FUNDAÇÃO UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

FACULDADE DE ENGENHARIAS, ARQUITETURA E URBANISMO E GEOGRAFIA

ENGENHARIA ELÉTRICA

Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais

Paulo Augusto Courbassier Simões

Campo Grande MS

14 de dezembro de 2020

FUNDAÇÃO UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

FACULDADE DE ENGENHARIAS, ARQUITETURA E URBANISMO E GEOGRAFIA

ENGENHARIA ELÉTRICA

Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais

Paulo Augusto Courbassier Simões

Orientador: Wesley Nunes Gonçalves

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Mato Grosso do Sul na Faculdade de Engenharias, Arquitetura e Urbanismo e Geografia, como requisito parcial para obtenção do título de Engenheiro(a) Eletricista.

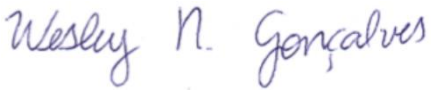
Campo Grande MS

14 de dezembro de 2020

Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais

Monografia apresentada à Universidade Federal de Mato Grosso do Sul na Faculdade de Engenharias, Arquitetura e Urbanismo e Geografia, para obtenção da Graduação em Engenharia Elétrica.

Banca Examinadora:



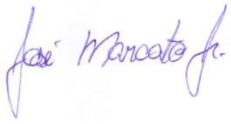
_____

Prof. Dr. Wesley Nunes Gonçalves

Orientador

_____

Prof. Dr. Edson Antonio Batista



_____

Prof. Dr. José Marcato Junior

Campo Grande MS

14 de dezembro de 2020

DECLARAÇÃO DE AUTORIA E RESPONSABILIDADE

Paulo Augusto Courbassier Simões, residente e domiciliado na cidade de Campo Grande, Estado do Mato Grosso do Sul, portador do RG de nº 1507499 e CPF nº 021.619.531-41, declaro que o "Trabalho de conclusão de curso" apresentado, com o título "Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais" é de minha autoria e assumo a total responsabilidade pelo seu conteúdo e pela originalidade do texto. Declaro que identifiquei e referenciei todas as fontes e informações gerais que foram utilizadas para construção do presente texto. Declaro também que este artigo não foi publicado, em parte, na íntegra ou conteúdo similar em outros meios de comunicação, tendo sido enviado com exclusividade para a Universidade Federal de Mato Grosso do Sul (UFMS).

Campo Grande, 14 de dezembro de 2020.

_____

Paulo Augusto Courbassier Simões

AGRADECIMENTOS

Gostaria de agradecer toda a minha família, em especial a minha mãe, Suely, que sempre me apoiou em toda essa jornada, sem isso eu, com certeza, não teria chego até aqui.

Agradeço também ao meu professor e orientador Wesley Nunes Gonçalves, que acreditou em mim e me ajudou durante cada etapa do projeto.

RESUMO

As linhas de transmissão percorrem vastas áreas, muitas vezes percorrendo terrenos de difícil acesso. Por causa disso, o trabalho de localiza-las para realizar construções ou observar o estado dos equipamentos e vegetação em sua volta para a realização de uma manutenção é muito trabalhoso e consome bastante tempo, portanto a automatização desse processo é ideal. Este trabalho propõe a utilização de redes neurais convolucionais para realizar essa tarefa. O trabalho foi dividido em três etapas, uma para a definição do banco de dados que será utilizado, contendo 4000 imagens de referência, a criação e treinamento do algoritmo e a validação dos métodos utilizados. Foram comparadas sete arquiteturas, a EfficientNetB0 [15], EfficientNetB7 [15], ResNet50V2 [17], ResNet101V2 [17], ResNet152V2 [17], VGG16 [16], VGG19 [16], as quais tiveram seus resultados comparados por precisão e tempo de inferência. Os valores variaram de 89,59% a 98,73% na medida F, e velocidade de inferência variando de 16,70 *FPS* (quadros por segundo) até 28,23 *FPS.* A EfficientNet B0 obteve o melhor desempenho médio geral dentre as arquiteturas testadas.

Palavras-Chave: Rede neural profunda, VANTs, aprendizado de máquina.

ABSTRACT

The transmission lines cover a large area, most of times going through some places of hard access. Because of this, the task of locating them to carry out constructions or observe the equipment conditions and vegetation around them for maintenance is very laborious and time consuming, so the automation of this process is ideal. This work proposes the use of convolutional neural networks to realize this task. This study is divided in three steps, one for the definition of the dataset, which has 4000 images as reference, then was the creation and training of the algorithm and finally the validation of the methods. Seven architectures were compared the EfficientNetB0 [15], EfficientNetB7 [15], ResNet50V2 [17], ResNet101V2 [17], ResNet152V2 [17], VGG16 [16], VGG19 [16], whose results ranged from 89.59% to 98.73% for the F score and 16.70 *FPS* (frames per second) to 28.23 *FPS*. With the EfficientNet B0 accomplishing the best average results.

Keywords: Deep neural network, UAV, machine learning.

LISTA DE FIGURAS

Número Página

Figura 1 – Exemplos de imagens com linha presentes no banco de dados. 16
Figura 2 – Exemplos de imagens sem linha presentes no bando de dados..... 16
Figura 3 – Exemplos de imagens de um mapa de ativação das arquiteturas. 17
Figura 4– Exemplos de aumento de dados utilizados. 18
Figura 5 – Exemplos de aumento de dados de rotação. 19
Figura 6– *Workflow* do processo de divisão das imagens e treinamento..... 20
Figura 7 – Precisão e perda, respectivamente, da arquitetura EfficientNet B0 sem aumento de dados. 21
Figura 8 – Precisão e perda, respectivamente, da arquitetura EfficientNet B0 com aumento de dados. 21
Figura 9 – Precisão e perda, respectivamente, da arquitetura EfficientNet B7 sem aumento de dados. 22
Figura 10 – Precisão e perda, respectivamente, da arquitetura EfficientNet B7 com aumento de dados. 22
Figura 11 – Precisão e perda, respectivamente, da arquitetura ResNet 50 V2 sem aumento de dados. 22
Figura 12 – Precisão e perda, respectivamente, da arquitetura ResNet 50 V2 com aumento de dados. 23
Figura 13 – Precisão e perda, respectivamente, da arquitetura ResNet 101 V2 sem aumento de dados. 23
Figura 14 – Precisão e perda, respectivamente, da arquitetura ResNet 101 V2 com aumento de dados. 23
Figura 15 – Precisão e perda, respectivamente, da arquitetura ResNet 152 V2 sem aumento de dados. 24
Figura 16 – Precisão e perda, respectivamente, da arquitetura ResNet 152 V2 com aumento de dados. 24
Figura 17 – Precisão e perda, respectivamente, da arquitetura VGG 16 sem aumento de dados. 24
Figura 18 – Precisão e perda, respectivamente, da arquitetura VGG 16 com aumento de dados. 25

Figura 19 – Precisão e perda, respectivamente, da arquitetura VGG 19 sem aumento de dados. 25
Figura 20 – Precisão e perda, respectivamente, da arquitetura VGG 19 com aumento de dados. 25
Figura 21– Exemplos de falsos positivos da EfficientNet B7, sem aumento de dados. 29
Figura 22– Exemplos de falsos negativos da EfficientNet B7, sem aumento de dados..... 29
Figura 23– Exemplos de falsos positivos da EfficientNet B0, sem aumento de dados.....30
Figura 24 – Exemplos de falsos negativos da EfficientNet B0, sem aumento de dados..... 30
Figura 25– Exemplos de falsos positivos da VGG 19, sem aumento de dados. 30
Figura 26 – Exemplos de falsos negativos da VGG 19, sem aumento de dados. 30
Figura 27 – Comparativo direto de velocidade e medida F. 32

LISTA DE TABELAS

Número Página

Tabela 1: Comparação do treinamento e validação das arquiteturas, sem aumento de dados. .. 26

Tabela 2: Comparação do treinamento e validação das arquiteturas, com aumento de dados. . 26

Tabela 3: Comparação estatística das arquiteturas, sem aumento de dados no conjunto de validação. 27

Tabela 4: Comparação estatística das arquiteturas, com aumento de dados no conjunto de validação. 27

Tabela 5: Comparação do tempo de inferência das arquiteturas, sem aumento de dados. 28

Tabela 6: Comparação do tempo de inferência das arquiteturas, com aumento de dados. 28

SUMÁRIO

1. INTRODUÇÃO 12

1.1. Justificativa para o desenvolvimento do trabalho 12

1.2. Estado da arte 12

1.3. Objetivos 13

1.3.1. Objetivo Geral 13

1.3.2. Objetivos Específicos 13

1.4. Organização do Trabalho: 13

2. Metodologia e Desenvolvimento 15

2.1. desenvolvimento do trabalho 15

2.2. Conclusão do desenvolvimento 21

3. Resultados 26

3.1. Discussão dos resultados 28

4. Conclusões Gerais 31

5. Referências 33

6. Apêndice A-Algorítmo 35

1. INTRODUÇÃO

1.1. JUSTIFICATIVA PARA O DESENVOLVIMENTO DO TRABALHO

As linhas de transmissão percorrem extensões muito grandes e devem ter uma faixa de servidão a sua volta [8]. Por isso, faz-se necessário saber a localização dessas linhas quando se quer realizar construções no terreno onde possa haver sua presença ou observar o estado de seus equipamentos e vegetação a sua volta para a realização de uma manutenção. Porém esse trabalho acaba sendo realizado de forma manual por um ser humano[19][20]. Dessa forma este projeto propõe a utilização de Redes Neurais Convolucionais como o primeiro passo para automatizá-lo.

