataque.

As Figuras 5.1, 5.2, 5.3 apresentam exemplos de ataques bem-sucedidos. À esquerda, temos a imagem original e, à direita, a imagem atacada com êxito. Mesmo que o classificador tenha acertado as identidades das imagens à esquerda, o classificador foi incapaz de identificar os rostos da direita.

Figura 5.1: Anita Lipnicka identificada como sendo Jessica Lange.

Figura 5.2: Wolfgang Niedecken identificado como sendo Alex Van Warmerdam.

Figura 5.3: Sharon Stone identificada como sendo Carolina Dieckmann.

5.1 Resultados dos Casos de Êxito

Da totalidade do conjunto no experimento, 34% das imagens de teste foram atacadas com sucesso, ou seja, o modelo as classificou incorretamente. Para esta contagem, bastou que uma das dez formas conseguisse enganar a rede. A Tabela 5.1 mostra o percentual de ataques bem-sucedidos sobre todas as imagens de teste.

Tabela 5.1: Tabela de ataques bem-sucedidos por execução com *restarts*.

Bem-sucedidos	66.0%
Malsucedidos	34.0%

O experimento acima tinha como objetivo checar o quão frequentemente alguma das dez formas propostas, as quais são significativamente mais simples do que as tipicamente encontradas por métodos do estado da arte, conseguem enganar a classificação da rede. Como esperado, devido ao fato de estarmos motivados por ataques com formas simples, já esperávamos que essas formas com restrição de complexidade não obteriam o mesmo grau de performance que formas arbitrariamente complexas. A taxa de sucesso observada de 34% é de fato inferior a taxa de sucesso do método do estado da arte que permite formas arbitrariamente complexas, por exemplo, no artigo (SHARIF et al., 2016) obtiveram taxas de sucessos de 88%. Embora nossa taxa de sucesso tenha sido menor, conseguimos obter sucesso com formas geométricas simples, mais fáceis de manipular no mundo real

Quando executamos os nossos experimentos para cada um dos padrões de ataque, rodamos o otimizador três vezes a fim de evitar que ele não encontrasse uma solução devido a possibilidade de convergência para mínimos locais. Esse tipo de estratégia de rodar um processo de otimização, a partir de pontos iniciais diferentes várias vezes, se chama técnica de otimização com *restarts*. A fim de estimar o quão suscetível o método de Powell é de convergir pra ótimos locais pobres, nós mensuramos não apenas a frequência que ele consegue atacar a rede neural, no caso de *restarts*, mas o quanto frequentemente ele consegue atacar a rede caso seja executado apenas uma vez. Esses resultados são apresentados na figura. Como é possível ver, quando o algoritmo é executado apenas uma vez a frequências de ataques bem-sucedidos é de apenas 4.7%, se executado com três *restarts*, a frequência cresce para 34%. Portanto, isso evidencia a necessidade de uso de método de otimização robustos que possam lidar com ótimos locais, o que justifica nossa

decisão de fazer a otimização com vários *restarts*. A Tabela 5.2 apresenta uma tabela quando se consideram execuções sem *restarts*.

Tabela 5.2: Tabela de ataques bem-sucedidos por execução sem *restarts*.

Bem-sucedidos	95.3% (2858)
Malsucedidos	4.7% (142)

Há de se considerar, neste resultado, o funcionamento do otimizador. Por depender dos parâmetros iniciais, acreditamos que, durante o processo de busca por novos valores, ótimos locais foram encontrados, impedindo que o otimizador encontrasse uma configuração que minizasse a probabilidade desejada. A taxa de sucesso observada, portanto, é um limite inferior a taxa alcançável ao se utilizar os padrões geométricos propostos, e é influenciado pelo método de otimizacão específico utilizado.

A avaliação da uma imagem pelo modelo de rede neural *resnet50* consome um considerável recurso computacional, visto que o modelo *resnet50* possui ao todo 50 camadas(HE et al., 2016). Uma imagem do conjunto de teste, otimizado três vezes para cada um dos padrões geométricos base. pode requerer até 12 horas de processamento.

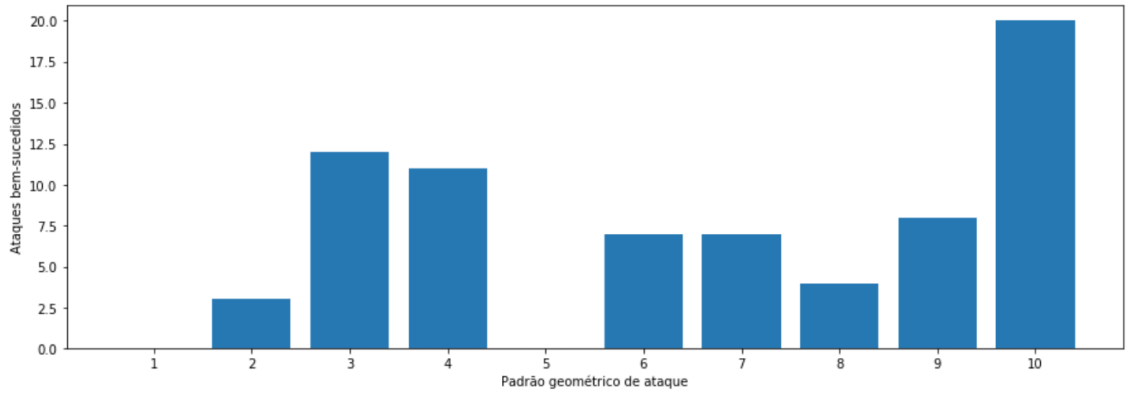
5.2 Resultados por Tipo Padrão Geométrico de Ataque

Em muitos casos, mais de um padrão foi capaz de resultar em um ataque de sucesso. Desta maneira, medimos a performance de cada uma das 10 formas dada a eficiência em atingir o objetivo. A Figura 5.4 mostra a quantidade de vezes em que cada forma atingiu o objetivo de enganar a rede neural com êxito.

