Combinando *Feature squeezing* e *Lottery Tickets* para Detecção de Exemplos Adversariais

Leon Tenório da Silva e Fabio Augusto Faria
Instituto de Ciência e Tecnologia – Universidade de São Paulo (ICT-UNIFESP)
São José dos Campos, São Paulo
Email: {leon.silva, ffaria}@unifesp.br

***Resumo*—As redes neurais estão cada vez mais sendo utilizadas em diversas aplicações reais em diferentes domínios de conhecimento (e.g., medicina, agricultura e segurança). Entretanto, pequenas variações nos dados de entrada podem causar comportamentos totalmente inesperados do modelo aprendido. Uma técnica chamada *feature squeezing* tem objetivo de identificar quando imagens de teste podem causar esse fenômeno, combinada com uma estratégia de aprendizagem chamada *lottery tickets* mostra ser uma boa solução para tornar um modelo mais robusto aos ataques adversariais e com baixo consumo computacional.**

Abstract—Neural networks have been applied to many real applications in different knowledge domains (e.g., medicine, agriculture, and security). However, little changes in the input data might bring totally unexpected behaviours from the learned models. A well-known technique called feature squeezing, which aims to identify adversal examples combined with a learning strategy called lottery tickets show to be a good solution to make a more robust model to adversarial attacks.

I. Introdução

De acordo com os resultados de [1], os modelos de inteligência artificial baseados nas mais modernas técnicas de aprendizagem, mesmo obtendo um excelente desempenho nos casos de teste, podem não estar realmente aprendendo os conceitos que determinam a predição correta. Em vez disso, essas técnicas estão construindo uma *Potemkin village* que significa uma construção que induz os observadores a pensar que a situação daquele local é muito boa e que trabalha bem nas ocorrências naturais dos dados, porém é exposto a uma farsa quando são explorados pontos no espaço que não tem uma alta probabilidade na distribuição de treino.

Em conjunto a isso, visto que nas situações reais não é possível prever as condições que esse modelo estará exposto, a habilidade do modelo de identificar quando ele está em situações adversas, i.e., aquelas em que o modelo não tem recursos suficientes para classificar de forma correta as novas imagens, é de extrema relevância. Isso pode ocorrer quando muito provavelmente essas imagens não estejam dentro das condições razoavelmente próximas ao conjunto de treinamento estando assim em um domínio diferente (exemplos adversariais).

Neste sentido, o desenvolvimento de novas técnicas de deteção de exemplos adversariais se faz necessário. Portanto, neste trabalho, uma técnica chamada *feature squeezing* [2] tem sido estudada e combinada com uma abordagem de poda de redes neurais (*lottery ticket hypothesis*) com objetivo criar um arcabouço para deteção de exemplos adversariais com baixo consumo computacional.

II. Fundamentação Teórica

Nesta seção são mostrados conceitos importantes para um melhor entendimento do trabalho.

A. Criação de dados sintéticos

Como apresentado na introdução, os modelos de classificação podem não estar corretamente ajustados aos casos do mundo real, assim, atacar uma rede com variados exemplos pode expor uma falta de semântica do modelo. Nessa seção serão explicadas diferentes formas de criação de dados sintéticos (exemplos adversariais) utilizadas nos experimentos.

1) ***Ruído gaussiano****:* O ruído gaussiano, também chamado de ruído normal, geralmente perturba os valores de cinza nas imagens digitais. Isso se deve ao fato de que ele é caracterizado por seu *Probability Density Function (PDF)* ou normalização do histograma em relação ao valor de cinza [3].

2) ***Ruído sal e pimenta****:* O ruído sal e pimenta se refere a uma classe de transformações que degradam poucos pixeis na imagem, mas os poucos degradados são atingidos com grande intensidade [4]. Basicamente, esse ruído substituiu os valores afetados pelo máximo ou mínimo possível na escala em que os pixeis estão.

3) ***Ruído poisson****:* De acordo com [5] o ruído *poisson* é uma forma de perturbar a entrada de dados que tem magnitude dependente da quantidade de luz das imagens.