VANTS (veículos aéreos não tripulados) poderão aplicar este algoritmo de forma a realizar essa identificação de forma autônoma. Este projeto espera, também, ser mais um ponto de referência para a utilização de Redes Neurais Convolucionais para a autonomia de sistemas de detecção de linhas de transmissão.

1.2. ESTADO DA ARTE

O uso de inteligência artificial está crescendo cada vez mais na comunidade [14] e Redes Neurais Convolucionais é uma técnica amplamente utilizada para o reconhecimento e classificação de imagens [18]. Em 2017 Nguyen et al. [3] fizeram um estudo sobre Redes Neurais Convolucionais como solução para o problema de detecção e mapeamento dos componentes de uma linha de transmissão ou distribuição de forma completamente autônoma, onde foram argumentadas as vantagens de um sistema que aprende sozinho à detectar as linhas e seus componentes para a substituição dos serviço manual. Em 2019, Dong et al. [4] desenvolveram um método onde é possível identificar a linhas e medir sua curvatura com uma precisão de 99,94%. Criando outra ferramenta para a detecção autônoma de problemas em LTs (linhas de transmissão). Em 2020 Nguyen et al. [2] criaram uma Rede Neural Convolucional utilizando a LS-Net para criar um sistema de detecção que ocorre quase em tempo real (20.4 *FPS*).

1.3. OBJETIVOS

1.3.1. Objetivo Geral

Tem-se, como objetivo, criar um algoritmo capaz de reconhecer a existência de linhas de transmissão através de imagens capturadas por VANTs (Veículos Aéreos Não Tripulados) utilizando as teorias de Redes Neurais Convolucionais, através da técnica de transferência de conhecimento e comparação de desempenho das arquiteturas recentes.

1.3.2. Objetivos Específicos

- Avaliar as arquiteturas recentes de redes neurais convolucionais que utilizam a transferência de conhecimento da ImageNet [13] entre a EfficientNetB0 [15], EfficientNetB7 [15], ResNet50V2 [17], ResNet101V2 [17], ResNet152V2 [17], VGG16 [16] e VGG19 [16] tem o melhor desempenho para a tarefa proposta.

- Verificar se o aumento de dados aplicado na detecção de linhas aumenta a acurácia. E se sim, para quais arquiteturas.

- Observar se apenas imagens RGB obtêm resultados satisfatórios. Pois a visualização das linhas no espectro visível pode ser dificultada em algumas imagens.

1.4. ORGANIZAÇÃO DO TRABALHO:

No Capítulo 1, é apresentada uma revisão bibliográfica básica no intuito de situar o trabalho no contexto geral, e a motivação para o desenvolvimento do trabalho.

No Capítulo 2, é apresentada com detalhes o desenvolvimento do projeto de Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais.

No Capítulo 3, são apresentados resultados sobre o Reconhecimento de linhas de transmissão em imagens usando redes neurais convolucionais.

No Capítulo 4, são apresentadas as conclusões finais e propostas de continuidade do trabalho.

2. METODOLOGIA E DESENVOLVIMENTO

Neste capítulo descreve-se os processos e ferramentas utilizadas para o desenvolvimento deste trabalho. Para melhor atender aos objetivos, este foi dividido em 3 etapas:

Etapa 1: **Criação de um banco de dados**. Essa é uma etapa crucial do projeto, pois é a partir desse que o treinamento do modelo poderá ser feito. Esse trabalho utilizou o banco de dados "Powerline Image Dataset (*Infrared-IR and Visible Light-VL*)" [9] o qual contém uma coletânea de 4000 imagens em luz visível e 4000 em infravermelho, estando divididas com metade sendo imagens com a presença de linhas de transmissão e a outra metade sem a linhas. Este trabalho utilizou-se apenas das imagens em luz visível.

Etapa 2: **Implementação dos métodos**. Foi criado um algoritmo na linguagem Python [Anexo A] utilizando o Google Colab, um ambiente de programação em nuvem para *machine learning*, as bibliotecas do TensorFlow [12], do keras [11] e do Scikit Learn [10]. Todas criadas para o desenvolvimento de projetos de Redes Neurais Convolucionais. Através dessas, foi-se realizando o treinamento diversas vezes, empiricamente, com números diferentes de tamanho de passos e gerações, de forma a evitar *overfitting* e se obter os melhores resultados.

Etapa 3: **Validação dos métodos**. Nessa etapa da pesquisa os métodos foram executados e avaliados através dos resultados obtidos. Os quais foram organizados por arquitetura utilizada, sendo feita a comparação entre estas e depois entre técnicas já existentes na literatura.

2.1. DESENVOLVIMENTO DO TRABALHO

A criação do banco de dados é um dos pontos principais de qualquer projeto de Redes Neurais Convolucionais. A partir deste que todo o treinamento e validação se baseia. Tal banco deve conter imagens variadas e claras sobre as categorias a serem divididas, nesse caso, com linha e sem linha linhas de transmissão.

Então, utilizou-se o banco de dados "Powerline Image Dataset (*Infrared-IR and Visible Light-VL*)" [9] o qual tem 8000 imagens, das quais foram utilizadas 4000 em luz visível, divididas

em 2000 contendo linha e 2000 sem linha. Todas com resolução de 180x180 pixels, divididas em 80% para treino e 20% para validação.

Nas Figuras 1 e 2 apresentam-se exemplos de imagens com e sem linha de transmissão, respectivamente. Pode-se observar que o banco de imagens apresenta uma variabilidade considerável, o que é importante para avaliar os métodos em diversos cenários. Enquanto na Figura 3 observam-se exemplos de mapas de ativação.

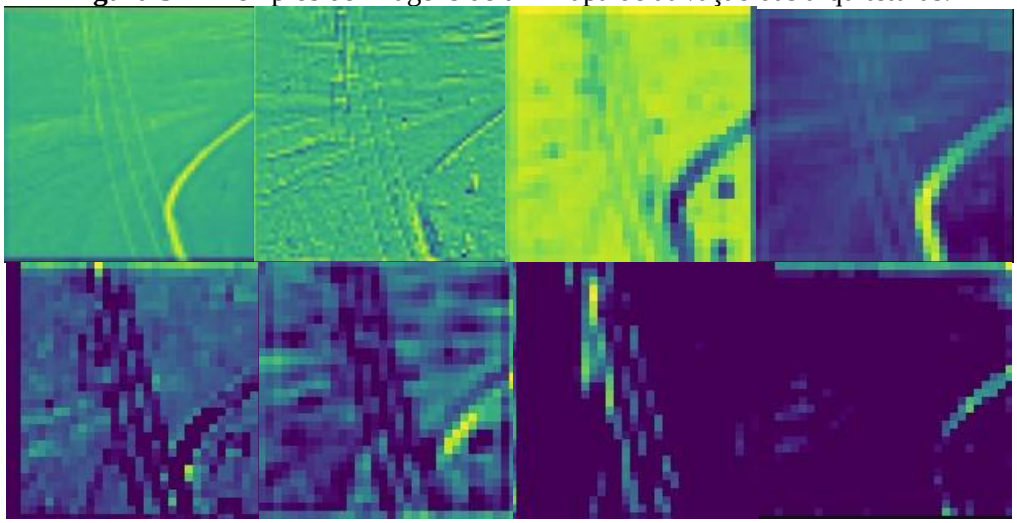
Figura 1 – Exemplos de imagens com linha presentes no banco de dados.