Notamos, na Fig 5.4, que os padrões de ataque 1 e 5 não foram eficientes comparados com os demais, possivelmente pelo fato de serem os mais simples de todos, possuindo apenas uma forma em cada padrão de ataque. Mesmo com o fator máximo de escala permitido aplicado sobre o padrão geométrico, gerando a perturbação máxima do padrão de ataque, não foi o suficiente para induzir o classificador ao erro uma única vez. Isto evidencia a relevância do tamanho da perturbação para realizar ataques a algoritmos de aprendizado de máquina.

Os padrões 2 e 8, por outro lado, foram capazes de gerar classificações incorretas. No entanto, obtiveram os resultados mais pobres em relação ao conjunto que atacou o

Figura 5.4: Frequência de ataques bem-sucedidos para cada um dos 10 tipos de padrões geométricos de ataque.



classificador com sucesso. Estas possuíam dois padrões geométricos, resultando em uma área maior do que os padrões 1 e 5. Neste caso, o fator principal pelo sucesso dos ataques poderia ser observado por quão bom foram os parâmetros iniciais aleatórios fornecidos para o otimizador, pois não houve nenhum padrão entre eles que descrevesse os êxitos dos ataques.

Os padrões padrões 3, 4, 6, 7, 9, tiveram desempenho médio. As pequenas diferenças de vitórias possivelmente se devem à aleatoriedade do algoritmo de otimização. Os padrões de ataque 3, 4 obtiveram perfomance superiores provavelmente pela maior complexidade, possuindo três formas.

Já o padrão 10 foi o melhor colocado considerando a eficácia. Por possuír uma área total maior que o restante dos padrões, o otimizador encontrava mais facilmente uma região do rosto própria para maximizar a chance de ser bem-sucedida. Por consequência, fica claro que, para este experimento, a área total é um fator relevante para o seu sucesso.

5.3 Resultados por Número de Iterações

Adicionalmente, foi calculada uma métrica para descrever o momento em que a rede passou a classificar a imagem equivocadamente para cada uma dos 10 padrões de ataque. Para isto, contabilizou-se o número de vezes em que os parâmetros do padrão foram alterados pelo otimizador até a convergência, isto é, o momento em que o classificador passou a se enganar. A Tabela 5.3 demonstra os valores encontrados desta métrica. Os padrões 1 e 5 foram omitidos da tabela, uma vez que não enganaram a rede neural.

Tabela 5.3: Número de iterações necessários pelo processo de otimização até o êxito do ataque.

Padrão de ataque	Mín.	Max.	Média
2	26	430	256.2
3	62	1404	555.0
4	80	971	451.2
6	112	1265	623.1
7	308	2009	713.1
8	69	1008	600.7
9	86	2191	612.3
10	23	1404	422.8

Observando os dados da Tabela 5.3, nota-se que para os padrões 3, 4, 6, 7, 8, 9 e 10, a média de passos de iterações necessárias pelo processo de otimização até a convergência é semelhante, variando 422.8 e 713.1. Isso evidência que o esforço computacional para encontrar parâmetros adequados pra cada uma dessas formas é mais ou menos independente da forma específica, dependendo mais da solução inicial dada ao otimizador. A exceção é a forma 2 que pareceu requerer menos passos. Entretanto, perceba que a forma 2, embora tenha exigido menos passos, também é um das menos bem-sucedidas. Na Figura 5.6 podemos observar que dentre todas as 10 analisadas, a forma 2 é a que possui a pior performance.

O número total de iterações que o otimizador utiliza aumenta linearmente com a quantidade de parâmetros. A convergência aparentemente não ocorre simplesmente por haver mais iterações, sustentando o argumento de que a solução candidata inicial é um dos principais fatores para o sucesso do ataque. Outro caso a ser observado são os arrendondamentos para valores discretos. Isto pode ter interferido nas direções do possível mínimo que o método poderia encontrar.

5.4 Observações

Foi observado que, na maioria dos ataques bem-sucedidos, a região mais atingida foi a região nasal. É possível que os ataques só obtivessem êxito quando os parâmetros de posições iniciais estivessem próximos desta região ou quando alguma iteração gerou

valores de posição próximos desta. Outro ponto notável é que normalmente os valores de RGB envolvidos no ataque convergiam para tons mais escuros. Esta combinação pode demonstrar uma potencial vulnerabilidade na rede neural *resnet50*, a qual pode estar baseando suas classificações muito fortemente com base nas cores da imagem e não tanto com base nas características geométricos no rosto. As Figuras 5.5 a 5.7 demonstram outras evidências de ataques na região nasal.

Figura 5.5: Channing Tatum classificado como sendo Marco Girnth.

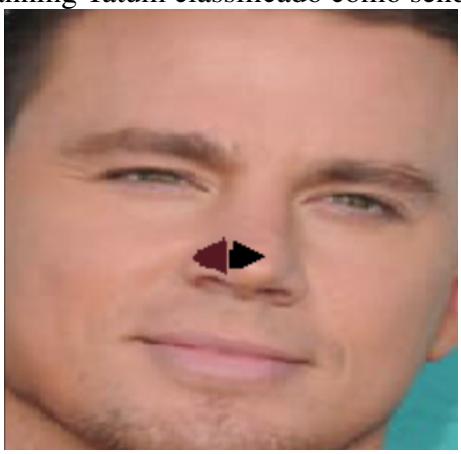


Figura 5.6: James Cameron classificado como sendo Tom Vilsack.