4) ***Ruído speckle****:* De acordo com [6] o ruído *speckle* é inerente as imagens capturadas por ondas sonoras, de radar, a laser, entre outras, e sua manifestação ocorre devido a interferência aleatória das ondas na superfície do objeto.

5) ***Fast Gradient Sign Method Attack (FGSM)****:* De acordo com [1] esse método, para cada imagem passada pelo modelo, aplica uma perturbação na direção do gradiente da função de perda dessa predição com amplitude controlado pelo parâmetro *épsilon*.

6) ***Carlini & Wagnet attack****:* Esse método proposto em [7] utiliza uma minimização da proporção da perturbação nas entradas utilizando *binary search*.

B. The Lottery Ticket Hypothesis (LTH)

Em LTH [8] é explorada a técnica de treinamento dos modelos em que uma rede neural inicializada de forma aleatória pode existir um subconjunto de *perceptrons* que quando utilizado de forma isolada pode atingir no mínimo a acurácia que a rede neural original atinge, ou seja, é possível que uma rede com menos parâmetros seja no mínimo igual a rede original. Esta técnica reduz a quantidade de parâmetros (pesos) de uma rede neural, criando modelos aprendidos mais leves e assim, necessitando de menos recursos computacionais para sua utilização. No artigo original é possível verificar uma redução de até 90% dos parâmetros de uma rede neural, mantendo os resultados de classificação das redes iniciais, i.e., aquelas redes com todos os parâmetros em uso.

C. Feature Squeezing

Baseado na ideia que o espaço dimensional da imagem de entrada contém uma vasta gama de elementos que podem ser alterados para geração de um exemplo adversarial, a aplicação de transformações que suprimem essas pequenas variações e reduzem as oportunidades de ataque pode ser resumida como o grande intuito do método *feature squeezing* [2].

A ideia principal desse método é comparar as predições tanto dos modelos com transformações da entrada de dados, quanto no modelo sem nenhuma transformação [2]. Essa comparação é feita, para cada imagem, por meio de uma distância euclidiana das saídas dos modelos e se um desses valores é maior que um parâmetro fixado a imagem é considerada adversarial, senão legítima. Esse método é exemplificado na Figura 1.

III. Metodologia Experimental

Nessa seção são abordados os tópicos necessários para o entendimento dos procedimentos realizados no trabalho. Todo o desenvolvimento foi feito utilizando a linguagem de programação *python*, as bibliotecas *tensorflow*, *keras*, *numpy*, *cleverhans*, *scikit-image* e *opencv*.

A. Conjunto de dados

O *dataset* (conjunto) utilizado foi o *CIFAR10* [9] que consiste em 60 mil imagens coloridas nas dimensões 32x32x3 dividas igualmente sobre as suas 10 classes. Esse conjunto é divido em 50 mil imagens de treino e 10 mil imagens de teste. Para os nossos experimentos, a partir do conjunto de treino disponibilizado pelo *dataset* foram extraídas as 5 mil últimas imagens que compuseram o conjunto de validação, já as 45 mil imagens restantes no conjunto de treino original foram utilizadas como treino propriamente dito e as 10 mil imagens de teste foram utilizadas como teste.

Nos casos de teste e validação foram criadas as bases de dados com o ataque adversarial *FGSM*, ruído sal e pimenta com 3 níveis de perturbação, ruído gaussiano com 3 níveis de perturbação, ruído *speckle* com 3 níveis de perturbação e ruído *poisson* com 1 nível.

Para a criação das bases ruidosas foi utilizada a biblioteca *scikit image* [10] e para cada uma das bases temos os seguintes parâmetros: (i) ruído gaussiano alto, médio e baixo com *var*, respectivamente, 0,020, 0,010, 0,002 e *mean* = 0; (ii) ruído sal e pimenta alto, médio e baixo com *amount*, respectivamente, 0,0630, 0,0285 e 0,0080; (iii) ruído *speckle* alto, médio e baixo com (*var*; *mean*), respectivamente, (0,0810; 0.067), (0,0295; 0,020) e (0,0080; 0,000); (iv) ruído *poisson* sem parâmetros controláveis.