Fonte: Powerline Image Dataset (*Infrared-IR and Visible Light-VL*).

Figura 2 – Exemplos de imagens sem linha presentes no bando de dados.

Fonte: Powerline Image Dataset (*Infrared-IR and Visible Light-VL*).

Figura 3 – Exemplos de imagens de um mapa de ativação das arquiteturas.



Fonte: Autoria própria.

As imagens originais desse banco de dados estão em .bmp, um formato contendo quatro canais, RGBA, (vermelho, verde, azul e alpha), porém, o código utilizado foi projetado para os três canais de cores padrão, RGB, com isso, fez-se necessária a conversão das imagens para o formato .png. E esse novo banco de dados foi salvo no Google Drive para fácil utilização com o Google Colab.

Com o banco de dados estabelecido, começou-se a estruturação do código, o qual foi todo feito na linguagem Python, através do Google Colab. O Colab é uma ferramenta para programação em nuvem com foco em aprendizado de máquinas, tendo suporte para uma grande variedade de bibliotecas que auxiliam no desenvolvimento de algoritmos para o treinamento de redes neurais. Uma grande vantagem dessa ferramenta é a capacidade de se utilizar os computadores dos servidores da Google para realizar o treinamento, o qual tem as seguintes características: 1 GPU Tesla P100 da Nvidia, um processador Xeon Intel com 2.2Ghz e 13GB de Memória RAM.

Dentre as várias bibliotecas utilizadas, três serão destacadas por serem desenvolvidas para redes neurais convolucionais. O TensorFlow [12], Keras [11] e o Scikit [10]. Todas com tutoriais bem explicativos, os quais foram de supra importância para o aprendizado do acadêmico e desenvolvimento do código final. No qual, parte desse, utilizou o tutorial de classificação de flores através de redes neurais convolucionais do TensorFlow [12] como sua base. Nessa estrutura existe uma parte utilizada para a aumento de dados, o que consiste em criar novas imagens a partir das imagens existentes. Esse projeto espelhou imagens verticalmente, horizontalmente e também

aplicou mudanças de contraste, todos realizados de forma aleatória em uma taxa de 20% do banco de dados total.

Na Figura 4 mostra-se exemplos das imagens modificadas com o aumento de dados, pode-se notar que todas as imagens exemplificadas são baseadas na mesma figura, mas resultam em dados completamente novos para a máquina aprender.

Figura 4– Exemplos de aumento de dados utilizados.

Fonte: Modificação da Powerline Image Dataset (*Infrared-IR and Visible Light-VL*).

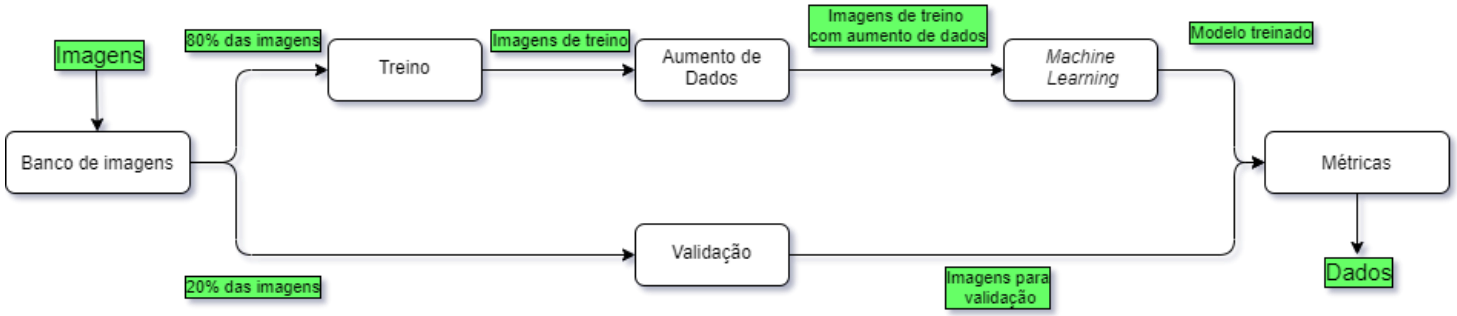
Na Figura 5 apresenta-se a tentativa de aplicar o aumento de dados com rotação, porém, pela baixa qualidade das imagens utilizadas, o resultado ficou distorcido, sendo assim abandonado nos treinos finais.

Figura 5 – Exemplos de aumento de dados de rotação.

Fonte: Modificação da Powerline Image Dataset (*Infrared-IR and Visible Light-VL*).

Utilizou-se a biblioteca Keras [11] para a aplicação da técnica de "Transferência de conhecimento" para a criação do modelo. Essa técnica consiste em treinar uma arquitetura pré-treinada em uma base de imagens consideravelmente maior, aproveitando os pesos conhecidos na nova aplicação. As arquiteturas EfficientNetB0 [15], EfficientNetB7 [15], ResNet50V2 [17], ResNet101V2 [17], ResNet152V2 [17], VGG16 [16], VGG19 [16] foram pré-treinadas no banco de dados da ImageNet [13] e utilizadas para a realização da tarefa proposta. A Figura 6 é um *workflow* desse processo.

Figura 6– *Workflow* do processo de divisão das imagens e treinamento.



Fonte: Autoria própria

E por fim a biblioteca SciKit [10] foi utilizada para a coletar dados do desempenho de cada arquitetura. Os dados coletados são expressos pelas Equações 1, 2 e 3:

Precisão:

Capacidade de evitar falsos positivos.

$$P = \frac{T_P}{T_P + F_p} \qquad \text{Equação 1}$$

Revocação:

Capacidade de evitar falsos negativos.

$$R = \frac{T_P}{T_P + F_n} \qquad \text{Equação 2}$$

Medida F:

Média harmônica entra precisão e revocação.

$$F1 = 2x\frac{PxR}{P+R} \qquad \text{Equação 3}$$

Sendo:

Tp é o total de verdadeiros positivos.

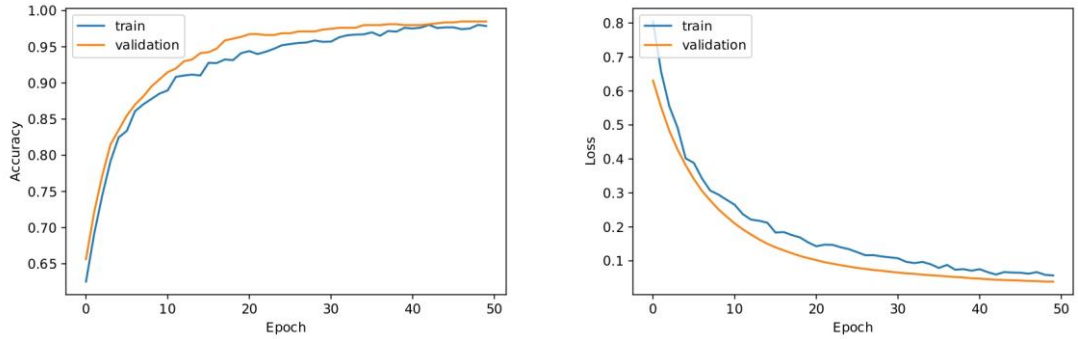
Fp são os falsos positivos.

Fn são os falsos negativos.

2.2. CONCLUSÃO DO DESENVOLVIMENTO

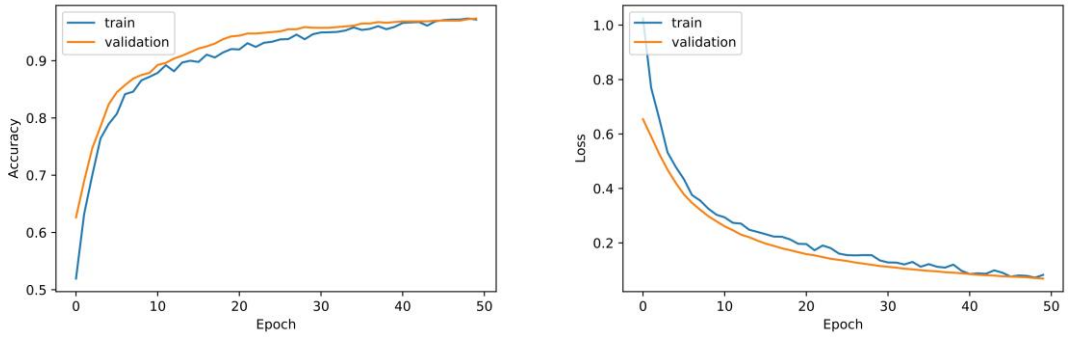
O banco de dados contendo 4000 imagens separadas em duas categorias, foi dividido em 3200 imagens para treino e 800 para validação. Na decisão dos meta parâmetros, foram testados empiricamente valores entre 10^{-4} até 10^{-6} para a taxa de aprendizagem e 50 a 100 para o número de épocas. Os gráficos das Figuras 7 até a Figura 20 mostram os resultados do treinamento e validação, é possível notar que os treinamentos foram adequados pois a perda e acurácia tanto no conjunto de treinamento como validação foram próximos.