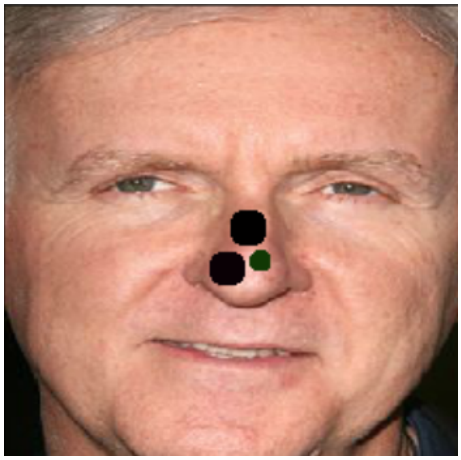
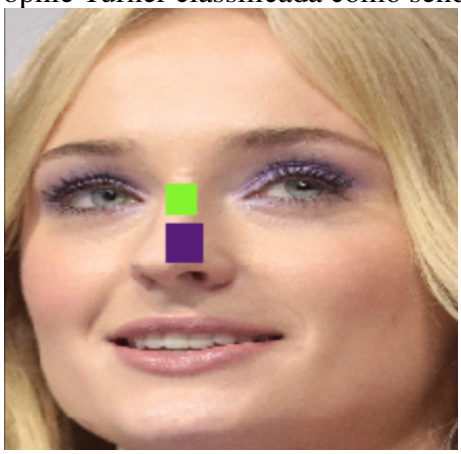


Figura 5.7: Sophie Turner classificada como sendo Pip Pellens.



6 CONCLUSÕES

Redes neurais convolutivas são atualmente importantes devido ao sucesso de suas aplicações, especialmente por realizar tarefas melhor que um ser humano. Desde jogos até a medicina, RNCs estão presentes e são constantemente estudadas. Novos modelos são propostos e, no entanto, propriedades intrigantes podem ser analisadas e estudadas.

Este trabalho propôs uma possível técnica de criação de padrões geométricos de ataque utilizando com objetivo de enganar uma rede neural convolutiva pré-treinada. Em particular, demonstramos empiricamente que essas redes são suscetíveis a ataques mesmo quando os padrões geométricos de ataque são substancialmente mais simples do que aqueles otimizados por redes do estado da arte. Tais redes frequentemente obtém taxas altas de sucesso apenas quando alteram as imagens usando padrões extremamente complexos. Nesse trabalho, demonstramos que mesmo caso os padrões sejam simples advindo de um pequeno conjunto de 10 tipos padrões de formas geométricas, ainda assim é possível enganar o classificador. Em nossos experimentos, analisamos as performances de destes 10 tipos de padrões geométricos candidatos, mensurando quão frequentemente destes padrões simplificados tiveram sucesso em seus ataques, mensurando também o esforço computacional necessário para executar esses ataques e, por fim, mensurando a taxa de sucesso geral do método em enganar a rede utilizando qualquer um dos padrões de ataque candidatos.

O trabalho aqui proposto pode ser melhorado de várias formas. Uma modificação no método proposto por este trabalho é utilizar um método que permita restrições nas variáveis da função objetivo, como o DE, por exemplo, pode ser uma direção a ser tomada. Como o método utilizado neste trabalho é irrestrito, todas as restrições foram feitas via código, o que pode ter prejudicado a exploração no espaço de soluções do algoritmo utilizado. Assim, a implementação com a utilização de um otimizador restrito poderia ser comparado com um outro irrestrito.

Outro caminho sugerido é criação de novos padrões geométricos e desenhos para novos experimentos. Formas mais complexas, como padrões de cores diferentes inscritos em outros, são sugestões pois ainda são possíveis de serem reconstruídas no mundo real. Círculos, triângulos e quadrados de cores únicas podem não ser o suficiente para causar perturbações relevantes em posições arbitrárias.

Uma sugestão é gerar parâmetros iniciais em torno de heurísticas. No nosso método, por exemplo, a posição inicial gerada era completamente aleatória. Um novo

estudo poderia sempre colocar a posição inicial fixamente no centro da imagem, pois é uma região que ficará, na maioria das vezes, próxima em regiões possivelmente sensíveis, como observado neste trabalho, como o nariz dos indivíduos contidos nas imagens. Delimitar o domínio do RGB inicial para valores mais próximos de zero também é uma heurística recomendada.

Por fim, observamos que muito frequentemente os padrões bem-sucedidos de ataque envolviam o uso de formas com cores escuras. Isso possivelmente indica que as redes de convolução acabam aprendendo classificadores que se baseiam muito fortemente nas cores das imagens e não apenas nos padrões geométricos presentes nos rostos das pessoas. Isso evidencia que a cor do padrão de ataque é um elemento crítico explorado pelos processos de otimização, assim como o tamanho da área sendo atacada. Quanto maior for a área do rosto passível de ataque, mais facilmente a rede pode ser enganada, uma vez que uma parte grande do rosto pode ser simplesmente coberta pelo padrão utilizado. Como trabalho futuro, sugerimos mensurar quantitativamente o grau de sucesso de ataque em função da área do padrão sendo utilizado no ataque.

REFERÊNCIAS

BRYSON, A. E.; DENHAM, W. F.; DREYFUS, S. E. Optimal programming problems with inequality constraints. **AIAA journal**, v. 1, n. 11, p. 2544–2550, 1963.

