MODELO DE MACHINE LEARNING PARA PREDIÇÃO DE FOLHAS SAUDÁVEIS E DOENTES NO CAFÉ

ISAC FERNANDES COSTA[1]; VITOR HUGO BOZO DE CARVALHO[1]; MARCEL SANTOS SILVA[2];

[1] Discente em Big Data no Agronegócio na FATEC Pompéia “Shunji Nishimura”, Pompeia-SP, isac.costa@fatec.sp.gov.br; vitor.carvalho6@fatec.sp.gov.br
[2] Docente em Big Data no Agronegócio na FATEC Pompéia “Shunji Nishimura”, Pompeia-SP, marcel.silva9@fatec.sp.gov.br

RESUMO:

A utilização de tecnologias avançadas na agricultura, como o processamento por algoritmos de aprendizado de máquina *(ML),* tem se tornado cada vez mais relevante para auxiliar os produtores rurais. No entanto, as doenças do café como a Phoma causada pelo fungo (*Phoma spp*), e a infestação do bicho mineiro (*Leucoptera Coffeella*), uma praga que cria galerias nas folhas do café estão se tornando cada vez mais severas. Essas doenças podem ser causadas por vários agentes, como fungos, bactérias, vírus, nematóides e condições climáticas desfavoráveis. Diante desse cenário, é de extrema importância implementar medidas preventivas eficazes, utilizando tecnologias como o *ML*, para mitigar a evolução dessas doenças, garantindo mais saúde e produtividade das plantações de café. Por esse motivo, o presente trabalho tem como objetivo desenvolver um modelo de *ML* para a detecção e classificação de folhas do café saudáveis e doentes, contribuindo com o produtor na melhoria da qualidade da sua produção. O modelo foi desenvolvido no Jupyter Notebook, uma das ferramentas mais populares para a exploração e desenvolvimento de projetos de ciência de dados e aprendizado de máquina, e busca abordar um problema mais generalizado, detectando se a folha está doente ou saudável, oferecendo uma solução automatizada na detecção de possíveis doenças no café.

Palavras-chave: Inteligência Artificial; Aprendizado de Máquina; detecção de doenças.

INTRODUÇÃO

Desde a chegada ao Brasil em 1727, o café tem cumprido um papel fundamental na geração de riqueza e se estabeleceu como o principal produto na história do País. Atualmente, o Brasil mantém sua posição como o maior produtor global de café, contribuindo significativamente para a economia nacional, com uma média de 75 milhões de sacas exportadas anualmente. O café continua sendo um importante gerador de receita, contribuindo com mais de 2% do valor total das exportações brasileiras, além de

responder por mais de um terço da produção mundial de café. Esse mercado encontra-se em expansão contínua, com o agronegócio do café movimentando impressionantes 91 bilhões de dólares globalmente, alimentado pela produção anual de cerca de 28,9 sacas por hectare. Vale destacar, que essa atividade envolve milhares de pessoas, desde a produção até o consumo final(EMBRAPA, 2023).

O cafeeiro enfrenta várias doenças tanto na fase de viveiro como no campo, o que resulta na redução da produtividade e da qualidade do café, além do aumento dos custos de produção. A ocorrência e os prejuízos causados por essas doenças dependem de alguns fatores, como os patógenos, as plantas de café, o ambiente e a influência humana. Tais fatores variam entre regiões e até mesmo dentro de uma mesma região, contribuindo para a profundidade desse desafio (CARVALHO; CHALFOUN; CUNHA, 2013, p 13).

Entre as pragas e doenças que trazem prejuízos para a produção do café, podemos citar o Bicho Mineiro e a Mancha de Phoma. O Bicho Mineiro (Leucoptera Coffeella) é a praga mais influente nos cafezais, com grande impacto nas plantações das Américas e da África. Essa praga causa uma perda considerável de folhagem das plantas, resultando em sua debilitação e comprometimento da colheita. No Brasil, o inseto é encontrado principalmente em áreas de clima quente e com menor disponibilidade de água. Pesquisas realizadas em Minas Gerais, Espírito Santo e São Paulo mostram que a ação do bicho mineiro pode reduzir o potencial produtivo do cafeeiro em 50% a 80%, dependendo da intensidade, duração e época do ataque (AGRO BAYER, 2019).

A presença inicial da mancha de Phoma (*Phoma costaricensis*) no Brasil foi registrada durante a década de 1970. A partir desse momento, a doença se espalhou pelas principais áreas de produção do País. A origem do seu nome deriva dos fungos do gênero Phoma spp., responsáveis por sua ocorrência. Esses fungos têm a capacidade de entrar nas folhas, frutas e brotos do cafeeiro, provocando infecção. Isso pode ser facilitado por danos mecânicos anteriores, como os que podem ser causados pela colheita mecanizada (BAPTISTELLA, 2021).

Destaca-se a importância de tecnologias de *ML* para a identificação e distinção de folhas saudáveis e enfermas na plantação de café. Por esta razão, o objetivo da pesquisa é desenvolver um algoritmo para buscar e analisar características e padrões presentes nas

folhas, utilizando técnicas de *ML,* a fim de fornecer uma ferramenta eficiente para auxiliar na detecção precoce de doenças foliares do cafeeiro.

MATERIAIS E MÉTODOS:

O principal foco do desenvolvimento, conforme supramencionado, consiste em construir um algoritmo que seja capaz de identificar doenças na folha do café, tal como fazer a classificação se a planta está saudável ou doente. Com base no estudo de Ferreira (2017), que aborda a eficácia do uso do algoritmo de Redes Neurais Convolucionais para a identificação de padrões nas imagens como bordas, cores, texturas, relevos e outras características, utilizou-se este algoritmo para classificação binária das imagens da folha do café.

