UNIVERSIDADE ESTADUAL DE PONTA GROSSA
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

ROVILSON ENDRIGO MORAES

AVALIAÇÃO DO *FEW SHOT LEARNING* PARA CLASSIFICAÇÃO DE IMAGENS DE PRODUTIVIDADE DA SOJA OBTIDAS POR AERONAVE REMOTAMENTE PILOTADA

PONTA GROSSA

2024

ROVILSON ENDRIGO MORAES

AVALIAÇÃO DO *FEW SHOT LEARNING* PARA CLASSIFICAÇÃO DE IMAGENS DE PRODUTIVIDADE DA SOJA OBTIDAS POR AERONAVE REMOTAMENTE PILOTADA

Dissertação apresentada ao Programa de Pós-Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Orientação: Prof.ª Dr.ª Alaine Margarete Guimarães
Coorientação: Prof. Dr. Eduardo Fávero Caires

PONTA GROSSA

2024

Moraes, Rovilson Endrigo
M827 Avaliação do few shot learning para classificação de imagens de produtividade da soja obtidas por aeronave remotamente pilotada / Rovilson Endrigo Moraes. Ponta Grossa, 2024.
124 f.

Dissertação (Mestrado em Computação Aplicada - Área de Concentração: Computação para Tecnologias em Agricultura), Universidade Estadual de Ponta Grossa.

Orientadora: Profa. Dra. Alaine Margarete Guimarães.
Coorientador: Prof. Dr. Eduardo Fávero Caires.

1. Imagens - classificação. 2. Few shot learning. 3. Deep learning. I. Guimarães, Alaine Margarete. II. Caires, Eduardo Fávero. III. Universidade Estadual de Ponta Grossa. Computação para Tecnologias em Agricultura. IV.T.

CDD: 004

Ficha catalográfica elaborada por Maria Luzia Fernandes Bertholino dos Santos- CRB9/986

UNIVERSIDADE ESTADUAL DE PONTA GROSSA
Av. General Carlos Cavalcanti, 4748 - Bairro Uvaranas - CEP 84030-900 - Ponta Grossa - PR - https://uepg.br

TERMO DE APROVAÇÃO

Rovilson Endrigo Moraes

AVALIAÇÃO DO FEW SHOT LEARNING PARA CLASSIFICAÇÃO DE IMAGENS DE PRODUTIVIDADE DA SOJA OBTIDAS POR AERONAVE REMOTAMENTE PILOTADA

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Programa de Pós- Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, pela seguinte banca examinadora:

Profª. Drª. Alaine Margarete Guimarães (UEPG - Presidente)

Prof. Dr. José Carlos Ferreira da Rocha (UEPG)

Profª. Drª. Mauren Louise Sguário Coelho de Andrde(UTFPR)

Ponta Grossa, 01 de abril de 2024.

Documento assinado eletronicamente por **Jose Carlos Ferreira da Rocha**, **Coordenador(a) do Programa de Pós-Graduação em Computação Aplicada - Mestrado**, em 21/05/2024, às 13:56, conforme Resolução UEPG CA 114/2018 e art. 1º, III, "b", da Lei 11.419/2006.

Documento assinado eletronicamente por **Alaine Margarete Guimaraes**, **Professor(a)**, em 22/05/2024, às 11:45, conforme Resolução UEPG CA 114/2018 e art. 1º, III, "b", da Lei 11.419/2006.

Documento assinado eletronicamente por **MAUREN LOUISE SGUARIO COELHO DE ANDRADE**, **Usuário Externo**, em 22/05/2024, às 14:49, conforme Resolução UEPG CA 114/2018 e art. 1º, III, "b", da Lei 11.419/2006.

A autenticidade do documento pode ser conferida no site https://sei.uepg.br/autenticidade informando o código verificador **1931748** e o código CRC **4ED784BA**.

RESUMO

Dada a crescente importância da soja na economia agrícola global e a necessidade de aprimorar as práticas agrícolas para manter o equilíbrio entre segurança alimentar e o meio ambiente, este estudo buscou avaliar a efetividade da utilização de um subconceito de *Deep Learning (DL)* o método de *Few Shot Learning* (*FSL*), aplicado na ocasião para a classificação de imagens de produtividade do cultivo da soja, adquiridas por meio de aeronave remotamente pilotada. Foram utilizadas imagens RGB em duas diferentes resoluções, 10 cm/px e 26 cm/px, obtidas no mesmo dia. Após o pré-processamento das imagens, incluindo o balanceamento de classes, foi obtida uma base de dados contendo 9.721 imagens distribuídas em quatro classes de produtividade da soja: baixa, média, alta e muito alta. Foram explorados os algoritmos: Rede Neural Convolucional (CNN) genérica modificada, *resNet50* e *denseNet121* aplicando o conceito de *Meta-Learning* baseado em inicialização junto com *FSL*, esse conceito foi relacionado com a técnica de *FSL* e ambos contribuíram para melhorar a métrica estatística de acurácia média na classificação das imagens. Foi explorado também o conceito baseado em métricas do *Meta-Learning* através dos algoritmos rede siamesa e rede siamesa tripla. Com os resultados obtidos foi notável o aumento da acurácia, especialmente quando os modelos foram treinados utilizando o algoritmo *Reptile* junto a técnica de *FSL* sendo em um conjunto de imagens similares. Na resolução de imagens de 26 cm/*px* foram obtidos os melhores resultados, essa resolução quando utilizada no modelo *DenseNet121* otimizado com *FSL* e *Reptile* atingiu acurácia de 81,3%, mostrando eficácia quando comparado com a mesma arquitetura da *DenseNet121* padrão sem o uso de tais técnicas. Os modelos redes siamesas e rede siamesa tripla também apresentaram bom desempenho, destacando a importância da aprendizagem métrica no campo de *Meta-Learning.* Os resultados indicam um caminho promissor para estes métodos contribuindo para a construção de instrumentos futuros de inteligência artificial que possam ajudar a aprimorar as práticas agrícolas relacionadas a mensurar a produtividade através de imagens.

Palavras-Chave: Classificação de imagens. *Few Shot Learning. Deep Learning*

ABSTRACT

Given the growing importance of soy in the global agricultural economy and the need to improve agricultural practices to maintain the balance between food security and the environment, this study sought to evaluate the effectiveness of using a sub-concept of Deep Learning (DL) the method of Few Shot Learning (FSL), applied at the time to classify productivity images of soybean cultivation, acquired using a remotely piloted aircraft. RGB images were used in two different resolutions, 10 cm/px and 26 cm/px, obtained on the same day. After pre-processing the images, including class balancing, a database was obtained containing 9,721 images distributed into four soybean productivity classes: low, medium, high and very high. The following algorithms were explored: modified generic Convolutional Neural Network (CNN), resNet50 and denseNet121 applying the concept of Meta-Learning based on initialization together with FSL, this concept was related to the FSL technique and both contributed to improving the statistical accuracy metric average in image classification. The concept based on Meta-Learning metrics was also explored through the Siamese network and triple Siamese network algorithms. With the results obtained, the increase in accuracy was notable, especially when the models were trained using the Reptile algorithm together with the FSL technique on a set of similar images. The best results were obtained at an image resolution of 26 cm/px. This resolution, when used in the DenseNet121 model optimized with FSL and Reptile, achieved an accuracy of 81.3%, showing effectiveness when compared with the same architecture of the standard DenseNet121 without the use of such techniques. The Siamese network and triple Siamese network models also performed well, highlighting the importance of metric learning in the field of Meta-Learning. The results indicate a promising path for these methods, contributing to the construction of future artificial intelligence instruments that can help improve agricultural practices related to measuring productivity through images.

Keywords: Image classification. Few Shot Learning. Deep Learning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AGI	Inteligência Artificial Geral
AP	Agricultura de Precisão
CNN	Rede Neural Convolucional
DenseNet	Redes Neurais Densas
DL	*Deep Learning*
ELU	*Exponential Linear Unit*
EXIF	*Exchangeable Image File*
FC	*Fully-Connected*
FN	Falsos Negativos
FP	Falsos Positivos
FSL	*Few-Shot Learning*
GCP	Ponto de Controle no Terreno (do inglês *Ground Control Point*)
GPL	*General Public License*
GPS	Sistema de Posicionamento Global (do inglês *Navigation Satellite Systems*)
GNSS	Sistemas Globais de Navegação por Satélite (do inglês G*lobal Navigation Satellite Systems*)
IA	Inteligência Artificial
IVs	Índices de Vegetação
LCAD	(Laboratório de Computação de Alto Desempenho)
MAML	*Model Agnostic Learning*
MLP	Multicamada (*perception) (Feedforward*)
ReLu	*Rectified Linear Unit*
ResNet	Redes Residuais
RGB	Vermelho, Verde e Azul
RNA	Redes Neurais Artificiais
RPA	Aeronave pilotada remotamente (do inglês *Remotely Piloted Aircraft*)
RPAS	Sistema aéreo pilotado remotamente (do inglês *Remotely Piloted Aircraft System*)
SGD	*Stochastic Gradient Descendent*
SIG	Sistema de Informação Geográfica
UAVs	*Unmanned Aerial Vehicles*
UEPG	Universidade Estadual de Ponta Grossa
VANTS	*Unmanned Aerial Vehicles*
VGG	*Visual Geometry Group*

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos o cultivo da cultura da soja teve grande participação na balança comercial brasileira. A receita proveniente das exportações da soja em 2020 superou os US$ 100 bilhões. O Brasil ocupa atualmente a posição de maior produtor de soja mundial (Silva *et al.*, 2022).

No mesmo período em que o Brasil consolida-se como maior produtor de soja do mundo a inovação industrial permitiu que o ano de 2021 fosse oficialmente marcado pela comissão europeia como o início da era da “Indústria 5.0” no mundo, incluindo o setor agrícola, esse ano marca a aproximação dos sistemas digitais e automatizados com a terceira e quarta revolução industrial que trouxeram: máquinas, sensores, aeronaves remotamente pilotadas (*RPA*, do inglês *Remotely Piloted Aircraft*), sistemas de telecomunicações, manipulação genética, entre outras tecnologias para a agricultura (Martos *et al.*, 2021).

A população mundial deve crescer em torno de 2 bilhões nos próximos 30 anos, a previsão de produtividade precisa e não destrutiva do rendimento das colheitas em grandes áreas a baixo custo tem importância econômica, essa atividade necessita ser minimamente sustentável (Luz *et al.*, 2020).

Dessa forma, fica evidente a necessidade de dispor de alternativas eficazes para a agricultura seja realizada de maneira racional, buscando atender o máximo de produtividade.

Para contribuir com tais aspectos, o processamento digital de imagens e técnicas de Inteligência Artificial (IA) pode ser utilizado em conjunto para realizar procedimentos nas imagens capturadas com objetivos de gerar resultados satisfatórios para a visualização e posteriormente tomadas de decisões, ou serem utilizadas em automações de processos.

A captura de imagens utilizando *RPAs* proporciona aquisição de imagens com alta resolução. O uso de *RPAs* para classificação de imagens de produtividades envolve um conjunto de tecnologias de processamento e análise de imagem, gerenciamento de dados geográficos e técnicas de IA para classificação são necessárias, essas tecnologias ainda precisam ser adaptadas e combinadas para permitir mais precisão nas imagens classificadas (Viniski, 2019).

Deep learning (*DL*) é um subcampo da IA que tem se destacado por oferecer métodos que podem ser eficientes para classificar imagens entre outros tipos de

dados. Tais métodos podem incluir diversos algoritmos, componentes e técnicas distintas a fim de compor um modelo de IA.

Ainda existem algumas áreas incertas em relação à eficácia desses métodos, a compreensão das situações em que funcionam melhor, bem como suas limitações ainda é uma lacuna a ser preenchida.

As limitações atualmente em *DL* dizem respeito às grandes quantidades de dados rotulados necessários para aprender. Enquanto que, para os seres humanos aprenderem e interpretarem um conceito simples basta poucos exemplos.

A Aprendizagem de Poucos Tiros *FSL* (do inglês *Few-Shot Learning*) permite que a aprendizagem de um modelo com um pequeno número de exemplos possa ser feita (Wang *et al.*, 2020), outro conceito chamado *Meta-Learning*, pode ser utilizado em conjunto com o *FSL* para desenvolver modelos de aprendizado que posam se adaptar as tarefas de classificação, dessa forma aprendendo utilizando com poucos exemplos de treinamento.

Situações peculiares para a adoção de *FSL* são aquelas onde é difícil ou impossível de se obter o número de exemplos suficientes, seja devido a questões de privacidade, segurança, ética, custo ou ainda por dificuldades de acesso (Wang *et al.*, 2020).

O estudo de modelos de IA na agricultura é de suma importância para se obter condições de melhorar a produção agrícola em relação à crescente demanda do mercado. A capacidade de prever a produtividade antes do período de colheita é utilizada para construir estratégias como: determinação de épocas de colheita, planejamento de recursos financeiros, equipamentos, capacidade de armazenagem e logística. (Santos; Santos; Rolim, 2021).

Considerando a importância do cenário agrícola em que o mesmo necessita evoluir constantemente assim como, pesquisas correlatas no quadro 2 indicando que não existem estudos para classificação de imagens de produtividade especificamente para o cultivo da cultura da soja, este estudo propõe a utilização de imagens obtidas por *RPA* referentes à produtividade de soja e processadas com sistemas de georreferenciamento para criar um conjunto de imagens a serem submetidas aos modelos de classificação utilizando IA.

Aplicar uma combinação de técnicas de *DL*, *Meta-Learning* e o método *FSL* para comporem modelos de IA voltados para a classificação de imagens em um conjunto de imagens agrícolas de produtividade do cultivo de soja, deve contribuir

para avaliar o quanto esses métodos são importantes para gerar modelos de IA precisos.

Dessa forma o objetivo deste estudo é explorar tais conceitos especialmente o uso do *FSL* em duas diferentes resoluções de imagens coletadas no mesmo dia por meio de um RPA.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Avaliar a eficácia de modelos que contemplam o método de *FSL* para classificar imagens de produtividade da cultura da soja.

2.2 OBJETIVOS ESPECÍFICOS

Avaliar imagens *RGB* (Vermelho, Verde, Azul, do inglês Red, Green, Blue) obtidas por *RPA* para classificação de produtividade no cultivo da soja considerando as resoluções 10 e 26 cm/*px*.

Mensurar resultados de modelos de *DL* para classificação das imagens com um algoritmo de *Meta-Learning* e *FSL* incorporado.

Utilizar técnicas de otimização para melhorar a eficácia dos modelos propostos.

Analisar a eficácia de todas as arquiteturas propostas utilizando a medida estatística acurácia média.

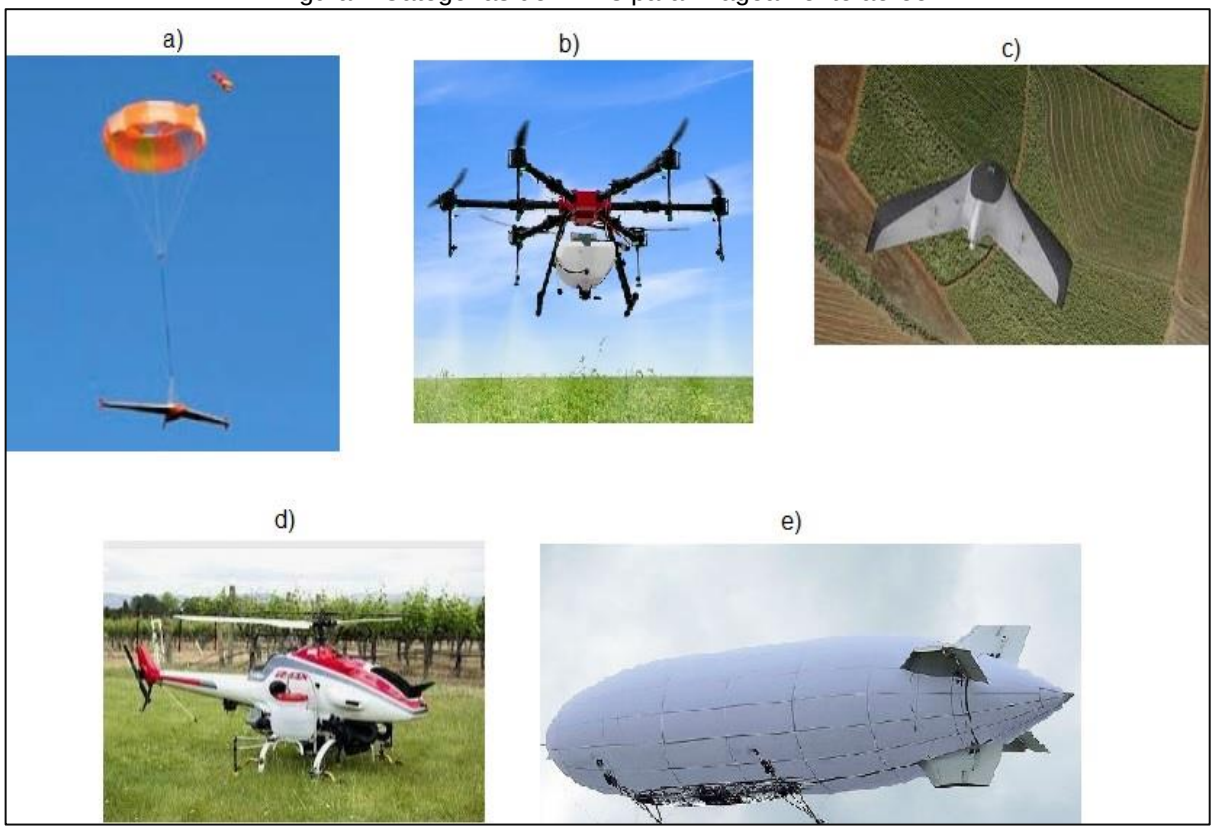
3 REVISÃO DE LITERATURA

3.1 AERONAVES NA AGRICULTURA

Drones e veículos aéreos não tripulados (*VANTs* ou *UAVs*, do inglês *Unmanned Aerial Vehicles*) são formas usuais para se denominar *RPAs*. O conceito "*RPA*" foi introduzido em 2015, juntamente com a definição de sistemas aéreos pilotados remotamente (*RPAS* - Sistema Aéreo Pilotado Remotamente). O *RPAS* refere-se ao conjunto representado pela aeronave e outros equipamentos necessários para sua operação, como equipamentos de lançamento e recuperação, enlaces de comando e controle e estações de pilotagem remota (ANAC, 2015).

Usualmente dividem-se as *RPAs* em cinco diferentes categorias (Sankaran et al.,2015): paraquedas (Figura 1a), multi-rotores (Figura 1b), asa fixa (Figura 1c), helicópteros (Figura 1d) e dirigível (Figura 1e).

Figura 1.Categorias de RPAs para imageamento aéreo.



Fonte: O Autor (2023).

O interesse no uso de imagens aéreas aumentou significativamente nos últimos anos com foco em *RPA* que demonstraram maior flexibilidade para aquisição de imagens de alta resolução espacial (alguns centímetros) e temporal (diariamente). No entanto as *RPAs* denotam algumas barreiras operacionais, como

por exemplo o tamanho da área a ser coberta, afinal existe uma certa limitação no tempo de uso da bateria, as condições climáticas também interferem no uso pois ventos fortes, chuvas excessivas ou temperaturas muito altas ou extremamente baixas alteram ou impedem o uso de *RPAs .* (Rebelo; Nascimento, 2021).

Contudo, logram êxito ao serem usados para monitoramento de safras, gestão e logística da produção, estimativa de produção, contagem de plantas, detecção de falhas de plantio, estresse hídrico, nutrição e detecção de doenças e pragas, assim sendo a *RPA* se tornou uma opção importante para a agricultura de precisão (Chaves *et al.*, 2015). Estudos sobre uso de *RPAs* para obtenção de imagens associadas à produtividade de culturas podem ser vistos no Quadro 1.

Quadro 1. RPAs na agricultura associados a produtividade de culturas.

Cultura	**Tipo de Aeronave**	**Técnica**	**Autor**
Soja	Multi-rotor	Fusão de Dados Multimodal	MAIMAITIJIANG, M. et al. 2020
Trigo	Asa fixa/ multi-rotor	Técnicas de Mineração	PRESTES, C. D. P., 2020
Cana-de-açúcar	Multi-rotor	Regressão Linear	MARTELLO, M. , 2017
Trigo	Asa fixa/ multi-rotor	Regressão Linear	STACHAK, A., 2018
Trigo/Soja	Asa fixa	Técnicas de Regressão	VINISKI, A. D., 2018
Trigo	Asa fixa	Técnicas de Regressão	GERKE, T. , 2017
Soja	Asa fixa	Regressão Linear	OLIVEIRA NETO, d. , 2020
Soja	Multi-rotor	Análise de Variância	FACCO, F. L. B., 2022 NANDES, P
Milho	Asa Fixa	Regressão Linear	FERNANDES, P., 2016
Café	Multi-rotor	Sensoriamento remoto e visão computacional	SANTANA, L. S., et al 2023

Fonte: O Autor (2023).

Com o levantamento feito no quadro 1, nota-se que foram identificados poucos trabalhos envolvendo o uso de RPAs para temas correlacionados a

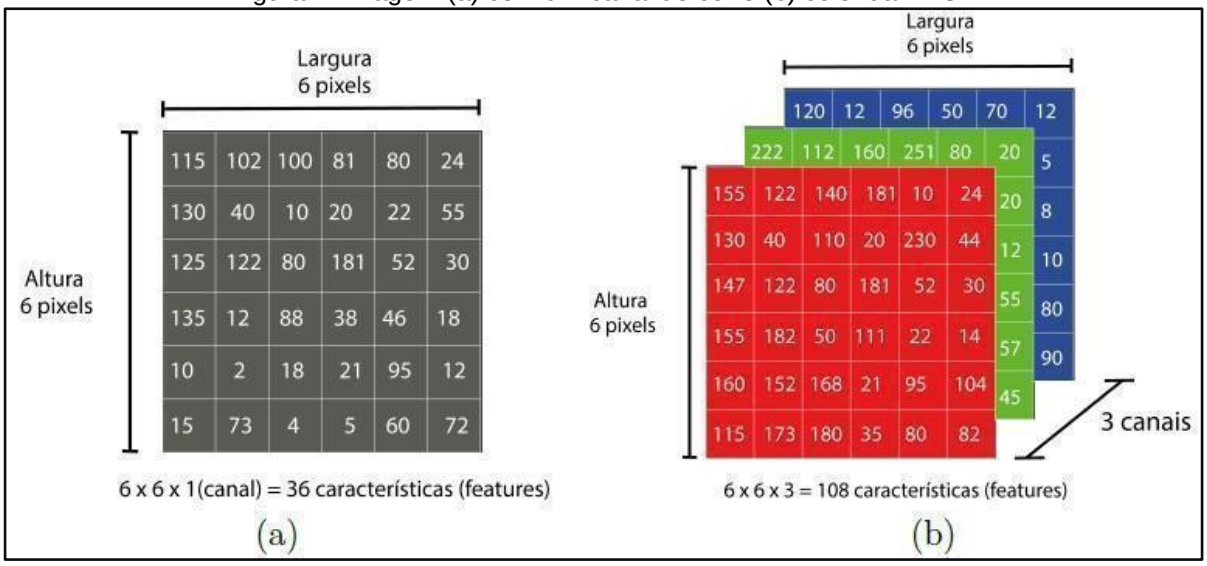
produtividade no cultivo da soja, observa-se também que os trabalhos não fazem uso de nenhuma técnica de IA. Por tanto não se sabe de forma concreta quais são os detalhes técnicos das imagens que fornecerão maior precisão quando utilizadas em modelos de IA. Com isso, pesquisas devem ser realizadas visando o conhecimento de quais imagens podem proporcionar mais precisão.

3.2 IMAGENS DIGITAIS

Toda imagem digital é constituída por uma unidade chamada de *pixel*, a qual pode ser definida como a menor unidade de uma imagem digital, quando colorida é constituída por três matrizes de cores, sendo elas respectivamente: vermelho, verde e azul (*RGB*). Cada canal, pode variar de 0 a 255 *pixel* na imagem, o que lhe confere intensidade de cor, onde 0 é intensidade de cor desligada e 255 é a intensidade mais forte do canal (Silva, 2001, p.5 *apud* Gonzales; Woods, 2009).

A Figura 2 (a) representa uma imagem com escala de cinza matizada por um canal de cor, e (b) uma imagem colorida mesclada pelos três canais de cores (*RGB*).

Figura 2. Imagem (a) com um canal de cor e (b) colorida - *RGB*



Fonte: SILVA (2021)

A Figura 2 (a) demonstra uma imagem em escala cinza composta por 6x6 *pixels* de largura e de altura, totalizando 36 tipos de características ou *features* (dados que representam medidas obtidas da imagem considerando textura, objetos, porte, tamanho, talhe, etc.). A Figura 2 (b), é a representação de uma imagem 3D colorida constituída pelos três canais de cores *RGB*, onde cada canal possui valores

entre 0 a 255 para cada *pixel. P*or conter mais canais que a imagem em escala cinza, passa a ter três vezes mais *features* do que a imagem de escala única.

A Figura 3 representa uma imagem *RGB* em forma de cubo 3D, que permite perceber com clareza as posições das cores e suas respectivas nuances em níveis em cinza e *RGB* usando os três atributos da cor (matiz, saturação e intensidade/brilho). A mudança no ponto exibido no cubo altera seu local em relação ao *pixel*, alterando assim a sua cor. De forma exemplificada: a variação de matiz faz com que a cor dominante observada seja alterada, quando o eixo alterado é o da saturação significa que há uma maior ou menor “mistura” de luz branca, alterando o que é chamado de pureza da cor, no entanto quando o eixo alterado é o de intensidade/brilho o conceito usado é o de luz, tornando a imagem mais clara ou escura, ou seja aproximando-a da cor branca ou preta.

Figura 3. Representação do modelo de cor RGB em forma de cubo.

Fonte: ANTUNES (2021)

3.3 REDES NEURAIS ARTIFICIAIS

O nome Redes Neurais Artificiais (RNA) As RNAs podem ser ilustradas como um grupo de neurônios artificiais que estão interligados através de muitos vínculos (por vetores e matrizes de pesos sinápticos geralmente unidirecionais). Os pesos armazenam as representações do conhecimento adquirido e são usados para calcular as entradas dos próximos neurônios da rede. (Silva; Sapatti; Flauzino, 2016).

Existem várias áreas com de aplicabilidade para as redes neurais; basicamente todo processo que faz uso de algum tipo de dado tem potencial de aplicação, seja através da utilização de imagem, áudio ou texto.

Algumas características marcantes das RNAs são:

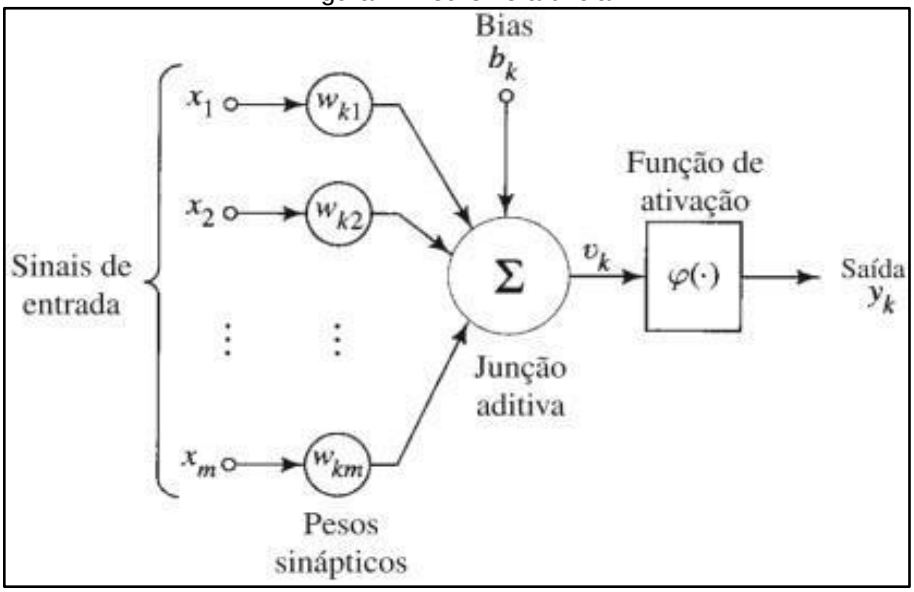
- Utiliza constantes repetições de exemplos que refletem o comportamento de um processo, tornando possível a aprendizagem por meio de experimentações, adaptando os seus pesos ao longo do processo.
- Uma vez treinada, a rede neural tem a capacidade de expandir o conhecimento adquirido, permitindo fazer previsões para situações anteriormente desconhecidas e também conhecidas, esse processo é conhecido como generalização.
- Organizar dados com base em características semelhantes para fornecer um agrupamento de padrões para a sua organização interna a respeito de um determinado processo.
- A informação sobre o comportamento em um processo específico é distribuída entre os seus neurônios artificiais, permitindo assim um aumento na confiabilidade da arquitetura frente a eventuais neurônios falhos.
- Possibilidade de serem embarcadas em *software* ou em *hardware* conforme os objetivos do projeto, considerando que após o procedimento de treinamento, seus resultados são obtidos através de algumas operações matemáticas. (Silva; Sapatti; Flauzino, 2016).

Este tipo de rede é uma aproximação muito grosseira da complexidade de uma verdadeira rede neural biológica. Quanto a analogia entre neurônios biológicos e artificiais, alguns pesquisadores recomendam descartar completamente para não restringir a criatividade e evolução das IAs.

3.3.1 Estrutura De Um Neurônio Artificial

A estrutura de um neurônio artificial (Figura 4) inspirado nos neurônios biológicos contém cinco principais elementos. Os sinais de entrada são os dados recebidos para processamento. Os pesos sinápticos determinam a importância de cada sinal de entrada. O *bias* é uma constante que ajuda a ajustar a saída do neurônio. A junção aditiva é onde todos os sinais de entrada, ponderados pelos pesos sinápticos, são somados com o *bias*. A função de ativação é uma função não-linear que transforma o resultado da junção aditiva. A saída do neurônio é a aplicação da função de ativação ao resultado da junção aditiva. (Haykin, 2007).

Figura 4. Neurônio artificial



Fonte: HAYKIN(2007).

A função de ativação é quem determina se o próximo neurônio deve ser ativado ou não, com base na soma ponderada das entradas.

As funções de ativação tem papel fundamental no treinamento de redes neurais, permitindo-lhes aprender representações não lineares. Em *DL* essas funções necessitam ser não lineares e capazes de ser diferenciadas continuamente além de se aproximar da função de identidade perto do zero (Aghdam; Heravi, 2017).

Identidade perto do zero significa que para os valores de entrada a função de ativação deve retornar um valor mais próximo do próprio argumento de entrada, resultando em um gradiente que facilita o processo de treinamento da rede neural. Ainda que existam métodos de treinamento que não dependam de gradiente, os métodos que se baseiam em gradientes são os mais utilizados. (Aghdam; Heravi, 2017).

Normalmente, os pesos e o *bias* (b) são inicializados com valores próximos de zero pelo método gradiente descendente. Em termos de descida do gradiente, é um gradiente forte que ajuda a convergência mais rapidamente do treinamento (Aghdam; Heravi, 2017).

3.3.2 Arquitetura Geral De Uma RNA

As conexões na camada de uma RNA podem constituir infinitas estruturas diferentes. Quando se tem todas as saídas dos neurônios de certa camada

conectadas com todos os neurônios da camada subsequente, define-se essa rede como plenamente conectada (*fully connected*) (Ludwing Jr; Montgomery, 2007).

O desenho e a estrutura de uma RNA não são fixos e podem ser ajustados conforme as necessidades do projeto. No entanto, as diretrizes fornecidas pelas ferramentas e teorias existentes podem ajudar a ajustar a quantidade de parâmetros maleáveis na rede em relação ao número de exemplos de treinamento utilizados. (Ludwing Jr; Montgomery, 2007).

Durante o processo de treinamento (ou aprendizado) o *feedback* permite que os algoritmos aprendam e se ajustem. O objetivo do treinamento é sempre encontrar um ajuste ótimo com seus pesos sinápticos associados para alguma instância de problema.

O *feedback* funciona em forma de uma função de erro ou função de perda, que a Rede Neural Artificial (RNA) usa para ajustar seus pesos, melhorando assim o desempenho na tarefa, principalmente através de um processo chamado *backpropagation* (retropropagação). (Géron, 2019)

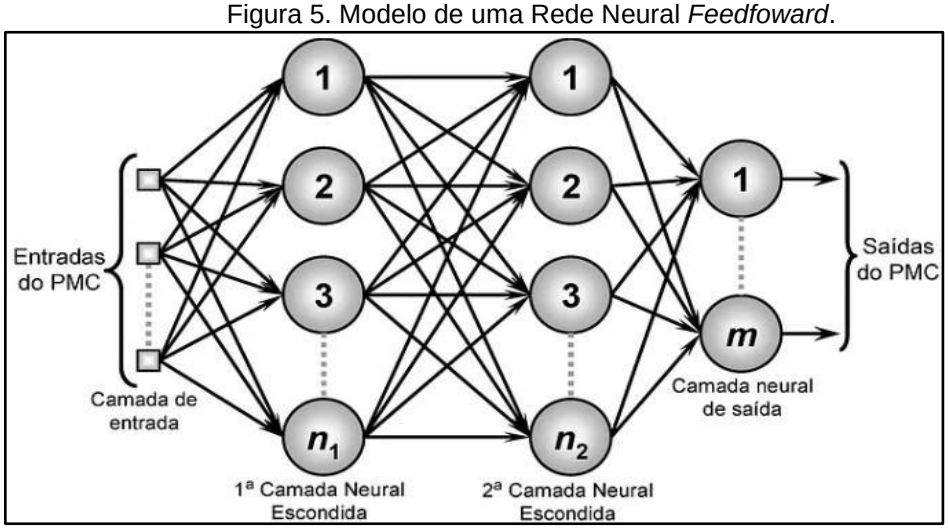
Quando uma RNA faz uma previsão, a saída é comparada com o valor esperado. A diferença entre essas duas quantidades é calculada usando uma função de perda ou erro, produzindo um valor de erro. Esse erro é então alimentado de volta na rede de saída para entrada, através do processo de *backpropagation*, entretanto nem toda RNA tem esse processo em sua estrutura.

Durante o *backpropagation* o erro é calculado em relação a cada peso na rede e, por final, os pesos são atualizados (Géron, 2019).

Essa sequência de passos é repetida muitas vezes em "épocas" ou passagens através do conjunto de treinamento para que gradualmente melhore a precisão da rede.

A Figura 5 representa a estrutura *feedforward* multicamada que apresenta um fluxo de informação unidirecional, da camada até a camada de saída, sem *backpropagation*. Estas redes consistem em uma camada de entrada, uma ou mais camadas ocultas e uma camada de saída, todas interconectadas. (Silva; Sapatti; Flauzino, 2016)

Figura 5. Modelo de uma Rede Neural *Feedfoward*.

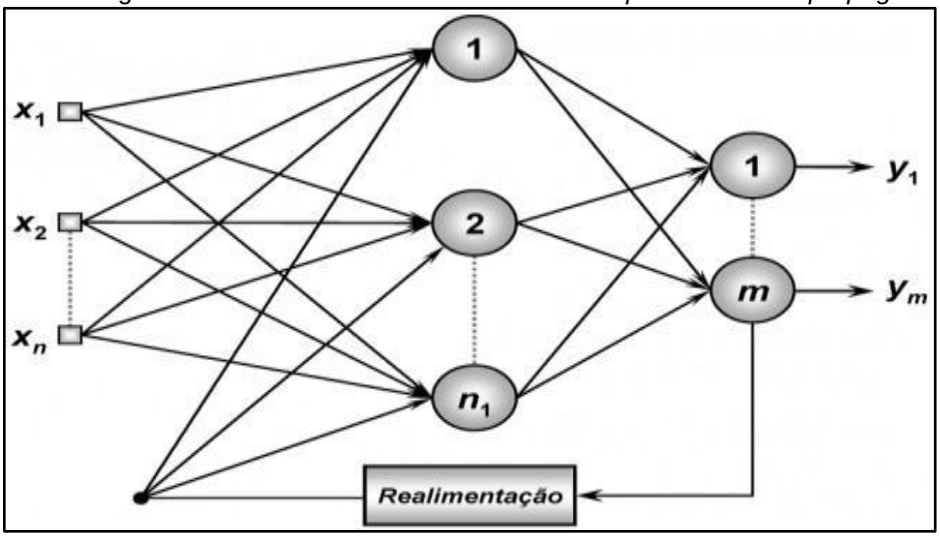


Fonte: (SILVA, SAPATTI, FLAUZINO, 2016).

Um conceito inicial fundamental é o modelo *Perceptron*, criado por Rosenbaltt em 1958, inspirado ainda por McCulloch e Ptts de 1943, mesmo com o passar do tempo o seu modelo se mantém como base fundamentalista das redes neurais (Siebert, 2022).

A rede *Perceptron* com *backpropagation* (Figura 6) é também conhecida como *Perceptron* recorrente, porque inclui conexões de *feedback*. Em vez de fluxo unidirecional de entrada para saída, a *Perceptron* com *backpropagation* permite que a saída volte para a entrada, formando um circuito de *feedback* ou *loop*. Tal modelo processa dados, levando em consideração a dependência do histórico de entradas e saídas anteriores (Silva; Sapatti; Flauzino, 2016)

Figura 6. Modelo de uma Rede Neural *Perceptron* com *backpropagation.*



Fonte: (SILVA, SAPATTI, FLAUZINO, 2016)

A arquitetura da rede neural define os limites e as capacidades da rede para aprender, adaptar-se e resolver problemas complexos.

3.3.3 Aprendizagem Em Redes Neurais Artificiais

Os pesos da rede podem ser ajustados de muitas maneiras diferentes. Um algoritmo de aprendizado é utilizado para ajustar os pesos do modelo usando um conjunto de dados (Borrero; Arias, 2021).

Com base no tipo de dados utilizados sendo de imagens de produtividade da cultura da soja obtidas por aeronave para ajustar o modelo, destaca-se aqui a utilização da aprendizagem supervisionada.

Na aprendizagem supervisionada, uma condição fundamental é a existência de um "professor" para fornecer correções para as saídas. A aprendizagem supervisionada estabeleceu-se como um conceito poderoso para projetos de redes neurais artificiais (Haykin, 2007).

O aprendizado supervisionado é uma forma bastante utilizada de treinamento, consiste em se ter um grande conjunto de dados, cada um rotulado com sua categoria. Durante o treinamento é apresentado o dado que produz uma saída na forma de um vetor de pontuações. Uma função mede o erro (ou distância) entre as pontuações de entrada e o padrão de pontuações desejado para as suas saídas correspondentes. A RN então modifica seus pesos ajustáveis para reduzir esse erro (LeCun; Bengio; Hinton, 2015).

No processo de treinamento consiste a aplicação de passos para a sintonização e ajustes dos pesos e limiares de seus neurônios, tendo como objetivo final a generalização de soluções (Silva; Sapatti; Flauzino, 2016).

Generalização é quando após a rede tiver aprendido a partir das representações de amostras, ela se torna capaz de reproduzir resultados próximos aos quais foram aprendidos usando dados nunca vistos antes, sendo esses, geralmente do mesmo gênero dos dados aprendidos.

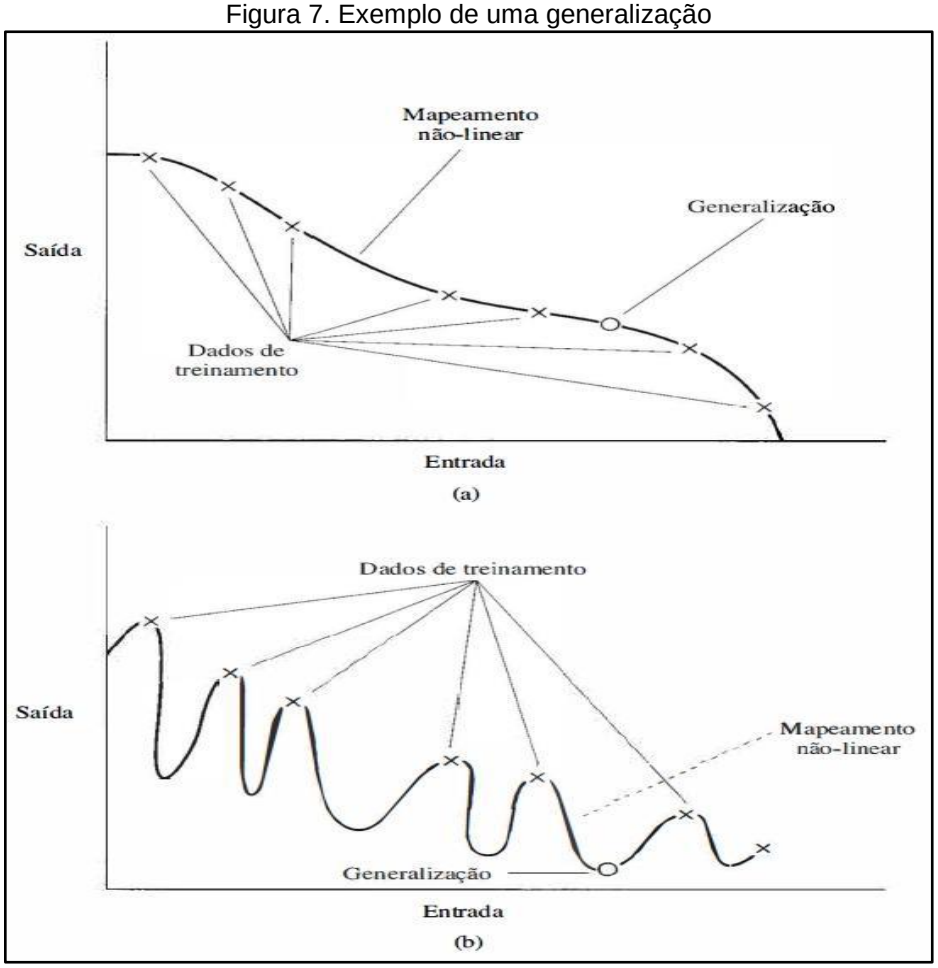
O conjunto total de amostras para o problema é dividido em dois subconjuntos de treinamento e de validação, com cerca de 70% a 90% destinados para o treinamento da rede e 10% a 30% para a validação da rede (Silva; Sapatti; Flauzino, 2016).

O processo é repetido várias vezes para o conjunto de validação até que a média da função pare de diminuir. O conjunto de validação é uma porcentagem da amostra que a rede neural nunca viu antes, são amostras sem rotulagem. A

validação (também chamada de teste) verifica se os aspectos referentes a generalização estão em níveis aceitáveis.

A Figura 7(a) mostra uma generalização que pode ocorrer hipoteticamente. O mapeamento não-linear de entrada-saída apresentado na curva é o resultado da aprendizagem dos dados rotulados (dados de treinamento),o ponto marcado sobre a curva como "generalização" é visto como o resultado da interpolação realizada pela rede. (Haykin, 2007).

Figura 7. Exemplo de uma generalização



Fonte (HAYKIN, 2007).

Na Figura 7(a) os dados ajustados adequadamente obtiveram boa generalização. Já na Figura 7(b) os dados foram ajustados em excesso resultando em uma generalização pobre.

A seleção apropriada de dados de qualidade bem como a seleção dos parâmetros e a estrutura da rede neural são muito importantes antes dela ser treinada para que solução de um determinado problema seja precisa. (Haykin, 2007).

3.3.3.1 *Overfitting*

O treinamento de um modelo pode superestimar a sua capacidade, isso é conhecido como *overfitting.* Ocorre quando o modelo aprende demais os dados de treinamento e com isso não consegue realizar previsões precisas em dados nunca vistos antes, mesmo sendo do mesmo gênero (Géron, 2019).

Esse problema pode ocorrer principalmente em *DL* por ser capaz de detectar ruídos nos dados, isso pode ser um desafio se o conjunto de treinamento for ruidoso demais ou de pequeno porte, o que pode introduzir mais ruído na amostragem. Diante dessa condição o modelo pode acabar reconhecendo padrões que são resultados do próprio ruído nos dados, sem ser capaz de diferenciar se é um padrão genuíno ou não (Géron, 2019).

3.3.4 Limitações Nas Redes Neurais Artificiais

Analisar as limitações das RNA requer uma consideração das características específicas de cada tipo de rede. Uma das desvantagens das redes neurais com *backpropagation*, por exemplo, é a falta de controle do usuário sobre o treinamento. Uma vez configurada a arquitetura da rede, o usuário só pode alimentar o sistema com as entradas e esperar pela saída. As redes neurais podem ser mais lentas para treinar e em algumas circunstâncias podem requerer milhares de épocas, o que pode ser problemático para redes muito grandes com grandes conjuntos de dados (Haykin, 2009).

Para fazer uma boa generalização, os dados de treinamento e validação devem ser da mesma classe e ter quantidade abundante de dados disponíveis, principalmente quando os dados de treinamento não são idênticos aos dados de validação (Marcus, 2018).

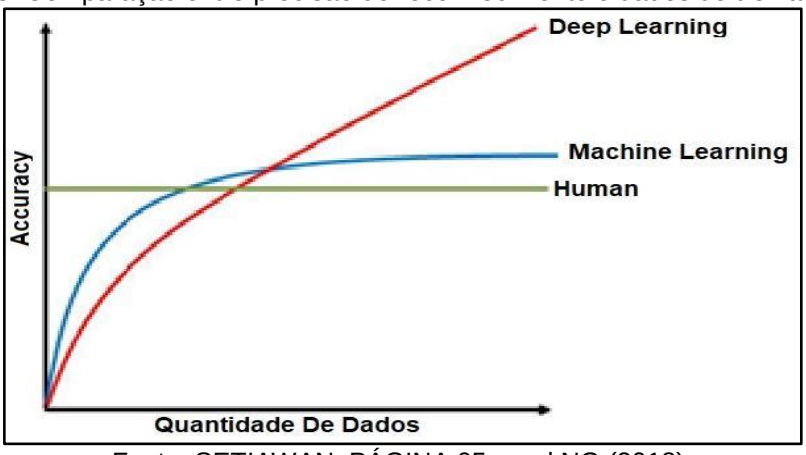
A maneira adequada de saber o quão bem um modelo generaliza em novos casos é de fato testá-lo em novos casos. O *DL* faz uso mais intensivo das RNAs, elevando a capacidade de generalização.

3.4 *DEEP LEARNING*

O *DL* basicamente é uma estrutura de RNA que também conta com cinco elementos fundamentais: funções de ativação, neurônios, pesos, conexões (ou sinapses) e polarização (ou *bias*) (Zou, 2022).

O avanço do poder computacional e a disponibilidade de dados viabilizaram as pesquisas com sucesso em diversas tarefas principalmente em relação ao *DL* em que quanto mais dados de treinamento forem utilizados, maior será a precisão obtida.

Figura 8. Comparação entre precisão de reconhecimento e dados de treinamento

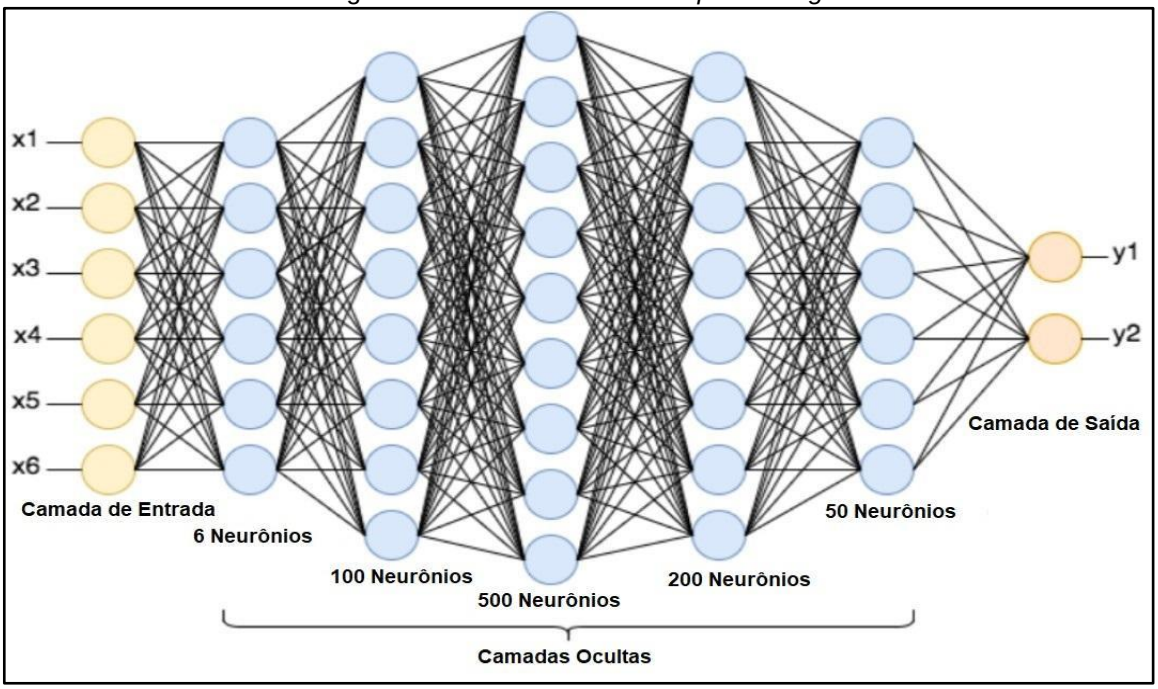


Fonte: SETIAWAN, PÁGINA 05 apud NG (2018)

A Figura 8 mostra uma comparação onde a medida em que a quantidade de dados aumenta, a precisão do reconhecimento também aumenta com a *DL* em relação a humanos e o aprendizado de máquina convencional (Setiawan, 2020 página 05 apud NG, 2018)

O grande diferencial dessa arquitetura é a presença de um número elevado de camadas como observado na Figura 9. Quanto ao processo de aprendizagem em relação a sua antecessora RNA pode haver mais hiperparâmetros para serem ajustados, porém o processo de aprendizagem em sua essência é o mesmo.