CAO, Q.; SHEN, L.; XIE, W.; PARKHI, O. M.; ZISSERMAN, A. Vggface2: A dataset for recognising faces across pose and age. Em: IEEE. **2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)**. [S.l.], 2018. p. 67–74.

CHEN, G.; HAN, T. X.; HE, Z.; KAYS, R.; FORRESTER, T. Deep convolutional neural network based species recognition for wild animal monitoring. Em: IEEE. **2014 IEEE International Conference on Image Processing (ICIP)**. [S.l.], 2014. p. 858–862.

DALVI, N.; DOMINGOS, P.; SANGHAI, S.; VERMA, D. et al. Adversarial classification. Em: ACM. **Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining**. [S.l.], 2004. p. 99–108.

GARCÍA, G. B.; SUAREZ, O. D.; ARANDA, J. L. E.; TERCERO, J. S.; GRACIA, I. S.; ENANO, N. V. **Learning image processing with opencv**. [S.l.]: Packt Publishing Ltd, 2015.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. <http://www.deeplearningbook.org>.

GOODFELLOW, I. J.; SHLENS, J.; SZEGEDY, C. Explaining and harnessing adversarial examples. **arXiv preprint arXiv:1412.6572**, 2014.

GU, T.; DOLAN-GAVITT, B.; GARG, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. **arXiv preprint arXiv:1708.06733**, 2017.

GULLI, A.; PAL, S. **Deep Learning with Keras**. [S.l.]: Packt Publishing Ltd, 2017.

HAYKIN, S. **Redes neurais: princípios e prática**. [S.l.]: Bookman Editora, 2007.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. Em: **Proceedings of the IEEE conference on computer vision and pattern recognition**. [S.l.: s.n.], 2016. p. 770–778.

KURAKIN, A.; GOODFELLOW, I.; BENGIO, S. Adversarial machine learning at scale. **arXiv preprint arXiv:1611.01236**, 2016.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. et al. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, Taipei, Taiwan, v. 86, n. 11, p. 2278–2324, 1998.

MATHEWS, J. H.; FINK, K. D. et al. **Numerical methods using MATLAB**. [S.l.]: Pearson Prentice Hall Upper Saddle River, NJ, 2004.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

PARKHI, O. M.; VEDALDI, A.; ZISSERMAN, A. Deep face recognition. Em: **British Machine Vision Conference**. [S.l.: s.n.], 2015.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011.

POWELL, M. J. An efficient method for finding the minimum of a function of several variables without calculating derivatives. **The computer journal**, Oxford University Press, v. 7, n. 2, p. 155–162, 1964.

SHARIF, M.; BHAGAVATULA, S.; BAUER, L.; REITER, M. K. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. Em: ACM. **Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security**. [S.l.], 2016. p. 1528–1540.

SU, J.; VARGAS, D. V.; SAKURAI, K. One pixel attack for fooling deep neural networks. **IEEE Transactions on Evolutionary Computation**, IEEE, 2019.

SZEGEDY, C.; ZAREMBA, W.; SUTSKEVER, I.; BRUNA, J.; ERHAN, D.; GOODFELLOW, I.; FERGUS, R. Intriguing properties of neural networks. **arXiv preprint arXiv:1312.6199**, 2013.

WALT, S. v. d.; COLBERT, S. C.; VAROQUAUX, G. The numpy array: a structure for efficient numerical computation. **Computing in Science & Engineering**, IEEE, v. 13, n. 2, p. 22–30, 2011.

APÊNDICE A — ARQUITETURA DO MODELO *RESNET50*

Abaixo segue a implementação do *resnet50* do keras utilizado no trabalho. Foi utilizada a função *summary()*.

Figura A.1: Arquitetura do *resnet50*.

Layer (type)	Output Shape	Param #	Connected to
input_12 (InputLayer)	(None, 224, 224, 3)	0	
conv1/7x7_s2 (Conv2D)	(None, 112, 112, 64)	9408	input_12[0][0]
conv1/7x7_s2/bn (BatchNormaliza	(None, 112, 112, 64)	256	conv1/7x7_s2[0][0]
activation_540 (Activation)	(None, 112, 112, 64)	0	conv1/7x7_s2/bn[0][0]
max_pooling2d_12 (MaxPooling2D)	(None, 55, 55, 64)	0	activation_540[0][0]
conv2_1_1x1_reduce (Conv2D)	(None, 55, 55, 64)	4096	max_pooling2d_12[0][0]
conv2_1_1x1_reduce/bn (BatchNor	(None, 55, 55, 64)	256	conv2_1_1x1_reduce[0][0]
activation_541 (Activation)	(None, 55, 55, 64)	0	conv2_1_1x1_reduce/bn[0][0]
conv2_1_3x3 (Conv2D)	(None, 55, 55, 64)	36864	activation_541[0][0]
conv2_1_3x3/bn (BatchNormalizat	(None, 55, 55, 64)	256	conv2_1_3x3[0][0]
activation_542 (Activation)	(None, 55, 55, 64)	0	conv2_1_3x3/bn[0][0]
conv2_1_1x1_increase (Conv2D)	(None, 55, 55, 256)	16384	activation_542[0][0]
conv2_1_1x1_proj (Conv2D)	(None, 55, 55, 256)	16384	max_pooling2d_12[0][0]
conv2_1_1x1_increase/bn (BatchN	(None, 55, 55, 256)	1024	conv2_1_1x1_increase[0][0]
conv2_1_1x1_proj/bn (BatchNorma	(None, 55, 55, 256)	1024	conv2_1_1x1_proj[0][0]
add_177 (Add)	(None, 55, 55, 256)	0	conv2_1_1x1_increase/bn[0][0] conv2_1_1x1_proj/bn[0][0]
activation_543 (Activation)	(None, 55, 55, 256)	0	add_177[0][0]
conv2_2_1x1_reduce (Conv2D)	(None, 55, 55, 64)	16384	activation_543[0][0]
conv2_2_1x1_reduce/bn (BatchNor	(None, 55, 55, 64)	256	conv2_2_1x1_reduce[0][0]
activation_544 (Activation)	(None, 55, 55, 64)	0	conv2_2_1x1_reduce/bn[0][0]
conv2_2_3x3 (Conv2D)	(None, 55, 55, 64)	36864	activation_544[0][0]
conv2_2_3x3/bn (BatchNormalizat	(None, 55, 55, 64)	256	conv2_2_3x3[0][0]
activation_545 (Activation)	(None, 55, 55, 64)	0	conv2_2_3x3/bn[0][0]

