Diagnóstico do Cancro Europeu da Macieira com Redes Neurais Convolucionais

Kevin Mondadori Mattos[1], **Yan Felipe Burigo Ribeiro**[1], **Carlos Andres Ferrero**[1], **Wilson Castello Branco Neto**[1]

[1]Instituto Federal de Santa Catarina (IFSC) - Campus Lages
Lages - SC - Brasil
R. Heitor Villa Lobos, 225 - Sao Francisco, Lages - SC, 88506-400

{kevin.mm, yan.br}@aluno.ifsc.edu.br

{andres.ferrero, wilson.castello}@ifsc.edu.br

Abstract. *In 2002 the Apple Canker reached the orchards of the southern region of Brazil. This disease causes a high economic impact and therefore it is essential that there is a quick and assertive diagnosis. In this paper the use of convolutional neural networks to identify the disease through image analysis is proposed. Different network models were tested with 135 images obtained from Epagri and the Cancontrol platform. The results showed that the VGG model had an accuracy of 0.909 and a recall of 0.769. The main limitation of this work is the small amount of images available to train and test the model, but even so, it is a promising solution to help fight and eradicate the disease.*

Resumo. *Em 2002 o Cancro Europeu da Macieira chegou aos pomares da região sul do Brasil. Esta doença causa um impacto econômico alto e por isso é fundamental que haja um diagnóstico rápido e assertivo. Neste trabalho é proposto o uso de redes neurais convolucionais para a identificação da doença através da análise de imagens. Foram testados diferentes modelos de redes com 135 imagens obtidas junto à Epagri e da plataforma Cancontrol. Os resultados mostraram que o modelo VGG obteve uma precisão de 0.909 e um recall de 0.769. A principal limitação do trabalho é a pequena quantidade de imagens disponíveis para treinar e testar o modelo, mas mesmo assim a solução já se demonstra promissora para ajudar a combater e erradicar esta doença.*

1. Introdução

O Cancro Europeu da Macieira (CEM) é uma doença que atinge pomares em várias regiões do mundo (Weber, 2014). Ele é causado pelo fungo *Neonectria ditissima* e surgiu no Brasil pela primeira vez em 2002. O fungo apareceu de forma desconhecida em um viveiro no estado do Rio Grande do Sul, se espalhando por grande parte do estado e infectando também os estados de Santa Catarina e Paraná (Alves e Czermainski, 2015).

A produção de maçãs no Brasil está toda contida no sul do país. O estado de Santa Catarina é responsável por 52.6% da produção nacional, tendo como seu principal produtor o município de São Joaquim. O Rio Grande do Sul é responsável por 44.8% da produção nacional, majoritariamente com sua produção no município de Vacaria, e 2.6% vem do estado do Paraná (Kist, 2019). Das áreas plantadas, 98.8% estão muito próximas

umas das outras, fazendo com que o CEM consiga se espalhar rapidamente, caso não seja monitorado e tratado adequadamente (Araujo et al., 2019).

Um levantamento realizado pela Secretaria de Estado de Agricultura, Pecuária e Irrigação do Rio Grande do Sul (SEAPI), no qual foram analisadas 277 propriedades entre 2015 e 2016, indica que a doença já atinge cerca de 51% dos pomares do estado. Somente no município de Vacaria, onde há maior incidência de CEM, das 46 unidades produtivas (pomares) de macieira amostradas, 96% apresentavam plantas com sintomas de Cancro Europeu e com incidências maiores de 1%, mostrando a rápida propagação do fungo *Neonectria ditissima* (Araujo et al., 2019).

Apesar do estado do Rio Grande do Sul ser o mais afetado, Santa Catarina e Paraná também sentem o impacto da doença em sua produção. Em Santa Catarina, o CEM chegou por volta de 2012, segundo os levantamentos da Companhia Integrada de Desenvolvimento Agrícola de Santa Catarina (CIDASC). Cinco anos após a chegada da doença, o estado já possuía cerca de 159 casos positivos (plantas contaminadas) em 1567 unidades produtivas de macieira. Dos casos ativos no estado, a maior parte está localizada no município de São Joaquim (Araujo et al., 2019).

Embora o estado catarinense já seja impactado pela doença, seus casos ainda são baixos. De acordo com dados levantados pela CIDASC, somente 10% das unidades produtoras estão infectadas e 93% destas com incidência inferior a 1% (Araujo et al., 2019). Estes dados baixos de CEM no estado possibilitam pensar em medidas de erradicação da doença, algo mais difícil de realizar no Rio Grande do Sul devido a sua alta contaminação, sendo mais viável a convivência e controle da doença.

Já no Paraná, dados levantados pela Agência de Defesa Agropecuária do Paraná (ADAPAR) nos anos de 2015 e 2016, mostram que foram identificados apenas 11 casos positivos de cancro em 59 unidades produtoras avaliadas, sendo que em algumas áreas com maior incidência o cancro já foi ou está em processo de erradicação (Araujo et al., 2019).

De acordo com alguns estudos sobre o ciclo de vida do CEM realizados e publicados pela Empresa Brasileira de Pesquisa Agropecuária (Embrapa), a infecção do cancro só pode ser estabelecida por meio de ferimentos, sejam eles causados pelo homem, por outras pragas e doenças ou por eventos climáticos (Alves e Czermainski, 2015). Além disso, um experimento realizado em condições de viveiro, mostram que os sintomas podem demorar de três meses até três anos para aparecerem (McCracken, 2003).

Segundo Lazzarotto e Alves (2015), "nas condições ambientais brasileiras, a doença tem se manifestado de maneira bastante agressiva", acarretando prejuízos em curto, médio e longo prazo. Os mesmos autores afirmam que o CEM pode inviabilizar a produção de maçãs nas regiões atingidas e projetam os resultados econômicos de quatro cenários distintos em relação ao nível de infecção e às medidas de controle adotadas:

• Cenário 1 - sem incidência: a lucratividade de um pomar é de 21.91% e sua vida útil de 20 anos;

• Cenário 2 - incidência baixa: a lucratividade cai para 17.33% e a vida útil permanece em 20 anos;

• Cenário 3 - incidência média a alta com medidas de controle: a vida útil per-

manece em 20 anos, mas a lucratividade cai para 8.86% em função do alto custo para controle da doença, bem como da redução de produtividade;

• Cenário 4 - incidência média a alta sem medidas de controle: além da lucratividade cair para 5.57%, a vida útil do pomar é de apenas 12 anos.

Segundo Werner (2019), a presença do CEM nos pomares do Sul do Brasil pode elevar o custo de produção em até 28% para os fruticultores. No ano de 2019, quando o custo de produção girava em torno de R$ 30.000,00 por hectare a cada ano (ha/ano), a presença desta doença levaria a um acréscimo de R$ 8.526,89 ha/ano. Ao extrapolar para a área total de macieira plantada no Brasil, que era de 33.235 ha (CEPA, Centro de Socioeconomia e Planajamento Agrícola, 2017) este aumento no custo de produção para controle de uma única doença chegaria a marca R$ 283.391.189,20 por ano.

Para conseguir realizar o controle da doença, evitando que ela se propague entre pomares, é recomendado o uso de fungicidas protetores, porém nem os melhores fungicidas conseguem controlar 100% da doença, pois eles não são capazes de curar as plantas já infectadas. Por isso, é recomendado pela Epagri a remoção e incineração de frutos e galhos infectados, além da erradicação da planta toda em propriedades com incidência menor de 5%.

Para realizar o diagnóstico correto da doença, os produtores e responsáveis técnicos realizam a coleta do material para que seja feita a análise pelos fitopatologistas[1] da Epagri. Esse processo pode acabar desmotivando alguns produtores, pois nem todos moram perto da Epagri ou de outra instituição que realiza este análise. Para facilitar esse processo, foi desenvolvida pelo Instituto Federal de Santa Catarina (IFSC), em parceria com a Epagri, a plataforma Cancontrol. Com ela, o produtor pode tirar fotos através do aplicativo e enviar para a análise dos fitopatologistas, sendo notificado do resultado posteriormente (Branco Neto et al., 2021).

A foto e outros dados do monitoramento realizado são enviados para uma API[2] que os salva em um banco de dados. Esta mesma API apresenta os dados em um site disponibilizado para os fitopatologistas da Epagri que podem avaliar as imagens e dar o diagnóstico positivo ou negativo para a presença de CEM. Caso não fique claro através das imagens, os fitopatologistas podem solicitar a análise presencial da infecção. Além disso, são disponibilizados no Cancontrol diversos conteúdos sobre a doença como notícias, imagens e vídeos, tanto no aplicativo quanto no site, de forma pública e gratuita, com o objetivo de tornar a doença mais conhecida e facilitar o seu diagnóstico pelos próprios produtores ou responsáveis técnicos.

O Cancontrol é uma ferramenta importante e muito útil para a conscientização e diagnóstico do CEM, porém ela ainda depende de uma análise humana para dar o resultado, o que pode demorar. Neste caso, modelos de visão computacional podem auxiliar diagnosticando alguns casos de forma simples e rápida, o que reduz a carga de trabalho dos fitopatologistas.

Modelos de visão computacional auxiliam computadores a processar imagens ou

[1] Profissional especializado no estudo das doenças de plantas.

[2] API Application Programming Interface ou Interface de programação de aplicações, é um conjunto de definições e protocolos para criar e integrar softwares de aplicações ou plataforma baseada na Web.

dados multidimencionais. Eles são capazes de interpretar informações através da análise de características das imagens como os pixeis, luminosidade, cores, formas, entre outras. Esses algoritmos podem ser utilizados no Cancontrol para agilizar ainda mais o processo, emitindo o diagnóstico para a presença do fungo ao analisar fotos tiradas pelos produtores sem precisar aguardar pelo diagnóstico dos fitopatologistas.

Este trabalho tem como objetivo desenvolver um novo módulo para o Cancontrol que seja capaz de identificar o CEM por meio da análise de imagens, usando Redes Neurais Convolucionais. Para atingir esse objetivo são definidos os seguintes objetivos específicos:

- Preparar as imagens e treinar diferentes modelos de classificação.
- Selecionar e avaliar os modelos para identificar o CEM.
- Integrar o modelo com o aplicativo do Cancontrol.

As próximas seções do documento estão dispostas da seguinte forma. Na seção 2 é apresentado o referencial teórico necessário para o entendimento do trabalho. Na Seção 3 é detalhado o desenvolvimento do projeto e na Seção 4 são apresentados e discutidos os resultados. Por fim, na Seção 5 são apresentadas as conclusões e algumas sugestões de trabalhos futuros.

2. Referencial Teórico

Na subseção 2.1 são apresentadas as funcionalidades do aplicativo Cancontrol. Após isso, os conceitos fundamentais para entendimento sobre redes neurais e sua importância para este trabalho são abordados na subseção 2.2. Por fim, na subseção 2.3 são apresentados trabalhos que foram realizados com a finalidade de resolver problemas semelhantes utilizando redes neurais.

2.1. Cancontrol

O Cancontrol[3] é uma plataforma composta por um aplicativo e um site. Nos dois sistemas é possível encontrar notícias, informações de conscientização, imagens e vídeos explicativos, sem a necessidade de autenticação.

O Cancontrol foi desenvolvido utilizando a arquitetura descrita na Figura 1. Foi criado um banco de dados relacional em PostgreSQL, onde as informações dos produtores e responsáveis técnicos cadastrados são armazenadas, bem como os monitoramentos realizados por eles. A API é responsável por acessar o banco e realizar as regras de negócio a partir das informações enviadas pelos usuários por meio do aplicativo e do site (Branco Neto et al., 2021).

O aplicativo do Cancontrol possui duas versões, uma desenvolvida de forma nativa para o sistema operacional Android utilizando a linguagem Kotlin e a outra desenvolvida para o sistema IOS utilizando o Flutter, que é um *framework* para desenvolvimento multiplataforma criado pela Google que utiliza a linguagem Dart. Estes aplicativos são usados pelos produtores para enviar suas informações e as informações dos monitoramentos para a API, que foi desenvolvida em JavaScript, sendo executada sobre o NodeJS.

