UNIVERSIDADE FEDERAL DE JUIZ DE FORA
FACULDADE DE ENGENHARIA / INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM ENGENHARIA COMPUTACIONAL

Brian Luís Coimbra Maia

Classificação de imagens de bovinos utilizando redes neurais convolucionais

Juiz de Fora
2023

Brian Luís Coimbra Maia

Classificação de imagens de bovinos utilizando redes neurais convolucionais

Monografia apresentada ao Curso de Graduação em Engenharia Computacional da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Bacharel em Engenharia Computacional.

Orientador: Prof. Dr. Luiz Maurílio da Silva Maciel

Juiz de Fora

2023

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

C. Maia, Brian Luís.
Classificação de imagens de bovinos utilizando redes neurais convolucionais / Brian Luís Coimbra Maia. – 2023.
56 f. : il.

Orientador: Luiz Maurílio da Silva Maciel
Trabalho de Conclusão de Curso – Universidade Federal de Juiz de Fora, Faculdade de Engenharia / Instituto de Ciências Exatas. Bacharelado em Engenharia Computacional, 2023.

1. Visão Computacional. 2. Redes neurais convolucionais. 3. Aprendizado profundo. 4. Pecuária de precisão. I. Maciel, Luiz Maurílio, orient. II. Título.

Brian Luís Coimbra Maia

Classificação de imagens de bovinos utilizando redes neurais convolucionais

Monografia apresentada ao Curso de Graduação em Engenharia Computacional da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Bacharel em Engenharia Computacional.

BANCA EXAMINADORA

Prof. Dr. Luiz Maurílio da Silva Maciel - Orientador
Universidade Federal de Juiz de Fora

Dr. Bruno Campos de Carvalho
Embrapa Gado de Leite

Prof. Dr. Marcelo Bernardes Vieira
Universidade Federal de Juiz de Fora

Prof. Dr. Saulo Moraes Villela
Universidade Federal de Juiz de Fora

Dedico este trabalho aos meus pais e ao meu irmão, pelo incentivo aos estudos, amor e apoio em cada momento da minha trajetória escolar e acadêmica.

AGRADECIMENTOS

Agradeço aos meus pais, Irineu e Janete, que nunca mediram esforços para me apoiar em cada etapa da minha vida e sempre estiveram próximos nos momentos de alegria e preocupação. Ao meu irmão, pelo companheirismo e pelo exemplo que sempre procurei seguir. Aos meus avós, José Irineu e Vera, pelo suporte e por tanto carinho.

Ao professor orientador, Luiz Maurílio, por me abrir as portas para vários projetos da Universidade Federal de Juiz de Fora e pelo apoio e compreensão em diversos momentos das orientações. A todos/as professores/as e servidores/as que foram fundamentais na minha caminhada no ensino médio e ensino superior. Em especial, à toda a equipe do Arquivo Central da UFJF pelo acolhimento e conhecimentos passados.

Às estimadas amizades de Mathews Edwirds, Gabriel Rezende e Thaís Marins pelo companheirismo nas disciplinas da graduação e por todas as realizações alcançadas através dos projetos de pesquisa.

Aos meus estimados amigos Cristiano Nascimento, Matheus Estevão, Bruno Brito, Luiz Henrique de Souza, Matheus Casarim e William Mendes pelo companheirismo nas disciplinas da graduação.

À Universidade Federal de Juiz de Fora por proporcionar um ensino público de qualidade e uma gama de experiências extremamente enriquecedoras para toda a comunidade acadêmica.

À Embrapa Gado de Leite pela oportunidade de participar do projeto *Happy Cow ID* e ao supervisor do projeto, Bruno Campos de Carvalho, pelo apoio.

"Os limites da minha linguagem são os limites do meu mundo" – (Ludwig Wittgenstein)

RESUMO

As atividades pecuárias contam cada vez mais com tecnologias da informação que podem contribuir em diversas etapas da cadeia produtiva do setor. Em particular, muitos problemas de Visão Computacional voltados para o setor pecuário vêm sendo abordados através de técnicas de aprendizado profundo e os resultados atingidos têm sido satisfatórios, demonstrando o potencial que tais técnicas possuem para beneficiar essas atividades. Além disso, abordagens desse tipo podem ser muito vantajosas, já que se tratam de aplicações não invasivas e que podem gerar resultados em tempo real. Dessa forma, este trabalho propõe o desenvolvimento de três modelos de Redes Neurais Convolucionais (*Convolutional Neural Networks* - CNNs) para classificação de imagens de faces de bovinos de acordo com: categoria, pose e raça. Foi realizada uma série de experimentos, dos quais os melhores resultados são reportados neste trabalho. A partir de uma série de experimentos e avaliações, foi possível atingir resultados satisfatórios para os casos abordados. Mais especificamente, os modelos classificadores de pose e categoria atingiram acurácias superiores a 90%, enquanto os modelos gerados para o problema de raça atingiram resultados mais desfavoráveis (em torno de 83%), o que demonstrou que se trata de um problema mais desafiador. Ao analisar todos os resultados, verificou-se o potencial que tais abordagens possuem para atingir resultados ainda melhores em trabalhos futuros.

