UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS SOBRAL

CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

JOSÉ LOPES DE SOUZA FILHO

DESENVOLVIMENTO DE API PARA APRENDIZADO PROFUNDO DE IMAGENS DE PEQUENOS RUMINANTES

SOBRAL

2022

JOSÉ LOPES DE SOUZA FILHO

DESENVOLVIMENTO DE API PARA APRENDIZADO PROFUNDO DE IMAGENS DE PEQUENOS RUMINANTES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Ialis Cavalcante de Paula Júnior

SOBRAL

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S238d Souza Filho, José Lopes de.
Desenvolvimento de API para aprendizado profundo de imagens de pequenos ruminantes / José Lopes de Souza Filho. – 2022.
52 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2022.
Orientação: Prof. Dr. Ialis Cavalcante de Paula Júnior.

1. FAMACHA. 2. Ovinos. 3. Redes Neurais Convolucionais. 4. Implantação. 5. Aprendizagem de Máquina. I. Título.

CDD 621.39

JOSÉ LOPES DE SOUZA FILHO

DESENVOLVIMENTO DE API PARA APRENDIZADO PROFUNDO DE IMAGENS DE PEQUENOS RUMINANTES

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação do Campus Sobral da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: 29 de Novembro de 2022

BANCA EXAMINADORA

Prof. Dr. Ialis Cavalcante de Paula Júnior (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Me. Fernando Rodrigues de Almeida Júnior
Universidade Federal do Ceará (UFC)

Prof. Dr. Jarbas Joaci de Mesquita Sá Junior
Universidade Federal do Ceará (UFC)

À minha mãe.

AGRADECIMENTOS

Aos meus falecidos pais, que sempre me incentivaram a estudar. Ao meu orientador, Prof. Dr. Iális Cavalcante, pelo apoio e ensinamentos. À banca examinadora pelo tempo disponibilizado para participar da apresentação e avaliação deste Trabalho de Conclusão de Curso. Aos professores do curso de Engenharia de Computação da Universidade Federal do Ceará, campus Sobral, que me mostraram tantas coisas novas e me fizeram ver um mundo totalmente novo. Aos amigos que fiz nessa jornada, que o tempo todo serviram de inspiração e apoio. À minha esposa e filha, sem vocês nada disso teria sido possível.

Muito Obrigado.

“Machine intelligence is the last invention that humanity will ever need to make.”

(Nick Bostrom)

RESUMO

A criação de ovinos e caprinos sempre foi de fundamental importância para a região nordeste do Brasil, sendo parte fundamental da economia local. Tais rebanhos, quando criados em confinamento, podem apresentar diversas enfermidades e parasitoses, tais como o *Haemonchus contortus*, que causa anemia e perdas substanciais aos criadores. Existem métodos para controle de anemia em ovinos e caprinos, sendo um deles o método FAMACHA, cujo objetivo é identificar clinicamente animais resistentes, resilientes e sensíveis às infecções parasitárias, otimizando o tratamento de forma seletiva em situações reais no campo, sem a necessidade de recursos laboratoriais. (MOLENTO *et al.*, 2004). O estudo contido nesse documento se propôs a criar um modelo de classificação de imagens da mucosa de animais em anêmicos e não anêmicos utilizando redes neurais convolucionais assim como realizar a implantação desse modelo no ambiente de produção por meio de frameworks Python.

Palavras-chave: FAMACHA. Ovinos. Redes Neurais Convolucionais. Implantação. Aprendizagem de Máquina

ABSTRACT

Sheep and goat farming has always been very important to Brazil's northeastern region, being a key part for the local economy. These livestocks, when raised in confinement, can present many sicknesses and parasitosis, like *Haemonchus contortus*, that may cause anemia and important losses to the local farmers. There are methods to control anemia in sheep and goat livestocks, one of them is FAMACHA© method, which aims to clinically identify resistant animals, resilient and sensitive to parasitic infections, optimizing the treatment selectively in real situations at the countryside, without the need of using laboratory resources. (MOLENTO *et al.*, 2004). The study of this document aims to create a machine learning model which classifies animal's mucosa as anemic or non-anemic using convolutional neural networks as well as deploying this model in the production environment through Python frameworks.

Keywords: FAMACHA. Sheep. Convolutional Neural Networks. Deploy. Machine Learning

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AM	Aprendizado de Máquina
AP	Aprendizagem Profunda
API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
IA	Inteligência Artificial
IDE	Integrated development environment
PIL	Python Imaging Library
REST	Representational State Transfer
RNC	Redes Neurais Convolucionais

SUMÁRIO

1 INTRODUÇÃO

A criação de ovinos sempre foi de grande importância para o Nordeste do país, especialmente para a região do sertão. Compõe parte essencial da renda de muitas famílias, sendo também uma alternativa para a produção de leite, carne e derivados. De acordo com o último Censo Agropecuário realizado, o rebanho de ovinos cresceu 10,78% de 2016 a 2020 no Brasil (IBGE, 2020).

A criação de ovinos em confinamento pode causar diversas enfermidades no rebanho sendo o *Haemonchus Contortus* o verme mais patogênico e de maior impacto na ovinocultura, sendo encontrado em 75% a 100% dos resultados de exames realizados nos rebanhos (MORTENSEN *et al.*, 2003). Este parasita causa anemia e pode levar a perdas no rebanho e consequentemente prejuízo aos criadores. Portanto, tornou-se essencial controlar essa verminose e a anemia por ela gerada.

Existem diversos métodos para controle da anemia em ovinos. Entre eles o FAMACHA© é o mais conhecido e indicado. O *Haemonchus Contortus*, ao se alimentar do sangue do animal pode causar anemia. O FAMACHA© consiste em usar um cartão para identificar os diferentes graus de anemia em que o animal pode se encontrar comparando este à tonalidade de cor da mucosa do olho do animal. De acordo com o resultado, deve-se aplicar (ou não) o vermífugo para controle da doença.

O método FAMACHA© mostrou-se eficaz em reduzir as aplicações com medicação parasitária em ovinos (MOLENTO; DANTAS, 2001), porém alguns cuidados com sua aplicação se fazem necessários; entre eles o fato de apenas pessoas treinadas deverem utilizar o método, pois uma interpretação errônea pode vir a causar mais problemas ao animal - seja deixando de vermifugar animais doentes causando sua morte, ou vermifugando de forma desnecessária animais saudáveis, prejudicando a saúde dele por excesso de drogas; acumulando assim mais prejuízos aos produtores.

Com o objetivo de tornar a leitura do método mais precisa quanto possível, assim como torná-lo acessível ao maior número possível de produtores, nasceu a ideia de utilizar técnicas de aprendizagem de máquina de forma a criar um sistema computacional especialista que efetue a leitura das imagens da mucosa dos olhos desses animais e reconheça de forma automatizada por meio de técnicas de classificação, o grau de anemia do animal e portanto, se ele deve ou não ser vermifugado.

Neste trabalho, a utilização de um banco de imagens da mucosa dos olhos dos ovinos,

coletado por Almeida (2021) em parceria com a EMBRAPA será utilizado para que, por meio de técnicas de aprendizagem profunda, criem-se aplicações web que permitam identificar animais que precisem ou não ser vermifugados.

1.1 Justificativa

A alta taxa de incidência de anemia em ovinos gera uma alta demanda de testagem nos animais, podendo esta frequência chegar a ser semanal. Desta forma, de acordo com o tamanho do rebanho, pode ser um trabalho repetitivo e altamente dispendioso de tempo e recursos para o produtor, além do fato de aumentar consideravelmente as chances de erro humano, aumentando ainda mais os prejuízos dos produtores.

Este trabalho propõe a utilização de algoritmos de aprendizagem de máquina para criar uma aplicação capaz de classificar os animais como doentes ou não doentes, de acordo com o método FAMACHA©, utilizando o banco de imagens da mucosa dos olhos dos ovinos desenvolvido por Almeida (2021) em parceria com a EMBRAPA e assim, levar o modelo criado para níveis de produção desenvolvendo uma aplicação acessível e funcional para os produtores, agilizando o trabalho dos especialistas e criadores além de minimizar as taxas de erro e reduzindo os custos dos produtores.

1.2 Trabalhos Relacionados

O trabalho de Molento *et al.* (2004) avalia o uso do método FAMACHA© como estratégia auxiliar no controle do parasita *Haemonchus contortus*. O estudo indica uma considerável redução do uso de medicamentos nos ovinos quando comparado a outros métodos aplicados a todo o rebanho.

Em Tamir *et al.* (2017) propõe-se a criação de um sistema de detecção de anemia em seres humanos como alternativa aos métodos invasivos atuais de diagnóstico atingindo um maior impacto, especialmente em regiões subdesenvolvidas do mundo.

O trabalho de Mitani *et al.* (2020) conseguiu mostrar que a anemia pode ser detectada via algoritmos treinados de aprendizagem de máquina utilizando imagens de fundo de olho.

O estudo de Souza *et al.* (2021) desenvolve uma aplicação android utilizando o kit de desenvolvimento *flutter* para aquisição de imagens e, em seguida, um classificador que se baseia no método FAMACHA© apresentando precisão superior a 83%.

Outro trabalho que se propõe a desenvolver mais uma aplicação móvel capaz de detectar anemia em ovinos através do método FAMACHA© é o de Demoliner e Alves (2017) que utilizou-se das bibliotecas OpenCV, WEKA e o algoritmo de aprendizagem *Naive Bayes* para classificar as imagens de acordo com o método.

1.3 Objetivos

1.3.1 Objetivos Gerais

Este trabalho tem como objetivo geral desenvolver um modelo preditivo que identifique os animais como doentes ou saudáveis utilizando uma rede neural convolucional, e utilize esse modelo para criar duas APIs a nível de produção a fim de contribuir com os produtores de ovinos.

1.3.2 Objetivos Específicos

- Treinar uma rede que classifique imagens da mucosa dos olhos dos ovinos;
- Utilizar o modelo criado para produção de duas API's que utilizarão essas redes para realizar a classificação das imagens;
- Avaliar os resultados obtidos;

2 FUNDAMENTAÇÃO TEÓRICA

De modo a atingir os objetivos deste trabalho, faz-se necessário um estudo teórico que embase os fundamentos apresentados ao longo deste documento para que a solução seja alcançada. Tais conceitos serão apresentados a seguir.

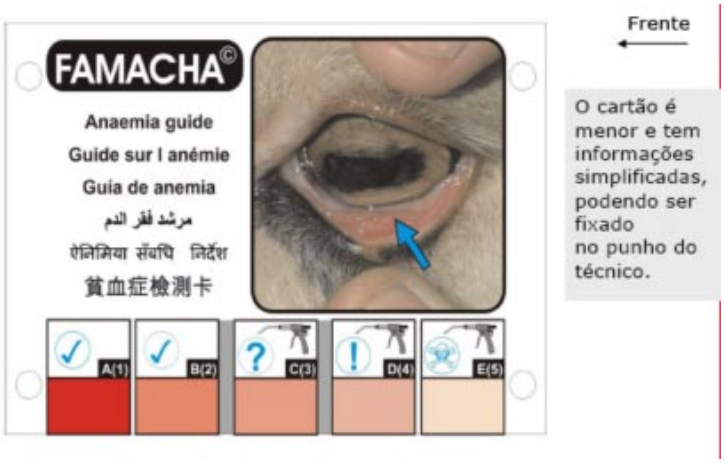
2.1 O método FAMACHA©

O método FAMACHA© é uma forma de controle do *Haemonchus contortus* - considerado o principal parasita de pequenos ruminantes em todas as regiões brasileiras (ARO-SEMENA *et al.*, 1999).