[3]App Store: https://apps.apple.com/br/app/cancontrol/id1568502826
Play Store: https://play.google.com/store/apps/details?id=br.edu.ifsc.cancontrol
Site: http://www.cancroeuropeu.com.br/

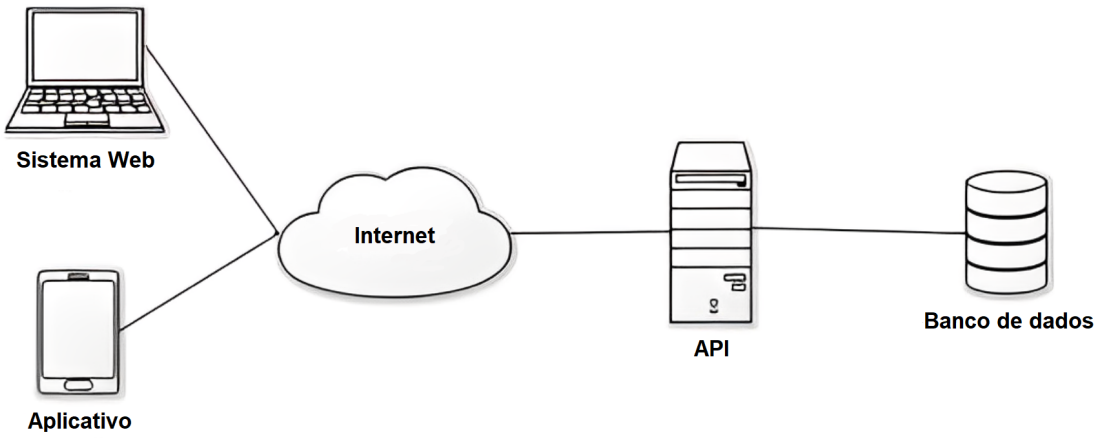

Figura 1. Arquitetura da plataforma

Fonte: Branco Neto et al. (2021)

No aplicativo é possível encontrar algumas informações sobre o CEM, dentre elas notícias, informações de conscientização, imagens e vídeos, conforme mostrados na Figura 2.

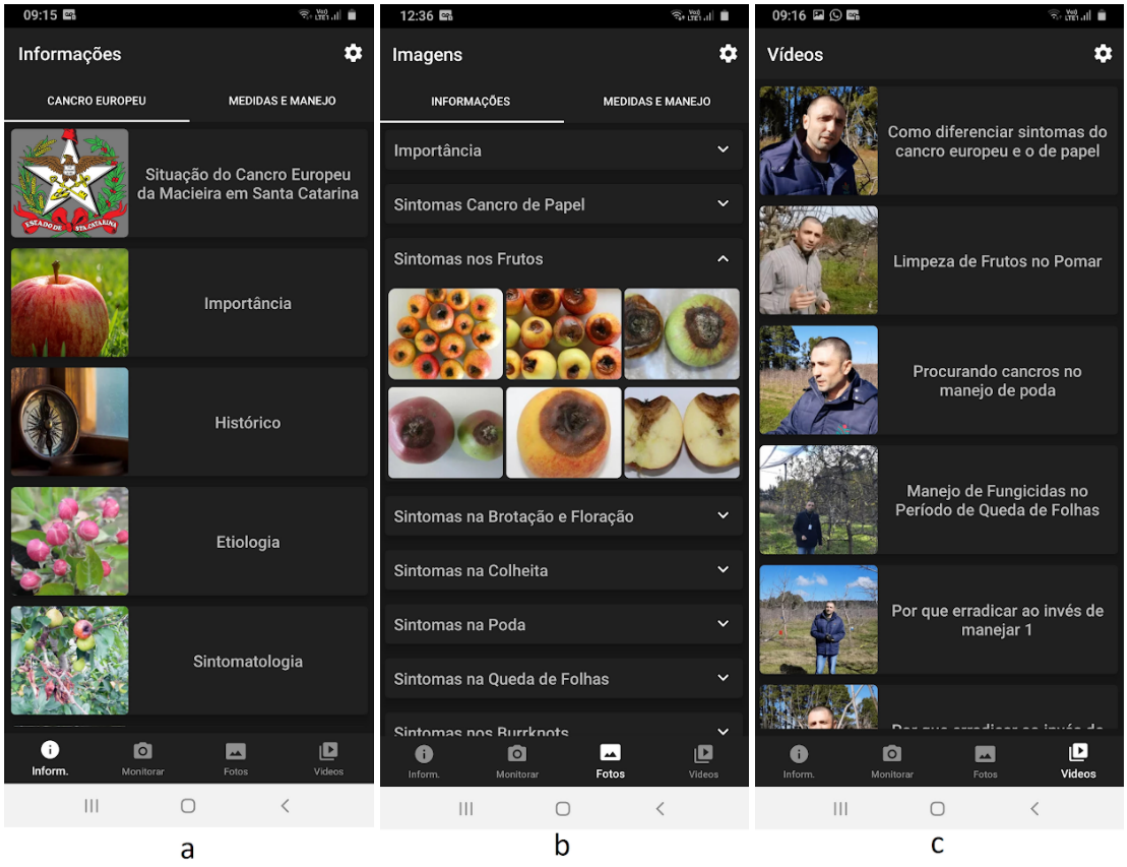

Figura 2. Interface do aplicativo móvel - Informações, Imagens e Vídeos

Fonte: Branco Neto et al. (2021)

Na seção de monitoramento, exclusiva do aplicativo, apresentada na Figura 3, o usuário pode cadastrar suspeitas de infecção enviando as informações necessárias junto com fotos da planta (3a). Também é possível monitorar o status da análise e visualizar o resultado do diagnóstico assim que ficar disponível (3b). No caso de infecção, as coordenadas da planta ficam disponíveis em um mapa (3c). Vale ressaltar que este mapa não é público, ficando visível somente para o usuário que realizou o cadastro e no módulo desenvolvido para os fitopatologistas da Epagri, que exige a necessidade de autenticação para ter acesso. Caso o usuário não esteja conectado à internet, o aplicativo salva as informações para realizar o envio posteriormente. Após os fitopatologistas realizarem a análise, o usuário recebe uma notificação no celular com o resultado.

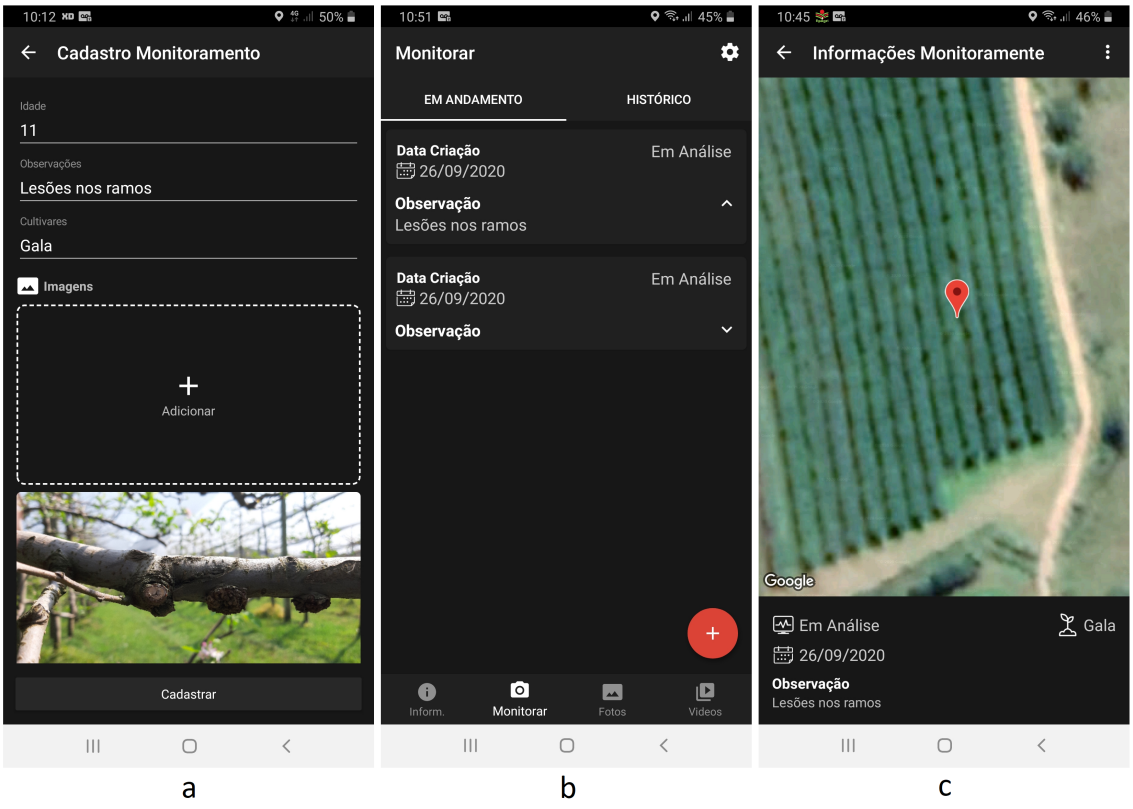

Figura 3. Interface do aplicativo móvel - Monitoramento

Fonte: Branco Neto et al. (2021)

Já o site do Cancontrol foi desenvolvido utilizando Vue.js, que é um *framework* de código aberto que funciona sobre as linguagens JavaScript e TypeScript. No site, além de apresentar as mesmas informações do aplicativo, foi desenvolvido, para os fitopatologistas, um módulo exclusivo, no qual é possível ter acesso aos dados dos monitoramentos. O site utiliza a mesma API do aplicativo e nele é possível dar o diagnóstico do monitoramento ou, caso necessário, pedir uma análise presencial para o produtor.

No site, após realizarem o login, os pesquisadores tem acesso à seção de monitoramento, que contém uma lista de monitoramentos enviados. Ao acessar esses monitoramentos, informações mais detalhadas e imagens são apresentadas. Para facilitar o acompanhamento e a fiscalização, também é possível observar os casos registrados do

CEM em um mapa. Esta interface, apresentada na Figura 4, é utilizada pelos fitopatologistas para analisar as fotos enviadas pelos produtores. É possível emitir 3 resultados:

• Positivo, quando é possível identificar por meio das fotos que se trata do CEM;

• Negativo, quando é a doença não está presente;

• Análise Presencial Necessária, quando não é possível concluir o diagnóstico a partir das fotos. Para obter um resultado, o produtor deve enviar o material para a Clínica Fitopatológica da Estação Experimental de São Joaquim da Epagri (Epagri-EESJ) para que o processo seja feito presencialmente.

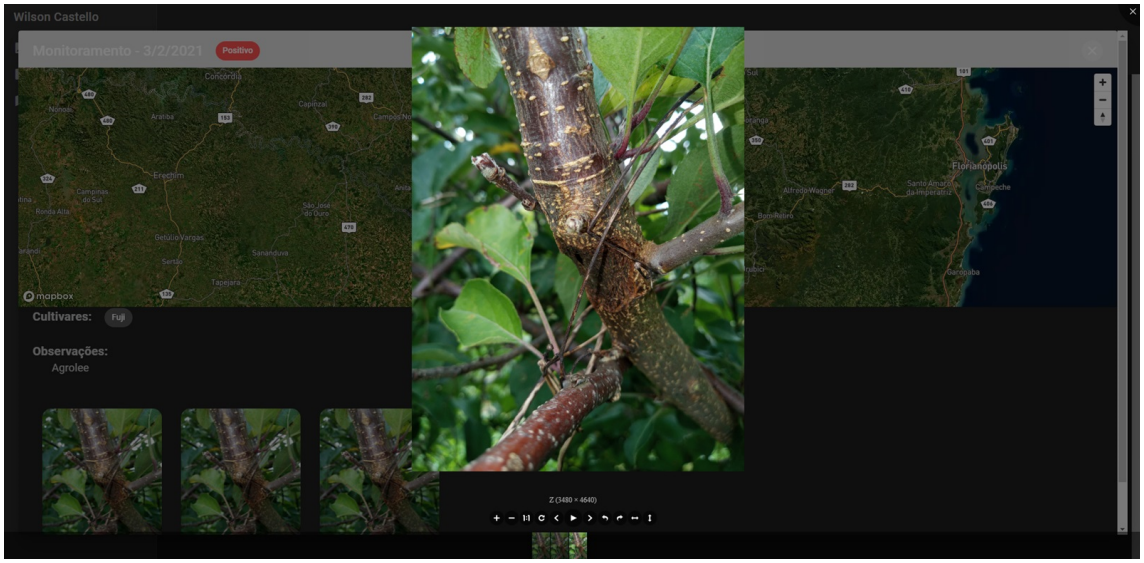

Figura 4. Interface do site - Monitoramento
Fonte: Branco Neto et al. (2021)

O aplicativo auxilia os produtores a controlarem a doença no pomar e ajuda as entidades a monitorarem a propagação na região, porém, ele ainda é dependente da análise dos fitopatologistas. Devido à necessidade de uma análise humana, pode ser que o resultado demore, pois conforme a quantidade de usuários e monitoramentos crescem, os fitopatologistas acabam sendo sobrecarregados. Automatizar este processo de análise diminui o tempo de espera dos produtores, além de diminuir a carga de trabalho dos fitopatologistas, permitindo que eles possam se concentrar somente nos casos mais complexos e que realmente precisem de uma análise presencial.