ABSTRACT

The livestock activities increasingly rely on information technologies that can contribute in several steps of the productive chain of the sector. In particular, many problems of Computer Vision focused on the livestock sector have been addressed through deep learning techniques and the results achieved have been satisfactory, demonstrating the potential that such techniques have to benefit these activities. Furthermore, approaches like this can be very advantageous, since they are non-invasive applications that can generate results in real time. Thus, this work proposes the development of three types of image classification models generated from Convolutional Neural Networks (CNNs): category, pose, and breed. A series of experiments were performed, of which the best results are reported in this paper. From a sequence of experiments and evaluations, it was possible to achieve satisfactory results for the cases addressed. More specifically, the pose and category classifier models achieved accuracies greater than 90%, while the models generated for the breed problem achieved worse results (around 83%), which demonstrated that it is a more challenging problem. Analyzing all the results, we verified the potential that such approaches have to achieve even better results in future works.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CNN	*Convolutional Neural Network*
LASSO	*Least Absolute Shrinkage and Selection Operator*
MLP	*Multilayer Perceptron*
MSE	*Mean Squared Error*
ReLU	*Rectified Linear Unit*
SVM	*Support Vector Machine*

SUMÁRIO

1 INTRODUÇÃO

As tecnologias da informação estão cada vez mais presentes nas atividades rurais e podem causar um impacto positivo em diversos segmentos desse setor. Nesse contexto, a pecuária de precisão é um conceito fundamental para entender a importância e as implicações do uso dessas tecnologias nas atividades que envolvem a pecuária. De acordo com a definição de Wathes et al. (2008), a pecuária de precisão é a gestão da produção animal utilizando os princípios e a tecnologia da engenharia de processos. Bernardi et al. (2017) afirmam que o conhecimento do comportamento animal e a utilização de técnicas de manejo racional podem aumentar a produtividade do negócio e o bem-estar dos animais, além de melhorar a qualidade do produto final. No entanto, apesar de várias publicações destacarem os benefícios técnicos da pecuária de precisão, ela ainda é olhada com desconfiança por algumas pessoas, principalmente por causa da carência de informações confiáveis e o desconhecimento de estudos de viabilidade dos investimentos necessários para a sua utilização (FERREIRA; SIQUEIRA; PEREIRA, 2015). Dessa forma, desenvolver métodos que englobam o campo da pecuária de precisão e que sejam viáveis para a maioria dos produtores é algo extremamente benéfico.

A Visão Computacional é uma área do conhecimento que procura emular a visão humana, isto é, engloba processos que possuem como entrada uma imagem e a sua saída é uma interpretação dessa imagem como um todo, ou parcialmente (MARENGONI; STRINGHINI, 2009). Como exemplo de aplicação, é possível citar o uso da Visão Computacional na medicina, através da extração de informações nas imagens de radiografias e ultrassonografias para fazer diagnósticos (PICCIALLI et al., 2021), e no desenvolvimento de carros autônomos, que são capazes de reconhecer com assertividade as sinalizações nas vias em tempo real (GRIGORESCU et al., 2020).

Atualmente, muitos problemas de Visão Computacional vêm sendo abordados através de aprendizado profundo (*deep learning*), uma subárea do aprendizado de máquina (*machine learning*), que procura simular o comportamento do cérebro humano em tarefas de reconhecimento visual, análise de texto, reconhecimento de fala, processamento de linguagem natural, entre outras. De modo geral, os algoritmos de aprendizado profundo têm como objetivo produzir representações hierárquicas de alto nível dos dados de entrada através de camadas de processamento sequencial em uma rede neural artificial (BEZERRA, 2016).

Na literatura, existem numerosos exemplos de métodos que utilizam aprendizado profundo e atingem resultados extremamente satisfatórios em tarefas de Visão Computacional associadas a problemas da pecuária. Weber et al. (2020) usaram redes neurais convolucionais (*Convolutional Neural Networks* — CNNs) para a tarefa de reconhecimento individual de bovinos da raça Pantaneiro e alcançaram uma acurácia superior a 99% em

todos os experimentos realizados. Hu et al. (2020) propuseram um método que envolve a utilização de CNNs e máquinas de vetores suporte (*Support Vector Machine* — SVM) para reconhecer bovinos individualmente a partir de imagens laterais de 93 animais diferentes e, com essas condições, atingem uma acurácia de 98,36%. Outro exemplo que pode ser citado é o trabalho de Tian et al. (2019) que utiliza técnicas de aprendizado profundo para desenvolver um método que faz a contagem automática de suínos em uma imagem. Através dos resultados experimentais, é possível notar que o método tem uma boa precisão, já que foi reportado um valor de $1,67$ para o erro médio absoluto por imagem, um valor que representa a diferença média entre o resultado previsto pelo algoritmo e o resultado real. Além disso, a velocidade de detecção dos animais foi de 42 ms por imagem, o que também é muito satisfatório para esse tipo de problema. Visto que o uso de métodos de aprendizado profundo tem-se demonstrado viável e vantajoso em diversos estudos na área da pecuária, é possível afirmar que tais métodos podem beneficiar consideravelmente a administração de uma fazenda e até mesmo a saúde de animais de uma determinada espécie.