Após a escolha do algoritmo a ser utilizado, foram realizadas buscas por um conjunto de dados que tivesse uma quantidade significativa de imagens da folha do café, tanto folhas saudáveis quanto folhas que possuíssem algum tipo de doença, e após a avaliação do dataset disponibilizado e hospedado pelo site *National Library of Medicine* (2021), que conta com um conjunto de dados de 18.985 imagens de folhas do café saudáveis, 6.572 imagens da folha do café com a doença de *Phoma* e 16.979 imagens da folha do café prejudicadas pelo *Bicho Mineiro*, concluiu-se que esse *dataset* seria suficiente para suprir a necessidade de aprendizado de um modelo de *machine learning,* conforme abordagem descrita por Rolnick *et al. (*2017)*,* onde explora o impacto da quantidade de dados no treinamento de modelos de *deep learning* para classificação de imagens. As imagens do dataset de folhas do café com a doença de *Phoma* e *Bicho Mineiro* foram agrupadas em um único diretório (totalizando 23.551 imagens de plantas doentes), e todo o excesso de imagens de folhas com o *Bicho Mineiro* foi removido até que os dados de folhas doentes (já somados as duas doenças) fosse equivalente aos dados de folha saudável, totalizando 18.985 imagens de folhas do café saudáveis e 18.985 imagens de folhas do café doente (com a doença da *Phoma* e *Bicho Mineiro*) após a junção. Isso foi feito para que os dados de folhas doentes fossem equivalentes aos dados de folhas saudáveis .

Foi realizada a separação de 30% dos dados para o diretório *test* (Teste) e 70% dos dados para *train (treinamento).* Lembrando que, foram separados igualmente para *healthy* (Saudáveis) e *sick* (Doentes)*,* sendo assim, foi realizado o cálculo do total de

imagens utilizadas no modelo, portanto foi extraído 30% (5.695) de 18.985 para *test* e os 70% (13.289) restantes para *train* para ambas as classes.

Por fim, os subdiretórios receberam a quantidade equivalente de dados, com o objetivo de obter um balanceamento durante a realização do treinamento.

Para montagem do ambiente de desenvolvimento foi usado a tecnologia de execução de instâncias de uma aplicação por meio de containers com docker, onde se instalou o Jupyter Notebook, uma ferramenta muito usada para análise de dados com a linguagem de programação Python (ARCANJO, 2022). Além disso, foram realizadas as instalações de todas as dependências (bibliotecas do Python) necessárias para a implementação do código. Dentre elas, as principais são descritas abaixo:

- *TensorFlow*: é uma biblioteca criada pela *Google* usada para trabalhar com computação numérica e aprendizado de máquina baseada em tensores (abstrações para qualquer tipo lista ou matrizes multidimensionais). Além disso, o TensorFlow inclui como um dos seus recursos a *API* do Keras e de subclassificação de modelos (DEVELOPER RESOURCE CENTER, 2022);
- *Keras*: é uma biblioteca de *deep learning* implementada utilizando o TensorFlow para diversas linguagens, como Python e R, com foco em sua facilidade de utilização. Ele permite modelar e treinar modelos de redes neurais com poucas linhas de código e linguagem de alto nível (DIDÁTICA TECH, 2022);
- *Scikit-Learn*: é uma biblioteca de aprendizado de máquina de código aberto para a linguagem de programação Python. Ela é uma das principais bibliotecas de aprendizado de máquina disponíveis atualmente e é amplamente usada na indústria e na academia (PACHECO, 2022);
- *Pillow*: é uma biblioteca de processamento de imagens em Python que fornece recursos avançados para manipulação de imagens. Com o Pillow, os usuários podem realizar várias operações em imagens, como abrir, salvar e manipular imagens em diferentes formatos (JPEG, PNG, BMP, GIF etc.), redimensionar, recortar, girar, aplicar efeitos, como filtro e brilho (AMARAL, 2020);
- *NumPy*: é uma biblioteca de computação científica em Python que fornece suporte para *arrays* (listas) e matrizes multidimensionais, além de sua ampla variedade de funções matemáticas para operações em *arrays* (SANTIAGO, 2018);
- *Matplotlib*: é uma biblioteca de visualização de dados em Python. Ela é usada para criar gráficos, diagramas, histogramas, *plots* (gráficos de dispersão) e outras visualizações de dados em 2D e 3D. Com o Matplotlib, é possível criar uma ampla variedade de visualizações, desde simples gráficos de linha e barras até gráficos

mais complexos, como gráficos de dispersão, de contorno, de superfície, entre outros (BHANDARI, 2020);

- *Keras Tuner*: é uma biblioteca de otimização de hiperparâmetros para modelos de aprendizado de máquina construídos com o Keras. Ela ajuda a encontrar a melhor configuração de hiperparâmetros para um modelo de aprendizado de máquina, automatizando o processo de ajuste de hiperparâmetros. Este processo de ajuste envolve a escolha das melhores configurações para parâmetros que não são aprendidos pelo modelo durante o treinamento, como o número de camadas e neurônios em uma rede neural, a taxa de aprendizagem, a função de ativação entre outros (ANDRADE, 2020).

Após a instalação das dependências do projeto, atribuiu-se os caminhos (*paths*) dos diretórios de treino e teste no diretório raiz do projeto, utilizou-se a biblioteca '*OS*' (*Operating System*, ou Sistema Operacional em português) que é uma biblioteca do próprio Python para lidar com operações que envolvem o sistema operacional. Já com os diretórios definidos, foi aplicada a primeira técnica para evitar o sobreajuste (*overfitting*) e melhorar o desempenho do modelo (ALTO, 2020), chamada de aumento dos dados (*data augmentation*), conforme exibido pela figura 1. Essa técnica consiste em aumentar a quantidade de dados utilizados no aprendizado do modelo por meio de transformações em dados já existentes na base de dados, essas transformações podem incluir a rotação, zoom, deformações de cisalhamento, ajuste de brilho, mudanças aleatórias no canal de cores da imagem, dentre outras, para gerar novas imagens e aumentar o conjunto de treinamento.

Figura 1. Imagem da função de *Data Augmentation.*

```
train_datagen = ImageDataGenerator(
    rotation_range=20,
    zoom_range=0.2,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
    brightness_range=[0.5, 1.5],
    fill_mode='nearest',
    channel_shift_range=10.0
)
```

Fonte: Elaborado pelos autores (2023).