2.2. Classificação de imagens

Técnicas de aprendizado de máquina podem ser utilizadas para automatizar tarefas. Essas tarefas envolvem problemas de classificação e regressão. Caso o objetivo do modelo seja prever um conjunto finito de dados, como ensolarado, nublado ou chuvoso, esse é um problema de classificação, entretanto, se o objetivo for descobrir o preço de uma casa, será definido como um problema de regressão (Han et al., 2011).

Independente de ser um modelo de regressão ou de classificação, o treinamento deste modelo pode ser feito de diferentes maneiras. Segundo Russell e Norvig (2013) existem três principais tipos de aprendizado:

• Supervisionado: O agente observa alguns exemplos de pares de entrada e saída, e aprende uma função que faz o mapeamento da entrada para a saída.

• Não supervisionado: O agente aprende padrões na entrada através de associações, embora não seja fornecido nenhum *feedback* explícito.

• Por reforço: O agente aprende a partir de uma série de reforços sejam eles recompensas ou punições.

Nas tarefas de aprendizagem supervisionada, todos os dados estão rotulados e o modelo precisa aprender a identificar os padrões a partir desses rótulos (Russell e Norvig, 2013). Aprender a diferenciar árvores saudáveis de doentes é um exemplo de aprendizagem supervisionada, pois o resultado está presente no conjunto de dados de treinamento, ou seja, durante o treinamento, além das imagens de plantas saudáveis e infectadas, é passado para o modelo um rótulo que indica o resultado desejado para cada imagem.

Conforme observado na Figura 5, existe uma função desconhecida que separa os objetos das duas classes. O objetivo dos modelos classificadores é encontrar uma função, chamada hipótese, que se aproxima dessa função desconhecida.

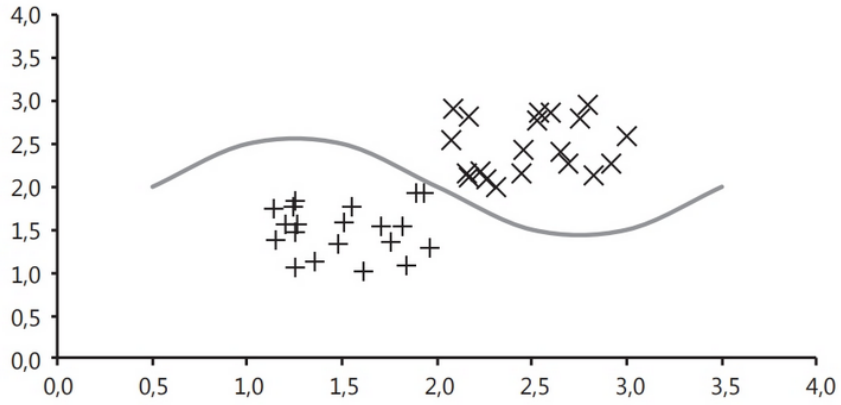

Figura 5. Representação de uma função de classificação

Fonte: Ferrari e Silva (2016)

Existem diversos tipos de classificadores, como classificadores lineares com o limiar rígido. Eles possuem uma fronteira de decisão, que pode ser uma linha definida ou uma superfície, dependendo da dimensão. O limite da decisão linear é chamado de separador linear e os dados utilizados são linearmente separáveis (Russell e Norvig, 2013).

Outro tipo de classificador é o de árvores de decisão. Segundo Amaral (2011), o modelo cria uma estrutura de decisão em formato de árvore. Toda árvore é formada por vários nodos que são particionados de acordo com os dados e o algoritmo selecionado. Cada nova instância percorre a árvore até chegar em um nodo final, onde fica a classe, conforme apresentado na Figura 6.

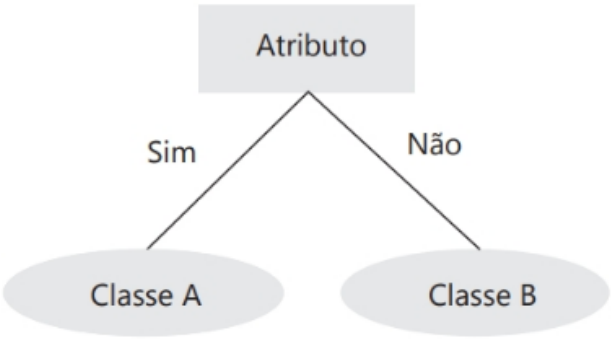

Figura 6. Representação de um modelo baseado em árvores
Fonte: Ferrari e Silva (2016)

Classificadores *K Nearest Neighbor*(KNN) são algoritmos que aprendem por analogias, comparando tuplas que são similares entre si. As tuplas de treinamento possuem n atributos e o modelo separa elas em espaços n-dimencionais baseados nos padrões dos dados. Quando o modelo recebe uma tupla desconhecida, o modelo procura o espaço com as características mais parecidas (Han et al., 2011).

Diferente dos classificadores tradicionais, as Redes Neurais Artificiais (RNA) replicam o funcionamento do cérebro através de funções não lineares para processar informações e identificar padrões mais complexos. Possuem aplicações em diversos problemas de reconhecimento de padrões, como reconhecimento de voz, biometria, reconhecimento facial, detecção de fraude, entre outros (Abiodun et al., 2019).

2.2.1. Redes Neurais

Segundo Haykin (2001), uma rede neural é uma máquina projetada para modelar a maneira como o cérebro realiza uma tarefa particular ou uma função de interesse. De acordo com ele, o trabalho em redes neurais artificiais tem sido motivado desde o começo pelo reconhecimento de que o cérebro humano processa informações de uma forma inteiramente diferente do computador digital convencional.

Para Haykin (2001), o cérebro é um computador (sistema de processamento de informação) altamente complexo, não-linear e paralelo. Ele tem a capacidade de organizar seus constituintes estruturais, conhecidos por neurônios, de forma a realizar certos processamentos muito mais rapidamente que o mais rápido computador digital.

Segundo Russell e Norvig (2013) as redes neurais são compostas por nós ou unidades, mostradas na Figura 7, conectadas por ligações direcionadas. Uma ligação da unidade i para a unidade j serve para propagar a ativação a_i de i para j. Cada ligação também tem um peso numérico $W_{i,j}$ associado a ele, que determina a força e o sinal de conexão. Assim como em modelos de regressão linear, cada unidade tem uma entrada fictícia $a_0 = 1$ com peso associado $W_{0,j}$, conhecido como bias.

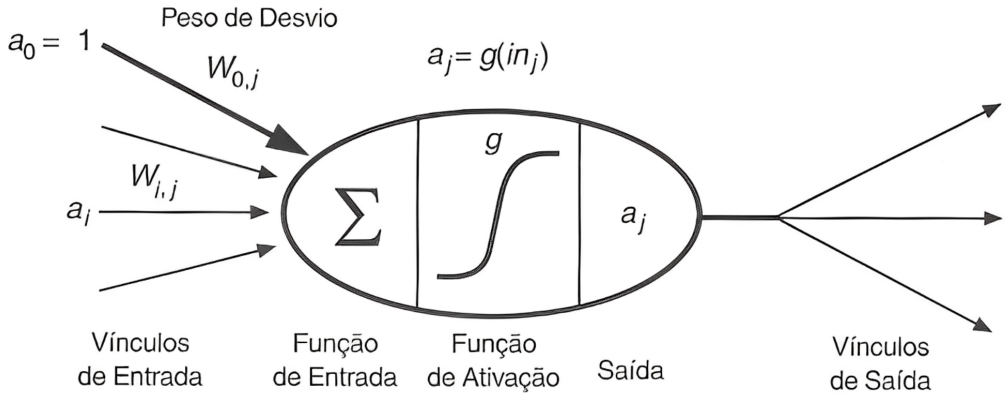

Figura 7. Modelo matemático de neurônio simples

Fonte: Russell e Norvig (2013)

Para Haykin (2001), um neurônio é uma unidade de processamento de informação que é fundamental para a operação de uma rede neural. Na Figura 7 é possível identificar três elementos básicos do modelo neuronal:

1. Um conjunto de sinapses ou elos de conexão, cada uma caracterizada por um peso ou força própria. Ao contrário de uma sinapse do cérebro, o peso sináptico de um neurônio artificial pode estar em um intervalo que inclui valores negativos bem como positivos.

2. Um somador para somar os sinais de entrada, ponderados pela respectivas sinapses do neurônio.

3. Uma função de ativação para restringir a amplitude da saída de um neurônio. A função de ativação é também referida como função restritiva já que restringe (limita) o intervalo permissível de amplitude do sinal a um valor finito. As funções mais usadas nos modelos neurais são a função limiar, função limiar por partes e a função sigmóide, sendo esta a mais usada para construção de redes neurais artificiais de acordo com Haykin (2001).

A organização e posicionamento dos neurônios são fundamentais para a construção de um rede neural, além de estar intimamente ligada com o algoritmo de aprendizagem utilizado para treinar a rede. Existem diversas maneiras no qual se pode realizar a montagem dessa arquitetura, porém, conforme abordado por Haykin (2001), é possível identificar três tipos principais que são utilizados na maior parte das redes:

• Redes alimentadas adiante com camada única: Neste tipo de rede, os neurônios estão organizados na forma de camadas. Na forma mais simples de uma rede em camadas, temos uma camada de entrada de nós de fonte que se projeta sobre uma camada de saída de neurônios, mas não vice-versa. Em outras palavras, esta rede é estritamente do tipo alimentada adiante ou acíclica. Ela é ilustrada conforme a Figura 8a.

• Redes alimentadas diretamente com múltiplas camadas: Esta estrutura de rede se distingue pela presença de uma ou mais camadas ocultas, cujos nós computacionais são chamados correspondentemente de neurônios ocultos ou unidades ocultas. A função dos neurônios ocultos é intervir entre a entrada externa e a saída da rede de uma maneira útil. Adicionando-se uma ou mais camadas ocultas, tornamos a rede capaz de extrair es-

tatísticas de ordem elevada. Em um sentido bastante livre, a rede adquire uma perspectiva global apesar de sua conectividade local, devido ao conjunto extra de conexões sinápticas e da dimensão extra de interações neurais (Churchland e Sejnowski, 1992 apud Haykin, 2001). Esta arquitetura é ilustrada conforme a Figura 8b.

• Redes recorrentes: Uma rede neural recorrente se distingue de uma rede neural alimentada adiante por ter pelo menos um laço de realimentação. Uma rede recorrente pode consistir, por exemplo, de uma única camada de neurônios com cada neurônio alimentando seu sinal de saída de volta para as entradas de todos os outros neurônios, conforme ilustrada na Figura 8c.

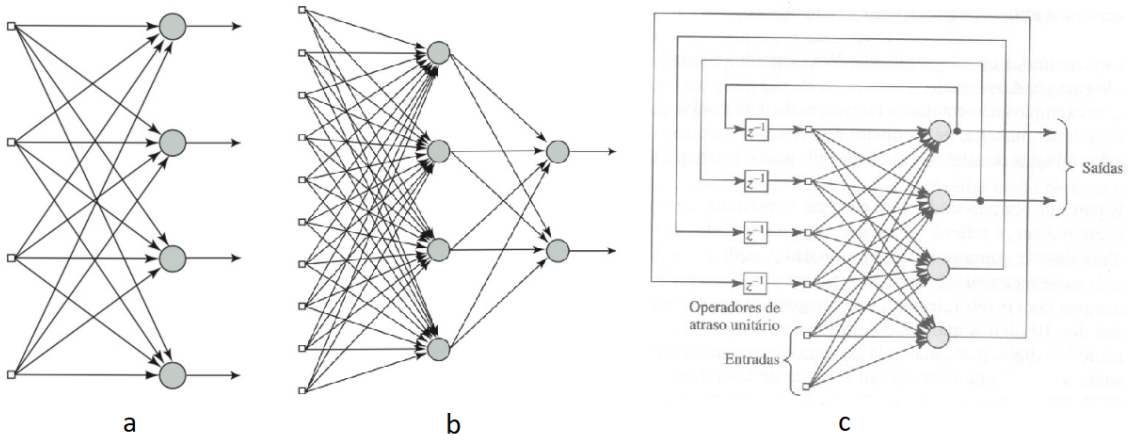

Figura 8. Arquiteturas de redes neurais
Fonte: Haykin (2001)

Além dos neurônios e da arquitetura da rede neural, uma outra propriedade que é de importância primordial é a sua habilidade de aprender a partir de seu ambiente e de melhorar o seu desempenho através da aprendizagem. A melhoria do desempenho ocorre com o tempo de acordo com alguma medida preestabelecida. Uma rede neural aprende acerca do seu ambiente através de um processo interativo de ajustes aplicados a seus pesos sinápticos e níveis de bias. Idealmente, a rede se torna mais instruída sobre o seu ambiente após cada iteração do processo de aprendizagem (Haykin, 2001).