Nos casos de teste, para o ataque *FGSM* foi utilizado *tensorflow* com a função de perda *binary crossentropy* (mesma função de perda utilizada nos modelos) e o *keras* com *épsilon* = $16/255$, e para o ataque *CW* a biblioteca *cleverhans* [11] da implementação *CarliniWagnerL2* com os parâmetros, além do predefinido, *confidence* = 10, *learning_rate* = 0,1, *binary_search_steps* = 5, *maxiterations* = 300, *abort_early* = *True*, *initial_const* = 0,04.

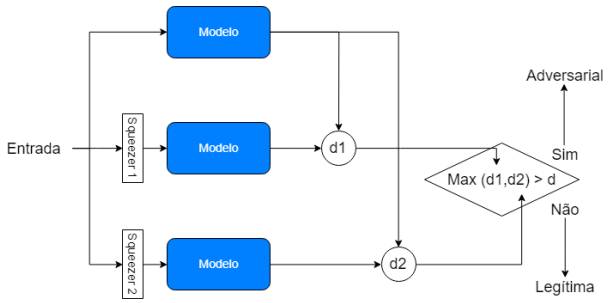

Figura 1. Exemplificação do funcionamento do *feature squeezing*.

B. Arquitetura de rede neural convolucional

Nesse trabalho foi utilizada a arquitetura *LeNet5* [12] com uma modificação na dimensão dos dados de entrada para 32x32x3. Para isso foi implementada a sua estrutura no *keras* com *beckend tensorflow* e inicialização dos parâmetros pelo algoritmo *glorot_normal*.

C. Técnica de treinamento por tickets

Na etapa de treinamento o modelo foi explorado em 10 janelas com 50.000 iterações cada. A cada iteração o modelo era exposto a 60 imagens aleatórias do conjunto de treinamento, a cada 100 iterações o modelo era validado na base limpa de validação e quando atingisse o seu menor valor na função de perda o modelo foi considerado o melhor da janela, e no fim da janela de validação o melhor modelo era avaliado nos casos de teste sem a base *FGSM* e posteriormente era realizada a poda permanecendo 80% dos valores dos *perceptrons* dessa janela para a próxima. O modelo global foi obtido com o mesmo critério entre os modelos de todas as janelas.

D. Técnicas usadas nos squeezers

Para a utilização do procedimento de *Feature squeezing* foram exploradas outras transformações em conjunto com as citadas em [2].

*1) **Profundidade de cores**:* A escala de cores presentes em uma imagem, quanto maior, mais nitidez e detalhes uma pessoa pode observar. Contudo, para classifica-lá não é necessário um nível de detalhe muito alto. Logo, a diminuição da profundidade de cores, até um certo nível, pode ajudar o modelo discretizando em uma escala com uma menor gama de níveis. Nas nossas aplicações foi utilizada a profundidade de cores igual a 4.

*2) **Median filter**:* Essa transformação substitui cada pixel pela mediana dos pixeis vizinhos em um certo janelamento, causando um efeito que borra a imagem de entrada e diminui a gama de contraste. Nessa aplicação foi utilizada implementação presente na biblioteca *scipy* [13] com um janelamento de tamanho 2.

*3) **Tv chambolle denoising**:* De acordo com [14] essa é uma técnica para diminuir a variação total de uma imagem. Foi utilizada a biblioteca *scikit image* [10] com *weight* = 0,1, *eps* = 0,0002, *n_iter_max* = 200, *multichannel* = *True*.

*4) **Bilateral denoising**:* Foi utilizada a implementação presente na biblioteca *scikit image* [10]. Nessa documentação temos que esse filtro reduz o ruído preservando as bordas, calculando a média dos pixeis com base na proximidade espacial e semelhança radiométrica. Foram utilizados os parâmetros *win_size* = *None*, *sigma_color* = 0,05, *sigma_spatial* = 5, *bins* = 10000, *mode* = '*constant*', *cval* = 0, *multichannel* = True, *sigma_range* = *None*.

*5) **Wavelet denoising**:* De acordo com [15], essa técnica de recuperação de imagens com ruído se baseia em uma transformação de domínio da imagem de entrada para um outro domínio que concentra a energia (ou representatividade) em um espectro onde os pontos mais importantes da imagem se mostram com grande amplitude e os pontos com perturbações se mostram com pouca energia, logo é possível eliminar esses pontos e obter uma imagem recuperada. A implementação utilizada desse filtro está presente na biblioteca *scikit image* [10]. Foram utilizados os parâmetros *sigma* = *None*, *wavelet* = 'db1', *mode* = '*soft*', *wavelet_levels* = *None*, *multichannel* = *True*, *convert2ycbcr* = *True*.