Em seguida, foi realizado o carregamento dos dados de acordo com os *paths* definidos para treinamento e teste. Há alguns pontos importantes relacionados aos parâmetros de entrada (*input*) que precisam ser destacados, como por exemplo, o tamanho das imagens de entrada que foram definidos como 128 por 128 pixels, o tipo de cores das imagens como sendo *RGB* (acrônimo para *Red*, *Green* e *Blue*) e o tamanho do lote (*batch size*) que define a quantidade de amostras de imagens que serão processadas de uma só vez antes que o algoritmo ajuste seus pesos como 32. De acordo com Kandel (2020), o tamanho do lote tem um impacto direto no treinamento do modelo, um tamanho de lote menor pode levar a uma atualização mais frequente dos pesos, o que poderia ajudar a evitar que o modelo fique preso em mínimos locais. No entanto, isso pode aumentar o tempo de treinamento, pois a atualização dos pesos é computacionalmente intensiva. Por outro lado, um tamanho de lote maior pode reduzir o tempo de treinamento, mas pode levar a uma menor precisão do modelo, pois a atualização dos pesos pode ser menos frequente. Abaixo está o código referente ao carregamento dos dados de treino e teste.

Figura 2. Divisão dos dados de treino e teste.

```
train_dataset = train_datagen.flow_from_directory(
                    train_folder,
                    target_size=(128, 128),
                    batch_size=32,
                    color_mode='rgb',
                    shuffle=True,
                    class_mode='binary'
                )


test_dataset = image_dataset_from_directory(
                    test_folder,
                    image_size=(128, 128),
                    batch_size=32
                )
```

Fonte: Elaborado pelos autores (2023).

Para montagem da rede neural convolucional foram usadas diferentes camadas, cada uma desempenhando papéis importantes no processo de convolução. Essas camadas são: camada de convolução, agrupamento (*pooling*), achatamento (*flatten*), densa, abandonar (*dropout*), normalização de lote (*batch normalization*) e regularização L2 (*L2 regularization*).

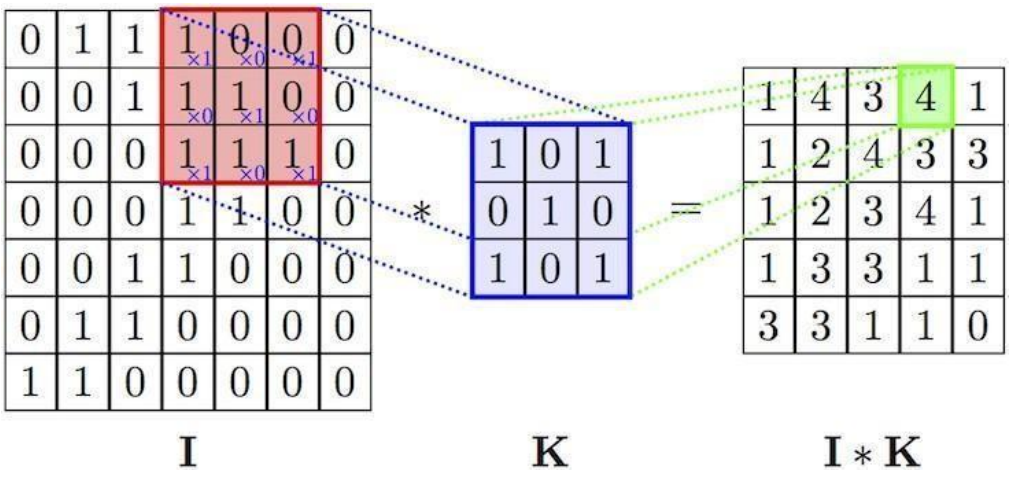
1. MONTAGEM DAS CAMADAS DA REDE NEURAL

A montagem de camadas em redes neurais é crucial para o aprendizado de máquina. As camadas de Pooling reduzem a dimensionalidade dos dados, a camada Flatten transforma dados multidimensionais em vetores unidimensionais, e a camada Dense conecta neurônios para operações lineares e não lineares, essenciais na extração de características e aprendizado de representações complexas.

1.1 Rede Neural Convolucional

O processo de convolução é usado para extrair recursos, (também chamados de características ou *features*) de uma imagem. Essa operação envolve a movimentação de uma matriz pequena, também conhecida como filtro ou *kernel,* pela imagem de entrada. Os valores do filtro são multiplicados pelos valores dos pixels correspondentes na imagem, gerando um resultado que é então somado e armazenado em uma nova matriz, chamada de mapa de características. O processo de convolução é repetido em diferentes áreas da imagem de entrada, gerando múltiplos mapas de características (CHOUDHARI, 2020). Esses mapas de características são posteriormente utilizados como entrada para as camadas seguintes da rede neural, que combinam essas informações para realizar tarefas específicas, como classificação de imagens, conforme demonstrado pela figura 3.

Figura 3. Ilustração da operação de Convolução.



Fonte: MOHAMED (2017).

1.2 Pooling

O *pooling,* ou *max poolin*g, segundo a *Data Science Academy* (2022), é uma operação usada em redes neurais convolucionais para reduzir a dimensionalidade dos mapas de características gerados pela convolução. Seu objetivo consiste em preservar as características mais relevantes da imagem, enquanto reduz o tamanho da representação de entrada para tornar o processo de treinamento mais eficiente. A operação de *max pooling* envolve a divisão da imagem em regiões retangulares, chamadas de janelas, e a seleção do valor máximo dentro de cada janela. O resultado é uma nova matriz com uma resolução reduzida, mas com as características mais importantes preservadas.