Segundo Abiodun et al. (2019) as redes neurais artificiais são cada vez mais atraentes, eficazes, eficientes e bem-sucedidas em alcançar o reconhecimento de padrões em muitos problemas, como o reconhecimento de imagens. Para ajudar a resolver este tipo de problema foram criadas as redes neurais convolucionais, que são redes especializadas em análise de imagens e reconhecimento de padrões visuais, as quais são utilizadas para o desenvolvimento deste trabalho.

2.2.2. Redes Neurais Convolucionais

As redes neurais convolucionais (CNN) são um grupo de redes neurais artificiais profundas, com várias camadas alimentadas diretamente (*feedforward*), normalmente aplicadas para analisar imagens. A CNN possui três entendimentos estruturais que trazem mudança

em relação às redes neurais convencionais, campos receptivos locais, replicação de peso e subamostragem espacial ou temporal (LeCun et al., 1998).

Elas foram proposta inicialmente por LeCun et al. (1998) e sua arquitetura foi inspirada no funcionamento do córtex visual, onde os neurônios das regiões iniciais são responsáveis por detectar formas geométricas simples na imagem, como cantos e bordas, ao passo que os neurônios das regiões finais detectam formas gráficas mais complexas. Este processo se repete através das regiões até que os neurônios da região final têm a atribuição de detectar características de mais alto nível de abstração, como faces ou objetos específicos. Esse tipo de rede é utilizada quando se deseja encontrar informações importantes que estão implícitas em um conjunto de dados, através de operações de Convolução e *Pooling* (Becker, 2017).

As operações de convolução são realizadas na camada convolucional, que é composta por diversos neurônios, sendo cada um destes responsável por aplicar um filtro em um determinado pedaço da matriz de entrada. Uma convolução consiste na aplicação de uma série de filtros deslizantes em toda a extensão da matriz de entrada, e o resultado da aplicação destes filtros é conhecido como *feature map*. De modo geral, pode-se considerar os parâmetros das camadas convolucionais como um conjunto de *n* filtros capazes de aprender. Estes filtros, que não precisam ser muito grandes, percorrem toda a matriz de entrada. O tamanho de cada deslocamento de um filtro é chamado *stride* (Becker, 2017).

Já as operações de *pooling* são realizadas na camada *pooling*, que implementa uma função não-linear de subamostragem com o objetivo de diminuir a dimensionalidade e capturar pequenas invariâncias. Basicamente, essa técnica reduz a dimensionalidade de um mapa de características fornecido como entrada e produz outro mapa de características, criando assim uma espécie de resumo do primeiro. A cada filtro, é selecionado o maior valor presente (*max-pooling*) ou então é calculada a média dos valores presentes (*average-pooling*) (Becker, 2017). A Figura 9 ilustra como essa operação ocorre.

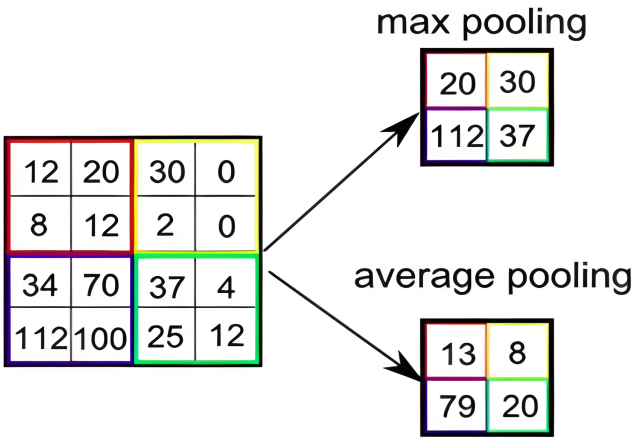

Figura 9. Exemplo de max pooling e average pooling
Fonte: Canto (2019)

Ao final deste processo são geradas diversas camadas com fragmentos menores da imagem inicial, o que possibilita reduzir a dimensionalidade do problema e realçar

as partes mais representativas da imagem para o problema de classificação. A imagem 10 mostra um exemplo de uma imagem de 28 por 28 *pixels* passando pelas camadas de convolução e *pooling* respectivamente.

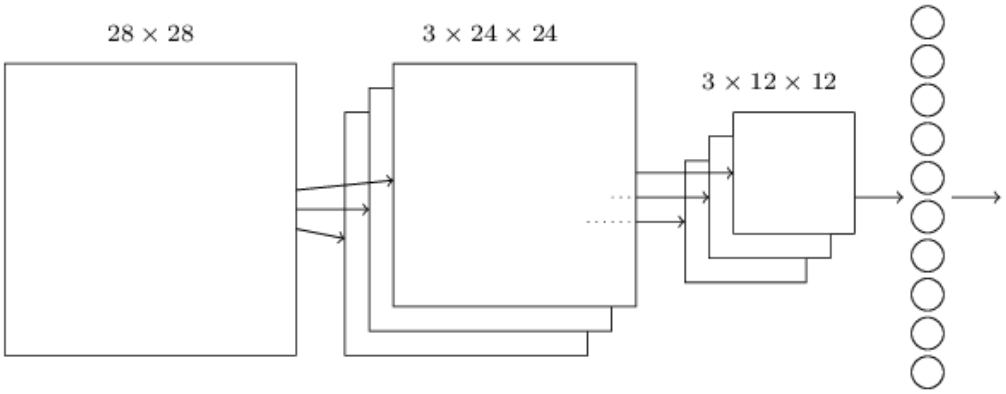

Figura 10. Arquitetura básica de uma CNN
Fonte: Data Science Academy (2022)

Após as camadas convolucionais e de *pooling*, normalmente, são encontradas as camadas totalmente conectadas. Nessas camadas os *features* extraídos nas camadas de convolução anteriores são utilizados para se ter a saída de cla ssificação da rede, conforme apresentadas na Figura 11 (Rodrigues e Batista, 2018).

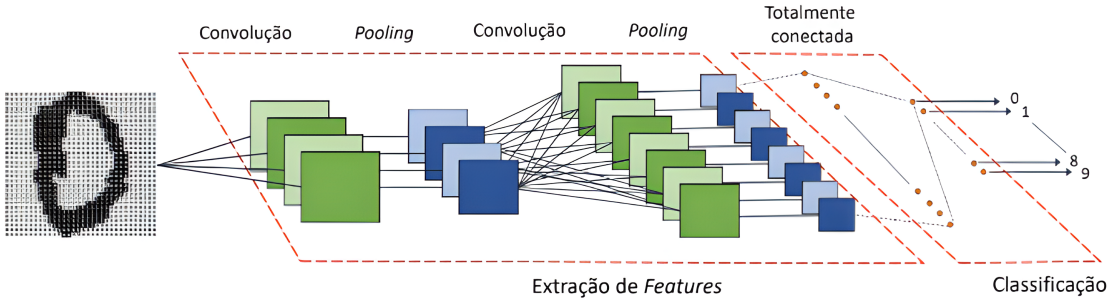

Figura 11. Arquitetura completa de uma CNN baseada na LeNet
Fonte: Becker (2017)

CNNs possuem uma grande quantidade de parâmetros, e por isso tem um um custo alto de treinamento se feito do ínicio. Para isso, através de um processo conhecido como *fine-tuning*, uma rede pré-treinada é utilizada e somente as camadas finais de classificação são treinadas (Yu, 2016).

A rede neural da Figura 11 tem como objetivo identificar dígitos manuscritos, por isto ela possuiu dez neurônios de saída, um para cada digito que pode ser classificado, já a rede utilizada neste trabalho tem dois neurônios de saída, positivo e negativo para o diagnóstico do CEM.

2.3. Trabalhos similares

Para buscar os trabalhos similares foi feita uma revisão sistemática através da plataforma *Google Scholar*. A *string* de pesquisa utilizada foi *"Convolutional Neural Network for Plant Disease Diagnosis"* e foram filtrados trabalhos realizados entre 2012 à 2022. Após os trabalhos serem buscados, os *abstracts* dos vinte primeiros artigos foram lidos e os cinco trabalhos considerados mais relevantes pelos autores foram selecionados para uma leitura completa. Vale ressaltar que não foi encontrado nenhum trabalho com o objetivo de diagnosticar o CEM, por isto foram analisados trabalhos que visam o diagnóstico de outras doenças usando Redes Neurais Convolucionais.

Abade et al. (2021) realizaram um estudo sobre os 121 trabalhos sobre a análise neural convolucional de fitopatologias no geral. De acordo com suas análises, destes 121 trabalhos, 25 propuseram novos métodos, os demais visam realizar melhorias nas arquiteturas existentes por meio da sua customização, ajustes dos hiperparâmetros, etc. Já em relação ao conjunto de dados utilizado, 66 trabalhos utilizam um conjunto customizado e 44 utilizaram o conjunto *PlantVillage*[4]. Sobre o tipo de sensores e perspectiva para a classificação, a maioria (88 trabalhos) utiliza imagens *RGB* com visão *Top View*. Sobre os *frameworks* mais utilizados, o *TensorFlow* ficou em primeiro lugar com 38 utilizações, o *Keras* ficou em segundo com 25 utilizações, já o *PyTorch*, *framework* utilizado neste trabalho, ficou em quinto lugar com 10 utilizações. Na questão dos algoritmos predominantes de CNN o primeiro lugar foi *AlexNet* com 27 utilizações, seguido por *VGG* com 22. Vale ressaltar que a maçã é a segunda cultura mais investigada e o agente causador que mais foi abordados nos trabalhos foram fungos.

De acordo com Abade et al. (2021), ao analisar as categorias com customização de camadas e novas arquiteturas, fica evidente que para abordar os desafios relacionados à identificação de doenças de plantas, os modelos gerados tomam um caminho inverso das arquiteturas tradicionais. Enquanto arquiteturas tradicionais, como, por exemplo, *AlexNet*, *VGG16* e *ResNet* construíram sua notoriedade por sua capacidade de generalização, os modelos investigados tendem a ser cada vez mais especializados em um tipo específico de cultivo. Este fenômeno pode ser explicado, considerando que as arquiteturas e modelos utilizados para identificar doenças de plantas e, consequentemente, os conjuntos de dados utilizados, só podem estabelecer um padrão para a extração de características discriminantes usando as características morfológicas da planta, considerando que um determinado patógeno pode causar diferentes sintomas e lesões dependendo do tipo de cultura avaliada.

No trabalho desenvolvido por Toda e Okura (2019) é feita uma avaliação de uma série de métodos de visualização para interpretar a representação de doenças de plantas que as CNNs conseguem diagnosticar. Estas visualizações são utilizadas para poder selecionar o melhor método de classificação possível para o diagnóstico das doenças.

Toda e Okura (2019) também propuseram diversas arquiteturas de redes neurais convolucionais, uma delas que obteve bastante destaque foi uma baseada na *GoogLeNet*, a qual contém apenas 5.167.878 parâmetros, enquanto o modelo original possui 21.880.646. O modelo proposto conseguiu apresentar 97.14% de precisão e valor de perda de 0.097, que foram equivalentes aos do modelo original (97.15% e 0.098, res-

[4]Dados da *PlantVillage* estão disponíveis em: https://www.kaggle.com/datasets/emmarex/plantdisease

pectivamente), ou seja, 75% dos parâmetros no modelo inicial podem ser omitidos sem degradação do desempenho. Quando usado em situações práticas, como diagnóstico de plantas com dispositivos móveis, a redução do número de parâmetros de rede é importante para a eficiência de memória e cálculo.

Em um trabalho realizado por Liu et al. (2018), foi proposta uma nova arquitetura de rede neural convolucional utilizada para detectar as seguintes doenças da macieira: manchas de alternaria, mosaico da macieira, ferrugem e mancha de marssonina. Neste trabalho foi comparado o desempenho da arquitetura proposta pelos autores com as arquiteturas mais utilizadas de CNN, como a *AlexNet*, *GoogLeNet*, *VGGNet-16* e *ResNet-20*.