Figura 7 – Precisão e perda, respectivamente, da arquitetura EfficientNet B0 sem aumento de dados.



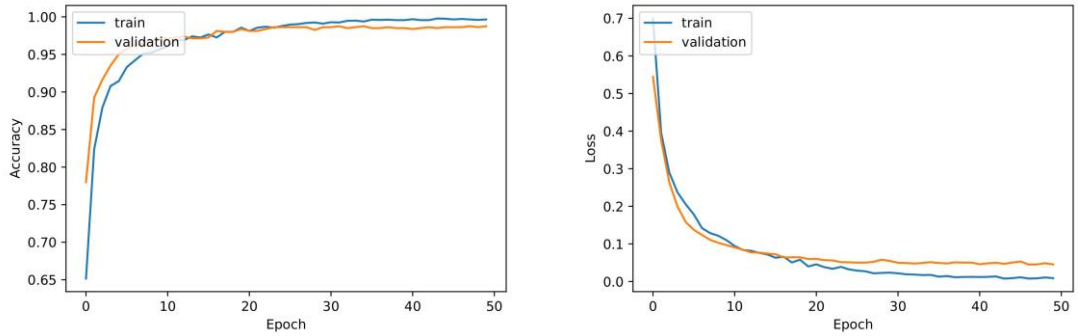
Fonte: Autoria própria

Figura 8 – Precisão e perda, respectivamente, da arquitetura EfficientNet B0 com aumento de dados.



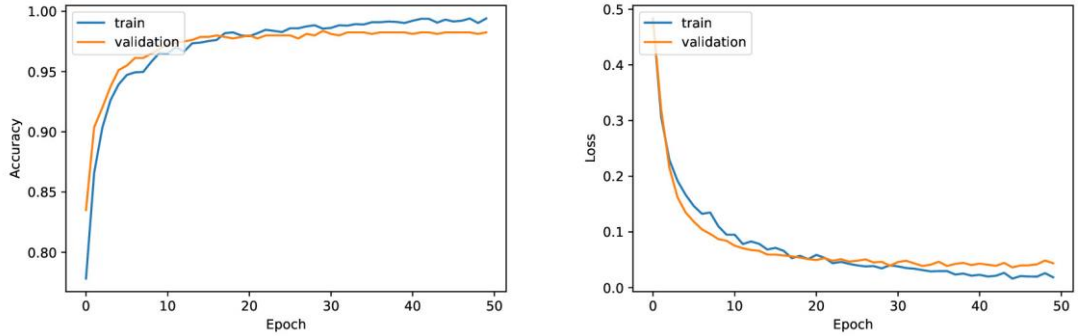
Fonte: Autoria própria

Figura 9 – Precisão e perda, respectivamente, da arquitetura EfficientNet B7 sem aumento de dados.



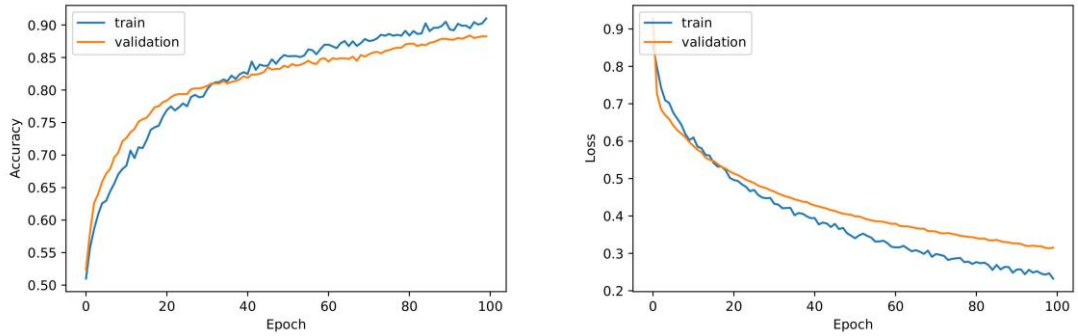
Fonte: Autoria própria

Figura 10 – Precisão e perda, respectivamente, da arquitetura EfficientNet B7 com aumento de dados.



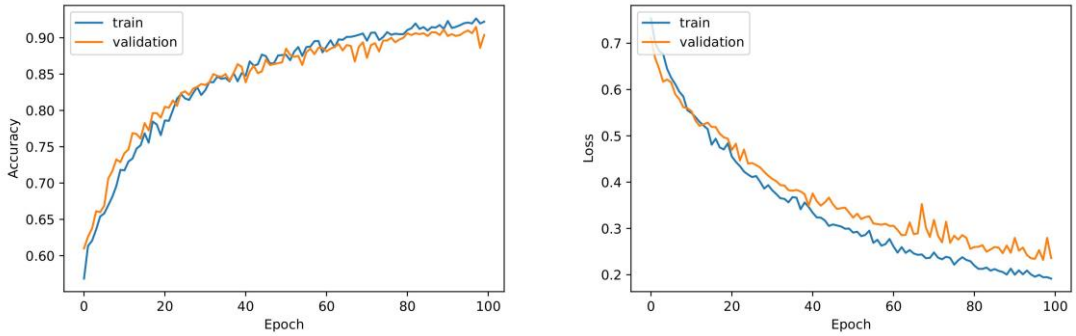
Fonte: Autoria própria

Figura 11 – Precisão e perda, respectivamente, da arquitetura ResNet 50 V2 sem aumento de dados.



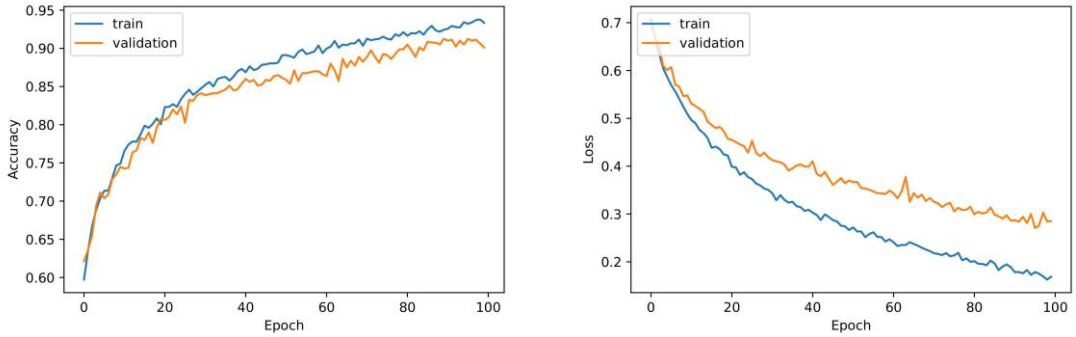
Fonte: Autoria própria

Figura 12 – Precisão e perda, respectivamente, da arquitetura ResNet 50 V2 com aumento de dados.



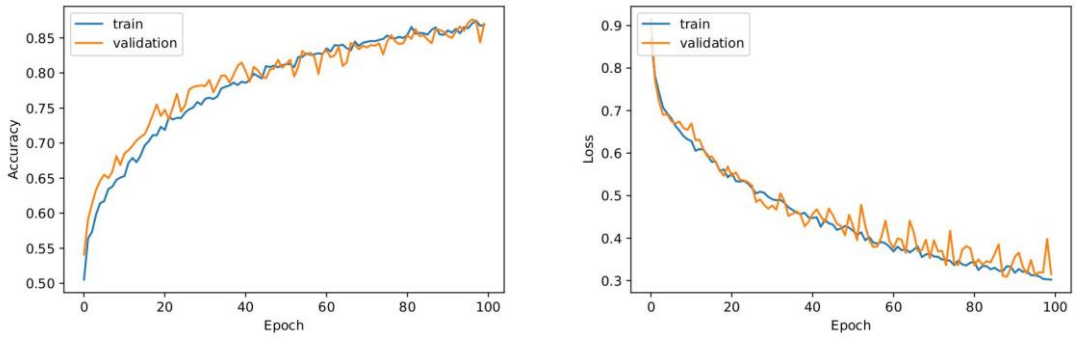
Fonte: Autoria própria

Figura 13 – Precisão e perda, respectivamente, da arquitetura ResNet 101 V2 sem aumento de dados.