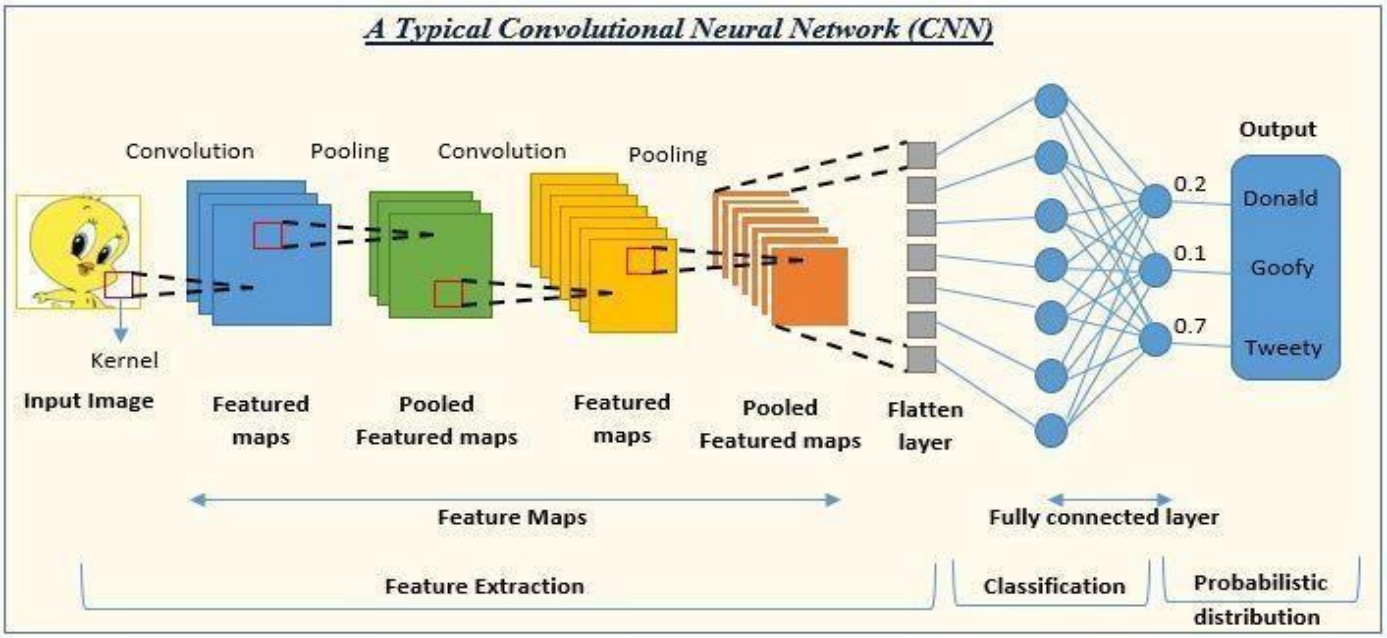
1.3 Flatten

A camada de *flatten* é uma camada de processamento de dados utilizada em redes neurais que transforma uma matriz multidimensional em um vetor unidimensional, "achatando" os dados de entrada. Isso é feito para que a saída da camada de *flatten* possa ser conectada a uma camada densa, *fully-connected* ou totalmente conectada, da rede neural. Em outras palavras, a camada de *flatten* permite que uma rede neural trabalhe com dados multidimensionais como se fossem dados unidimensionais (CANU, 2022).

1.4 Dense

A camada densa ou *fully connected*, devido às várias ligações entre os neurônios das camadas anteriores e posteriores, é adicionada após as camadas convolucionais e de pooling em uma rede neural convolucional e é responsável por aprender representações mais abstratas dos recursos extraídos pelas camadas anteriores. Portanto, a camada densa é usada para classificar as características extraídas pelas camadas anteriores em categorias específicas (CANU, 2022). Este processo é ilustrado pela figura 4.

Figura 4. Ilustração do processo completo de Convolução até a classificação.



Fonte: SHAH (2022).

2. REGULARIZADORES

Nesta seção, é abordado dois métodos comuns de regularização em modelos de aprendizado de máquina: a Regularização L2, também conhecida como decadência de peso, e o Dropout.

2.1 Regularização l2

O L2 *regularization*, também conhecido como decadência de peso (*weight decay*), é uma técnica comum de regularização usada em modelos de aprendizado de máquina para evitar o overfitting. Nas redes neurais convolucionais, a regularização L2 funciona adicionando um termo de penalização ao cálculo da função de custo. Esse termo penaliza pesos grandes na rede, o que impede que o modelo se ajuste demais aos dados de treinamento e ajuda a melhorar a capacidade de generalização do modelo (OMAR, 2020).

2.2 Dropout

Assim como o L2 *regularization,* o *dropout* também é usado para reduzir o *overfitting* em modelos de redes neurais. Essa técnica funciona removendo

aleatoriamente alguns dos neurônios em uma camada durante o treinamento da rede, o que força a rede a aprender características mais robustas e menos dependentes de neurônios específicos. Em cada etapa do treinamento, cada neurônio tem uma probabilidade de ser temporariamente removido com base em uma taxa de *dropout* pré-definida. Essa taxa de *dropout* é um hiperparâmetro que determina a fração de neurônios que serão desligados em cada camada (OMAR, 2020).

3. NORMALIZADORES

Nesta seção o foco é falar do Batch Normalization, uma técnica de normalização utilizada em redes neurais para acelerar o treinamento e aumentar a estabilidade do modelo.

3.1 Batch Normalization

O *batch normalization* é uma técnica de normalização que tem como objetivo normalizar as ativações de cada camada, tornando o treinamento mais rápido e estável. Essa técnica envolve o cálculo da média e do desvio padrão das ativações em um mini lote de dados e, em seguida, normaliza as ativações subtraindo a média e dividindo pelo desvio padrão (SAXENA, 2021).

4. OTIMIZADORES

O foco desta sessão é apresentar o Adam Optimizer, um algoritmo de otimização usado em *deep learning* e redes neurais artificiais.