Esta arquitetura proposta por Liu et al. (2018) teve uma precisão de 97.62% no conjunto de testes, que é superior aos demais modelos. Além disso, o modelo *AlexNet* tem uma boa capacidade de reconhecimento e obtém uma precisão média de 91.19%. Já o *GoogLeNet*, que tem várias *Inceptions* e possui a habilidade para extração de recursos multidimensionais, teve uma taxa de reconhecimento final de 95.69%. O *ResNet-20* obteve uma precisão de 92.76% e o *VGGNet-16* obteve uma taxa de reconhecimento de 96.32%. Além disso, o modelo SVM com o otimizador SGD e redes neurais *backward propagation* (BP) obtiveram uma precisão de 68.73% e 54.63%, respectivamente.

Elhassouny e Smarandache (2019) propõem um modelo que pode ser utilizado em celulares ou sistemas embarcados para reconhecer doenças na folha do tomate, pois os métodos existentes que utilizam redes neurais necessitam de computadores com alto poder de processamento. Para isso, foi proposto neste estudo um modelo baseado na rede *Mobilenet*, que foi desevolvida por pesquisadores da Google e necessita de oito à nove vezes menos poder de computação. Eles adaptaram a *Mobilenet* para reconhecer os 10 tipos mais comuns de doenças na folha, utilizando 7176 imagens, e obtiveram uma taxa de reconhecimento de até 90.7%, dependendo da taxa de aprendizado.

Agarwal et al. (2020) também analisam a folha do tomate para identificar doenças, porém, diferente de Elhassouny e Smarandache (2019), eles propuseram uma nova arquitetura que não utiliza nenhum modelo pré-treinado e compararam com outros modelos que usaram modelos pré-treinados com a *Mobilenet*, *VGG 16* e *InceptionV3*. No modelo proposto, o número de parâmetros é o menor entre os modelos comparados, o que contribui para que o armazenamento necessário seja menor, utilizando somente 1.6 KB, enquanto o *Mobilenet*, que foi o segundo que menos requer armazenamento, utiliza 86 MB. Com relação a acurácia também conseguiram resultados positivos, 91.2%, comparando com 63.75% do modelo que utiliza a *Mobilenet*, 77.2% do *VGG 16* e 63.4% do *InceptionV3*.

No Quadro 1 é possível observar as principais características analisadas em cada um dos trabalho que utilizaram redes neurais convolucionais para a identificação de fitopatologias. Neste quadro, é possível comparar os trabalhos que utilizaram ou não uma arquitetura própria, quais que foram as precisões obtidas, qual o modelo que foi utilizada na construção da rede neural e a doença na qual o trabalho se propôs a diagnosticar.

Como esses trabalhos abordam doenças e culturas diferentes e nenhum tem como objetivo o diagnóstico do CEM, comparar a acurácia ou outras medidas de desempenho que cada um obteve não é relevante para o desenvolvimento do trabalho atual, porém, o que pode ser observado foi o caminho utilizado para obter os resultados.

Quadro 1: Trabalhos que utilizaram CNN para diagnóstico de fitopatologias

	How Convolutional Neural Networks Diagnose Plant Disease	**Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks**	**Smart mobile application to recognize tomato leaf diseases using Convolutional Neural Networks**	**ToLeD: Tomato Leaf Disease Detection using Convolution Neural Network**
Propôs arquitetura própria	Sim	Sim	Não	Sim
Precisão obtida	97.14%	97.62%	90.7%	91.2%
Modelos	Baseado na GoogLeNet	Próprio	MobileNet	Próprio
Doenças	Fitopatologias no geral	Doenças da macieira	Doenças nas folhas do tomate	Doenças nas folhas do tomate

Apesar dos estudos que propuseram novas arquiteturas alcançarem melhores resultados, arquiteturas já consolidadas trazem maior agilidade para desenvolver os modelos, por isso, uma rede pré-treinada será utilizada inicialmente, mas posteriormente isso pode ser adaptado.

3. Materiais e Métodos

Quanto a sua natureza, este trabalho classifica-se como pesquisa aplicada, com abordagem do problema quantitativa. Sob o ponto de vista de seus objetivos, é uma pesquisa descritiva e quanto aos procedimentos técnicos é uma pesquisa bibliográfica e experimental.

Este trabalho foi desenvolvido em três etapas. A primeira etapa consiste em preparar imagens do CEM com a finalidade de treinar a rede neural para que ela seja capaz de identificar plantas contaminadas pelo fungo *Neonectria ditissima*. Foram utilizadas imagens obtidas pelos fitopatologistas da Epagri ou enviadas pelos produtores que utilizam o Cancontrol.

A segunda etapa consiste em treinar e avaliar diferentes modelos para que o modelo com melhor performance seja utilizado. Para realizar este treinamento foi desenvolvido uma rede neural utilizando a linguagem de programação Python, que possui uma vasta variedade de bibliotecas que auxiliaram na construção e avaliação dos diferentes modelos utilizados neste trabalho. A avaliação foi realizada comparando algumas métricas como a acurácia, precisão, *F-Score*, entre outros.

A terceira etapa consiste na integração do aplicativo Cancontrol com a rede neural convolucional criada. Nela foi desenvolvido um módulo em uma API separada que realiza a análise das imagens usando a rede neural. Sua vantagem é que por estar localizada em um servidor, seu processamento é mais rápido e por estar separada da API do Cancontrol, suas atualizações e melhorias são mais simples de realizar. Este módulo foi integrado com a API do Cancontrol. Quando este recebe as requisições de análise, é feita uma chamada

para a API da rede neural, que realiza e retorna o diagnóstico do CEM para a plataforma já existente.

3.1. Obtenção e Preparação das Imagens

Para as atividades de treinamento e teste da rede neural convolucional, obteve-se um conjunto com 135 imagens, deste total, 58 imagens foram obtidas junto aos fitopatologistas da Epagri e as outras 77 imagens foram retiradas da base de dados que armazena as imagens enviadas pelos produtores por meio do aplicativo Cancontrol. A tabela 1 mostra a origem das imagens e a quantidade de casos positivos e negativos presente no *dataset*.

Tabela 1. Números de imagens utilizadas no projeto

Origem das Imagens	Resultado Positivo	Resultado Negativo	Total
Fitopatologistas Epagri	34	24	58
Cancontrol	28	49	77
Total	62	73	135

3.2. Indução de Modelos de Classificação

Para realização do treinamento da rede neural foram selecionados diversos modelos pré-treinados disponibilizados pelo *framework torchvision*[5], são eles: *ResNet*, *VGG16* e *VGG*, *AlexNet*, *SqueezeNet* e *DenseNet*, além das versões de *Mobilenet* como a *v2*, *v3 small* e *v3 large*. Estes modelos são todos pré-treinados e obtiveram bons resultados para a classificação de imagens em geral. Em muitos casos, utilizar modelos pré-treinados acaba sendo mais vantajoso, pois para treiná-los do zero é necessário um grande número de imagens, fazendo com que o processo demore mais tempo. Isso também facilita o trabalho, pois eles já conseguem extrair características mais gerais das imagens, como bordas, texturas, formas e cores, para aplicar no processo de treinamento. Assim, é possível transferir esse conhecimento através de um processo chamado de *Fine-Tuning*, no qual a rede continua a ser treinada com um *dataset* mais objetivo e pode ser utilizada para outros problemas de classificação.

Primeiramente, conforme mostrado na Figura 12, alguns parâmetros globais que controlam o processo de treinamento devem ser definidos. Dentre eles estão o *batch size*, que define a quantidade de amostras de dados que serão utilizados para o treinamento da rede antes que a rede propague os pesos internos para realizar o treinamento de uma nova amostra, e *epochs* que define quantas vezes o modelo será treinado com o *dataset* todo. O conjunto de imagens de treinamento é separado em *batches* de imagens e, cada vez que um *batch* de imagens passa pela rede os pesos da rede são ajustados. Um *epoch* de treinamento da rede neural é concluída quando todo o conjunto de imagens tiver passado pela rede. Neste trabalho foi utilizado um número máximo de 10 *epochs* de treinamento e cada oito imagens formam um *batch*.

Após essa configuração inicial, são criados os *data loaders*, que realizam o carregamento das imagens para o processamento neural. Eles são divididos em dois, um para os dados de treino e outros para os dados de testes, além de receberem o *batch size*, conforme mostrado na linha 4 da Figura 12.

[5]Documentação disponível em: https://pytorch.org/vision/stable/index.html

```
1   batch_size = 8
2   num_epochs = 10
3   ...
4   neuralLoader = preLoading.dataLoaders(batch_size,
5                                         neuralData.train_data,
6                                         neuralData.test_data)
7   ...
8   model_ft, input_size = nm.initialize_model(neural_model_name,
9                                              num_classes,
10                                             feature_extract,
11                                             use_pretrained = use_pretreined)
12  ...
13  modelFitData = nm.optimize_model(device.device, feature_extract, model_ft)
14  ...
15  model_ft = training.train_and_evaluate(model_ft, num_epochs,
16                                         neuralLoader.train_loader.loader,
17                                         neuralLoader.test_loader.loader,
18                                         modelFitData.optimizer_ft,
19                                         criterion, device.device)
20  ...
21  training.exportModel(model_ft, neural_model_name.name)
```

Figura 12. Código da estrutura principal da CNN

Depois da criação dos *data loaders*, é realizada a inicialização do modelo, conforme a linha 8 da Figura 12. Como cada um tem suas características, a configuração varia entre eles. A Figura 13 mostra o código da parte do método *initialize_model* que realiza a configuração do modelo *DenseNet*.

```
1   if model_name == ModelName.DENSENET:
2       """ Densenet
3       """
4       model_ft = models.densenet121(pretrained=use_pretrained)
5       set_parameter_requires_grad(model_ft, feature_extract)
6       num_ftrs = model_ft.classifier.in_features
7       model_ft.classifier = nn.Linear(num_ftrs, num_classes)
8       input_size = 224
```

Figura 13. Código da configuração do DenseNet

Essa inicialização é bastante semelhante para os diferentes modelos. Inicialmente, é feita a seleção do modelo que será utilizado, conforme mostrada na linha 4, após essa seleção é feita a alteração do parâmetro *require grad* para o valor falso, este parâmetro serve para o caso do modelo exija gradiente. Por fim, defini-se o tipo de classificador, conforme mostrado na linha 7, e o tamanho da entrada de dados, conforme a linha 8. O classificador utilizado é o *linear* para todos os modelos treinados, exceto o *squeezenet*, no qual foi utilizado o *Conv2d*.

Depois da configuração inicial, os modelos são otimizados de acordo com os

parâmetros passados para que apresentem melhores resultados, conforme apresentado na linha 13 da Figura 12. Nessa otimização foi utilizado o gradiente descendente estocástico, que é uma otimização do gradiente descendente. O gradiente descendente é um algoritmo de otimização cuja função é reajustar os parâmetros do modelo. Ele funciona de forma iterativa e minimiza a função para que o modelo se ajuste melhor aos dados que estão sendo utilizados no treino.

Na otimização do modelo, inicialmente são selecionados os parâmetros dos modelos que necessitam de gradientes que são atribuídos a uma lista denominada *params_to_update*. Após essa seleção, é realizada a otimização desse modelo utilizando a função *optiom.SGD* da biblioteca *torch*, a qual recebe a lista de parâmetros, junto com os valores do *learning rate* e um valor para o *momentum*, conforme mostrado na linha 17 da Figura 14

```
1   def optimize_model(device, feature_extract, model_ft) -> ModelFitData:
2       model_ft = model_ft.to(device)
3
4       params_to_update = model_ft.parameters()
5       print("Params to learn:")
6       if feature_extract:
7           params_to_update = []
8           for name,param in model_ft.named_parameters():
9               if param.requires_grad == True:
10                  params_to_update.append(param)
11                  print("\t",name)
12      else:
13          for name,param in model_ft.named_parameters():
14              if param.requires_grad == True:
15                  print("\t",name)
16
17      optimizer_ft = optim.SGD(params_to_update, lr=0.001, momentum=0.9)
18      return ModelFitData(optimizer_ft, model_ft)
```

Figura 14. Código da otimização dos Modelos

Após a otimização do modelos, eles são treinados de fato. Para a realização do treinamento, são passados como parâmetros os *data loaders*, a otimização do modelo, o critério de treinamento, neste caso foi utilizado a *cross entropy loss*, e a informação do *device*, que define se a rede será treinada em CPU ou em núcleos Cuda[6], conforme mostrado na linha 15 da Figura 12.