Figura 9. Estrutura de uma *Deep Learning.*



Fonte: MERIEM; BATOUCHE (2018)

O *DL* é mais eficiente para processar grandes volumes de dados tornando-o ideal para aplicações complexas, como reconhecimento de imagem e voz, condução autônoma, entre muitas outras áreas de aplicações.

Possui habilidade de extrair recursos dos dados e produzir uma representação hierárquica distribuída, este processo envolve extrair características dos dados de maneira incremental, da mais simples a mais complexa, de forma autônoma, estabelecendo a característica única de cada dado. (Borrero; Arias, 2021)

Na classificação de imagens, os algoritmos de aprendizado profundo primeiro processam a imagem de entrada em um nível primário, identificando características de baixa complexidade, tais como cores e padrões de borda. Com a imagem decomposta em seus elementos fundamentais, esses algoritmos avançam para as camadas superiores. (Borrero; Arias, 2021).

Essas características de alto nível são consolidadas para produzir a saída do modelo, compondo a imagem original, camada por camada, extraindo e aprendendo recursos ao longo do caminho. (Borrero; Arias, 2021)

A medida que os dados avançam pela rede, cada camada extrai características. Esta é a essência das técnicas de *DL*, uma representação gradativa de características complexas partindo das mais simples.

Um subtipo específico dentro de *DL* que é o conceito mais indicado para se trabalhar com imagens chama-se Rede Neural Convolucional (*CNN*), na próxima subseção esse tema é abordado em detalhes.

3.4.1 Rede Neural Convolucional (*CNN*)

As Redes Neurais Convolucionais (*CNNs*) representam uma abordagem em que a principal característica que distingue as *CNNs* de outras redes neurais é que elas usam uma operação matemática chamada convolução dentro de sua rede neural em pelo menos uma de suas camadas. Essas camadas de *CNN* possuem como foco processar e identificar de forma mais eficiente, padrões espaciais presentes em dados estruturados de grade, como imagens (grade de *pixels* 2-D) ou também em séries temporais (grade 1-D) (Goodfellow; Bengio; Courville, 2016).

O uso de camadas convolucionais favorece os resultados ao se trabalhar com imagens, tornando as *CNNs* uma ferramenta fundamental para o avanço das tecnologias em IA.

3.4.2 Arquitetura da *CNN*

Em geral, as arquiteturas das *CNNs* são sequências de camadas que funcionam juntas para extrair características úteis para que as camadas de redes neurais classifiquem essas características. O processo de otimização e alinhamento dessas camadas dentro de uma rede neural convolucional passa por uma série de experimentações até que seja alcançada a melhor combinação dos parâmetros (Borrero; Arias, 2021).

A arquitetura apresentada na Figura 10 funciona em camadas para fornecer classificação de imagens com alta precisão. Os elementos de uma CNN geral incluem (Sikka, 2021):

Camada Convolucional

Pooling

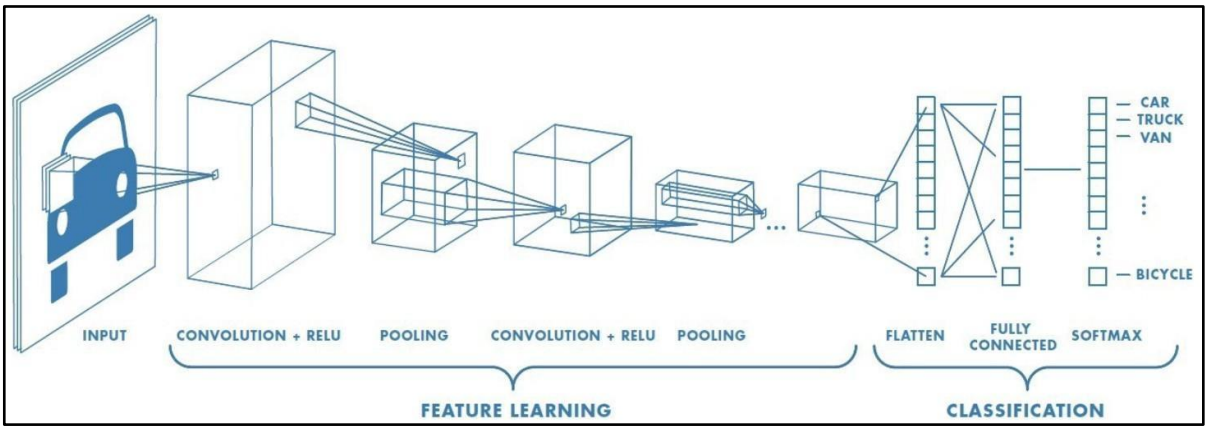
Função De Ativação

Flatten

Camada Completamente Conectada.

A sequência, quantidade, assim como a configuração dos parâmetros destes elementos podem variar de acordo com o projeto a ser implementado, conforme representado pela Figura 10.

Figura 10. Estrutura CNN



Fonte: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

3.4.3 Operação de Convolução

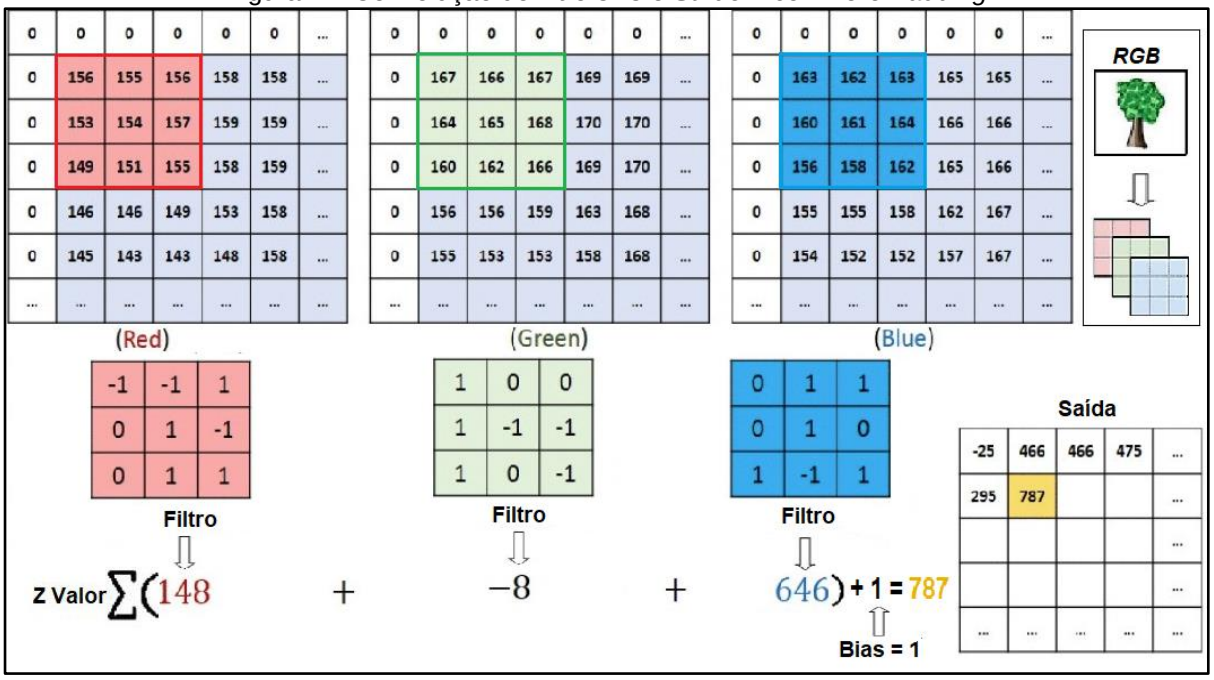
A operação de convolução serve para uma extração de características em profundidade dos dados de entrada (dos *pixels* da imagem) usando filtros de convolução para obter mapas de características. Sem este processo a rede neural praticamente iria apenas quebrar a imagem grosseiramente em *pixels* ou em conjuntos de *pixels* e não iria buscar aprender em profundidade as camadas do *RGB* existentes em uma imagem.

Para realizar as operações de convolução, os filtros (conhecidos também como *kernels*) deslizam sobre cada região da imagem para fazer uma multiplicação de matrizes elemento a elemento e somar os resultados gerais. A soma obtida irá para o mapa de características. Uma imagem é representada como uma matriz 3D, então as convoluções são realizadas em 3D com dimensões de profundidade, altura e largura onde corresponde a *RGB* (Millstein, 2018).

O mapa de características fornecerá respostas para o filtro. A rede aprende automaticamente todos os filtros que são ativados ao se deparar com características visuais, como mancha de cor ou borda. Existe uma coleção de filtros em cada camada convolucional, conforme visto na Figura 11, cada um produz mapas diferentes. Os mapas são então empilhados ao longo da dimensão para formar o volume de saída. Cada mapa de características individuais é bidimensional, mas

quando são empilhados juntos, eles formam um cubo tridimensional de dados (Millstein, 2018).

Figura 11. Convolução de filtro 3X3 e *Stride* 1 com zero *Padding*



Fonte: Adaptado de KROHN; BEYLEVELD; BASSENS (2019)

Para cada posição da imagem o processo é repetido conforme o filtro vai deslizando pela imagem o valor de Z vai sendo calculado para cada uma dessas nove posições que é mostrado no canto inferior direito da Figura 11. (Krohn; Beyleveld; Bassens, 2019)

Stride, assim chamada a ação de deslocamento do filtro (ou passo). Se *stride* = 1, o filtro em cima da imagem de entrada é deslocado um *pixel* para a direita. (Setiawan, 2020)

O *Padding* (ou preenchimento) envolve adição de camadas extras de *pixels* neutros (valores zero) ao redor da borda de uma imagem para garantir que a operação de convolução seja feita para todos os *pixels*. Evitando a perda de informações nas bordas da imagem. Permite que a primeira camada de uma *CNN* considere todos os *pixels* da imagem original, mesmo aqueles nos cantos e nas bordas. (Setiawan, 2020)

Padding pode ajudar no desempenho da rede porque o filtro consegue mapear todas as informações dos valores contidos na imagem, ele acaba fornecendo uma forma para não denegrir o tamanho de saída em relação ao tamanho de entrada da imagem, possibilitando a extração de recursos mais precisos para as próximas camadas da rede.

3.4.4 *Pooling*

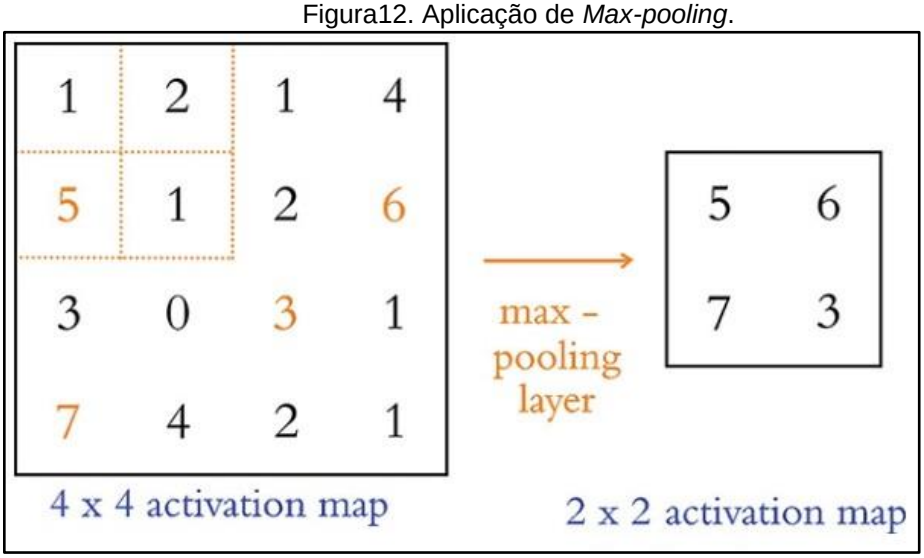
É uma prática comum combinar as camadas convolucionais com camadas de *pooling* (agrupamento). O objetivo das camadas de *pooling* é a agregação aplicada a cada mapa de características. A justificativa para a aplicação deste processo é que o *pooling* contribui para a diminuição da quantidade de dados, consequentemente, se torna mais eficiente ao utilizar recursos computacionais. (Hope; Resheff; Lieder, 2017)

A camada de *pooling* tem um papel importante em uma rede neural convolucional, pois ajuda a diminuir a dimensionalidade dos mapas de ativação, essa redução de parâmetros acelera o processo computacional ajudando também na prevenção do *overfitting.* Cada camada convolucional possui um número variado de filtros. A saída de uma camada convolucional é um conjunto de mapas de ativação tridimensionais. Quando se trata da solicitação de profundidade de uma camada convolucional, ele corresponde ao número de filtros presentes. (Krohn; Beyleveld; Bassens, 2019)

A camada de *pooling* reduz a dimensionalidade desses mapas de ativação, mas sem afetar sua profundidade. Ela também possui um tamanho de filtro e um passo, com o seu movimento em torno de sua entrada semelhante ao de uma camada convolucional em uma imagem.

Geralmente, as camadas de *pooling* utilizam a operação *'max'*, tornando-se assim conhecidas como camadas de *'max-pooling'*. Elas operam de tal maneira que apenas o maior valor é mantido, enquanto todos os outros são descartados. Vamos supor que a camada de *pooling* tenha um filtro de tamanho 2 x 2 e uma etapa de 2. Neste caso, a camada de *pooling* irá considerar quatro ativações, reter apenas a maior e reduzir o tamanho das ativações por um fator de 4 conforme exemplo da Figura 12 (Krohn; Beyleveld; Bassens, 2019).

Figura12. Aplicação de *Max-pooling*.



Fonte:KROHN; BEYLEVELD; BASSENS (2019)

Preservando a informação mais importante e indiretamente acaba reduzindo também a quantidade de neurônios que serão necessários na camada da RNA, como a *full connected* (totalmente conectada).

3.4.5 Principais Funções De Ativação

A função *ReLU (Rectified Linear Unit*) é uma função de ativação não linear que não atinge a saturação, gerando o gradiente nesta região. Esta característica ajuda a evitar o problema de dissipação do gradiente, por esse motivo, é recomendada para redes neurais profundas (Hope; Resheff; Lieder, 2017).

A função retorna zero para todos os valores negativos e o próprio valor para todos os valores positivos, isso introduz a não-linearidade no modelo sem alterar a natureza dos dados de entrada. Todas as entradas que resultam em zero podem ser eliminadas do processo de treinamento aliviando o peso computacional (Hope; Resheff; Lieder, 2017).

A questão é a criação de neurônios zerados (neurônios mortos), isso se a entrada de um neurônio se tornar negativa, a saída será sempre zero, independe dos valores de entrada posteriores, podendo prejudicar a precisão da rede, pois esses neurônios deixarão de contribuir na aprendizagem.

ELU (*Exponential Linear Unit*) foi proposta como uma alternativa à função de ativação *ReLU*. *ELU* introduz valores de entrada menores que zero. Esta modificação resulta em uma ativação que passa por um valor negativo muito

próximo à origem ajudando o treinamento na busca pelo equilíbrio do gradiente (Hope; Resheff; Lieder, 2017).

ELU pode evitar o problema de "neurônios mortos" com um maior custo computacional.

Os valores negativos das *ELUs* contribuem para que a média das ativações sejam mais próximas de zero, o que permite uma aprendizagem acelerada na medida em que o gradiente se aproxima do seu nível natural. Além disso, as *ELUs* têm a propriedade de saturar para um valor negativo quando o argumento (valor de entrada) é bastante pequeno. Esta saturação resulta numa derivada menor, o que reduz a variação e a informação que é transmitida para a camada subsequente (Clevert; Unterthiner; Hochreiter, 2016).

Softmax é uma função de ativação usada principalmente na fase final da camada totalmente conectada é voltada para problemas de classificação, funciona como uma camada de ativação na classificação de classes produzindo valores de probabilidade para cada classe a maior probabilidade indica a classe selecionada.

A classificação baseada em *softmax* usa a operação como uma função de ativação para converter a saída da rede neural em uma distribuição de probabilidade de classe. (Palczynski et al., 2022).

A saída de um neurônio individual depende das saídas de todos os outros neurônios na mesma camada. Isso exige que a soma de todas as saídas seja sempre igual a 1, facilitando a interpretação das saídas como probabilidades. (Buduma; Locascio, 2017).

Para previsões, uma distribuição de probabilidade concentrada em uma única classe com valor próximo a 1, e as outras classes terão valores próximos a 0. Por outro lado, quando o modelo está incerto, as probabilidades serão mais distribuídas pelas classes. Assim, a camada *softmax* é fundamental para quantificar a confiança nas previsões. (Buduma; Locascio, 2017).

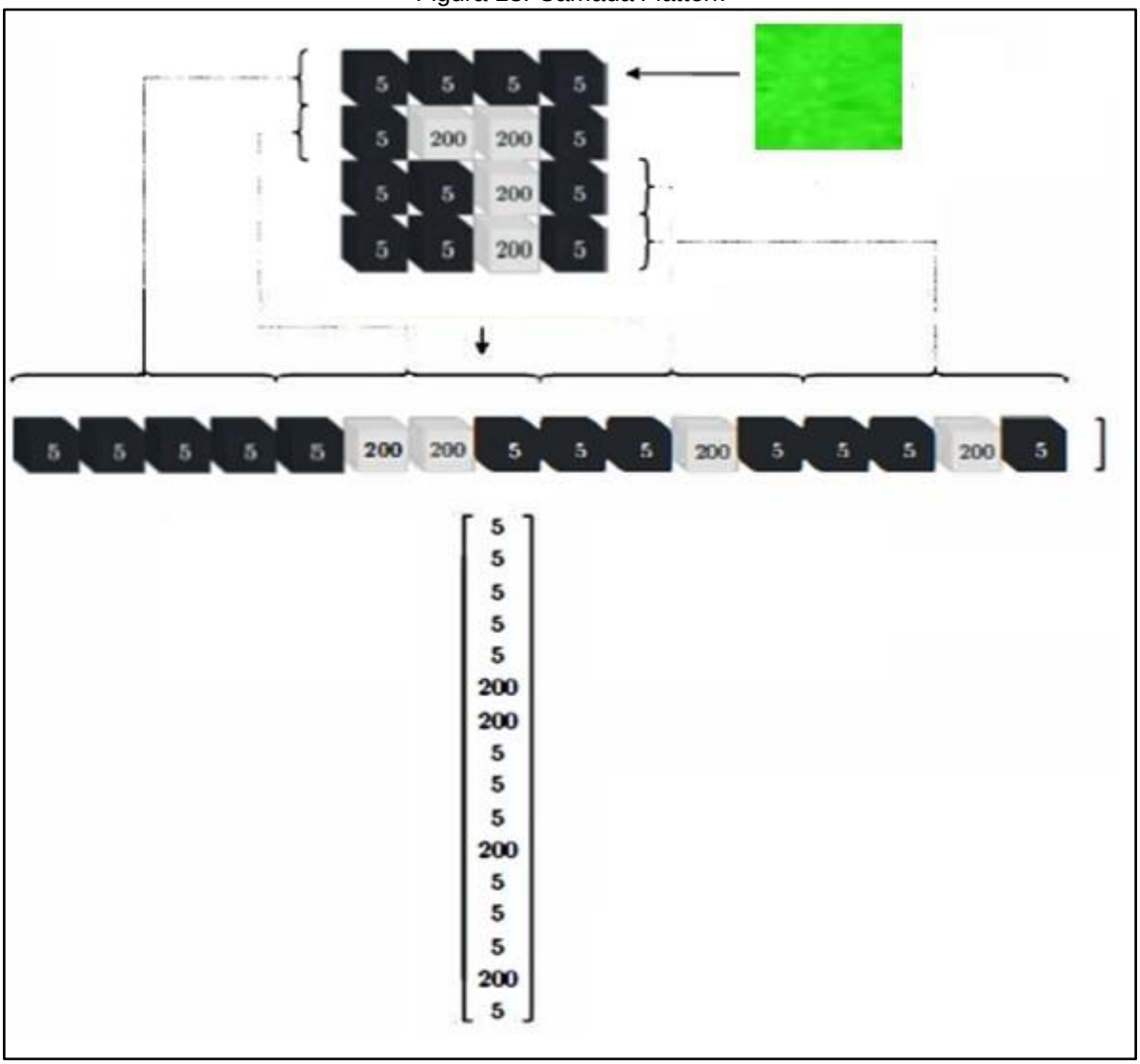
3.4.6 *Flatten*

Usado em *DL* principalmente para problemas que envolvem imagens digitais, serve para converter matrizes multidimensionais em vetores unidimensionais, simplificando desta forma a entrada para posterior processamento.

Na RNA tanto a entrada quanto a saída são vetores. Portanto, para processar dados que possuem dimensões espaciais, como imagens digitais, é necessário convertê-los em uma representação vetorial (Borrero; Arias, 2021).

A camada *flatten* está localizada logo após todas as camadas de convolução e *pooling* conforme a Figura 13. Os vetores são encaminhados para as camadas densas. A camada *flatten* representada pela figura 13 também é conhecida como uma ponte entre as últimas camadas de convolução/*pooling* e as primeiras camadas densas de uma rede neural convolucional.

Figura 13. Camada *Flatten*.



Fonte: Adaptado de BORRERO; ARIAS (2021)

A utilização desse recurso é obrigatória porque em uma *CNN* as saídas das camadas convolucionais e *pooling* têm a forma de volumes 3D, enquanto as camadas totalmente conectadas exigem vetores unidimensionais, este processo consiste em reorganizar o volume 3D de números em um vetor unidimensional (Millstein, 2018).

Flatten é um termo genérico usado para descrever o processo de transformação (achatamento) de uma estrutura multidimensional em um vetor unidimensional. Pode ser implementado de várias maneiras, dependendo do *framework* a ser utilizado.

3.4.7 Camada *Fully Connected*

Camada *fully connected* (camada totalmente conectada) faz parte integralmente das *CNNs*, que funciona como uma *perceptron* multicamadas seguido geralmente pela função de ativação *softmax*. Todos os neurônios da camada anterior estão integralmente conectados aos neurônios na camada subsequente. As *CNNs* utilizam as camadas de convolução e *pooling* para extrair características das imagens. Uma *fully connected* interpreta essas características para ajudar a categorizar as imagens entre classes conforme designado pelo conjunto de dados de treinamento do modelo (Millstein, 2018).

Incorporar camadas *fully connected* às *CNNs* não é apenas uma prática comum, mas também uma maneira eficiente para aprimorar os resultados na aprendizagem de dados de imagem.

Ao implementar uma camada *fully connected*, a soma das probabilidades de saída deve ser igual a um; para garantir isso funções de ativação *softmax* são usadas na sequência (Millstein, 2018).

3.4.8 *ResNet*

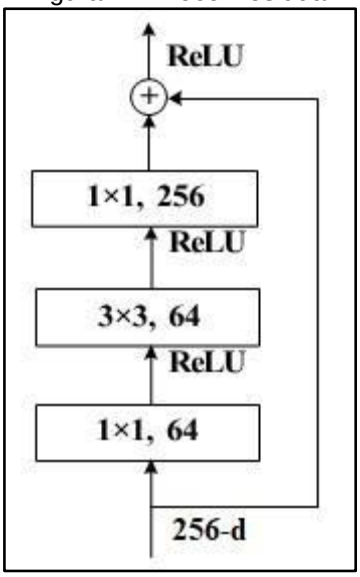
A primeira rede residual profunda, *ResNet*, foi introduzida pela *Microsoft Research* em 2015, logo tornando-se o líder do pacote de algoritmos de aprendizagem profunda que superou o desempenho humano no reconhecimento de imagens daquele ano (Krohn; Beyleveld; Bassens, 2019).

A *ResNet,* emprega um design que incorpora blocos residuais de três camadas ao invés do típico bloco de duas camadas conforme ilustrado na Figura 14. Este tipo de bloco é muitas vezes referido como "módulo gargalo" devido à sua forma específica, as duas terminações do bloco são mais estreitas que a seção do meio (Li et al. 2021).

A utilização de uma convolução de núcleo de 1×1 diminui o número de parâmetros na rede, o que pode aumentar a eficiência computacional e reduzir o risco de *overfitting*. Também realça a não linearidade da rede, o que pode melhorar a capacidade do modelo em aprender com dados (Li et al. 2021).

O uso desses módulos gargalo na *ResNet*, torna a rede mais eficiente, como também aumenta o poder da rede para realizar tarefas de aprendizado complexas, principalmente no campo do reconhecimento de imagens.

Figura 14. Bloco Residual



Fonte: LI et al. (2021)

A arquitetura do bloco residual da Figura 14, que é a base da *ResNet,* utiliza um método conhecido como mapeamento de identidade. Em contraste com outras arquiteturas onde os valores são passados diretamente através de cada camada empilhada da rede, a abordagem do bloco residual ajusta um mapeamento subjacente desejado ou constitui claramente camadas com mapeamento residual. (Krohn; Beyleveld; Bassens, 2019).

A equipe que desenvolveu a *ResNet* descobriu que essa configuração facilita a otimização do mapeamento residual em comparação com a otimização do mapeamento original. Isso é particularmente benéfico quando se trata de lidar com redes mais profundas. Portanto, ao aprofundar a rede, as *ResNets* alcançam ganhos de precisão consideravelmente melhores.

A rede utiliza as seguintes camadas a seguir:

Convolucional

Max pool

Average pool

Fully Connected

Softmax

Existem algumas derivações das arquiteturas da *ResNet.* A figura 15 apresenta a versão de 34 camadas, mas existem diferentes configurações conforme a seguir: ResNet18: 18 camadas, com convoluções 3x3

ResNet34: 34 camadas, com convoluções 3x3

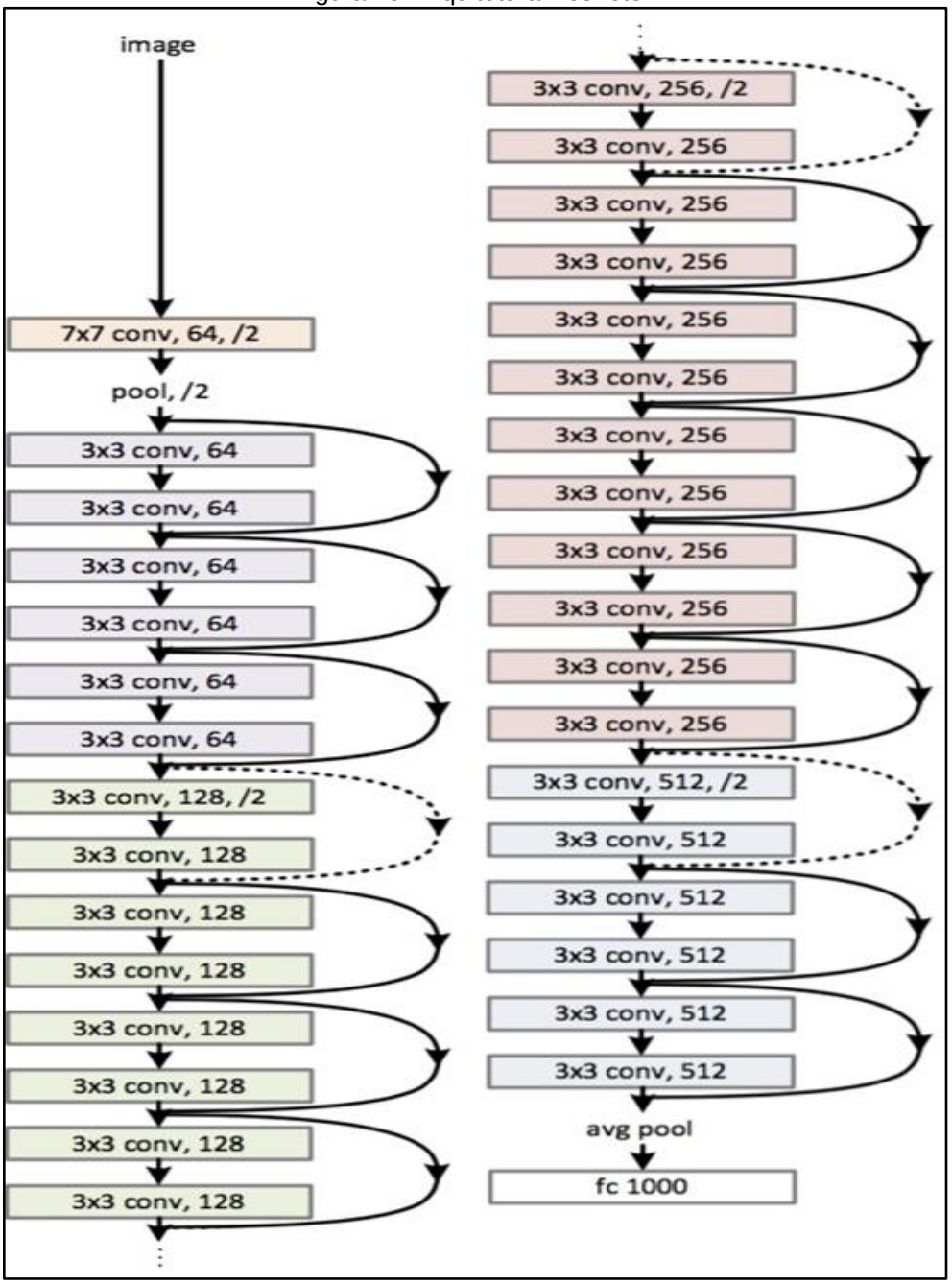
ResNet50: 50 camadas, com convoluções 3x3 e 1X1

ResNet101: 101 camadas, com convoluções 3x3 e 1X1

ResNet152: 152 camadas, com convolução 3x3 e 1x1

(Krohn; Beyleveld; Bassens, 2019).

Figura 15. Arquitetura Resnet34



Fonte: Adaptado KROHN; BEYLEVELD; BASSENS (2019)

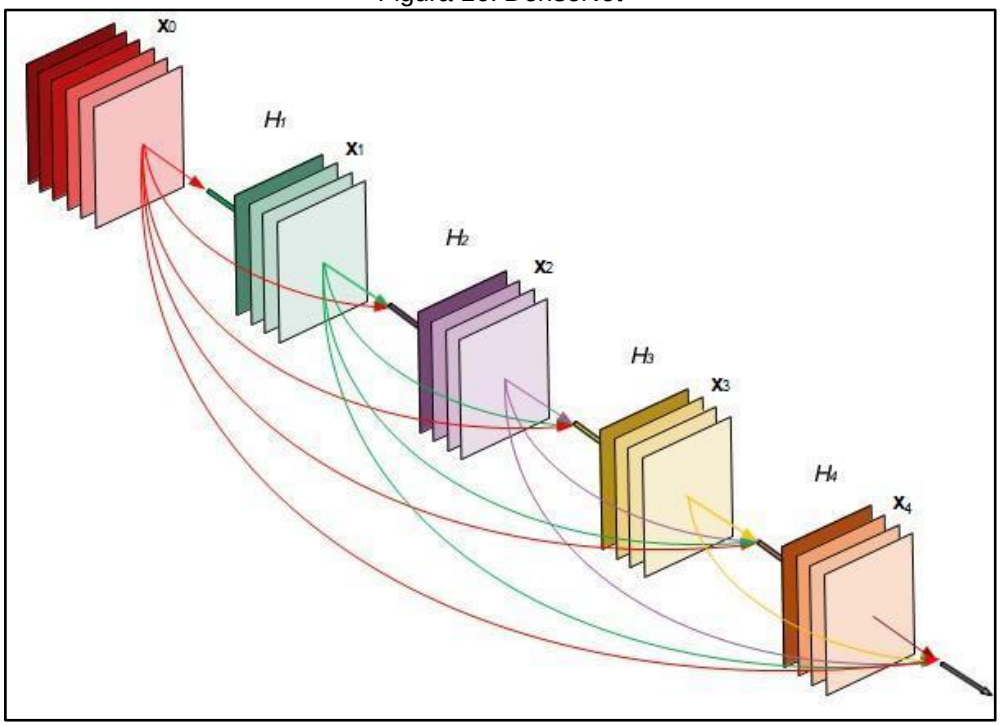
3.4.9 *DenseNet*

A *DenseNet* introduziu a ideia de redes neurais convolutivas completamente conectadas, onde todas as camadas de convolução estavam interligadas aos seus antecessores permitindo o compartilhamento de mapas de características entre camadas, ao mesmo tempo que reduz o número de parâmetros necessários para que a rede aprenda (Siebert, 2022).

Todas as camadas anteriores fornecem entrada para as camadas subsequentes, e a saída de uma camada específica é compartilhada com todas as camadas seguintes. Isto é conseguido através do uso da concatenação, criando uma forma de "sabedoria coletiva" entre as camadas. Desta maneira, é possível gerar um volume fixo de mapas de características para cada camada (Siebert, 2022).

Conforme Figura 16, no bloco denso de cinco camadas cada camada recebe todos os mapas de recursos anteriores como entrada.

Figura 16. *DenseNet*



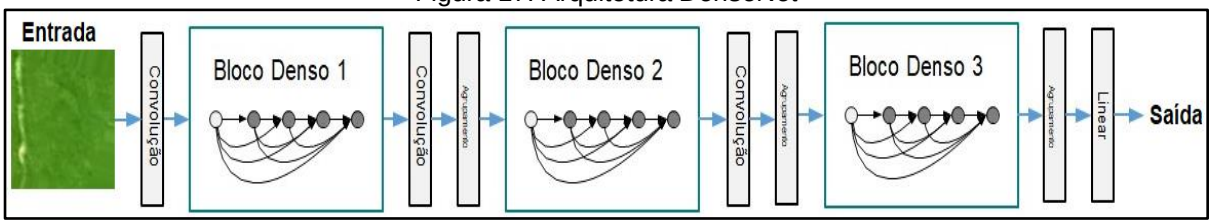
Fonte: HUANG, G. et al (2017)

A camada é otimizada para gerar mapas de características conforme a taxa de crescimento, também conhecida como 'k' (Huang, G., et al., 2017). Isso permite que as redes sejam mais compactas com menor número de canais, além de serem mais eficientes.

Existem duas subcamadas possíveis dentro de um bloco denso: a camada de gargalo e a camada densa ou camada de composição. A camada de gargalo é projetada para reduzir a complexidade e o tamanho da rede. Primeiro, realiza uma normalização em lote, seguida por uma ativação *ReLU* e finalmente uma convolução 1x1, resultando em saídas de 4 x k canais. A camada densa realiza uma convolução 3x3 com saídas 'k', que são concatenadas com os mapas de características já produzidos (Huang, G. et al., 2017).

A transição entre os blocos densos é gerenciada por camadas de transição. Estas camadas são responsáveis por adaptar a resolução dos mapas e diminuir o custo computacional da rede (Huang, G. et al, 2017).

Figura 17. Arquitetura *DenseNet*



Fonte: Adaptado de: HUANG, G. et al (2017)

Figura 17: *DenseNet* com três blocos densos. As camadas entre dois blocos adjacentes são chamadas de camadas de transição e alteram os tamanhos do mapa de recursos por meio de convolução e *pooling*. (Huang, G. et al 2017)

3.5 OTIMIZANDO OS MODELOS

Existem várias formas de avaliação do desempenho dos resultados de algoritmos usados para classificação de imagens. As medidas de qualidade, chamadas de métricas, registram exemplos reconhecidos como corretos ou incorretos em cada, e são construídas a partir da matriz de confusão. Para que não se favoreça nenhuma classe no desempenho da classificação não se foca em uma classe específica comparando-se de maneira geral os algoritmos. (Sokolova *et al.*, 2006).

Uma medida empírica muito empregada é a acurácia, que serve para definir o nível de exatidão dos resultados obtidos no experimento. Representando a proporção de predições corretas feitas pelo modelo em relação ao total de previsões, sem distinguir entre o número de rótulos corretos entre diferentes classes.

Baixa acurácia e *overfitting* são desafios que podem surgir, para estas questões foram desenvolvidas técnicas a fim de melhorar a eficiência dos modelos. Algumas das principais técnicas são: *Dropout, Data Augmentation, Data Preprocessing, Early Stopping, Learning Transfer* e *Grid Search*. Cada uma delas tem as suas próprias maneiras de promover melhorias e contribuir para aperfeiçoar os resultados, esses conceitos serão apresentados nos subtópicos desta sessão.

3.5.1 *Dropout*

O *dropout* é uma técnica de treinamento que elimina aleatoriamente neurônios durante a fase de treinamento para evitar o problema de sobreajuste. Durante cada interação de treinamento, o neurônio tem uma chance de 50% de ser eliminado, essa aleatoriedade ajuda a tornar o modelo mais robusto, pois o modelo não pode ter como principal referência um neurônio individual, ou apenas poucos neurônios dentro dos milhões existentes na rede (Krizhevsky *et al*., 2012).

Durante a fase de teste, todos os neurônios são usados, mas suas saídas são ajustadas para compensar o fato de que mais neurônios estão ativos do que durante o treinamento. Com a aplicação dessa técnica, o tempo de convergência do modelo pode dobrar, mas com o benefício de redução de erros (Krizhevsky *et al*., 2012).

Essa técnica resulta em uma arquitetura em que, em distintos momentos do treino, a rede possui diferentes neurônios ativados. O que implica em menos *overfitting* e em uma rede de aprendizado com atributos de forma mais robusta.

3.5.2 *Data-Preprocessing*

Esta é uma etapa que vai depender do propósito a ser alcançado pelo modelo de inteligência artificial, basicamente consiste em preparar o *dataset* para posterior processamento da melhor forma possível. Afinal inserir dados ruins em qualquer sistema é muito prejudicial principalmente em uma *DL* que estará avaliando minuciosamente cada partícula dos dados inseridos.

Pré-processar os dados é um passo que pode incluir tarefas como: limpeza, transformação, redução, integração de dados e ampliação de dados, conforme o problema a ser resolvido (Garcia; Luengo; Herrera, 2015).

Na limpeza erros são corrigidos, campos vazios preenchidos, valores discrepantes removidos. A transformação altera a estrutura dos dados para facilitar a análise priorizando uma padronização. A redução busca simplificar o conjunto de dados através da subamostragem ou da redução de dimensionalidade. A integração visa combinar dados de diferentes fontes em um único conjunto. Por fim, a ampliação de dados envolve a criação de novos dados por meio de modificações nos dados originais existentes (Garcia; Luengo; Herrera, 2015).

Segundo Géron (2019), as sugestões a seguir contribuem para reduzir o problema de o*verfitting:*

• Simplificar o modelo: escolher um modelo com menos parâmetros (menos camadas).

• Coletar mais dados de treinamento: um conjunto de dados de treinamento maior pode ajudar a suavizar o ruído e a permitir que o modelo reconheça padrões de forma mais eficaz.

• Reduzir o ruído nos dados de treinamento: isso pode envolver a limpeza nos dados corrigindo ou removendo os *outliers*.

3.5.2.1 *Data Augmentation*

O D*ata Augmentation* consiste em aumentar artificialmente uma base de dados desbalanceada gerando imagens novas a partir das imagens já existentes por meio da aplicação de transformações de rotação, espelhamento, redimensionamento, mudança de brilho, entre outros, gerando imagens artificiais para aumentar o *dataset* (Maione, 2020).

É um dos processos que podem ser utilizados para fazer o balanceamento de dados equiparando um determinado conjunto de dados cuja quantidade de amostras de cada classe disponível para análise tornando-se igualmente ou aproximadamente representadas diante do processo de aprendizagem.

A importância deste processo se dá pelo fato de que diversos modelos de classificação utilizados frequentemente em projetos são para trabalhar com conjuntos de dados balanceados. Entretanto, ao se trabalhar com conjuntos de dados e aplicações reais, o provável é que os dados disponíveis para análise estejam desbalanceados, dados desbalanceados são inconvenientes e prejudicam o desempenho (Maione, 2020).

3.5.3 *Grid Search*

No *grid search* o usuário seleciona o conjunto de valores para testar. O algoritmo então executa o treino do modelo para cada especificação dos melhores valores de hiperparâmetros informados, produzindo pelo experimento o melhor conjunto utilizado para o melhor resultado.

As listas de valores a serem parametrizadas com base em resultados de bons experimentos anteriores, para garantir que o valor ótimo provavelmente esteja no intervalo selecionado. Geralmente uma pesquisa com essa ferramenta envolve escolher valores aproximadamente em uma escala.

O problema com o *grid search* é o custo computacional que cresce exponencialmente com muitos hiperparâmetros pré-estabelecidos para treinamento. (Krohn; Beyleveld; Bassens, 2019)

3.5.4 *Learning Transfer*

Essa vertente dedica-se ao processo de aproveitar a aprendizagem obtida anteriormente para facilitar o aprendizado em novos problemas.

A aprendizagem por transferência transfere o conhecimento aprendido de modelos sem ter que utilizar grandes conjuntos de dados para se obter melhores resultados na generalização. Pré-Treinamento, *File-Tuning*, *Meta-learning* e FSL são alguns dos conceitos que existem em aprendizado por transferência (Wang; Chen, 2023).

Na maioria das aplicações de pré-treinamento e *file-tuning*, suas categorias não são as mesmas, em situações do mundo real as redes não são treinadas do zero para uma nova tarefa, por que é extremamente caro, especialmente no caso em que os dados de treinamento não são o suficiente.

Os modelos pré-treinados são frequentemente treinados em grandes conjuntos de dados, o que aumenta a capacidade de generalização do modelo ao realizar o *file-tuning* (Wang; Chen, 2023).

O pré-treinamento e o ajuste fino apresentam melhoria de desempenho em comparação com o treinamento do zero, quanto melhor for o modelo pré-treinado, melhores serão os resultados. Em comparação com a inicialização aleatória, o ganho por pré-treinamento e ajuste fino são melhores (Wang; Chen, 2023).

Seu funcionamento começa após o treinamento de um modelo em um grande conjunto de dados, esse modelo é salvo para uso futuro. O *file-tuning* envolve pegar o modelo salvo e executar um refinamento para ajustar os pesos profundos do novo modelo para uma nova tarefa.

Meta-learning e *FSL* são conceitos correlacionados que juntos podem aprender o conhecimento a partir de múltiplas tarefas inteiras treinadas em

conjuntos de dados diferentes, como classificadores onde os termos treinamento e validação estão presentes em cada tarefa e são chamados de conjunto de suporte e conjunto de consulta, desta forma elas também possuem relação com os conceitos de *learning transfer* (Wang; Chen, 2023).

O *FSL* refere-se à situação em que não há recursos de *hardware* ou dados rotulados suficientes para treinar um modelo ou mesmo realizar o ajuste fino adaptando o modelo para novas tarefas.

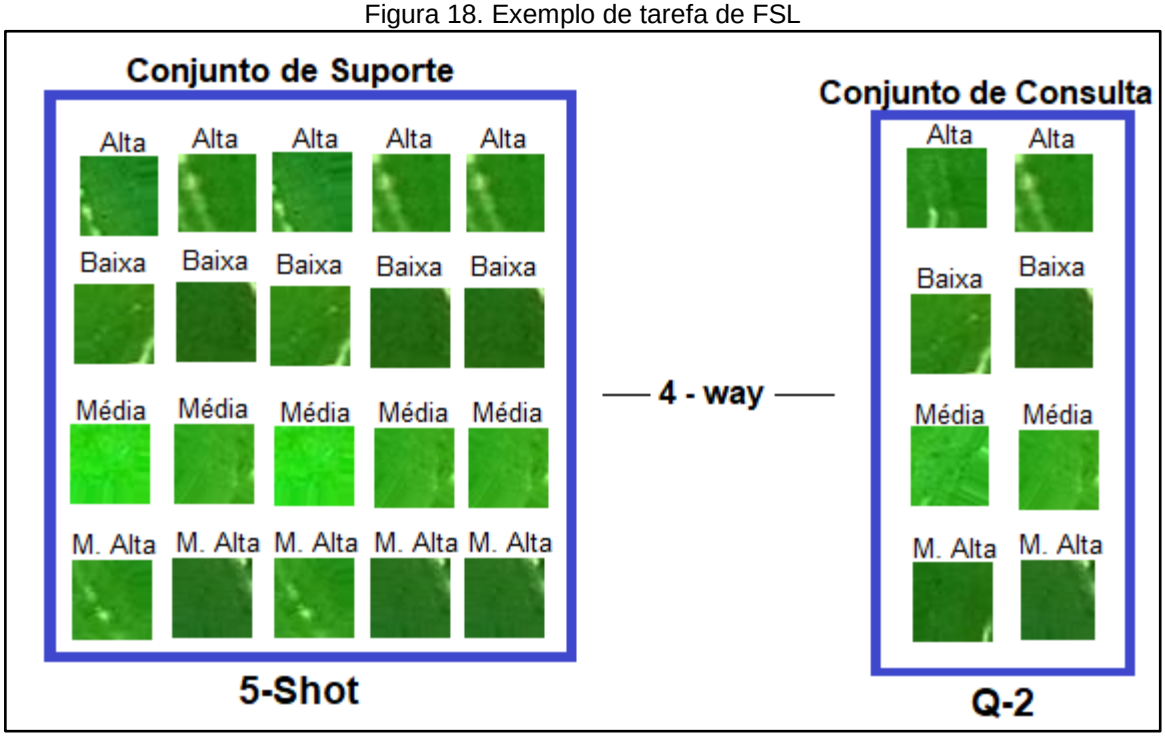
3.6 *FEW-SHOT LEARNING*

A pesquisa de Koch *et al*. (2015) foi uma das primeiras a utilizar a combinação de *FSL* com CNN.

Os modelos de *FSL* são compostos por tarefas de classificação baseadas no paradigma *N-way*, *K-shot*. O parâmetro N é o número de classes envolvidas e K representa as amostras de suporte de cada uma das classes. O parâmetro Q representa o número de exemplos para consulta. Normalmente um volume considerável de dados se faz necessário para a maioria das aplicações. *FSL* considera a quantidade mínima para treinamento por classe, por isso o parâmetro K costuma variar entre 1 e 10 *shots* (Gomes, 2022).

A Figura 18 mostra um exemplo da utilização do *FSL* cujo N representa a quantidade de tarefas de classificação, N=4 (*n-way*) sendo elas: alta, baixa, média e muito alta. O K=5 (*k-shot*) representa a quantidade de imagens contidas em cada classe. O conjunto de consulta Q=2 é a quantidade de 2 amostras.

Figura 18. Exemplo de tarefa de FSL



Fonte: O autor (2023).

Este subcampo do aprendizado de máquina concentra-se na quantidade de utilização mínima de exemplos, em vez de grandes quantidades de dados para treinamento. O *FSL* é geralmente visto como uma tarefa a ser resolvida através do conceito de *Meta-Learning*, que por sinal é uma das aplicações ou propósitos da *Meta-Learning*. Embora o *FSL* e a *Meta-Learning* pareçam semelhantes, a *Meta-Learning* é muito mais ampla. (Zou, 2022).

FSL é sobre aprender utilizando um pequeno número de exemplos. O objetivo é criar um modelo que possa realizar essa tarefa.

Meta-Learning é sobre algoritmos de aprendizagem que aprendem com outros algoritmos de aprendizagem para se adaptar rapidamente a novas tarefas.

FSL e *Meta-Learning* têm conexão quando a capacidade de um modelo de aprender rápido (objetivo do *Meta-Learning)* é crucial para o bom funcionamento em tarefas de *FSL*. Se existe um modelo que pode ser eficaz na aprendizagem a partir de poucos exemplos (objetivo do *FSL*), ele está demonstrando uma habilidade de se adaptar rapidamente a uma nova tarefa, por isso muitas soluções que envolvem *FSL* são baseadas em técnicas de *Meta-Learning*

3.7 *META-LEARNING* PARA *FSL*

A *Meta-Learning* é uma das áreas de pesquisa mais promissoras no campo da inteligência artificial para alcançar a Inteligência Geral Artificial (*AGI*, do inglês, *Artificial General Intelligence*) (Ravichandiran, 2018).

AGI é considerada uma IA forte e refere-se a um tipo de IA que possui a capacidade de entender, aprender, adaptar-se e implementar o conhecimento em tarefas de maneira geral, semelhante à inteligência humana.

Algoritmos aprendem do zero, já os humanos entram na tarefa com uma grande quantidade de conhecimento prévio em seus cérebros. Em vez de aprender do zero, os humanos estão ajustando e recombinando um conjunto de habilidades pré-existentes (Nichol; Schulman, 2018).

A *Meta-Learning* visa “aprender como modelos de aprendizagem aprendem”, e não apenas generalizar os dados como faz as principais abordagens (Rêgo *et al.*, 2022).

Os algoritmos de aprendizado geralmente dominam apenas uma tarefa, *Meta-Learning* no entanto, permite treinar um modelo em várias tarefas relacionadas com poucos dados.

Segundo Ravichandiran (2018), a *Meta-Learning* é subdividida em três tipos de aprendizagem e todas elas contemplam utilização do método de *FSL.*

Aprendizagem métrica.

Aprendizagem por otimização.

Aprendizagem das inicialização.

Nas próximas duas subseções serão apresentadas as duas categorias de *Meta-Learning* que foram utilizadas na pesquisa, a aprendizagem métrica e a aprendizagem por inicialização.

3.7.1 Aprendizagem Métrica

Este método é composto por uma *CNN* que extrai recursos de duas imagens ao mesmo tempo e encontra a semelhança calculando a distância entre as características extraídas das duas imagens. Os modelos baseados em aprendizagem métrica dizem respeito à eficiência na comparação de características nas imagens. Essa abordagem é amplamente usada em um ambiente *FSL*, as

Redes Siamesas são consideradas um modelo de aprendizagem métrica. (Ravichandran, 2018).

A métrica de distância adequada para o desempenho do modelo deve representar a relação entre as entradas no espaço métrico e auxiliar na resolução de problemas (Zou, 2022).

3.7.1.1 Redes Siamesas

Esse é um modelo clássico de aprendizagem métrica onde os pares de imagens utilizam e compartilham a estrutura de rede, estrutura essa que é duplicada para ambas imagens, para a extração de características de cada uma delas, com compartilhamento de pesos entre as duas imagens. O modelo apresentado na Figura 19 aprende usando uma diferença métrica que é calculada entre as características extraídas das imagens que pode ser a distância euclidiana, *manhattan*, cosseno ou outra distância métrica. A rede neural siamesa é comumente usada quando as entradas apresentam similaridade (Zou, 2022).

Figura 19. Estrutura das Redes Siamesas.