4.1 Adam Optimizer

O otimizador Adam "*Adam Optimizer*" é um algoritmo de otimização de gradiente descendente usado em *deep learning* e redes neurais artificiais. Esse é um método estocástico que combina a taxa de aprendizado adaptativa, e o momento para atualizar os pesos da rede. A ideia é ajustar a taxa de aprendizado com base no histórico dos gradientes anteriores, além de incorporar uma técnica de momento para ajudar a evitar a convergência em mínimos locais. O Adam é considerado um dos melhores otimizadores para treinar redes neurais profundas e é amplamente utilizado na comunidade de

aprendizado profundo (ALABDULLATEF, 2020).

5. FUNÇÕES DE ATIVAÇÃO

Nesta seção, são discutidas duas funções de ativação amplamente utilizadas em modelos de deep learning: ReLU e Sigmoid.

5.1 ReLU

A função de ativação *ReLU* (Rectified Linear Unit, ou Unidade Linear Retificada) é uma das funções matemáticas mais comuns e mais usadas em modelos de deep learning. A função *ReLU* define a saída como zero para todos os valores de entrada negativos e mantém a saída linear para todos os valores de entrada positivos. Matematicamente, a *ReLU* é considerada vantajosa para redes neurais porque é fácil de computar, e ajuda a resolver o problema de gradientes desaparecendo em redes muito profundas (SHARMA, 2017).

5.2 Sigmoid

A função de ativação *sigmoid* é uma função matemática que transforma um valor de entrada em um valor de saída dentro de um intervalo entre 0 e 1. Ela é utilizada para classificação binária, onde é necessário decidir se a entrada pertence a uma das duas classes possíveis. Em redes neurais convolucionais, a função sigmoid é geralmente usada como a função de ativação na camada de saída quando há apenas duas classes possíveis. Ela é útil para problemas de classificação binária, como reconhecimento de imagens, onde é necessário decidir se a imagem contém ou não um objeto específico (SHARMA, 2017).

Todas as técnicas apresentadas foram utilizadas para a construção de uma rede neural que fosse capaz de apresentar o melhor desempenho na hora de classificar as folhas do café, onde se realizou diferentes tipos de combinações na montagem dessas camadas, tanto na sequência quanto na quantidade de hiperparâmetros em cada camada. Para agilizar o processo de busca pelas melhores combinações de hiperparâmetros, foram utilizadas técnicas de busca aleatória (*random search*). O *random search* seleciona aleatoriamente uma amostra de configurações de hiperparâmetros que foram definidos previamente para treinar e avaliar o modelo, além de utilizar a técnica de

cross-validation (validação cruzada) que visa avaliar a capacidade de generalização do modelo dividindo os dados de treino em subconjuntos de dados, chamados de *folds* ou dobras. Estes subconjuntos de dados são usados tanto para teste quanto para treino do modelo, e o processo é repetido ‘k’ vezes de forma que cada subconjunto seja utilizado para teste pelo menos uma vez.

Ao final, os resultados são combinados e uma métrica de avaliação é calculada, o objetivo é ter uma estimativa mais precisa do desempenho do modelo em dados não vistos durante o treinamento (RAMEZAN *et al*, 2019). Também foi feito um cálculo utilizando a função *rescaling* (redimensionando) sobre o valor dos pixels de cada imagem, para que os valores ficassem entre 0 e 1. Sendo assim, qualquer pixel de uma imagem com canal RGB que tenha um valor entre 0 e 255, variando entre preto (0, 0, 0) e branco (255, 255, 255), seria dividido por 255 para retornar um valor entre 0 e 1. Isso foi feito para normalizar as intensidades dos pixels das imagens de entrada e facilitar o cálculo dos valores nas operações de convolução, já que seria mais fácil de trabalhar com valores menores na hora dos ajustes dos pesos.