Dito isso, este trabalho se propõe a utilizar métodos de aprendizado profundo para fazer a classificação de imagens de bovinos. Dessa forma, será apresentado o processo de desenvolvimento de três modelos de classificação: categoria, pose e raça. A classificação da categoria é usada para informar se um dado animal pertence ao grupo das vacas ou dos bezerros. A classificação de pose tem o objetivo de identificar se uma determinada imagem contém a face frontal ou lateral do animal. Por fim, a classificação de raça pode ser usada para retornar a raça à qual um bovino encontrado em uma imagem pertence. Neste contexto, o problema deste trabalho compreende a geração de modelos que consigam classificar satisfatoriamente imagens de bovinos de acordo com esses três atributos.

Classificar corretamente os três segmentos citados no parágrafo anterior pode ser muito vantajoso em diferentes situações. Para um sistema de reconhecimento facial em uma fazenda, pode ser interessante contar com algumas informações extraídas dos modelos que este trabalho apresenta. Por exemplo, em uma determinada situação, pode ser necessário fazer o reconhecimento facial apenas de bezerros da raça holandesa. Nesse caso, antes de reconhecer a face do animal, será necessário verificar se ele se encaixa na categoria e raça desejadas. Um outro exemplo em que um desses modelos de classificação pode ser utilizado é para o caso em que é necessário avaliar, através de imagens, o estado de saúde dos bovinos de uma determinada fazenda. Dessa forma, o modelo citado anteriormente seria importante para identificar automaticamente o grupo alvo e, assim, permitir que apenas os animais da pose desejada sejam avaliados. Isto pode ser vantajoso, visto que determinadas características associadas à dor podem ser melhor identificadas através de uma imagem frontal e outras em uma imagem lateral.

Para abordar os problemas tratados neste trabalho, o primeiro passo consistiu em construir e rotular um conjunto de dados, propor modelos de redes neurais para a

classificação de imagens e, a partir de uma série de experimentos, avaliar e fazer ajustes nesses modelos. Dessa forma, o conjunto de dados criado e os modelos de classificação gerados compreendem as principais contribuições que podem ser aproveitadas em outros trabalhos.

1.1 OBJETIVOS

Este trabalho tem como objetivo principal o desenvolvimento e avaliação de três modelos de redes neurais convolucionais que sejam capazes de classificar a categoria, pose e raça de bovinos a partir de uma base de imagens. Os objetivos específicos são:

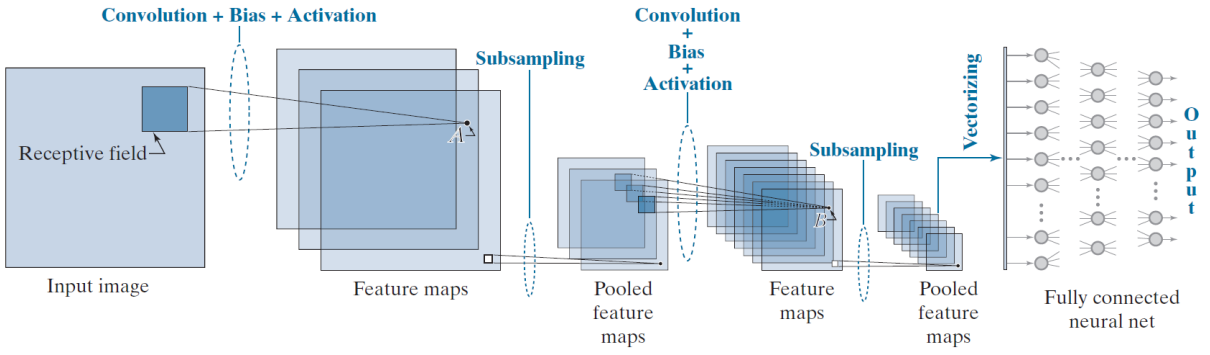
- Estudar o funcionamento de redes neurais convolucionais para classificação de imagens;
- Organizar o conjunto de dados (*dataset*) de imagens usado para treinar as redes;
- Definir os parâmetros das redes através de resultados experimentais;
- Avaliar o desempenho das redes a partir de determinadas métricas.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 REDES NEURAIS CONVOLUCIONAIS