É um método de tratamento que se diferencia dos demais pois estabelece um critério para vermifugação dos animais, selecionando apenas os animais que realmente necessitam de vermifugação, reduzindo assim os custos com a produção, além de reduzir significativamente a resistência dos animais à medicação.

Após pesquisas, descobriu-se que existe relação entre a cor da mucosa ocular dos pequenos ruminantes com 5 níveis de anemia detectados por meio de exame de hematócrito - um exame popular usado para determinar a saúde do animal. Desta forma, esses 5 graus de cores são representados em um cartão.

Figura 1 – Frente do cartão FAMACHA©



Fonte: Carvalho e Molento (2015)

A Figura 1 ilustra um dos modelos de cartão utilizado pelos criadores para detectar diferentes graus de anemia nos animais.

O método é simples e consiste basicamente em comparar a cor da mucosa ocular do

animal com as diferentes colorações do cartão para que dessa forma possa-se ter um indicativo do nível de saúde do animal, determinando se esse animal precisa ou não ser medicado.

Os graus 1 e 2 são de coloração bem avermelhada o que indica que praticamente não existem traços de anemia no animal, dispensando qualquer tipo de vermifugação. A partir do grau 3, a vermifugação já é recomendada e nos graus 4 e 5 ela é obrigatória.

Tabela 1 – Relação entre o grau FAMACHA© e a coloração da conjuntiva ocular e o hematócrito, orientando ou não o tratamento

Grau FAMACHA©	Coloração	Hematócrito	Atitude clínica
1	vermelho robusto	>27	não tratar
2	vermelho rosado	23 a 27	não tratar
3	rosa	18 a 22	tratar
4	rosa pálido	13 a 17	tratar
5	branco	<13	tratar

Fonte: CHAGAS *et al.* (2007)

A Tabela 1 apresenta os diferentes graus FAMACHA© que vão de uma escala de 1 a 5, onde 1 indica o animal em seu estado mais saudável e 5 o maior grau de anemia. Cada um dos graus possui uma coloração de mucosa equivalente indo do vermelho robusto até o mais branco; um estimativa de número de hematócritos correspondentes assim como a necessidade ou não de submeter o animal à tratamento.

2.2 Inteligência Artificial

Inteligência Artificial (IA) é o termo criado para descrever máquinas capazes de reproduzir habilidades cognitivas que até então seriam inerentes dos seres humanos e algumas espécies animais. Entre essas habilidades podemos destacar a capacidade de aprendizagem.

Em Inteligência Artificial: Uma abordagem moderna, Stuart Russell and Peter Norvig definem IA como o ato de criar e desenvolver os chamados *agentes inteligentes* que são capazes de identificar o ambiente que estão inseridos e agir de modo a afetar esse ambiente (RUSSELL; NORVIG, 2009).

Atualmente podemos encontrar inúmeros exemplos de aplicação da inteligência artificial seja em sistemas de recomendação, assistentes de direção inseridos em automóveis modernos, sistemas de diagnóstico de doenças e jogadores robôs capazes de vencer grandes campeões. As aplicações são muitas e em todas as áreas de conhecimento.

Os conceitos de inteligência artificial não são recentes. Podemos tomar como

exemplo a ideia do neurônio artificial que data de meados da década de 40. Conforme as décadas foram se passando, os estudos de IA foram alternando momentos de extremo entusiasmo por parte dos pesquisadores com momentos de desânimo - especialmente por conta das muitas barreiras encontradas ao longo das décadas e da falta de tecnologias que oferecessem suporte para o desenvolvimento da IA, o que levava a uma falta de financiamento e interesse por parte de muitos pesquisadores.

Na década de 80, a IA experimentou um novo momento de alta, especialmente pela popularização e exploração comercial de sistemas especialistas - sistemas capazes de análise que simulam as habilidades de especialistas humanos.

Desde então, a IA vem recuperando sua posição de interesse por parte de empresas, governos e pesquisadores e ao longo das décadas de 90 e 2000 cresceu muito, entre outros fatores, por conta do desenvolvimento de máquinas cada vez mais poderosas em termos de processamento e sobretudo um massivo armazenamento e compartilhamento de dados por parte das pessoas, o que fez com que modelos de aprendizagem baseados em dados pudessem se tornar muito poderosos e com níveis de precisão nunca antes vistos.

2.3 Aprendizado de Máquina

Aprendizado de Máquina (AM), é um ramo da IA capaz de fazer com que computadores aprendam (melhorem o próprio algoritmo) a partir de dados e, dessa forma, sejam capazes de reagir a uma nova situação nunca antes processada por eles com base nos dados utilizados para treiná-los. É particularmente útil, pois evita que seja necessário reprogramar o sistema de forma explícita a cada nova informação que surgir. O sistema baseado em AM é capaz de realizar uma série de tarefas computacionais, onde reprogramar o sistema seria algo impraticável (para não dizer impossível).

Entre as inúmeras aplicações atuais de AM podemos citar as tecnologias de reconhecimento ótico de caracteres (OCR), reconhecimento de fala, visão computacional, reconhecimento de escrita, diagnóstico médico e muitas mais.

O AM permite que pesquisadores, engenheiros e empresas possam produzir decisões confiáveis com base em dados assim como descobrir informações implícitas dentro de conjuntos de dados, graças as observações de tendências nos dados.

Suas tarefas são geralmente classificadas em três categorias:

- Aprendizado supervisionado

- Aprendizado não supervisionado
- Aprendizado por reforço

No aprendizado supervisionado, os dados apresentados ao computador mostram também a saída desejada; daí o nome "supervisionada". É como se um professor mostrasse os dados ao aluno e fosse dizendo o que é cada coisa. Em seguida outros dados diferentes daqueles utilizados no treinamento são apresentados e o sistema deve dizer do que se trata com base no que foi ensinado a ele. Temos como exemplo de tarefas de aprendizado supervisionado a classificação e regressão.

Na aprendizagem não supervisionada nenhuma saída é apresentada. O próprio sistema deve analisar os dados e tentar encontrar padrões. É como começar um estudo sem saber ao certo o que vai estudar. E sem nenhum professor. Tarefas comuns de aprendizado não supervisionado são a clusterização (agrupamentos), detecção de anomalias (elementos que fogem ao padrão dos dados), redução de dimensionalidade (quando queremos simplificar dados perdendo o mínimo de informação) e a aprendizagem por regra de associação (quando se deseja descobrir relações interessantes entre os atributos).

Na aprendizagem por reforço o sistema computacional inteligente (chamado de agente) deve interagir com um certo ambiente, que pode ser o trânsito de uma cidade ou simplesmente um jogo qualquer. O sistema deve desempenhar um objetivo nesse ambiente. O agente inteligente é então premiado para cada interação correta e punido por cada interação errada à medida que navega no universo do ambiente ao qual ele está inserido.

2.3.1 Aprendizagem Profunda

A aprendizagem profunda (AP) é uma tentativa de modelar um sistema computacional de forma semelhante, porém bastante simplificada ao córtex cerebral humano utilizando pilhas de camadas de neurônios artificiais - chamadas de redes neurais profundas.Treinar redes neurais tão complexas até alguns anos atrás era uma tarefa praticamente impossível, devido sobretudo às limitações de poder de processamento dos computadores até meados da década de 1990.

Hinton *et al.* (2006) demonstrou em seu artigo *A Fast Learning Algorithm for Deep Belief Nets* como treinar uma rede neural profunda que reconhecia números escritos à mão conseguindo uma precisão de mais de 98%, chamando essa técnica pela primeira vez de *Aprendizagem Profunda*.

Esse artigo foi um marco e reacendeu de vez o interesse dos pesquisadores que publicaram mais e mais estudos que comprovaram que a AP era capaz de resultados surpreendentes, sobretudo quando treinadas com quantidades gigantescas de dados.

2.3.2 Visão Computacional

Visão computacional é o ramo da IA capaz de dar aos sistemas inteligentes a capacidade de "enxergar". Na prática, nada mais é do que a capacidade que o sistema tem de extrair informações diversas a partir de dados de imagem, que podem ser fotos ou vídeos.

A visão computacional pode ser considerada certamente como um dos grandes avanços recentes da IA. Só foi possível graças aos avanços na pesquisa de AM e AP.

Entre as aplicações da visão computacional, podemos citar o reconhecimento facial para segurança, reconhecimento de trajeto e obstáculos de percurso, reconhecimento de veículos, monitoramento de tráfego, identificação de produtos defeituosos na indústria, contagem de rebanho, diagnóstico médico, entre diversas outras.

2.3.3 Redes Neurais Convolucionais

As Redes Neurais Convolucionais (RNC) não são um conceito recente; seus estudos se originaram em meados dos anos 80 a partir dos estudos do córtex visual do cérebro. Seu uso tem se popularizado em função do aumento do poder de processamento recente.

Uma grande vantagem da RNC é a sua versatilidade. Elas não se limitam apenas à percepção visual, também podendo ser usadas em reconhecimento de voz e processamento de linguagem natural.

Esses estudos do córtex visual deram origem ao neocognitron (FUKUSHIMA *et al.*, 1983) que aos poucos foi se desenvolvendo, chegando ao que conhecemos hoje por RNC.

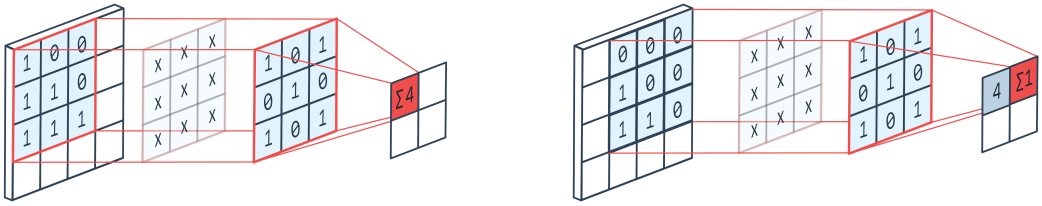
2.3.3.1 Camadas Convolucionais

Um dos conceitos mais importantes em uma RNC é o da camada convolucional. Uma convolução é uma operação matemática na qual a partir de duas funções dadas, retorna uma terceira a partir da integral da sua multiplicação pontual. As redes convolucionais usam correlações muito semelhantes às convoluções.

Os neurônios da primeira camada convolucional não estão ligados a cada pixel da

imagem de entrada, mas somente aos pixels em seus campos receptivos, ou seja, cada neurônio da camada seguinte está ligado a uma dada região da camada anterior, reduzindo assim o tamanho das imagens e preservando as características mais fortes da camada anterior. Essa característica se observa no mundo real e por conta disso as RNCs são tão bem adaptadas para reconhecimento de imagens.

Figura 2 – Camadas convolucionais



Fonte: StackExchange (2020)

A Figura 2 ilustra as conexões entre as diferentes camadas da rede convolucional e como as características mais relevantes da camada anterior se preservam na camada seguinte, tornando-as ideais para reconhecimento de imagens.