Figura A.2: Continuação da arquitetura do *resnet50*.

conv2_2_1x1_increase (Conv2D)	(None, 55, 55, 256)	16384	activation_545[0][0]
conv2_2_1x1_increase/bn (BatchN	(None, 55, 55, 256)	1024	conv2_2_1x1_increase[0][0]
add_178 (Add)	(None, 55, 55, 256)	0	conv2_2_1x1_increase/bn[0][0] activation_543[0][0]
activation_546 (Activation)	(None, 55, 55, 256)	0	add_178[0][0]
conv2_3_1x1_reduce (Conv2D)	(None, 55, 55, 64)	16384	activation_546[0][0]
conv2_3_1x1_reduce/bn (BatchNor	(None, 55, 55, 64)	256	conv2_3_1x1_reduce[0][0]
activation_547 (Activation)	(None, 55, 55, 64)	0	conv2_3_1x1_reduce/bn[0][0]
conv2_3_3x3 (Conv2D)	(None, 55, 55, 64)	36864	activation_547[0][0]
conv2_3_3x3/bn (BatchNormalizat	(None, 55, 55, 64)	256	conv2_3_3x3[0][0]
activation_548 (Activation)	(None, 55, 55, 64)	0	conv2_3_3x3/bn[0][0]
conv2_3_1x1_increase (Conv2D)	(None, 55, 55, 256)	16384	activation_548[0][0]
conv2_3_1x1_increase/bn (BatchN	(None, 55, 55, 256)	1024	conv2_3_1x1_increase[0][0]
add_179 (Add)	(None, 55, 55, 256)	0	conv2_3_1x1_increase/bn[0][0] activation_546[0][0]
activation_549 (Activation)	(None, 55, 55, 256)	0	add_179[0][0]
conv3_1_1x1_reduce (Conv2D)	(None, 28, 28, 128)	32768	activation_549[0][0]
conv3_1_1x1_reduce/bn (BatchNor	(None, 28, 28, 128)	512	conv3_1_1x1_reduce[0][0]
activation_550 (Activation)	(None, 28, 28, 128)	0	conv3_1_1x1_reduce/bn[0][0]
conv3_1_3x3 (Conv2D)	(None, 28, 28, 128)	147456	activation_550[0][0]
conv3_1_3x3/bn (BatchNormalizat	(None, 28, 28, 128)	512	conv3_1_3x3[0][0]
activation_551 (Activation)	(None, 28, 28, 128)	0	conv3_1_3x3/bn[0][0]
conv3_1_1x1_increase (Conv2D)	(None, 28, 28, 512)	65536	activation_551[0][0]
conv3_1_1x1_proj (Conv2D)	(None, 28, 28, 512)	131072	activation_549[0][0]
conv3_1_1x1_increase/bn (BatchN	(None, 28, 28, 512)	2048	conv3_1_1x1_increase[0][0]

Figura A.3: Continuação da arquitetura do *resnet50*.

conv3_1_1x1_proj/bn (BatchNorma	(None, 28, 28, 512)	2048	conv3_1_1x1_proj[0][0]
add_180 (Add)	(None, 28, 28, 512)	0	conv3_1_1x1_increase/bn[0][0] conv3_1_1x1_proj/bn[0][0]
activation_552 (Activation)	(None, 28, 28, 512)	0	add_180[0][0]
conv3_2_1x1_reduce (Conv2D)	(None, 28, 28, 128)	65536	activation_552[0][0]
conv3_2_1x1_reduce/bn (BatchNor	(None, 28, 28, 128)	512	conv3_2_1x1_reduce[0][0]
activation_553 (Activation)	(None, 28, 28, 128)	0	conv3_2_1x1_reduce/bn[0][0]
conv3_2_3x3 (Conv2D)	(None, 28, 28, 128)	147456	activation_553[0][0]
conv3_2_3x3/bn (BatchNormalizat	(None, 28, 28, 128)	512	conv3_2_3x3[0][0]
activation_554 (Activation)	(None, 28, 28, 128)	0	conv3_2_3x3/bn[0][0]
conv3_2_1x1_increase (Conv2D)	(None, 28, 28, 512)	65536	activation_554[0][0]
conv3_2_1x1_increase/bn (BatchN	(None, 28, 28, 512)	2048	conv3_2_1x1_increase[0][0]
add_181 (Add)	(None, 28, 28, 512)	0	conv3_2_1x1_increase/bn[0][0] activation_552[0][0]
activation_555 (Activation)	(None, 28, 28, 512)	0	add_181[0][0]
conv3_3_1x1_reduce (Conv2D)	(None, 28, 28, 128)	65536	activation_555[0][0]
conv3_3_1x1_reduce/bn (BatchNor	(None, 28, 28, 128)	512	conv3_3_1x1_reduce[0][0]
activation_556 (Activation)	(None, 28, 28, 128)	0	conv3_3_1x1_reduce/bn[0][0]
conv3_3_3x3 (Conv2D)	(None, 28, 28, 128)	147456	activation_556[0][0]
conv3_3_3x3/bn (BatchNormalizat	(None, 28, 28, 128)	512	conv3_3_3x3[0][0]
activation_557 (Activation)	(None, 28, 28, 128)	0	conv3_3_3x3/bn[0][0]
conv3_3_1x1_increase (Conv2D)	(None, 28, 28, 512)	65536	activation_557[0][0]
conv3_3_1x1_increase/bn (BatchN	(None, 28, 28, 512)	2048	conv3_3_1x1_increase[0][0]
add_182 (Add)	(None, 28, 28, 512)	0	conv3_3_1x1_increase/bn[0][0] activation_555[0][0]
activation_558 (Activation)	(None, 28, 28, 512)	0	add_182[0][0]