As CNNs são uma classe das redes neurais que aceitam imagens como entrada e são ideais para tarefas de classificação e aprendizado automático (GONZALEZ; WOODS, 2017). A ideia da utilização das redes neurais convolucionais foi popularizada a partir do trabalho de LeCun et al. (1998), que introduziu a rede LeNet e aplicou ao problema de reconhecimento de dígitos. Uma diferença entre a arquitetura das CNNs e as arquiteturas das outras redes neurais é que as entradas das redes convolucionais são matrizes 2D (imagens), enquanto as entradas das outras redes são vetores unidimensionais. Outra diferença é que, ao invés de ligar todas as unidades de uma camada a todas as unidades de uma camada anterior, as CNNs organizam cada camada em mapas de características, que podem ser entendidos como canais paralelos. No entanto, os cálculos realizados pelas CNNs e os outros tipos de redes são semelhantes, pois forma-se uma soma de produtos, adiciona-se um valor de viés (*bias*), o resultado passa por uma função de ativação e, por fim, o valor de ativação torna-se uma única entrada para a camada seguinte (GONZALEZ; WOODS, 2017). A rede neural convolucional da Figura 1 contém os elementos básicos da arquitetura LeNet. As principais camadas que compõem uma rede convolucional são as camadas convolucionais, camadas de agrupamento (ou *pooling*) e camadas totalmente conectadas. Ao longo desta seção, o funcionamento e as características das redes convolucionais são melhor detalhados.

Figura 1: Arquitetura da rede LeNet.



Fonte: Gonzalez e Woods (2017).

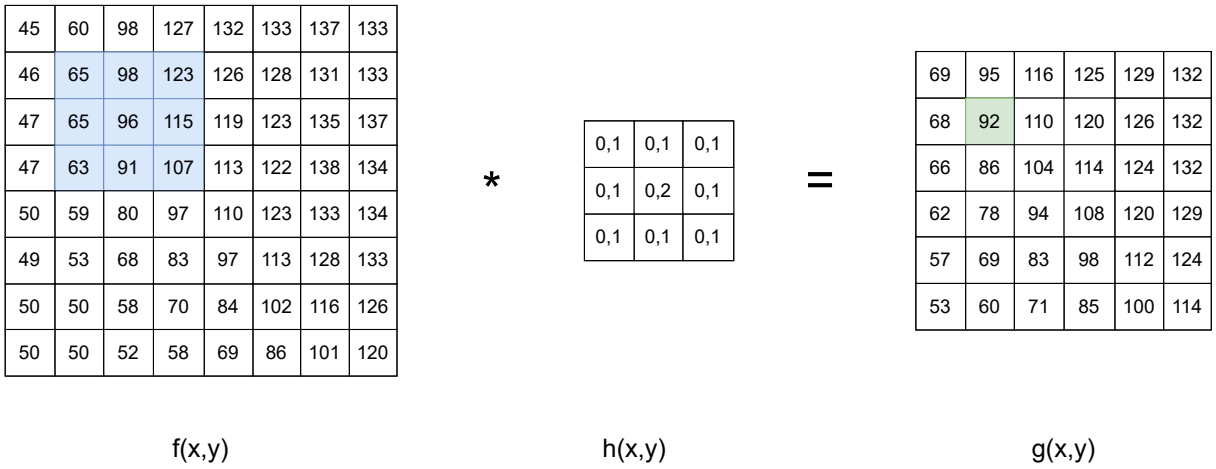
As redes neurais convolucionais recebem esse nome por causa da operação de convolução que pode ser definida pela equação:

$$g(x, y) = \sum_{k,l} f(x + k, y + l)h(k, l), \tag{2.1}$$

onde $f(x, y)$ representa a imagem original, $h(k, l)$ o núcleo (*kernel*) da convolução e $g(x, y)$ a imagem resultante.

Considerando a estrutura matricial de uma imagem, a vizinhança de um *pixel* corresponde aos *pixels* que estão em posições próximas a ele em termos de linhas e colunas na matriz. Esse tipo de relação de vizinhança é importante para a compreensão de como ocorre a filtragem no domínio do espaço. A vizinhança é definida na operação de filtragem por meio do *kernel*, normalmente representado por uma matriz. As entradas no *kernel* $h(k, l)$ são frequentemente chamadas de coeficientes de filtragem. Dessa forma, para cada localização, a convolução calcula uma soma de produtos entre os *pixels* na vizinhança e um conjunto de pesos do *kernel*. Esta operação é realizada em todos os locais espaciais da imagem de entrada e o resultado em cada local (x, y) é um valor escalar (GONZALEZ; WOODS, 2017). A Figura 2 mostra um exemplo de convolução entre um *kernel* de tamanho 3×3 e uma imagem de tamanho 8×8.