2.3.3.2 Camadas de pooling

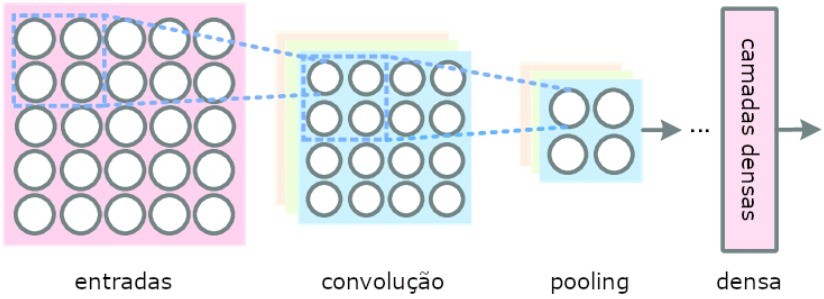
As camadas de pooling em uma RNC têm por objetivo simplesmente *subamostrar*, ou seja, encolher a imagem de entrada. Essa função possui duas utilidades: a primeira é reduzir a carga computacional e a segunda é reduzir a probabilidade de haver *overfitting* - quando a rede está excessivamente ajustada aos dados de treinamento (alto índice de acertos) porém pouco preparada para lidar com dados reais (alto índice de erros nos dados de validação).

2.3.3.3 Arquitetura das RNCs

Uma arquitetura típica de uma RNC empilha várias camadas convolucionais, ligadas a uma camada de pooling, depois outras poucas camadas convolucionais, depois outra camada de pooling e assim sucessivamente. Conforme avança na rede, a imagem vai ficando cada vez menor, porém mais carregada de características por conta das camadas convolucionais.

Todas as diferentes arquiteturas citadas seguem a estrutura padrão ilustrada na Figura 3, o que muda é tão somente o arranjo dessas camadas.

Figura 3 – Arquitetura das RNCs



Fonte: Wang *et al.* (2017)

2.3.3.4 AlexNet

A AlexNet foi desenvolvida por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton (KRIZHEVSKY *et al.*, 2012). É uma rede similar à LeNet-5 porém ela é muito maior e mais profunda e foi a primeira a empilhar várias camadas convolucionais uma em cima da outra ao invés de colocar uma camada de pooling acima de cada camada convolucional como vinha sendo feito até então.

2.3.3.5 GoogLeNet

A arquitetura GoogLeNet foi criada por Christian Szegedy et al. (SZEGEDY *et al.*, 2015). Essa arquitetura foi a vencedora do desafio de reconhecimento de imagem ILSVRC de 2014. Ela obteve um considerável redução no erro se comparada à rede AlexNet - vencedora do ano de 2012.

Essa arquitetura é muito diferente das anteriores. Ela usa muitos métodos diferentes que permitem criar uma arquitetura bem mais profunda.

2.3.3.6 LeNet-5

A arquitetura LeNet-5 é uma das arquiteturas mais conhecidas. Foi criada por Yann LeCun em 1998 e é uma das arquitetura mais usadas para reconhecimento de números escritos à mão (LECUN *et al.*, 1998).

2.3.3.7 ResNet

As redes ResNet (Residual Network) (HE *et al.*, 2015) venceram o desafio ILSVRC de 2015 utilizando redes extremamente profundas constituídas por 152 camadas (existem variações com 34, 50 e 101) confirmando a tendência de que os modelos foram ficando cada vez mais profundos, demandando cada vez mais recursos computacionais, com cada vez menos parâmetros.

2.3.3.8 SENet

Essa arquitetura (Squeeze and Excitation Network) (HU *et al.*, 2017) amplia as arquiteturas recentes, com as redes *inception* e as ResNets, aumentando seu desempenho e reduzindo a taxa de erro para supreendentes 2,25% na ILSVRC de 2017, aumentando a precisão em 25% em relação a performance da vencedora do ano anterior.

2.3.3.9 VGGNet

A VGGNet (SIMONYAN; ZISSERMAN, 2014) desenvolvida por Karen Simonyan e Andrew Zisserman do Laboratório de Pesquisa Visual Geometry Group (VGG) possuía uma arquitetura muito simples e clássica, com duas ou três camadas convolucionais e uma de pooling mais uma rede densa final com duas camadas ocultas e mais uma camada de saída.

2.3.3.10 Xception

A rede Xception (CHOLLET, 2016) é uma variante da rede GoogLeNet. Foi proposta no ano de 2016 por François Chollet (criador da biblioteca Keras). Combina as ideias da GoogLeNet e da ResNet.

2.3.3.11 ZFNet

Matthew Zeiler e Rob Fergue em 2013 criaram uma arquitetura baseada na AlexNet que foi a vencedora da ILSVRC no mesmo ano. O nome ZFNet foi dado em homenagem aos seus criadores. Esta rede utiliza kernels de tamanho 7x7 ao invés de 11x11, diminuindo siginificativamente o número de pesos e aumentando sua acurácia.

2.3.4 Ambientes de pesquisa e produção

A implementação de um modelo de AM é o processo de integrar tal modelo em diferentes ambientes. O termo *ambiente* se refere ao estado ou atributo de um computador no qual seus softwares ou outros produtos são desenvolvidos ou colocados em operação.

Tais modelos são desenvolvidos em um ambiente chamado de ambiente de pesquisa - que nada mais é do que um conjunto de softwares e ferramentas adequados para a análise dos dados assim como a criação do modelo de AM que seja capaz de apresentar os mesmos resultados em qualquer ambiente que o execute. No ambiente de pesquisa também analisa-se o potencial de uso do modelo em uma situação real por meio de métricas como precisão, perda, matriz de confusão entre outras.

O ambiente de produção é um estado onde software e hardware rodam em tempo real e integram as operações diárias da organização. É neste ambiente onde o modelo desenvolvido no ambiente de pesquisa fica disponibilizado para uso real para seus usuários permitindo a eles usarem o modelo como um serviço.

Portanto, as técnicas, ferramentas e softwares utilizadas nos ambientes de pesquisa e produção não são as mesmas, pois os dois ambientes possuem diferentes propósitos.

2.3.5 Micro-Framework

Micro-Frameworks são versões reduzidas de um framework padrão, somente com o básico para funcionar. São normalmente utilizados para implementação de APIs REST - que são usadas para estabelecer comunicação entre aplicações para consumo de informações de forma rápida e segura.

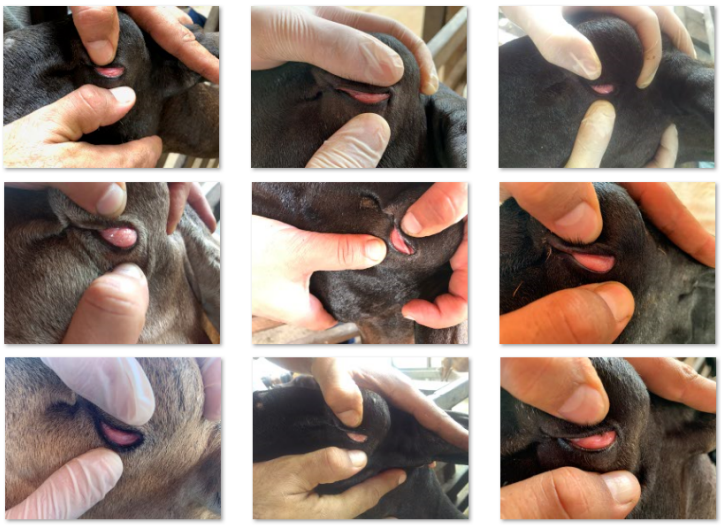
3 METODOLOGIA

As próximas seções deste documento descrevem as etapas de desenvolvimento do projeto. Se dividem em duas etapas: o ambiente de pesquisa - que compreende a parte de análise exploratória do banco de imagens assim como a criação do modelo; e o ambiente de produção - que utiliza os dados de pesquisa de modo a gerar o código que será usado na criação da aplicação por parte dos usuários.

3.1 Base de Imagens

A base de imagens utilizada para este trabalho foi gentilmente cedida por Almeida (2021) em parceria com a EMBRAPA e Universidade Federal do Ceará - Campus Sobral. A base contém um total de 128 amostras de imagens, das quais 64 de animais com necessidade de vermifugação e 64 de animais que não necessitam de vermifugação.

Figura 4 – Amostras da base fornecida



Fonte: Almeida (2021)

A Figura 4 ilustra uma pequena amostra do banco de imagens utilizado neste trabalho. São imagens de mucosas oculares de ovinos saudáveis e doentes.

A classificação dos animais como doentes ou saudáveis foi inicialmente feita por meio de exame de hematócritos que determinou o grau de anemia desses animais. Para fins de simplificação, animais cujos resultados de exames de hematócritos ficaram abaixo de 24% foram classificados como doentes e iguais ou acima de 24% foram classificados como saudáveis. Este dado pode ser encontrado na nomenclatura de cada um dos arquivos da base, conforme indicado

no exemplo abaixo.

Exemplo: 16470_MN_**31**_19-09-2018_Android_082949161.JPG

O termo 31 (em negrito) indica a porcentagem (%) do resultado do exame de hematócritos.

3.2 Linguagem e Bibliotecas

Para a realização do projeto descrito neste trabalho, foi utilizada como base a linguagem de programação *Python*[1] (Versão 3.6.10), por conta de seu alto grau de adaptação para tarefas de AM e pelo volume de bibliotecas e frameworks disponíveis.

As principais bibliotecas e frameworks utilizados estão descritos abaixo.

3.2.1 NumPy - Numerical Python

O NumPy[2] (Versão 1.19.5) criado em 2005 por Trevis Oliphan, é uma biblioteca de código aberto Python para realizar operações em arrays multidimensionais (nesta biblioteca, chamadas de ndarray).

Na área de análise de dados é uma biblioteca imprescindível, pois possui funcionalidades para tratamento, limpeza e filtragem, descrição e manipulação de dados assim como outros tipos de processamento.

O NumPy também possui funções matemáticas que permitem operações rápidas em arrays evitando assim ter que escrever laços, recursos de álgebra linear, geração de números aleatórios, e transformadas de Fourier, por exemplo.

A biblioteca NumPy não disponibiliza funcionalidades científicas nem de modelagem, porém é uma biblioteca imprescindível para o uso de bibliotecas baseadas em arrays, como o Pandas.

3.2.2 Pandas

A biblioteca *Pandas*[3] (Versão 1.0.5), criado por Wes McKinney em 2008, utiliza séries e DataFrames para a manipulação de dados tabulares como em SQL, ordenados, matrizes ou qualquer outro tipo de dados, não precisando sequer estarem rotulados, com poucos comandos.

[1] Disponível em: <https://www.python.org/>. Acesso em: 24 mai. 2022.
[2] Disponível em: <https://numpy.org/>. Acesso em: 24 mai. 2022.
[3] Disponível em: <https://pandas.pydata.org/>. Acesso em: 24 mai. 2022.

3.2.3 Matplotlib

A biblioteca *Matplotlib*[4] (Versão 3.2.2) foi utilizada para visualizar as imagens assim como plotar os gráficos com o desempenho do modelo de AM gerado.

3.2.4 Scikit-learn

O *Scikit-learn*[5] (Versão 0.23.1) é uma biblioteca Python desenvolvida especificamente para a aplicação de AM. Foi utilizada neste trabalho para separar os dados de treino, teste e validação, assim como gerar métricas para análise de desempenho do modelo.

3.2.5 TensorFlow

O *TensorFlow*[6] (Versão 2.6.2) é uma das principais bibliotecas utilizadas para construir modelos de AM e AP. É um sistema focado em treinamento de redes neurais usado para o aprendizado de padrões e correlações. Foi desenvolvido pela Google em 2015. Neste trabalho, foi utilizada para modelagem e treinamento da RNC.