Figura A.4: Continuação da arquitetura do *resnet50*.

conv3_4_1x1_reduce (Conv2D)	(None, 28, 28, 128)	65536	activation_558[0][0]
conv3_4_1x1_reduce/bn (BatchNor	(None, 28, 28, 128)	512	conv3_4_1x1_reduce[0][0]
activation_559 (Activation)	(None, 28, 28, 128)	0	conv3_4_1x1_reduce/bn[0][0]
conv3_4_3x3 (Conv2D)	(None, 28, 28, 128)	147456	activation_559[0][0]
conv3_4_3x3/bn (BatchNormalizat	(None, 28, 28, 128)	512	conv3_4_3x3[0][0]
activation_560 (Activation)	(None, 28, 28, 128)	0	conv3_4_3x3/bn[0][0]
conv3_4_1x1_increase (Conv2D)	(None, 28, 28, 512)	65536	activation_560[0][0]
conv3_4_1x1_increase/bn (BatchN	(None, 28, 28, 512)	2048	conv3_4_1x1_increase[0][0]
add_183 (Add)	(None, 28, 28, 512)	0	conv3_4_1x1_increase/bn[0][0] activation_558[0][0]
activation_561 (Activation)	(None, 28, 28, 512)	0	add_183[0][0]
conv4_1_1x1_reduce (Conv2D)	(None, 14, 14, 256)	131072	activation_561[0][0]
conv4_1_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_1_1x1_reduce[0][0]
activation_562 (Activation)	(None, 14, 14, 256)	0	conv4_1_1x1_reduce/bn[0][0]
conv4_1_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_562[0][0]
conv4_1_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_1_3x3[0][0]
activation_563 (Activation)	(None, 14, 14, 256)	0	conv4_1_3x3/bn[0][0]
conv4_1_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_563[0][0]
conv4_1_1x1_proj (Conv2D)	(None, 14, 14, 1024)	524288	activation_561[0][0]
conv4_1_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_1_1x1_increase[0][0]
conv4_1_1x1_proj/bn (BatchNorma	(None, 14, 14, 1024)	4096	conv4_1_1x1_proj[0][0]
add_184 (Add)	(None, 14, 14, 1024)	0	conv4_1_1x1_increase/bn[0][0] conv4_1_1x1_proj/bn[0][0]
activation_564 (Activation)	(None, 14, 14, 1024)	0	add_184[0][0]
conv4_2_1x1_reduce (Conv2D)	(None, 14, 14, 256)	262144	activation_564[0][0]

Figura A.5: Continuação da arquitetura do *resnet50*.

conv4_2_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_2_1x1_reduce[0][0]
activation_565 (Activation)	(None, 14, 14, 256)	0	conv4_2_1x1_reduce/bn[0][0]
conv4_2_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_565[0][0]
conv4_2_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_2_3x3[0][0]
activation_566 (Activation)	(None, 14, 14, 256)	0	conv4_2_3x3/bn[0][0]
conv4_2_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_566[0][0]
conv4_2_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_2_1x1_increase[0][0]
add_185 (Add)	(None, 14, 14, 1024)	0	conv4_2_1x1_increase/bn[0][0] activation_564[0][0]
activation_567 (Activation)	(None, 14, 14, 1024)	0	add_185[0][0]
conv4_3_1x1_reduce (Conv2D)	(None, 14, 14, 256)	262144	activation_567[0][0]
conv4_3_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_3_1x1_reduce[0][0]
activation_568 (Activation)	(None, 14, 14, 256)	0	conv4_3_1x1_reduce/bn[0][0]
conv4_3_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_568[0][0]
conv4_3_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_3_3x3[0][0]
activation_569 (Activation)	(None, 14, 14, 256)	0	conv4_3_3x3/bn[0][0]
conv4_3_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_569[0][0]
conv4_3_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_3_1x1_increase[0][0]
add_186 (Add)	(None, 14, 14, 1024)	0	conv4_3_1x1_increase/bn[0][0] activation_567[0][0]
activation_570 (Activation)	(None, 14, 14, 1024)	0	add_186[0][0]
conv4_4_1x1_reduce (Conv2D)	(None, 14, 14, 256)	262144	activation_570[0][0]
conv4_4_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_4_1x1_reduce[0][0]
activation_571 (Activation)	(None, 14, 14, 256)	0	conv4_4_1x1_reduce/bn[0][0]
conv4_4_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_571[0][0]