Figura 2: Exemplo de operação de convolução entre os *pixels* da imagem de entrada $f(x, y)$ e o *kernel* $h(x, y)$ para formar a imagem de saída $g(x, y)$. Os *pixels* azuis claros foram utilizados juntamente com o conjunto de pesos de $h(x, y)$ para calcular o *pixel* de destino destacado em verde claro.



Fonte: Adaptado de Szeliski (2010).

A cada operação de convolução, é adicionado um *bias* e depois o resultado é passado através de uma função de ativação para gerar um único valor. Após isso, este valor é alimentado para a localização correspondente (x, y) na entrada da camada seguinte. Quando repetido para todas as localizações da imagem de entrada, o processo gera uma nova imagem como entrada na próxima camada que é chamada de mapa de características (*feature map*) (GONZALEZ; WOODS, 2017). É importante destacar que, nas camadas convolucionais, é necessário definir o número de filtros utilizados. Dessa forma, uma camada convolucional que utiliza um número maior de filtros tende a extrair mais características da entrada.

Além do que já foi discutido, existem alguns outros parâmetros de uma camada convolucional que merecem atenção:

- *Stride*: define o deslocamento que será dado pelo *kernel* durante a convolução. De acordo com Gonzalez e Woods (2017), uma motivação importante para utilizar passos maiores que um é a redução de dados. Por exemplo, alterar o *stride* de um para dois reduz a resolução da imagem pela metade em cada dimensão espacial, o que gera uma redução de três quartos na quantidade de dados por imagem. É possível também definir valores diferentes para a direção horizontal e vertical, mas normalmente se utiliza o mesmo valor em ambas as direções.

- *Padding*: após a convolução, pode ocorrer uma redução na imagem dependendo do tamanho do *kernel*, pois esta operação não pode ser feita nos pontos próximos à borda da imagem. Nesse caso, a borda pode ser estendida através de alguma alternativa, como o preenchimento de zeros ou a replicação de *pixels* (SZELISKI, 2010). Este procedimento é conhecido como *padding* e pode ser usado para controlar a redução das imagens ao longo das camadas convolucionais da rede. Dessa forma, ao utilizar um determinado valor de *padding*, é possível fazer com que as imagens não diminuam mais rápido do que é necessário para o aprendizado.

A Equação (2.2) mostra como o tamanho da saída em uma camada convolucional pode ser calculado:

$$O = \frac{(W - K + 2P)}{S} + 1, \tag{2.2}$$

em que O representa o tamanho da saída, W o tamanho da imagem, K o tamanho do *kernel*, S o tamanho do *stride* e P o tamanho do *padding*. Por exemplo, para anular a redução da imagem após uma convolução com *kernel* de 5 e *stride* de 1, basta utilizar um *padding* de tamanho 2.

É necessário destacar também a importância da utilização de funções de ativação em CNNs. Estas funções decidem se um neurônio (ou nó) deve ser ativado ou não. De outro modo, elas decidem se a informação que o neurônio está recebendo é relevante ou pode ser descartada. As primeiras redes neurais da literatura utilizaram funções de ativação sigmoidais semelhantes às utilizadas na regressão logística. Já nas redes convolucionais mais recentes, é frequente a utilização da função de ativação unidade linear retificada (*Rectified Linear Unit* — ReLU) ou variantes dela. A ReLU é definida como:

$$h(y) = max(0, y), \tag{2.3}$$

e a sua derivada é dada por:

$$h'(y) = \begin{cases} 1, & \text{se } y \geq 0. \\ 0, & \text{caso contrário.} \end{cases} \tag{2.4}$$

Observando as Equações 2.3 e 2.4, é possível notar que, caso a entrada na função seja negativa, ela será transformada em zero e o neurônio não será ativado, o que pode ser

vantajoso, já que esse processo impede que todos os neurônios sejam ativados ao mesmo tempo, o que torna a rede mais eficiente. Dessa forma, uma das vantagens da ReLU é a sua rapidez para efetuar os cálculos. De acordo com os resultados experimentais, a ReLU tende a superar outras funções de ativação como a sigmoide e a tangente hiperbólica (GONZALEZ; WOODS, 2017). Um problema já identificado na ReLU é que algumas unidades podem simplesmente "morrer" durante o treinamento através de uma ocorrência que faz com que o neurônio passe a produzir apenas zeros. Com a fixação da taxa de aprendizagem, este problema se torna menos frequente. Além disso, em redes utilizadas para a classificação, é comum utilizar a função de ativação *softmax* na camada final para converter ativações com valor real em probabilidades de classe (SZELISKI, 2010). Dessa forma, essa função faz com que valores que saem de uma rede neural sejam transformados em probabilidades que somam 1 e, com isso, é possível obter a probabilidade de uma entrada pertencer a uma determinada classe.