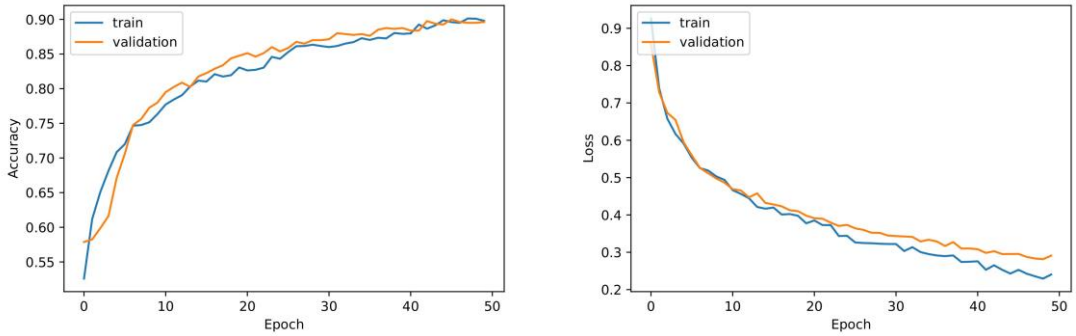
Fonte: Autoria própria

Figura 14 – Precisão e perda, respectivamente, da arquitetura ResNet 101 V2 com aumento de dados.



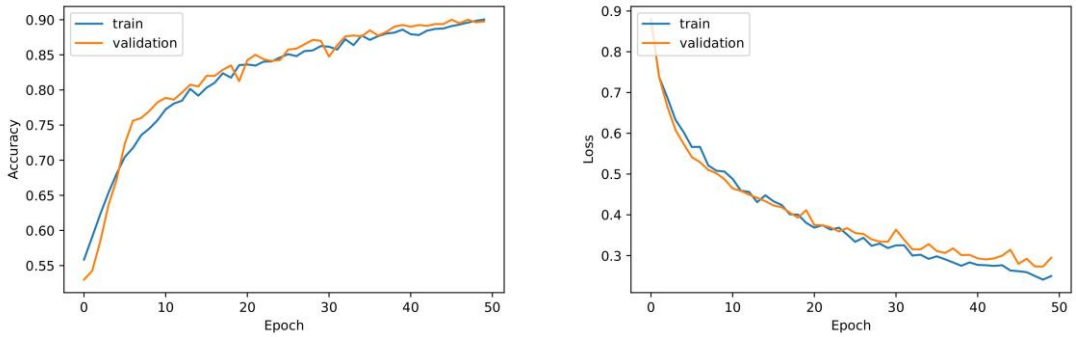
Fonte: Autoria própria

Figura 15 – Precisão e perda, respectivamente, da arquitetura ResNet 152 V2 sem aumento de dados.



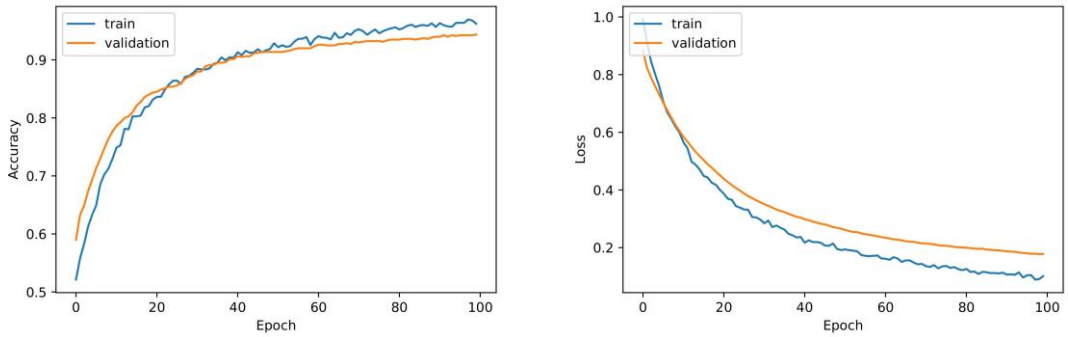
Fonte: Autoria própria

Figura 16 – Precisão e perda, respectivamente, da arquitetura ResNet 152 V2 com aumento de dados.



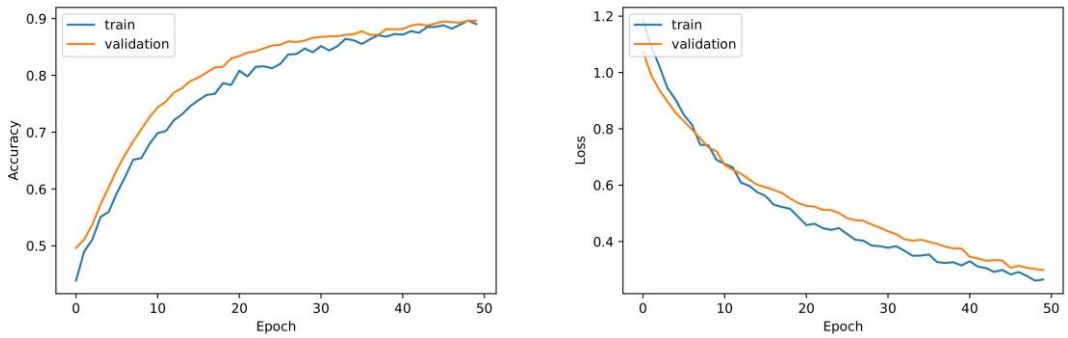
Fonte: Autoria própria

Figura 17 – Precisão e perda, respectivamente, da arquitetura VGG 16 sem aumento de dados.



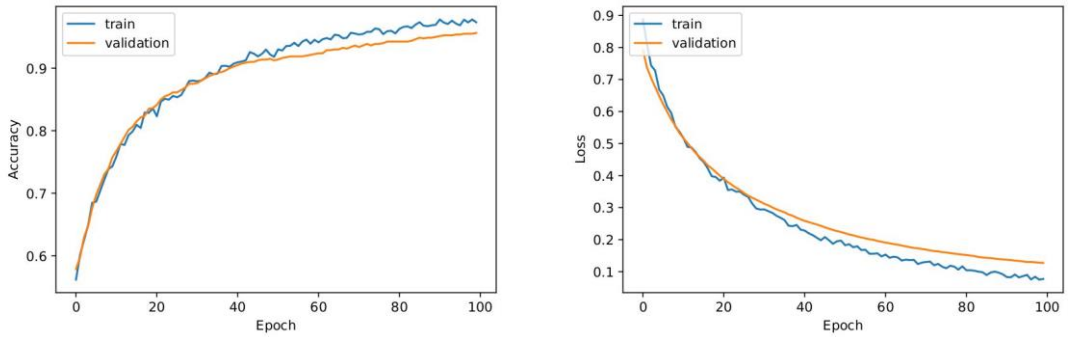
Fonte: Autoria própria

Figura 18 – Precisão e perda, respectivamente, da arquitetura VGG 16 com aumento de dados.



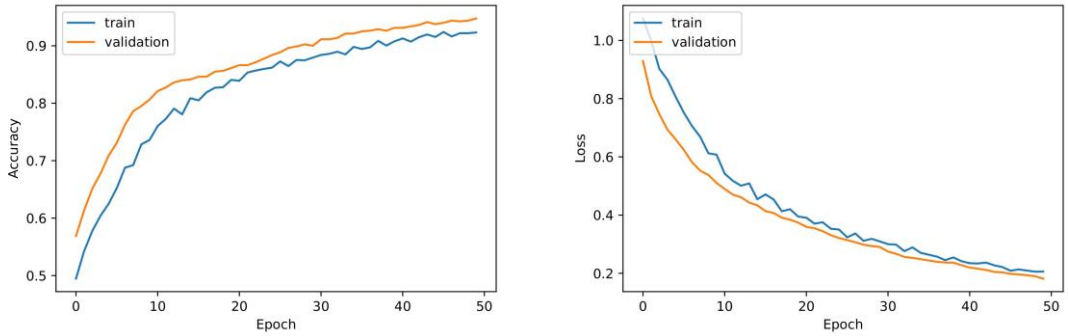
Fonte: Autoria própria

Figura 19 – Precisão e perda, respectivamente, da arquitetura VGG 19 sem aumento de dados.