*6) **NL smoothing**:* Essa técnica substituiu o valor do pixel pela média dos valores em um janelamento o que de certa forma borra a imagem. Foi utilizada a biblioteca *opencv* [16] com os parâmetros dst = *None*, h = 10, *hColor* = 10, *templateWindowSize* = 7 e *searchWindowSize* = 21.

IV. Desenvolvimento

Foram obtidos 6 modelos com aplicação dos *squeezers* explicados na Seção III-D (modelos secundários) e um modelo sem transformação (modelo principal). Depois, foram exploradas todas as combinações possíveis de arcabouços com no mínimo o modelo principal e 2 modelos secundários totalizando 3 modelos. Observando os dados da máxima distância de predição com intervalo de confiança de 80% no conjunto de validação (justificado devido a restrição de 10% de falsos positivos em uma distribuição normal unilateral) excluímos os casos que continham o valor $0,775$, o que nos resultou em 13 possibilidades.

Cada uma dessas possibilidades foi validada nos casos limpos e com *FGSM*. Foi avaliada a taxa de detecção com a distância máxima de $0,555$, $0,600$, $0,650$, $0,700$, $0,750$, $0,775$ e $0,800$, e a melhor possibilidade apresenta os squeezers *nl smoothing* e *tv chambolle* com distâncias $0,775$ e $0,800$. Por fim, essa possibilidade foi explorada nos casos de teste variando a distância máxima entre $0,750$ até $0,800$ com um passo de $0,005$.

V. Resultados

A. Tickets

Na Tabela I temos a avaliação dos considerados melhores modelos pela técnica de *tickets* aplicada em cada janelamento. Os valores em negrito destacam o modelo considerado o melhor globalmente para cada um dos modelos, evidenciando o fato de que apenas nos modelos profundidade de cores e *NL smoothing* o critério adotado acertou explicitamente o melhor caso de teste. Além disso, podemos observar que sempre tivemos uma janela com um taxa de acerto maior que a primeira janela, confirmando a teoria dos *tickets*. Notamos também que para cada um dos modelos o melhor caso só foi encontrado a partir da janela que tem a permanência de $32,76\%$ ou menos dos *perceptrons*.

B. Feature Squeezing

Na Tabela II podemos observar o teste final do arcabouço com a variação da distância máxima de predição para ser considerado adversarial entre $0,785$ até $0,830$ nas bases de teste ruidosas e nos ataques *FGSM* e *CW*.

Nela observamos que existe uma distinção entre as taxas de detecção adversarial dos casos limpos, ruidosos e ataques. Para as imagens ruidosas as taxas de acerto não decaem muito e existe um crescimento proporcional na taxa de detecção entre os níveis de ruído. Já para os ataques, percebemos que as taxas de detecção sempre são muito superiores as taxas de detecção das outras bases.

Não obstante, como nosso objetivo é detectar uma quantidade satisfatória de ataques adversariais e rejeitar pouco dos casos limpos, escolhemos o melhor caso a distância = $0,825$ que tem $9,99\%$ de falsos positivos, $78,30\%$ de detecções corretas em *FGSM* e $75,5\%$ de detecções corretas em *CW*. Importante comentar que no artigo base [2], no mesmo conjunto de imagens e 5% de falsos positivos, foi relatado uma detecção *FGSM* de 20,80% e nos casos *CW* 100%.

Além disso, percebemos o nível de implicação de erros que os ataques conseguirem induzir no modelo, atingindo taxas de acerto gerais, respectivamente para *FGSM* e *CW*, de 81,71% e 83,01% e que após a exclusão dos casos considerados adversariais a taxa de acerto sobe, para os dois casos respectivamente, para 82,50% e 86,50%. Sabendo que a taxa de erro para o conjunto total é $(1-CA)*(1-TASA)$, temos, respectivamente para *FSGM* e *CW*, erros de 9,05% e 3.29% nas imagens desses ataques.