Após o processo de convolução e ativação, acontece o agrupamento, também conhecida como *pooling*, que tem a vantagem de reduzir o volume de dados em processamento, o que pode ser muito útil quando se trabalha com uma grande quantidade de imagens. Uma forma de entender o *pooling* é pensar que ele busca produzir mapas de características agrupados, ou seja, mapas de características que tiveram a resolução espacial reduzida. Dessa forma, a operação é feita a partir da subdivisão de um mapa de características em um conjunto de pequenas regiões (tipicamente de tamanho 2×2) e substituindo todos os elementos de uma tal região por um único valor (GONZALEZ; WOODS, 2017). É preciso destacar que o mapa de características é subdividido em regiões que são adjacentes, então não há sobreposição entre elas. Para substituir todos os elementos de uma tal região por um único valor, é possível utilizar diversos métodos, sendo que os mais comuns são: (1) *max-pooling*, que substitui os valores de uma região pelo valor máximo dos seus elementos; (2) *min-pooling*, que substitui os valores de uma região pelo valor mínimo dos seus elementos; (3) *average pooling*, em que o resultado é dado pela média dos valores da região; e (4) L^2 *pooling*, em que o valor resultante é a raiz quadrada da soma dos quadrados dos valores da região.

Na maioria dos casos, o objetivo em uma CNN é utilizar as características aprendidas para fazer uma classificação. Na rede da Figura 1, a classificação é feita a partir da inserção dos valores da última camada de *pooling* em uma rede neural totalmente conectada, em que todos os neurônios de uma camada se conectam a todos os neurônios da camada seguinte. É preciso destacar que as saídas de uma CNN podem ter diferentes dimensões. Para o caso das redes neurais abordadas no presente trabalho, as saídas são matrizes 2D e a entrada para as redes totalmente conectadas são vetores. Nesse caso, é necessário linearizar os mapas de características agrupados na camada anterior à camada totalmente conectada, ou seja, transformá-los em vetores. Após isso, todos os vetores gerados são concatenados para formar um único vetor que será propagado através da rede totalmente

conectada. Para um problema de classificação, o número de saídas da rede totalmente conectada é igual ao número de classes existentes no problema (GONZALEZ; WOODS, 2017). Dessa forma, cada valor de saída representa a probabilidade da entrada pertencer a uma determinada classe.

2.2 APRENDIZADO EM REDES NEURAIS

Para introduzir o funcionamento de uma rede neural, é importante discutir brevemente sobre os *perceptrons* (ROSENBLATT, 1958), elementos computacionais não utilizados nas arquiteturas de redes neurais atuais, mas muito semelhantes aos neurônios artificiais, que são unidades computacionais base para as redes neurais. Nesse ponto, é importante destacar que a estrutura dos neurônios artificiais é composta por modelos matemáticos inspirados nos neurônios biológicos.

O *perceptron* é um dos modelos de rede neural artificial mais antigos e lida com um único neurônio que recebe sinais de entrada a partir de dados de treinamento que são ponderados e combinados na seguinte equação linear:

$$z = \sum_{i=1}^{n} w_i x_i + w_{n+1}, \tag{2.5}$$

em que z é denominado de potencial de ativação, w é um peso da rede, x é uma entrada, i é o índice de um peso ou uma entrada e w_{n+1} é o *bias*, que é um peso especial que não é multiplicado por nenhuma entrada, ou seja, é sempre multiplicado pela constante 1. Normalmente, o *bias* também é aprendido pela rede (GONZALEZ; WOODS, 2017). É possível também representar o padrão de entrada e os pesos como vetores coluna $\mathbf{x}$ e $\mathbf{w}$ da seguinte forma:

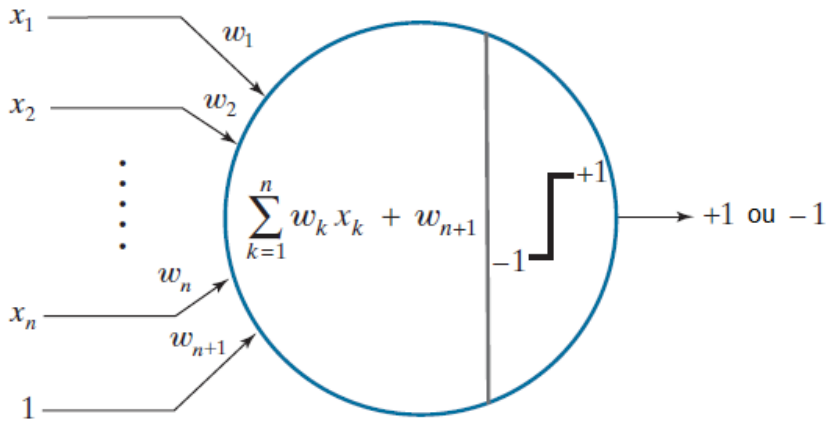
$$z = \mathbf{w}^T \mathbf{x} + w_{n+1}. \tag{2.6}$$