3.2.6 Keras

O *Keras*[7] (Versão 2.6.0) é uma biblioteca de manipulação de redes neurais escrita em Python. A principal diferença desta biblioteca, em relação a outras bibliotecas que se propõem a fazer a mesma coisa, é que esta funciona junto ao próprio TensorFlow, o que faz com que a biblioteca Keras seja entendida com uma espécie de interface para essas bibliotecas maiores. Oferece abstrações bastante intuitivas, o que facilita bastante o desenvolvimento de redes neurais profundas.

3.2.7 Flask

A biblioteca *Flask*[8] (Versão 1.1.1) é um micro-framework de desenvolvimento Python. Criado por Armin Ronacher em 2010 e é destinada sobretudo a pequenas aplicações - por exemplo, um site básico como o que foi desenvolvido para este projeto.

[4] Disponível em: <https://matplotlib.org/>. Acesso em: 24 mai. 2022.
[5] Disponível em: <https://scikit-learn.org/>. Acesso em: 24 mai. 2022.
[6] Disponível em: <https://www.tensorflow.org/>. Acesso em: 24 mai. 2022.
[7] Disponível em: <https://keras.io/>. Acesso em: 24 mai. 2022.
[8] Disponível em: <https://flask.palletsprojects.com/en/1.1.x/>. Acesso em: 24 mai. 2022.

O Flask tem como características a sua simplicidade e a rapidez no desenvolvimento o que faz com que seus projetos tendam a ser menores e mais leves; porém, nada o impede de desenvolver projetos mais robustos, dada a sua alta capacidade de personalização.

De acordo com o *Stack Overflow Developer Survey de 2021*[9], o Flask é o sétimo web framework mais usado pelos participantes da pesquisa.

3.2.8 FastAPI

A biblioteca *FastAPI*[11] (Versão 0.79.0) é um web-framework de desenvolvimento em Python para o desenvolvimento de APIs REST.

De acordo com o *Stack Overflow Developer Survey de 2021*[12], o FastAPI é o décimo quinto web framework mais usado pelos participantes da pesquisa.

3.3 Ambientes de desenvolvimento do projeto

Visando uma maior organização e, sobretudo, o versionamento do projeto visando garantir a reprodutibilidade, os seguintes ambientes de desenvolvimento foram usados:

3.3.1 Jupyter Notebook

O ambiente principal de desenvolvimento utilizado sobretudo no ambiente de pesquisa foi o *Jupyter Notebook*[14]. Originou-se a partir do antigo IPython no ano de 2014, possui suporte a ambientes de execução em dezenas de linguagens de programação, sendo as mais utilizadas as linguagens *Julia*, *Python* e *R* (originando o seu nome).

Uma das grandes vantagens do seu uso é a facilidade para criação de protótipos de forma rápida (ideal para ambientes de pesquisa) além de uma interface amigável, necessidade de instalações extras mínimas, além da possibilidade de escritas de textos em Markdown, o que torna a documentação muito mais poderosa e fácil de fazer.

---

[9] Disponível em: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks>. Acesso em: 24 jun. 2022.

[11] Disponível em: <https://fastapi.tiangolo.com/>. Acesso em: 24 mai. 2022.

[12] Disponível em: <https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks>. Acesso em: 24 jun. 2022.

[14] Disponível em: https://jupyter.org/. Acesso em: 24 jun. 2022.

3.3.2 Anaconda

Anaconda[15] é uma distribuição livre, baseada em Python, que realiza a gerência de pacotes e implantação de sistemas.

Utilizado principalmente para ciência de dados em virtude da forma como os pacotes são instalados e gerenciados; diferente do modo de instalação PIP que instala apenas o pacote solicitado, um ambiente Anaconda também irá instalar as dependências necessárias para o correto funcionamento daquele pacote, o que, para ciência de dados, é de fundamental importância, pois muitas vezes dependências de diferentes versões podem não fazer com que o código apresente erros explícitos, o que leva o desenvolvedor a crer que o modelo está funcionando de maneira ideal, o que nem sempre é verdade. O gerenciamento de pacotes Anaconda resolve esse problema.

3.3.3 Visual Studio (VS) Code

O *VS Code*[16] é um editor de código fonte criado pela Microsoft©. Gratuito e de código aberto, é uma plataforma leve, que é executada nativamente em Windows, Linux e MacOS.

Atende uma enorme gama de projetos em mais de 30 diferentes linguagens de programação disponíveis.

De acordo com o *Stack overflow Developer Survey de 2021*[17], o VS Code é a IDE mais usada por 71,06% dos desenvolvedores pesquisados.

3.4 Desenvolvimento em ambiente de produção

A partir dessa etapa, todos os códigos descritos serão criados em um novo caderno Jupyter - focado especificamente para o ambiente de produção.

O primeiro passo para a criação desse novo caderno é um estudo minucioso do código de pesquisa, de modo a determinar quais bibliotecas deverão ser importadas para execução de uma consulta, pois algumas bibliotecas são exclusivas para a análise dos dados, divisão de dados de treino e teste, modelagem da rede, plotagem de gráficos e treinamento de geração de arquivo de modelo - procedimentos que não voltarão a ser repetidos no ambiente de produção do projeto

[15] Disponível em: https://www.anaconda.com/. Acesso em: 24 jun. 2022.
[16] Disponível em: https://code.visualstudio.com/. Acesso em: 24 jun. 2022.
[17] Disponível em: <https://insights.stackoverflow.com/survey/2021z#integrated-development-environment>. Acesso em: 24 jun. 2022.

desenvolvido nesse estudo.

Dependendo das características e extensão do pipeline - se fizesse necessário funções de retreino e reimplantação automatizado por exemplo; poderíamos ter que reaproveitar também outras funções para aplicar técnicas de CI/CD (Continuous Integration / Continuous Delivery) - um método que engloba um conjunto de práticas de modo que as equipes de desenvolvimento possam entregar alterações no código / modelo com mais frequência e confiabilidade. Porém tais técnicas fogem do escopo desse estudo.

Para este trabalho, o nosso arquivo de produção irá carregar tão somente a biblioteca Keras para carregar o modelo .h5 criado no ambiente de pesquisa, e as bibliotecas NumPy para transformação das imagens recebidas e conversão no formato de array (necessário para correta leitura do modelo) e a biblioteca PIL que será usada tão somente para abrir as imagens enviadas para a API.

3.4.1 Implementação da API REST usando FastAPI

O primeiro passo para a implantação da API REST em FastAPI é a instalação do próprio em seu ambiente Anaconda além da biblioteca Uvicorn[19], também necessária para a correta execução do FastAPI.

O próximo passo será a criação de um arquivo main.py, que será a nossa API propriamente dita. Para isso, basta tão somente importar as bibliotecas da mesma forma como foi feito no caderno Jupyter de produção, e em seguida transpor as etapas de importação de modelo, carregamento e tratamento de imagem, e predição em 3 funções POST que deverão ser chamadas na execução da APIs - seguindo a estrutura básica de um arquivo FastAPI de acordo com o *Tutorial básico*[20].

Em seguida, deve-se criar um arquivo .txt com todos os requisitos a serem instalados para o correto funcionamento da API. No caso desse projeto, nosso arquivo conterá tão somente os nomes e versões do NumPy, TensorFlow (inclui o Keras em sua biblioteca) e o PIL. O FastAPI precisa desse arquivo (normalmente chamado de *requirements.txt*) para fazer automaticamente a instalação dessas dependências de modo que o sistema funcione corretamente.

Para finalizar a estrutura da aplicação, devemos inserir na pasta o arquivo .h5 gerado na etapa de pesquisa.

[19] Disponível em: https://www.uvicorn.org/ Acesso em: 24 jun. 2022.
[20] Disponível em: https://fastapi.tiangolo.com/tutorial/ Acesso em: 24 jun. 2022.

No caso desse projeto, também foi inserido uma pasta chamada *samples* onde foram inseridas várias imagens de mucosa de ovinos para teste.

Uma vez que a estrutura está finalizada, devemos iniciar o servidor através do seguinte comando:

uvicorn main:app –reload

Em seguida, será informado o endereço local onde o servidor poderá ser acessado - normalmente em: *https://127.0.0.1:8000*. Ao abrir o endereço anterior a API estará funcionando normalmente.

3.4.2 Implementação da API REST + Site usando Flask

Iniciaremos nossa implantação instalando em nosso ambiente Anaconda a biblioteca Flask.

Em seguida, uma nova pasta para a nova aplicação deverá ser criada. Para isso podemos usar a estrutura básica de acordo com a documentação de *início rápido do Flask*[21].

Dentro da pasta *app* deverá ser criado um arquivo chamado *views.py*. Esse arquivo conterá uma versão adaptada do nosso caderno Jupyter de produção onde de acordo com as rotas deverão ser inseridas as funções de leitura do modelo, carregamento de imagens, tratamento de imagens e classificação da imagem enviada.

O próximo passo é criar dentro da pasta *template* um arquivo *index.html* que irá ser o nosso site que irá receber a imagem do usuário, irá armazenar a imagem na pasta *uploads*, irá consultar o arquivo views.py e retornará na tela o resultado da classificação.

Executar o projeto é simples. Basta executar o arquivo app.py na pasta raíz do projeto e abrir a URL informada em um navegador.

[21] Disponível em: https://flask.palletsprojects.com/en/1.1.x/quickstart/ Acesso em: 24 jun. 2022.

4 RESULTADOS

Para a análise dos resultados foram criados três modelos em ambiente de pesquisa. Estes modelos foram detalhadamente descritos no apêndice B deste trabalho e o que apresentou melhor acurácia foi levado para o ambiente de produção, sendo disponibilizado em uma API REST criada em FastAPI, e um pequeno site que faz uso da API criada em Flask. Analisaremos a seguir os principais resultados encontrados.

4.1 Análise das métricas e estrutura do modelo de RNC

Nesta seção serão apresentadas as métricas dos modelos de AM criados no ambiente de pesquisa.

4.1.1 Características dos modelos e resultados

4.1.1.1 Modelo 1: CNN baseada na arquitetura AlexNet

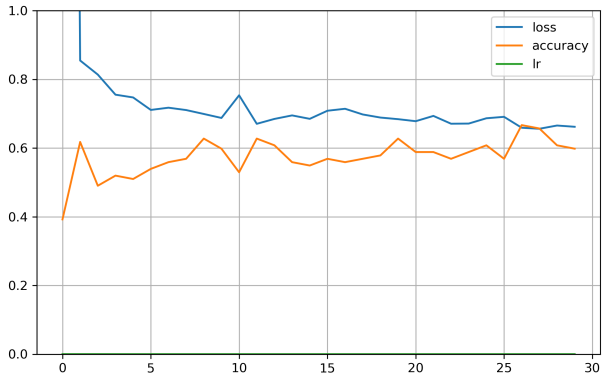
No gráfico da Figura 5, temos o gráfico da perda versus a precisão do modelo. A perda se trata de uma medida em que podemos mensurar o quão distante da resposta correta ocorreu a predição do modelo. No gráfico da Figura 5, esta medida está representada pela linha em azul. Ao observar esta métrica, é possível notar que ela possui uma tendência de descida conforme o treino avança, que é o comportamento esperado. Porém, a queda total foi relativamente baixa, o que irá fazer com que este modelo não seja talvez o mais adequado para uma situação de produção.