Figura A.6: Continuação da arquitetura do *resnet50*.

```
conv4_2_1x1_reduce/bn (BatchNor (None, 14, 14, 256)  1024        conv4_2_1x1_reduce[0][0]
__________________________________________________________________________________________
activation_565 (Activation)     (None, 14, 14, 256)  0           conv4_2_1x1_reduce/bn[0][0]
__________________________________________________________________________________________
conv4_2_3x3 (Conv2D)            (None, 14, 14, 256)  589824      activation_565[0][0]
__________________________________________________________________________________________
conv4_2_3x3/bn (BatchNormalizat (None, 14, 14, 256)  1024        conv4_2_3x3[0][0]
__________________________________________________________________________________________
activation_566 (Activation)     (None, 14, 14, 256)  0           conv4_2_3x3/bn[0][0]
__________________________________________________________________________________________
conv4_2_1x1_increase (Conv2D)   (None, 14, 14, 1024) 262144      activation_566[0][0]
__________________________________________________________________________________________
conv4_2_1x1_increase/bn (BatchN (None, 14, 14, 1024) 4096        conv4_2_1x1_increase[0][0]
__________________________________________________________________________________________
add_185 (Add)                   (None, 14, 14, 1024) 0           conv4_2_1x1_increase/bn[0][0]
                                                                 activation_564[0][0]
__________________________________________________________________________________________
activation_567 (Activation)     (None, 14, 14, 1024) 0           add_185[0][0]
__________________________________________________________________________________________
conv4_3_1x1_reduce (Conv2D)     (None, 14, 14, 256)  262144      activation_567[0][0]
__________________________________________________________________________________________
conv4_3_1x1_reduce/bn (BatchNor (None, 14, 14, 256)  1024        conv4_3_1x1_reduce[0][0]
__________________________________________________________________________________________
activation_568 (Activation)     (None, 14, 14, 256)  0           conv4_3_1x1_reduce/bn[0][0]
__________________________________________________________________________________________
conv4_3_3x3 (Conv2D)            (None, 14, 14, 256)  589824      activation_568[0][0]
__________________________________________________________________________________________
conv4_3_3x3/bn (BatchNormalizat (None, 14, 14, 256)  1024        conv4_3_3x3[0][0]
__________________________________________________________________________________________
activation_569 (Activation)     (None, 14, 14, 256)  0           conv4_3_3x3/bn[0][0]
__________________________________________________________________________________________
conv4_3_1x1_increase (Conv2D)   (None, 14, 14, 1024) 262144      activation_569[0][0]
__________________________________________________________________________________________
conv4_3_1x1_increase/bn (BatchN (None, 14, 14, 1024) 4096        conv4_3_1x1_increase[0][0]
__________________________________________________________________________________________
add_186 (Add)                   (None, 14, 14, 1024) 0           conv4_3_1x1_increase/bn[0][0]
                                                                 activation_567[0][0]
__________________________________________________________________________________________
activation_570 (Activation)     (None, 14, 14, 1024) 0           add_186[0][0]
__________________________________________________________________________________________
conv4_4_1x1_reduce (Conv2D)     (None, 14, 14, 256)  262144      activation_570[0][0]
__________________________________________________________________________________________
conv4_4_1x1_reduce/bn (BatchNor (None, 14, 14, 256)  1024        conv4_4_1x1_reduce[0][0]
__________________________________________________________________________________________
activation_571 (Activation)     (None, 14, 14, 256)  0           conv4_4_1x1_reduce/bn[0][0]
__________________________________________________________________________________________
conv4_4_3x3 (Conv2D)            (None, 14, 14, 256)  589824      activation_571[0][0]
```

Figura A.7: Continuação da arquitetura do *resnet50*.

conv4_4_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_4_3x3[0][0]
activation_572 (Activation)	(None, 14, 14, 256)	0	conv4_4_3x3/bn[0][0]
conv4_4_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_572[0][0]
conv4_4_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_4_1x1_increase[0][0]
add_187 (Add)	(None, 14, 14, 1024)	0	conv4_4_1x1_increase/bn[0][0] activation_570[0][0]
activation_573 (Activation)	(None, 14, 14, 1024)	0	add_187[0][0]
conv4_5_1x1_reduce (Conv2D)	(None, 14, 14, 256)	262144	activation_573[0][0]
conv4_5_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_5_1x1_reduce[0][0]
activation_574 (Activation)	(None, 14, 14, 256)	0	conv4_5_1x1_reduce/bn[0][0]
conv4_5_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_574[0][0]
conv4_5_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_5_3x3[0][0]
activation_575 (Activation)	(None, 14, 14, 256)	0	conv4_5_3x3/bn[0][0]
conv4_5_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_575[0][0]
conv4_5_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_5_1x1_increase[0][0]
add_188 (Add)	(None, 14, 14, 1024)	0	conv4_5_1x1_increase/bn[0][0] activation_573[0][0]
activation_576 (Activation)	(None, 14, 14, 1024)	0	add_188[0][0]
conv4_6_1x1_reduce (Conv2D)	(None, 14, 14, 256)	262144	activation_576[0][0]
conv4_6_1x1_reduce/bn (BatchNor	(None, 14, 14, 256)	1024	conv4_6_1x1_reduce[0][0]
activation_577 (Activation)	(None, 14, 14, 256)	0	conv4_6_1x1_reduce/bn[0][0]
conv4_6_3x3 (Conv2D)	(None, 14, 14, 256)	589824	activation_577[0][0]
conv4_6_3x3/bn (BatchNormalizat	(None, 14, 14, 256)	1024	conv4_6_3x3[0][0]
activation_578 (Activation)	(None, 14, 14, 256)	0	conv4_6_3x3/bn[0][0]
conv4_6_1x1_increase (Conv2D)	(None, 14, 14, 1024)	262144	activation_578[0][0]