Por fim, após a finalização do treinamento, são exportadas as métricas e os modelos em formato .pth, os quais são utilizados para a implantação da rede neural, conforme mostrado na linha 21 da Figura 12.

[6]Compute Unified Device Architecture - Plataforma de computação paralela da Nvidia proporcionada por placa de vídeo

3.3. Avaliação dos modelos

Para avaliação e seleção do melhor modelo, o qual será utilizado para a integração com o Cancontrol, foram utilizadas diferentes métricas para avaliar a qualidade na identificação do CEM, dentre elas a acurácia *(accuracy)*, a matriz de confusão, *f1 score*, *recall* e precisão *(precision)*.

A matriz de confusão apresenta o desempenho detalhado do modelo. Por exemplo, em um problema de classificação binária em que existem algumas amostras pertencentes a duas classes: SIM ou NÃO, um classificador prevê uma classe para uma determinada amostra de entrada. Ao testar esse modelo podem ser obtidos quatro resultados diferentes, como apresentado no Quadro 2.

Quadro 2: Exemplo de matriz de confusão

	Predito: NÃO	**Predito: SIM**
Real: NÃO	Verdadeiro Negativo (TN - True Negative)	Falso Positivo (FP - False Positive)
Real: SIM	Falso Negativo (FN - False Negative)	Verdadeiro Positivo (TP - True Positive)

Existem 4 termos importantes para a matriz do confusão:

- Verdadeiros positivos: Os casos em que o modelo previu SIM e a saída real também era SIM.

- Verdadeiros negativos: Os casos em que o modelo previu NÃO e a saída real era NÃO.

- Falsos positivos: Os casos em que o modelo previu SIM e a saída real era NÃO.

- Falsos negativos: Os casos em que o modelo previu NÃO e a saída real era SIM.

A acurácia avalia um modelo com base nas predições corretas que ele realizou. Ela é a razão entre o número de previsões corretas e o número total de amostras de entrada, representada pela Equação 1.

$$Acurácia = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

Embora a acurácia seja uma excelente medida de avaliação, ela não é adequada para medir principalmente *datasets* desbalanceados na matriz de confusão. Para resolver esse problema, foram criadas mais duas métricas: precisão(*precision*) e *recall*.

A precisão avalia a qualidade do modelo com base nos resultados positivos que a rede neural previu. Ela é o número de resultados positivos corretos dividido pelo número de resultados positivos previstos pelo classificador, tanto corretos, quanto incorretos, representada pela Equação 2.

$$Precisão = \frac{TP}{TP + TN} \tag{2}$$

No geral, a precisão determina a proporção de identificações positivas corretas, ou seja, quanto maior a precisão, menor o número de casos positivos avaliados incorretamente.

Já o *recall* considera os acertos de resultados positivos em relação a todos os positivos presente no *dataset*, ele é o número de resultados positivos corretos dividido pelo número de todas as amostras positivas, representado pela Equação 3. De maneira geral, o *recall* determina a proporção de resultados positivos verdadeiros que foram identificados corretamente, ou seja, quanto maior o *recall*, menor é o número de casos positivos identificados como negativos.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

Em muitos modelos, a precisão e o *recall* acabam sendo divergentes, pois em alguns casos que a precisão é alta, o *recall* acaba sendo baixo e vice-versa. Por causa disso, dependendo da finalidade do modelo, acaba sendo difícil classificar se ele é bom ou não utilizando apenas essas duas métricas. Então para solucionar esse problema, foi criado o *f1 score*.

O *f1 score* é a média harmônica entre precisão e *recall*. O intervalo para *f1 score* é [0, 1]. Ele informa o quão preciso é o classificador (quantas instâncias ele classifica corretamente), bem como quão robusto ele é (não perde um número significativo de instâncias). Alta precisão, mas baixo *recall*, fornece uma classificação extremamente assertiva, mas perde um grande número de instâncias não classificadas. Já o inverso, no qual o *recall* é alto e a precisão baixa, fornece uma classificação onde todos os dados relevantes foram classificados, porém perde em relação a assertividade. Quanto maior o *f1 score*, melhor é o desempenho do modelo considerando os dois aspectos.

Matematicamente, o *f1 score* é representado pela Equação 4. Quando o valor beta (β) é igual a 1, o valor resultante é a verdadeira média harmônica entre precisão e *recall*. Mas ele pode receber valores diferentes de 1 para dar maior ênfase a algum dos dados. Quanto menor o valor beta (β) maior é o peso da precisão, e quanto maior o valor, maior foco no *recall*.

$$f1\,score = \frac{(1 + \beta^2) * Precisão * Recall}{\beta^2 * Precisão + Recall} \tag{4}$$

Para o cálculo dessas métricas neste trabalho, foi utilizada a biblioteca *metrics* do *sklearn*[7], que realiza a análise de uma maneira simples, recebendo como parâmetros somente o valor verdadeiro das imagens e a predição do modelo no qual está sendo avaliado.

3.4. Implementação do Modelo

Para a construção da API que implementa a rede neural, foi utilizada a linguagem de programação Python, que possui fácil acesso a bibliotecas já consolidadas para a implantação do modelo neural. Esta API foi desenvolvida utilizando o *framework Flask*[8],

[7]Documentação disponível em: https://scikit-learn.org/stable/modules/model_evaluation.html

[8]Biblioteca escrita em python para a criação de pequenas aplicações e APIs. Disponível em: https://flask.palletsprojects.com/en/2.2.x/

que possui diversas ferramentas que foram utlizadas para a construção da API, inclusive com documentação do *Swagger*[9].

A Figura 15 apresenta o código responsável por diagnosticar o CEM. Inicialmente ele carrega os dados e modelos. Na linha 1 são carregadas todas as classes possíveis para o diagnóstico, após isso, na linha 2, é realizada a chamada para o método responsável por realizar o *download* do modelo, isto é necessário devido ao tamanho dos modelos, que podem passar de 500MB. Por fim, nas linhas 3 e 4 é feito o carregamento deste modelo para a biblioteca do *PyTorch*. Vale ressaltar que estas tarefas são realizadas enquanto a API inicializa, deixando tudo pronto para poder receber as requisições.

A rede neural utilizada na API foi exportada para um arquivo no formato .pth após a realização do treinamento da rede, conforme mencionado na seção 3.2. Esse arquivo gerado após o treinamento também poderia ser importado diretamente por um aplicativo do celular. A vantagem desta abordagem é que o aplicativo funcionaria independente da internet, que pode ser precária em algumas áreas rurais, retornando um diagnóstico instantâneo mesmo nas áreas mais remotas. Porém, devido ao alto poder de processamento necessário para executar a rede neural nos aplicativos Android e IOS, foi decidido centralizá-la em uma API. Isso também facilita a atualização do modelo e a geração de análises e relatórios sobre o desempenho da rede.

```
1  imagenet_class_index = json.load(open('models/classes.json'))
2  ModelInitialize.initialize()
3  model = torch.load('models/model.pth')
4  model.eval()
5  threshold = 0.53
6
7  def transform_image(image_bytes):
8      my_transforms = transforms.Compose([
9                          transforms.RandomResizedCrop((224, 224)),
10                         transforms.Resize((224, 224)),
11                         transforms.RandomHorizontalFlip(),
12                         transforms.ToTensor(),
13                         ])
14     image = Image.open(io.BytesIO(image_bytes))
15     return my_transforms(image).unsqueeze(0)
16
17
18 def get_prediction(image_bytes):
19     tensor = transform_image(image_bytes=image_bytes)
20     output_with_threshold = (tensor > threshold).float()
21     outputs = model.forward(output_with_threshold)
22     _, y_hat = outputs.max(1)
23     predicted_idx = str(y_hat.item())
24     return imagenet_class_index[predicted_idx]
```

Figura 15. Código utilizado realizar o dignóstico de CEM na API Neural

[9]Biblioteca responsável pela documentação de APIs, dando visibilidade para seus endpoints e facilitando a realização de testes com sua interface gráfica. Disponível em: https://swagger.io/

Para realizar o diagnóstico da imagem, é acionado o método *get_prediction*, presente na linha 18, este método realiza algumas transformações e padronizações na imagem utilizando o método presente na linha 7. Após a tratativa da imagem, é aplicado o *Threshold*, que é o valor limite que define quais diagnósticos serão positivos e quais serão negativos. Este valor foi definido buscando obter a melhor precisão possível do modelo. Por fim, a imagem é passada pela rede neural e é retornado o diagnóstico com base no arquivo *json* carregado na linha 1 durante a inicialização da API.

Foi criado o endpoint *request-analysis*, mostrado na Figura 16. Este endpoint tem como objetivo receber a requisição que solicita a análise da rede neural convolucional, ele recebe as requisições utilizando o protocolo *multipart/form-data*, que é um protocolo HTTP (*Hypertext Transfer Protocol*) que permite a transferência de arquivos entre APIs, inclusive no formato de imagens.

Após a leitura da imagem e a transformação de seu conteúdo em bytes, realizadas nas linhas 13 e 14, estes dados são enviados para o método que realiza a análise da imagem, na linha 15.

```
1  @rest_api.route('/api/request-analysis',
2  doc={"description": 'Analysis Images In Convolutional Neural Network'},
3  methods=['POST'])
4  class Items(Resource):
5
6      @rest_api.expect(parser)
7      def post(self):
8          """
9              Returns if a plant is infected with Canker
10         """
11
12         if request.method == 'POST':
13             file = request.files['file']
14             img_bytes = file.read()
15             class_id, class_name = get_prediction(image_bytes=img_bytes)
16             return jsonify({'class_id': class_id, 'class_name': class_name})
17
18         return {"success": False}, 405
```

Figura 16. Código utilizado para receber requisições na API Neural

3.5. Implantação do Modelo Selecionado

Como mostra a Figura 1, a plataforma do Cancontrol é composta por uma API, que recebe as informações enviadas por meio dos aplicativos e as salva no banco de dados para que os fitopatologistas da Epagri possam emitir os diagnósticos. Esta mesma API possui as funções que notificam os produtores após a realização destes diagnósticos. Para realizar a integração da rede neural com a plataforma do Cancontrol, foram implementadas alterações no código desta API. Com essas alterações, ao receber as imagens, a API do Cancontrol, além de salvá-las no banco de dados, as envia para a API que implementa a rede neural, que realiza a análise das imagens e retorna, quando possível, o resultado que é persistido no banco como diagnóstico da doença.

A Figura 17 apresenta o trecho de código da API do Cancontrol que recebe as requisições de armazenamento de imagens do aplicativo. A linha 3 até a linha 14 mostram o processo já existente de receber a imagem e o identificador do monitoramento à qual ela pertence e de movê-la para o disco do servidor em que a API está hospedada.

```
1   async store({ request, response }) {
2     const trx = await Database.beginTransaction();
3     const { monitoring_id } = request.all();
4
5     const image = request.file("image", {
6       types: ["image"],
7       size: "10mb",
8     });
9
10    const name = `${Date.now()}.${image.extname}`
11    ...
12    await image.move(process.env.DIR_IMAGE, {
13      name: name,
14    });
15
16    await ImageService
17      .submitNeuralEvaluation(trx, monitoring_id, name);
18
19    return response.status(201).send(imageData);
20  }
```

Figura 17. Código de armazenamento de imagens da API Cancontrol

Nas linhas 16 e 17 é realizado o processo de envio para análise da rede neural, nessa parte o identificador do monitoramento e a imagem são passados para a classe responsável por enviar essas informações para a rede neural e tratar o resultado obtido. Vale ressaltar que foi realizada a implementação de um método com tratativas de erros, pois caso ocorra algum imprevisto que gere um erro na API da rede neural, este erro não trava o fluxo e mantém o processo já existente de enviar para análise dos fitopatologistas.

A Figura 18 apresenta o método que envia a imagem através de uma requisição HTTP para a API da rede neural. Da linha 3 até a 12, os dados da imagem são colocados em um *FormData*, que é um protocolo de comunicação utilizado para enviar arquivos através da internet. Na linha 15 foi utilizado o pacote *axios*, que é uma biblioteca responsável por realizar requisições HTTP de maneira simplificada, para enviar a imagem para a API da rede neural, onde é feita a avaliação.