Tabela I
Loterry tickets - TAXA DE ACERTO DOS MELHORES MODELOS DE CADA JANELAMENTO NOS CASOS DE TESTE.

Modelos	Janela (porcentagem remanescente de perceptrons)									
	1 (100%)	2 (80%)	3 (64%)	4 (51%)	5 (41%)	6 (33%)	7 (26%)	8 (21%)	9 (17%)	10 (13%)
Sem transformação	93,23%	93,31%	93,49%	93,43%	93,56%	93,66%	93,59%	93,65%	**93,55%**	93,60%
Denoise bilateral	92,87%	92,90%	93,06%	93,01%	93,09%	93,09%	93,18%	**93,05%**	92,99%	93,03%
Prof. de cores	93,12%	93,26%	93,36%	93,48%	93,45%	93,41%	93,59%	**93,62%**	93,53%	93,62%
Median filter	92,75%	92,80%	92,89%	93,00%	92,97%	**93,03%**	93,03%	93,13%	93,02%	93,03%
NL smoothing	92,84%	92,83%	92,92%	92,99%	92,93%	92,95%	92,96%	92,95%	**93,03%**	93,03%
TV chambolle	92,96%	93,21%	93,36%	93,48%	93,31%	**93,41%**	93,38%	93,29%	93,36%	93,37%
Wavelet	92,99%	93,26%	93,38%	93,48%	93,46%	93,55%	93,42%	93,58%	**93,24%**	93,40%

Tabela II
Feature squeezing - TESTES FINAIS DO ARCABOUÇO VARIANDO A DISTÂNCIA (VALORES EM PORCENTAGEM). (TA = TAXA DE ACERTO; CA = CLASSIFICADOS COMO ADVERSARIAL; TASA = TAXA DE ACERTO SEM OS CASOS CLASSIFICADOS COMO ADVERSARIAIS).

		Distâncias																			
		0,785		0,790		0,795		0,800		0,805		0,810		0,815		0,820		**0,825**		0,830	
	TA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA	CA	TASA
Teste original	93,55	12,39	93,88	12,07	93,87	11,73	93,86	11,51	93,85	11,22	93,85	10,95	93,83	10,66	93,82	10,36	93,81	**9,99**	93,80	9,74	93,79
Gaussiano alto	90,93	20,68	91,99	20,22	91,96	19,80	91,95	19,41	91,94	18,96	91,92	18,61	91,90	18,15	91,87	17,62	91,85	17,23	91,82	16,77	91,79
Gaussiano médio	92,32	15,59	93,01	15,15	93,00	14,78	92,98	14,40	92,97	14,12	92,95	13,85	92,94	13,46	92,93	13,17	92,92	12,78	92,90	12,40	92,89
Gaussiano baixo	93,46	12,97	93,90	12,63	93,88	12,25	93,86	11,98	93,87	11,71	93,85	11,43	93,85	11,15	93,84	10,89	93,83	10,54	93,82	10,19	93,81
Poisson	93,49	12,89	93,94	12,58	93,93	12,22	93,91	11,76	93,90	11,46	93,89	11,09	93,87	10,87	93,87	10,48	93,85	10,24	93,84	9,96	93,83
Sal e pimenta alto	90,84	20,63	91,72	20,10	91,69	19,60	91,67	19,12	91,64	18,69	91,63	18,23	91,60	17,77	91,58	17,36	91,56	16,96	91,54	16,53	91,52
Sal e pimenta médio	92,61	16,89	93,26	16,46	93,23	15,96	93,22	15,54	93,21	15,11	93,20	14,72	93,17	14,40	93,15	14,06	93,13	13,70	93,12	13,43	93,11
Sal e pimenta baixo	93,41	13,65	93,80	13,33	93,79	12,86	93,78	12,55	93,76	12,24	93,75	11,93	93,73	11,60	93,71	11,31	93,71	10,91	93,70	10,56	93,70
Speckle alto	91,47	20,30	92,36	19,73	92,33	19,23	92,30	18,79	92,28	18,27	92,26	17,88	92,24	17,39	92,22	16,95	92,21	16,55	92,18	16,07	92,16
Speckle médio	92,90	15,03	93,47	14,76	93,46	14,32	93,45	13,84	93,43	13,48	93,41	13,11	93,40	12,72	93,39	12,52	93,38	12,16	93,37	11,82	93,36
Speckle baixo	93,48	12,89	93,86	12,45	93,85	12,17	93,84	11,93	93,83	11,55	93,83	11,31	93,82	11,06	93,82	10,79	93,81	10,46	93,80	10,15	93,79
FGSM	81,71	52,75	82,61	52,20	82,59	51,66	82,58	51,01	82,56	50,48	82,54	49,85	82,53	49,32	82,53	48,88	82,52	**48,30**	**82,50**	47,84	82,49
CW	83,01	78,15	86,79	77,74	86,74	77,35	86,68	77,09	86,65	76,75	86,63	76,53	86,61	76,21	86,55	75,92	86,53	**75,59**	**86,50**	75,29	86,46