Fonte: Autoria própria

Figura 20 – Precisão e perda, respectivamente, da arquitetura VGG 19 com aumento de dados.



Fonte: Autoria própria

3. RESULTADOS

Nesta seção serão expostos e discutidos os resultados obtidos ao final do projeto, divididos em dois comparativos entre as arquiteturas EfficientNetB0 [15], EfficientNetB7 [15], ResNet50V2 [17], ResNet101V2 [17], ResNet152V2 [17], VGG16 [16], VGG19 [16]. O primeiro sem aumento de dados e depois um com aumento de dados.

Nas Tabelas 1 e 2 apresentam-se os resultados de perda e acurácia obtidos durante o treinamento com aumento de dados e sem aumento de dados, respectivamente.

Tabela 1: Comparação do treinamento e validação das arquiteturas, sem aumento de dados.

	EfficientNet B0	**EfficientNet B7**	**ResNet 50V2**	**ResNet 101V2**	**ResNet 152V2**	**VGG 16**	**VGG 19**
Perda em treino	0,057	0,009	0,233	0,169	0,240	0,101	0,070
Acurácia em Treino	97,87%	99,66%	91,00%	93,34%	89,78%	96,25%	97,34%
Perda em Validação	0,098	0,045	0,315	0,285	0,291	0,178	0,123
Acurácia em Validação	98,50%	98,75%	88,25%	90,13%	89,63%	94,38%	95,75%

Fonte: Autoria própria

Tabela 2: Comparação do treinamento e validação das arquiteturas, com aumento de dados.

	EfficientNet B0	**EfficientNet B7**	**ResNet 50V2**	**ResNet 101V2**	**ResNet 152V2**	**VGG 16**	**VGG 19**
Perda em treino	0,082	0,019	0,110	0,192	0,250	0,265	0,206
Acurácia em Treino	97,19%	99,41%	95,94%	92,22%	90,03%	89,03%	92,31%
Perda em Validação	0,069	0,044	0,174	0,236	0,295	0,299	0,182
Acurácia em Validação	97,37%	98,25%	94,38%	90,38%	89,75%	89,63%	94,75%

Fonte: Autoria própria

Nas as Tabelas 3 e 4 comparam-se as estatísticas de revocação, precisão, medida F, assim como os valores absolutos da quantidade de acertos e erros em cada categoria, sem aumento de dados e com aumento de dados, respectivamente.

Tabela 3: Comparação estatística das arquiteturas, sem aumento de dados no conjunto de validação.

	EfficientNet B0	**EfficientNet B7**	**ResNet 50V2**	**ResNet 101V2**	**ResNet 152V2**	**VGG 16**	**VGG 19**
Revocação	97,74%	97,99%	86,43%	86,68%	85,43%	92,71%	96,48%
Precisão	99,24%	99,49%	89,58%	92,99%	93,15%	95,84%	94,82%
Medida F	98,48%	98,73%	87,98%	89,73%	89,12%	94,25%	95,64%
Verdadeiros Positivos	399	400	362	376	377	386	381
Verdadeiros Negativos	389	390	344	345	340	369	384
Falsos Positivos	9	8	54	53	58	29	14
Falsos Negativos	3	2	40	26	25	16	21

Fonte: Autoria própria

Tabela 4: Comparação estatística das arquiteturas, com aumento de dados no conjunto de validação.

	EfficientNet B0	**EfficientNet B7**	**ResNet 50V2**	**ResNet 101V2**	**ResNet 152V2**	**VGG 16**	**VGG 19**
Revocação	96,73%	96,99%	94,98%	88,69%	83,17%	89,70%	94,47%
Precisão	97,96%	99,49%	94,38%	91,69%	95,67%	89,47%	94,95%
Medida F	97,35%	98,22%	94,38%	90,17%	88,98%	89,59%	94,71%
Verdadeiros Positivos	394	400	377	370	387	360	382
Verdadeiros Negativos	385	386	378	353	331	357	376
Falsos Positivos	13	12	20	45	67	41	22
Falsos Negativos	8	2	25	32	15	42	20

Fonte: Autoria própria

Os tempos de inferência médio, seu desvio padrão e *FPS*, sem e com aumento de dados, respectivamente, estão nas Tabelas 5 e 6.

Tabela 5: Comparação do tempo de inferência das arquiteturas, sem aumento de dados.

	EfficientNet B0	**EfficientNet B7**	**ResNet 50V2**	**ResNet 101V2**	**ResNet 152V2**	**VGG 16**	**VGG 19**
Tempo médio de Inferência	39,47 ms	59,16 ms	39,69 ms	50,27 ms	52,65 ms	35,18 ms	37,15 ms
Desvio padrão do tempo de inferência	38,45 ms	14,14 ms	36,78 ms	42,39 ms	35,28 ms	15,67 ms	04,81 ms
Quadros por segundo médio	25,34	16,90	25,20	19,89	18,99	28,43	26,92

Fonte: Autoria própria

Tabela 6: Comparação do tempo de inferência das arquiteturas, com aumento de dados.

	EfficientNet B0	***EfficientNet B7***	***ResNet 50V2***	***ResNet 101V2***	***ResNet 152V2***	***VGG 16***	***VGG 19***
Tempo médio de Inferência	36,52 ms	59,89 ms	39,80 ms	50,33 ms	52,58 ms	35,43 ms	36,71 ms
Desvio padrão do tempo de inferência	41,14 ms	13,64 ms	21,25 ms	23,26 ms	33,43 ms	30,17 ms	04,66 ms
Quadros por segundo médio	27,38	16,70	25,13	19,87	19,02	28,23	27,25

Fonte: Autoria própria

3.1. DISCUSSÃO DOS RESULTADOS

As arquiteturas EfficientNet B0 [15], EfficientNet B7 [15], VGG16 [16] e VGG19 [16], sem aumento de dados, obtiveram as melhores medidas F, com o menor valor sendo 94,25% para a VGG 16 [16] e o maior sendo para a EfficientNet B7 [15] com 98,73%. Dessas quatro arquiteturas, três foram as que realizaram as inferências no menor tempo, a EfficientNet B0 [15], VGG16 [16] e VGG19 [16] as quais resultaram em valores variando de 25,34 *FPS* até 28,43 *FPS*.

É notável que o aumento de dados trouxe uma melhoria significativa na taxa de acertos apenas para a arquitetura ResNet 50V2 [17], de 87,98% para 94,38%. Nas arquiteturas das EfficientNet B0 [15], EfficientNet B7 [15], ResNet 101V2 [17], ResNet 151V2 [17] e VGG 19 [16] a variação de precisão no treino foi de próxima de 1%, o que pode ser considerado um empate técnico. O efeito sobre o tempo de inferência foi muito pequeno para ser considerado expressivo. Esse resultado pode ser atribuído ao treino prévio na ImageNet [13], que foi feito com aumento de dados e já ensinou a rede neural a extrapolar as imagens.

A EfficientNet B7 [15], com a melhor precisão, teve apenas 8 falsos positivos e 2 falsos negativos, a Figura 21 e 22 mostram alguns desses erros.

Figura 21– Exemplos de falsos positivos da EfficientNet B7, sem aumento de dados.

Fonte: Autoria própria.

Figura 22– Exemplos de falsos negativos da EfficientNet B7, sem aumento de dados.

Fonte: Autoria própria.

As EfficientNet B0 [15] e a VGG 19 [16] obtiveram ao segunda e terceira melhor precisão, respectivamente, com 9 falsos positivos e 3 falsos negativos para a EfficientNet B0 [15] e 14 falsos positivos e 21 falsos negativos para a VGG 19 [16]. Nas Figuras 23 a 26 exemplificam-se alguns desses erros.

Figura 23– Exemplos de falsos positivos da EfficientNet B0, sem aumento de dados

Fonte: Autoria própria.

Figura 24 – Exemplos de falsos negativos da EfficientNet B0, sem aumento de dados.

Fonte: Autoria própria.

Figura 25– Exemplos de falsos positivos da VGG 19, sem aumento de dados.

Fonte: Autoria própria

Figura 26 – Exemplos de falsos negativos da VGG 19, sem aumento de dados.

Fonte: Autoria própria.