As linhas 18 a 33 processam o retorno da rede neural, quando um resultado é obtido. O resultado é enviado para ser salvo no banco, junto com o identificador do técnico que fez a análise, neste caso a rede. Caso o resultado da rede neural seja zero, a API do Cancontrol salva no banco de dados o resultado *negativo*, caso seja um, ela salva *positivo* e caso a rede neural não tenha conseguido chegar a um resultado (neste caso ela retorna o valor 2), a API não realiza persistência da avaliação no banco e o fluxo já

existente é mantido, deixando como responsáveis pelo diagnóstico os fitopatologistas da Epagri.

```
1  var ImageService = {
2      submitNeuralEvaluation:
3          async function (trx, monitoring_id, image_name) {
4              const imageFormData = new FormData();
5              const urlNeuralApi = process.env.URL_NEURAL_API;
6              const buffer = fs.readFileSync(
7                  rocess.env.DIR_IMAGE + '\\' + image_name
8              );
9
10             imageFormData.append('file', buffer, {
11                 name: 'file',
12                 filename: image_name
13             });
14
15             await axios.post(
16                     urlNeuralApi + '/request-analysis',
17                     imageFormData
18                 ).then(async function (response) {
19                     var result = response.data.class_id;
20
21                     if(result != 2){
22                         let isCancro = result == 1;
23                         var technical = await TechnicalService
24                             .createOrRetrieveNeuralTechnical(trx);
25                         await MonitoringService
26                             .updateAndNotify(trx,
27                                 monitoring_id,
28                                 technical.id,
29                                 isCancro ? "Positivo" : "Negativo"
30                             );
31                         return true;
32                     }
33                     else
34                         return false;
35                 }
36             )
37         }
38 }
```

Figura 18. Código da API Cancontrol que realiza requisições para a API Neural

Quando a rede neural consegue emitir um diagnóstico, para a persistência das informações da análise no banco foi utilizado o código mostrado na Figura 19. Na linha 4, o monitoramento é obtido do banco, nas linhas 7 a 12 o monitoramento é editado com o

estado novo que a rede neural previu e com o identificador do técnico que fez a análise, no caso a rede neural. Após todo esse processo é realizado o *commit* da transação no banco.

```
1  var MonitoringService = {
2      updateAndNotify:
3          async function (trx, id, technical_id, state) {
4              const monitoring = await Monitoring
5                  .findOrFail(id);
6
7              monitoring.merge({
8                  technical_id,
9                  state,
10             });
11             await monitoring.save(trx);
12             await trx.commit();
13
14             const { token } = await Farmer
15                 .findOrFail(monitoring.farmer_id);
16
17             await Firebase.sendNotification(
18                 token,
19                 "Resultado da analise",
20                 monitoring
21             ).then(() => {
22                 return monitoring;
23             }).catch((error) => {
24                 throw error;
25             });
26
27         }
28 }
```

Figura 19. Código de persistência e notificação do resultado neural

Os comandos da linha 14 até a 25 enviam uma notificação para o produtor que solicitou a análise com o resultado, tanto para casos em que a rede neural realizou o diagnóstico, quanto para casos em que os fitopatologistas foram responsáveis pelo diagnóstico. A única diferença vai ser a velocidade da análise e o técnico responsável. Essa notificação foi implementada utilizando o *Firebase Cloud Messaging*[10], que é uma plataforma da Google que oferece o envio de mensagens para dispositivos móveis de maneira segmentada e personalizada.

4. Resultados

Cada modelo citado na seção 3.2 (*Alexnet, Densenet, Mobilenet V2, Mobilenet V3 Large, Mobilenet V3 Small, Resnet, Squeezenet, VGG, VGG 16*) foi treinado utilizando um con-

[10]Mais informações disponíveis em: https://firebase.google.com/products/cloud-messaging

junto de imagens que foram disponibilizadas pela EPAGRI e retirados do aplicativo do Cancontrol. Este conjunto foi separado em duas pastas para facilitar a identificação de quais imagens possuíam o CEM e quais não (62 possuíam e 73 não possuíam). Após a separação do conjunto, foi realizada outra separação entre as imagens usadas para o treinamento e para o e teste, seguindo a proporção de 80% para treino e 20% para teste. Após o treinamento, os modelos foram testados e os resultados tabulados gerando a Tabela 2, que contém os valores da matriz de confusão, limiar *(threshold)*, precisão, *recall*, e *F-Score* para cada modelo. Esta tabela apresenta os resultados dos modelos com o valor do *threshold* padrão de 0.5. Como pode-se ver, os melhores modelos são o *VGG* e o *Resnet*, que obtiveram destaque tanto no valor da precisão como no do *recall* gerando, por consequência, o melhor valor de *F-Score*.

Tabela 2. Avaliação com Threshold 0.5

Modelo	Threshold	TN	FP	FN	TP	Precisão	Recall	F-Score
Alexnet	0.5	5	10	4	9	0.474	0.692	0.562
Densenet	0.5	13	2	10	3	0.600	0.231	0.333
Mobilenet V2	0.5	10	5	5	8	0.615	0.615	0.615
Mobilenet V3 Large	0.5	10	5	7	6	0.545	0.462	0.500
Mobilenet V3 Small	0.5	10	5	7	6	0.545	0.462	0.500
Resnet	0.5	12	3	5	8	0.727	0.615	0.667
Squeezenet	0.5	12	3	9	4	0.571	0.308	0.400
VGG	0.5	12	3	5	8	0.727	0.615	0.667
VGG 16	0.5	10	5	6	7	0.583	0.538	0.560

Embora os dados da Tabela 2 já mostrem quais modelos geraram melhores resultados, ainda não é possível definir o modelo a ser utilizado, pois suas métricas de precisão, *recall*, e consequentemente, o *F-Score* acabaram obtendo valores abaixo do esperado.

Para melhorar a qualidade da rede neural, foram estudadas alternativas que possibilitassem melhorar a precisão, sem comprometer o *recall*. Para conseguir cumprir esta tarefa, diferentes *thresholds* foram utilizados para todos os modelos com o intuito de verificar qual deles gerava os melhores resultados com os parâmetros utilizados. Conforme observado na Tabela 3, o *VGG* com *threshold* de 0.530 obteve uma precisão mais alta e conseguiu manter um *recall* bom. Os demais valores levaram a resultados não tão bons quanto o esperado, pois, em muitos casos que a precisão foi alta, o *recall* acabou sendo baixo e vice-versa.

Com base na Tabela 3 foi criado um gráfico que mostra a relação entre *F-Score* e o *threshold*, mostrado na Figura 20. Nele pode-se observar o pico na medida do *F-Score*, simbolizado por um ponto que representa o *threshold* utilizado na rede neural deste trabalho.

Os valores de precisão de 0.909 e *recall* de 0.769, mostrados na Tabela 3, indicam que do total de imagens analisadas, cerca de 91% da imagens que a rede diagnosticou como positivo realmente eram positivos e, que de todas as imagens infectadas por CEM, a rede conseguiu detectar que 77% estavam contaminadas.

Tabela 3. Valores do VGG

Modelo	Threshold	TN	FP	FN	TP	Precisão	Recall	F-Score
VGG	0.128	4	11	0	13	0.542	1.000	0.703
VGG	0.136	4	11	1	12	0.522	0.923	0.667
VGG	0.165	5	10	1	12	0.545	0.923	0.686
VGG	0.182	5	10	2	11	0.524	0.846	0.647
VGG	0.191	6	9	2	11	0.550	0.846	0.667
VGG	0.262	7	8	2	11	0.579	0.846	0.688
VGG	0.302	7	8	3	10	0.556	0.769	0.645
VGG	0.365	8	7	3	10	0.588	0.769	0.667
VGG	0.374	9	6	3	10	0.625	0.769	0.690
VGG	0.416	10	5	3	10	0.667	0.769	0.714
VGG	0.446	11	4	3	10	0.714	0.769	0.741
VGG	0.472	12	3	3	10	0.769	0.769	0.769
VGG	0.505	13	2	3	10	0.833	0.769	0.800
VGG	0.530	14	1	3	10	0.909	0.769	0.833
VGG	0.559	14	1	4	9	0.900	0.692	0.783
VGG	0.581	14	1	5	8	0.889	0.615	0.727
VGG	0.642	14	1	6	7	0.875	0.538	0.667
VGG	0.664	15	0	6	7	1.000	0.538	0.700
VGG	0.703	15	0	7	6	1.000	0.462	0.632
VGG	0.743	15	0	8	5	1.000	0.385	0.556
VGG	0.911	15	0	9	4	1.000	0.308	0.471
VGG	0.940	15	0	10	3	1.000	0.231	0.375
VGG	0.971	15	0	11	2	1.000	0.154	0.267
VGG	0.974	15	0	12	1	1.000	0.077	0.143

Este processo de geração das tabelas com os diferentes *thresholds* foi realizado para todos os modelos e a Tabela 4 mostra o resultado para cada modelo com o *threshold* que levou ao melhor índice *F-Score*. Nesta tabela, nota-se que o *VGG* foi o melhor modelo novamente.

Tabela 4. Avaliação F-Score com beta 1

Modelo	Threshold	TN	FP	FN	TP	Precisão	Recall	F-Score
Alexnet	0.968	14	1	5	8	0.889	0.615	0.727
Densenet	0.143	6	9	2	11	0.550	0.846	0.667
Mobilenet V2	0.360	11	4	3	10	0.714	0.769	0.741
Mobilenet V3 Large	0.430	7	8	2	11	0.579	0.846	0.688
Mobilenet V3 Small	0.202	3	12	0	13	0.520	1.000	0.684
Resnet	0.161	5	10	1	12	0.545	0.923	0.686
Squeezenet	0.115	5	10	1	12	0.545	0.923	0.686
VGG	0.530	14	1	3	10	0.909	0.769	0.833
VGG 16	0.066	6	9	0	13	0.591	1.000	0.743

Gráficos de precisão e *recall* também foram gerados para facilitar a análise dos

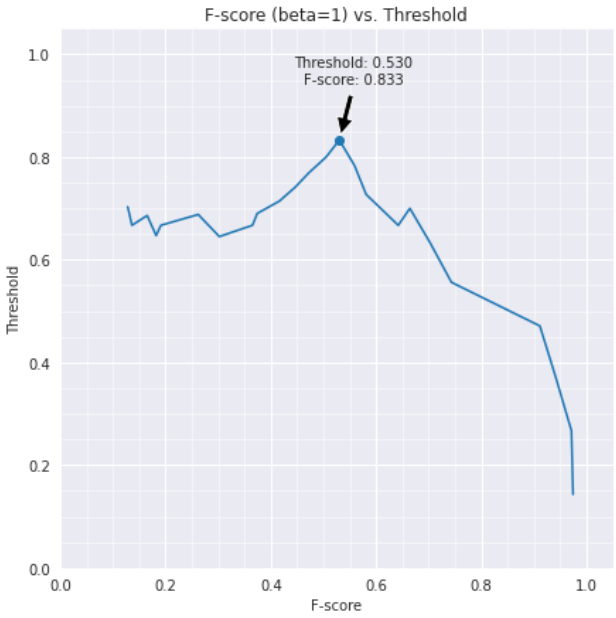

Figura 20. Relação do *Threshold* e *F-Score* do *VGG* com *beta* = 1

thresholds com base na identificação do ponto que mantém uma boa precisão com *recall* aceitável.

Na Figura 21, que apresenta o gráfico de *Precision X Recall* do modelo *VGG*, pode-se observar que a precisão consegue se manter em 1.0 até que o *recall* seja de pouco mais de 0.5 do conjunto de dados utilizados no teste. Naturalmente, quanto mais aumenta a quantidade de casos positivos detectados, menor é a precisão do modelo.

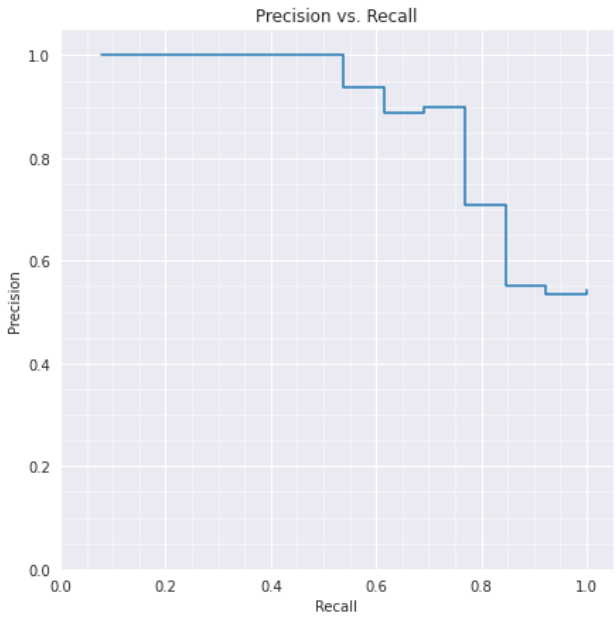

Figura 21. Relação da precisão *(Precision)* e *Recall* do *VGG*.