A acurácia - métrica geral baseada no número de acertos - que se encontra indicada no gráfico da Figura 5 pela linha em amarelo aumenta conforme o treinamento avança. Porém, ao final do treinamento, o aumento total não atingiu níveis satisfatórios - valores acima de 80%, o que torna o modelo inadequado para uso em ambiente de produção.

Outra métrica importante de ser discutida ao se estudar um modelo de Machine Learning é a matriz de confusão, que nada mais é do que uma tabela que indica de uma forma visualmente mais amigável o número de acertos e erros de um dado modelo.

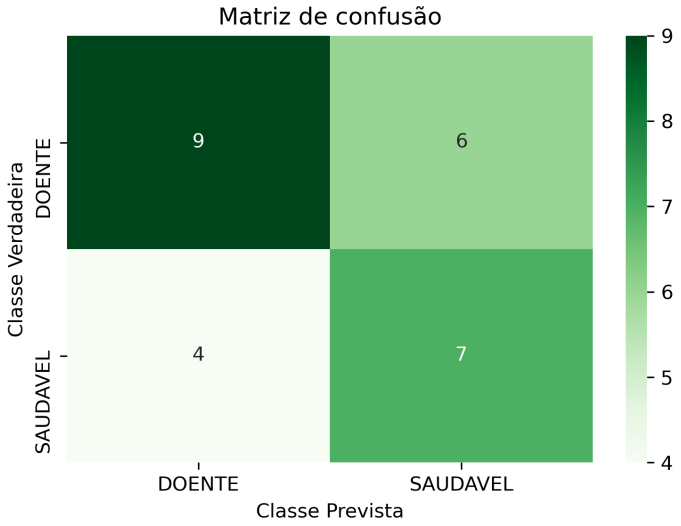
A matriz de confusão deste modelo se encontra ilustrada na Figura 6. Através dela a matriz foi dividida em 4 partes. Pode-se observar que de 13 amostras animais doentes, o modelo acertou a classificação de 9 e das 13 amostras de animais saudáveis, o modelo acertou 7.

Figura 5 – Gráfico de perda versus precisão ao longo do treinamento da rede AlexNet



Fonte: Elaborado pelo autor.

Figura 6 – Matriz de confusão da rede AlexNet



Fonte: Elaborado pelo autor.

Podemos observar que esse modelo teve uma alta quantidade de previsões erradas, confirmando o observado na discussão anterior. Esse modelo teve desempenho abaixo do ideal para ter seu uso cogitado em um ambiente de produção.

A seguir vamos mensurar numericamente a performance desse modelo. Para isso utilizaremos o relatório de classificação do Scikit-Learn - uma biblioteca de aprendizado de máquina para a linguagem de programação Python.

O relatório de classificação é utilizado para medir a qualidade das predições de um determinado modelo. O relatório fornece 4 informações: precisão, recuperação, pontuação f1 e suporte.

A precisão indica qual a porcentagem das previsões foram feitas de forma correta. A

recuperação indica qual a porcentagem dos casos positivos que foram capturados pelo modelo. É uma métrica calculada a partir da proporção entre o total de casos positivos e a soma dos positivos verdadeiros e os falsos negativos.

A pontuação f1 é uma métrica que indica a porcentagem de previsões corretas de um dado modelo. É uma média harmônica ponderada da precisão e a recuperação. O suporte é simplesmente o número de elementos de cada classe que foram analisados.

A acurácia do modelo, indicada no relatório de classificação, baseada no pontuação f1, será usada como medida geral da qualidade de cada modelo.

O relatório de classificação deste modelo se encontra ilustrado na Tabela 2.

Tabela 2 – Relatório de Classificação do modelo baseado na arquitetura AlexNet

	precisão	recuperação	pontuação f1	amostras
doente	0.69	0.60	0.64	15
saudável	0.54	0.64	0.58	11
acurácia			0.62	26
média aritmética	0.62	0.62	0.61	26
média ponderada	0.63	0.62	0.62	26

Fonte: Elaborado pelo autor.

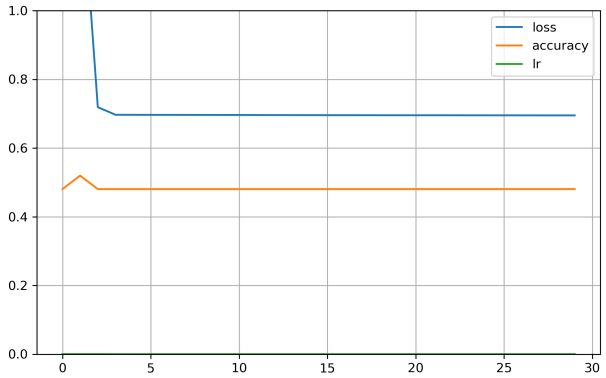
A Tabela 2 mostra uma acurácia de 0.62 (62%), apresentando alguns erros consideráveis ao classificar animais doentes como saudáveis. Isso pode ser um problema em uma aplicação real deste modelo, pois pode fazer com que um número considerável de animais doentes não sejam vermifugados e venham a apresentar problemas mais graves ou até mesmo óbito, gerando prejuízos aos produtores. Uma possibilidade seria retreinar a rede com uma base de imagens maior, ou trabalhar outras arquiteturas de rede, que é o que foi feito a seguir.

4.1.1.2 Modelo 2: CNN baseada na arquitetura LeNet-5

O modelo 2 - que utilizou-se de uma arquitetura baseada na LeNet-5, foi certamente a que pior se apresentou das três. Analisando inicialmente o gráfico de perda vs precisão da Figura 7 já é possível observar que nem sequer o comportamento padrão da perda e precisão se comportaram da forma esperada. A perda cai muito rapidamente e permanece sem modificações ao longo de todo treinamento, e a precisão muda ligeiramente no início do treinamento, porém volta ao estado inicial e não muda ao longo de todo o resto do treinamento.

A matriz de confusão da Figura 8 indica o problema inicial apresentado anteriormente.

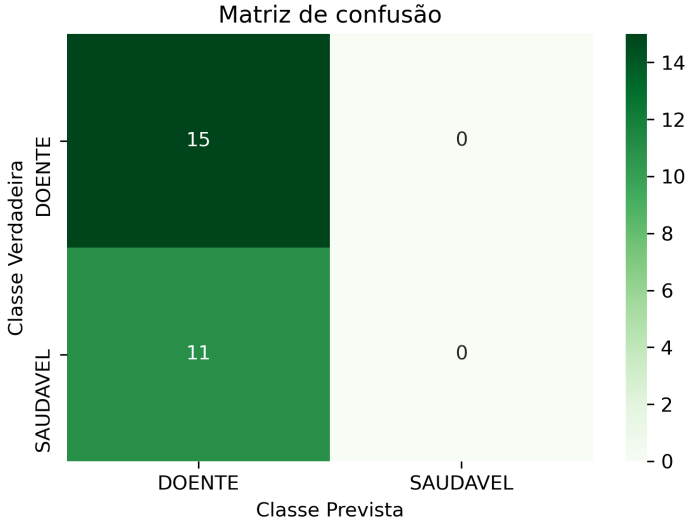
Figura 7 – Gráfico de perda versus precisão ao longo do treinamento da rede LeNet-5



Fonte: Elaborado pelo autor.

Este modelo classificou todas as amostras apenas em uma classe, o que torna esse modelo totalmente inviável para nossa aplicação.

Figura 8 – Matriz de confusão da rede LeNet-5



Fonte: Elaborado pelo autor.

O relatório de classificação deste modelo também reforça o problema apresentado na matriz de confusão. O modelo classificou todas as amostras apenas em uma classe, gerando uma precisão geral baixa e baixou a zero a acurácia de uma das classes, inviabilizando seu uso.

O modelo baseado na arquitetura LeNet5 apresentou o pior comportamento de todas arquiteturas testadas, inviabilizando totalmente seu uso para fins de produção.

Tabela 3 – Relatório de Classificação do modelo baseado na arquitetura LeNet-5

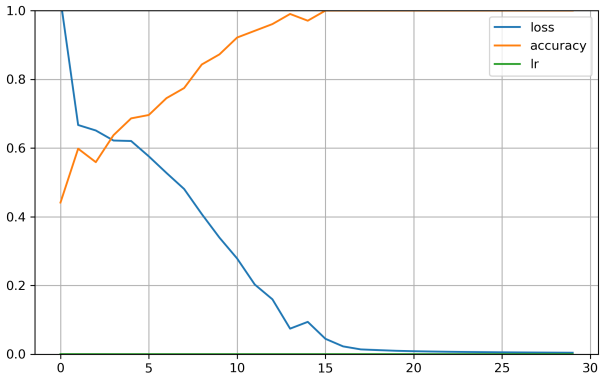
	precisão	recuperação	pontuação f1	amostras
doente	0.58	1.00	0.73	15
saudável	0.00	0.00	0.00	11
acurácia			0.58	26
média aritmética	0.29	0.50	0.37	26
média ponderada	0.33	0.58	0.42	26

Fonte: Elaborado pelo autor.

4.1.1.3 Modelo 3: CNN baseada na arquitetura ZFNet

A arquitetura ZFNet foi a que de longe melhor se comportou para a aplicação deste trabalho, apresentando melhores métricas que serão discutidas parte a parte a seguir.

Figura 9 – Gráfico de perda versus precisão ao longo do treinamento da rede ZFNet

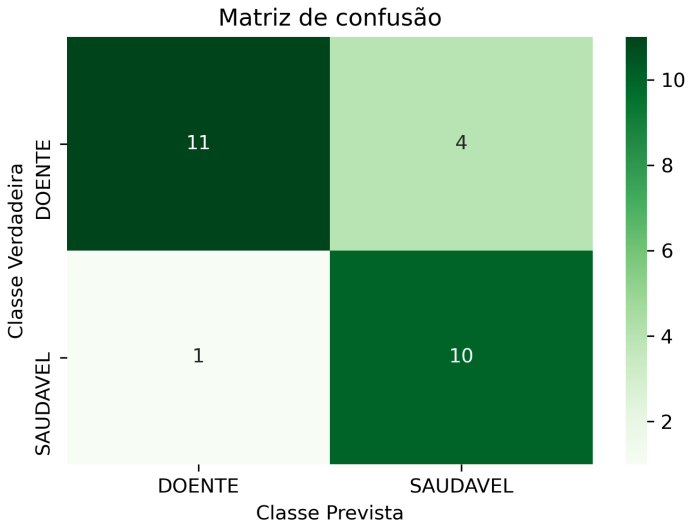


Fonte: Elaborado pelo autor.

O gráfico da Figura 9 apresenta exatamente o comportamento esperado de um modelo de Machine Learning. A acurácia sobe o mais próximo possivel de 1 (100%), enquanto a perda desce rumo a valores próximos de zero. Neste modelo, percebe-se que a rede já converge rapidamente - próximo a 18ª época - que é o número de iterações nos dados em um ciclo de treinamento - permanecendo estável até o final de todo o treino.

A matriz de confusão ilustra bem o que o gráfico de treinamento da rede mostra. Além do que, pode-se observar na matriz que ele não apenas classificou bem de uma forma geral, mas cada classe teve altos índices de acerto de classificação - errando apenas 1 de 11 para animais saudáveis e 4 de 15 para animais doentes, o que torna esse modelo de longe o mais promissor para uso em produção.

Figura 10 – Matriz de confusão da rede ZFNet



Fonte: Elaborado pelo autor.