Fonte: ZOU (2022)

A função de distância métrica busca minimizar a distância entre imagens da mesma classe e maximizar a distância entre imagens de classes diferentes (Argüeso, et al., 2020)

Durante o treinamento é comparado a distância entre as representações das duas entradas na rede siamesa. Esta comparação resulta em um valor de perda, que é então usado para atualizar os pesos das sub-redes através do processo de *backpropagation* (Goodfellow et al., 2016).

A ideia principal de se usar as arquiteturas de redes siamesas é aprender a discriminar melhor entre as imagens de entrada. O compartilhamento de pesos garante que duas imagens semelhantes não sejam mapeadas para locais diferentes no espaço de incorporações de recursos (Jadon; Garg, 2020).

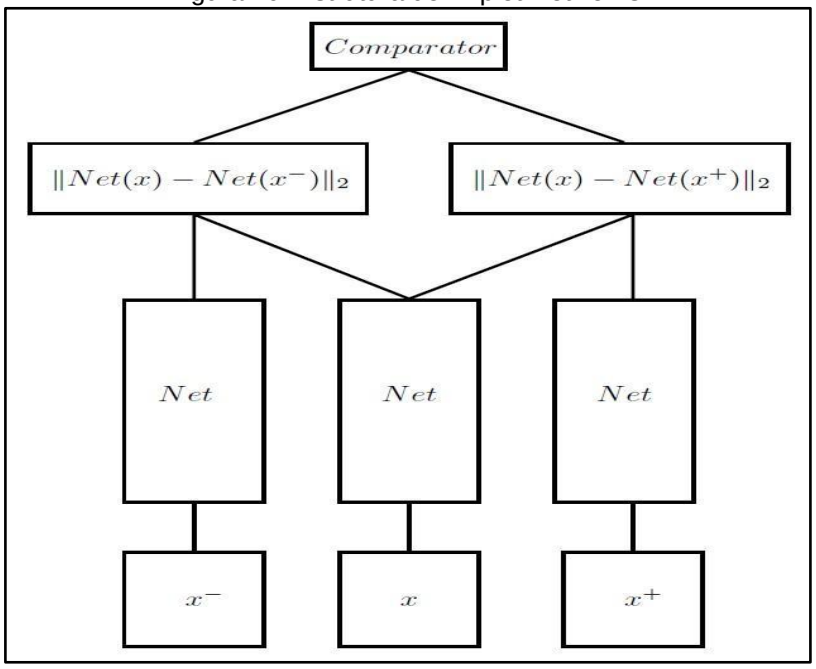
Todavia, modificações na estrutura dessas redes permitem a sua utilização para tarefas de classificação, uma delas é a utilização da camada de saída *softmax* que é uma função normalmente usada em camadas de saída de redes destinadas à classificação.

3.7.1.2 *Triplet Networks*

As redes *Triplet Networks* operam de maneira similar às Redes Siamesas. As três imagens utilizam a mesma estrutura de camadas, com a diferença que nesse modelo são três instâncias (com parâmetros compartilhados) ao invés de duas.

Em uma tarefa de classificação: a imagem X, um exemplo positivo X+ pertencente à mesma classe de X. Um exemplo negativo X- pertence a uma classe diferente. É aferido o par de distâncias entre cada de X+ e X- contra o referência X conforme ilustração da Figura 20 (Hoffer; Ailon, 2015).

Figura 20. Estrutura de Triplet Networks.



Fonte: HOFFER; AILON(2015).

O treinamento é realizado alimentando a rede com amostras onde a *CNN* faz a extração dos recursos das imagens para então fazer três comparações de distâncias em vez de uma como nas Redes Siamesas. A questão dos parâmetros compartilhados significa que todas as três segmentações de camadas são interligadas para que as mesmas tenham exatamente os mesmos valores de pesos e hiperparâmetros. É comparado então o (x, x+) e (x, x-), a terceira e última comparação é entre resultados dessas duas comparações.

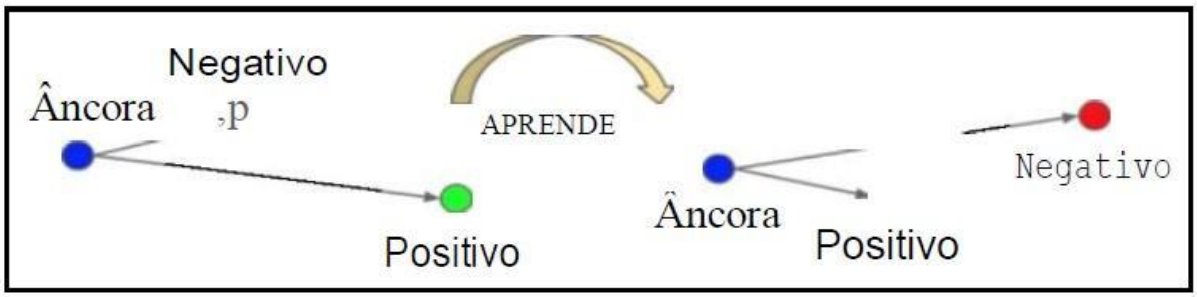
O treinamento desse modelo busca se aproximar das características entre as imagens da mesma classe e se distanciar das imagens das classes diferentes, conforme sugerido a seguir (Jadon; Garg, 2020):

• Âncora (A): Dados Principais (âncora)

• Positivo (P): Dados Positivos (semelhante a âncora)

• Negativo (N): Dados negativos (diferentes da âncora)

Figura 21. Esquema função de perda triplete



Fonte: JADON; GARG (2020)

A Figura 21 representa a função de *triplet loss* que converge melhor que a função de perda contrastiva (considera apenas par a par na rede siamesa) porque considera três exemplos por vez, mantendo a distância entre os pontos positivo e negativo (Jadon; Garg, 2020).

3.7.2 Aprendizagem Por Inicialização

É usada para ajustar os modelos de aprendizado para aprender através da otimização dos pesos iniciais. O processo de otimização irá iterar através de diferentes valores dos pesos para encontrar a combinação que resulta no melhor desempenho do modelo.

Este método tenta aprender valores iniciais ótimos para os parâmetros (pesos) do modelo. Após iniciar o treinamento a perda é calculada e minimizada por meio de um gradiente descendente, durante as iterações. Se os pesos forem iniciados com valores ótimos ou próximos dos valores ótimos a convergência pode ser atingida rapidamente (Ravichandiran, 2018).

Um algoritmo bastante utilizado para implementar esse tipo de otimização é o *Reptile* que propõe uma abordagem para otimização dos parâmetros, com foco na rapidez e eficiência no aprendizado de múltiplas tarefas, com menor custo computacional e melhor desempenho que o seu antecessor o *MAML (Model Agnostic Learning)* (Zou, 2022).

3.7.2.1 Algoritmo *Reptile*

Segundo os seus criadores, Nichol e Schulman (2018), o algoritmo *Reptile* foi proposto como uma melhoria para um algoritmo chamado *MAML* que tem bastante destaque no âmbito do M*eta-Learning*.

O *Reptile* se concentra em minimizar a função de perda de expectativa e maximizar a capacidade de generalização em cada tarefa. *Reptile* não precisa de treinamento por meio de testes de validação cruzada de dados para cada tarefa. (Zou, 2022)

O algoritmo *Reptile* é iterativo e cada iteração é chamada de 'episódio'. Num episódio, o algoritmo seleciona uma tarefa aleatoriamente, realiza múltiplas atualizações dos pesos da rede neural usando *SGD* (do inglês *Stochastic Gradient*

Descendent) no conjunto de treinamento dessa tarefa, e então move os pesos iniciais em direção aos pesos finais após o treinamento. A proporção dessa movimentação é controlada por um hiperparâmetro chamado taxa de aprendizado externa (Ravichandiran, 2018).

Reptile possui uma vantagem crucial, que é uma implementação mais simples: ao contrário do *MAML*, ele não requer *backpropagation* através das etapas de otimização interna, o que pode ser computacionalmente caro, na Figura 22 é apresentado algoritmo (Ravichandiran, 2018).

Figura 22. Algoritmo *Reptile*

Algorithm 2 Reptile, batched version

Initialize ϕ
for iteration $= 1, 2, \ldots$ **do**
  Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
  for $i = 1, 2, \ldots, n$ **do**
    Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \phi, k)$
  end for
  Update $\phi \leftarrow \phi + \epsilon \frac{1}{k} \sum_{i=1}^{n} (W_i - \phi)$
end for

Fonte: (Nichol; Schulman, 2018).

A fórmula atualiza os pesos da seguinte forma:

$\theta \leftarrow \theta + \varepsilon(\theta' - \theta)$

onde,

θ são os pesos iniciais;

θ' são os pesos após várias atualizações no SGD na tarefa;

ε é a taxa de aprendizagem.

Essa atualização desloca os pesos iniciais em direção aos pesos finais após o treinamento na tarefa. O quanto é deslocado é controlado pela taxa de aprendizado ε (Nichol; Schulman, 2018).

Em cada episódio do *Reptile ocorrem os seguintes passos*:

1. Seleciona-se uma tarefa aleatória do seu conjunto de tarefas (se existir mais de uma tarefa).

2. Faz-se uma cópia dos pesos iniciais atuais da rede.

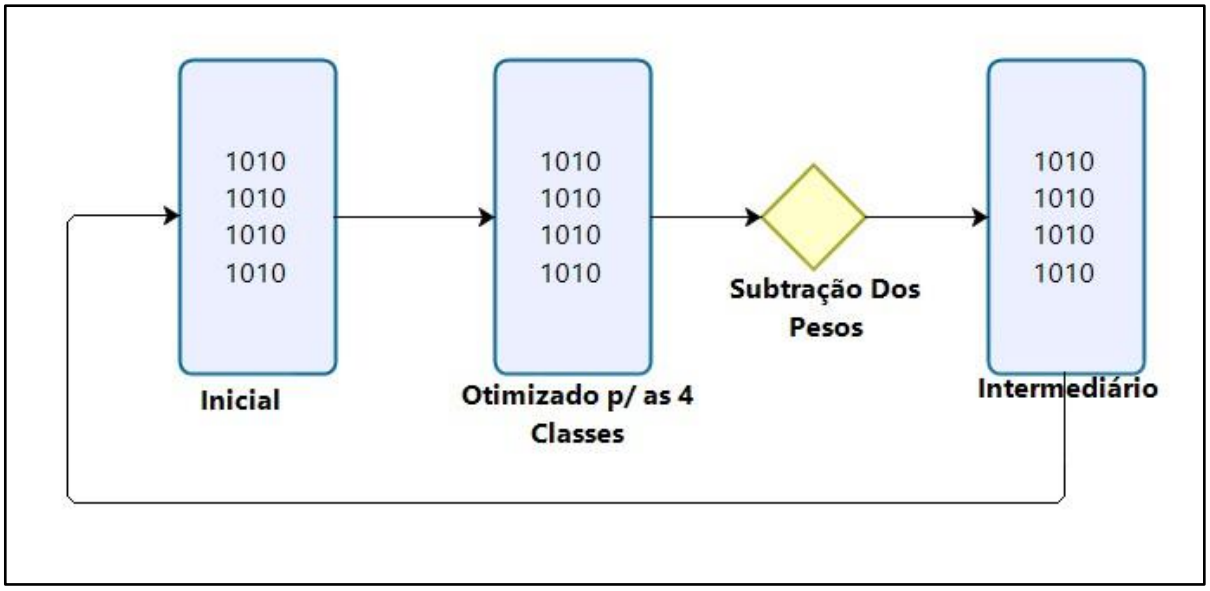
3. Executa-se várias etapas do *SGD* nas tarefas selecionadas.

4. Atualiza-se os pesos iniciais usando a fórmula: $\theta \leftarrow \theta + \varepsilon(\theta' - \theta)$

Esta atualização de pesos é então repetida para múltiplos episódios, criando um *loop* de aprendizagem que, em última análise, permite que o modelo se adapte rapidamente para desempenhar bem novas tarefas. A cada episódio, o algoritmo

Reptile aprende a iniciar a partir de um conjunto de pesos que são mais fáceis de ajustar para executar bem a tarefa. Assim, permite que o modelo seja adaptado rapidamente a novas tarefas (Nichol; Schulman, 2018). Esse fluxo está representado na Figura 23.

Figura 23. Fluxo do Reptile



Fonte: O autor (2023).

No algoritmo *Reptile*, na prática, fornece-se ao modelo uma quantidade limitada de imagens (conceito do *FSL*). Após cada sessão, analisa-se o desenvolvimento do modelo e as modificações efetuadas.

Ao invés de simplesmente preservar a aprendizagem obtida após cada tarefa, busca-se investigar quais mudanças foram compartilhadas entre todas as tarefas durante a sessão. O intuito é descobrir os aspectos benéficos para a identificação geral das imagens.

O modelo vai se ajustando até se tornar gradualmente mais hábil na identificação de novas categorias de imagens com base naquilo que já aprendeu. Repete-se o procedimento por várias vezes, em cada repetição segue-se selecionando tipos distintos de imagens. Isso resulta em um modelo que se torna progressivamente melhor em identificar categorias de imagens.

3.8 TRABALHOS CORRELATOS

O Quadro 2 apresenta um resumo das pesquisas identificadas utilizando imagens obtidas por *RPAs* e técnicas de *FLS* na agricultura.

Por meio do levantamento feito (Quadro 2) e ao decorrer do estudo foi possível perceber que atualmente não consta nenhum estudo utilizando a técnica de *FSL* para classificar imagens de produtividade da soja, tornando-se um campo de pesquisa a ser explorado para a construção de modelos de IA com este conceito.

Os estudos de Gomes (2022) e Tassis e Krohling (2022) têm um foco similar entre si ao utilizarem *FSL* em combinação com *CNNs* e Redes *Triplet* para a classificação de insetos e doenças em plantas, respectivamente. Os estudos mostraram resultados promissores na aplicação das técnicas. No trabalho de Gomes (2022), foi proposto um modelo para classificação de imagens de insetos da agricultura, alcançando melhoria na acurácia de 3.62% em tarefas de classificação *1-shot* e 2.82% em tarefas de *5-shot.*

Já Tassis e Krohling (2022) utilizaram *FSL* com CNNs e Redes *Triplet* para a classificação de estresse biótico em folhas de café. Ambos os estudos demonstram a aplicabilidade na agricultura, mas em contextos distintos utilizando imagens de RPA e técnicas de *FSL* aplicadas na agricultura.

Quadro 2. Imagens de RPA e técnicas de FSL na agricultura.

Autor	**Finalidade**	**Método**
GOMES, J. C. (2022)	Classificação de Insetos.	*Few-shot learning* + CNN + *Triplet network*
KARAMI, A.; CRAWFORD, M.; DELP, E. J (2020)	Contagem e Localização Da Planta De Milho	*Few-shot learnin*g + CNN
TASSIS, L. M.; KROHLING, R. A. (2022)	Classificação de doenças em folhas de Café	*Few-shot learning* +CNN + *Triplet network*
GUI, J.; XU, H.; FEI, J. (2022)	Detecção de Pragas na cultura da Soja	*Resnet + Meta-learning (métricas)*
LI, Y.; YANG, J. (2021)	Detecção de Pragas ou Plantas	*Few-shot learning* + CNN
Li, Y.; CHAO, X. (2021)	Detecção de Pragas em diversas culturas.	*Few-shot learning* + CNN
GARG, S.; SINGH, P (2023)	Classificação De Doenças Foliares em plantas.	*Few-shot learning* + CNN + t*riplet network*
LI, M.; YAO, H.; WANG, Y. (2023)	Classificação De Doenças Em Plantas	*Few-shot learning* + CNNs diversas
PAN, J.; WANG, T.; WU, Q. (2022)	Identificação de Doenças no Arroz	*Few-shot learning +,* CNNs diversas + *Siamese Network*
KIM, W. ET AL (2021)	Classificação De Solos	*Few-shot learning* + CNN

Fonte: O autor (2023).

Similarmente, Karami, Crawford e Delp (2020) e o trabalho de Li e Chao (2021) também exploraram o uso de *FSL* e *CNNs* em seus estudos, centrando o foco na contagem e localização de plantas de milho e na detecção de pragas em diversas culturas. Com os resultados obtidos mostrou-se uma abordagem viável para as tarefas propostas.

O estudo de Garg e Singh (2023) utilizou a *ResNet* com um algoritmo de *Meta-Learning* para a detecção de pragas na cultura da soja e aponta para a necessidade de otimização de hiperparâmetros e seleção de resoluções. Estes desafios, também foram enfrentados na pesquisa atual que trabalha na identificação de imagens de produtividade da mesma cultura com o uso de diversas arquiteturas de *CNN*, como *ResNet* e *DenseNet*, acopladas à técnica *FSL* e *Meta-Learning*

Os estudos de Gui, Xu e Fei (2022) e também de Li, Yao e Wang (2023) utilizam o *Meta-Learning*, *FSL* e *CNN* na detecção de pragas na soja e na classificação de doenças em plantas, respectivamente. Gui, XU e Fei (2022), em particular, afirmam sobre a eficácia da combinação de técnicas hiperespectrais de imagem e de *Meta-Learning* na detecção de pragas da soja, atingindo uma precisão de 94,57%. Em contrapartida, Li, Yao e Wang (2023) focaram na questão da classificação das doenças das plantas de baixa granularidade e propuseram um módulo de atenção híbrida que auxiliou na melhora da precisão da classificação.

Da mesma forma, Li e Yang (2021) também aplicaram o *FSL* e as *CNNs* na detecção de pragas e plantas, tendo sido o primeiro trabalho a aplicar a classificação de *Meta-Learning* e *FSL* no campo agrícola.

Pan, Wang e Wu (2022) abordaram a identificação de doenças do arroz através de um método em dois estágios, incorporando o *Yolo* e uma Rede Siamesa, evidenciando o potencial das redes siamesas em associação com *FSL*.

Kim *et al.* (2021) demonstraram o potencial do *FSL* com *CNN* na segmentação efetiva do solo cultivado e não cultivado, além de suas implicações para a orientação autônoma de tratores.

Todos os trabalhos relacionados mencionados abordam o uso de *FSL* para IA aplicada na agricultura. Embora com abordagens e focos variados, é notável a semelhança nas técnicas adotadas, especialmente com uso de *CNNs*, reforçando a importância dessas redes para a classificação de imagens.

As comparações traçadas ressaltam a originalidade e relevância do presente estudo para novos avanços em um cenário de classificação imagens de produtividade no cultivo de soja.

4 MATERIAIS E MÉTODOS

4.1 ORIGEM DOS DADOS

A base de dados utilizada neste trabalho foi criada a partir das imagens obtidas durante o desenvolvimento do trabalho de Viniski (2019), no qual foram realizados voos com uma *RPA* e captadas imagens com o objetivo avaliar a eficiência do uso da mineração de dados clássica e espacial na estimativa de produtividade de grãos em imagens obtidas por meio de *RPA*.

O estudo foi realizado na Fazenda Paiquerê, município de Piraí do Sul, Paraná, Brasil, cujas coordenadas de referência são 50o7’ 51,70” S e 24^{o}21’23.485, com altitude média de 970m (Figura 24). A área imageada a aproximadamente 60,4 ha, com solo classificado como latossolo vermelho-amarelo com textura argilosa, cultivado sob o plantio direto. A cultura analisada foi a soja safra 2016-2017 (Figura 21). Detalhes sobre cultivares e adubação, assim como sobre os voos realizados e geração das imagens, estão descritos em Viniski (2019).

Figura 24. Área de estudo - soja de 2016-2017

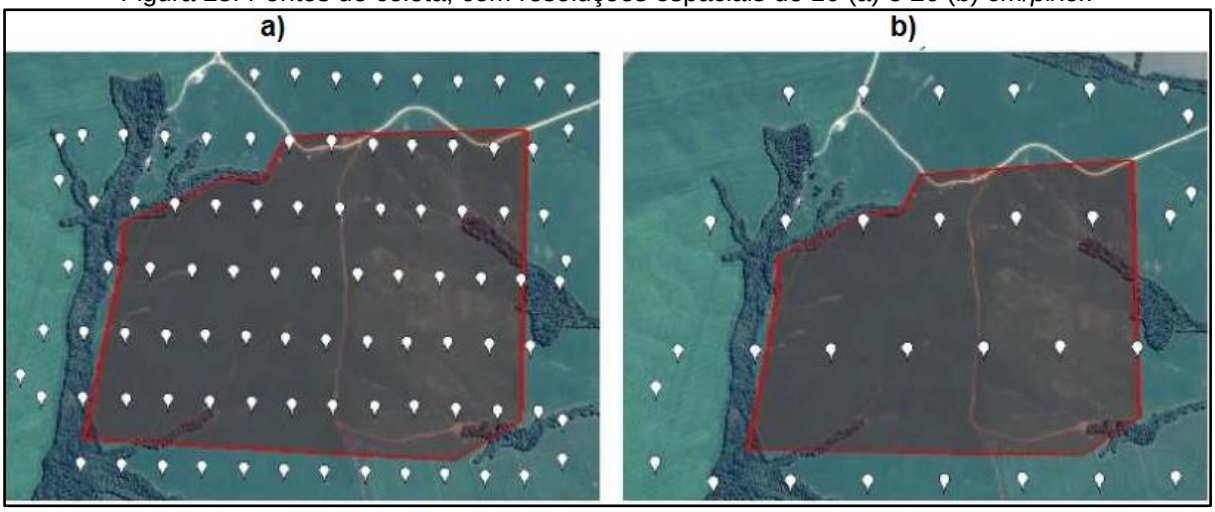
Fonte: Adaptado de VINISKI, 2019.

4.1.2 Coleta Dos Dados

As imagens originais (formato *JPEG*) foram obtidas com uma câmera *RGB*, de dois voos realizados em 20/02/2017, sendo um voo utilizando a resolução espacial de 10 cm/*pixel* e outro com resolução de 26 cm/*pixel*.

Na Figura 25 é possível verificar os pontos de tomada das imagens na área de soja, nas resoluções de 10cm/*píxel* (Figura 25a) e 26 cm/*píxel* (Figura 25b), com 91 e 25 imagens, respectivamente (Viniski, 2019).

Figura 25. Pontos de coleta, com resoluções espaciais de 10 (a) e 26 (b) cm/*pixel*.

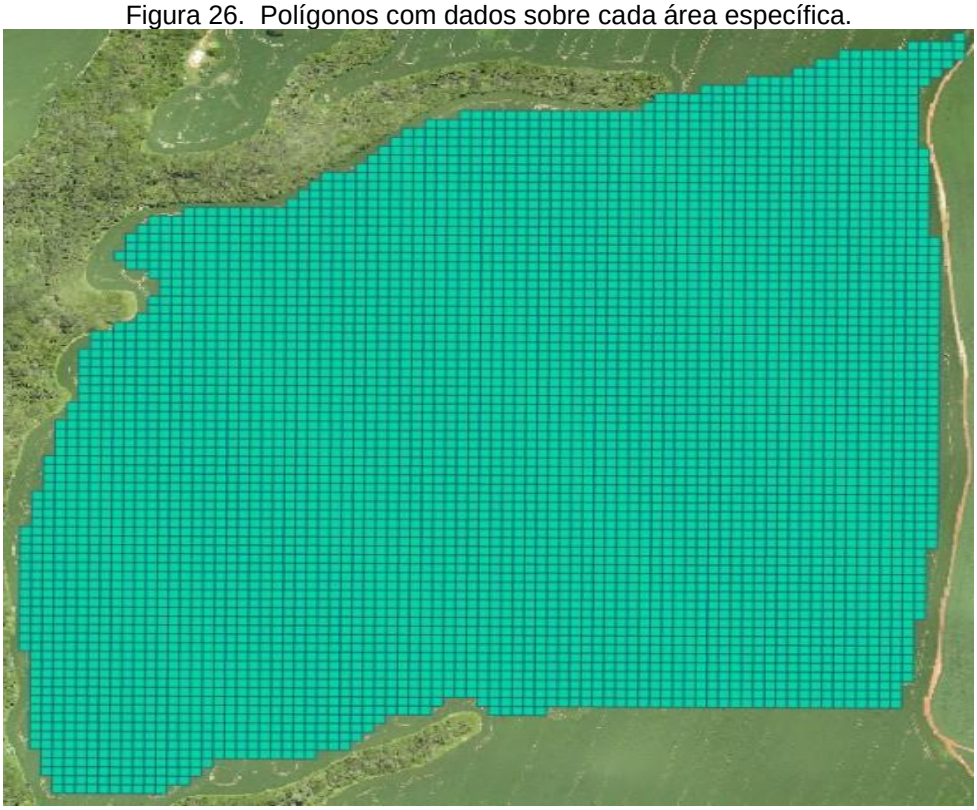


Fonte: Adaptado de VINISKI, 2019.

Os dados de produtividade, a partir dos quais as imagens foram rotuladas, também foram obtidos a partir do trabalho de Viniski (2019), segundo o qual foi utilizado duas colhedoras *John Deere* 9670 STS com monitor de colheita GS2 2630 (fornece informações da produtividade em tempo real) para a colheita da área imageada, em 03/04/2017. O mapa de produtividade foi gerado utilizando o *software GreenStar APEX* 3.7.1.0. A faixa de produtividade variou de 2.5 toneladas por hectare (t/ha) e foi dividida, de acordo com padrões agronômicos, em quatro classes, sendo elas: baixa (<2.5 t/ha), média (2.5 – 4.0 t/ha), alta (4.0 – 6.0 t/ha) e muito Alta (>6.0 t/ha), (Viniski, 2019).

A extração dos dados foi realizada no *software* QGIS versão 2.18.12. Foram gerados pontos regulares das áreas de interesse, sendo que cada ponto estava a uma distância de aproximadamente 7 metros dos vizinhos. Esses pontos regulares foram transformados em conjuntos de polígonos que estão representados na Figura 26. (Viniski, 2019).

Figura 26. Polígonos com dados sobre cada área específica.



Fonte: O autor (2023).

Estando os dados salvos nos pontos amostrais, estes pontos foram utilizados para a extração dos dados de produtividade, presentes nos mapas de produtividade oriundos do computador de bordo da colhedora utilizada. A extração desses dados foi realizada com o *plugin Point Sampling Tools* do *software* QGIS. (Viniski, 2019).

4.1.3 Construção Do *dataset*

Iniciando a analise dos dados notou-se que cada linha representada na Figura 27 corresponde a um dos polígonos georreferenciados da figura 26, com seus dados contidos na camada de polígonos, notou-se então a viabilidade para a construção do *dataset* dando sustentação a formulação do estudo. Duas camadas de polígonos foram utilizadas, uma para cada resolução espacial (10 e 26 cm/*pixel*).

Figura 27. Dados de produtividade, bandas e informações georreferenciados

	X	Y	produtividade	Banda R	Banda G	Banda B	altitude
1	587710.71429	7305400.63012	3.75	6104.08280	7476.99280	5858.04720	955.18823
2	587796.21207	7305904.93361	3.25	5987.75400	7347.82720	5704.39200	956.59320
3	587971.17372	7305516.57064	4.50	6449.08917	7855.62083	6102.50000	965.88226
4	587859.08763	7305828.47049	4.00	6111.54600	7605.58720	5889.97480	960.41876
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
12624	587796.08280	7305884.18773	3.50	5861.03760	7300.04160	5694.13600	958.05603
12625	587684.47127	7305255.56666	3.50	6379.48120	7766.12880	6048.71120	947.55493
12626	588275.76504	7305597.65216	4.25	6672.32960	8289.95040	6395.78640	971.38214
12627	588027.92782	7305474.72270	4.50	6290.19040	7747.49760	6023.30000	966.35602

Fonte: Adaptado de VINISKI, 2019.

A camada de polígonos está toda agrupada, ou seja, embora contenha dados individuais de cada polígono em forma de tabela, não é possível realizar o recorte sem antes separar cada um em uma camada individual.

A ferramenta *Split By Attributes* é um recurso disponível no *software ArcGIS* e foi utilizada para subdividir os polígonos em arquivos de camadas (arquivos *shapefiles*) individuais e por agrupamento de classes. Foi feita a divisão de acordo com a classe de cada polígono. Funciona da seguinte forma:

1. Seleciona-se o conjunto de dados que deseja dividir na ferramenta *Split By Attributes.*

2. Especificam-se quais colunas de atributos que serão usadas para a divisão. Pode-se selecionar a coluna classe e fazer a ordenação dessa coluna por classes de produtividade (alta, média, baixa e muito alta).

3. Define-se a pasta de destino onde os conjuntos de dados serão armazenados.

4. Após configurar estas opções, inicia-se o processo de divisão dos polígonos e juntamente com eles os seus respectivos dados gerando, após algumas horas os arquivos, e também uma tabela de dados individuais de cada polígono com os dados da Figura 28, obtendo-se os polígonos individuais e agrupados por classe de produtividade.

Na sequência, a partir desses arquivos, foi utilizada a ferramenta *Clip Raster by Mask Layer* do *QGIS* para fazer o processo de recorte das imagens, elas precisam estar separadas por classe para que o modelo de IA possa classificar.

Figura 28. Polígonos da classe média para recorte.

Fonte: O autor (2023).

O *ClipRaster by Mask Layer* é uma funcionalidade do *software QGIS* que permite o recorte de imagens em lotes de camadas. Deste modo, pode-se otimizar o processo de geração de imagens, realizando diversos recortes de mesma classe na imagem de uma só vez, fundamentalmente o procedimento é feito da seguinte maneira:

1. Seleciona-se a imagem a ser recortada na ferramenta *Clip Raster by Mask Layer.*

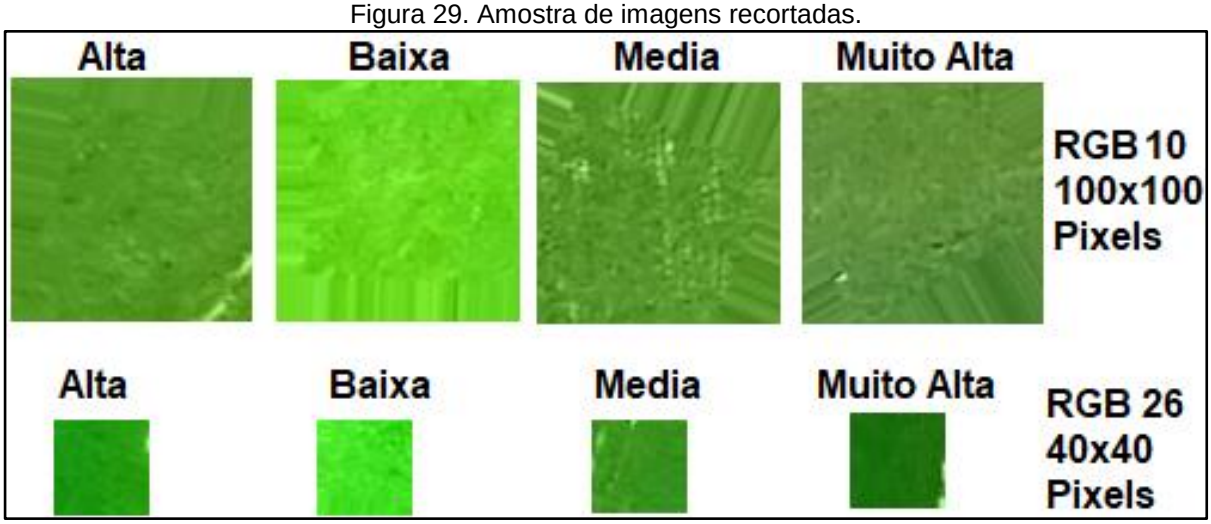
2. Os polígonos escolhidos servem como guia para os recortes. Na Figura 29, cada polígono da classe média é considerado uma camada individual que fica acima da imagem agrícola. Essa camada também é chamada de camada de máscara. É assim, classe a classe, que são feitos os recortes.

3. Define-se a extensão e a resolução do recorte, bem como o formato e a localização do arquivo de saída. Nesse caso optou-se por manter a resolução e o formato *JPEG* original da imagem.

4. Após a definição desses parâmetros, o processo de recorte é iniciado.

É importante observar que a ferramenta preserva as características originais da imagem, garantindo, deste modo, a consistência dos resultados. Assim, utilizando a funcionalidade *Clip Raster by Mask Layer*, foi possível realizar diversos recortes precisos das imagens para cada uma das classes: alta, baixa, média e muito alta.

Figura 29. Amostra de imagens recortadas.



Fonte: O autor (2023).

4.1.4 Balanceamento Das classes

A base de dados gerada apresentou problemas de desbalanceamento das classes como observado no Quadro 3. A classe com menor número de instâncias é a muito alta com 114, e a classe com maior número de instâncias é a alta com 2485, respectivamente.

Para minimizar os efeitos negativos do desbalanceamento de classes foram aplicadas técnicas de *Data Augmentation* em três classes: baixa, média e muito alta. A quantidade de imagens dessas classes foram ampliadas até alcançarem quantidades próxima da classe alta que manteve as 2.485 imagens originais.

A ferramenta *Keras ImageDataGenerator* permite dentro das suas várias finalidades efetuar o procedimento de *Data Augmentation* que se trata de uma classe dentro da biblioteca de aprendizado profundo *Keras*, que permite a implementação de código em *Python* para aplicar e aumentar a quantidade de imagens.

A base de dados final, após a aplicação do *Data Augmentation,* com um total de 9.721 imagens, foi a utilizada ao longo deste estudo.

Quadro 3. Quantidade de Imagens

Classe de Produtividade	**Faixa de Valores de Produtividade (t/ha)**	**Conjunto Original**	***Dataset* oficial após *Data Augmentation***
Baixa	< 2.500	401	2.343
Média	2.501 – 4.000	1.561	2.538
Alta	4.001 – 6.000	2.485	2.485
Muito Alta	>6.000	114	2.355
TOTAL		**4.561**	**9.721**

Fonte: O autor (2023).

4.2 MODELOS PROPOSTOS

Da mesma forma, como não existem estudos envolvendo *RPAs* para classificação de imagens de produtividade da soja, não se tem um protocolo definido quanto à utilização das abordagens de *Meta-Learning* para esta finalidade. O que se tem como base mais próxima são os estudos relacionados no Quadro 3, no entanto, nenhum deles é exatamente igual ao problema aqui proposto.

Visualmente notou-se que a base de dados deste estudo tem similaridade entre as classes, que é quando as imagens de classes diferentes têm forma, cor e textura semelhantes, tornando-se difíceis de serem distinguidas por um humano, nesse aspecto um modelo de IA pode ser mais eficaz que o um humano.

Testar diferentes configurações utilizando o *FSL* torna-se importante para aproximar-se de resultados adequados e fazer comparações entre as estratégias adotadas. Optando pela utilização da técnica *FSL* relacionado à *Meta-Learning* com o algoritmo de otimização *Reptile* a fim de buscar melhores resultados nos modelos de redes neurais *DenseNet121 e ResNet50.*

Também foi selecionado para o estudo as redes neurais siamesas e redes de trigêmeos (*Triplet Networks)* por se tratarem de redes criadas para explorar imagens com similaridade e também para explorar o subcampo chamado de aprendizagem métrica dentro do conceito geral do *Meta-Learning*.

Dessa forma, elas se mostram uma escolha promissora para a tarefa de classificação de imagens de produtividade da soja.

4.2.1 Arquitetura Geral Dos Modelos Propostos

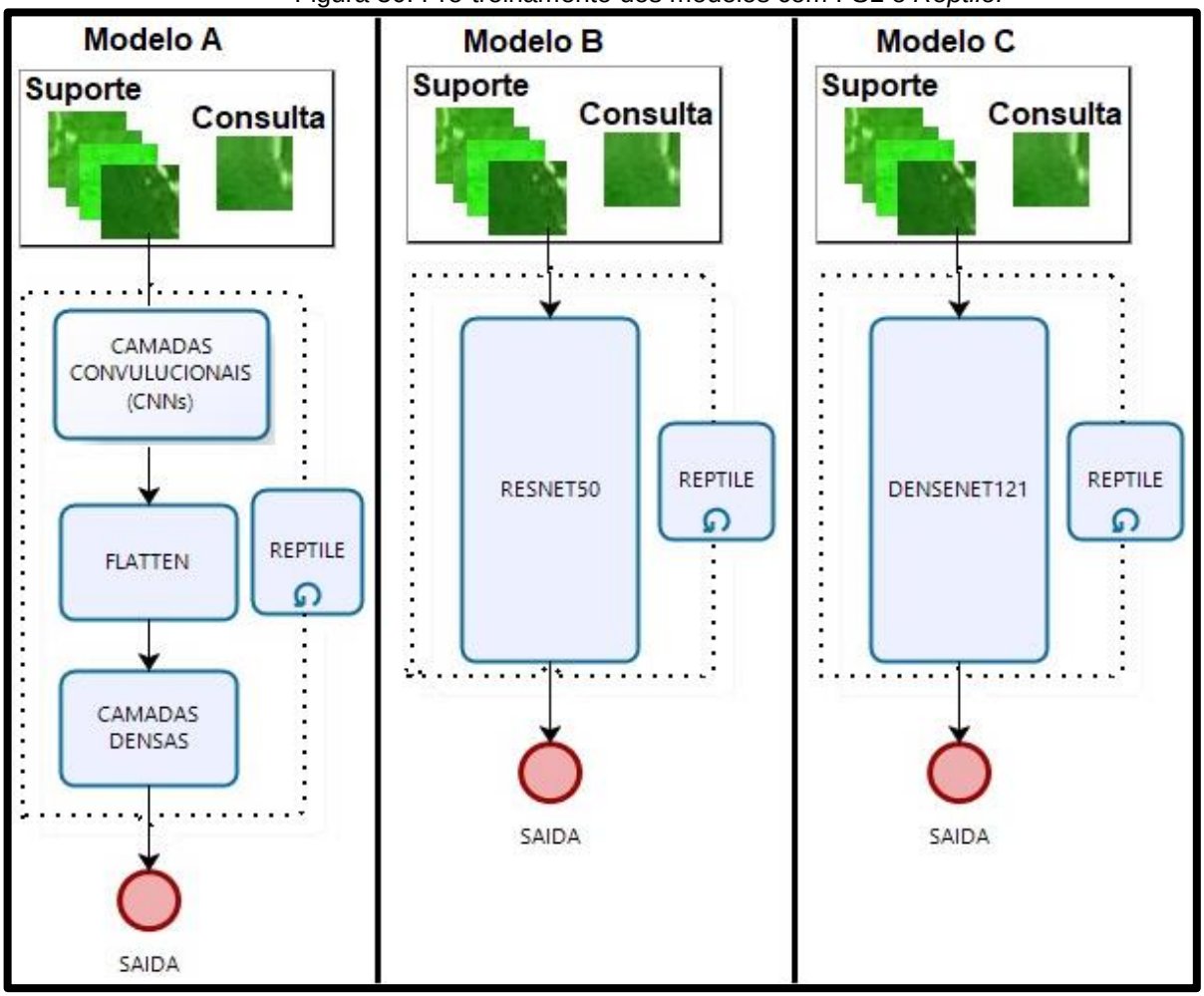
A codificação dos modelos foi feita na linguagem de programação *Python,* entre algumas bibliotecas utilizadas na construção dos modelos as principais foram: *TensorFlow, Teras, Scikit-Learn, OpenCV, Numpy, Seaborn, Pandas e Matplotlib.* Essas bibliotecas forneceram módulos que permitiram a construção e treinamento dos modelos com funções pré-definidas como: *Convolução 2D, Max Pooling, Batch Normalization, Relu, ELU*, *Dense, Softmax*.

Os três modelos (A, B e C) da Figura 30 foram programados para operar em uma configuração de *FSL,* com *K-shot, N-way* e *Q*.

O conjunto *Q* é a consulta, ele foi usado para testar a eficácia dos modelos com base em imagens que não foram previstas durante o treinamento.

Os modelos na Figura 30, foram treinados para classificar as imagens nas categorias: alta, baixa, média e muito alta.

Figura 30. Pré-treinamento dos modelos com FSL e *Reptile.*



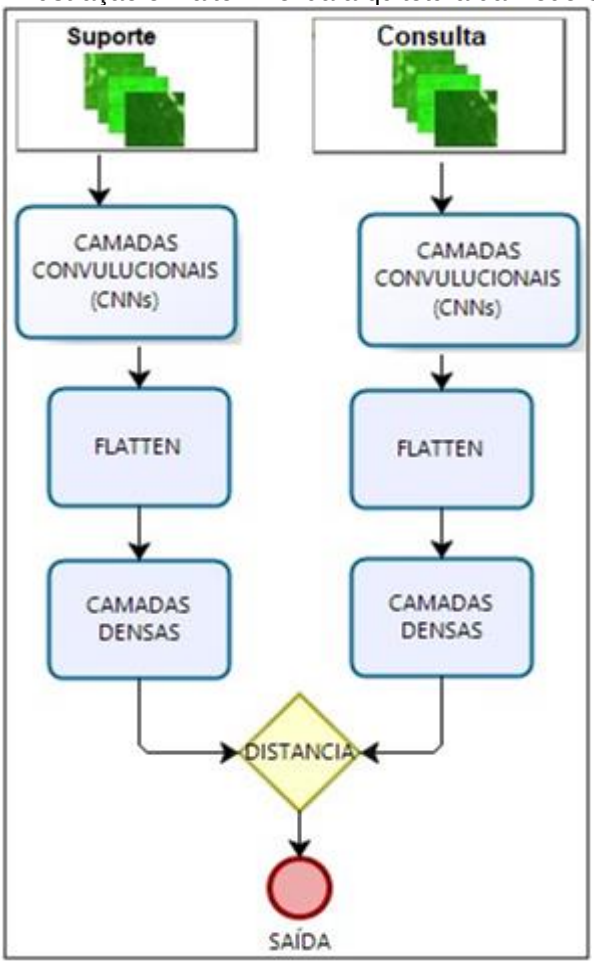
Fonte: O autor (2023).

A Rede Siamesa na Figura 31 e a Rede Siamesa *Triplet* da Figura 32 foram ambas construídas para utilizar em cada uma de suas ramificações a mesma arquitetura do modelo A.

Nessas Redes Siamesas para calcular a semelhança entre as imagens utilizou-se a distância euclidiana.

O cálculo de distância é essencial para determinar quão similar ou dissimilar as imagens são, aqui a distância Euclidiana tem um papel fundamental depois que as imagens passam pelas ramificações das redes e seus componentes onde cada imagem é processada separadamente, são gerados no final os vetores de características dessas imagens. A distância Euclidiana é essencialmente a linha mais curta entre dois pontos em um espaço de N dimensões (onde N é o número de dimensões em seu vetor de características). Nesse caso, ela figura como um indicador da similaridade entre duas imagens: uma distância menor indica que as imagens são muito similares, enquanto um valor maior indica uma baixa similaridade.

Figura 31. Ilustração em alto nível da arquitetura da Rede Siamesa.

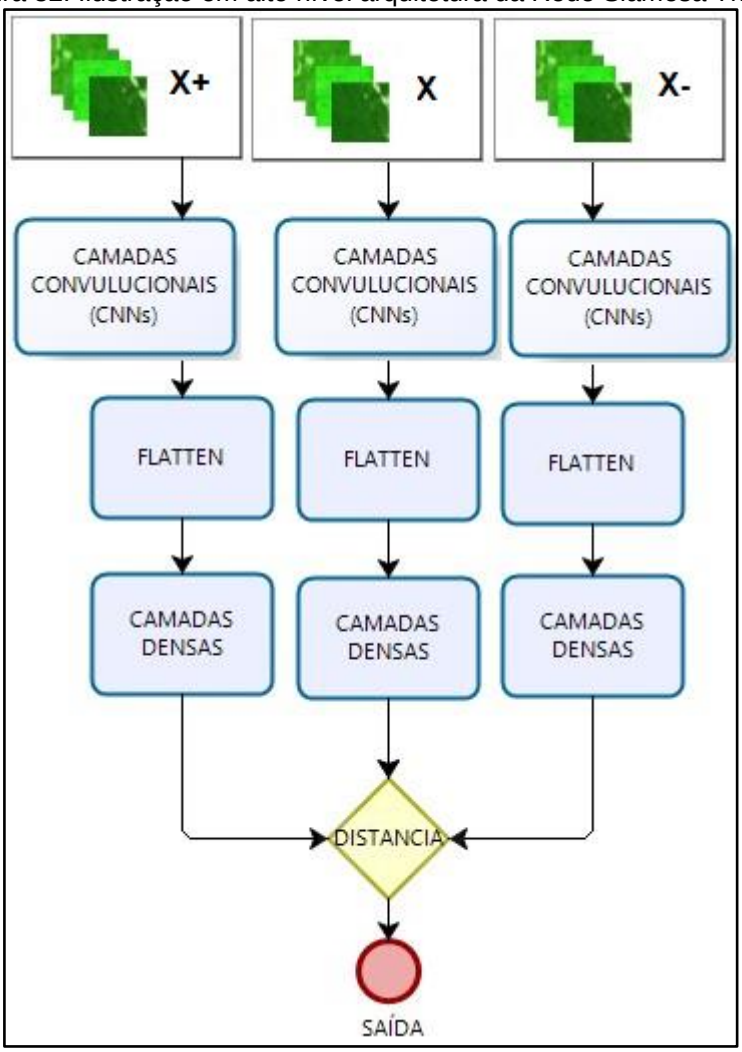


Fonte: O autor (2023).

Segundo Koch *et al*., (2015) os pares de dados na estrutura padrão *N-way, K-shot* de uma rede siamesa podem ser vistos como exemplos de suporte e consulta, respectivamente. Assim, a tarefa da Rede Siamesa em comparar cada par se torna uma questão de aprendizado com poucas amostras *FSL.*

Triplet Networks são uma extensão do conceito das Redes Siamesas alinhado a sua utilização em poucos exemplos as tornam uma opção de *FSL.*

Figura 32. Ilustração em alto nível arquitetura da Rede Siamesa Tripla.



Fonte: O autor (2023).

Após a extração de características por meio de cada ramificação, o modelo calcula a distância Euclidiana entre as saídas dos três ramos para minimizar a distância entre a imagem (X) e a imagem positiva (X+) e maximizar a distância entre a imagem (X) e a imagem negativa (X-).

Esses resultados são então utilizados em um terceiro cálculo, a função de perda de *triplet*, que busca garantir que a imagem (X) esteja mais próxima da imagem positiva do que da negativa. Portanto, apesar de envolver três

componentes, a função de perda resulta em um único valor que orienta o ajuste da rede durante o treinamento, permitindo que ela aprenda a aproximar imagens da mesma classe e afaste imagens de classes distintas.

O conceito de aprendizagem por inicialização do *Meta-Learning* foi utilizado, nesse cenário foi aplicado o algoritmo *Reptile* nos modelos R*esNet*, *DenseNet* e o modelo utilizado por Kinli (2018) com algumas modificações na arquitetura do modelo buscando melhorar os seus resultados em acurácia. São escolhas estratégicas, pois a arquitetura da *ResNet* e *DenseNet* tem capacidade para lidar com imagens complexas.

Utilizar os mesmos modelos, na mesma base de dados, sem a parte que compete ao *FSL* e *Meta-Learning* é válido para constatar se as metodologias aplicadas contribuíram ou não para melhores resultados.

4.2.2 Otimizações Nos Modelos A B e C

O modelo (A) apresentado na Figura 30 tem a sua a arquitetura baseada em Kinli (2018) que apresenta-se da seguinte forma: 8 Camadas convolucionais, camada de *flatten*, 4 camadas densas e a função de ativação utilizada é a *ReLu.*

Essa estrutura passou por uma reconstrução e após as modificações pertinentes cria-se o modelo (A) que utilizado nesse estudo, com a sua estrutura passando a se apresentar da seguinte forma: 6 Camadas convolucionais, camada de *flatten,* 3 camadas densas e a função de ativação foi alterada para a função de ativação *ELU*.

O modelo em questão passou a ser chamado de *CNN* modificada, cada conjunto de camadas convolucionais inclui camadas *Conv2D*, ativações *ELU*, normalização em lote (B*atch Normalization*), camadas de *MaxPooling2D* e por fim a camada de *Dropout* para evitar o *Overfitting.*

As saídas do *Flatten* passam por camadas densas (*Fully Connected).* Cada camada densa é seguida por ativação *ELU*, normalização em lote e *Dropout*. Isso ajuda a rede a evitar o *Overfitting*.

A função de ativação foi alterada de *ReLu* para *ELU* a fim de evitar um problema de gradientes pequenos em *'ReLu'* que pode ter uma média de saída mais próxima de 0 e nos testes iniciais a *ELU* se saiu melhor frente a *ReLu.*

As alterações foram feitas para reduzir a carga computacional e aumentar a capacidade preditiva do modelo. A redução de duas camadas convolucionais contribuiu para um modelo mais leve e mais rápido.

A inclusão da camada *Flatten* facilitou a conexão entre as camadas convolucionais e as camadas densas. A redução de uma camada densa também ajudou a prevenir o *overfitting.*

Com essas alterações o modelo A (*CNN* Modificada) mostrou-se otimizado tanto em desempenho computacional quanto em performance preditiva.

Foi implementada a técnica de *File-Tuning* nos modelos A, B e C que são previamente treinados com o algoritmo de *FSL Reptile* por 200 épocas para a otimização dos pesos iniciais.

A otimização dos hiperparâmetros é feita através do G*rid Search* a partir do modelo anteriormente pré-treinado. Este método busca a melhor combinação de hiperparâmetros através da realização de um treinamento para cada combinação, retornando aquela que proporciona o melhor desempenho.

A biblioteca utilizada *Skicit-learn* possui métodos para implementar o *Grid Search*. O método para calcular a precisão, cria modelos utilizando diferentes combinações de parâmetros (Kramer, 2016).

Os hiperparâmetros avaliados neste caso foram:

- Taxa de aprendizado (*Learning Rate*): um multiplicador que determina o quanto os pesos da rede são alterados em cada iteração de treinamento. Foi testado com os intervalos de 0,1, 0,01, 0,001 e 0,0001.

- Tamanho do lote (*Batch Size)*: é o número de exemplos de treinamento utilizados em uma única etapa. Foi testado com os intervalos de 32, 64, 128 e 256.