A Figura 3 mostra as operações realizadas pelo *perceptron*. Nesse caso, as entradas no neurônio são representadas pelo vetor $\mathbf{x} = [x_1, x_2, x_3, \ldots, x_n]$ que, ao chegarem no neurônio, são multiplicadas pelos respectivos pesos sinápticos do vetor $\mathbf{w} = [w_1, w_2, w_3, \ldots, w_n]$. Por fim, o resultado calculado na Equação (2.5) é transformado em um valor de saída ao passar por uma função de ativação linear da seguinte forma:

$$a = \begin{cases} +1, & \text{se } z > 0 \\ -1, & \text{se } z \leq 0. \end{cases} \tag{2.7}$$

Um aspecto importante sobre o algoritmo *perceptron* é que o processo de descida de gradiente estocástico é usado para avaliar e atualizar os pesos a cada iteração. Dessa forma, uma amostra de treinamento é apresentada ao *perceptron* e a classificação da sua saída é observada. Se a saída estiver incorreta, os pesos são ajustados.

Figura 3: Operações realizadas no algoritmo *perceptron*.



Fonte: Adaptado de Gonzalez e Woods (2017).

O objetivo é encontrar o vetor de pesos aumentado $\mathbf{w}$ que minimiza o erro quadrático médio (*Mean Squared Error* — MSE) entre as respostas desejadas e as respostas reais do *perceptron*. A função usada para este fim tem a seguinte forma:

$$E(\mathbf{w}) = \frac{1}{2}(r - \mathbf{w}^T\mathbf{x})^2, \tag{2.8}$$

em que E é a medida de erro, $\mathbf{w}$ é o vetor de pesos procurado, $\mathbf{x}$ é um padrão do conjunto de treinamento e r é a resposta que se busca que o *perceptron* tenha para esse padrão durante a fase de treinamento. No caso, tanto $\mathbf{w}$ como $\mathbf{x}$ são vetores aumentados, em que é acrescentado o valor 1 ao final do vetor $\mathbf{x}$ e o valor do *bias* w_{n+1} ao final do vetor $\mathbf{w}$. (GONZALEZ; WOODS, 2017). Essa função em que se deseja minimizar com relação aos pesos da rede é comumente chamada de função de perda (*loss function*).

O mínimo de $E(\mathbf{w})$ é encontrado através do algoritmo de descida de gradiente iterativo citado anteriormente, cuja forma é:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \alpha \left[\frac{\delta E(\mathbf{w})}{\delta \mathbf{w}}\right]_{\mathbf{w}=\mathbf{w}(k)}, \tag{2.9}$$

em que $\mathbf{w}(k)$ corresponde ao vetor de pesos na iteração k e α é um valor maior que zero e representa a taxa de aprendizagem. Existem muitas abordagens que procuram encontrar taxas ótimas de aprendizagem, mas se trata de um parâmetro que depende do problema e que, por isso, envolve a experimentação. Uma abordagem indicada é iniciar o treinamento com uma taxa de aprendizagem pequena (por exemplo, 0,01) e depois fazer experimentos com o conjunto de treinamento para definir um valor em uma determinada aplicação. Essa taxa é usada apenas durante o treinamento e, por isso, não tem qualquer efeito no desempenho operacional pós-treino (GONZALEZ; WOODS, 2017).

É preciso destacar também que uma taxa de aprendizagem muito pequena pode levar a uma convergência lenta. Por outro lado, a escolha de uma taxa de aprendizagem

muito grande pode causar instabilidade ou grandes oscilações de ambos os lados do mínimo (GONZALEZ; WOODS, 2017). Uma alternativa que pode ser usada em alguns caso é o programador da taxa de aprendizagem (*learning rate scheduler*), que é responsável por variar a taxa ao longo do processo de treinamento.

Como a função de perda é dada de forma analítica e é diferenciável, é possível expressar a Equação (2.9) de forma que não seja necessário fazer explicitamente o cálculo do gradiente a cada passo. A derivada parcial de $E(\mathbf{w})$ com respeito a $\mathbf{w}$ é dada por:

$$\frac{\delta E(\mathbf{w})}{\delta \mathbf{w}} = -(r - \mathbf{w}^T\mathbf{x})\mathbf{x}. \tag{2.10}$$

Substituindo este resultado na Equação (2.9), encontra-se:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \alpha \left[r(k) - \mathbf{w}^T(k)\mathbf{x}(k)\right] \mathbf{x}(k),$$

em que os termos são conhecidos ou calculados com facilidade e o vetor de pesos inicial $\mathbf{w}(1)$ é arbitrário (GONZALEZ; WOODS, 2017).