Figura 5. Montagem da Rede Neural.

```
model2 = keras.Sequential()
model2.add(Rescaling(scale=1.0/255, input_shape=(128, 128, 3)))

model2.add(Conv2D(16, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model2.add(BatchNormalization())

model2.add(Conv2D(32, kernel_size=(3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.02)))
model2.add(Dropout(0.25))

model2.add(MaxPooling2D(pool_size=(2, 2)))

model2.add(Conv2D(32, kernel_size=(4, 4), activation='relu', kernel_regularizer=regularizers.l2(0.02)))
model2.add(BatchNormalization())

model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Flatten())

model2.add(Dense(32, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model2.add(Dropout(0.25))

model2.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model2.add(BatchNormalization())

model2.add(Dense(1, activation='sigmoid', kernel_regularizer=regularizers.l2(0.01)))

model2.compile(loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

Fonte: Elaborado pelos autores (2023).

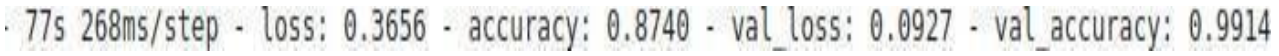
RESULTADOS E DISCUSSÃO

O modelo foi criado de acordo com a esquema demonstrado na figura 5, sendo

treinado e depois testado com 2 épocas (*epochs*). Cada época é equivalente a uma iteração sobre todos os dados, tanto de treino quanto de teste. Durante a primeira época o modelo alcançou uma acurácia de treino de 87,40% e função de perda (*loss*) de 36,56%, que representa o quão bem o modelo está se ajustando aos dados apresentados a ele, isso significa que o modelo está fazendo previsões mais precisas, já nos dados de teste alcançou uma acurácia de 99,14% e loss de 9,27%.

Na segunda época, o modelo atingiu uma acurácia de treino de 87,76% e *loss* de 31,87%, muito semelhante ao treino da primeira época. Já nos testes da segunda época houve uma diminuição da acurácia de aproximadamente 16,24%, atingindo 82,90% de acurácia e 49,30% de *loss,* incidindo em um aumento nos erros de teste da segunda época de aproximadamente 17,43%. A *Accuracy* faz referêcia a acurácia de treino; *loss* faz referência a função de perda no treino; *val_acurracy* faz referência a acurácia de teste; *val_loss* faz referência a função de perda no teste:

Figura 6. Resultado da primeira época.



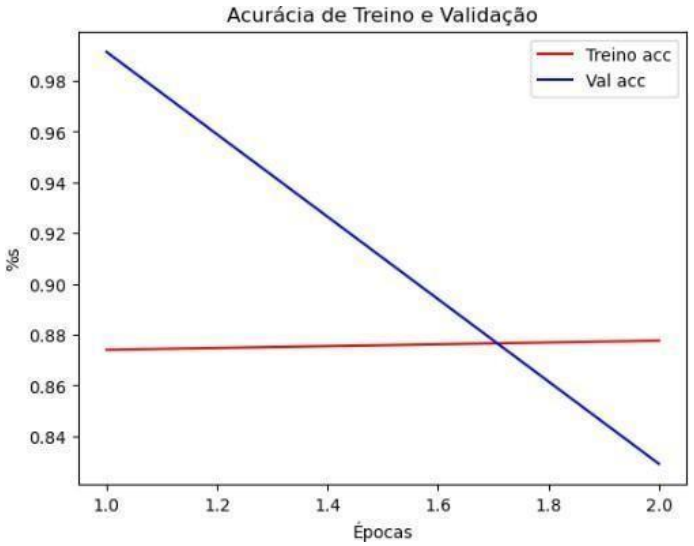
Fonte: Elaborado pelos autores (2023).

Figura 7. Resultado da segunda época.

Fonte: Elaborado pelos autores (2023).

Analisando os gráficos e os dados disponibilizados após o treinamento do modelo, conforme figuras 8 e 9, é possível observar que não houve mudança significativa no aprendizado do modelo durante as 2 épocas referentes à acurácia e *loss* de treino, mas houve uma diminuição expressiva na acurácia de teste na segunda época. Com base nos estudos realizados, pode-se concluir que essa diminuição da acurácia de testes se deve a um possível *overfitting* do modelo aos dados de testes na primeira época e um balanceamento dos pesos do modelo na segunda época. O gráfico abaixo mostra os resultados de acurácia, que é uma métrica de avaliação que mede a proporção de previsões corretas feitas por um modelo em relação ao total de previsões realizadas.

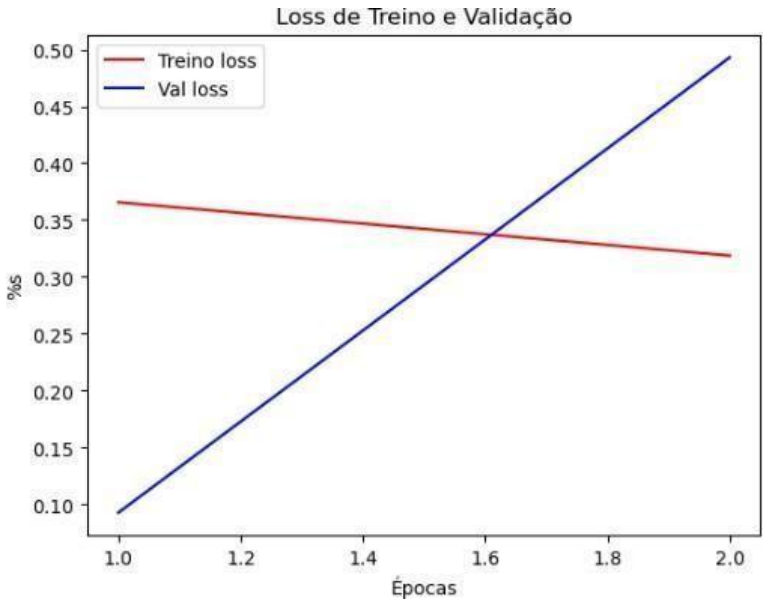
Figura 8. Gráfico da acurácia de treino e teste de acordo com as épocas.



Fonte: Elaborado pelos autores (2023).

Abaixo é mostrado o *loss*, que é uma medida que quantifica o quão bem um modelo de machine learning está performando em relação aos dados de treinamento e teste, comparando as predições do modelo com os valores reais dos dados e calculando a diferença entre eles.

Figura 9. Gráfico da função de perda de treino e teste de acordo com as épocas.



Fonte: Elaborado pelos autores (2023).

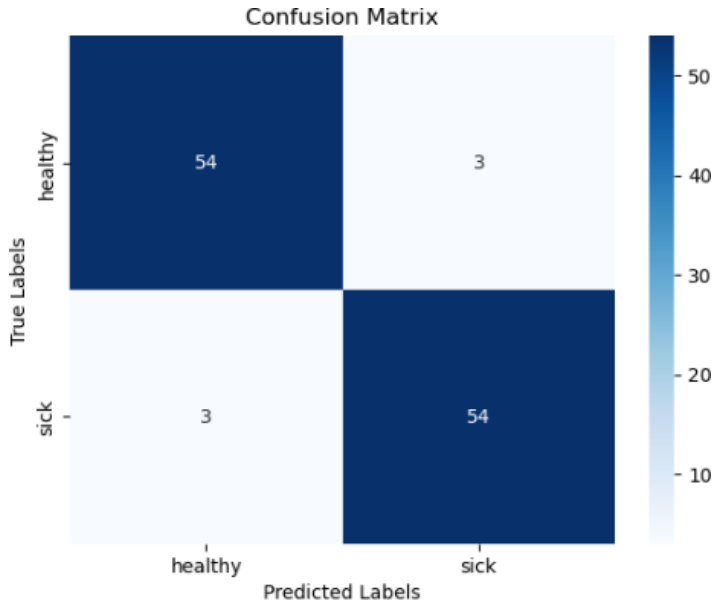
Após realizado o treinamento e o teste do modelo, foi utilizado um conjunto de dados para validação com imagens aleatórias da folha do café “buscadas” na internet para de fato se avaliar a capacidade do modelo em fazer predições de dados nunca vistos antes. No total foram adicionadas 114 novas imagens sendo 57 de folhas saudáveis e 57 de folhas com diferentes tipos de doenças, sendo elas: Phoma, Cercosporiose, Bicho Mineiro, Broca, Ferrugem e Seca dos Ponteiros. As imagens para validação incluem algumas que não dão ênfase na folha do café, ou seja, parte das imagens não tem nenhum tipo de zoom aplicado à folha, isso significa que algumas das imagens incluem várias folhas juntas e não apenas uma. Vale lembrar que o modelo foi treinado com imagens que possuem ênfase na folha, mas apesar disso, essa característica não interferiu diretamente na capacidade do modelo de identificar as plantas saudáveis e doentes, apesar de ser recomendado que as imagens destinadas a predição tenham algum tipo de ênfase na folha, devido aos dados de treinamento apresentados ao algoritmo, temos a seguir uma imagem que demonstra os dados de validação usados.

Figura 10. Foto de uma parte do diretório de imagens para validação.