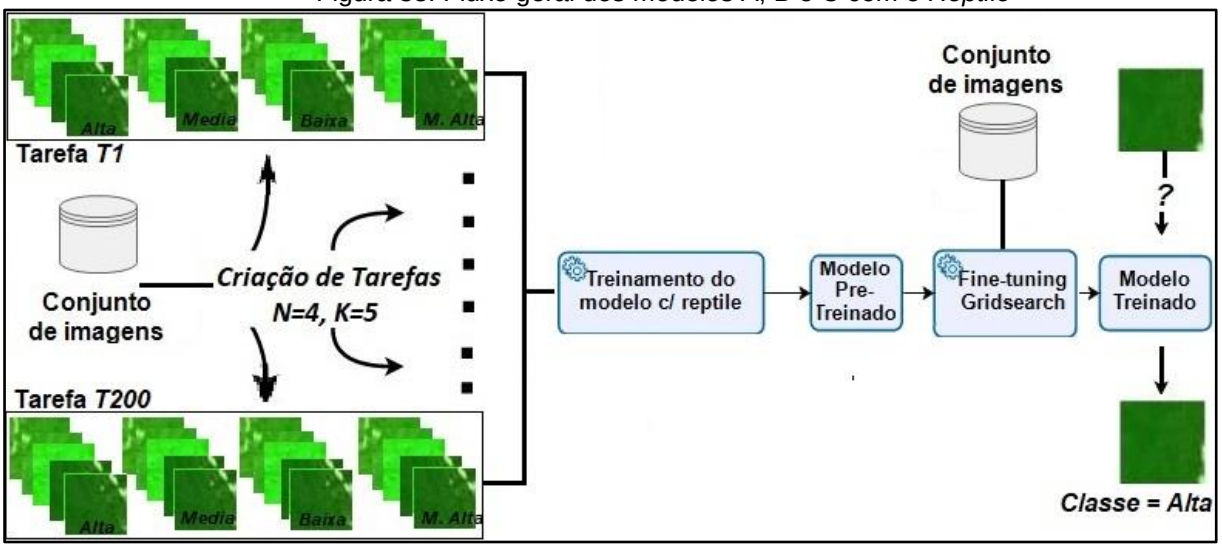
- Número de épocas (*Epochs*): representa as vezes que o algoritmo de otimização irá percorrer todo o conjunto de dados de treinamento. Foi testado com 50, 100, 200 e 400 épocas.

4.2.3 Treinamento Dos Modelos Propostos

A utilização nos modelos A, B e C com o *Reptile* é feita para um pré-treinamento em várias tarefas seguindo o conceito do *FSL*. O modelo base é então criado e seus pesos iniciais são otimizados pelo *Reptile* a cada uma das 200 épocas de treinamento de modo a minimizar a perda em cada tarefa.

Para treinar o modelo inicial que também pode ser chamado de meta-modelo ou modelo pré-treinado foram criadas 200 tarefas do tipo 4-*way*, *N-shot* (com variações de valores para N) a partir do conjunto de dados. O modelo pré-treinado com o *Reptile* até aqui adquiriu o conhecimento prévio das tarefas e seguiu para a etapa de *Fine-Tuning* onde esse modelo pré-treinado é invocado utilizando o mesmo conjunto de imagens e o *Grid Search* percorre a busca para a melhor definição *dos* hiperparâmetros até criar o modelo treinado final conforme imagem de fluxo geral na figura 33.

Figura 33. Fluxo geral dos modelos A, B e C com o *Reptile*



Fonte: O autor (2023).

O treinamento e avaliação do modelo final é feito produzindo métricas e gráficos de desempenho. O modelo final treinado é salvo novamente para uso futuro.

As Redes Siamesas não foram executadas utilizando o *Grid Search* para evitar custo computacional e também porque as ramificações da Rede Siamesa são as mesmas da arquitetura do modelo (A). Considerando esses aspectos, foram aproveitadas as definições de taxa de aprendizado e tamanho do lote do modelo (A), com exceção para o número de épocas que foi definido para rodar com 500 épocas.

Entretanto, foi definida uma função conhecida como "*Early Stopper*" de valor 10. Isso significa que após 10 épocas sem haver melhoria no desempenho mostrando estagnação na validação, o treinamento será interrompido para evitar o *overfitting*.

Embora o G*rid Search* seja uma excelente ferramenta para otimização, tem um custo computacional consideravelmente alto.

Os modelos A, B e C foram executados com variações dos números de *K-shot* (imagens por classe), para aferir o impacto dos valores nos modelos. Essa é uma das estratégias comumente usada em *FSL* para alcançar valores ideais de K em relação à acurácia. Optou-se por usar variações de 1, 5, 10 e 15 embora a grande maioria dos estudos envolvendo *FSL* utilizem os intervalos 1, 5 e 10, todavia não existem regras para tais valores, nota-se apenas que se trata de um padrão nos estudos pertinentes ao *FSL*. Notou-se o aumento do custo computacional quando se aumenta o número de *K-shots.*

Em relação ao *Dataset,* utilizado para todos os modelos com 9.721 imagens, foram organizadas em classes através de quatro pastas, uma para cada classe. Foi destinado 7.776 contabilizando 80% para a aprendizagem e 1.945 cerca de 20 % para a validação.

As imagens dentro do valor de 20% (1.945) são selecionadas automaticamente e aleatoriamente pelos modelos sem a utilização da estratificação, portanto a quantidade individual de imagens que cada classe utiliza para a validação poderá conter variação em torno de 3% para mais ou para menos.

4.3 PROCESSAMENTO

O processamento foi realizado usando três recursos em paralelo, dada a carga computacional: o LCAD (Laboratório de Computação de Alto Desempenho) vinculado ao Programa de Pós-Graduação em Computação Aplicada da UEPG, *Google Cloud* é um *Desktop.*

Para este estudo, o LCAD disponibilizou três servidores nos quais foram processados os modelos com o G*rid Search*, a fim de otimizar todos os modelos utilizando dois conjuntos de imagens *RGB* 10 e 26. Após esta fase, as imagens *RGB* 10, (maiores e que exigem mais recursos computacionais) foram processadas no mesmo ambiente. Já as imagens *RGB 26* foram processadas em paralelo no *google cloud* e no *desktop*.

As máquinas virtuais do *Google Cloud* são recursos computacionais personalizáveis e escalonáveis na nuvem, que proporcionam a flexibilidade de virtualizar servidores físicos para a execução de aplicações e cargas de trabalho dedicadas.

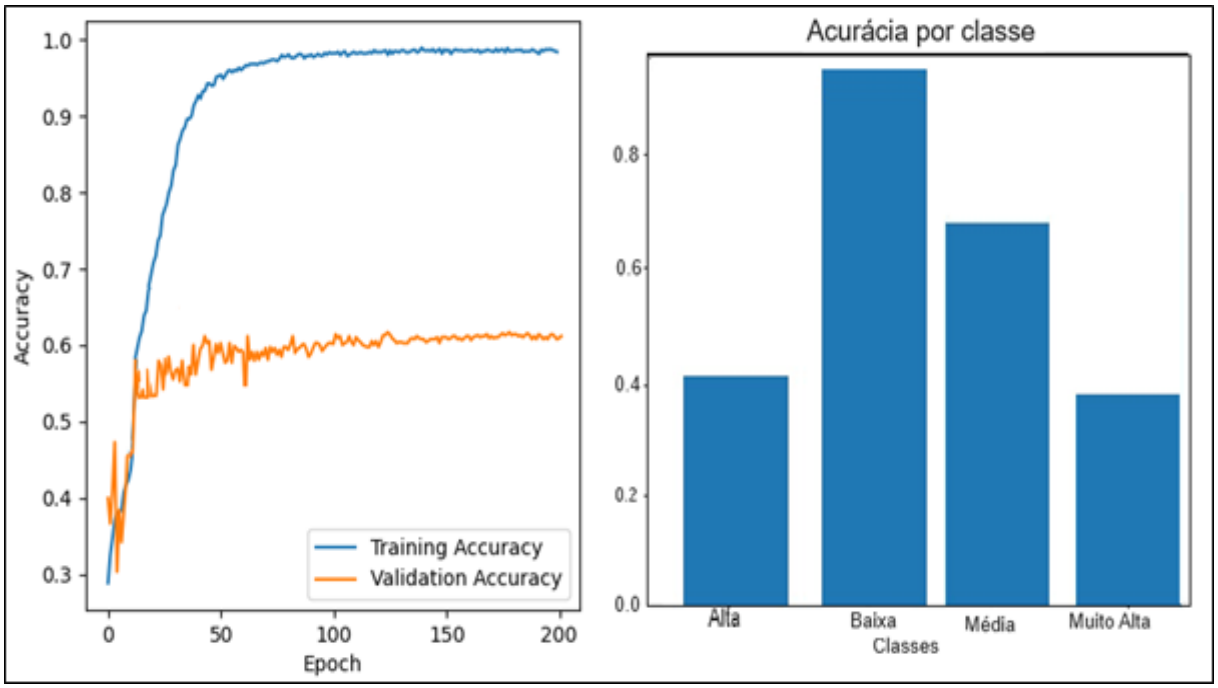
5 RESULTADOS

Os resultados obtidos com a aplicação do *FSL* para a classificação de imagens de produtividade da soja, obtidas por meio de *RPA* são apresentados considerando a acurácia de validação de cada modelo.

5.1 MODELO (A): *CNN* MODIFICADA SEM *FSL*/*REPTILE* E COM *FSL* / *REPTILE*

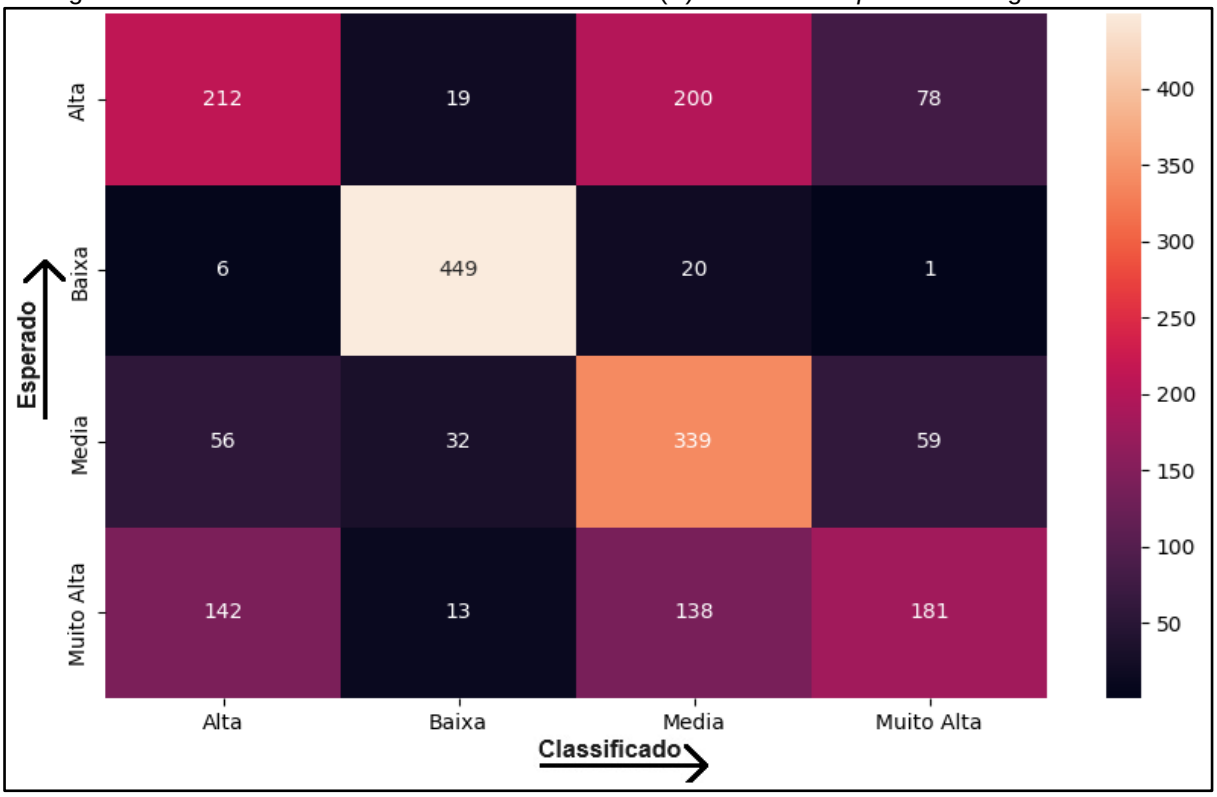
Inicialmente foi aplicado o modelo de aprendizado *CNN* modificada para o conjunto de imagens *RGB* 10 atingindo uma acurácia de 60.9 (Figura 34) e *RGB* 26, atingindo uma acurácia de 62,3 (Figura 36), conforme pode-se ver em cada matriz de confusão respectivamente *RGB* 10 (Figura 35) e *RGB* 26 (Figura 37).

Figura 34. Acurácia = 60.9 no modelo (A): *sem FSL/Reptile* nas imagens *RGB* 10



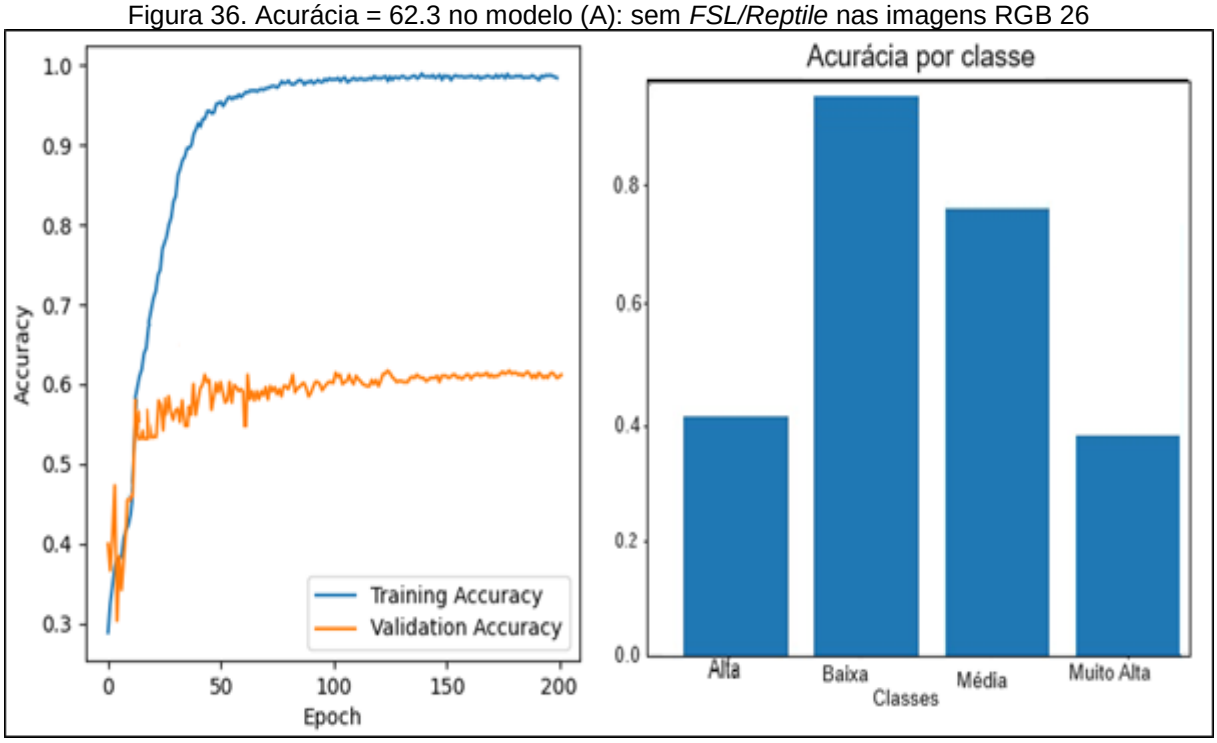
Fonte: O autor (2023).

Figura 35. Matriz de Confusão referente ao modelo (A): sem *FSL/Reptile* nas imagens RGB 10



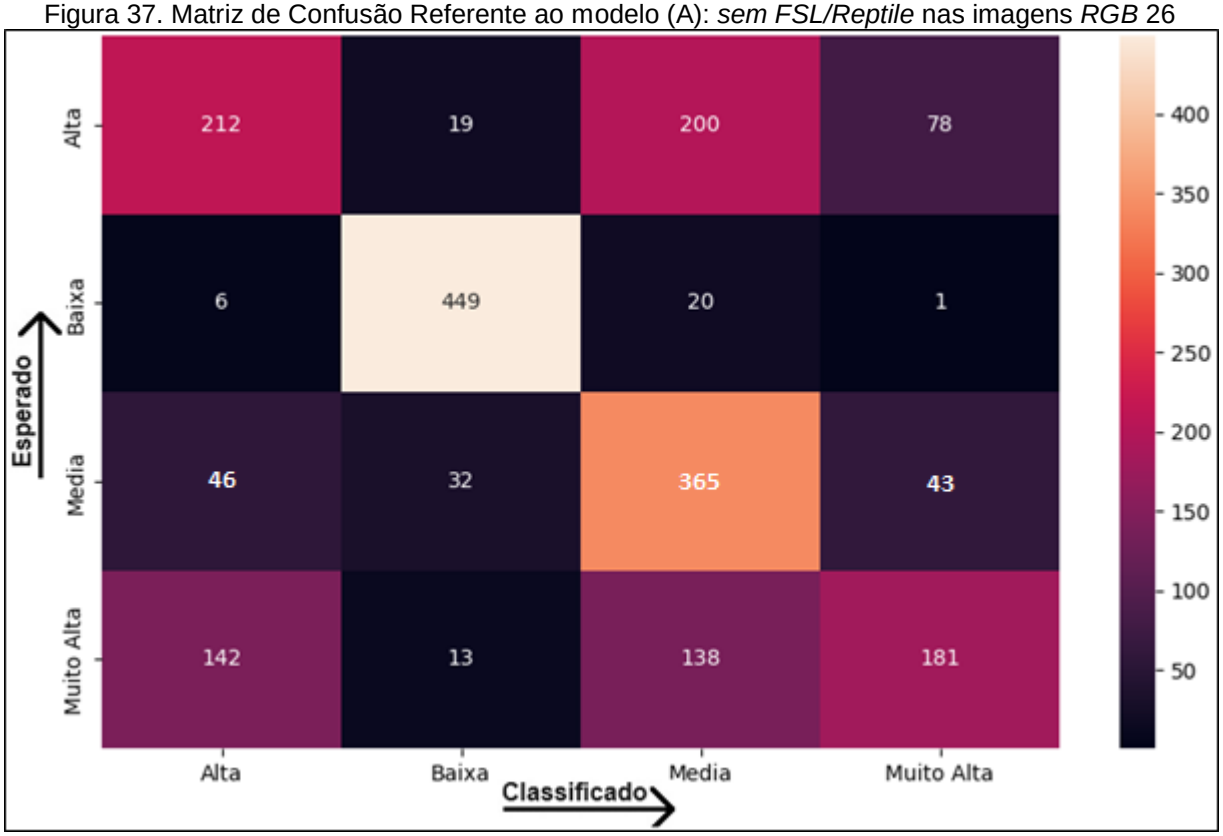
Fonte: O autor (2023).

Figura 36. Acurácia = 62.3 no modelo (A): sem *FSL/Reptile* nas imagens RGB 26



Fonte: O autor (2023).

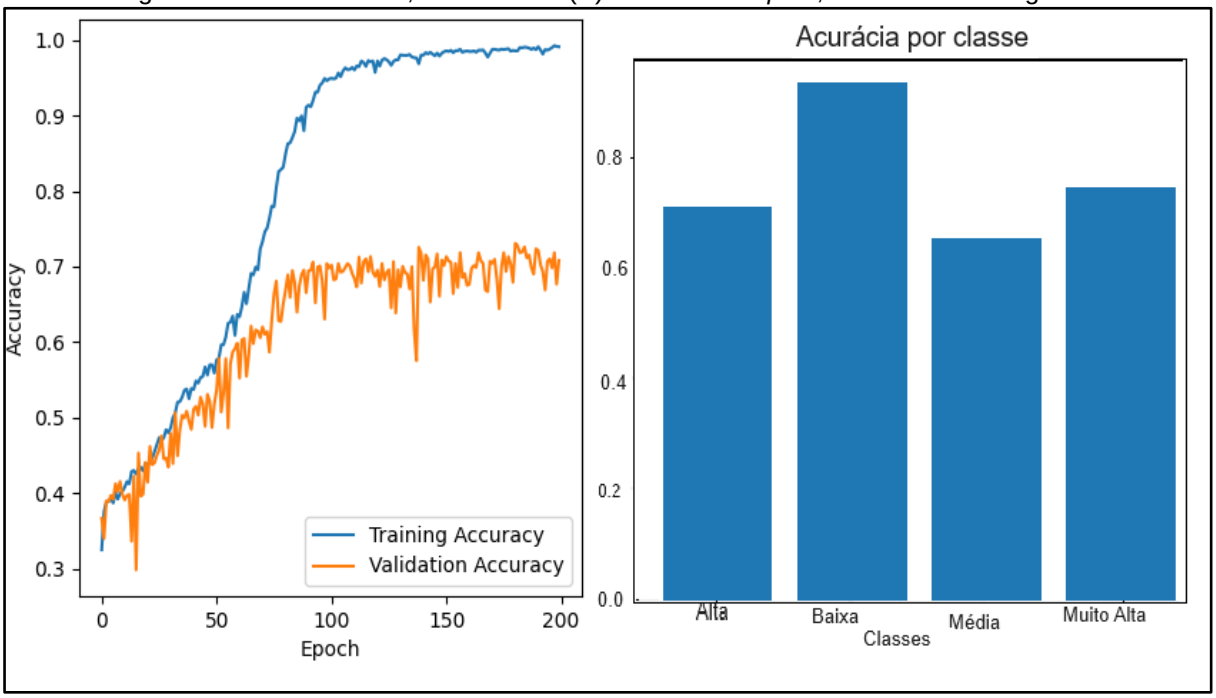
Figura 37. Matriz de Confusão Referente ao modelo (A): *sem FSL/Reptile* nas imagens *RGB* 26



Fonte: O autor (2023).

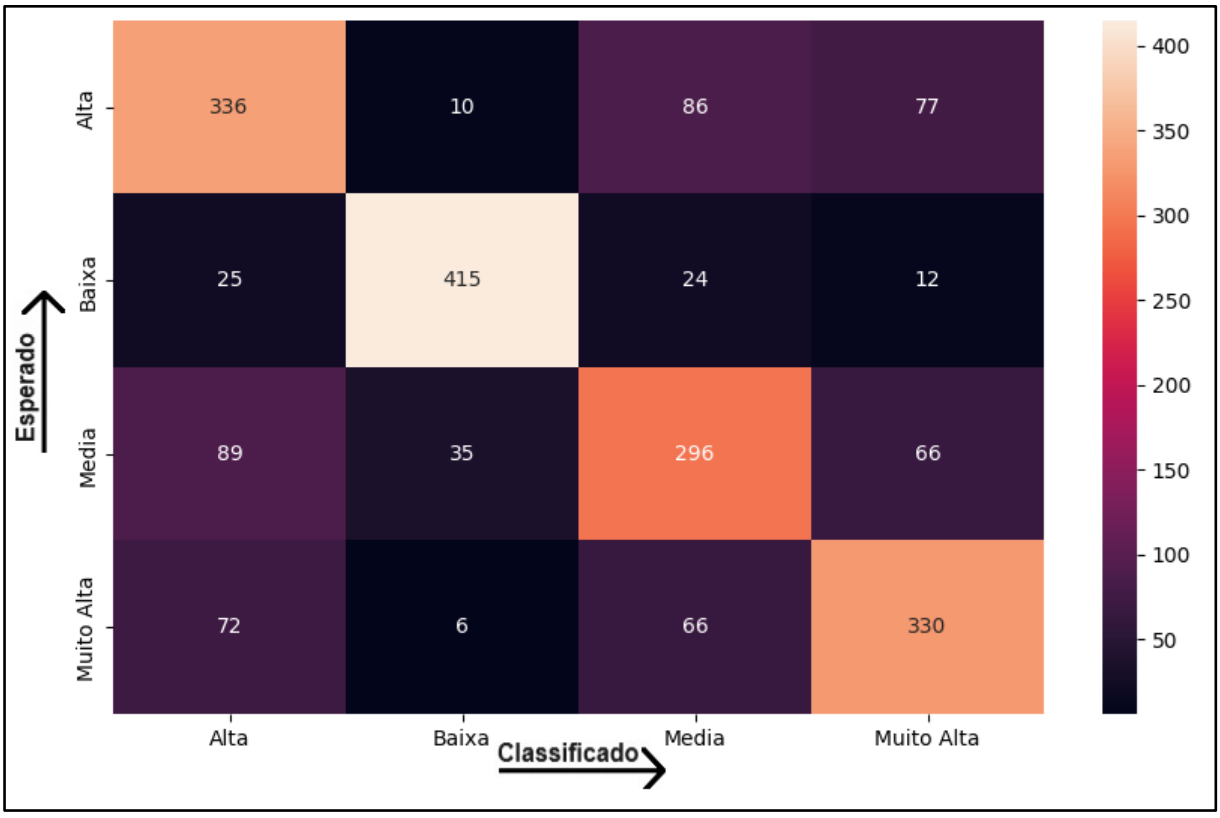
Ao utilizar a estratégia de aprendizado da *CNN* modificada com *FSL/Reptile* configurado para *Shot* = 1, com imagens na resolução 10 cm/*px*, foi atingida uma acurácia de 70,9 (Figura 38), o que pode ser conferido na matriz de confusão (Figura 39). Ao aplicar o mesmo Modelo A em imagens com resolução de 26 cm/*px* a acurácia foi elevada a 79,3 (Figura 40), assim como nos mostra a matriz de confusão (Figura 41), conclui-se assim que a classe melhor predita é a classe de produtividade baixa.

Figura 38. Acurácia = 70,9 no modelo (A): com *FSL/Reptile*, Shot =1 nas imagens *RGB* 10.



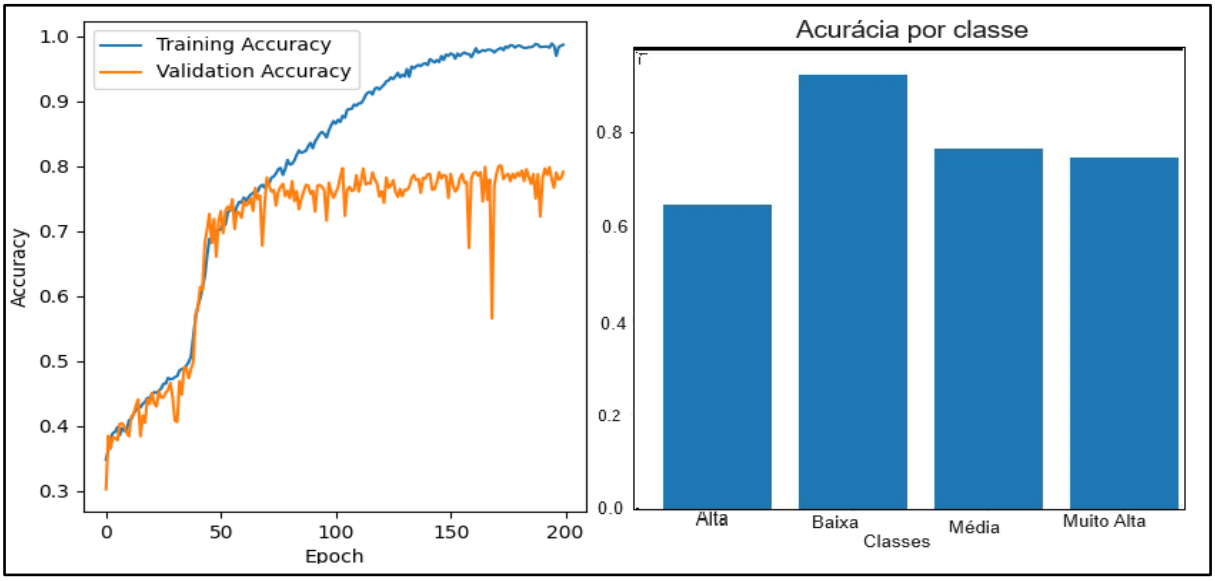
Fonte: O autor (2023).

Figura 39. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile, Shot*=1 nas imagens *RGB* 10



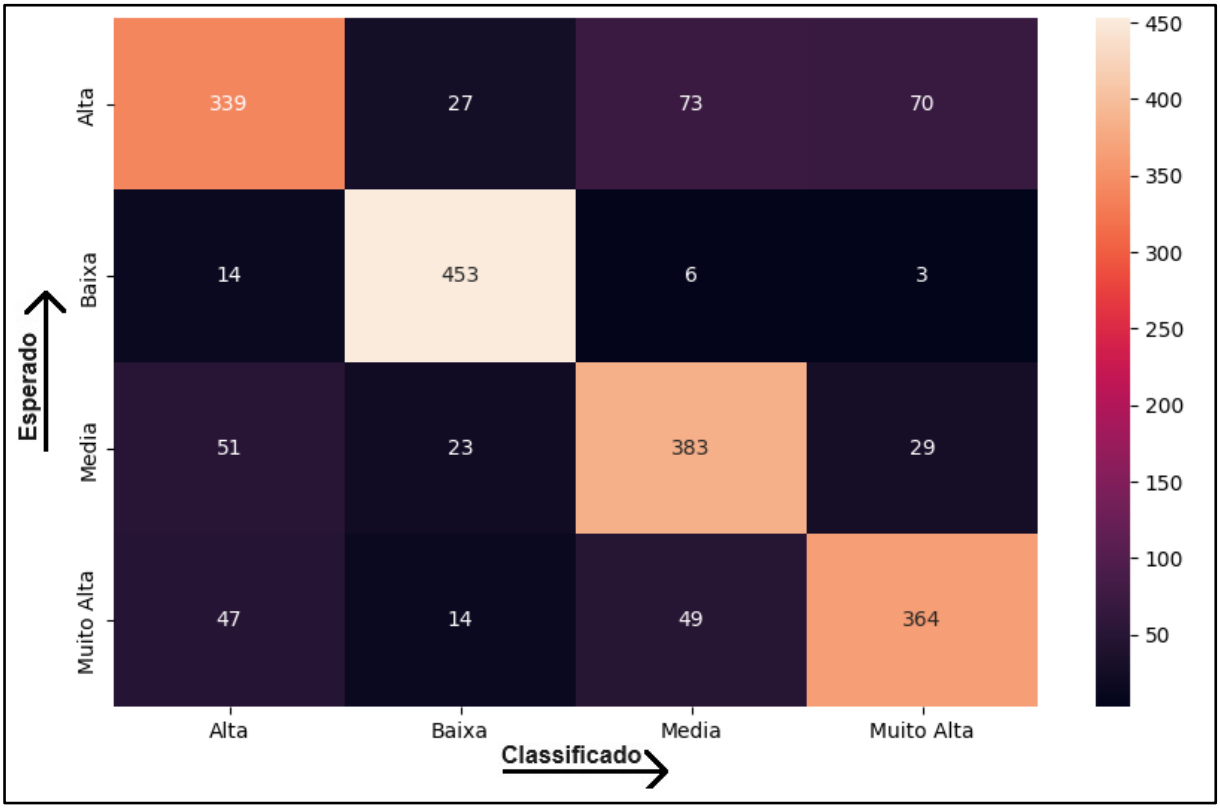
Fonte: O autor (2023).

Figura 40. Acurácia = 79.3 no modelo (A): com *FSL/Reptile*, *Shot* =1. nas imagens *RGB* 26



Fonte: O autor (2023).

Figura 41. Matriz de Confusão Referente. ao modelo (A): com *FSL/Reptile*, *Shot*=1.nas imagens *RGB* 26

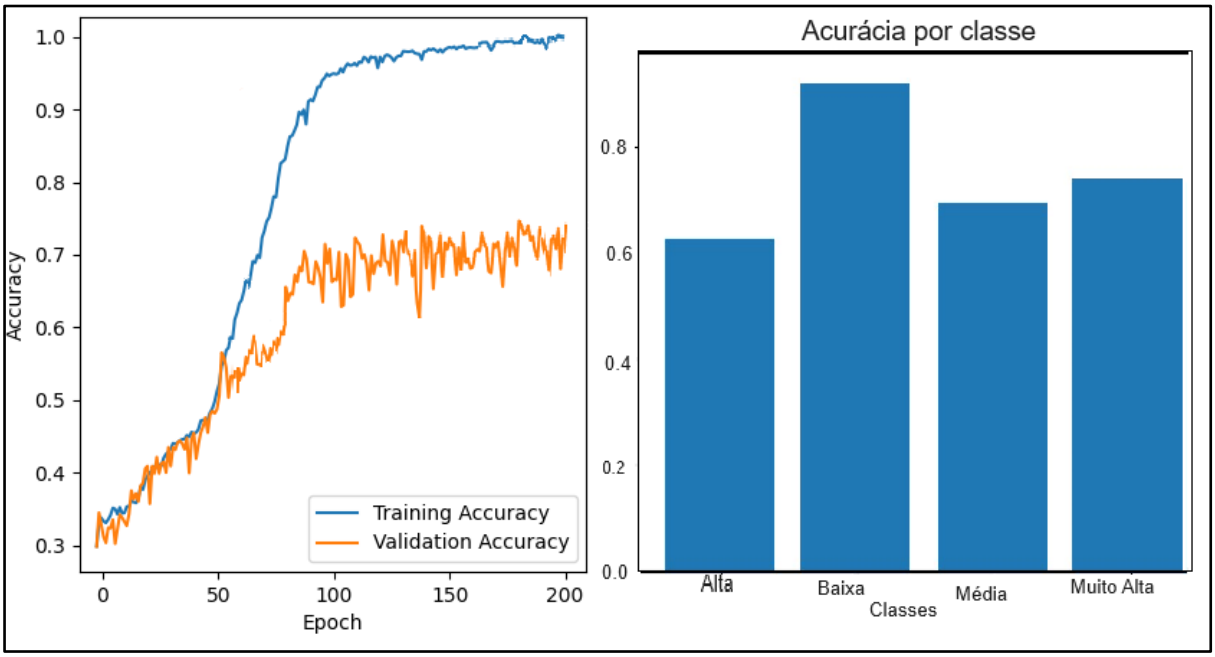


Fonte: O autor (2023).

Os resultados obtidos ao variar o número de *shots* para 5 para o Modelo de Aprendizado A, tanto para imagens *RGB* 10 (Figura 42) com a sua matriz de confusão representada pela (Figura 43) quanto a *RGB* 26 (Figura 44) representada pela sua matriz de confusão (Figura 45) mostraram um melhor desempenho apenas

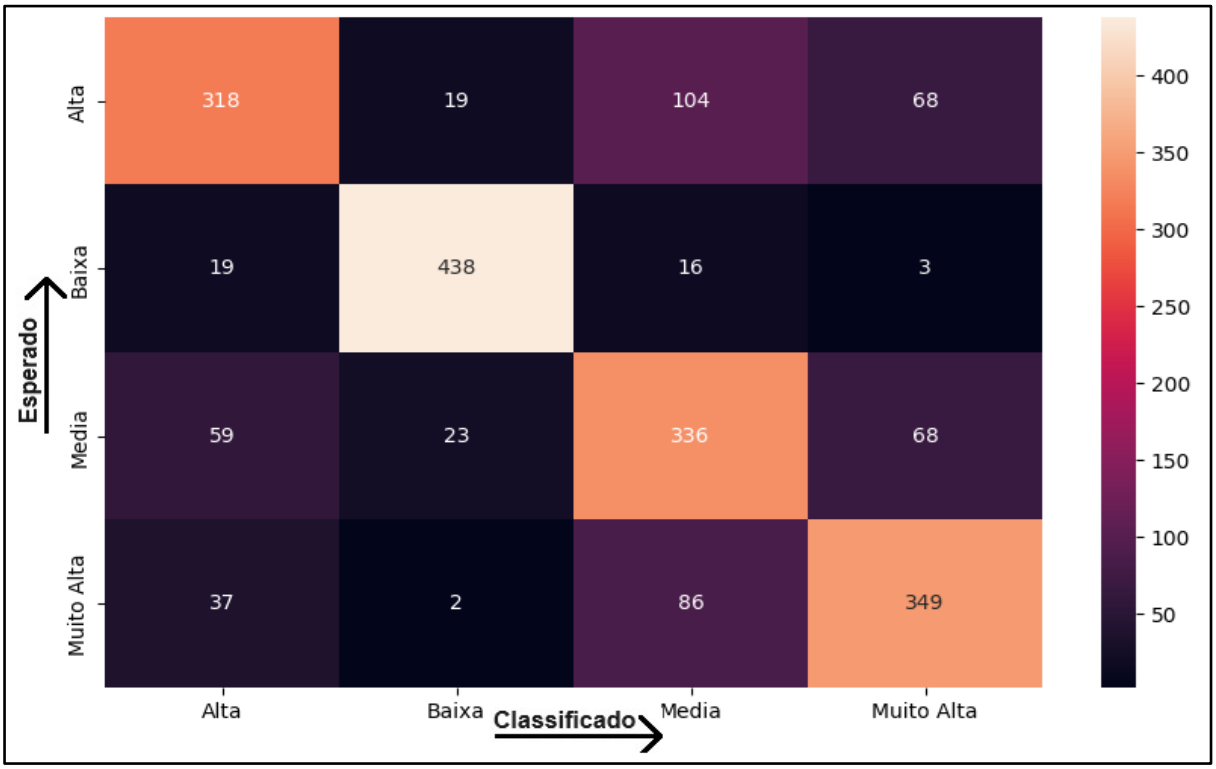
para imagens *RGB* 10, porém, ainda inferior aos resultados obtidos até aqui com imagens *RGB* 26.

Figura 42. Acurácia = 74.3 no modelo (A): com *FSL/Reptile*,*Shot*=5 nas imagens *RGB* 10.



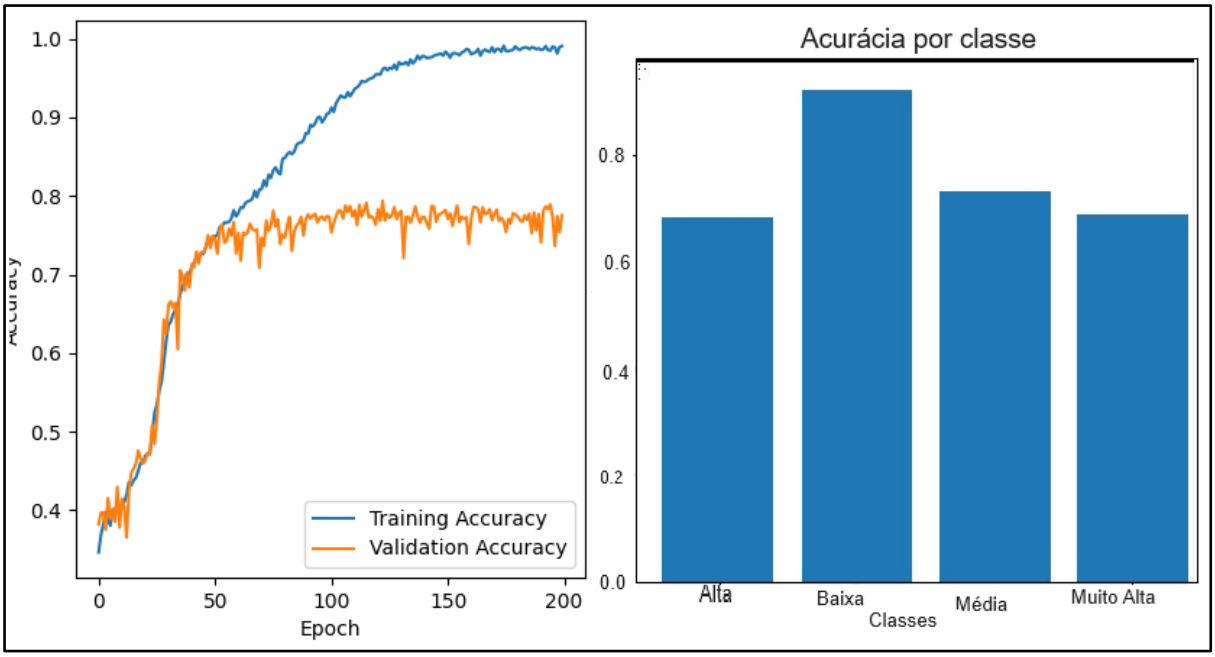
Fonte: O autor (2023).

Figura 43. Matriz de Confusão referente ao modelo (A): com *FSL/Reptile*, *Shot*=5 nas imagens *RGB* 10



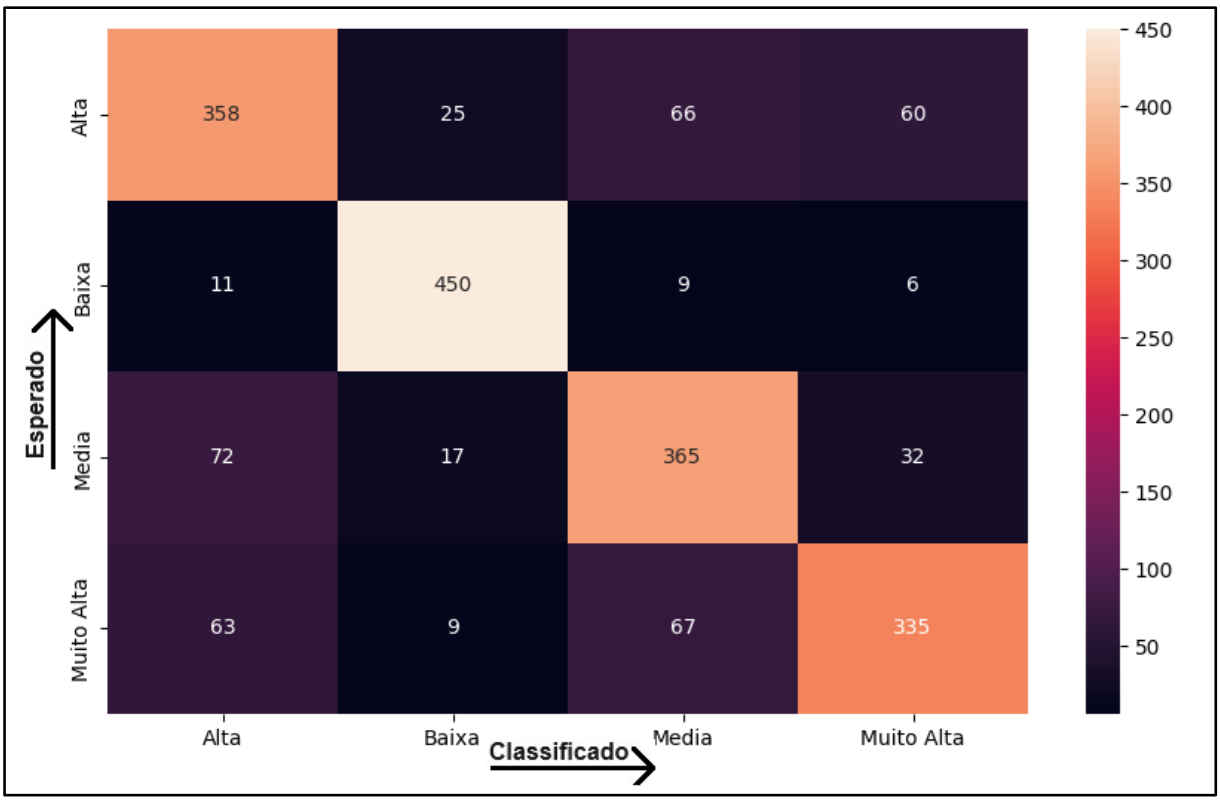
Fonte: O autor (2023).

Figura 44. Acurácia = 77.6 no modelo (A): com *FSL/Reptile*, *Shot*=5 nas imagens *RGB* 26



Fonte: O autor (2023).

Figura 45. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile, Shot*=5 nas imagens *RGB*26



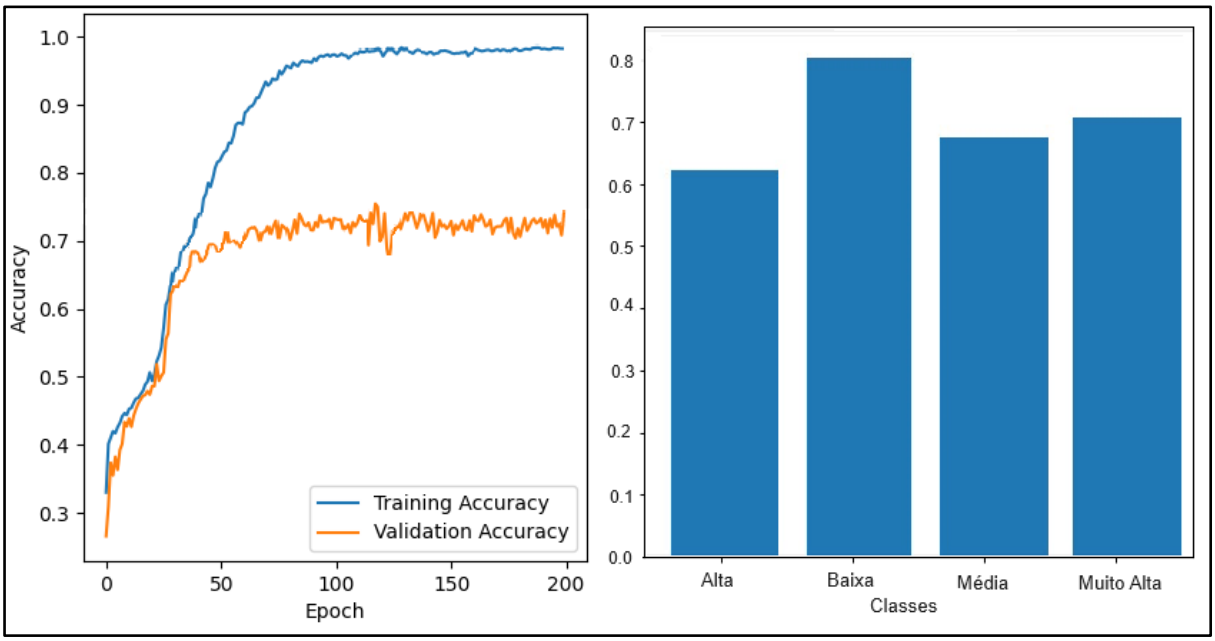
Fonte: O autor (2023).

Optou-se por elevar o número de *shots* para 10 e depois para 15, para verificar o comportamento dessa variável no desempenho do modelo gerado. Utilizando 10 *shots* a para o conjunto de imagens *RGB* 10 o desempenho piorou (Figura 46) conforme matriz de confusão (Figura 47) enquanto que houve uma

melhora de acurácia para as imagens *RGB* 26, atingindo o valor de 80.3 (Figura 48). Podendo ser confirmado através da sua matriz de confusão (Figura 49). O uso de 15 *shots* novamente piorou o desempenho com as imagens *RGB* 10 (Figura 50) e matriz de confusão respectivamente na (Figura 51).

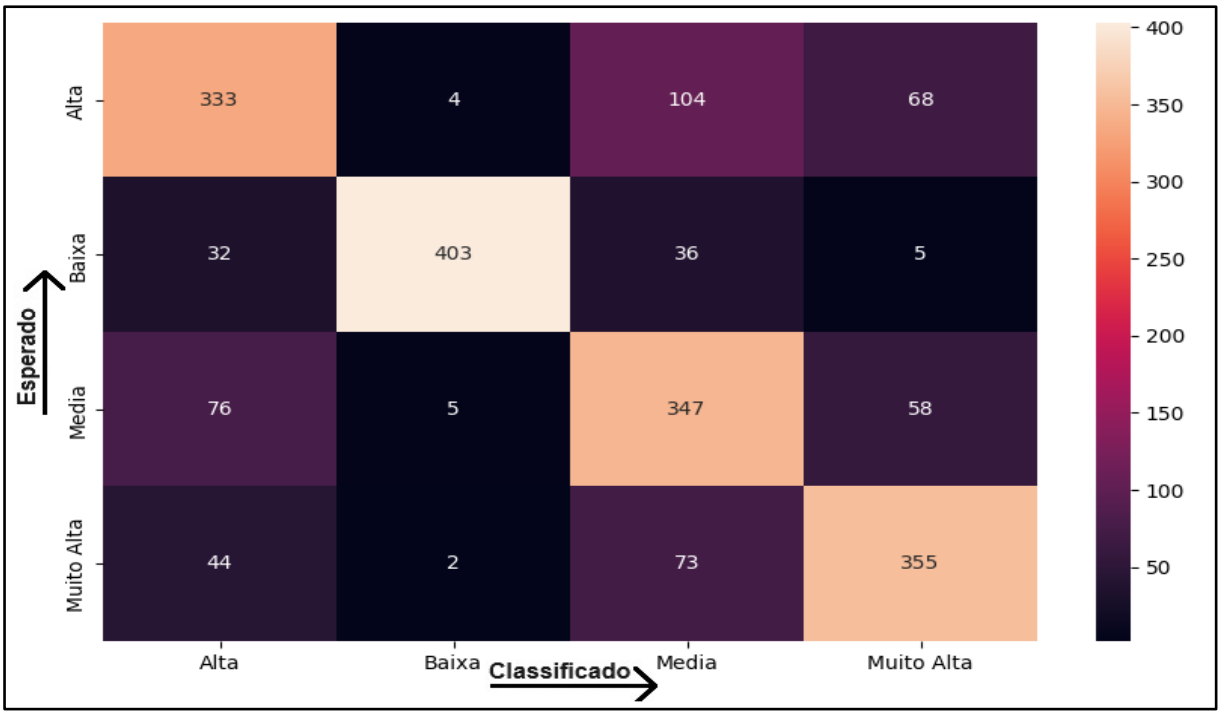
Manteve a mesma faixa de acurácia para as imagens *RGB* 26 (Figura 52), seguido por sua matriz de confusão respectivamente (Figura 53).