Tabela 4 – Relatório de Classificação do modelo baseado na arquitetura ZFNet

	precisão	recuperação	pontuação f1	amostras
doente	0.92	0.73	0.81	15
saudável	0.71	0.91	0.80	11
acurácia			0.81	26
média aritmética	0.82	0.82	0.81	26
média ponderada	0.83	0.81	0.81	26

Fonte: Elaborado pelo autor.

O relatório de classificação do modelo ilustrado na Tabela 4 também mostra em números o bom desempenho apresentado. O modelo apresentou uma acurácia geral de 0.81 (81%), sendo as pontuações f1 de cada classe 0.81 e 0.80 para cada uma das classes doente e saudável, respectivamente.

Por fim, podemos agrupar os três modelos e fazer a escolha de acordo com a Tabela 5.

Tabela 5 – Comparativo dos modelos e suas respectivas acurácias por classe e geral

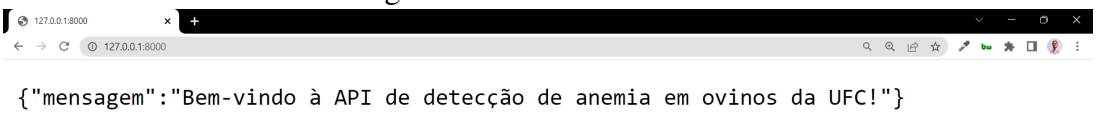
	AlexNet	Lenet-5	ZFNet
doente	0.64	0.73	0.81
saudável	0.58	0.00	0.80
acurácia	0.62	0.58	0.81

Fonte: Elaborado pelo autor.

De acordo com as métricas apresentadas, utilizaremos o modelo criado a partir da arquitetura baseada em ZFNet no ambiente de produção, em virtude da sua maior precisão, tanto geral quanto em cada uma das classes.

4.2 Resultados da implantação do modelo no FastAPI

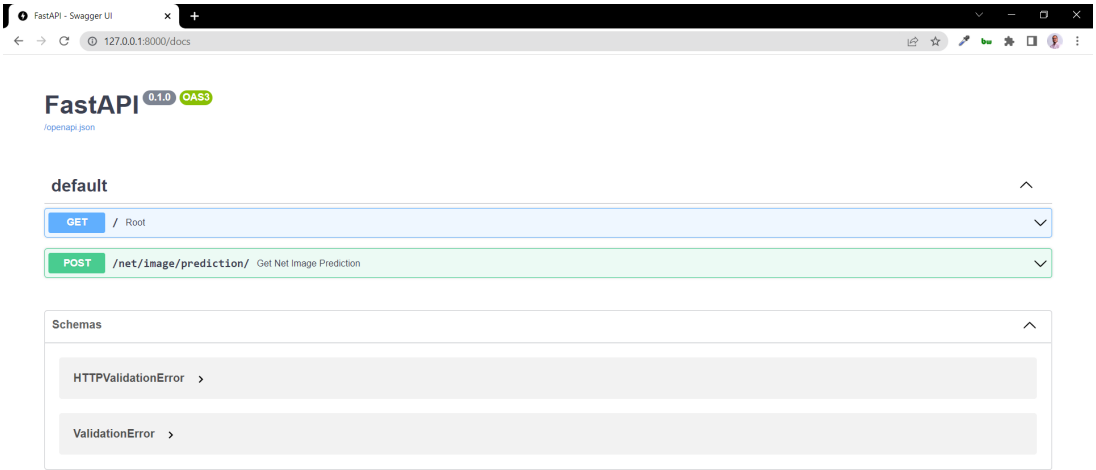
Figura 11 – Tela inicial da API



Fonte: Elaborado pelo autor.

Tela inicial da API com a mensagem inicial (Figura 11). Ao entrar na pasta *https://127.0.0.1:8000/docs* podemos ver a tela inicial da Swagger UI (Figura 12) onde poderemos fazer testes na API e verificar se ela está fazendo classificações de forma adequada de acordo com o modelo criado no ambiente de pesquisa (Figura 12).

Figura 12 – Tela inicial da Swagger UI

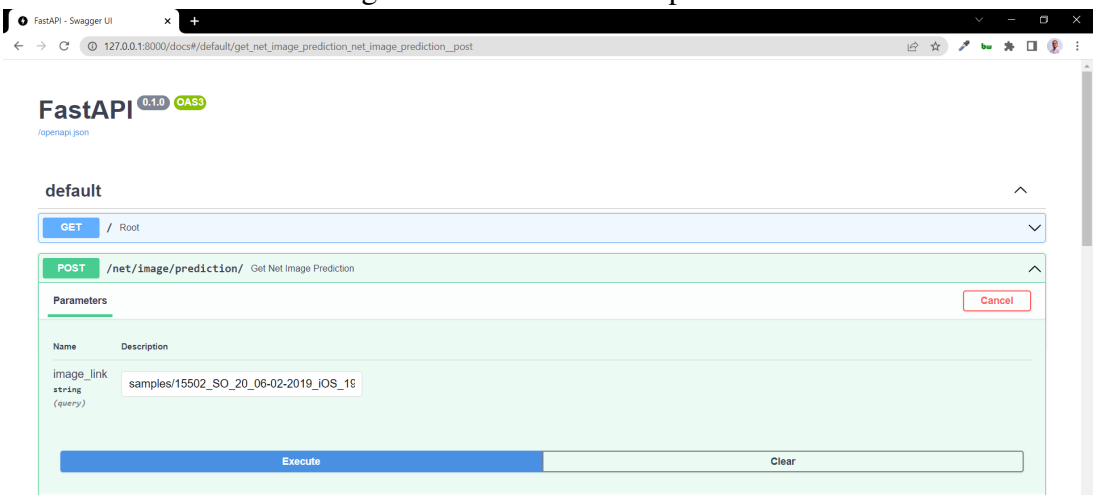


Fonte: Elaborado pelo autor.

Ao solicitar um método POST e clicar em *Try it out*, poderemos enviar o endereço de uma imagem para que a API retorne a classificação. Neste exemplo, foi usada uma imagem exemplo da pasta *samples* (Figura 13).

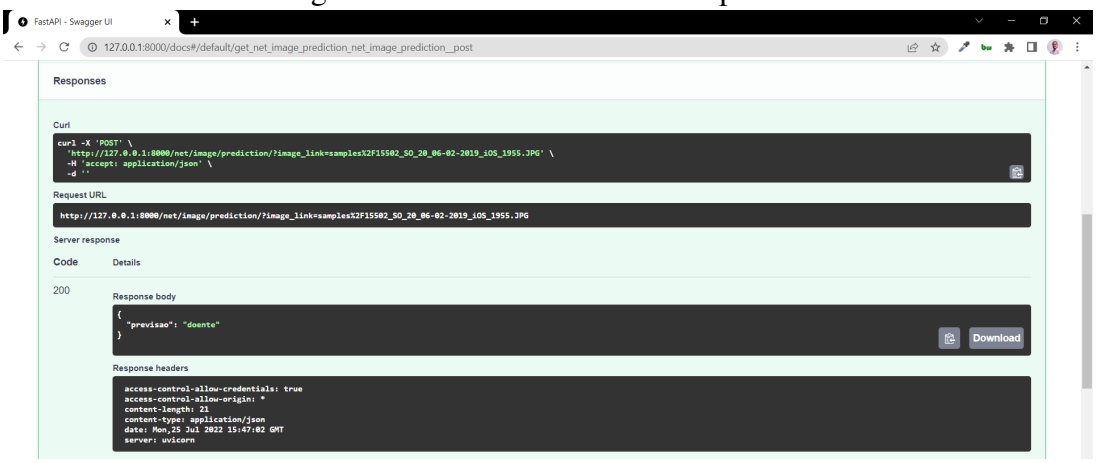
Ao clicar em *execute*, a API irá retornar o resultado da classificação (Figura 14).

Figura 13 – Fazendo uma previsão



Fonte: Elaborado pelo autor.

Figura 14 – Obtendo o retorno da previsão



Fonte: Elaborado pelo autor.

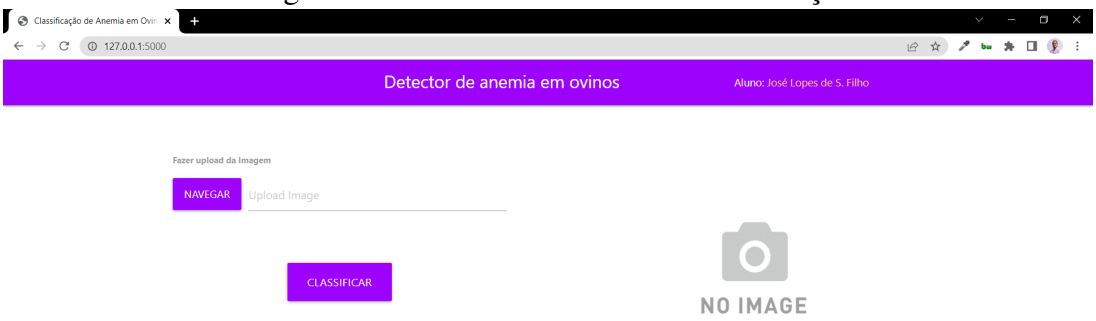
4.3 Resultados da implantação do modelo em Flask

A segunda implementação do modelo é um pequeno site que faz uso do modelo gerado no ambiente de pesquisa e retorna a classificação, de forma semelhante ao que foi visto no FastAPI.

Esta é a tela inicial do site criado com Flask que faz uso do modelo criado (Figura 15).

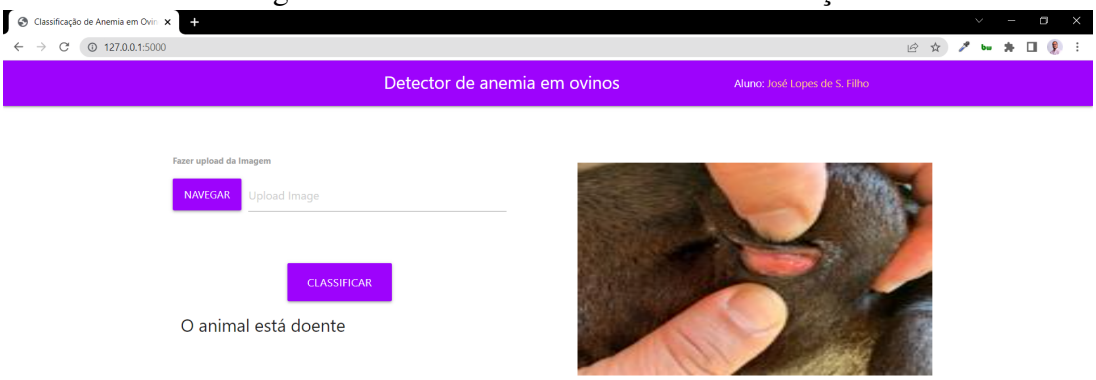
Esta é a tela do site criado com Flask ao retornar o resultado da classificação utilizando uma imagem de exemplo que faz uso do modelo criado (Figura 16).

Figura 15 – Tela inicial do site de classificação



Fonte: Elaborado pelo autor.

Figura 16 – Retorno de um teste de classificação



Fonte: Elaborado pelo autor.

4.4 Discussões

Este trabalho foi dividido em dois momentos. O trabalho no ambiente de pesquisa e o trabalho no ambiente de produção.