Fonte: Elaborado pelos autores (2023).

Abaixo é mostrado a matriz de confusão, que é uma tabela que resume o desempenho de um modelo de classificação, mostrando a contagem de previsões corretas e incorretas para cada classe.

Figura 11. Matriz de confusão da predição dos dados de validação.



Fonte: Elaborado pelos autores (2023).

Nota: *Healthy* faz referência às folhas saudáveis; *sick* faz referência às folhas doentes.

Observando a matriz de confusão, apresentada na figura 11, pode-se concluir que o modelo conseguiu acertar grande parte das imagens dadas à ele, de 57 imagens de folhas saudáveis o algoritmo acertou 54, verdadeiros negativos, e errou 3, falsos positivos, e de 57 imagens de folhas doentes o algoritmo acertou 54, verdadeiros positivos, e errou 3
falsos negativos).

Figura 12. Novas métricas de acordo com a validação com novos dados.

	Métrica	Valor
0	Taxa de acerto	94.74%
1	Taxa de erro	5.26%
2	Precisão	94.74%
3	Recall	94.74%
4	F1-score	94.74%
5	Taxa de erro da classe sick	5.26%
6	Taxa de erro da classe healthy	5.26%

Fonte: Elaborado pelos autores (2023).

Com essa validação, demonstrada na figura 12, realizada sobre esses dados nunca vistos antes, foram feitas novas avaliações do modelo de acordo com os dados que foram previstos, e novas métricas foram geradas. Calculou-se as taxas de acerto, ou acurácia, a taxa de erro, precisão, *recall*, *F1-score*, a taxa de erro para folhas da classe de plantas doentes, além da taxa de erro para folhas pertencentes a classe de plantas saudáveis. Os resultados foram bastante satisfatórios ao comparar os valores de predições com os valores reais e os resultados dessas métricas indicaram uma boa eficácia do modelo em realizar a detecção de plantas saudáveis e plantas doentes da folha do café.

CONCLUSÃO:

Tendo em vista o trabalho realizado, um modelo de *machine learning* para a predição de folhas saudáveis e doentes no café traz consigo a promessa de melhorar a qualidade da planta. Ao construir um modelo de *machine learning* para esse propósito, é crucial contar com uma base de dados abrangente e diversificada que represente adequadamente as diferentes doenças e condições que afetam as folhas do café. Isso permitirá que o modelo aprenda e identifique padrões distintos entre as folhas saudáveis e doentes, minimizando a chance de classificações incorretas.

Dessa forma, destaca-se a importância em utilizar algoritmos de aprendizado de máquina, para evitar prejuízos e aumentar a produção e qualidade, entretanto, é essencial reconhecer a complexidade do desafio, uma vez que as características deixadas pelas doenças nas folhas apresentam notáveis semelhanças podendo confundir o algoritmo, é essencial adotar precauções adicionais durante a coleta de dados e de treinamentos; e, o desenvolvimento do modelo. Essa semelhança demanda uma abordagem cuidadosa, visando garantir a precisão e a confiabilidade das previsões obtidas.

Como trabalho futuro, é sugerido a realização de melhorias no modelo de predição. Uma abordagem seria expandir o conjunto de dados utilizado, incorporando amostras de folhas de diferentes regiões geográficas.

Por fim, é possível concluir que o objetivo proposto foi plenamente alcançado. O modelo de machine learning desenvolvido demonstrou uma capacidade promissora na predição de folhas de café saudáveis e doentes, oferecendo uma ferramenta eficaz para auxiliar agricultores na detecção precoce de problemas fitossanitários nas plantações de café.

REFERÊNCIAS

AGRO BAYER - “**Bicho Mineiro evolução e danos no café**”. Disponível em: https://www.agro.bayer.com.br/conteudos/bicho-mineiro-evolucao-e-danos-no-cafe. Acesso em 9.mai.2023.

ALABDULLATEF, Layan (2020). “**Complete Guide to Adam Optimization**”. Disponível em: https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d. Acesso em: 12.mai.2023.

ALTO, Valentina (2020). “**Data Augmentation in Deep Learning**”. Disponível em: https://medium.com/analytics-vidhya/data-augmentation-in-deep-learning-3d7a539f7a28 . Acesso em: 10.mai.2023.

Amaral, BRUNA (2020). “**[Python] Composição de imagens utilizando Pillow”**.

Disponível em: https://amaral-bruna.medium.com/python-criando-composi%C3%A7%C3%A3o-de-imagem-utilizando-pillow-91022dff8369. Acesso em 09.mai.2023.

ANDRADE, Matheus (2020). “**Usando Hyperparameter Tuning e Regularização para melhorar uma Rede Neural Convolucional**”. Disponível em https://silvamathdd.medium.com/usando-hyperparameter-tuning-e-regulariza%C3%A7%C3%A3o-para-melhorar-uma-rede-neural-convolucional-659b5ac53191. Acesso em: 09.mai.2023.

ARCANJO, Jonys (2022). “**Jupyter Notebook- A melhor maneira de criar uma história com dados**”. Disponível em: https://medium.com/data-hackers/jupyter-notebook-a-melhor-maneira-de-criar-uma-hist%C3%B3ria-com-dados-dbc2e8e3dd9a. Acesso em: 10.mai.2023.