Figura 46. Acurácia = 74.0 no modelo (A): Com *FSL/Reptile*, *Shot*=10 nas imagens *RGB* 10



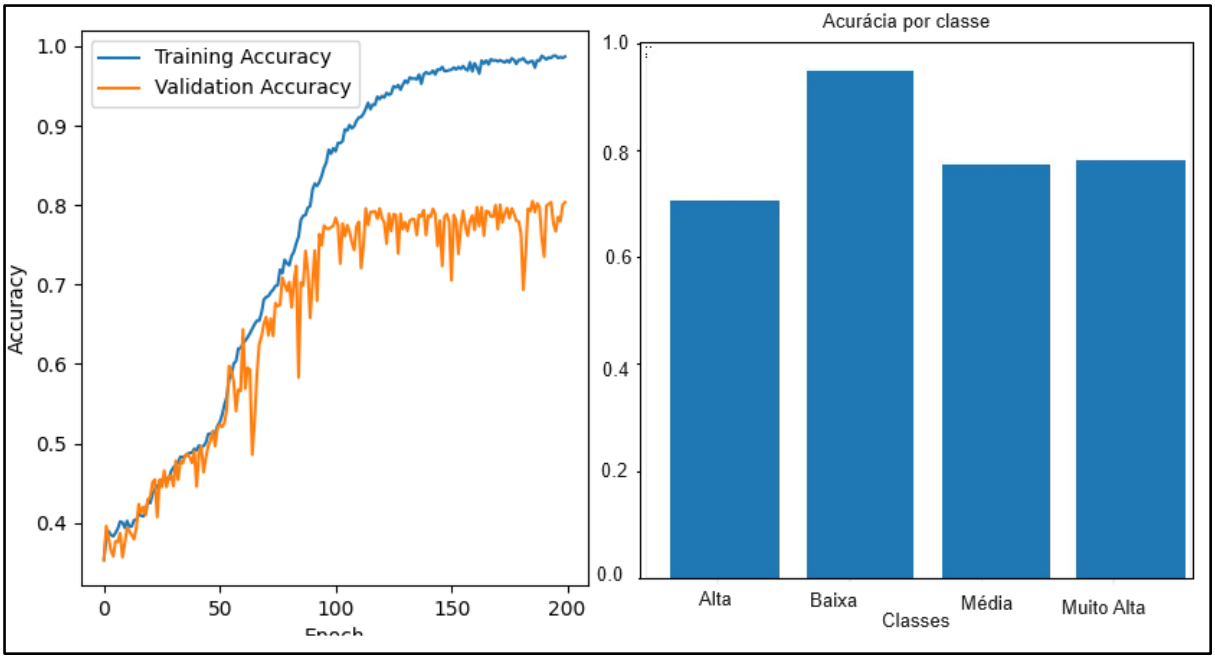
Fonte: O autor (2023).

Figura 47. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile*, *Shot*=10 nas imagens *RGB* 10



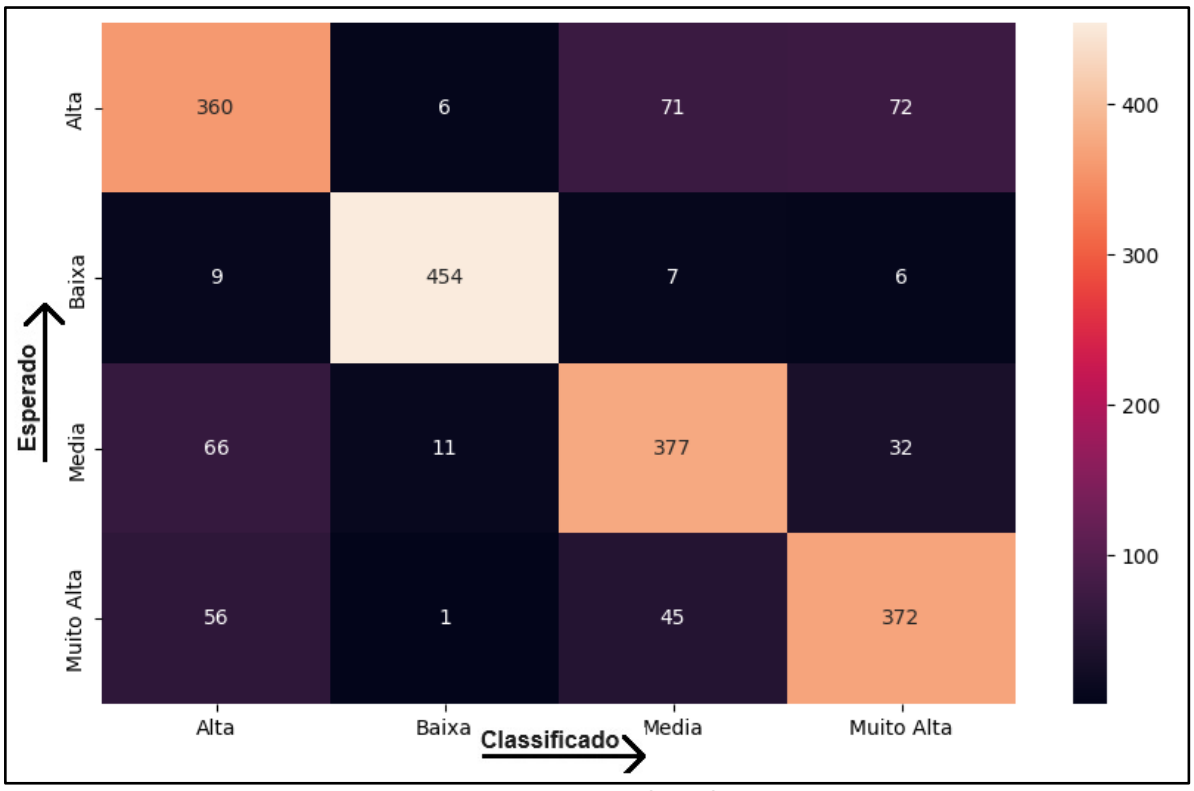
Fonte: O autor (2023).

Figura 48. Acurácia = 80.3 no modelo (A): com FSL/Reptile, Shot=10 nas imagens RGB 26



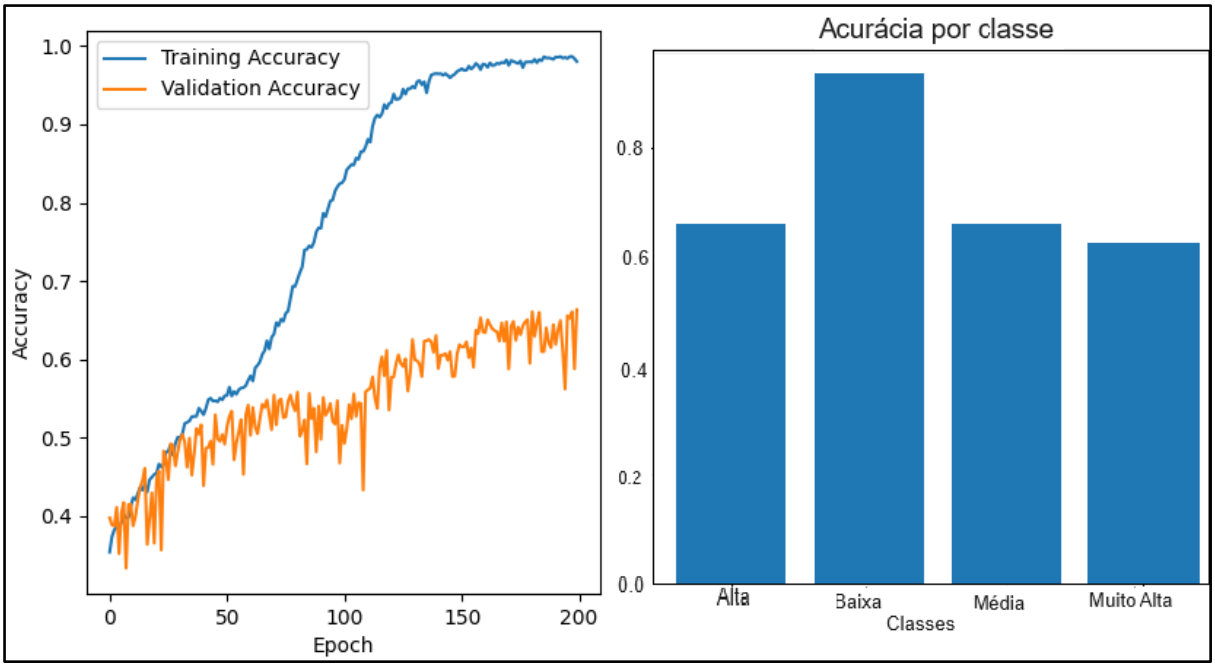
Fonte: O autor (2023).

Figura 49. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile, Shot*=10 nas Imagens *RGB* 26



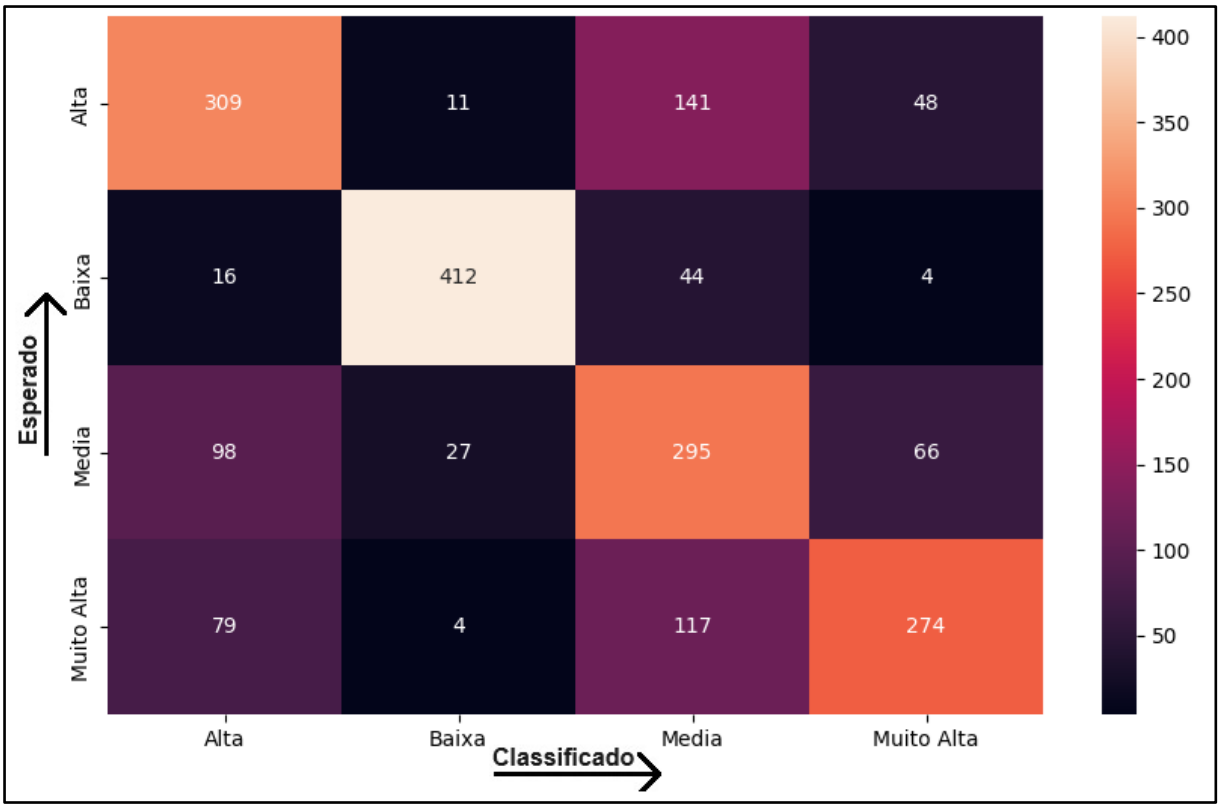
Fonte: O autor (2023).

Figura 50. Acurácia = 66.4 no Modelo (A): com *FSL/Reptile*, *Shot*=15 nas imagens *RGB* 10



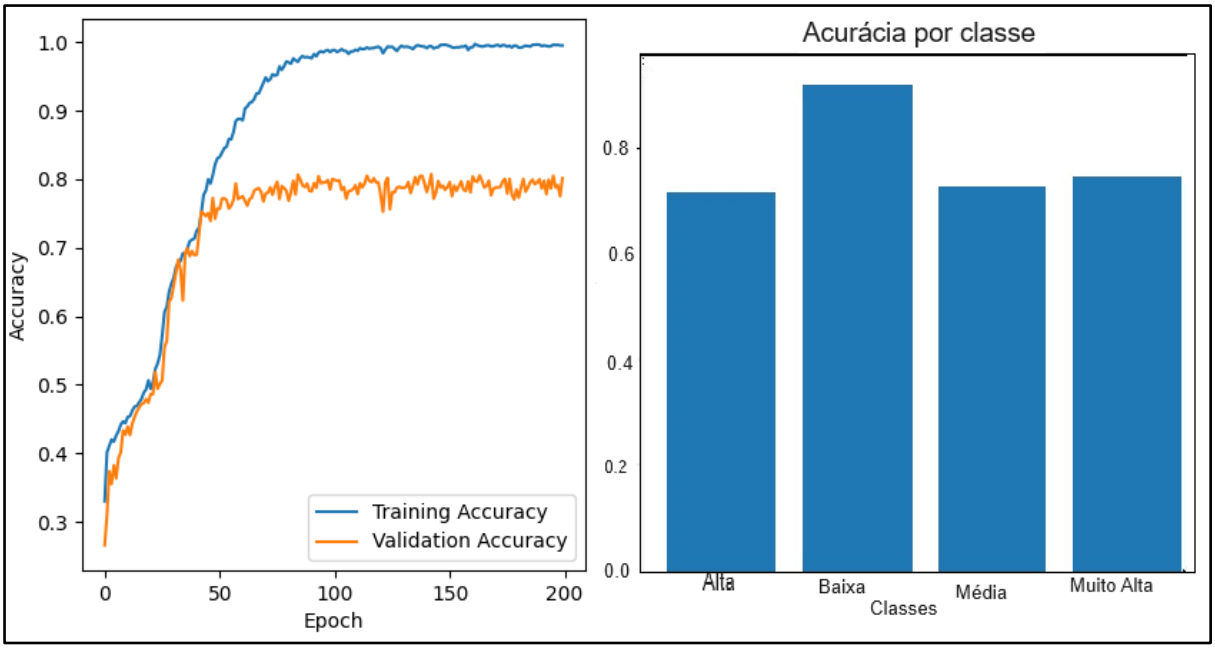
Fonte: O autor (2023).

Figura 51. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile*, *Shot*=15 nas imagens *RGB* 10



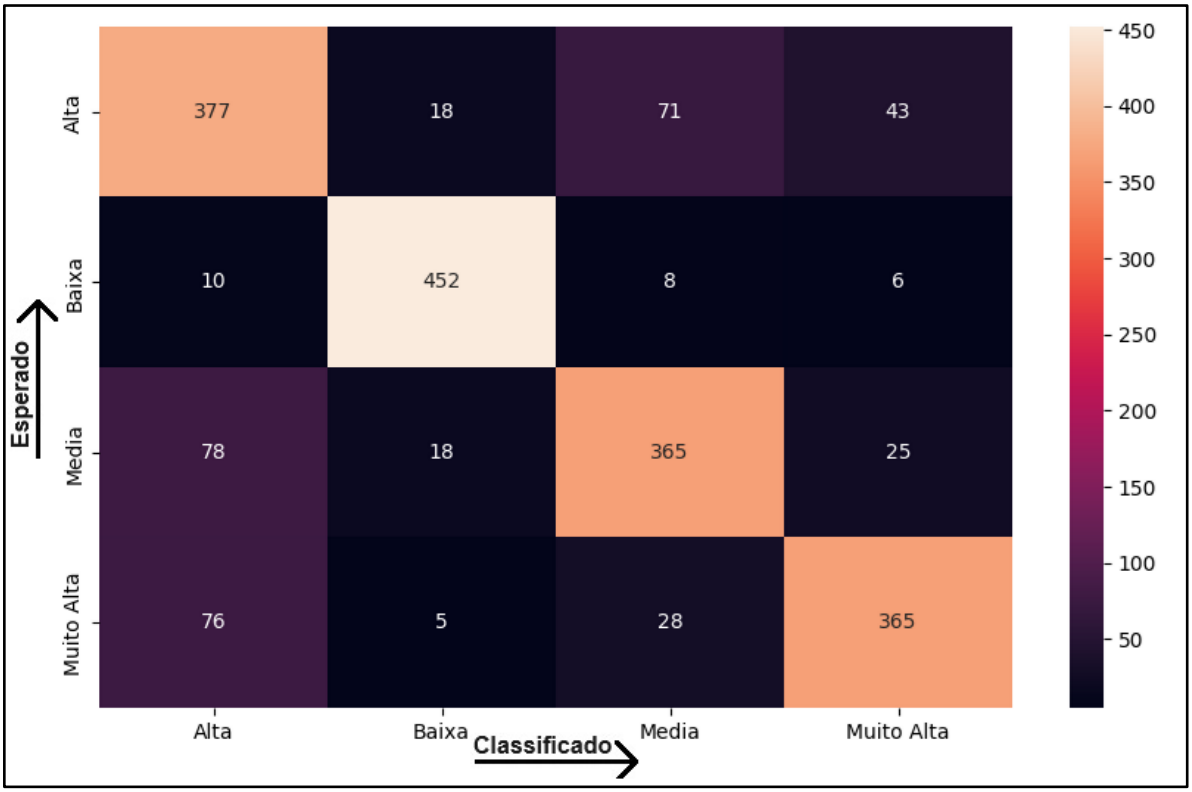
Fonte: O autor (2023).

Figura 52. Acurácia = 80.2 no modelo (A): com *FSL/Reptile, Shot*=15 nas imagens *RGB* 26



Fonte: O autor (2023).

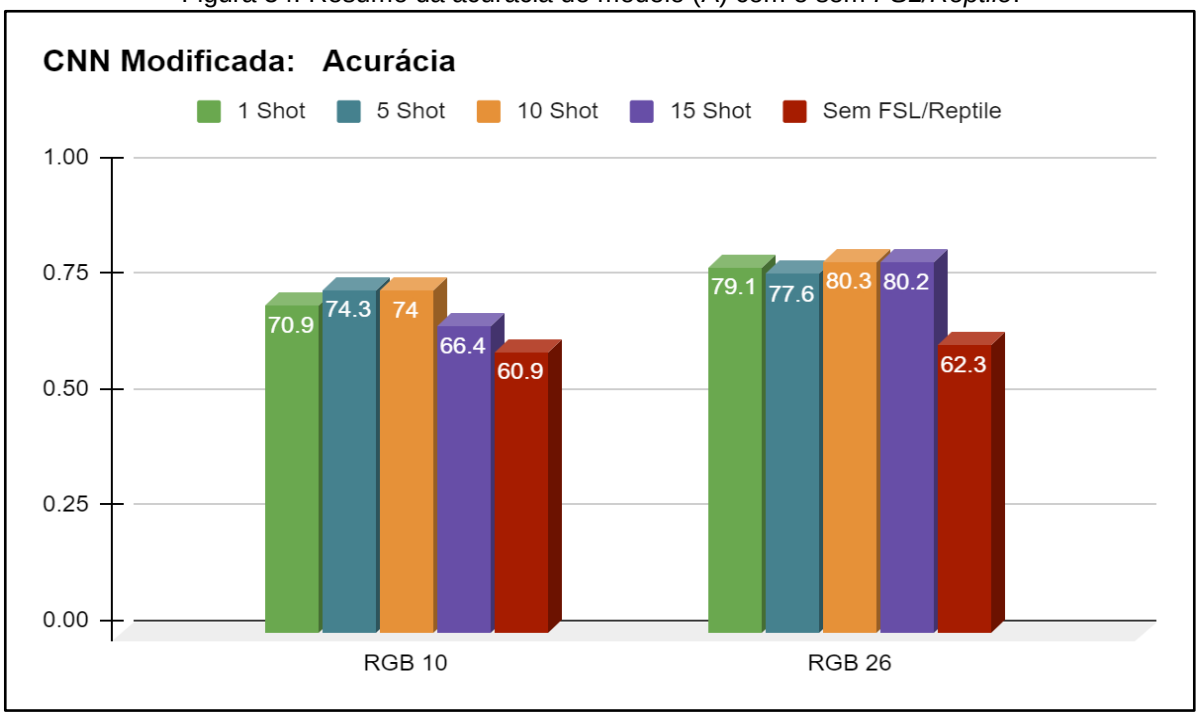
Figura 53. Matriz de Confusão Referente ao modelo (A): com *FSL/Reptile, Shot*=15 nas imagens *RGB* 26



Fonte: O autor (2023).

Os resultados para o modelo (A): *CNN* modificada indicaram que as melhores acurácias foram na resolução 26 cm/*px* com 10 e 15 *shots.* Na (Figura 54) está sendo apresentado o resumo dos resultados explorados para esse modelo de aprendizagem.

Figura 54. Resumo da acurácia do modelo (A) com e sem *FSL/Reptile*.

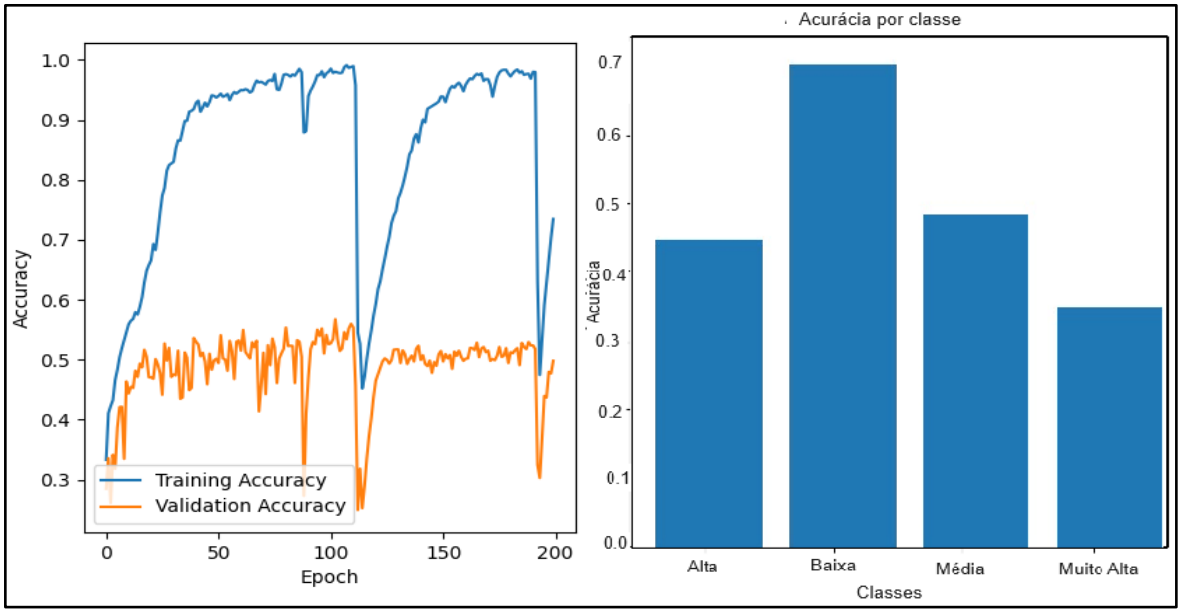


Fonte: O autor (2023).

5.2 MODELOS (B): *RESNET50* SEM *FSL/REPTILE* E COM *FSL / REPTILE*

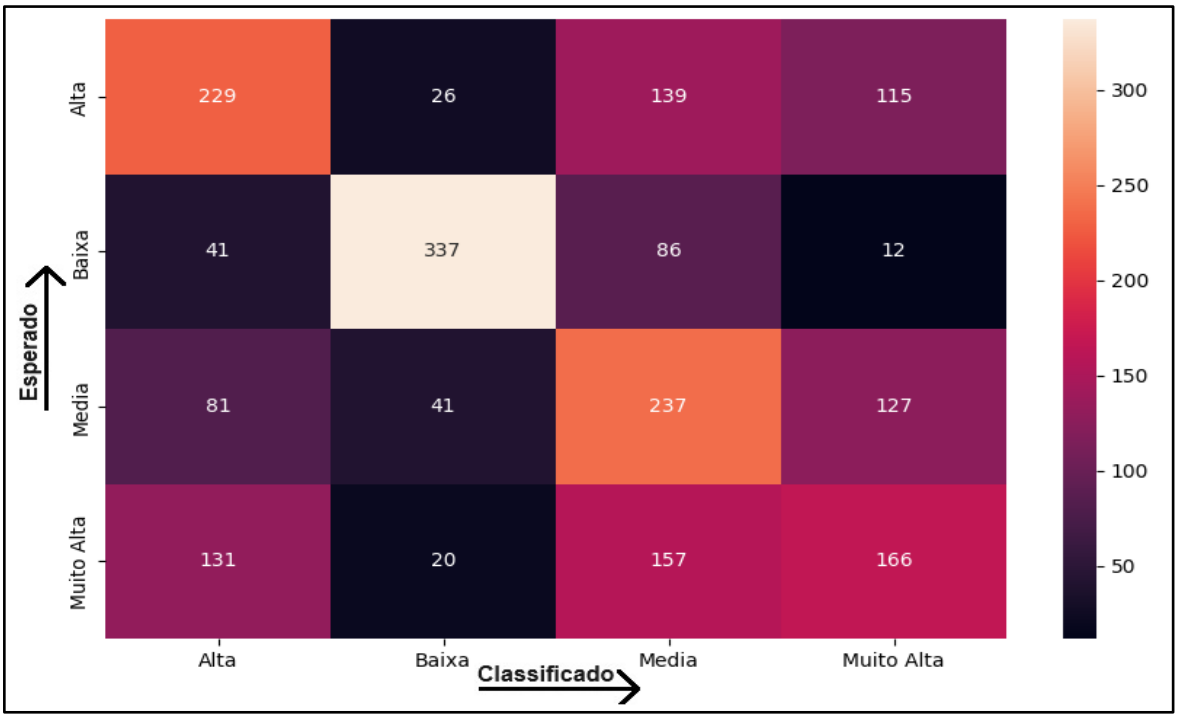
Aplicando os modelos de aprendizado *ResNet50* para o conjunto de imagens *RGB* 10 e *RGB* 26, sem *FSL*/*Reptile* atingiu uma acurácia de 49.8 (Figura 55) e 65.4 (Figura 57), respectivamente indicado pelos gráficos e por suas matrizes de confusão (Figura 56) e (Figura 58)

Figura 55.Acurácia = 49.8 no modelo (B): sem *FSL/Reptile* nas imagens *RGB* 10



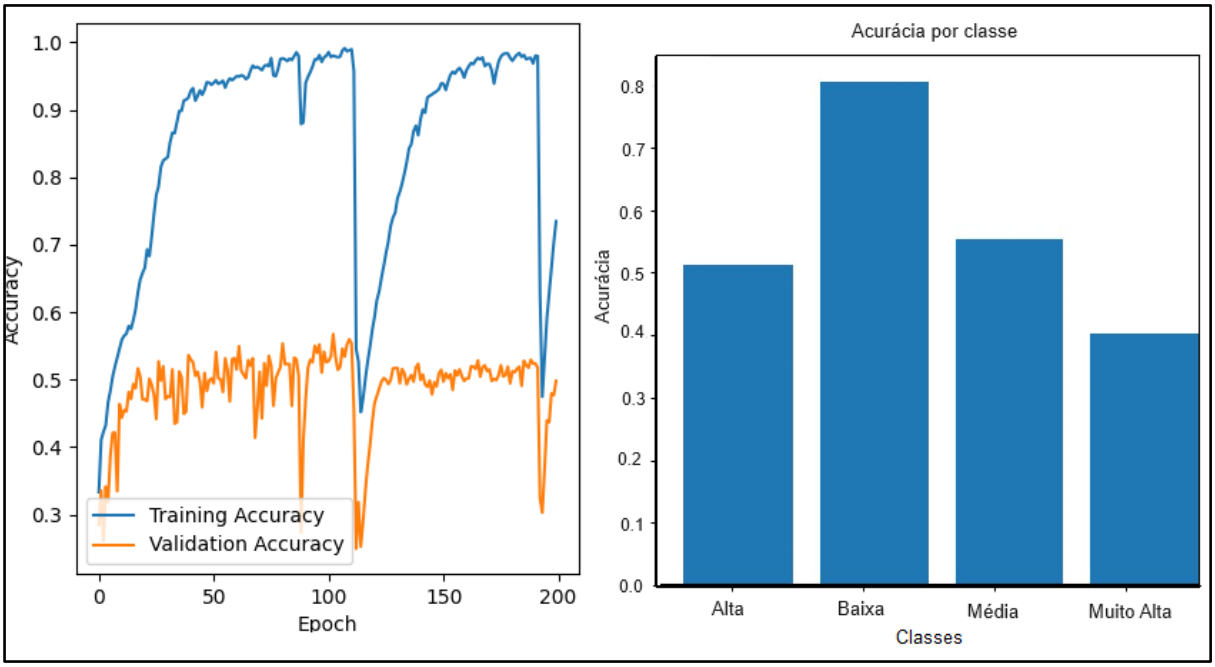
Fonte: O autor (2023).

Figura 56. Matriz de Confusão Referente ao modelo (B): sem *FSL/Reptile* nas imagens *RGB* 10



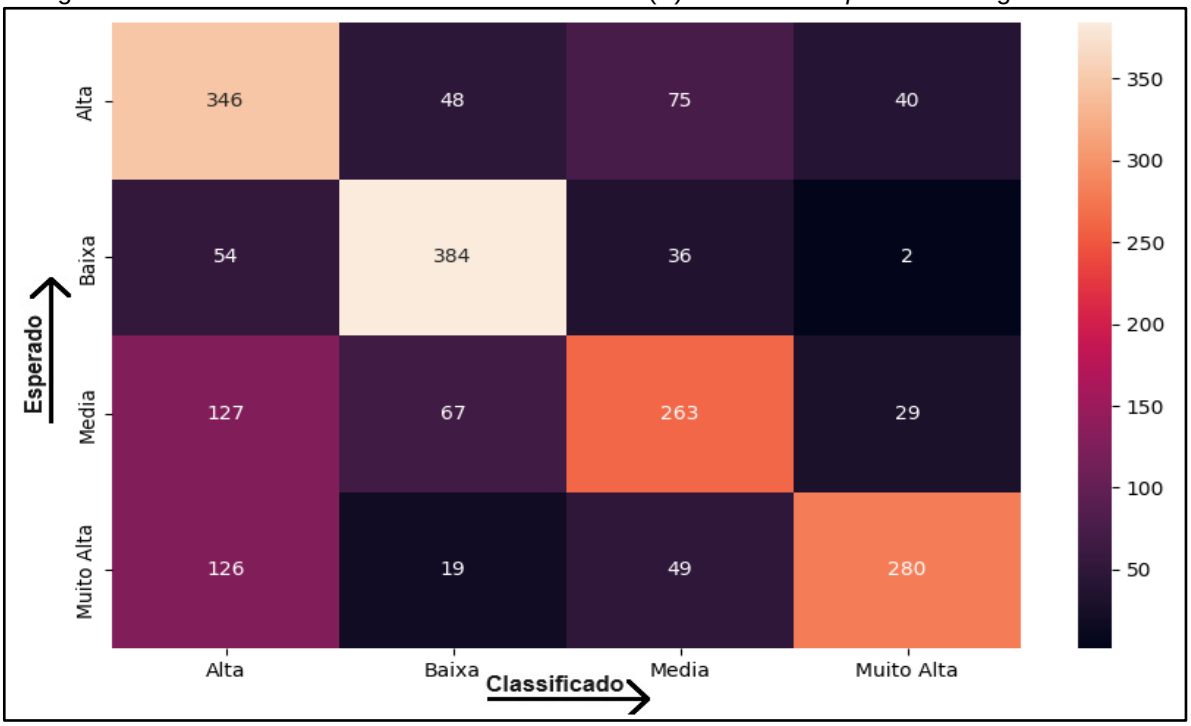
Fonte: O autor (2023).

Figura 57. Acurácia = 65.4 no modelo (B): *ResNet* sem *FSL/Reptile* nas imagens *RGB* 26



Fonte: O autor (2023)

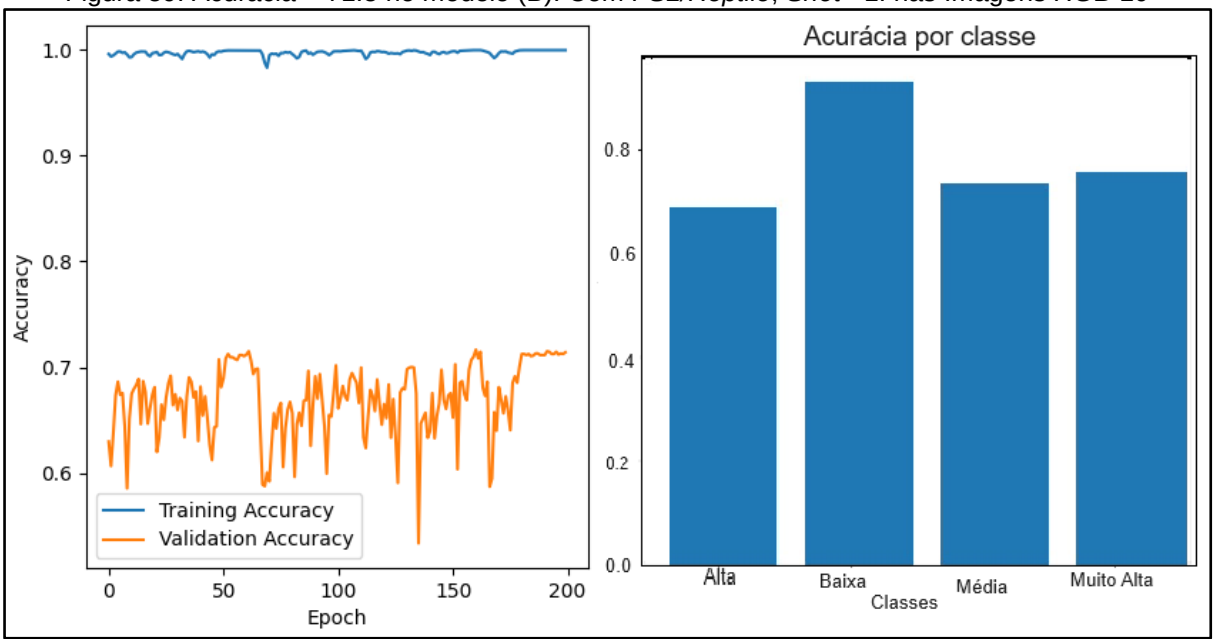
Figura 58. Matriz de Confusão referente ao modelo (B): sem *FSL/Reptile* nas imagens *RGB* 26



Fonte: O autor (2023)

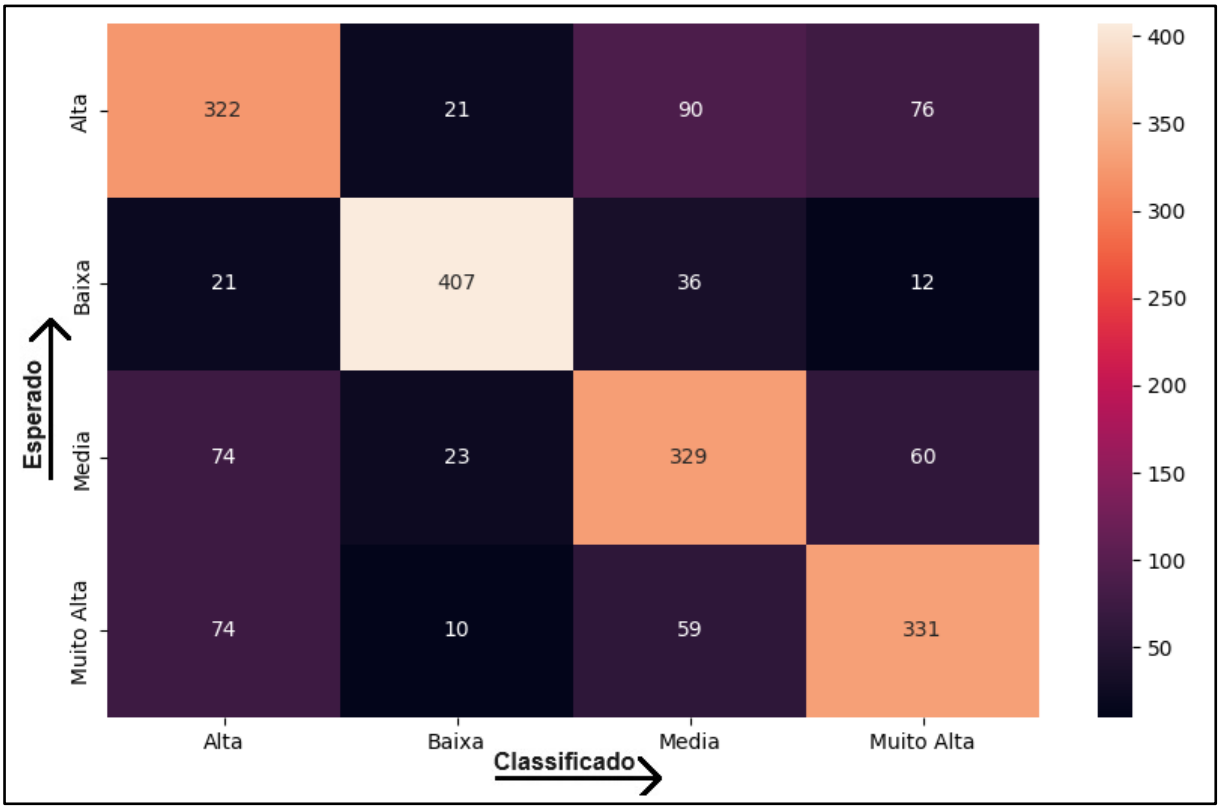
Utilizando a *ResNet50* com *FSL/Reptile* configurado para *Shot* = 1, com imagens na resolução 10 cm/*px*, foi atingida uma acurácia de 71.5 (Figura 59) indicado pelos gráficos e por sua matriz de confusão na (Figura 60). Ao aplicar o mesmo Modelo B em imagens com resolução de 26 cm/*px* a acurácia foi para 70.2 (Figura 61) a classe melhor predita é a classe de produtividade baixa, conforme gráficos da figura e sua matriz (Figura 62)

Figura 59. Acurácia = 71.5 no modelo (B): Com *FSL/Reptile*, *Shot* =1. nas Imagens *RGB* 10



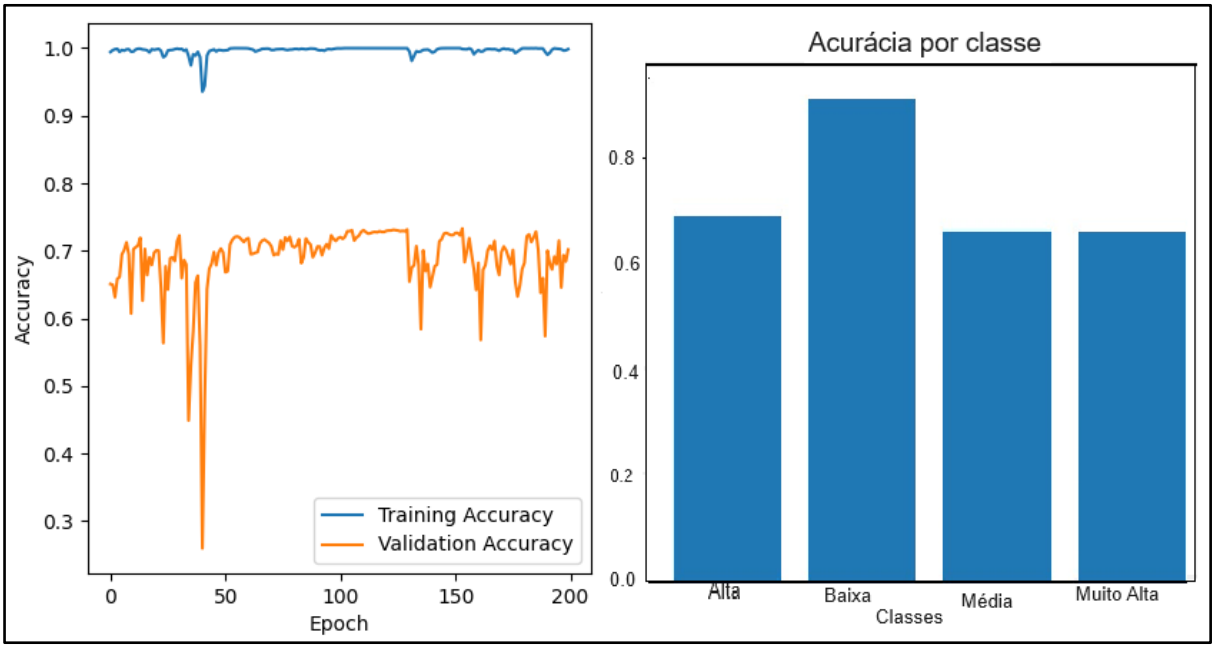
Fonte: O autor (2023).

Figura 60. Matriz de Confusão Referente ao modelo (B): Com *FSL/Reptile*, *Shot* =1 nas imagens *RGB* 10



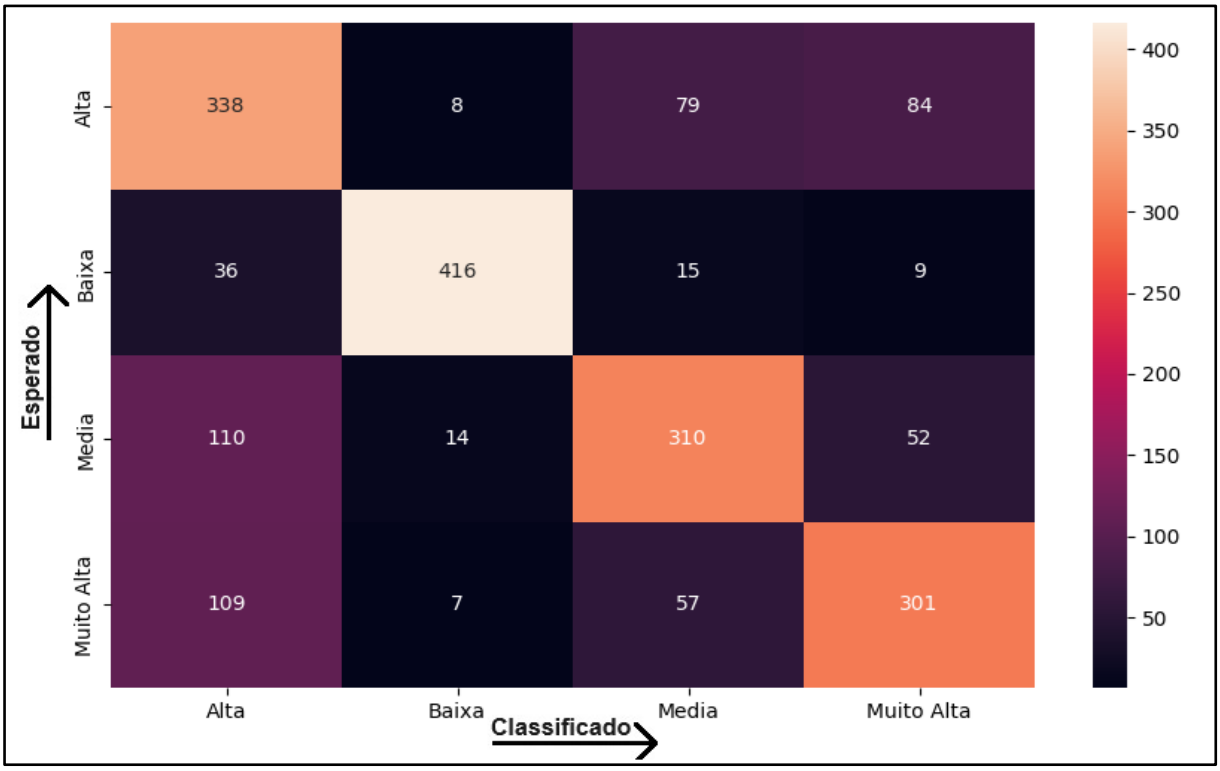
Fonte: O autor (2023).

Figura 61. Acurácia = 70.2 no modelo (B): Com *FSL/Reptile*, *Shot* =1. nas imagens *RGB* 26



Fonte: O autor (2023).

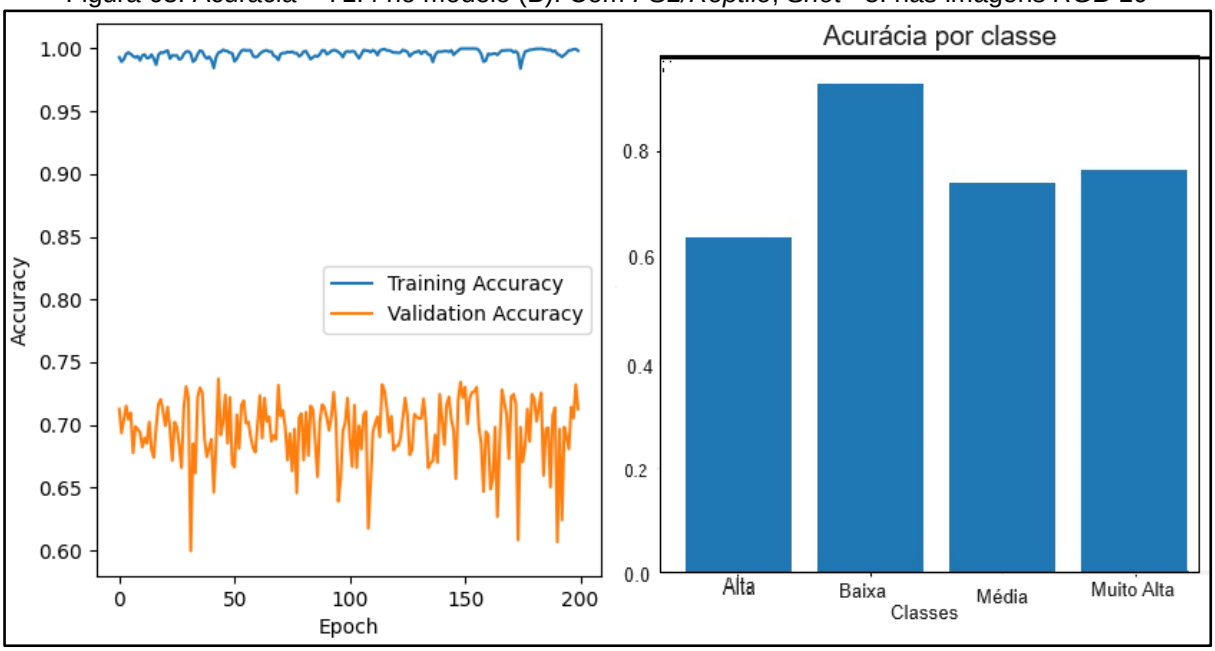
Figura 62. Matriz de Confusão Referente ao modelo (B): Com *FSL/Reptile*, *Shot* =1. nas imagens *RGB* 26



Fonte: O autor (2023).

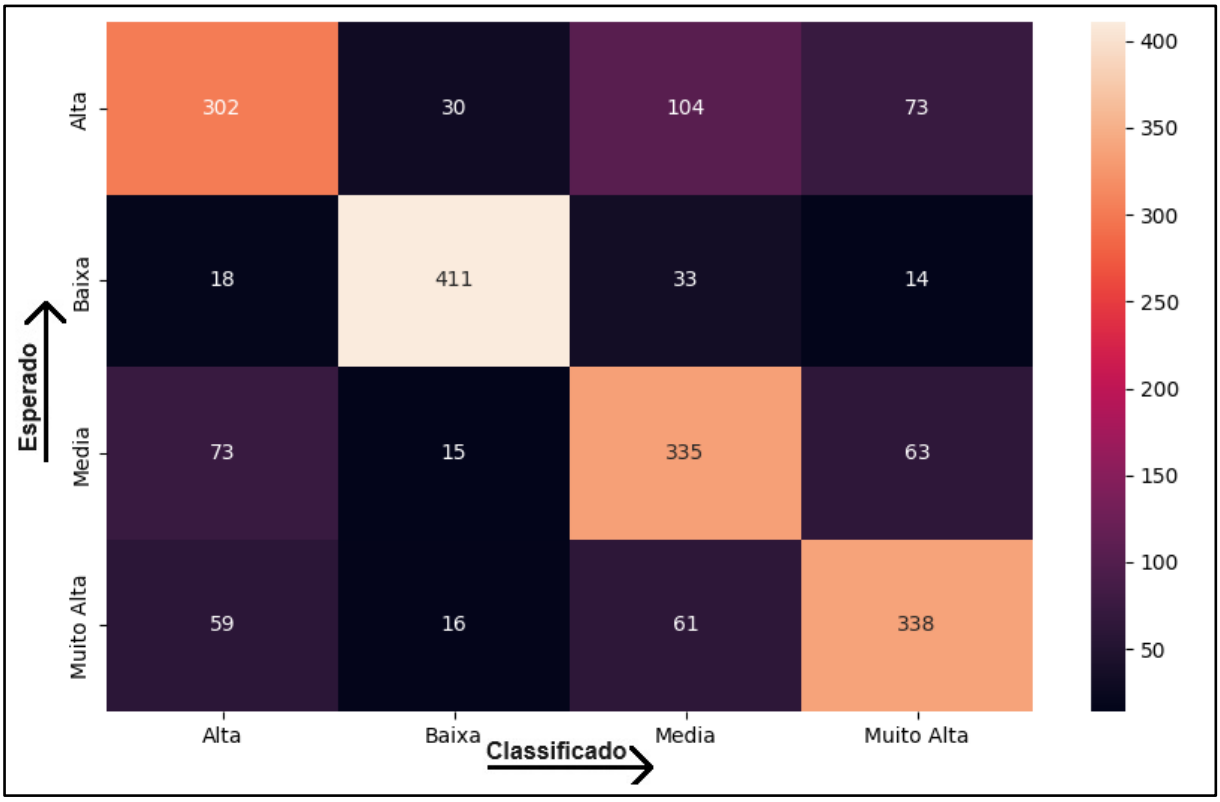
Os resultados obtidos ao variar o número de *shots* para 5 para o Modelo de Aprendizado B, tanto para imagens *RGB* 10 (Figura 63) quanto para *RGB* 26 (Figura 65), demonstram um resultado muito próximo para essas imagens com variação de 0,5. Com suas matrizes de confusão representadas em sequência (Figuras 64) e (Figura 66)

Figura 63. Acurácia = 71.4 no modelo (B): Com *FSL/Reptile*, *Shot* =5. nas imagens *RGB* 10



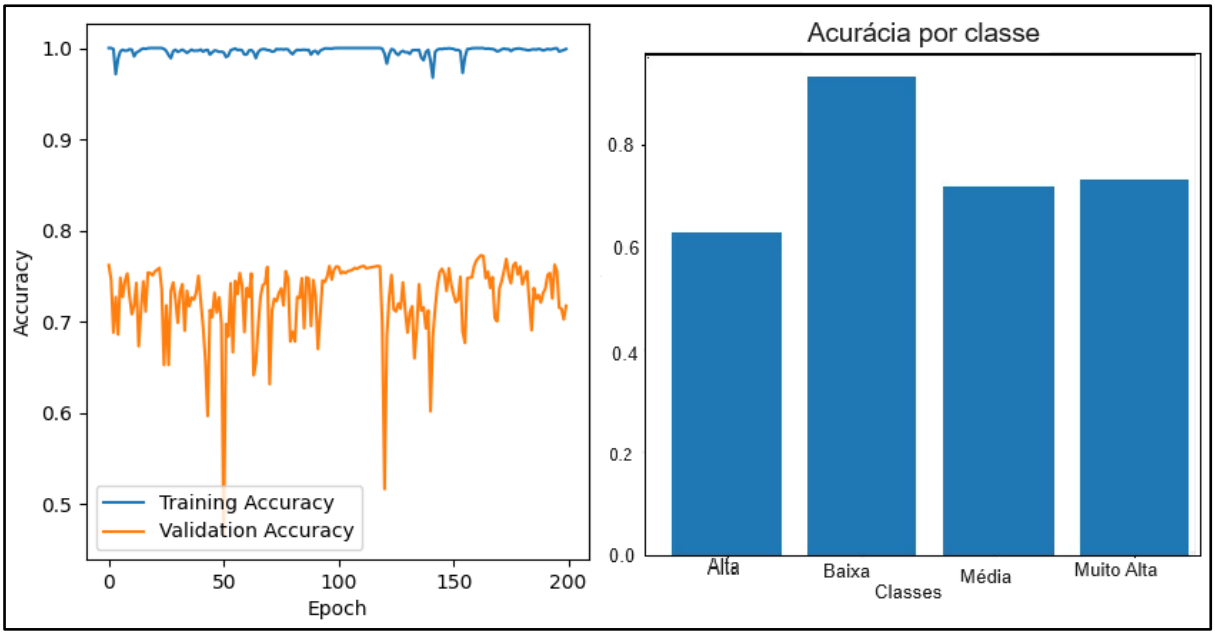
Fonte: O autor (2023).