A fase do ambiente de pesquisa gerou três modelos criados utilizando diferentes arquiteturas de RNCs, onde uma delas - a rede baseada na arquitetura ZFNet se mostrou bastante eficiente, pois mesmo utilizando menos recursos computacionais em função do baixo poder de processamento da máquina em que ocorreram os treinamentos, ainda assim obteve uma precisão bastante satisfatória para uma base que conta com apenas 128 imagens no total.

Já a fase de implantação apresentou resultados muito satisfatórios, o FastAPI se demonstrou, no final das contas, um micro-framework simples e poderoso, que apesar de novo em comparação com outras opções do mercado, já tem uma velocidade de adoção bastante elevada, o que prova a sua robustez e capacidade de atender os mais variados tipos de projetos.

O framework Flask também mostrou maiores possibilidades ao criarmos um pequeno site de teste capaz de acessar o modelo criado e oferecer o serviço de forma direta ao usuário.

Ambas opções se mostraram rápidas e estáveis para uso.

5 CONCLUSÕES

Ao longo deste trabalho, foram implementadas algumas técnicas de ciência de dados, voltadas para a área de visão computacional, no intuito de criar um modelo que fosse capaz de prever com uma certa margem de segurança a necessidade de vermifugar (ou não) ovinos por meio de fotos da mucosa ocular dos animais analisados.

Para isso, foram desenvolvidos três modelos que utilizam Redes Neurais Convolucionais - técnica robusta e eficiente, e que pode gerar bons resultados não apenas na área de visão computacional, mas também processamento de linguagem natural e reconhecimento de voz.

Ao longo deste trabalho, foi possível comprovar o potencial de tais redes, pois mesmo utilizando um banco de imagens considerado pequeno (128 amostras) e usando configurações que visaram torná-la mais leve em termos de necessidade de recursos computacionais, ainda assim foi possível obter um modelo baseado na arquitetura ZFNet com métricas promissoras para uso em uma aplicação real.

Porém, neste projeto, mais do que criar um modelo de classificação, foi proposta a possibilidade de usar esse modelo em situações próximas ao que seria um uso real desse modelo de classificação.

Para isso, foram escolhidos dois micro-frameworks baseados em Python bastante populares e muito usados pelos profissionais da área. O Flask - mais robusto, com maior poder de personalização foi usado para criar um mini website que fosse capaz de oferecer diretamente aos usuários a possibilidade deles mesmos enviarem suas imagens e obterem suas classificações.

Já com o FastAPI, foi criado uma API REST que pode ser servida nas mais diversas plataformas e serviços de terceiros como sites, aplicativos móveis e programas.

Podemos afirmar que todos os objetivos deste trabalho foram alcançados de forma bastante satisfatória. Com relação ao objetivo geral proposto, foi possível desenvolver um modelo preditivo bastante promissor, baseado na arquitetura ZFNet que apresentou um excelente percentual de precisão nas duas classes propostas.

Além disso, o mesmo modelo foi implementado de forma satisfatória tanto em uma API que se utiliza dele para fornecer dados preditivos, quando em um Web App que também realiza essas predições.

Quanto aos objetivos específicos, pode-se obter três modelos e compará-los utilizando como métrica para escolha a acurácia geral de cada um deles. O modelo escolhido foi capaz de classificar as imagens das mucosas dos ovinos e as API's foram criadas, classificando

as imagens testadas.

5.1 Trabalhos Futuros

Como trabalhos futuros, pretende-se continuar a coletar imagens de mucosas de animais saudáveis e doentes de modo a aumentar o banco de imagens e tornar as versões futuras dos modelos cada vez mais confiáveis.

Pretende-se também ampliar as possibilidades de implantação desses modelos para as principais plataformas de nuvem como Amazon Web Services (AWS), Microsoft Azure e Google Cloud.

Uma outra pretensão será de criar a partir desse estudo um pipeline bem mais robusto de aprendizagem de máquina, que utilize ferramentas como Docker e Circle CI para implementar um pipeline de retreino completamente automatizado, minimizando ao máximo a interferência humana.

REFERÊNCIAS

ALMEIDA, A. M. A. **Detecção de Anemia em Ovinos através de Aprendizagem Profunda em Imagens de Mucosa Ocular.** Dissertação (Mestrado) — Universidade Federal do Ceará, 2021.

AROSEMENA, N.; BEVILAQUA, C.; MELO, A.; GIRÃO, M. Seasonal variations of gastrointestinal nematodes in sheep and goats from semi-arid area in brazil. **Revue de Medecine Veterinaire (France)**, 1999.

CARVALHO, C.; MOLENTO, M. **Embrapa Famacha Circular Tecnica 52**. 2015.

CHAGAS, A. d. S.; CARVALHO, C. de; MOLENTO, M. Método famacha: um recurso para o controle da verminose em ovinos. **Embrapa Pecuária Sudeste-Circular Técnica (INFOTECA-E)**, São Carlos, SP: Embrapa Pecuária Sudeste, 2007., 2007.

CHOLLET, F. Xception: Deep learning with depthwise separable convolutions. **CoRR**, abs/1610.02357, 2016. Disponível em: <http://arxiv.org/abs/1610.02357>.

DEMOLINER, G.; ALVES, R. J. F. Anemimetro: app móvel para implementação do método famacha. **Unoesc & Ciência-ACET**, v. 8, n. 1, p. 25–32, 2017.

FUKUSHIMA, K.; MIYAKE, S.; ITO, T. Neocognitron: A neural network model for a mechanism of visual pattern recognition. **IEEE transactions on systems, man, and cybernetics**, IEEE, n. 5, p. 826–834, 1983.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Deep residual learning for image recognition. **CoRR**, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A Fast Learning Algorithm for Deep Belief Nets. **Neural Computation**, v. 18, n. 7, p. 1527–1554, 07 2006. ISSN 0899-7667. Disponível em: <https://doi.org/10.1162/neco.2006.18.7.1527>.

HU, J.; SHEN, L.; SUN, G. Squeeze-and-excitation networks. **CoRR**, abs/1709.01507, 2017. Disponível em: <http://arxiv.org/abs/1709.01507>.

IBGE. **Rebanho de Ovinos (Ovelhas e Carneiros) no Brasil | IBGE**. 2020. Urlhttps://www.ibge.gov.br/explica/producao-agropecuaria/ovino/br.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: PEREIRA, F.; BURGES, C.; BOTTOU, L.; WEINBERGER, K. (Ed.). **Advances in Neural Information Processing Systems**. Curran Associates, Inc., 2012. v. 25. Disponível em: <https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>.

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

MITANI, A.; HUANG, A.; VENUGOPALAN, S.; CORRADO, G. S.; PENG, L.; WEBSTER, D. R.; HAMMEL, N.; LIU, Y.; VARADARAJAN, A. V. Detection of anaemia from retinal fundus images via deep learning. **Nature Biomedical Engineering**, Nature Publishing Group, v. 4, n. 1, p. 18–27, 2020.

MOLENTO, M.; DANTAS, J. Validação do guia famacha para diagnóstico clínico de parasitoses em pequenos ruminantes no brasil: resultados preliminares. **ENCONTRO INTERNACIONAL SOBRE AGROECOLOGIA E DESENVOLVIMENTO RURAL SUSTENTÁVEL**, v. 1, p. 51, 2001.

MOLENTO, M. B.; TASCA, C.; GALLO, A.; FERREIRA, M.; BONONI, R.; STECCA, E. Método famacha como parâmetro clínico individual de infecção por haemonchus contortus em pequenos ruminantes. **Ciência Rural**, SciELO Brasil, v. 34, p. 1139–1145, 2004.

MORTENSEN, L. L.; WILLIAMSON, L. H.; TERRILL, T. H.; KIRCHER, R. A.; LARSEN, M.; KAPLAN, R. M. Evaluation of prevalence and clinical implications of anthelmintic resistance in gastrointestinal nematodes in goats. **Journal of the American Veterinary Medical Association**, Am Vet Med Assoc, v. 223, n. 4, p. 495–500, 2003.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence: a modern approach**. 3. ed. [S.l.]: Pearson, 2009.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SOUZA, L. F. de; MIOR, L. B.; COSTA, M. H.; RIET-CORREA, B. Sistema para classificação de infestação parasitária em pequenos ruminantes. 2021.

STACKEXCHANGE. **Understanding how convolutional layers work**. 2020. Urlhttps://datascience.stackexchange.com/questions/80436/understanding-how-convolutional-layers-work.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**. [S.l.: s.n.], 2015.

TAMIR, A.; JAHAN, C. S.; SAIF, M. S.; ZAMAN, S. U.; ISLAM, M. M.; KHAN, A. I.; FATTAH, S. A.; SHAHNAZ, C. Detection of anemia from image of the anterior conjunctiva of the eye by image processing and thresholding. In: IEEE. **2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)**. [S.l.], 2017. p. 697–701.

WANG, T.; WEN, C.-K.; WANG, H.; GAO, F.; JIANG, T.; JIN, S. Deep learning for wireless physical layer: Opportunities and challenges. **China Communications**, v. 14, p. 92–111, 10 2017.

APÊNDICE A – REPOSITÓRIO COM OS CÓDIGOS DO TRABALHO

Todos os códigos criados para este trabalho estão disponíveis na plataforma *GitHub* no repositório <https://github.com/mrjlsouza/tcc-deteccao-anemia>

APÊNDICE B – DESENVOLVIMENTO EM AMBIENTE DE PESQUISA

Essa etapa do projeto, tem por objetivo coletar as imagens que serão usadas nas etapas de treino, teste e validação, analisar o banco de imagens coletando informações relevantes a respeito delas, pré-processar para uso adequado dentro de uma RNC, treinar o modelo de classificação das imagens e avaliar a qualidade desse modelo por meio de métricas adequadas. Cada uma dessas etapas será descrita com mais detalhes a seguir:

B.0.1 Criação do ambiente de pesquisa

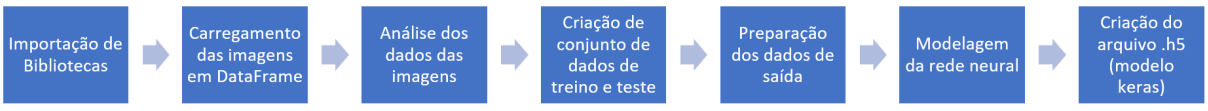
Para criar o ambiente de pesquisa, partiremos do princípio que o software de desenvolvimento Anaconda já está devidamente instalado na máquina. Caso não esteja basta realizar a instalação e criação do ambiente seguindo as instruções da *documentação oficial*[18] do software.

A criação de um ambiente básico já instala automaticamente vários pacotes além do Jupyter Notebook pronto para uso. Porém, faz-se necessário também a instalação das bibliotecas mencionadas anteriormente. As documentações de cada biblioteca possuem instruções para suas respectivas instalações no ambiente Anaconda.

B.0.2 Criação do código de pesquisa

A partir dessa etapa, todos os códigos deverão ser criados dentro de um caderno Jupyter seguido os passos do fluxograma abaixo.

Figura 17 – Fluxograma dos passos de criação do modelo de rede neural



Fonte: Elaborado pelo autor.

O primeiro passo para criação de um modelo de AM é a importação das bibliotecas necessárias para dentro do caderno Jupyter (não basta instalar em seu ambiente anaconda).