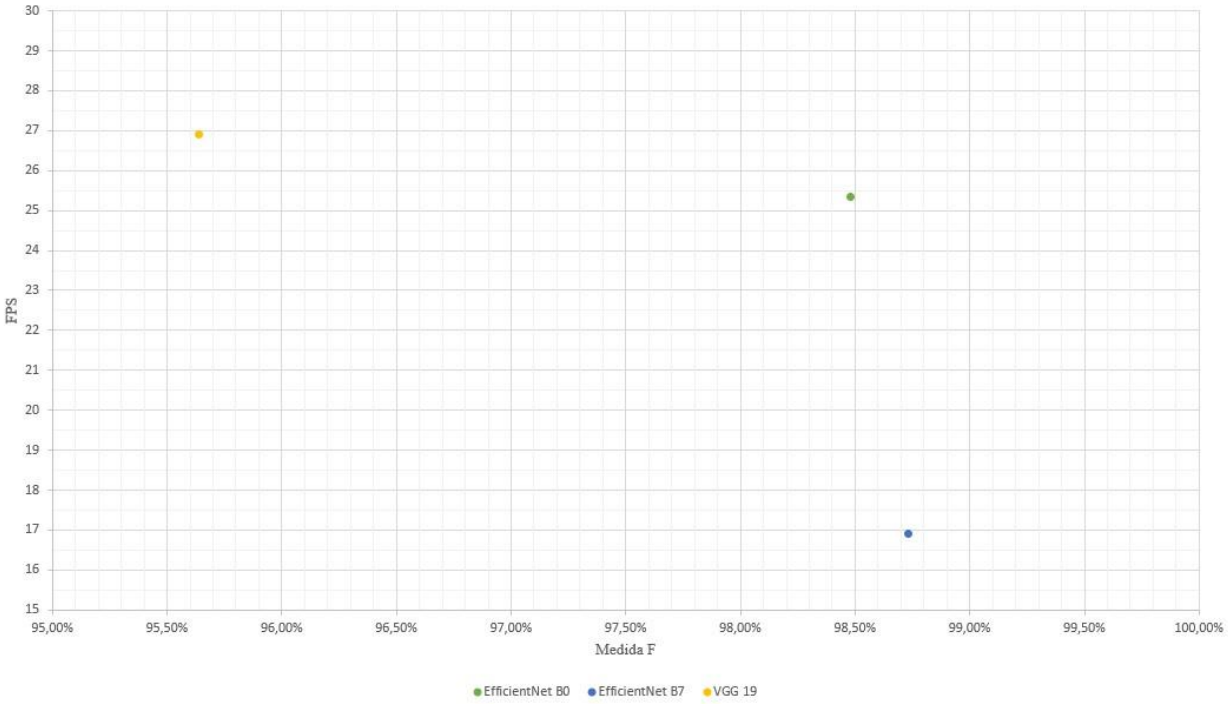
4. CONCLUSÕES GERAIS

Esse trabalho teve como objetivo avaliar arquiteturas recentes na classificação de linhas de transmissão. Nessa avaliação consideramos imagens RGB e a influência do aumento de dados na acurácia.

Os resultados mostraram que a EfficientNet B0 [15] e EfficientNet B7 [15], sem aumento de dados, obtiveram as melhores taxas de acertos verdadeiros dentre as analisadas (98,48% e 98,73%). A VGG 16 [16] e a VGG 19 [16] foram as mais rápidas (28,43 *FPS* e 26,92 *FPS*) e com a menor variação no tempo de inferência (15,67 ms e 04,81 ms).

Contudo, para o objetivo proposto, faz-se necessário tanto precisão como velocidade, apesar da alta velocidade das VGG 16 [16] e VGG 19 [16] suas precisões não foram muito altas, enquanto a EfficientNet B7 [15], com a melhor precisão entre todas, obteve o pior tempo de inferência (16,90 FPS). A Figura 27 mostra um comparativo direto entre a velocidade e a medida F das arquiteturas EfficientNet B0, EfficientNet B7 e a VGG 19. Onde a melhor opção é a rede mais à direita e a mais acima, simultaneamente. Portanto a EfficientNet B0 [15] tem os mais fortes indícios de ser a melhor escolha para a realização da atividade proposta, devido a sua alta velocidade (25,34 *FPS*) e precisão (Medida F de 98.48%).

Figura 27 – Comparativo direto de velocidade e medida F.



Fonte: Autoria própria.

Para a continuidade do trabalho, é proposto realizar o reconhecimento exato da posição das linhas de transmissão, para aplicar um sistema de controle aos VANTs, tornando-os completamente independentes para seguir linhas e desviar de obstáculos ao longo do percurso. Assim como o treinamento dessa rede neural convolucional para o reconhecimento do estado de manutenção em cada componente do sistema de transmissão. Sendo, assim, possível reconhecer e prevenir defeitos sem a necessidade de humanos terem que ir pessoalmente fazer essa verificação.

5. REFERÊNCIAS

x

[1] ABNT. (2018, Agosto) ABNT Catálogo. [Online]. http://www.abntcatalogo.com.br/norma.aspx?ID=86662

[2] Van Nhan Nguyen, Robert Jenssen and Davide Roverso. LS-Net: Fast Single-Shot Line-Segment Detector

[3] Van Nhan Nguyena, Robert Jenssen and Davide Roverso. Automatic Autonomous Vision-based Power Line Inspection: A Review of Current Status and the Potential Role of Deep Learning

[4] Jingjing Dong,Wei Chen and Chen Xu. Transmission line detection using deep convolutional neural network

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 25, pages 1097{ 1105. Curran Associates, Inc., 2012.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[7] Gomes, M.; Silva, J.; Gonçalves, D.; Zamboni, P.; Perez, J.; Batista, E.; Ramos, A.; Osco, L.; Matsubara, E.; Li, J.; Marcato Junior, J.; Gonçalves, W. Mapping Utility Poles in Aerial Orthoimages Using ATSS Deep Learning Method. Sensors 2020, 20, 6070.

[8] ABNT NBR 5422:1985 - Projeto de linhas aéreas de transmissão de energia elétrica

[9] Yetgin, Ömer Emre; GEREK, Ömer Nezih (2019), "Powerline Image Dataset (*Infrared-IR and Visible Light-VL*)", Mendeley Data, V8, em: https://data.mendeley.com/datasets/n6wrv4ry6v/8

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, Édouard Duchesnay; Scikit-learn: Machine Learning in Python, JMLR 12, pp. 2825-2830, 2011.

[11] Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras

[12] Abadi, Mart'in, Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … others. (2016). Tensorflow: A system for large-scale machine learning. In 12th $USENIX$ Symposium on Operating Systems Design and Implementation ($OSDI$ 16) (pp. 265–283).

[13] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248–255).

[14] Li, K.; Wan, G.; Cheng, G.; Meng, L.; Han, J. Object detection in optical remote sensing images: A survey 382 and a new benchmark. ISPRS Journal of Photogrammetry and Remote Sensing 2020, 159, 296 – 307. 383 doi:https://doi.org/10.1016/j.isprsjprs.2019.11.023.

[15] M Tan, QV Le, Efficientnet: Rethinking model scaling for convolutional neural networks. 2019. arXiv preprint arXiv:1905.11946.

[16] K. Simonyan, A Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv preprint arXiv:1409.1556

[17] K He, X Zhang, S Ren, J Sun, Deep Residual Learning for Image Recognition, Computer Vision and Pattern Recognition (CVPR), 2016.

[18] Samer Hijazi, Rishi Kumar, and Chris Rowen, IP Group, Cadence. Using Convolutional Neural Networks for Image Recognition. 2015

[19] TRIXEL. **INSPEÇÃO DE LINHAS DE TRANSMISSÃO.** Disponível em: <https://www.trixel.com.br/servico/inspecao-de-linhas-de-transmissao/>. Acesso em: 14 de dec de 2020.

[20] HEGARD. **As inspeções de linhas de transmissão sem drones.** Disponível em: <https://hegard.com.br/inspecao-de-linhas-de-transmissao-com-drones/>. Acesso em: 14 de dec de 2020.