Figura 64. Matriz de Confusão Referente ao modelo (B): Com *FSL/Reptile, Shot* =5 nas imagens *RGB*10



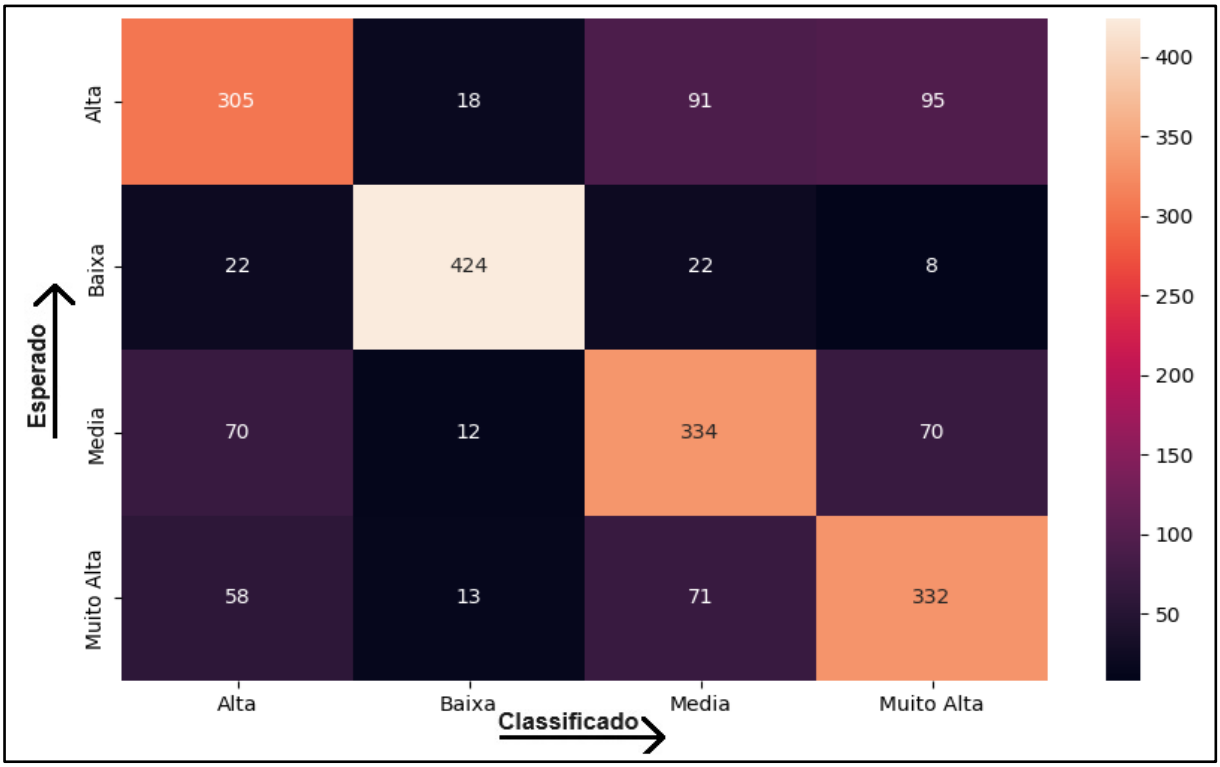
Fonte: O autor (2023).

Figura 65. Acurácia = 71.9 no modelo (B): Com *FSL/Reptile*, *Shot* =5. nas imagens *RGB* 26



Fonte: O autor (2023).

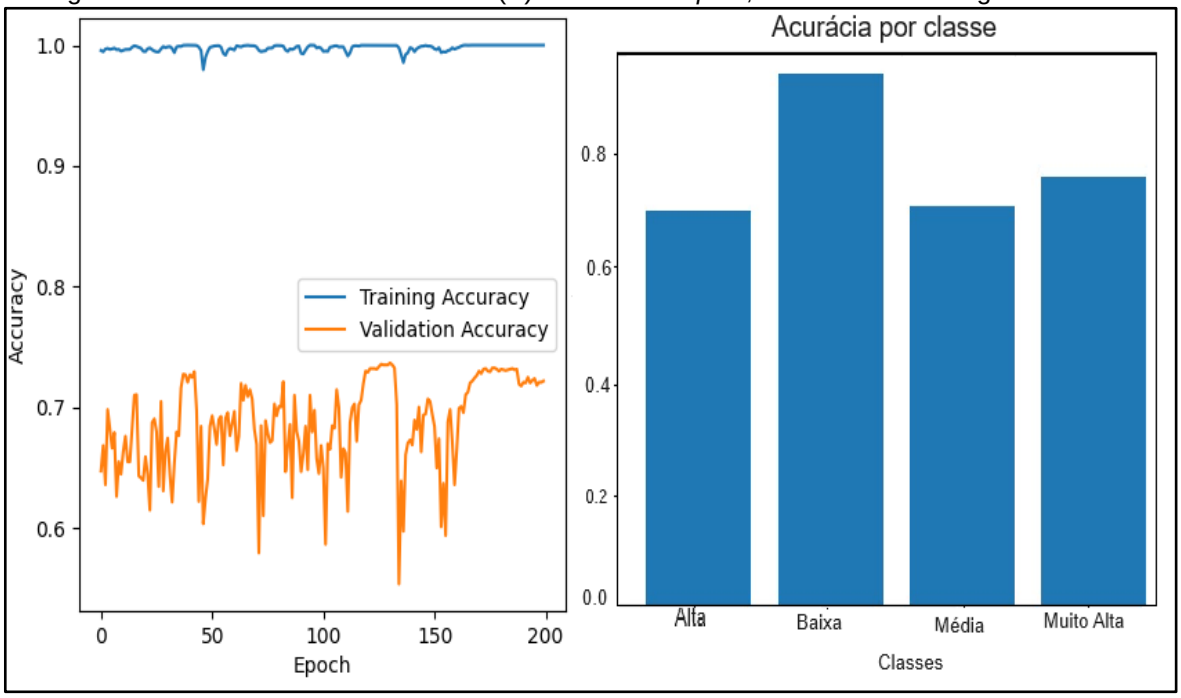
Figura 66. Matriz de Confusão referente ao modelo (B): Com *FSL/Reptile*, *Shot* =5 nas imagens *RGB* 26



Fonte: O autor (2023).

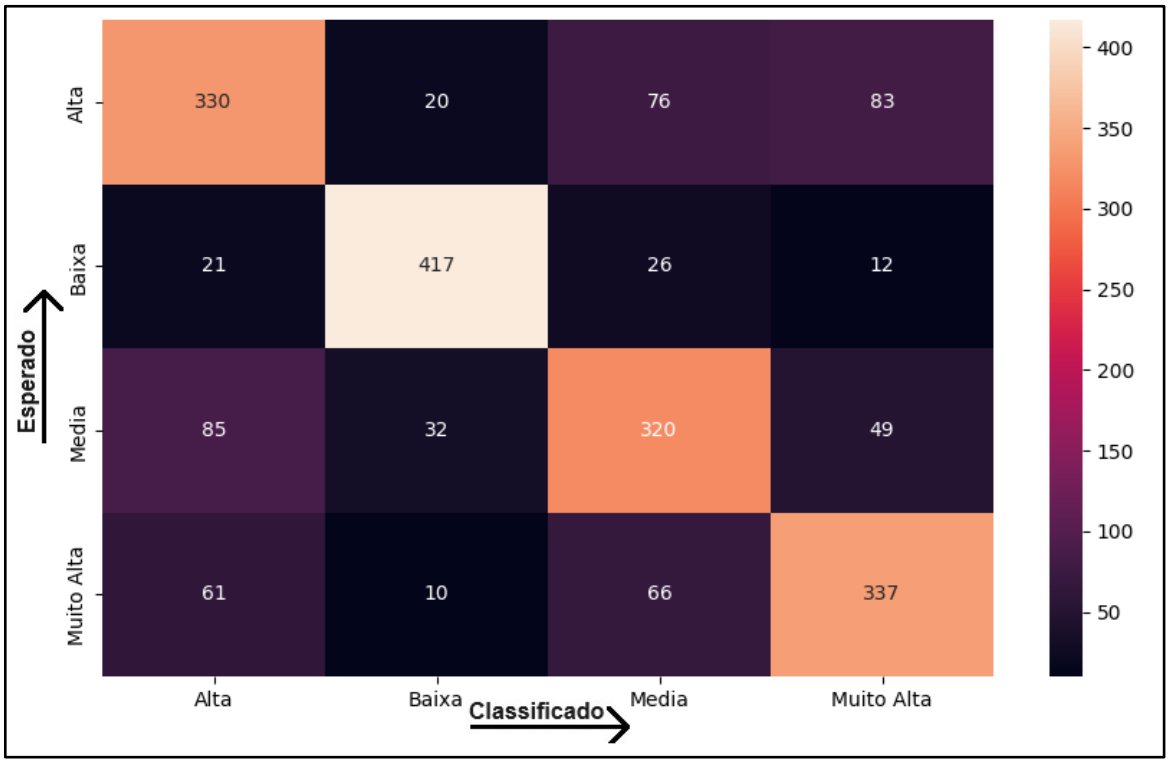
Ao aumentar o número de *shots* para 10, foi observado o aumento no desempenho do modelo. Como evidenciado na Figura 67, com sua matriz de confusão na (Figura 68). A (Figura 69) com sua matriz de confusão em sequência na (Figura 70), demonstraram uma melhoria no desempenho com base nas imagens *RGB* 26 e 10. Além disso, ao estender o número de *shots* para 15 uma melhoria adicional foi observada na acurácia com as imagens *RGB* 10 (Figura 71) com sua matriz de confusão na (Figura 72), observando também a maior pontuação da acurácia chegando a 75.8 na (Figura 73) nas imagens *RGB* 26 acompanhado de sua matriz de confusão na (Figura 74).

Figura 67. Acurácia = 72.3 no modelo (B): com *FSL/Reptile*, *Shot* =10. nas imagens *RGB* 10



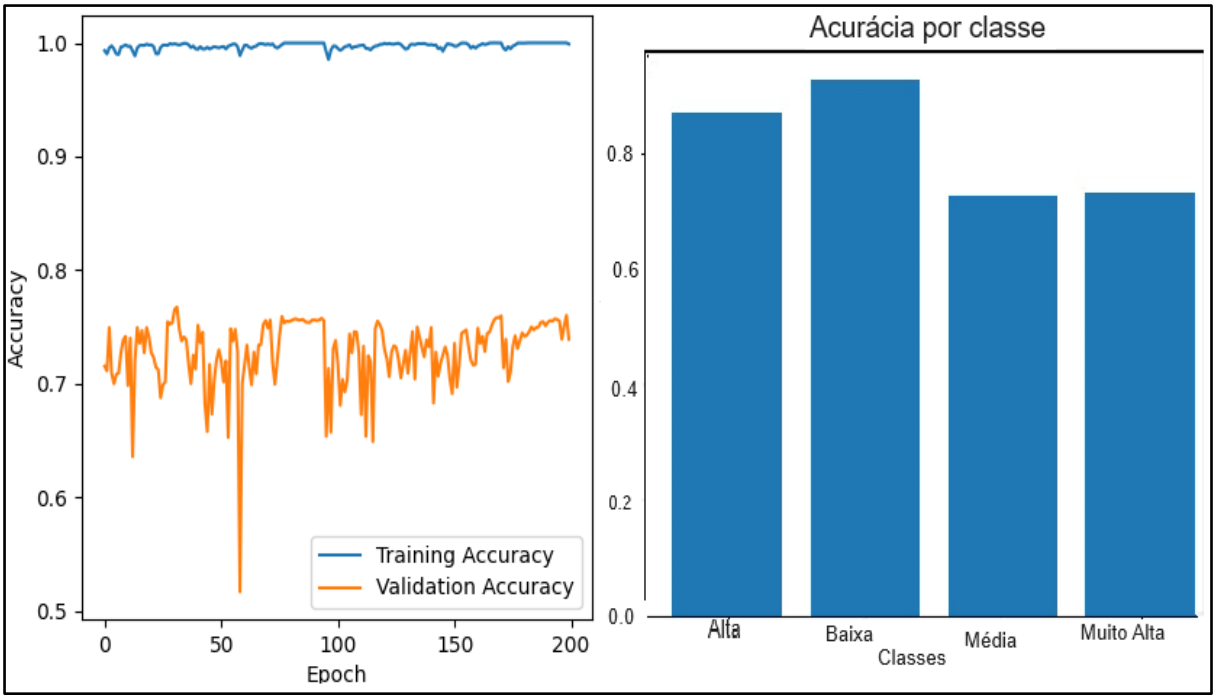
Fonte: O autor (2023).

Figura 68. Matriz de Confusão Referente ao modelo (B): com *FSL/Reptile*, Shot =10 nas Imagens RGB 10



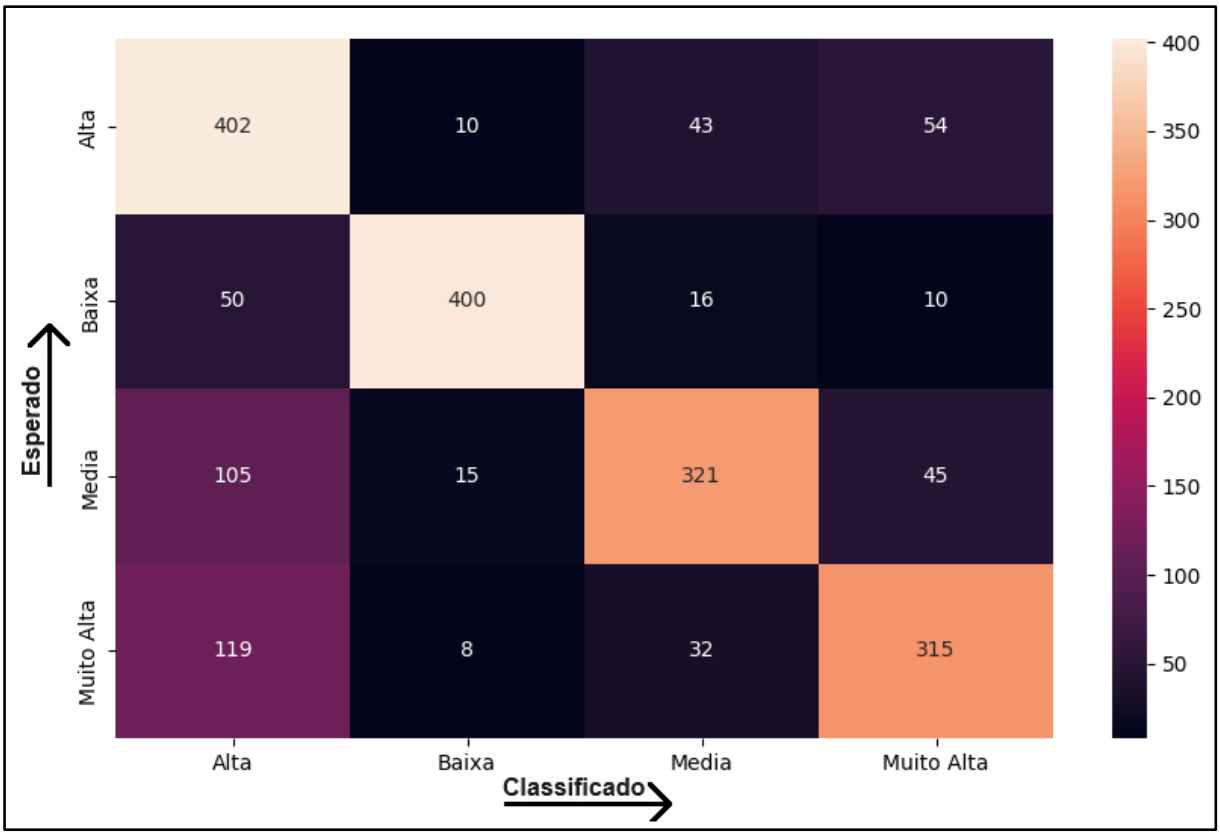
Fonte: O autor (2023).

Figura 69. Acurácia = 73.8 no modelo (B): com *FSL/Reptile*, *Shot* =10 nas imagens *RGB* 26



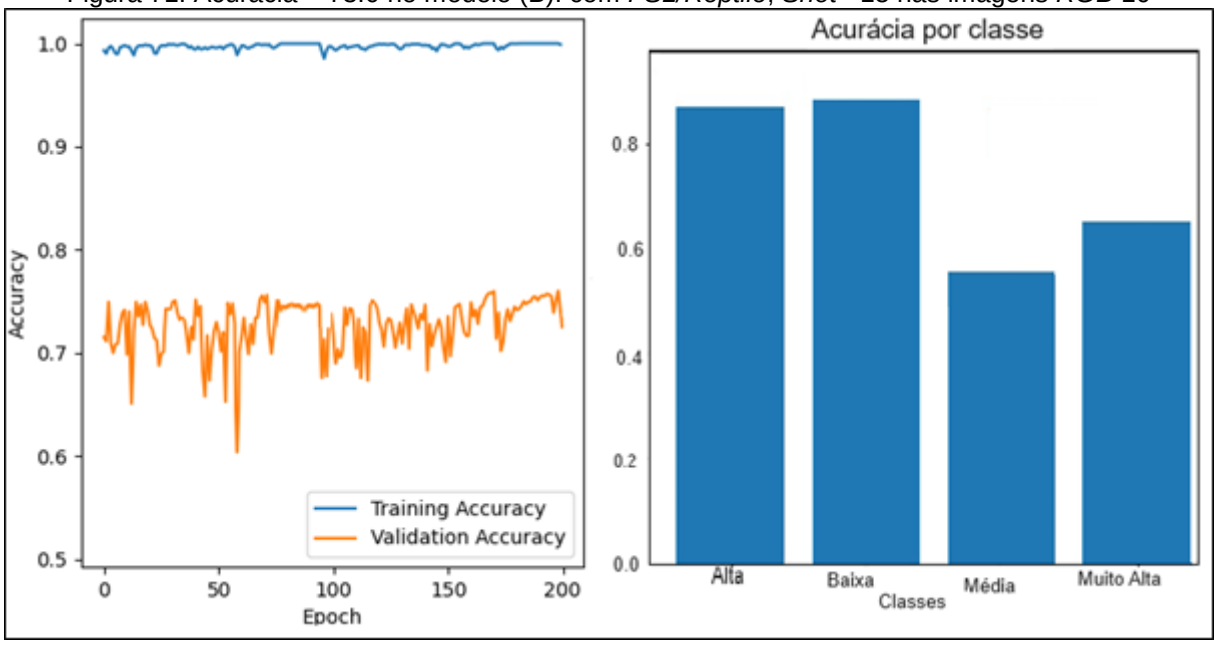
Fonte: O autor (2023).

Figura 70. Matriz de Confusão Referente ao modelo (B): com *FSL/Reptile*, *Shot* =10 nas imagens *RGB* 26



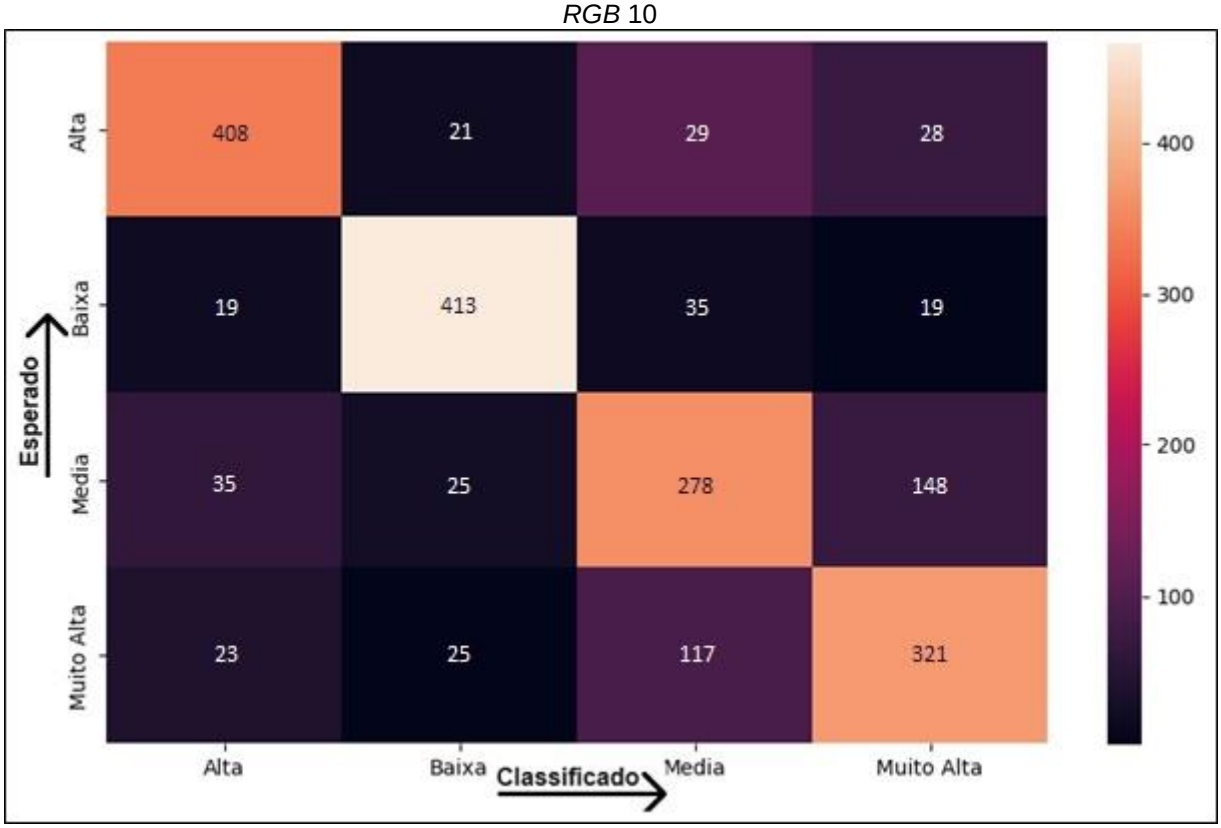
Fonte: O autor (2023).

Figura 71. Acurácia = 73.0 no modelo (B): com *FSL/Reptile*, *Shot* =15 nas imagens *RGB* 10



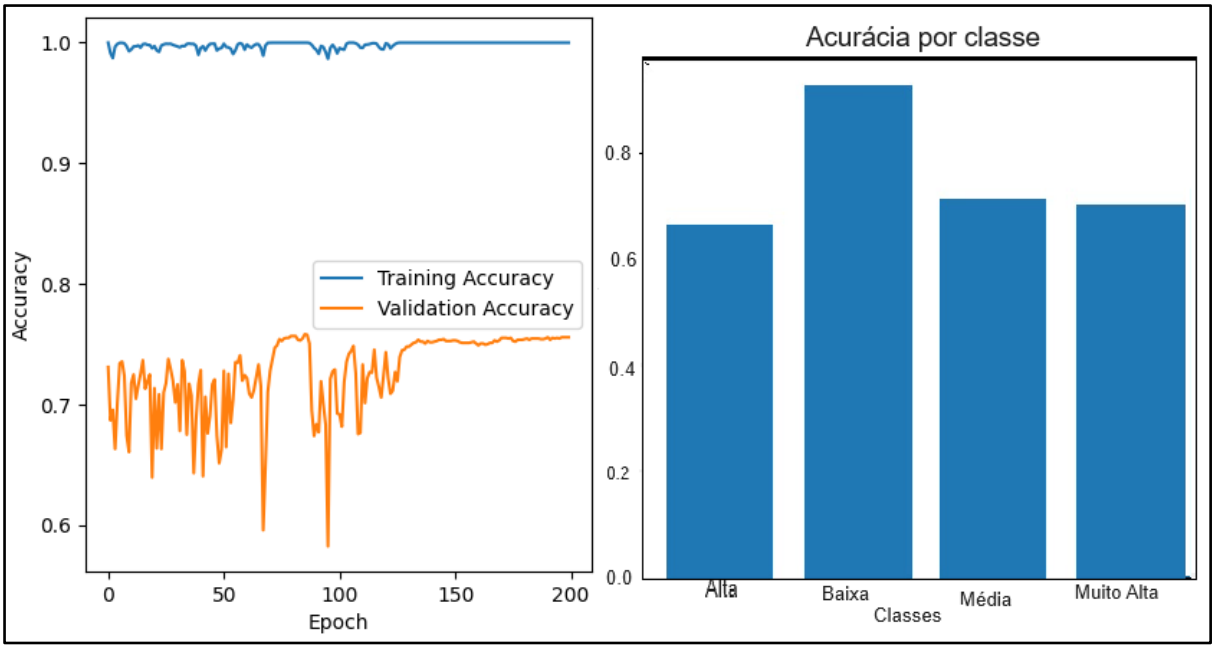
Fonte: O autor (2023).

Figura 72. Matriz de Confusão Referente ao modelo (B): com *FSL/Reptile*, *Shot* =15 nas imagens *RGB* 10



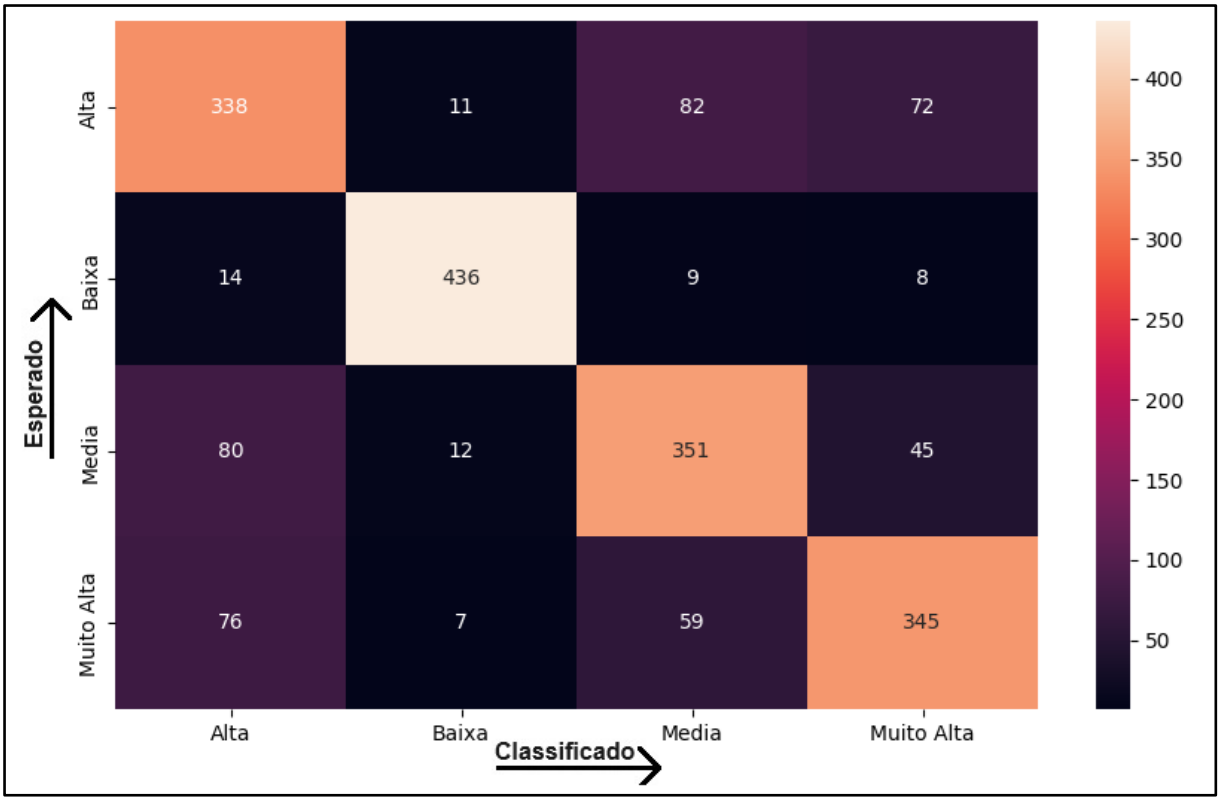
Fonte: O autor (2023).

Figura 73. Acurácia = 75.8 no modelo (B): com *FSL/Reptile*, *Shot* =15 nas imagens *RGB* 26



Fonte: O autor (2023).

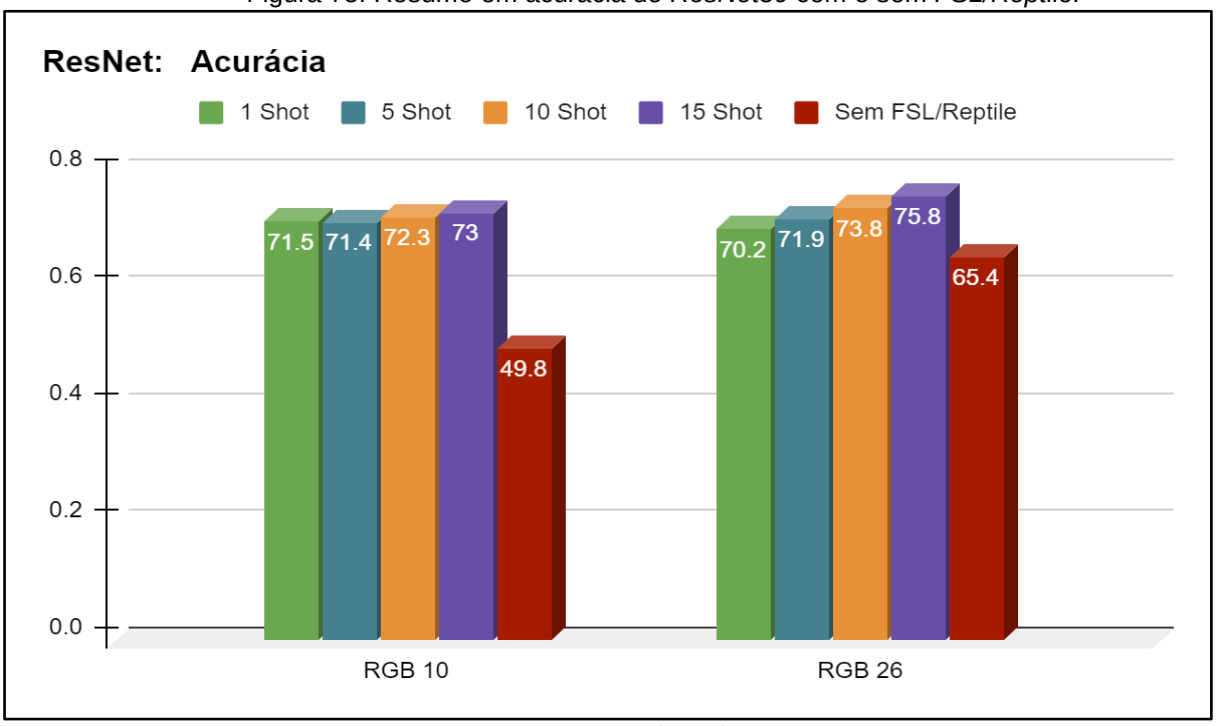
Figura 74. Matriz de Confusão Referente ao modelo (B): com *FSL/Reptile*, *Shot* =15 nas imagens *RGB* 26



Fonte: O autor (2023).

Os resultados dos modelos (B): *ResNet* demonstraram que as acurácias mais elevadas foram alcançadas utilizando a resolução de 26 cm/*px* e aplicando 10 e 15 *shots*. A (Figura 75) exibe uma representação gráfica dos resultados obtidos através dos modelos de aprendizagem.

Figura 75. Resumo em acurácia de *ResNet50* com e sem *FSL/Reptile.*

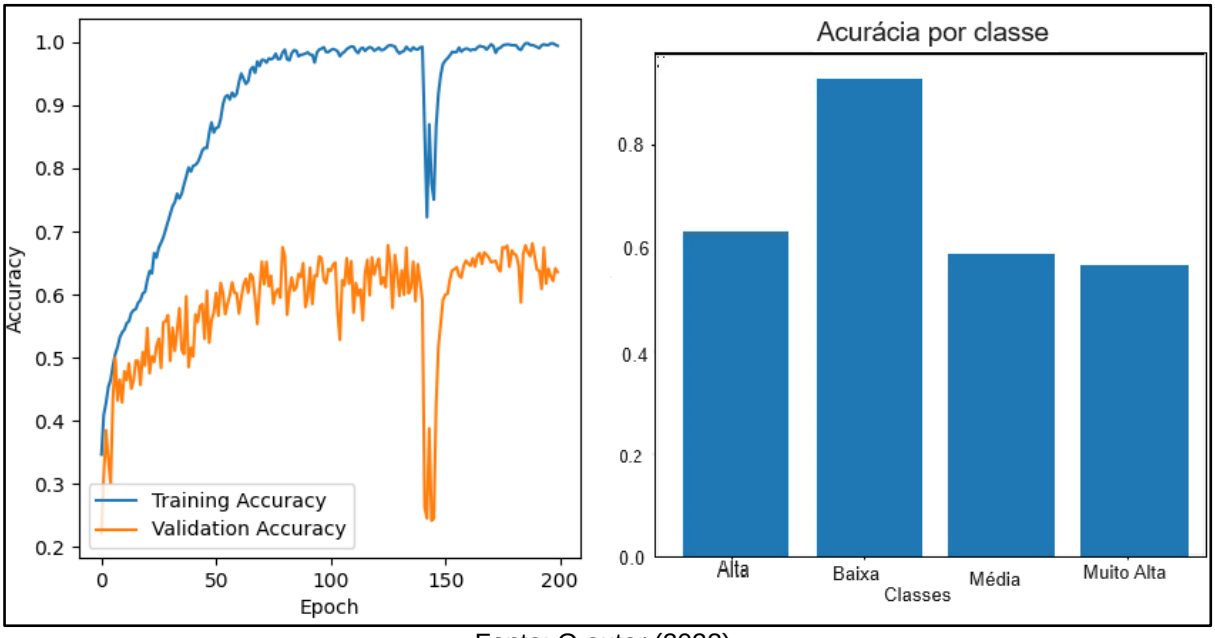


Fonte: O autor (2023).

5.3 MODELOS (C): *DENSENET121* SEM *FSL / REPTILE* E COM *FSL / REPTILE*

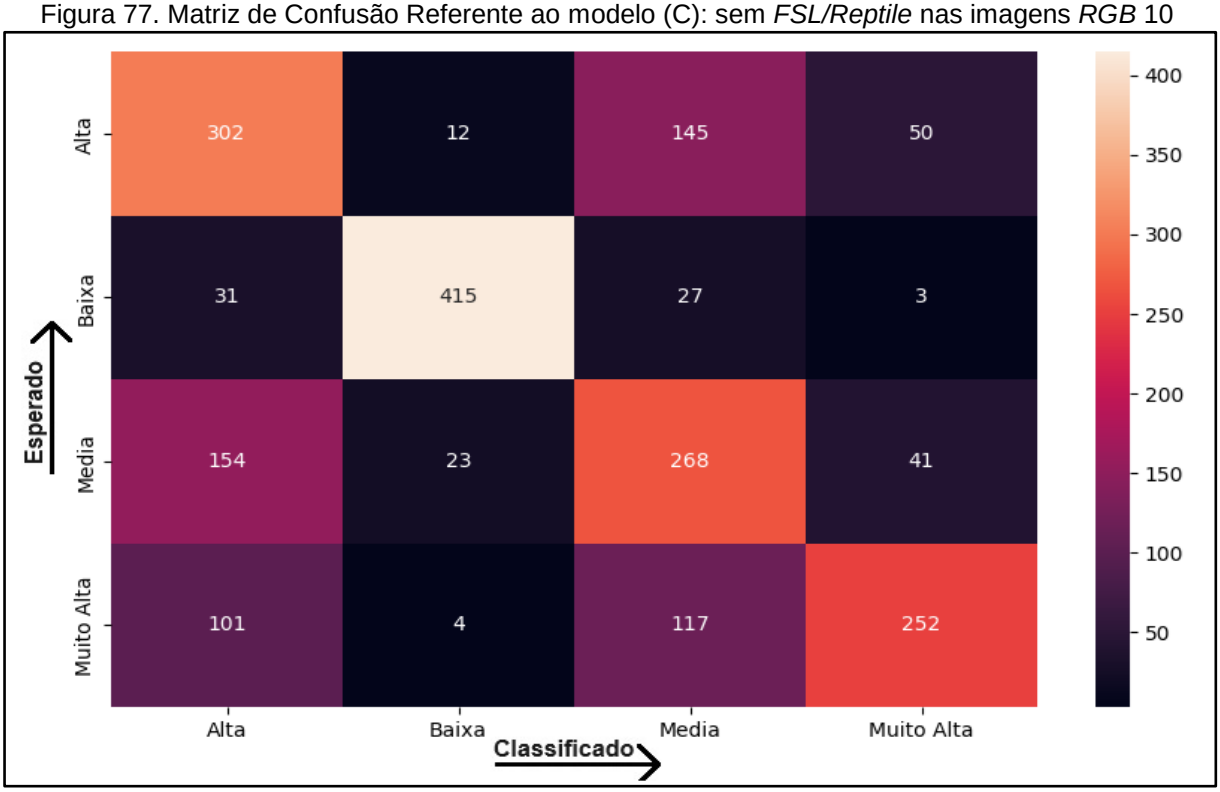
O modelo C foi aplicado no conjunto de imagens *RGB* 10 e 26 cm/*px* atingindo uma acurácia de 63.7 (Figura 76) acompanhado de sua matriz de confusão na (Figura 77) e acurácia de 72.0 (Figura 78), sem a utilização da metodologia *FSL/Reptile,* constatado pela matriz de confusão na (Figura 79).

Figura 76. Acurácia = 63.7 no modelo (C): sem *FSL/Reptile* nas imagens *RGB* 10



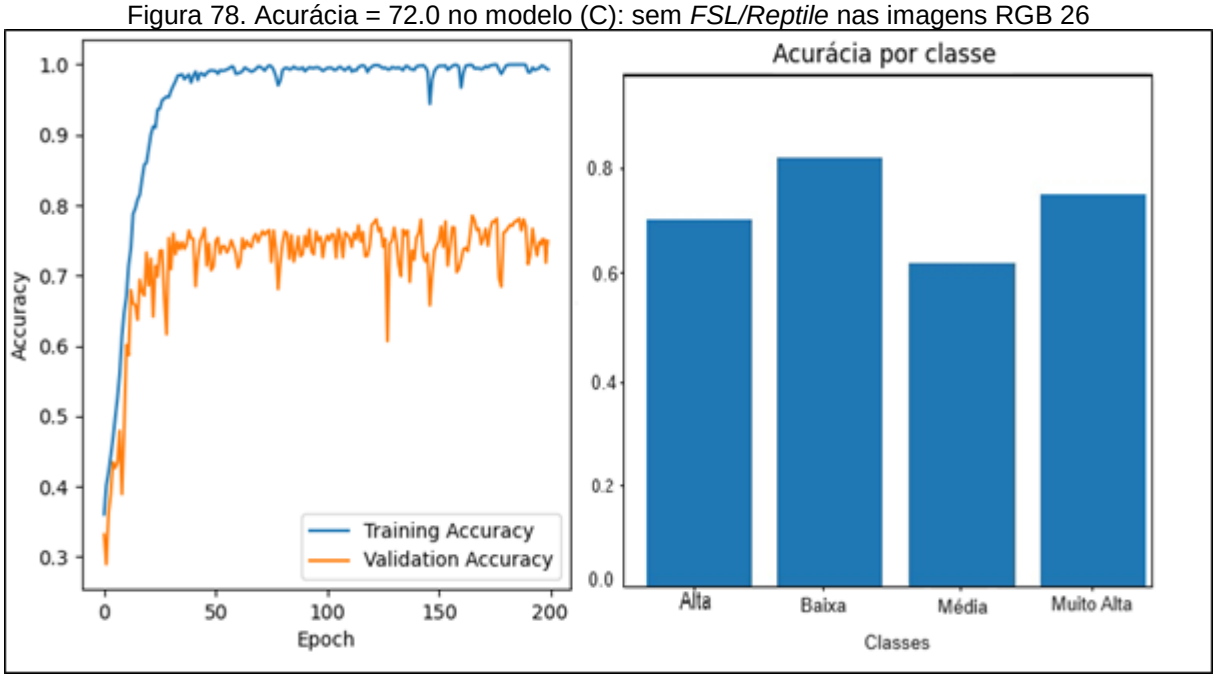
Fonte: O autor (2023).

Figura 77. Matriz de Confusão Referente ao modelo (C): sem *FSL/Reptile* nas imagens *RGB* 10



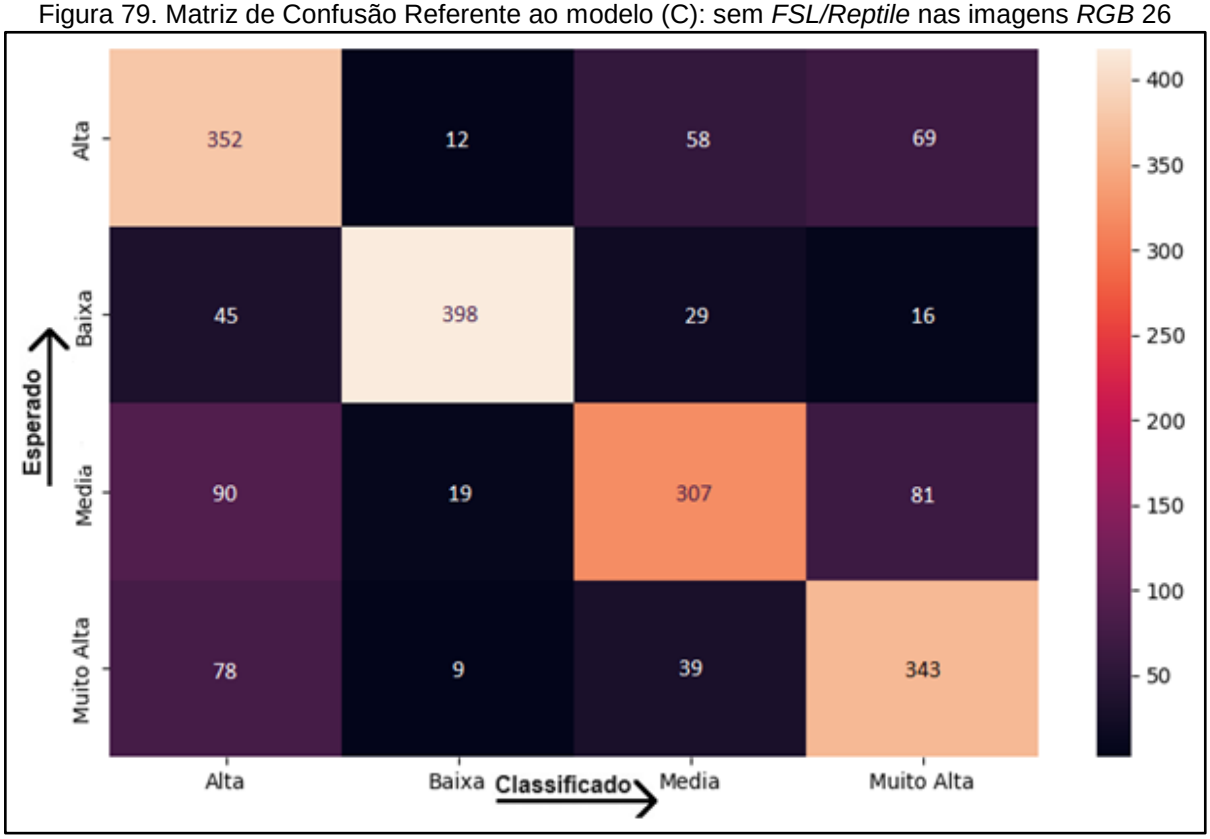
Fonte: O autor (2023).

Figura 78. Acurácia = 72.0 no modelo (C): sem *FSL/Reptile* nas imagens RGB 26



Fonte: O autor (2023).

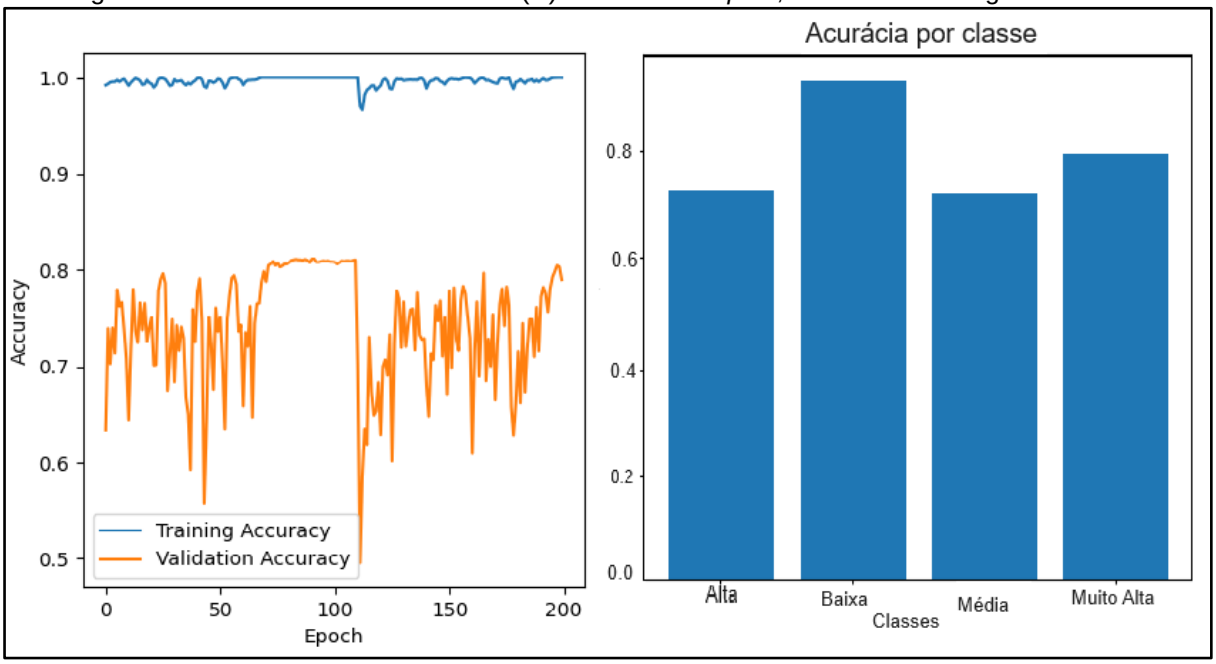
Figura 79. Matriz de Confusão Referente ao modelo (C): sem *FSL/Reptile* nas imagens *RGB* 26



Fonte: O autor (2023).