Uma vez importadas todas as bibliotecas, pode-se iniciar o passo seguinte, que é o do carregamento dos dados de imagens. Existem diversas formas de carregar um banco de imagens para dentro do código de pesquisa; para esse projeto foi decidido separar as classes

[18] Disponível em: <https://docs.anaconda.com/>. Acesso em: 24 jun. 2022.

de imagens doentes e saudáveis para duas pastas separadas e carregar os caminhos para cada imagem dentro de um DataFrame junto de suas respectivas classificações (saídas).

Figura 18 – Dataframe de caminhos de imagens e respectivas saídas

	image	target
0	imagens\doente\15511_SO_17_03-10-2018_iOS_3416...	doente
1	imagens\doente\15511_SO_23_30-01-2019_iOS_1617...	doente
2	imagens\doente\15807_SI_16_06-02-2019_iOS_2035...	doente
3	imagens\doente\15811_SI_22_27-02-2019_iOS_2448...	doente
4	imagens\doente\162_SI_22_13-02-2019_iOS_5189.jpg	doente
5	imagens\doente\162_SI_23_28-11-2018_iOS_0229.jpg	doente
6	imagens\doente\16509_SO_22_30-01-2019_iOS_1710...	doente
7	imagens\doente\16525_SO_23_12-09-2018_iOS_2366...	doente
8	imagens\doente\165_SI_23_13-02-2019_Android_08...	doente
9	imagens\doente\17052_MN_16_30-01-2019_iOS_1601...	doente

Fonte: Elaborado pelo autor.

A Figura 18 ilustra a saída de um dos DataFrames criados, onde pode-ser ver o caminho até cada uma das imagens e a saída da imagem como doente ou saudável.

Agora é possível analisar as imagens e obter informações importantes a respeito dos dados tanto para tratá-las no código de produção, quanto da qualidade do banco para treinamento do modelo nos passos seguintes.

Neste trabalho foram feitas algumas checagens simples iniciais como o número de imagens por classe (doentes e saudáveis) e foram abertas algumas imagens para entender como são as imagens e como elas estão sendo exibidas.

O próximo passo é a criação de novos conjuntos de dados para treino e teste. Para este trabalho utilizou-se a proporção de *80% dos dados para treino e 20% para teste*. Outras proporções foram testadas, porém sem apresentar nenhuma melhora significativa na avaliação final do modelo.

Após separar os dados, os índices estarão fora de ordem, portanto, se faz necessário atribuir novos índices para cada um dos novos DataFrames criados.

O próximo alvo é preparar os dados de saída. Como se está trabalhando com múltiplas classes, inclusive com possibilidades futuras dos dados passarem a apresentar mais de duas classes, haja vista que o método FAMACHA possui 5 graus de cores, iremos usar o *one-hot-encoding* para transformar a saída em colunas binárias.

Uma vez trabalhada a saída, o código de pesquisa irá iniciar a transformação das imagens para um formato mais leve e compatível com o uso em RNCs.

O primeiro passo foi redimensionar as imagens para o tamanho de 150x150 pixels - a RNC foi treinada em um computador local, e o tamanho foi reduzido a fim de reduzir o tempo de treinamento e evitar sobreajuste.

Em seguida o formato do conjunto dos dados precisou ser modificado para *(n1, n2, n3, n4)* - onde n1 é o número de observações, n2 e n3 são a largura e o comprimento das imagens e n4 indica o número de planos por imagem; como estas são imagens coloridas (RGB), então precisaremos ter 3 planos por imagem.

O próximo passo é modelar as Redes Neurais Convolucionais (RNC) em três diferentes modelos. Para este trabalho, foram escolhidos modelos baseados nas arquiteturas Alexnet, LeNet-5 e ZFNet. Em seguida, será avaliado o modelo com melhor acurácia e este será aplicado na aplicação de produção.

O primeiro modelo (baseado na arquitetura AlexNet) se encontra descrito na Tabela 6. Vamos discutir cada uma das suas características.

As primeiras três camadas da rede usam 32 filtros com kernels de tamanho 3x3, mas nenhum *stride* porque as nossas camadas de entrada não ficaram muito grandes. Define também um formato de entrada na rede de 150, 150, 3 pois as imagens têm 150x150 pixels com 3 canais de cores (ou seja, colorido em formato RGB). A função de ativação usada nessas duas primeiras camadas é a RELU, muito utilizada em processamento de imagens, pois não satura na região positiva e converge bem mais rápido que as demais.

Em seguida, temos uma camada máxima de pooling que usa um tamanho de pool de 2, então, ela divide cada dimensão espacial por um fator de 2.

Depois, inserimos uma camada de Dropout que desliga aleatoriamente alguns neurônios junto com suas conexões apenas durante o treinamento com o intuito de evitar sobreajuste. Durante a predição todos os neurônios são reativados. É necessário informar a porcentagem de desligamento dos neurônios. Para este modelo, definimos um valor de 30%.

Depois, repetimos a mesma estrutura duas vezes: duas camadas convolucionais, seguidas por uma camada máxima de pooling seguida de mais uma camada de dropout. Para imagens maiores, poderíamos repetir essa estrutura diversas vezes.

O número de filtros aumenta à medida que escalamos a RNC em direção à camada de saída. Iniciamos com 32, depois com 64, depois com 128. É uma boa prática dobrar o número

Tabela 6 – Modelo 1 de Rede Neural Convolucional baseado na arquitetura AlexNet

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 148, 148, 32)	896
conv2d_7 (Conv2D)	(None, 146, 146, 32)	9248
conv2d_8 (Conv2D)	(None, 146, 146, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 72, 72, 32)	0
dropout_4 (Dropout)	(None, 72, 72, 32)	0
conv2d_9 (Conv2D)	(None, 70, 70, 64)	18496
conv2d_10 (Conv2D)	(None, 68, 68, 64)	36928
conv2d_11 (Conv2D)	(None, 66, 66, 64)	36928
max_pooling2d_4 (MaxPooling	(None, 33, 33, 64)	0
dropout_5 (Dropout)	(None, 33, 33, 64)	0
conv2d_12 (Conv2D)	(None, 31, 31, 128)	73856
conv2d_13 (Conv2D)	(None, 29, 29, 128)	147584
conv2d_14 (Conv2D)	(None, 27, 27, 128)	147584
max_pooling2d_5 (MaxPooling	(None, 13, 13, 128)	0
dropout_6 (Dropout)	(None, 13, 13, 128)	0
flatten_1 (Flatten)	(None, 21632)	0
dense_2 (Dense)	(None, 256)	5538048
dropout_7 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 2)	514

...
Total params: 6,019,330
Trainable params: 6,019,330
Non-trainable params: 0
Fonte: Elaborado pelo autor.

de filtros após cada camada de pooling pois a mesma divide cada dimensão espacial por um fator de 2, portanto podemos dobrar o número de filtros sem medo de explodir o número de parâmetros, o que aumentaria consideravelmente a carga computacional.

O segundo modelo criado para este trabalho é um modelo baseado na arquitetura LeNet-5, e se encontra descrito na tabela 7.

A primeira camada da rede usa 6 filtros com kernels de tamanho 5x5, também sem nenhum stride. Em seguida, temos uma camada média de pooling que usa um tamanho de pool de 2, seguido de uma camada de ativação sigmóide.

Depois, repetimos a mesma estrutura descrita anteriormente mais duas vezes: uma camada convolucional, seguida por uma camada média de pooling seguida de mais uma camada de ativação.

O número de filtros aumenta à medida que escalamos a RNC em direção à camada de saída. Iniciamos com 6, depois com 16, depois com 120.

Tabela 7 – Modelo 2 de Rede Neural Convolucional baseado na arquitetura LeNet-5

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 146, 146, 6)	456
average_pooling2d (AveragePo	(None, 73, 73, 6)	0
activation (Activation)	(None, 73, 73, 6)	0
conv2d_1 (Conv2D)	(None, 69, 69, 16)	2416
average_pooling2d_1 (Average	(None, 34, 34, 16)	0
activation_1 (Activation)	(None, 34, 34, 16)	0
conv2d_2 (Conv2D)	(None, 30, 30, 120)	48120
flatten (Flatten)	(None, 108000)	0
dense (Dense)	(None, 84)	9072084
dense_1 (Dense)	(None, 2)	170

...
Total params: 9,123,246
Trainable params: 9,123,246
Non-trainable params: 0

Fonte: Elaborado pelo autor.

O terceiro modelo criado para este trabalho é um modelo baseado na arquitetura ZFNet, e se encontra descrito na tabela 8.

Tabela 8 – Modelo 3 de Rede Neural Convolucional baseado na arquitetura ZFNet

Layer (type)	Output Shape	Param #
conv2d_48 (Conv2D)	(None, 144, 144, 96)	14208
max_pooling2d_30 (MaxPooling	(None, 48, 48, 96)	0
lambda_21 (Lambda)	(None, 48, 48, 96)	0
conv2d_49 (Conv2D)	(None, 44, 44, 256)	614656
max_pooling2d_31 (MaxPooling	(None, 14, 14, 256)	0
lambda_22 (Lambda)	(None, 14, 14, 256)	0
conv2d_50 (Conv2D)	(None, 12, 12, 384)	885120
conv2d_51 (Conv2D)	(None, 10, 10, 384)	1327488
conv2d_52 (Conv2D)	(None, 8, 8, 256)	884992
max_pooling2d_32 (MaxPooling	(None, 2, 2, 256)	0
flatten_5 (Flatten)	(None, 1024)	0
dense_13 (Dense)	(None, 4096)	4198400
dense_14 (Dense)	(None, 4096)	16781312
dense_15 (Dense)	(None, 2)	8194

...
Total params: 24,714,370
Trainable params: 24,714,370
Non-trainable params: 0

Fonte: Elaborado pelo autor.

A primeira camada da rede usa 96 filtros com kernels de tamanho 7x7, sem nenhum

stride, seguido por uma ativação ReLU, um pooling máximo de tamaho 3x3 e uma normalização de contraste local.

Em seguida, temos mais uma camada da rede, porém agora usaremos 256 filtros com kernels de tamanho 3x3, com mais uma camada de pooling e normalizado.

A terceira e a quarta camadas de convolução possuem cada uma 384 filtros e kernels de tamanho 3x3 com ativação ReLU.

A quinta camada de convolução tem 256 filtros e um kernel de 3x3 seguida de outra camada de pooling. A sexta e sétima camadas abrigam 4096 neurônios densos cada.

Finalmente, uma última camada densa de 2 neurônios representando as duas saídas da rede.

Os modelos utilizados nas Tabelas 6, 7 e 8, foram escolhidos em função de sua simplicidade estrutural para aplicação em uma máquina local, reduzindo os tempos de treinamento. Para simplificar mais ainda a rede, reduziram-se alguns parâmetros como o número de camadas convolucionais e de pooling, por exemplo.

Após cada etapa de criação dos modelos, cada um deles é usado para treinar os dados do banco de imagens e um arquivo .h5 (modelo keras) é criado.

Para avaliar a qualidade de cada um dos modelos, assim como estabelecer um parâmetro para fins de comparação entre eles, foram registradas alguma métricas como:

- Precisão - número de previsões corretas que o modelo fez;
- Perda - medida de o distante do acerto uma previsão foi feita;
- Matriz de confusão - tabela que indica os acertos e erros de um dado modelo;
- Acurácia - medida de performance geral do modelo baseado no número de acertos.

A pesquisa é finalizada com um último teste de classificação para verificar se o modelo está classificando as imagens da forma esperada.