6. APÊNDICE A - ALGORÍTMO

```
# Montar o drive, escolher a conta, copiar e colar a chave

from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

# Separação do dataset em treino e validação

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

path_dataset = '/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/Visible_Light_png'
IMG_SIZE = 128
batch_size = 32
NUM_CLASSES = 2

train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    path_dataset,
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=batch_size,
)
```

```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    path_dataset,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=batch_size,
)

# imprimir exemplos de imagens do dataset

import matplotlib.pyplot as plt

class_names = train_ds.class_names

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[int(labels[i])])
        plt.axis("off")

# Parte de melhoria de dados, imprimindo exemplos

data_augmentation = keras.Sequential(
    [
        #layers.experimental.preprocessing.RandomFlip("horizontal"),
        #layers.experimental.preprocessing.RandomFlip("vertical"),
        layers.experimental.preprocessing.RandomRotation(0.1),
        #layers.experimental.preprocessing.RandomContrast(0.2),
    ]
```

```
)
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
    for i in range(9):
        augmented_images = data_augmentation(images)
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(augmented_images[0].numpy().astype("uint8"))
        plt.axis("off")

# Optimização

def input_preprocess(image, label):
    label = tf.one_hot(label, NUM_CLASSES)
    return image, label

size = (IMG_SIZE, IMG_SIZE)
train_ds = train_ds.map(lambda image, label: (tf.image.resize(image, size), label))
val_ds = val_ds.map(lambda image, label: (tf.image.resize(image, size), label))

train_ds = train_ds.map(input_preprocess)
#train_ds = train_ds.batch(batch_size=batch_size, drop_remainder=True)
#train_ds = train_ds.prefetch(tf.data.experimental.AUTOTUNE)

val_ds = val_ds.map(input_preprocess)
#val_ds = val_ds.batch(batch_size=batch_size, drop_remainder=True)

# Model

from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras.applications import EfficientNetB0, EfficientNetB7, ResNet50V2,
ResNet101V2, ResNet152V2, VGG16, VGG19, InceptionResNetV2
```

```
def build_model(num_classes):
    inputs = layers.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
    x = data_augmentation(inputs)
    model = EfficientNetB0(include_top=False, input_tensor=inputs, weights="imagenet")

    # Freeze the pretrained weights
    model.trainable = False

    # Rebuild top
    x = layers.GlobalAveragePooling2D(name="avg_pool")(model.output)
    x = layers.BatchNormalization()(x)

    top_dropout_rate = 0.2
    x = layers.Dropout(top_dropout_rate, name="top_dropout")(x)
    outputs = layers.Dense(num_classes, activation="softmax", name="pred")(x)

    # Compile
    model = tf.keras.Model(inputs, outputs, name="EfficientNetB0")
    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-6)
    model.compile(
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )
    return model

def unfreeze_model(model):
    # We unfreeze the top 20 layers while leaving BatchNorm layers frozen
    for layer in model.layers:
        if not isinstance(layer, layers.BatchNormalization):
            layer.trainable = True

    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-7)
    model.compile(
```

```
        optimizer=optimizer, loss="categorical_crossentropy", metrics=["accuracy"]
    )
    return model


# Ininicar Treinamento

model = build_model(num_classes=NUM_CLASSES)
model = unfreeze_model(model)

epochs = 100
hist = model.fit(train_ds, epochs=epochs, validation_data=val_ds, verbose=2)

# Imprimir gráficos e salvar o resultado

import matplotlib.pyplot as plt
import pickle
import os

def plot_hist(hist):
    plt.plot(hist["accuracy"])
    plt.plot(hist["val_accuracy"])
    plt.title("model accuracy")
    plt.ylabel("accuracy")
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()

def plot_loss(hist):
    plt.plot(hist["loss"])
    plt.plot(hist["val_loss"])
    plt.title("model Loss")
    plt.ylabel("accuracy")
```

```
    plt.xlabel("epoch")
    plt.legend(["train", "validation"], loc="upper left")
    plt.show()

model.save(os.path.join(path_dataset, 'Model_ResNet101v2_da.h5'))
pickle.dump(hist.history, open(os.path.join(path_dataset, "history_ResNet101v2_da.pkl"),
"wb"))
plot_hist(hist.history)
plot_loss(hist.history)

# Utilizar o resultado para testes singulares

from PIL import Image
import numpy as np
from tensorflow.keras.models import load_model
import os
import matplotlib.pyplot as plt

model = load_model(os.path.join(path_dataset, 'Model_VGG19.h5'))

#path_image = '/content/gdrive/My Drive/TCC/arvore.jpg'
path_image = '/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/teste.png'
#path_image = '/content/gdrive/My Drive/TCC_Paulo/dataset/Visi-
ble_Light_png/TV/TV_VL_0001.png'
#path_image = '/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/Visi-
ble_Light_png/Sem_Linha/TY_VL_2004.png'
img = Image.open(path_image).resize((IMG_SIZE,IMG_SIZE))
img = np.array(img).astype(np.float)
img = np.expand_dims(img, axis=0)

probs = model.predict(img)
c = np.argmax(probs)
```

```
print(probs)
print(c)
print(class_names[c])

plt.imshow(img[0].astype("uint8"))
plt.title(class_names[c])
plt.axis("off")

# Commented out IPython magic to ensure Python compatibility.
##########################################
# Utilizar o resultado para extrair dados
#
# Mudar o modelo e a pasta para salvar os erros (duas próximas variáveis)
#
# As imagens com erros serão salvas em '/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/'
#
##########################################

model_name = 'Model_VGG19_da.h5'
folder_erros = 'Model_VGG19_erros_da'

from sklearn.metrics import recall_score, precision_score, f1_score, confusion_matrix
from PIL import Image
import numpy as np
from tensorflow.keras.models import load_model
import os
import matplotlib.pyplot as plt
import time

print('Carregando modelo...')
model = load_model(os.path.join(path_dataset, model_name))
classes = ['Com_Linha', 'Sem_Linha']
```

```
y_true = []
y_pred = []
errors = []
tempos = []

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    path_dataset,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=(IMG_SIZE, IMG_SIZE),
    batch_size=1
)

if not os.path.isdir(os.path.join('/content/gdrive/My Drive/TCC/TCC_Paulo/da-
taset/',folder_erros)):
  os.makedirs(os.path.join('/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/',folder_erros))

imgs_errors = []
imgs_errors_classes = []
for i, infos in enumerate(val_ds):

  img = infos[0]
  labels = infos[1]
  start_time = time.time()
  probs = model.predict(img)
  elapsed_time = time.time() - start_time
  tempos.append(elapsed_time)

  cl_gt = labels.numpy()[0]
  cl = np.argmax(probs)
```

```
  y_pred.append(cl)
  y_true.append(cl_gt)
  if cl != cl_gt:
    imgs_errors.append(img[0])
    imgs_errors_classes.append([cl, cl_gt])
  print('(%d de %d) - pred: %d, gt: %d' % (i, len(val_ds), cl, cl_gt))



for i, img in enumerate(imgs_errors):
  filename = os.path.join('/content/gdrive/My Drive/TCC/TCC_Paulo/dataset/', folder_erros,
'erro_%d_gt_%s_pred_%s.png'
#                         % (i, classes[imgs_errors_classes[i][1]], classes[imgs_errors_clas-
ses[i][0]]))
  Image.fromarray(img.numpy().astype(np.uint8)).save(filename)

print('Revocação: %f:' % recall_score(y_true, y_pred))
print('Precisão: %f:' % precision_score(y_true, y_pred))
print('F1: %f:' % f1_score(y_true, y_pred))
print(confusion_matrix(y_true, y_pred))
print('Média tempo: %f:' % np.mean(tempos))
print('DP tempo: %f:' % np.std(tempos))

#######################
#
# Visualizando o que foi aprendido em uma determinada camada e em 8 filtros quaisquer
dessa camada
#
#########################

import matplotlib.pyplot as plt
```

```
model_name = 'Model_VGG19.h5'
model = load_model(os.path.join(path_dataset, model_name))
model.summary()

layer = 9

inputs = model.inputs
outputs = model.layers[layer].output

model_activation = tf.keras.Model(inputs=inputs, outputs=outputs)

path_image = os.path.join(path_dataset, 'Com_Linha/TV_VL_0011.png')
img = Image.open(path_image).resize((IMG_SIZE,IMG_SIZE))
img = np.array(img).astype(np.float)
img = np.expand_dims(img, axis=0)

activation_map = model_activation.predict(img)

print(activation_map.shape)
plt.figure(figsize=(10, 10))

ax = plt.subplot(3, 3, 1)
plt.imshow(img[0].astype(int))
plt.axis("off")

for f in range(8):
    ax = plt.subplot(3, 3, f + 2)
    plt.imshow(activation_map[0,:,:,f*10])
    plt.axis("off")
```