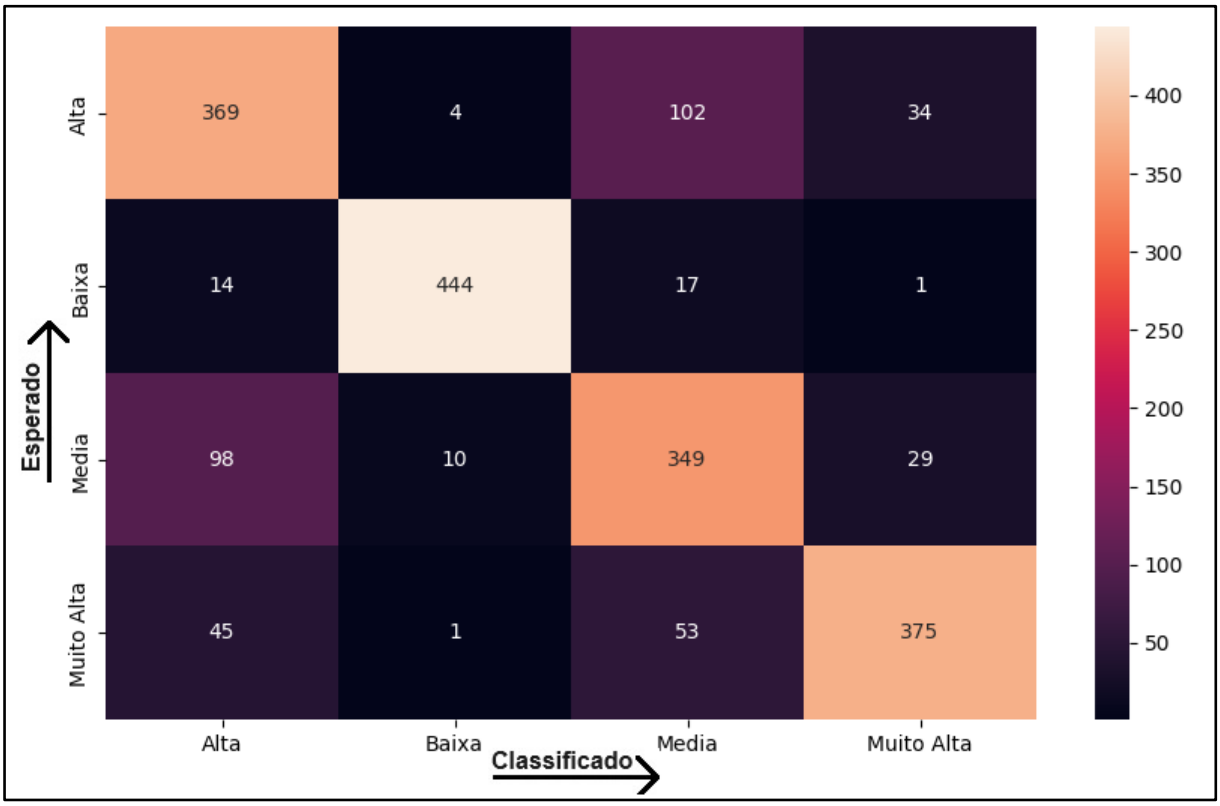
Ao executar os modelos (C): *DenseNet121* com *FSL*/*Reptile* configurado para 1 *shot,* notou-se diferenças no desempenho baseado nas imagens *RGB*. O Modelo apresentou uma acurácia de 79.1 quando aplicado em imagens *RGB* 10 (Figura 80), junto de sua matriz de confusão na (Figura 81), enquanto alcançou uma acurácia inferior de 76.5 quando aplicado em imagens *RGB* 26 (Figura 82) acompanhada de sua matriz de confusão na (Figura 83).

Figura 80. Acurácia = 79.1 no modelo (C): Com *FSL/Reptile*, *Shot* =1 nas imagens *RGB* 10



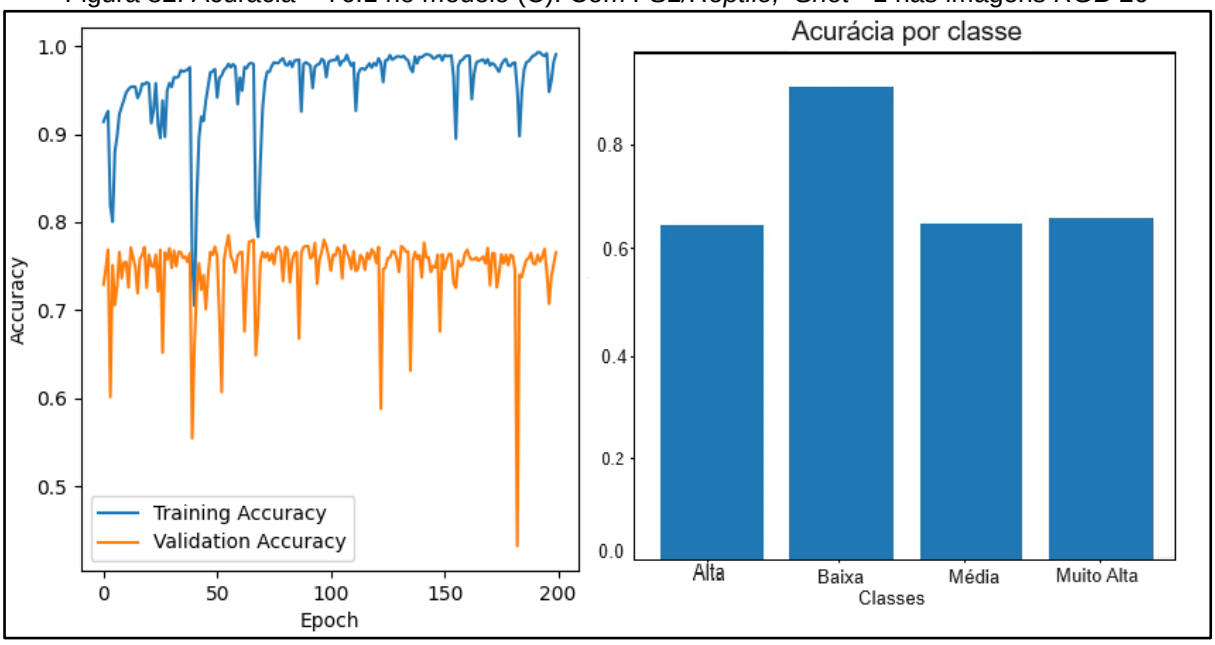
Fonte: O autor (2023).

Figura 81. Matriz de Confusão Referente ao modelo (C): Com *FSL/Reptile, Shot* =1 nas imagens *RGB* 10



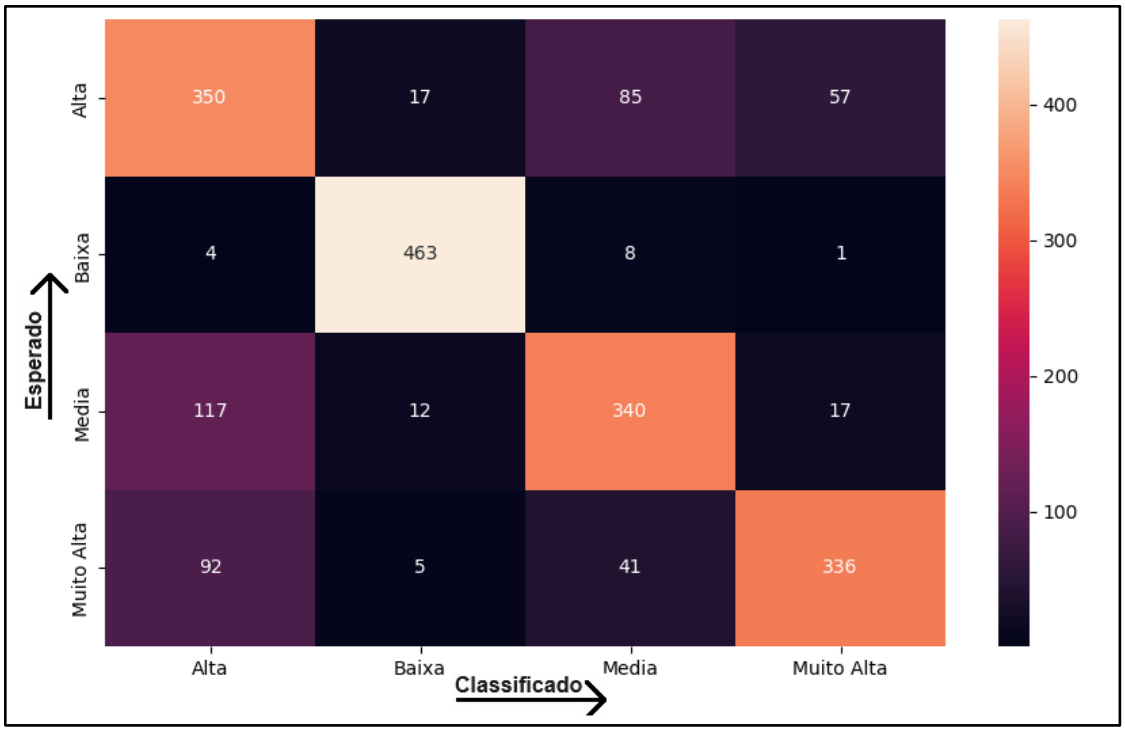
Fonte: O autor (2023).

Figura 82. Acurácia = 76.1 no modelo (C): Com *FSL/Reptile, Shot* =1 nas imagens *RGB* 26



Fonte: O autor (2023).

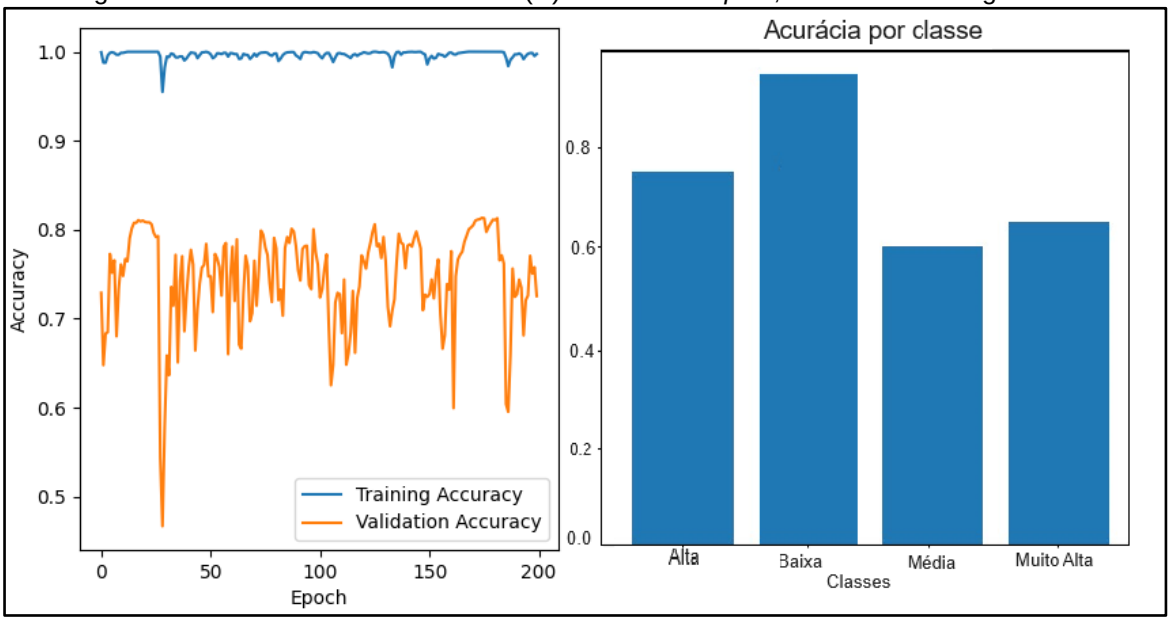
Figura 83. Matriz de Confusão Referente ao modelo (C): com *FSL/Reptile, Shot* =1 nas imagens *RGB* 26



Fonte: O autor (2023).

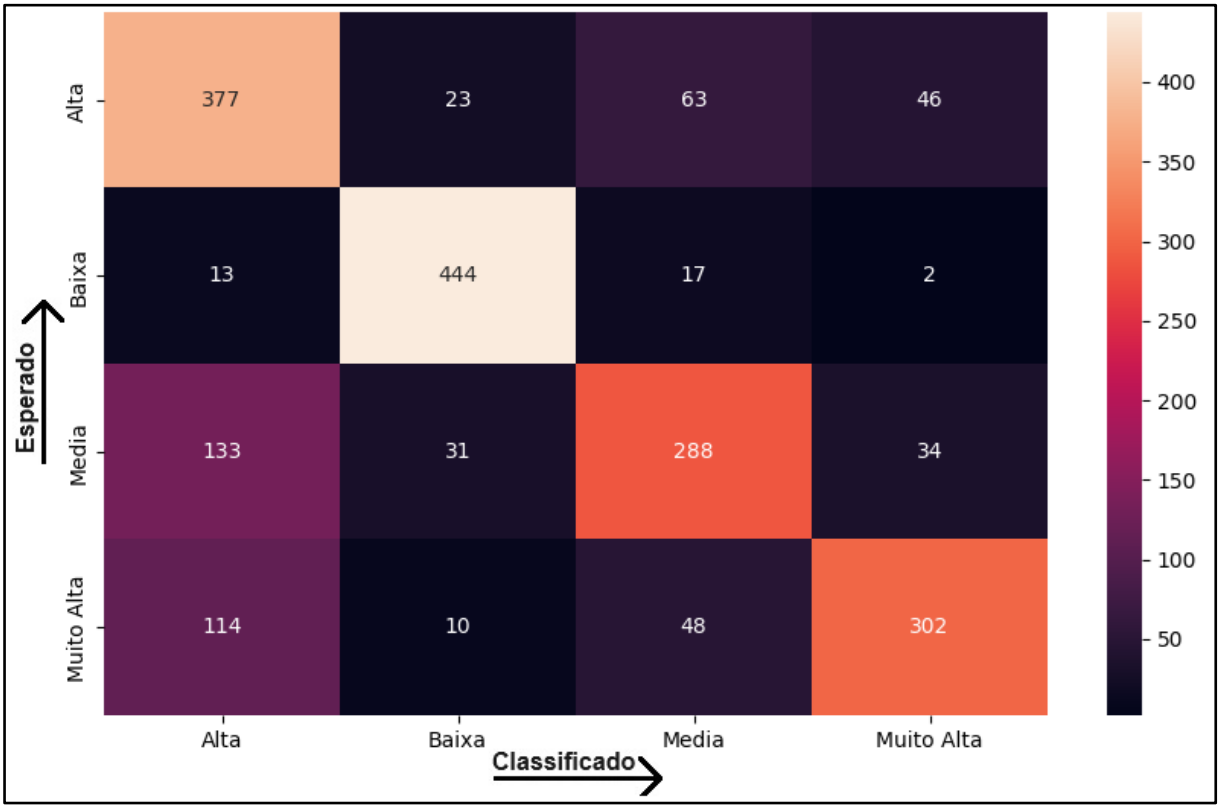
Configurado para 5 *shots*, identifica-se uma diferença substancial entre as imagens *RGB* utilizadas. O modelo obteve uma acurácia de 72.5 quando aplicado em imagens *RGB* 10 (Figura 84) e matriz de confusão na (Figura 85), porém a acurácia diminuiu para 67.3 quando o mesmo foi aplicado em imagens *RGB* 26 (Figura 86) com a sua matriz de confusão na (Figura 87).

Figura 84. Acurácia = 72.5 no modelo (C): com *FSL/Reptile*, *Shot* =5 nas imagens *RGB* 10



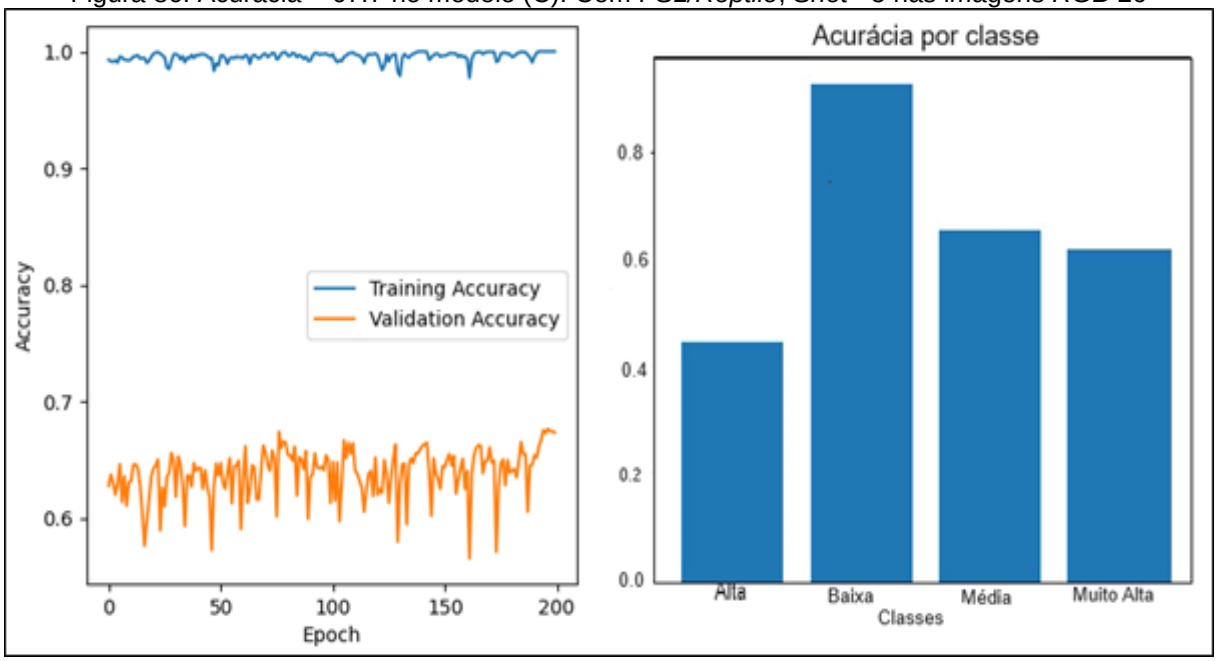
Fonte: O autor (2023).

Figura 85. Matriz de Confusão Referente ao modelo (C): com *FSL/Reptile, Shot* =5 nas imagens *RGB* 10



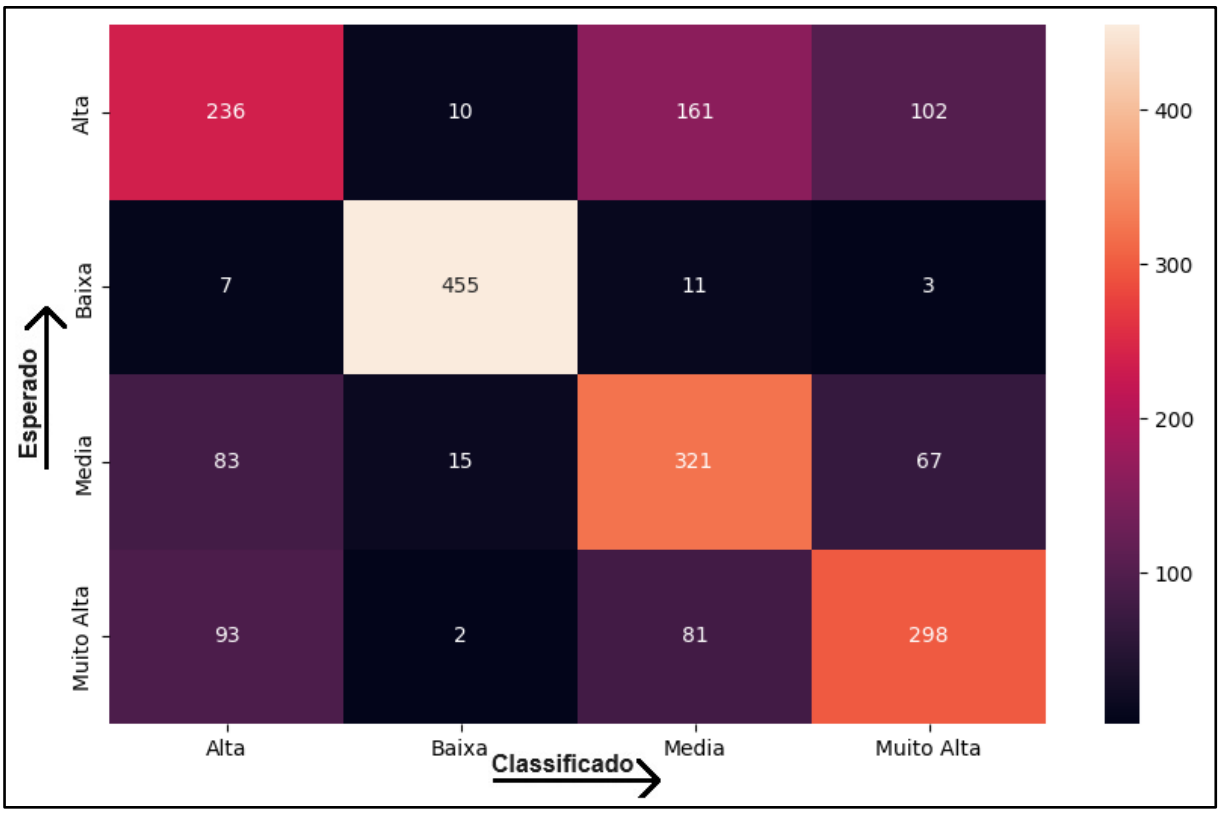
Fonte: O autor (2023

Figura 86. Acurácia = 67.7 no modelo (C): Com *FSL/Reptile*, *Shot* =5 nas imagens *RGB* 26



Fonte: O autor (2023).

Figura 87. Matriz de Confusão referente ao modelo (C): Com *FSL/Reptile*, *Shot* =5 nas imagens *RGB* 26

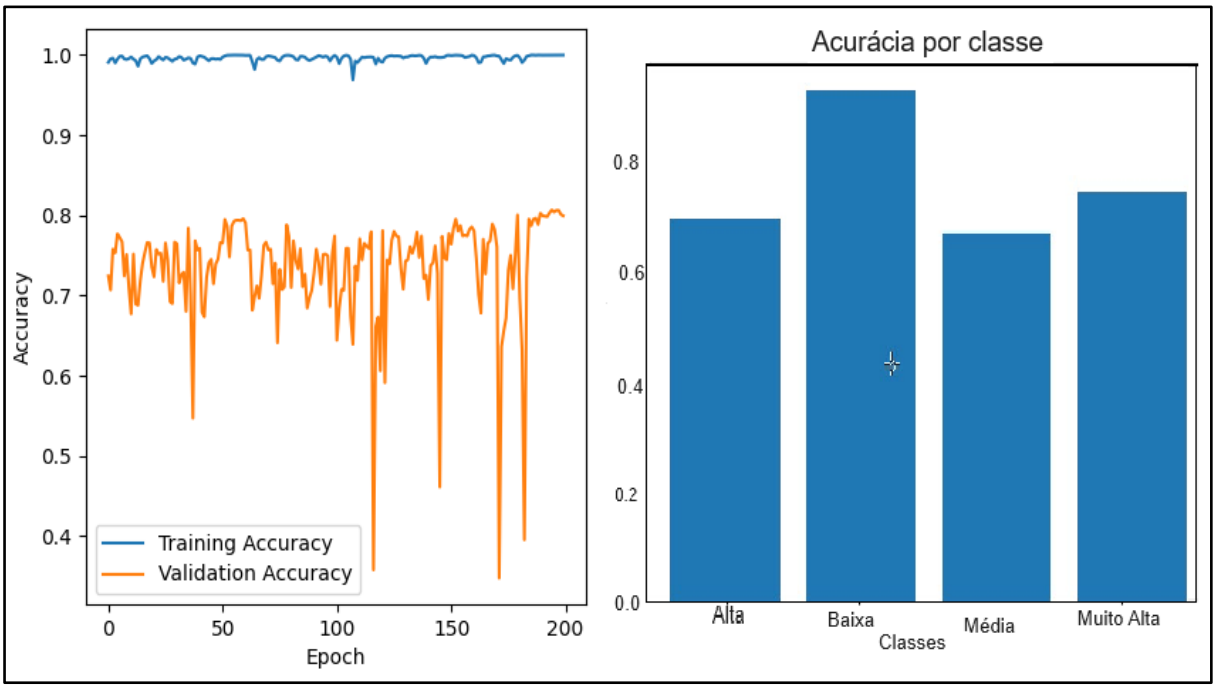


Fonte: O autor (2023).

O desempenho do Modelo (C): *DenseNet121* com *FSL/Reptile* em diferentes em *shots* de 10 e 15 o modelo alcançou acurácia de 80.0 em imagens *RGB* 10 (Figura 88), respectivamente confirmada pela sua matriz de confusão na (Figura 89)

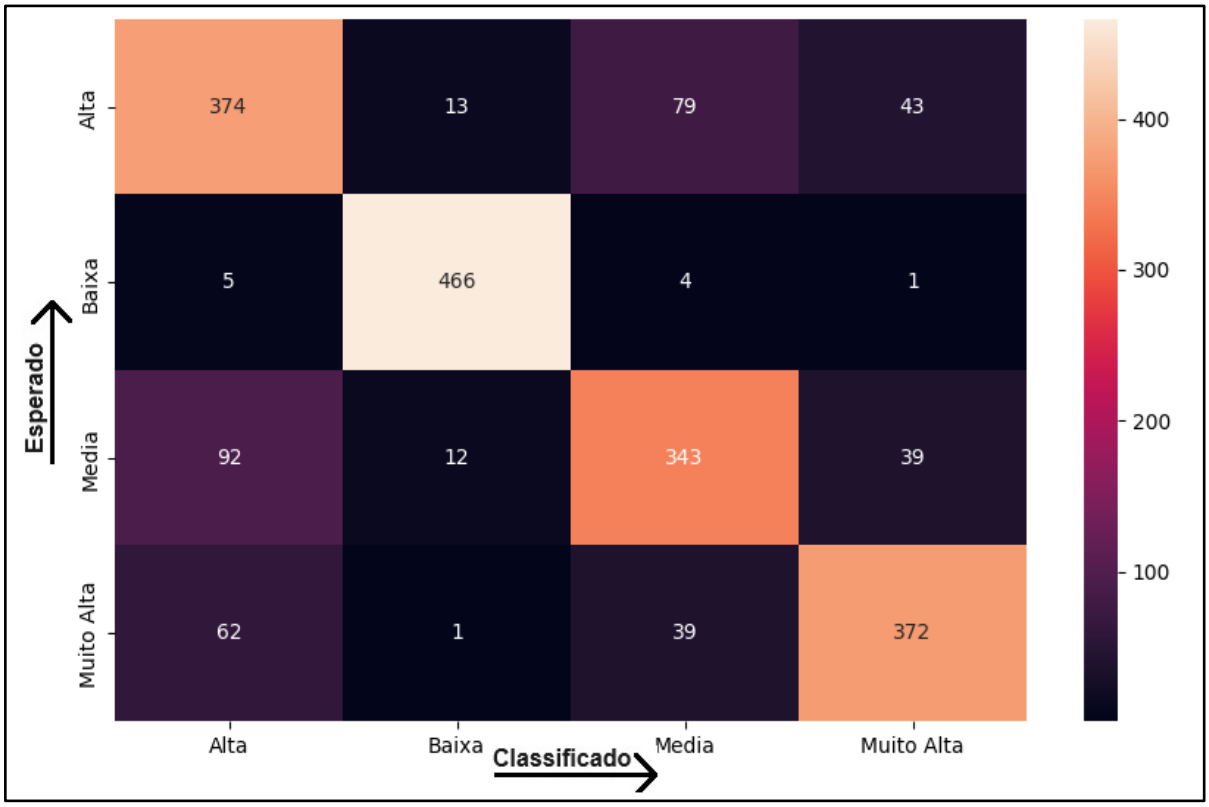
e 77.4 em imagens *RGB* 26 (Figura 90) sendo a sua matriz de confusão na (Figura 91). Aumentando para 15 *shots*, o modelo demonstrou um desempenho similar, assim atingindo 80.5 de acurácia em imagens *RGB* 10 (Figura 92), respectivamente com a sua matriz de confusão na (Figura 93) e chegando a 81.3 de acurácia em imagens *RGB 26* (Figura 94) sendo este o maior resultado de todos, representado também pela sua matriz de confusão na (Figura 95).

Figura 88. Acurácia = 80.0 no modelo (C): Com *FSL/Reptile*, *Shot* =10 nas Imagens *RGB* 10



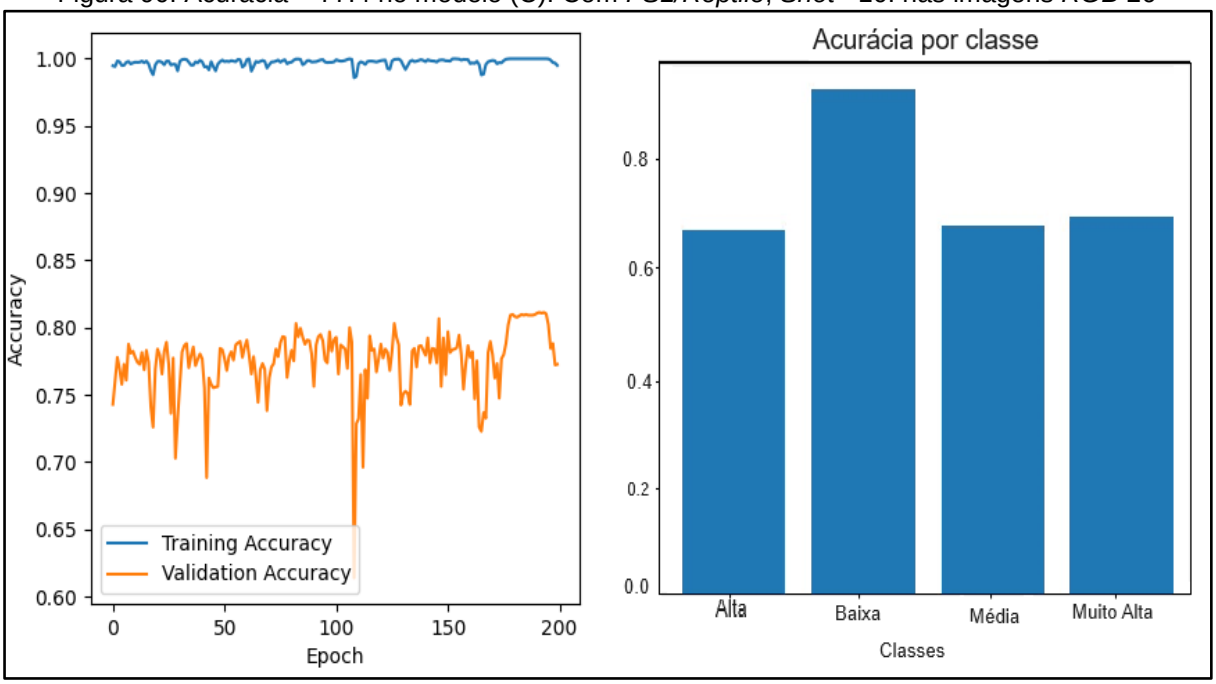
Fonte: O autor (2023).

Figura 89. Matriz de Confusão referente ao modelo (C): Com *FSL/Reptile*, *Shot* =10 nas imagens *RGB* 10



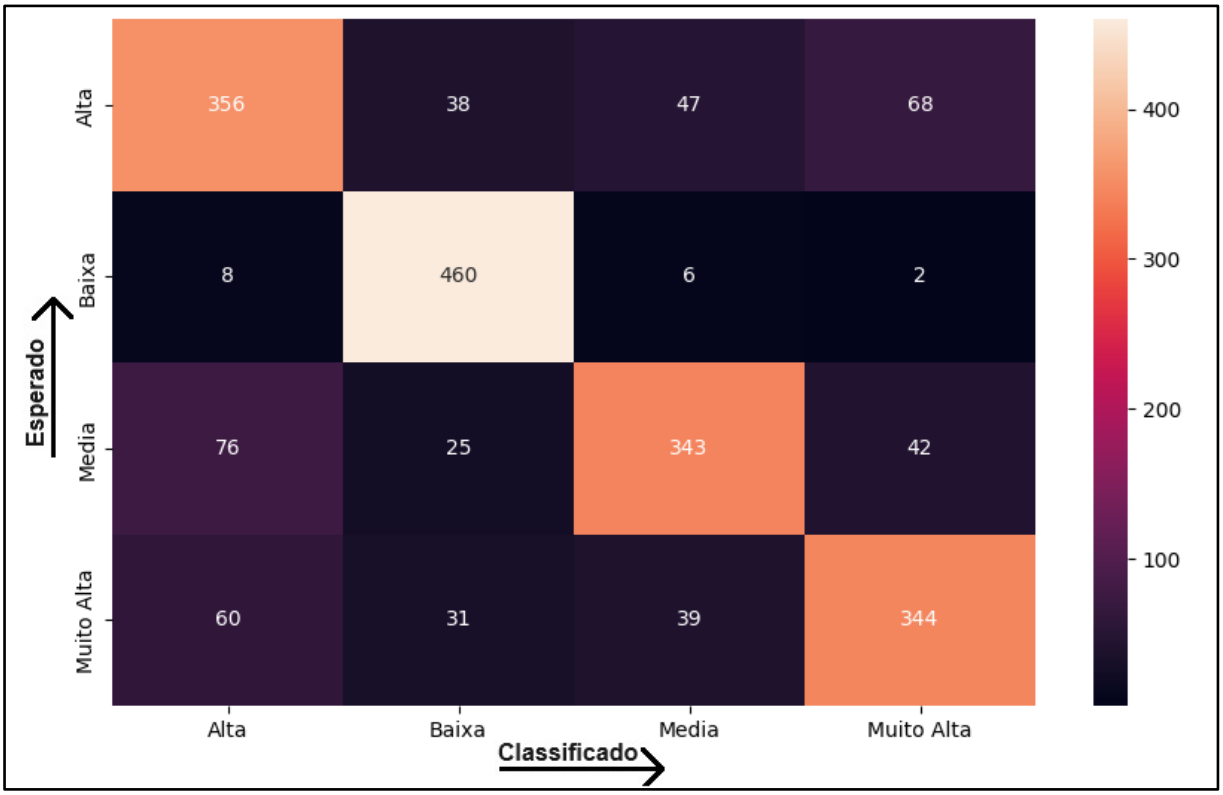
Fonte: O autor (2023).

Figura 90. Acurácia = 77.4 no modelo (C): Com *FSL/Reptile*, *Shot* =10. nas imagens *RGB* 26



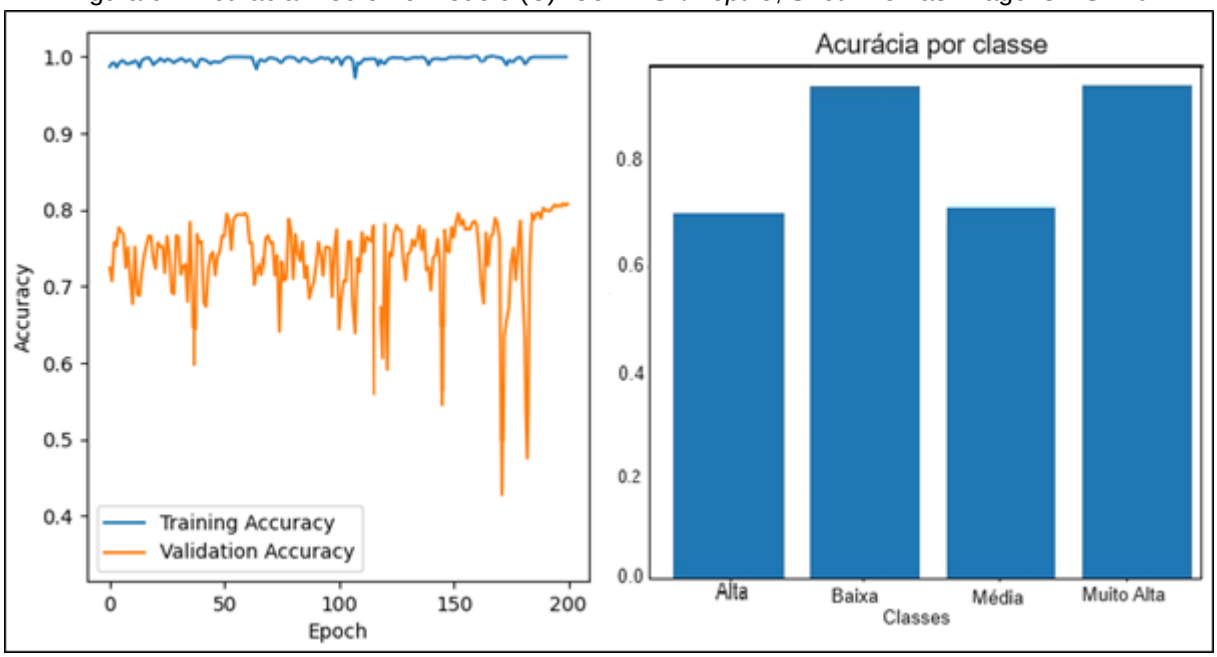
Fonte: O autor (2023).

Figura 91. Matriz de Confusão referente ao modelo (C): com *FSL/Reptile*, *Shot* =10. nas imagens *RGB* 26



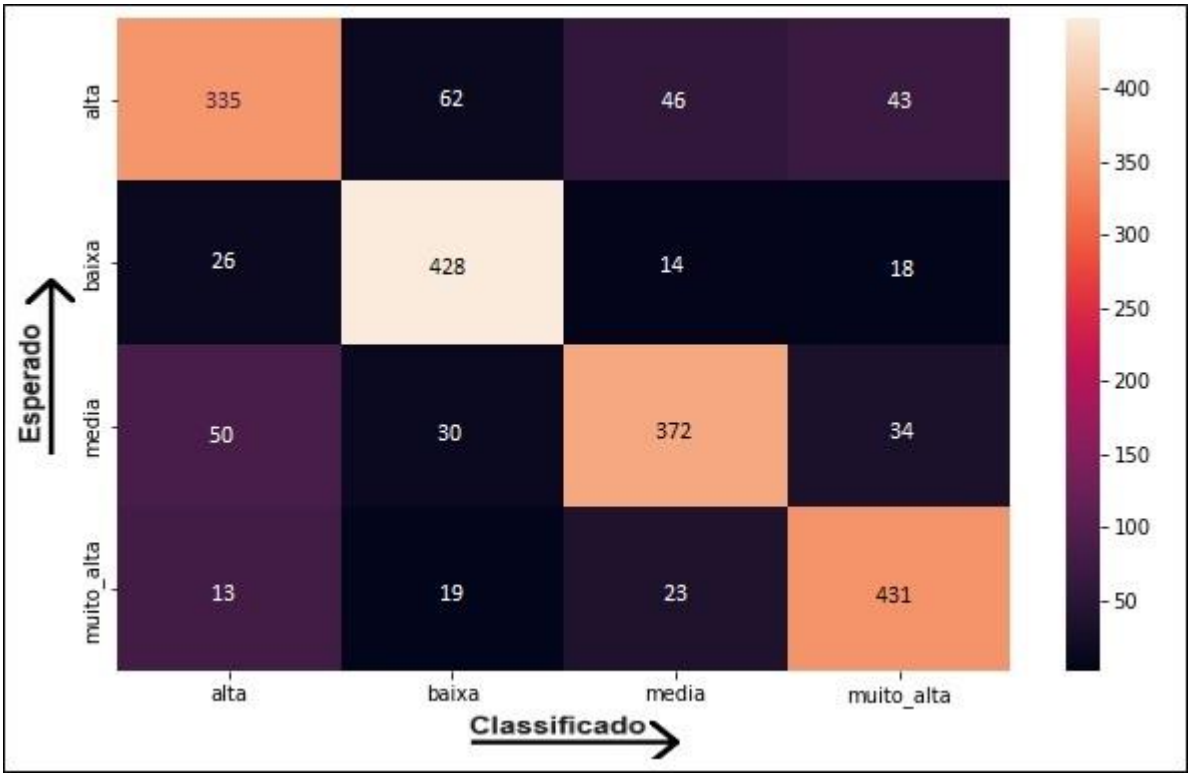
Fonte: O autor (2023).

Figura 92. Acurácia = 80.5 no modelo (C): Com *FSL/Reptile*, *Shot* =15 nas imagens *RGB10*



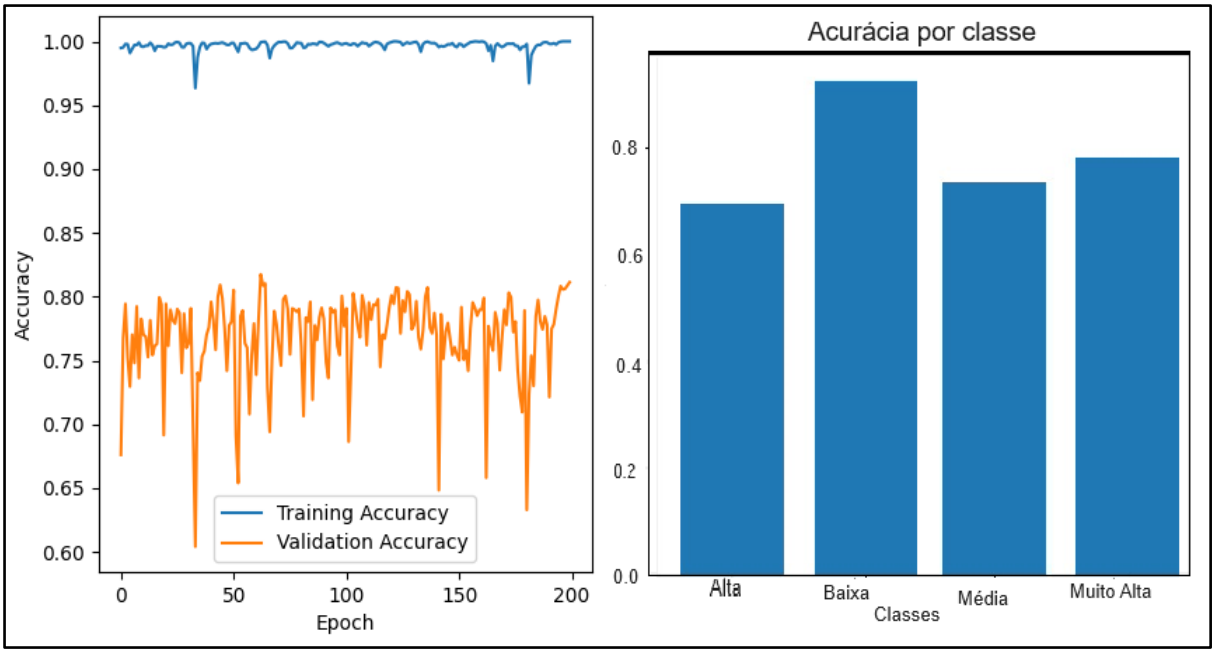
Fonte: O autor (2023).

Figura 93. Matriz de Confusão referente ao modelo (C): com *FSL/Reptile*, *Shot* =15. nas imagens *RGB* 10



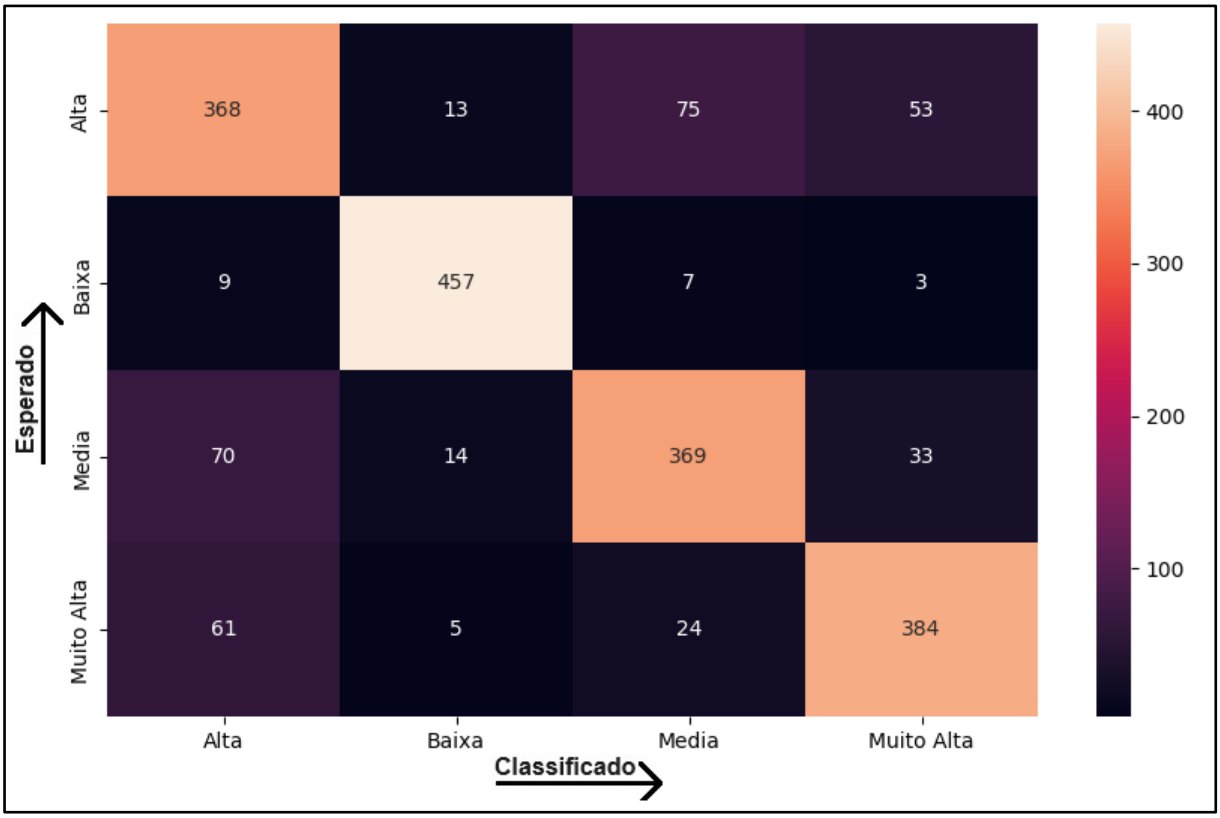
Fonte: O autor (2023).

Figura 94. Acurácia = 81.3 no modelo (C): Com *FSL/Reptile*, *Shot* =15. nas imagens *RGB* 26



Fonte: O autor (2023).

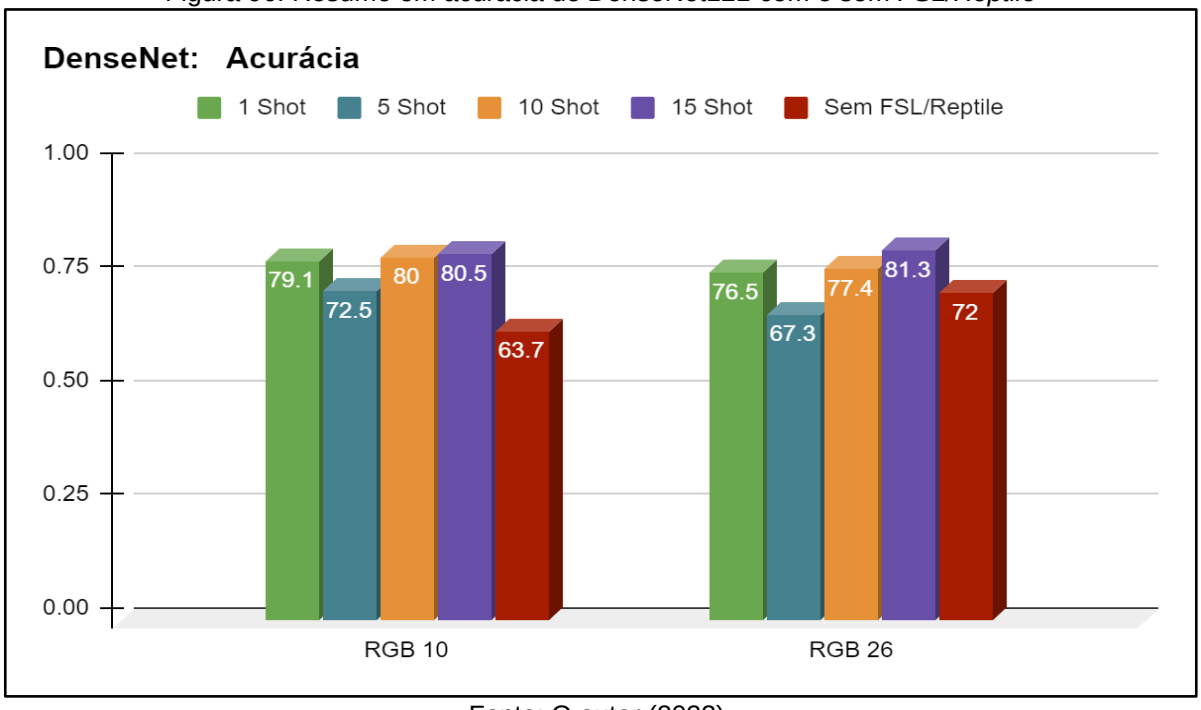
Figura 95. Matriz de Confusão referente ao modelo (C): Com *FSL/Reptile*, *Shot* =15 nas imagens *RGB* 26



Fonte: O autor (2023)

Os resultados para os modelos (C): *DenseNet121* mostraram que as melhores acurácias foram encontradas nesse modelo na faixa de 80.0. Na (Figura 96) está o resumo dos resultados para o modelo representados em forma gráfica.

Figura 96. Resumo em acurácia de *DenseNet121* com e sem *FSL/Reptile*

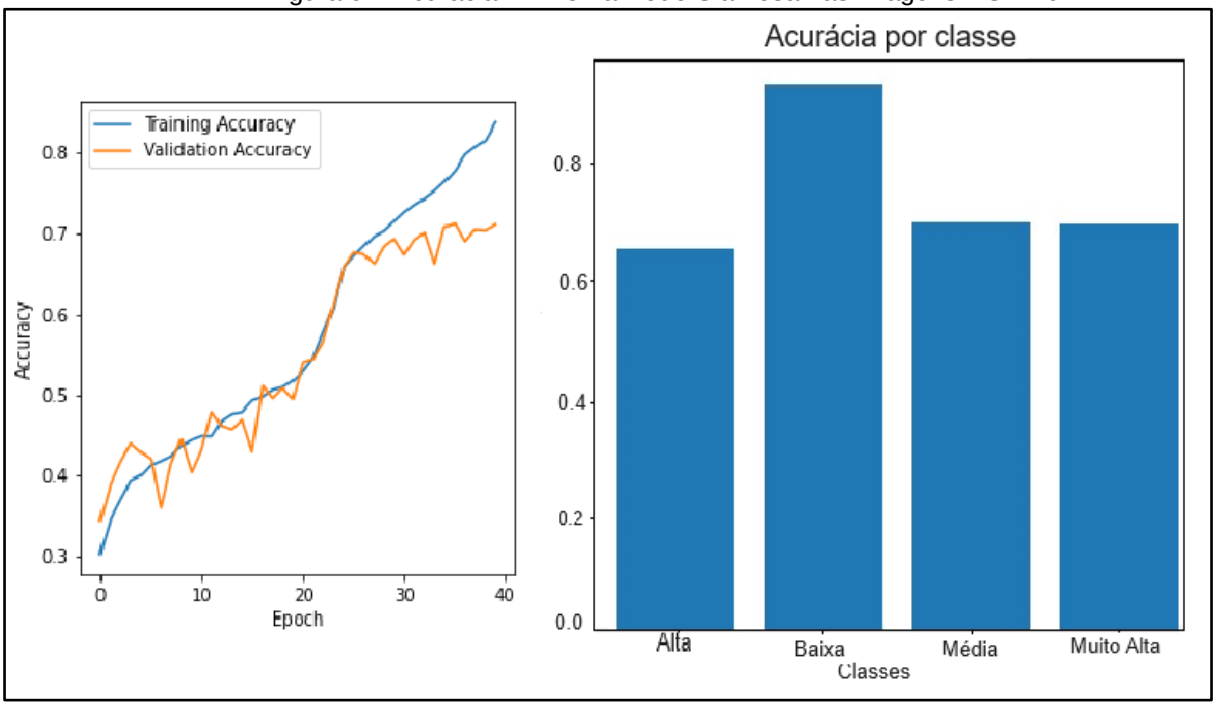


Fonte: O autor (2023).