O objetivo dessa análise é encontrar esse *threshold* ideal em que o modelo consiga

ter uma avaliação precisa sem errar muitos diagnósticos, mas que ao mesmo tempo consiga analisar o maior número de imagens possíveis, ignorando somente aquelas imagens que ele não possua certeza se tem CEM ou não.

Ao considerar o *F-Score* para determinar o melhor modelo, leva-se em consideração a sua precisão e recuperação de forma equânime. Entretanto, é necessário avaliar o tipo de erro mais relevante para o problema em questão.

Para o objetivo deste trabalho, a precisão tem um peso maior, pois o importante é minimizar a quantidade de falsos negativos (*FN*), pois, neste caso, as árvores contaminadas ficariam infectando outras nos pomares, causando cada vez mais prejuízo e disseminando a doença. Caso o resultado seja positivo, como a precisão do modelo é alta, o resultado é salvo e o produtor é notificado. Nesse caso, o pior que pode acontecer é uma árvore não contaminada ser erradicada, mas isto não leva a grandes prejuízos como deixar uma planta contaminada disseminando a doença. Considerando que a precisão é mais relevante que o *recall* neste trabalho, foi feita a mesma análise calculando o *F-Score* com um *beta* de 0.5, o que atribui maior peso à precisão do que ao *recall*. Pode-se observar, conforme mostra a Tabela 5, que o *VGG* obteve novamente o maior *F-Score*.

Tanto para um *beta* igual a 1 e *beta* igual a 0.5, o *VGG* obteve os mesmos resultados, porém, os outros modelos mudaram, com destaque para o *Resnet*, que conseguiu uma precisão de 1.000, porém com um *recall* de 0.308. Ou seja, 100% do que o modelo avaliar como CEM, realmente é, fazendo com que o modelo não gere casos falso positivos, porém este modelo só consegue identificar 30% dos casos positivos. Embora o falso negativo não seja um problema no contexto deste trabalho, já que os especialistas vão analisar posteriormente estes casos, este *recall* acaba sendo muito inferior ao 0.769 do *VGG*, tornando interessante aceitar a geração de alguns casos de falsos positivos para poder mais que dobrar a quantidade de verdadeiros positivos. O *recall* de 0.769 do *VGG* demonstra que 77% das imagens positivas foram identificadas, o que significa que 23% das imagens que eram de cancro o sistema identificou como não sendo, mas para estes casos quem fará a avaliação final serão os técnicos para que esse erro seja corrigido.

Tabela 5. Avaliação F-Score com beta 0.5

Modelo	Threshold	TN	FP	FN	TP	Precisão	Recall	F-Score
Alexnet	0.968	14	1	5	8	0.889	0.615	0.816
Densenet	0.143	6	9	2	11	0.550	0.846	0.591
Mobilenet V2	0.537	14	1	6	7	0.875	0.538	0.778
Mobilenet V3 Large	0.430	7	8	2	11	0.579	0.846	0.618
Mobilenet V3 Small	0.570	11	4	5	8	0.667	0.615	0.656
Resnet	0.678	15	0	9	4	1.000	0.308	0.690
Squeezenet	0.513	14	1	6	7	0.875	0.538	0.778
VGG	0.530	14	1	3	10	0.909	0.769	0.877
VGG 16	0.738	14	1	6	7	0.875	0.538	0.778

5. Conclusão e Trabalhos Futuros

O Cancro Europeu da Macieira atinge os pomares de maçã da região sul do Brasil, possui uma alta taxa de contaminação e causa grandes prejuízos para a produção de maçãs. O

diagnóstico rápido ajuda no combate e prevenção da doença antes que ela se espalhe por diferentes pomares, porém a realização deste diagnóstico pode demorar, devido à necessidade de levar uma amostra da árvore contaminada para análise de fitopatologistas, o que acaba desanimando muitos produtores a combater a doença.

O Cancontrol tem o intuito de auxiliar o diagnóstico do CEM proporcionando informações da doença e do tratamento e, também, facilitando o processo de avaliação das plantas. O aplicativo permite que os produtores enviem fotos para que sejam avaliadas pelos fitopatologistas que fazem o diagnóstico. Este trabalho propôs a criação de uma rede neural convolucional para a realização do diagnóstico desta fitopatologia. A rede neural, ao ser utilizada, agiliza o diagnóstico e permite que os produtores tomem as ações devidas rapidamente, sem ter que esperar por uma análise humana.

Primeiramente foi realizada a preparação das imagens e o treinamento de diferentes modelos de redes neurais. Nesta etapa alguns desafios surgiram, o principal deles foi a quantidade limitada de imagens para o treinamento e teste das redes. Inicialmente havia apenas 58 imagens disponibilizadas pelos fitopatologistas da Epagri, que, por ser um número pequeno, influencia fortemente na precisão do modelo. Por isto, depois de alguns meses, foram obtidas mais 77 imagens da plataforma do Cancontrol, o que ajudou a deixar o modelo mais preciso.

Após a realização do treino, o objetivo foi selecionar o modelo com melhores resultados para a identificação do CEM. Nesta etapa, os maiores desafios foram a identificação do *threshold* ideal para garantir uma precisão boa sem prejudicar o *recall* e a seleção de fato do melhor modelo, pois mais de um modelo se destacou, dependendo da métrica analisada. Para dar uma ênfase maior à precisão, a métrica utilizada para escolher o modelo foi o *F-Score* com o *beta* igual a 0.5. Nesta métrica, o melhor modelo foi o *VGG* que, com um *threshold* de 0.530, obteve a precisão de 0.909, *recall* de 0.538 e *F-Score* de 0.877.

Por fim, foi feita a integração desse modelo com o Cancontrol, tendo como principal desafio a escolha de como a rede seria integrada à plataforma. Inicialmente foram levantadas diversas possibilidades, como a alteração do aplicativo para enviar os dados diretamente para a API da rede neural e depois, dependendo da resposta, enviá-la para a API do Cancontrol ou até a possibilidade de colocar o modelo neural direto no aplicativo utilizado pelo produtor. Mas no final decidiu-se que a melhor maneira de realizar esta implementação é nas APIs, principalmente pelo tamanho do modelo selecionado (*VGG*), e com uma comunicação direta entre elas, sem a necessidade de alteração do aplicativo.

Embora o objetivo deste trabalho tenha sido concluído com sucesso, ficam algumas atividades que podem ser desenvolvidas futuramente. Uma delas é testar os modelos com um conjunto de imagens maior. Outro ponto que pode levar a resultados melhores, é a avaliação de diferentes modelos juntos, criando *ensembles*. Também fica como trabalho futuro a possibilidade de integrar o modelo direto no aplicativo do Cancontrol, resultando em diagnósticos mais rápidos e sem a necessidade de conexão com a internet.

Referências

Abade, A., Ferreira, P. A., e de Barros Vidal, F. (2021). Plant diseases recognition on images using convolutional neural networks: A systematic review. *Computers and Electronics in Agriculture*, 185:106125.

Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., e Kiru, M. U. (2019). Comprehensive review of artificial neural network applications to pattern recognition. *IEEE Access*, 7:158820–158846.

Agarwal, M., Singh, A., Arjaria, S., Sinha, A., e Gupta, S. (2020). Toled: Tomato leaf disease detection using convolution neural network. *Procedia Computer Science*, 167:293–301. International Conference on Computational Intelligence and Data Science.

Alves, S. A. M. e Czermainski, A. B. d. C. (2015). Controle do cancro europeu das pomáceas com base no novo ciclo neonectria ditissima – macieira, nas condições do brasil. In *Comunicado Técnico 178*. Embrapa.

Amaral, F. (2011). *Aprenda Mineração de Dados*. Morgan Kaufmann Publishers, 1st edition.

Araujo, L., Pinto, F. A. M. F., e Vieria, J. d. S. (2019). Situação do cancro europeu no brasil. In Alves, S. A. M. e Czermainski, A. B. d. C., editors, *O cancro europeu no Brasil*. Embrapa, 1th edition.

Becker, W. E. (2017). Uma abordagem de redes neurais convolucionais para analise de sentimento multi-lingual. Orientador: Rodrigo Coelho Barros. Tese (Doutorado em Ciência da Computação) - Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul. Porto Alegre, p. 87. 2017.

Branco Neto, W. C., Araújo, L., Pinto, F., Machado, R., Ribeiro, Y., Junior, W. C., e Mattos, K. (2021). Cancontrol: plataforma para diagnóstico do cancro europeu da macieira. In *Anais do XIII Congresso Brasileiro de Agroinformática*, pages 44–52, Porto Alegre, RS, Brasil. SBC.

Canto, V. H. B. (2019). Uso de redes neurais convolucionais para identificação biométrica de indivíduos em tempo real utilizando internet das coisas. Orientador: Aparecido Nilceu Marana. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Faculdade de Ciências, Universidade Estadual Paulista Júlio de Mesquita Filho. Bauru, p. 57. 2019.

CEPA, Centro de Socioeconomia e Planajamento Agrícola (2016-2017). Síntese anual da agricultura de santa catarina. page 203. Epagri, Florianópolis, SC, Brasil. Disponível em: http://docweb.epagri.sc.gov.br/website_cepa/publicacoes/Sintese-Anual-da-Agricultura-SC_2016_17.pdf. Acesso em 03 nov. 2022.

Churchland, P. e Sejnowski, T. (1992). *The Computational Brain*. MIT Press, 1st edition.

Data Science Academy (2022). Deep learning book. Disponível em: https://www.deeplearningbook.com.br/camadas-de-pooling-em-redes-neurais-convolucionais/. Acesso em 09 jun. 2022.

Elhassouny, A. e Smarandache, F. (2019). Smart mobile application to recognize tomato leaf diseases using convolutional neural networks. In *2019 International Conference of Computer Science and Renewable Energies (ICCSRE)*, pages 1–4.

Ferrari, D. G. e Silva, L. N. d. C. (2016). *Introdução à Mineração de Dados: Conceitos Básicos, Algoritmos e Aplicações*. Editora Saraiva.

Han, J., Kamber, M., e Pei, J. (2011). *Data Mining: Concepts and Techniques*. Editora Alta Books, 3rd edition.

Haykin, S. (2001). *Redes neurais princípios e prática*. Grupo A, 2nd edition.

Kist, B. B. (2019). Fatias da fruta. In Beling, R. R., editor, *Anuário Brasileiro da Maçã 2019*. Editora Gazeta Santa Cruz. Disponível em:

https://issuu.com/abpmabpm.org.br/docs/anuariobrasileirodamaca2019. Acesso em 15 abr. 2022.

Lazzarotto, J. J. e Alves, S. A. M. (2015). Prejuízos econômicos e financeiros associados ao cancro europeu em sistemas de produção de maçã de vacaria, RS. In *Comunicado Técnico 169*. Embrapa.

LeCun, Y., Bottou, L., Bengio, Y., e Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Liu, B., Zhang, Y., He, D., e Li, Y. (2018). Identification of apple leaf diseases based on deep convolutional neural networks. *Symmetry*, 10(1).

McCracken, A. R. (2003). *Relative significance of nursery infections and orchard inoculum in the development and spread of apple canker (Nectria galligena)*.

Rodrigues, D. A. e Batista, L. V. (2018). Deep learning e redes neurais convolucionais: Reconhecimento automático de caracteres em placas de licenciamento automotivo. Universidade Federal da Paraíba.

Russell, S. e Norvig, P. (2013). *Inteligência Artificial*. Grupo GEN, 3th edition.

Toda, Y. e Okura, F. (2019). How convolutional neural networks diagnose plant disease. *Plant Phenomics*, 2019:14.

Weber, R. W. S. (2014). Biology and control of the apple canker fungus neonectria. In *Erwerbs-Obstbau*. Springer-Verlag Berlin Heidelberg, 15th edition.

Werner, A. L. (2019). Manejo do cancro europeu - a visão do produtor. In *Encontro Nacional sobre Fruticultura de Clima Temperado, XVI, Anais palestras...Caçador*, pages 1:48–51. Epagri, Fraiburgo, SC, Brasil.

Yu, F. (2016). A comprehensive guide to fine-tuning deep learning models in keras (part i). Disponível em: https://flyyufelix.github.io/2016/10/03/fine-tuning-in-keras-part1.html. Acesso em 20 jun. 2022.