VI. CONCLUSÃO

Neste trabalho, a técnica de *feature squeezing* foi estudada e mostrou ser viável para a detecção de exemplos adversariais. Uma exploração mais aprofundada dessa técnica no futuro precisa ser realizada, experimentando outras transformações *squeezers*, variando os valores de distância máxima a serem considerados como parâmetro e/ou outras formas de correlacionar esses valores de distanciamento para classificação adversarial. Além disso, foi possível perceber que a técnica de treinamento com *lottery tickets* pode colaborar para a evolução do modelo e distanciamento dos casos de *overfitting* (superajuste nos casos de treinamento) e que existe muito a ser explorado podendo em estudos futuros utilizar outras métricas para a obtenção dos melhores modelos dentros dos janelamentos. Por fim, ressaltamos que o objetivo do trabalho foi concluído e que o arcabouço tem uma detecção satisfatória dos casos adversariais explorados.

AGRADECIMENTOS

Os autores agradecem o ICT-UNIFESP pela estrutura computacional utilizada nos experimentos e o Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela concessão da bolsa de Iniciação Científica PIBIC e projeto Universal (#408919/2016-7).

REFERÊNCIAS

[1] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations*, 2015.

[2] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[3] A. M. A.-A. Selami and A. F. Fadhil, "A study of the effects of gaussian noise on image features," *kirkuk university journal for scientific studies*, vol. 11, no. 3, pp. 152–169, 2016.

[4] A. C. Bovik, *Handbook of image and video processing*. Academic press, 2010.

[5] S. W. Hasinoff, "Photon, poisson noise." 2014.

[6] D. Kuan, A. Sawchuk, T. Strand, and P. Chavel, "Adaptive restoration of images with speckle," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 3, pp. 373–383, 1987.

[7] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. IEEE, 2017, pp. 39–57.

[8] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[9] V. N. e. G. H. Alex Krizhevsky, "Cifar-10 and cifar-100 datasets," https://www.cs.toronto.edu/~kriz/cifar.html, (Acessado em 04/07/2020).

[10] scikit image, "scikit-image 0.18.dev0 docs — skimage v0.18.dev0 docs," https://scikit-image.org/docs/dev/, (Acessado em 29/07/2020).

[11] N. Papernot, I. Goodfellow, R. Sheatsley, R. Feinman, and P. McDaniel, "tensorflow / cleverhans: uma biblioteca de exemplo adversária para construir ataques, construir defesas e comparar ambos," https://github.com/tensorflow/cleverhans, (Acessado em 08/09/2020).

[12] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[13] scipy, "scipy.ndimage.median_filter — scipy v1.5.2 reference guide," https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.median_filter.html, (Accessed on 07/29/2020).

[14] A. Chambolle, "An algorithm for total variation minimization and applications," *Journal of Mathematical imaging and vision*, vol. 20, no. 1-2, pp. 89–97, 2004.

[15] S. G. Chang, B. Yu, and M. Vetterli, "Adaptive wavelet thresholding for image denoising and compression," *IEEE transactions on image processing*, vol. 9, no. 9, pp. 1532–1546, 2000.

[16] O. S. C. Vision, "Opencv: Image denoising," https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html, (Acessado em 08/07/2020).