BAPTISTELLA, João (2021). **"Como evitar e controlar a mancha de Phoma no cafeeiro"**. Disponível em: https://blog.aegro.com.br/mancha-de-phoma-no-cafeeiro/. Acesso em 9.mai.2023.

BHANDARI, Aniruddha (2020). “**A Beginner’s Guide to matplotlib for Data Visualization and Exploration in Python**”. Disponível em: https://medium.com/analytics-vidhya/a-beginners-guide-to-matplotlib-for-data-visualization-and-exploration-in-python- 3fb32d03c3cd. Acesso em: 09.mai.2023.

CANU, Sergio (2022). “**Flatten and Dense layers | Computer Vision with Keras p.6**”. Disponível em: https://pysource.com/2022/10/07/flatten-and-dense-layers-computer-vision-with-keras-p-6/. Acesso em: 10.mai.2023.

CARVALHO, Vicente (2013). **"Doenças do cafeeiro: diagnose e controle"**. Disponível em: https://fatebtb.edu.br/novosite/wp-content/uploads/2021/04/15-Doen%C3%A7as-do-Cafeeiro.pdf. Acesso em 8.mai.2023.

CHOUDHARI, Pratik (2020). “**Understanding ‘convolution’ operations in CNN**”. Disponível em: https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4. Acesso em: 10.mai.2023.

Data Science Academy (2022). “**Capítulo 43 – Camadas de Pooling em Redes Neurais Convolucionais**”. Disponível em: https://www.deeplearningbook.com.br/camadas-de-pooling-em-redes-neurais-convolucionais/#:~:text=Podemos%20pensar%20em%20Max%2DPooling,elimina%20a%20informa%C3%A7%C3%A3o%20posicional%20exata. Acesso em: 10.mai.2023.

DEVELOPER RESOURCE CENTER (2022). “**O que é TensorFlow: um guia completo”**. Disponível em: https://developer.oracle.com/pt-BR/learn/technical-articles/1481879245886-120-what-is-tensorflow#:~:text=TensorFlow%20%C3%A9%20uma%20biblioteca%20de%20c%C3%B3digo%2Dfonte%20aberto%20amig%C3%A1vel%20para,acessar%20o%20reposit%C3%B3r io%20da%20biblioteca. Acesso em 09.mai.2023.

DIDÁTICA TECH (2022). “**O que é Keras? Para que serve?”**. Disponível em: https://didatica.tech/o-que-e-keras-para-que-serve/#:~:text=Para%20que%20serve%20o%20Keras,em%20permitir%20a%20experimenta%C3%A7%C3%A3o%20r%C3%A1pida. Acesso em 09.mai.2023.

EMBRAPA (2023). “**Produtividade média dos Cafés do Brasil estimada para 2023 é de 29 sacas por hectare”**. Disponível em: https://www.embrapa.br/busca-de-noticias/-/noticia/77979989/produtividade-media-dos-cafes-do-brasil-estimada-para-2023-e-de-29-sacas-por-hectare.
Acesso em 20.jun.2023.

FERREIRA, Alessandro (2017). “**Redes Neurais Convolucionais Profundas na Detecção de Plantas Daninhas em Lavoura de Soja”**. Disponível em https://repositorio.ufms.br/bitstream/123456789/3101/1/Redes%20Neurais%20Convolucionais%20Profundas%20na%20Detec%c3%a7%c3%a3o%20de%20Plantas%20Daninhas%20em%20Lavoura%20de%20Soja.pdf. Acesso em: 08.mai.2023.

KANDEL, Ibrahem (2020). “**The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset**”. Disponível em: https://www.sciencedirect.com/science/article/pii/S2405959519303455. Acesso em: 09.mai.2023.

MOHAMED, hab (2017). “**Detection and Tracking of Pallets using a Laser Rangefinder and Machine Learning Techniques**”. Disponível em: https://www.researchgate.net/publication/324165524_Detection_and_Tracking_of_Pallets

_ using_a_Laser_Rangefinder_and_Machine_Learning_Techniques. Acesso em: 09.mai.2023.

NATIONAL LIBRARY OF MEDICINE (2021). “**Conjunto de dados de imagens de folhas de café arábica para detecção e classificação de doenças foliares”**. Disponível em https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8165403/. Acesso em: 08.mai.2023.

OMAR, Ahmad (2020). “**Convolutional Neural Network and Regularization Techniques with TensorFlow and Keras**”. Disponível em: https://medium.com/intelligentmachines/convolutional-neural-network-and-regularization-techniques-with-tensorflow-and-keras-5a09e6e65dc7. Acesso em: 10.mai.2023.

PACHECO, André (2022). “**Introdução ao Scikit-learn - Parte 1: uma visão geral”**. Disponível em: http://computacaointeligente.com.br/outros/intro-sklearn-part-1/. Acesso em 09.mai.2023.

RAMEZAN Christopher, WARNER Timothy, MAXWELL Aaron (2019). “**Evaluation of Sampling and Cross-Validation Tuning Strategies for Regional-Scale Machine Learning Classification**”. Disponível em: https://www.mdpi.com/2072-4292/11/2/185. Acesso em: 12.mai.2023.

ROLNICK David, VEIT Andreas, BELONGIE Serge, SHAVIT Nir (2017): **“Deep Learning is Robust to Massive Label Noise”**. Disponível em https://arxiv.org/abs/1705.10694. Acesso em: 09.mai.2023.

SANTIAGO, Luiz (2018). “**Entendendo a biblioteca NumPy”**. Disponível em: https://medium.com/ensina-ai/entendendo-a-biblioteca-numpy-4858fde63355. Acesso em 09.mai.2023.

SAXENA, Shipra (2021). “**Introduction to Batch Normalization**”. Disponível em: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/. Acesso em: 12.mai.2023.

SHAH, Saily (2022). “**Convolutional Neural Network: An Overview**“. Disponível em: https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/. Acesso em: 10.mai.2023.

SHARMA, Sagar (2017). “**Activation Functions in Neural Networks**”. Disponível em: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Acesso em: 12.mai.2023.