5.4 MODELO DE APRENDIZADO: REDE SIAMESA

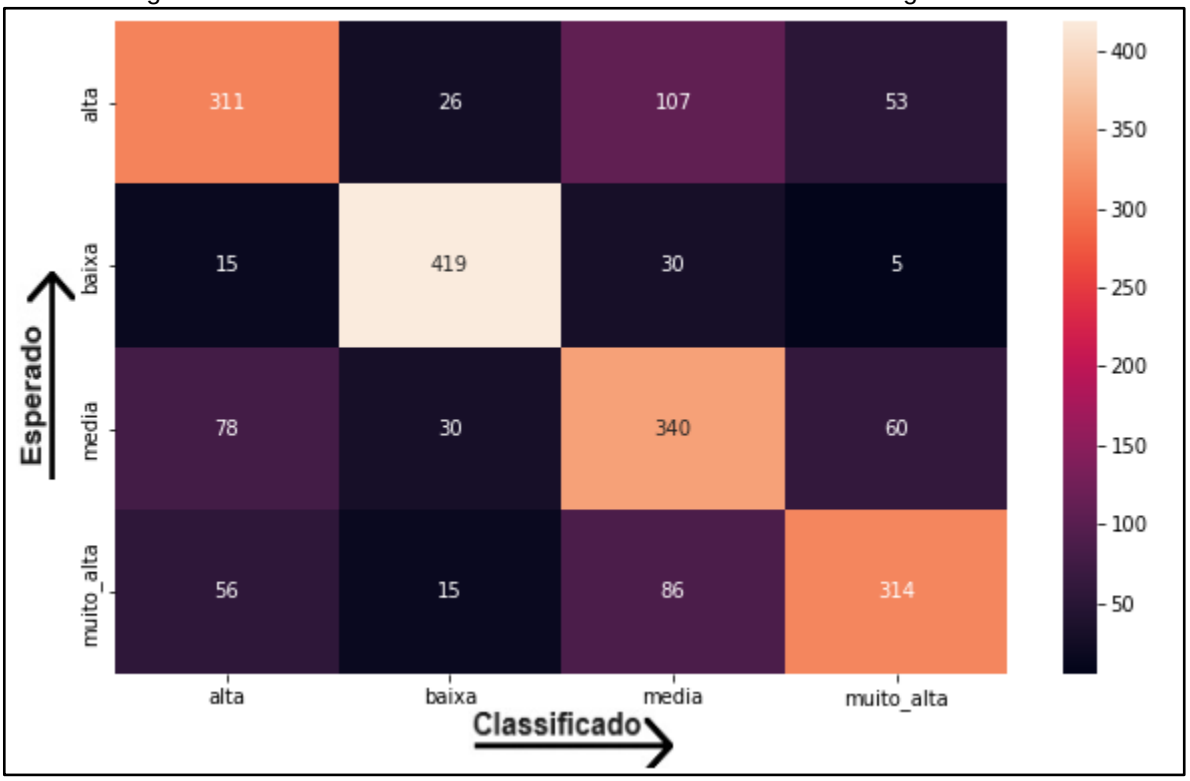
O desempenho do modelo de aprendizagem baseado na Rede Siamesa, teve uma variação na acurácia dependendo das imagens *RGB* utilizadas. O modelo alcançou acurácia de 71.1 quando treinado em imagens *RGB* 10 (Figura 97) representado pela sua matriz de confusão na (Figura 98). No entanto, sua performance melhorou, atingindo uma acurácia de 74.1, ao ser treinado em imagens *RGB* 26 (Figura 99), com sua matriz de confusão representada respectivamente na (Figura 100).

Figura 97. Acurácia = 71.3 na Rede Siamesa nas imagens *RGB* 10



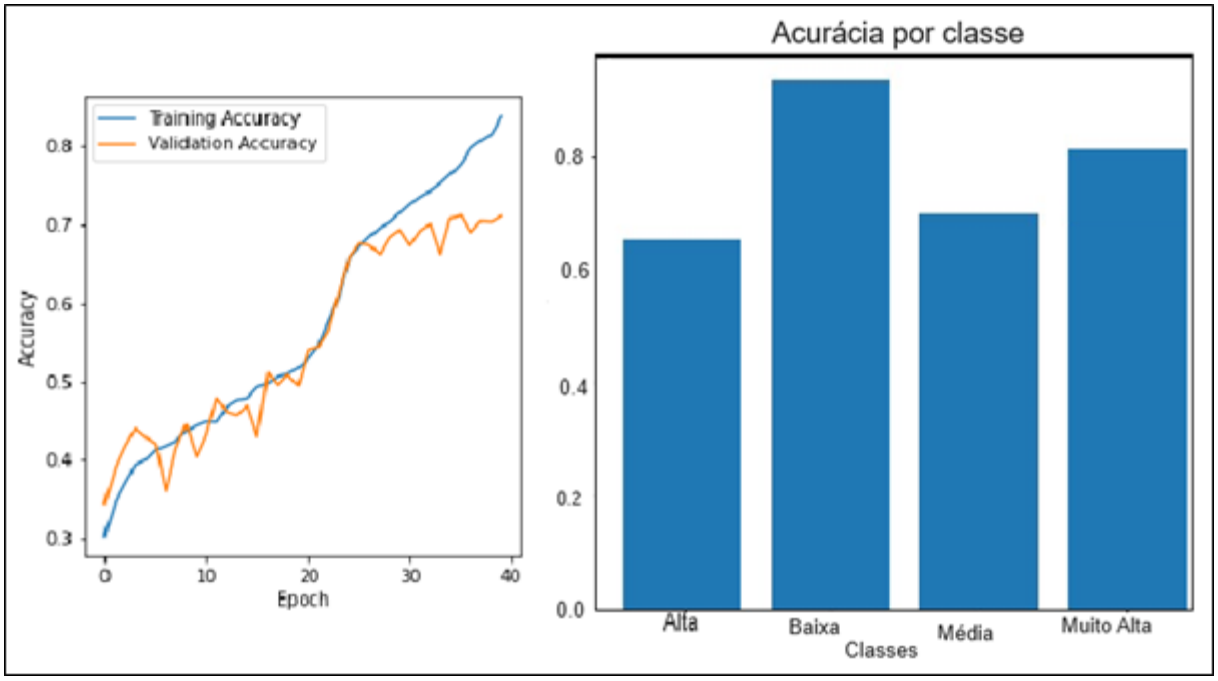
Fonte: O autor (2023).

Figura 98. Matriz de Confusão referente a Rede Siamesa nas imagens *RGB 10*



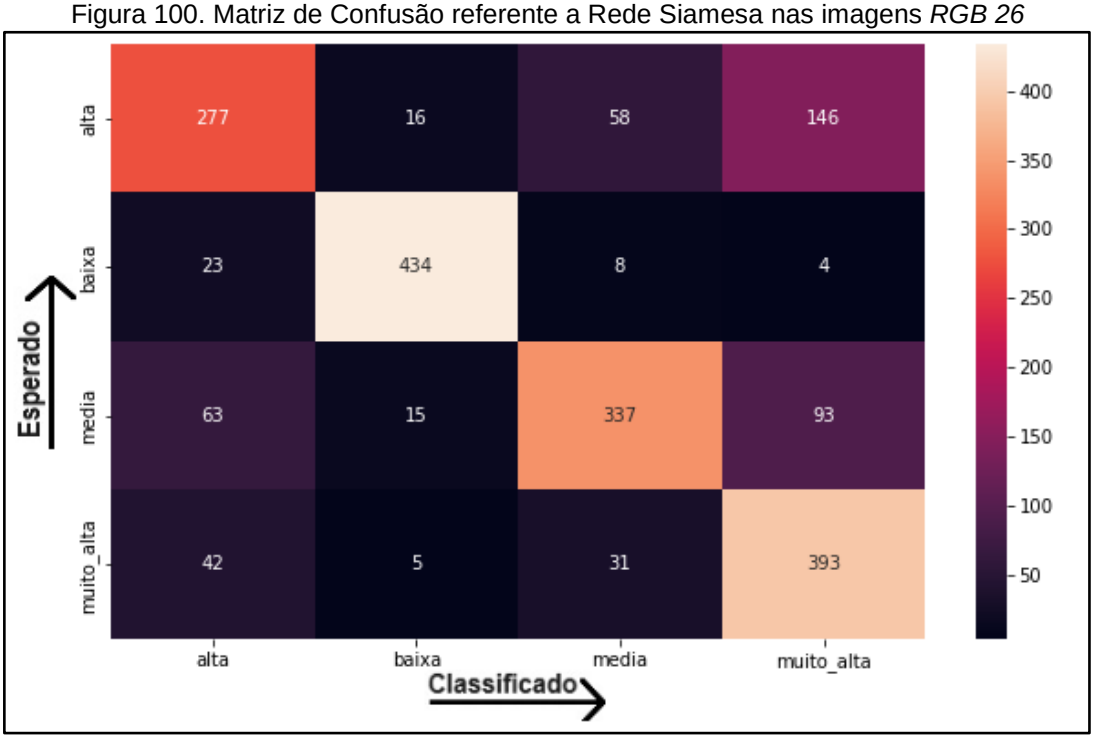
Fonte: O autor (2023).

Figura 99 Acurácia = 74.5 na Rede Siamesa nas imagens *RGB* 26



Fonte: O autor (2023).

Figura 100. Matriz de Confusão referente a Rede Siamesa nas imagens *RGB 26*

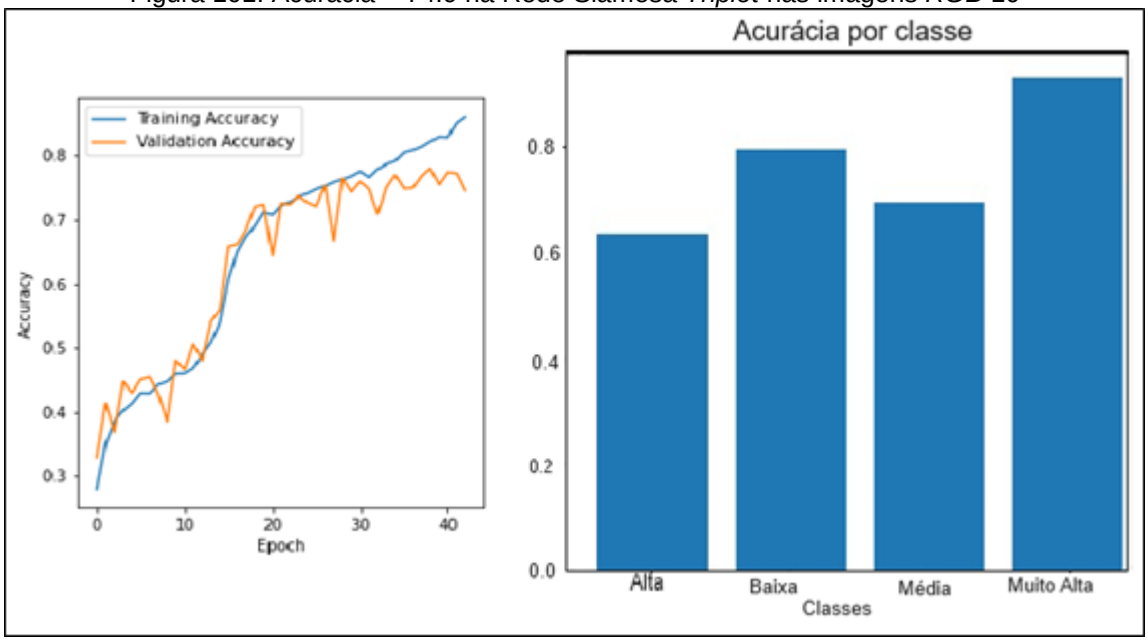


Fonte: O autor (2023).

5.5 MODELO DE APRENDIZADO: REDE SIAMESA *TRIPLET*

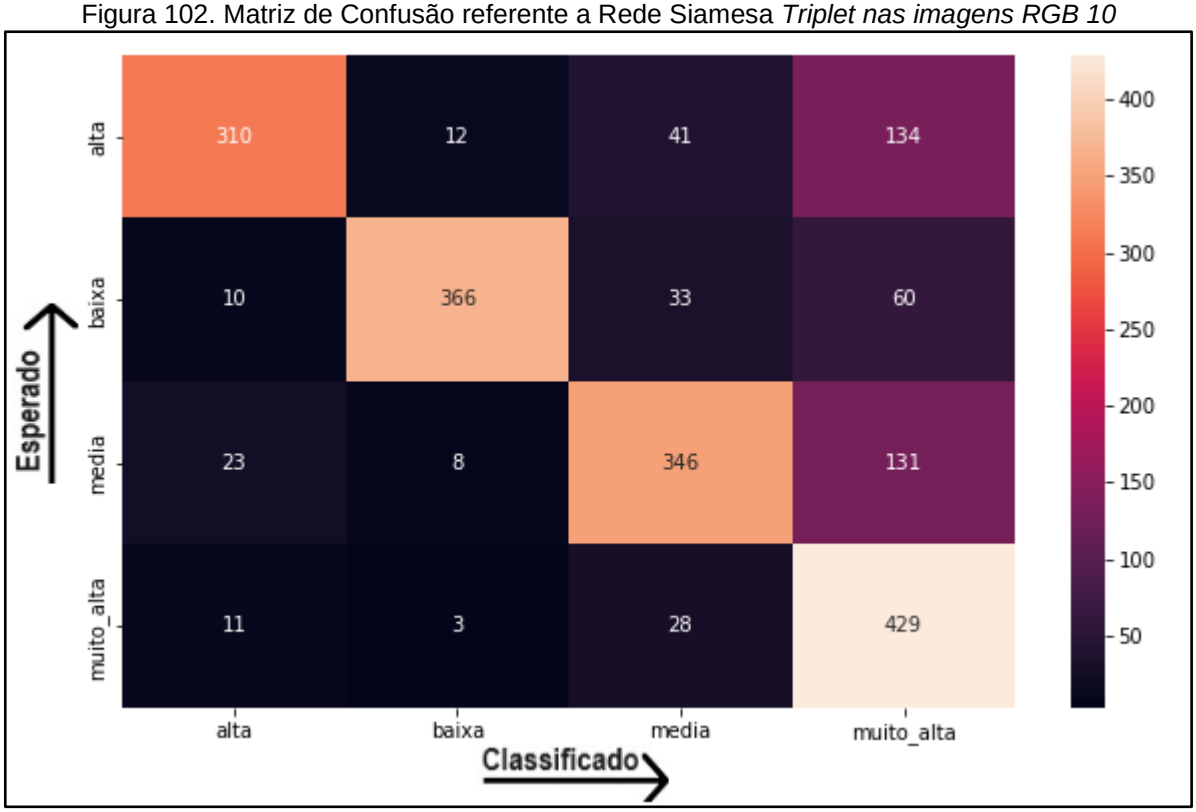
O desempenho do modelo baseado na Rede Siamesa *Triplet* revelou diferenças na acurácia, no conjunto das imagens *RGB* utilizadas. Com imagens *RGB* 10, o modelo apresentou uma acurácia de 74.9 (Figura 101), representado pela sua matriz de confusão na (Figura 102). No entanto, um resultado superior foi alcançado com uma acurácia de 78.4, ao utilizar imagens *RGB* 26 (Figura 103) com sua matriz de confusão sendo representada na (Figura 104).

Figura 101. Acurácia = 74.9 na Rede Siamesa *Triplet* nas imagens *RGB* 10



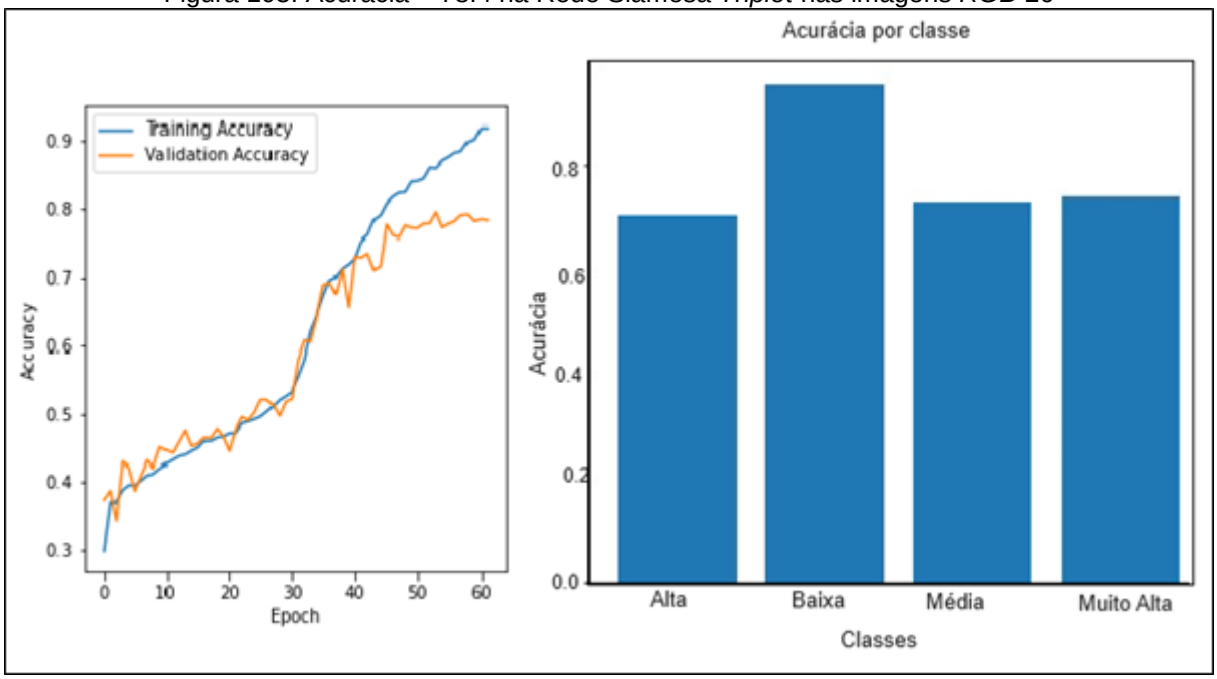
Fonte: O autor (2023).

Figura 102. Matriz de Confusão referente a Rede Siamesa *Triplet nas imagens RGB 10*



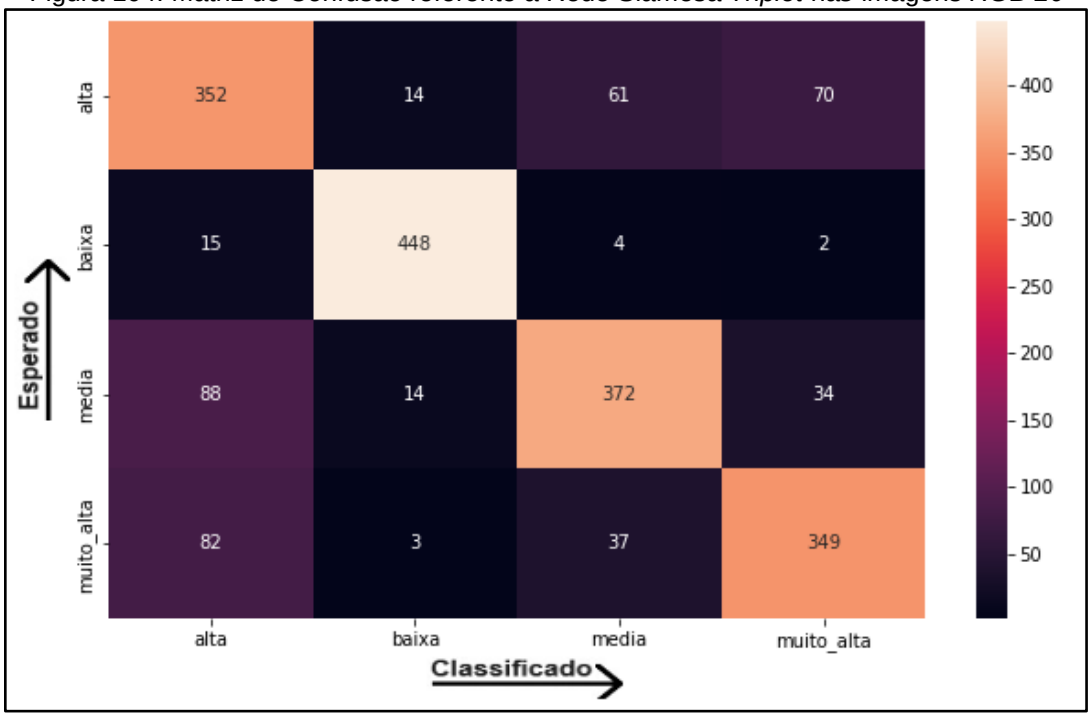
Fonte: O autor (2023).

Figura 103. Acurácia = 78.4 na Rede Siamesa *Triplet* nas imagens *RGB* 26



Fonte: O autor (2023).

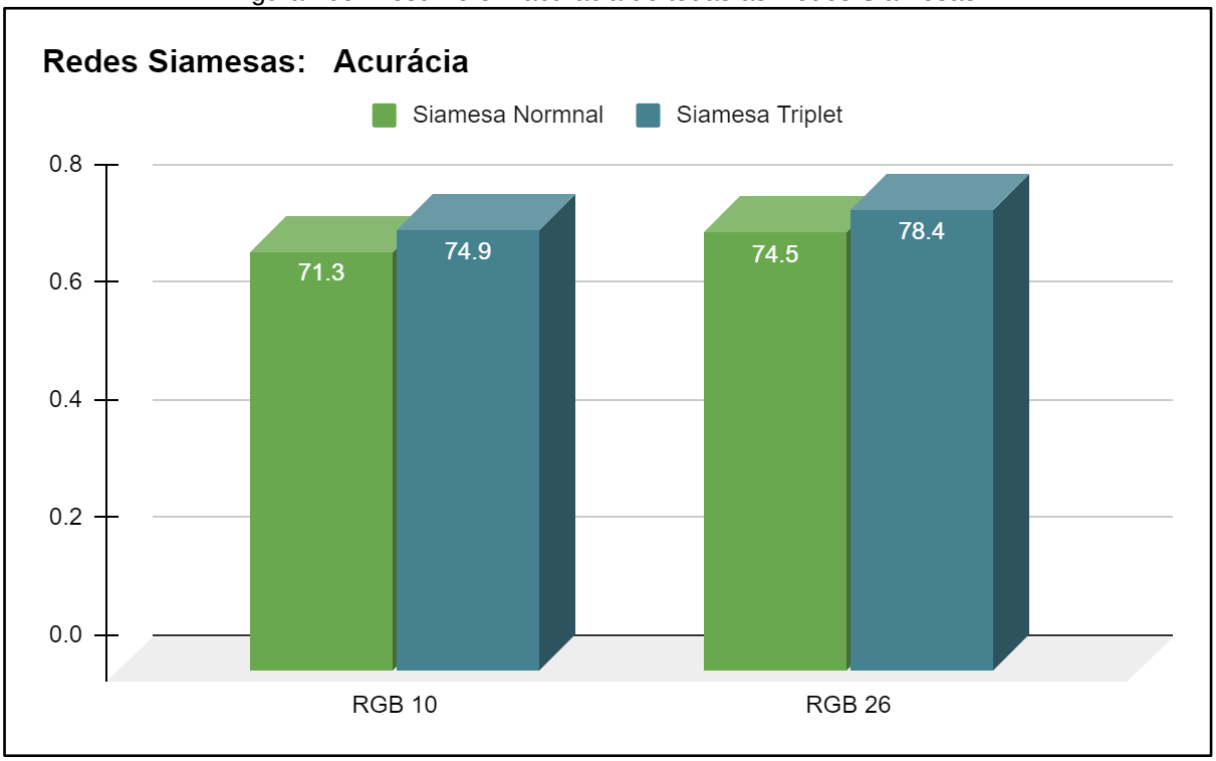
Figura 104. Matriz de Confusão referente a Rede Siamesa *Triplet* nas imagens *RGB 26*



Fonte: O autor (2023).

De modo geral observando a (Figura 105), nota-se que as redes siamesas que utilizaram em sua arquitetura principal a *CNN* modificada desenvolvida durante esta pesquisa também entregaram um bom resultado com os seus 78.4 de acurácia máxima alcançados através da Rede Siamesa *Triplet* nas imagens RGB 26.

Figura 105. Resumo em acurácia de todas as Redes Siamesas.



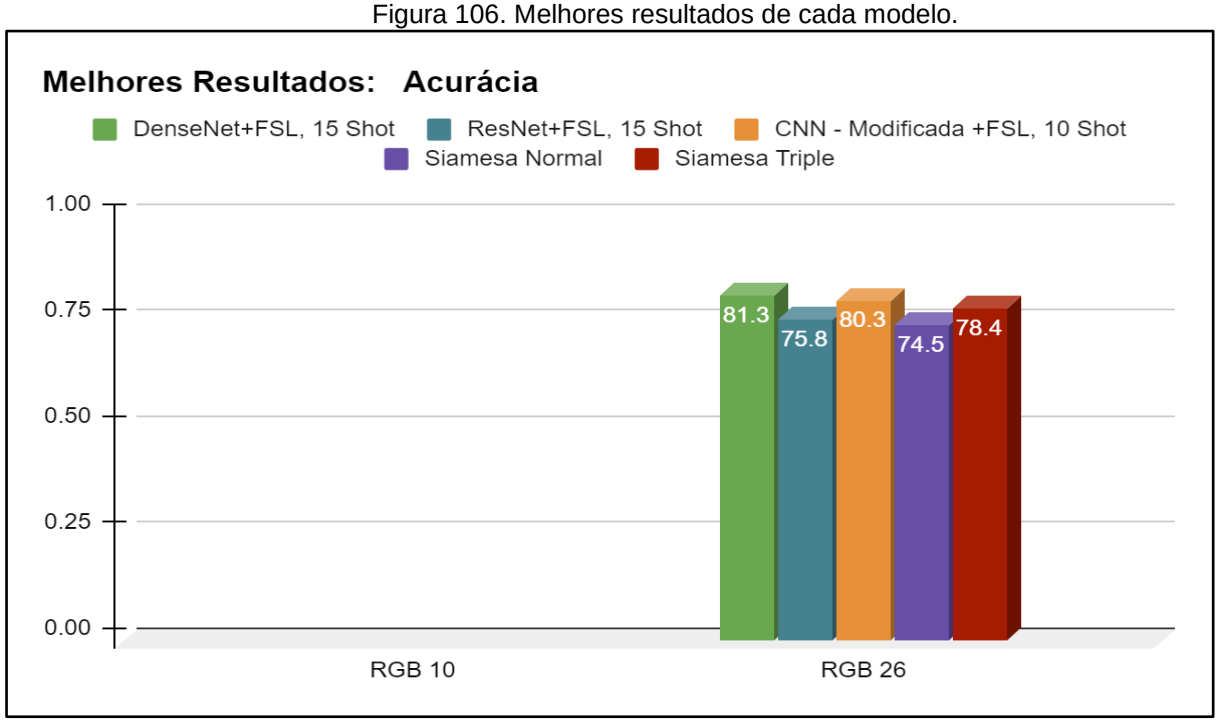
Fonte: O autor (2023).

6 DISCUSSÃO DOS RESULTADOS

Observa-se um desempenho superior dos modelos que utilizaram *FSL* em relação aos modelos sem esse mérito. Este resultado corrobora com a hipótese de que a técnica de *FSL* com o algoritmo *Reptile* contribui realmente para a melhora na acurácia da classificação de imagens, especialmente em um cenário onde a similaridade entre imagens é alta.

No caso da arquitetura *DenseNet121*, as melhorias na acurácia foram de 16,8% para resolução *RGB* 10 e 9,3% para resolução *RGB 26*. Para a *ResNet50*, as melhorias chegaram a 23,2% para resolução *RGB 10* e 10,4% para *RGB 26*. Todavia, dentre todas as arquiteturas, a *CNN* modificada (modelo A) também obteve melhorias, proporcionando uma escalada na acurácia de 18% para *RGB 26* e 5,5% para *RGB 10*. Os melhores resultados foram compilados e evidenciados na (Figura 106) em formato gráfico para demais observações.

Figura 106. Melhores resultados de cada modelo.

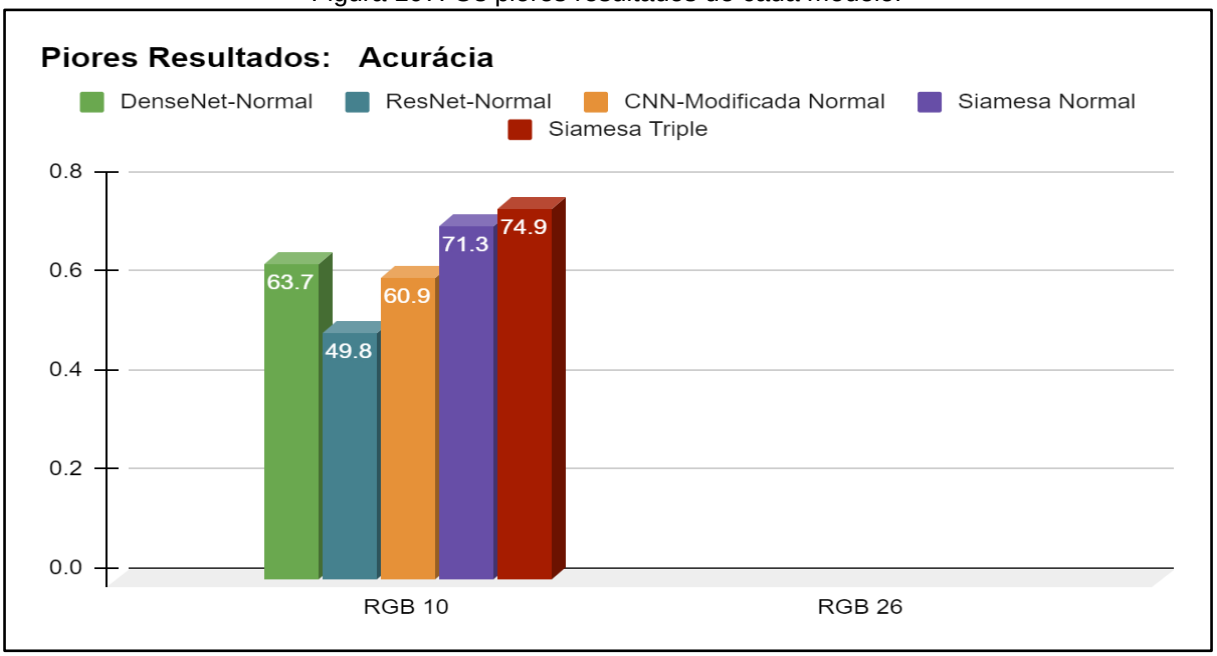


Fonte: O autor (2023).

O aumento no número de *K-shots* não garantiu uma melhora na acurácia em todos os casos. Isso sugere que existe um ponto ideal de equilíbrio entre a quantidade de *K-shots* e a acurácia, e que simplesmente elevar o número de *K-shots* indiscriminadamente pode não ser a melhor estratégia.

O desempenho das Redes Siamesas, tanto na versão normal quanto na versão tripla, reforça a capacidade dessas arquiteturas em tratar imagens semelhantes, capturando as particularidades de cada categoria.

Figura 107. Os piores resultados de cada modelo.



Fonte: O autor (2023).

Em questão de resolução destacou-se a resolução *RGB* 26 com os melhores resultados de acurácia nos modelos testados, enquanto todos os resultados das *RGB* 10 ficaram com os piores resultados em relação aos mesmos algoritmos executados. Evidenciado nos resumos gerais de resultados das (Figuras 106 e 107).

7 CONCLUSÃO

Os resultados obtidos sustentam a viabilidade do uso de *FSL* na classificação de imagens de produtividade da soja obtidas por *RPA*. Essencialmente, demonstram que a integração de técnicas de *Meta-Learning* e *FSL* pode potencializar a precisão dos modelos de *DL*, configurando-se como uma abordagem promissora para situações com poucas amostras.

A utilização do *Meta-Learning* com o algoritmo *Reptile* contribuiu para a eficácia do *FSL*, permitindo a rápida adaptação dos modelos às tarefas de classificação. Além disso, a otimização dos hiperparâmetros com a técnica G*rid Search*, bem como a escolha das resoluções e o número de *K-shots* são fatores consideráveis que tiveram impacto na qualidade do desempenho desses modelos.

Os modelos mais eficientes conforme a acurácia média, foram a *DenseNet*121 e a *CNN* modificada (Modelo A) com *FSL*, principalmente para imagens com resolução de 26cm/*px*. Os modelos de Redes Siamesas, tanto a normal como a tripla, também apresentaram desempenhos dignos de nota, destacando a importância da Aprendizagem Métrica no campo de *Meta-Learning.*

A resolução espacial de 26 cm/*px* esteve presente nos modelos com melhores resultados de acurácia, levando a conclusão de que, independentemente do modelo utilizado, essa resolução contribuiu para a obtenção de uma acurácia maior na classificação de imagens de produtividade da soja obtidas por *RPA*.

O principal desafio foi o processamento obrigando a limitar o número de *K-Shots* para o máximo de 15, devido ao tempo de processamento demandado, além disso alguns modelos em pleno funcionamento acabaram sendo totalmente descartados por conta dessa limitação, se tratavam de modelos de redes siamesas com maior número de camadas densas e camadas convolucionais, chegando a dobrar e triplicar o custo de processamento em comparação a *DenseNet*121 que foi o modelo mais caro computacionalmente.

Futuramente a exploração de imagens contendo o infravermelho próximo é pertinente para pesquisas correlatas, essas imagens podem ser utilizadas em outras abordagens envolvendo técnicas de *Meta-Learning* e FSL para a criação de novos modelos de IA. Outra sugestão é a aplicação de IA em trabalhos focados especificamente para estimar a produtividade da soja através de imagens, assim

como determinar os principais parâmetros para a coleta de dados para essa finalidade.

REFERÊNCIAS

AGHDAM, H. H; HERAVI, E.J., **Guide to Convolutional Neural Networks A pratical Apllication to Traffic – Sign Detection and Classification**. 1ª ed. Springer, 2017.

ANAC. **Rpas - Sistemas De Aeronaves Remotamente Pilotadas**. Disponível em: <http://www2.anac.gov.br/rpas/>. Acesso em: 11 jan. 2018.

ANTUNES, G. T. **Matemática em Pixels: o ensino de funções aplicado a criação de filtros de imagens digitais.** 2021 Curso de Licenciatura em Matemática Rio Grande, Rio Grande do Sul, Brasil Maio, 2021 Universidade Federal do Rio Grande – FURG Instituto de Matemática, Estatística e Física – IMEF, 2021.

BORRERO, I. P.; ARIAS, M.E.G. **Deep Learning:** fundamentos, teoria y aplicación. Hueha: Universidad de Hueha, 2021.

BUDUMA, N., LOCASCIO, N. **Fundamentals of Deep Learning** 2ª Ed. O´Reeilly. EUA, 2017.

CASCARDI, A.; MICELLI, F.; AIELLO, M. A. An artificial neural networks model for the prediction of the compressive strength of FRP-confined concrete circular columns. Engineering Structures, 2017. **Elsevier BV**, v. 140, p. 199–208, jun. 2017.

CHAVES, A. A. et al. Using UAVs and digital image processing to quantify areas of soil and vegetation. **Journal of Physics: Conference Series**, v. 633, p.12112, 21 set. 2015.

FACCO, F. L. B. **O uso de técnicas de sensoriamento remoto e das aeronaves remotamente pilotadas na agricultura: contribuições estatísticas e geotecnológicas**. 2022. (Dissertação mestrado) - Universidade Federal de Santa Maria, Centro de Ciências Naturais e Exatas, Programa de Pós-Graduação em Geografia, RS, 2022.

FERNANDES, P. **Corn yield estimates (Zea Mays L.) through images obtained by unmanned aerial vehicle**. 2016. 79 f. Dissertação (Mestrado em Agronomia) - Universidade Federal de Santa Maria, Santa Maria, 2016.

FREITAS, S. O. de. **Utilização de um Modelo Baseado em Redes Neurais para a Precificação de Opções**. 2001. (Dissertação Mestrado) — Universidade Federal de Minas Gerais, Belo Horizonte, 2001.

GARCIA, S.; LUENGO, J.; HERRERA F. Data Processing in Data Mining. **Springer**, 2015.

GARG, S.; SINGH, P. An aggregated loss function based lightweight few shot model for plant leaf disease classification. **Multimed Tools Appl** 82, 23797–23815 (2023).

GERKE, T. **Mineração de dados de imagens obtidas com aeronave remotamente pilotada para estimativa de produtividade do trigo**. 2017. (Dissertação Mestrado em Computação para Tecnologias em Agricultura) - UNIVERSIDADE ESTADUAL DE PONTA GROSSA, Ponta Grossa, 2017.

GÉRON, A. **Mãos à obra:** aprendizado de máquina com Scikit-Learn, *Keras* &TensorFlow: Conceitos, ferramentas e técnicas para a construção de sistemas inteligentes. Alta Books, Rio de Janeiro, 2019.

GIOVANIS, D. G. et al. Bayesian updating with subset simulation using artificial neural networks. Computer Methods in Applied Mechanics and Engineering, 2017. **Elsevier BV**, v. 319, p. 124–145, jun. 2017.

GLOROT, X., BORDES, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In: **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**, pages 315–323.

GOMES, J. C. **Uma arquitetura de few-shot learning para classificação de insetos na agricultura usando poucas amostras.** 2022. xii, 63 f., il. Dissertação (Mestrado em Sistemas Mecatrônicos) — Universidade de Brasília, Brasília, 2022.

GONZALEZ, R.; WOODS, R. **Processamento Digital De Imagens**. ADDISON WESLEY BRA, 2009. ISBN 9788576054016. Disponível em: <https: //books.google.com.br/books?id=r5f0RgAACAAJ>. Acesso em: 10 jan. 2024.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning.** Cambridge: Mit Press, p. 12-18, 192-200. 2016.

GUI, J.; XU, H.; FEI, J. Non-Destructive Detection of Soybean Pest Based on Hyperspectral Image and Attention-ResNet *Meta-Learning* Model. **Sensors**. 2023; 23(2):678. Acesso em: 20 nov. 2023.

HAYKIN, S. **Redes Neurais:** Princípios e Práticas. 2. ed. [S.l.]. Porto Alegre: Bookman, 2007.

HAYKIN, S. **Neural Networks and Learning Machines**. 3. ed. [S.l.]. Pearson Education, 2009.

HOFFER, E.; AILON, N. Deep metric learning using triplet network. In: International workshop on similarity-based pattern recognition. [S.l.]: **Springer**, 2015. p. 84–92.

HOPE, T.; RESHEFF, Y. ; LIEDER, I. **Learning Tensoflow:** A Guide to Building Deep Learning Systems**.** O´Reilly Media, 2017.

HUANG, G. *et al.* Densely connected convolutional networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). [S.l.]: **IEEE**, 2017.

JADON, S; GARG, A. **Hands-On One-shot Learning with Python:** Learn to implemente fast and accurate deep learning models with fewer training samples using. PyTorch. Packt, 2020.

KARAMI, A.; CRAWFORD, M.; DELP, E. J. Automatic Plant Counting and Location Based on a Few-Shot Learning Technique. **IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing**, VOL. 13, 2020.

KIM, W. *et al.* One-shot classification-based tilled soil region segmentation for boundary guidance in autonomous tillage. **Computers and Electronics in Agriculture** vol 189, p.106371. 2021. Acesso em: 25 out. 2023.

KINLI, F. Deep Learning Lab Episode-3: fer2013**.** *In* FURKAN KINLI. **medium.com,** 06 abr. 2018**.** Disponível em: https://medium.com/@birdortyedi_23820/deep-learning-lab-episode-3-fer2013-c38f2e052280 . Acesso em 30 set. 2023.

KOCH, G.; ZEMEL, R.; SALAKHUTDINOV, R. Siamese neural networks for one-shot image recognition. In: **LILLE. ICML deep learning workshop**. [S.l.], 2015. v. 2, p. 0.

KRAMER, O. (2016). Scikit-Learn. In: Machine Learning for Evolution Strategies. Studies in Big Data, vol 20. **Springer**, Cham. Acesso em: 06 dez. 2023.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. lmagenet classification with deep convolutional neural networks. ln: PEREIRA, F.; BURGES, e. J. C.; BOTTOU, L.; WEINBERGER, K. Q., (Eds.) **Advances in Neural Information Processing Systems 25**, (NIPS)., p. 1097-1105, 2012.

KROHN, J.; BEYLEVELD, G.; BASSENS, C**. Deep Learning Illustrated**: A visual Interactive Guide to Artificial Intelligence. Addison-Wesley, 2019.

LECUN, Y.; BENGIO, Y.; HINTON, G.. Deep Learning. **Nature**. 521. 436-44. 10.1038/nature14539, 2015.

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

Li, M.; Yao, H.; Wang, Y. Focus nuance and toward diversity: exploring domain-specific fine-grained few-shot recognition. **Neural Comput & Applic** 35, 21275–21290 (2023).

Li, Y.;CHAO, X. Semi-supervised few-shot learning approach for plant diseases recognition. **Plant Methods**. 17. 10.1186/s13007-021-00770-1. 2021.

LI, Y.; YANG, J. *Meta-Learning* baselines and database for few-shot classification in agriculture. **Computers and Electronics in Agriculture, Elsevie**r, v. 182, p. .106055, 2021.

LIN, X.;WANG, X.; HAO, Z. Supervised learning in multilayer spiking neural networks with inner products of spike trains. Neurocomputing, 2017. **Elsevier BV**, v. 237, p. 59–70, mai. 2017.X.

LIU, X.; ZHOU, F.; LIU, J.; JIANG, L. *Meta-Learning* based prototype-relation network for few-shot classification. Neurocomputing, **Elsevier**, v. 383, p. 224–234, 2020.

LUDWING JR, O; MONTGOMERY, E. M. **Redes Neurais:** Fundamentos e Aplicações com Programas em C. Rio de Janeiro: Editora Ciência Moderna Ltda. 2007.

LUZ, G.L. da. *et al.* **Introdução ao uso de imagens aéreas para manejo da cultura do milho**. 1 ed. Curitiba: Appris, 2020.

MAIMAITIJIANG, M.; *et al.* Soybean yield prediction from UAV using multimodal data fusion and deep learning. **Remote sensing of environment,** Suíça. Vol 237, p.111599, fev 2020.

MAIONE, C. **Balanceamento de dados com base em oversampling em dados transformados**. 2020. 135 f. Tese (Doutorado em Ciência da Computação em Rede) - Universidade Federal de Goiás, Goiânia, 2020.

MARCUS, G. Deep Learning: A Critical Appraisal **arXiv.** jan 2018.

MARTELLO, M. **Estimativa da altura e produtividade da cana-de-açúcar utilizando imagens obtidas por aeronave remotamente pilotada**. (Dissertação Mestrado) – Universidade de São Paulo, Piracicaba, 2017. Disponível em: http://www.teses.usp.br/teses/disponiveis/11/11152/tde-16102017-170204/. Acesso em: 08 dez. 2023.

MARTOS, V. ; AHMAD, A.; CARTUJO, P.; ORDOÑEZ, J. Ensuring Agricultural Sustainability through Remote Sensing in the Era of Agriculture 5.0. **Applied sciences.** Suíça, Vol.11 (13), p.5911, jun 2021.

MILLSTEIN, F. **Convolutional Neural Networks in Python Begginer´s Guide to Convolutional Neural Networks Python**. Create space Independent Pub, 2018.

NICHOL, A.; SCHULMAN J.; *Reptile*: a Scalable Meta learning Algorithm. **arXiv: Learning**, mar 2018.

OLIVEIRA NETO, D. de. **Índices de vegetação, rendimento de grãos e seus componentes em soja, em área com avaliação de agentes de controle biológico**. 2020. Dissertação (Mestrado em Agricultura de Precisão) – Universidade Federal de Santa Maria (UFSM, RS), 2020.

PALCZYNSKI, K.;SMIGIELl, S.; LEDZINSKI, D.; BUJNOWSKI, S. Estudo do aprendizado de poucos tiros para classificação de ECG com base no conjunto de dados PTB-XL. **Sensores 2022,22**, 904.

PAN, J.; WANG, T.; WU, Q. RiceNet: A two stage machine learning method for rice disease identification. **Biosystems Engineering** vol 225, p. 25-40. 2022. Acesso em: 08 dez. 2023.

PRESTES, C. D. P. **Predição de produtividade de trigo por meio de dados espectrais e altura estimada da planta obtidos por meio de aeronave remotamente pilotada**. 2020. Dissertação (Mestrado em Computação Aplicada) - Universidade Estadual de Ponta Grossa, Ponta Grossa, 2020.

RAGHAV, P. blog médium: O blog médio. Califórnia, Estados Unidos disponível em: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-*CNN*-deep-learning-99760835f148. Acesso em: 15 dez. 2023.

RAVICHANDIRANS.; **Hands-On *Meta-Learning* with Python** 1ª ed. Birmingham, Reino Unido: Packt Publishing Ltd., 2018.

REBELO, C.; NASCIMENTO, J. Measurement of Soil Tillage Using UAV High-Resolution 3D Data **Remote sensing.** Suíça. Vol.13 (21),p.4336, out 2021.

RÊGO, L. G. C. do et al.; (2022). Uso de *Meta-Learning* em Tarefas de Aprendizado Profundo. **Minicursos do SBBD** 2022 (pp.53-82). 2022. 10.5753/sbc.10309.7.3.

SANKARAN, S. *et al.* Low-altitude, high-resolution aerial imaging systems for row and field crop phenotyping: A review. **European Journal of Agronomy**, 2015.

SANTANA, L. S., *et al*; Identificação e contagem de cafeeiros com base em rede neural convolucional aplicada a imagens RGB obtidas por RPA. ***Sustentabilidade****,* 2023 ,15 (1), 820. Acesso em: 07 dez. 2023.

SANTOS, V. B; SANTOS, A. M. F.; ROLIM, G.S. Estimativa e previsão da produtividade da soja usando redes neurais artificiais. **Revista de Agronomia**. 2021; 113 p. 3193–3209. jun. 2021.

SETIAWAN, W. **Deep Learning menggunakan Convolutional Neural Network Teoridan Aplikasi.** MNC, Malang, 2020.

SIEBERT, C. R. **Otimizando Parâmetros de uma DenseNet:** através do controle de geração de mapas de características. Editora Dialética. São Paulo, 2022.

SILVA, D.G. da. **Autenticação utilizando Atributos Faciais obtidos por Redes Neurais Convolucionais em Sistema de Gestão de Aprendizado**. 2021. Dissertação (Mestrado) — Universidade Federal do ABC, Programa de Pós Graduação em Ciência da Computação, Santo André, 2021.

SILVA, F., et al. **Soja :** do plantio à colheita 2ª ed. São Paulo, SP : Oficina de Textos, 2022.

SILVA, I. N. da; SPATTI, D. H.; FLAUZINO, R. A. **Redes Neurais Artificiais:** para engenharia e ciências aplicadas fundamentos teóricos e aspectos práticos. São Paulo: Artliber.2016.

SIKKA, B. **Elements of Deep Learning for Computer Vision**. 1ª ed. BPB Publications, Índia, 2021.

SOKOLOVA, M.; JAPKOWICZ, N.; SZPAKOWICZ, S. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. In: 2006 AI Australasian Joint Conference on Artificial Intelligence (AI). [S.l.]: Lecture Notes in **Computer Science**. Vol. 4304. 1015-1021. 10.1007/11941439_114., 2006. Acesso em: 08 dez. 2023.

STACHAK, A. **Avaliação do estado nutricional de nitrogênio e estimativa da produtividade de biomassa de trigo por meio de mineração de dados de sensoriamento remoto.** 2018. Dissertação (Mestrado em Computação Aplicada), Universidade Estadual de Ponta Grossa, Ponta Grossa, 2018.

TASSIS, L. M.; KROHLING, R. A. "Few-shot Learning for Biotic Stress Classification of Coffee Leaves." **Artificial Intelligence in Agriculture** (2022): 55-67. Web.

VINISKI. A.D. **Avaliação da eficiência da mineração de dados clássica e espacial na estimativa de produtividade de grãos em imagens obtidas por meio de aeronave remotamente pilotada.** 2019. Dissertação (Mestrado - Computação Aplicada - Área de Concentração: Computação para Tecnologias em Agricultura) Universidade Estadual de Ponta Grossa, Ponta Grossa, 2019.

WANG, J.; CHEN, Y. Introduction to Transfer Learning Algorithms and Practice**. Springer**, Beijing, 2023.

WANG, Y.; YAO, Q.; KWOK, J. T.; NI, L. M. Generalizing from a Few Examples: A Survey on Few-Shot Learning. **ACM Comput. Surv.1**, a.1, mar. de 2020.

ZOU, L. *Meta-Learning*: Theory Algorithms and Applications. **Elsevier Science**, 1ª Ed., 2022.