Figura A.8: Continuação da arquitetura do *resnet50*.

conv4_6_1x1_increase/bn (BatchN	(None, 14, 14, 1024)	4096	conv4_6_1x1_increase[0][0]
add_189 (Add)	(None, 14, 14, 1024)	0	conv4_6_1x1_increase/bn[0][0] activation_576[0][0]
activation_579 (Activation)	(None, 14, 14, 1024)	0	add_189[0][0]
conv5_1_1x1_reduce (Conv2D)	(None, 7, 7, 512)	524288	activation_579[0][0]
conv5_1_1x1_reduce/bn (BatchNor	(None, 7, 7, 512)	2048	conv5_1_1x1_reduce[0][0]
activation_580 (Activation)	(None, 7, 7, 512)	0	conv5_1_1x1_reduce/bn[0][0]
conv5_1_3x3 (Conv2D)	(None, 7, 7, 512)	2359296	activation_580[0][0]
conv5_1_3x3/bn (BatchNormalizat	(None, 7, 7, 512)	2048	conv5_1_3x3[0][0]
activation_581 (Activation)	(None, 7, 7, 512)	0	conv5_1_3x3/bn[0][0]
conv5_1_1x1_increase (Conv2D)	(None, 7, 7, 2048)	1048576	activation_581[0][0]
conv5_1_1x1_proj (Conv2D)	(None, 7, 7, 2048)	2097152	activation_579[0][0]
conv5_1_1x1_increase/bn (BatchN	(None, 7, 7, 2048)	8192	conv5_1_1x1_increase[0][0]
conv5_1_1x1_proj/bn (BatchNorma	(None, 7, 7, 2048)	8192	conv5_1_1x1_proj[0][0]
add_190 (Add)	(None, 7, 7, 2048)	0	conv5_1_1x1_increase/bn[0][0] conv5_1_1x1_proj/bn[0][0]
activation_582 (Activation)	(None, 7, 7, 2048)	0	add_190[0][0]
conv5_2_1x1_reduce (Conv2D)	(None, 7, 7, 512)	1048576	activation_582[0][0]
conv5_2_1x1_reduce/bn (BatchNor	(None, 7, 7, 512)	2048	conv5_2_1x1_reduce[0][0]
activation_583 (Activation)	(None, 7, 7, 512)	0	conv5_2_1x1_reduce/bn[0][0]
conv5_2_3x3 (Conv2D)	(None, 7, 7, 512)	2359296	activation_583[0][0]
conv5_2_3x3/bn (BatchNormalizat	(None, 7, 7, 512)	2048	conv5_2_3x3[0][0]
activation_584 (Activation)	(None, 7, 7, 512)	0	conv5_2_3x3/bn[0][0]
conv5_2_1x1_increase (Conv2D)	(None, 7, 7, 2048)	1048576	activation_584[0][0]
conv5_2_1x1_increase/bn (BatchN	(None, 7, 7, 2048)	8192	conv5_2_1x1_increase[0][0]

Figura A.9: Continuação da arquitetura do *resnet50*.

```
add_191 (Add)                   (None, 7, 7, 2048)   0           conv5_2_1x1_increase/bn[0][0]
                                                                 activation_582[0][0]
__________________________________________________________________________________________________
activation_585 (Activation)     (None, 7, 7, 2048)   0           add_191[0][0]
__________________________________________________________________________________________________
conv5_3_1x1_reduce (Conv2D)     (None, 7, 7, 512)    1048576     activation_585[0][0]
__________________________________________________________________________________________________
conv5_3_1x1_reduce/bn (BatchNor (None, 7, 7, 512)    2048        conv5_3_1x1_reduce[0][0]
__________________________________________________________________________________________________
activation_586 (Activation)     (None, 7, 7, 512)    0           conv5_3_1x1_reduce/bn[0][0]
__________________________________________________________________________________________________
conv5_3_3x3 (Conv2D)            (None, 7, 7, 512)    2359296     activation_586[0][0]
__________________________________________________________________________________________________
conv5_3_3x3/bn (BatchNormalizat (None, 7, 7, 512)    2048        conv5_3_3x3[0][0]
__________________________________________________________________________________________________
activation_587 (Activation)     (None, 7, 7, 512)    0           conv5_3_3x3/bn[0][0]
__________________________________________________________________________________________________
conv5_3_1x1_increase (Conv2D)   (None, 7, 7, 2048)   1048576     activation_587[0][0]
__________________________________________________________________________________________________
conv5_3_1x1_increase/bn (BatchN (None, 7, 7, 2048)   8192        conv5_3_1x1_increase[0][0]
__________________________________________________________________________________________________
add_192 (Add)                   (None, 7, 7, 2048)   0           conv5_3_1x1_increase/bn[0][0]
                                                                 activation_585[0][0]
__________________________________________________________________________________________________
activation_588 (Activation)     (None, 7, 7, 2048)   0           add_192[0][0]
__________________________________________________________________________________________________
avg_pool (AveragePooling2D)     (None, 1, 1, 2048)   0           activation_588[0][0]
__________________________________________________________________________________________________
flatten_12 (Flatten)            (None, 2048)         0           avg_pool[0][0]
__________________________________________________________________________________________________
classifier (Dense)              (None, 8631)         17684919    flatten_12[0][0]
==================================================================================================
Total params: 41,246,071
Trainable params: 41,192,951
Non-trainable params: 53,120
```