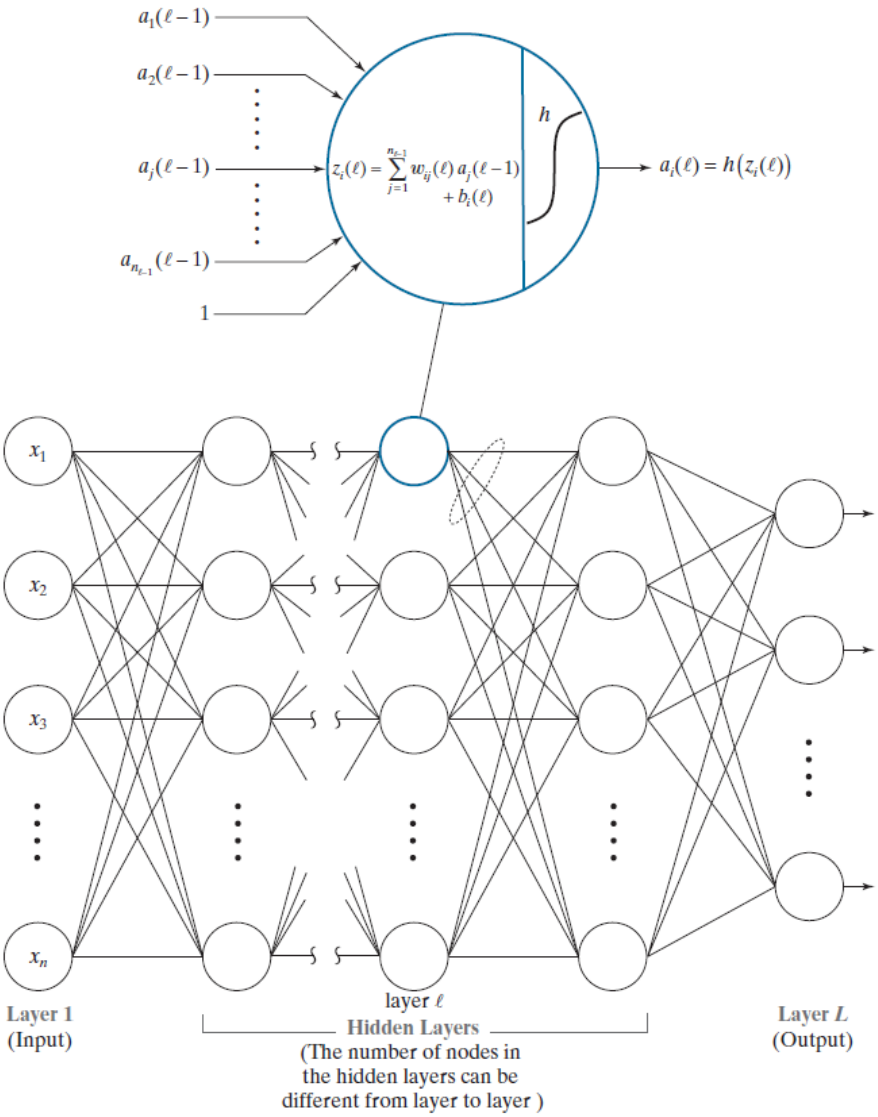
O *perceptron* é o tipo mais simples de rede neural artificial e pode ser considerado um algoritmo de classificação para problemas com duas classes, além de fornecer a base para o desenvolvimento de redes maiores. Dessa forma, observou-se que é possível combinar diferentes neurônios em uma estrutura de camadas, cada uma com uma quantidade diferente de neurônios, formando uma rede neural denominada *Perceptron* Multicamadas (*Multilayer Perceptron* — MLP).

Como é possível observar na Figura 4, a rede MLP possui uma camada de entrada, uma ou mais camadas ocultas com uma quantidade indeterminada de neurônios e uma camada de saída. A expressão "camada oculta" existe porque não é possível prever a saída esperada nas camadas intermediárias.

O *Perceptron* Multicamadas faz parte da classe das redes neurais do tipo *feedforward*, em que cada camada se conecta à próxima camada, mas não existe um caminho de volta. Nesse tipo de rede, todas as conexões possuem a mesma direção, partindo da camada de entrada até a camada de saída. As múltiplas linhas existentes nas saídas dos neurônios da Figura 4 indicam que a saída de cada neurônio está ligada à entrada de todos os neurônios da camada seguinte, o que forma uma rede que é conhecida como rede totalmente conectada (*fully connected network*) (GONZALEZ; WOODS, 2017). Outra característica que pode ser observada na Figura 4 é que uma camada da rede MLP é o conjunto de neurônios em uma coluna da rede, sendo que a estrutura de cada um desses neurônios é mostrada na Figura 5.

É possível organizar a rede MLP em várias camadas, fazendo com que ela se torne mais profunda e capaz de aprender relações mais complexas, já que passa a modelar funções não linearmente dependentes. Desse modo, esse tipo de rede é muito utilizada em trabalhos de *deep learning* (SOUSA et al., 2020).

Figura 4: Entrada, processamento de dados e sentido de saídas em uma rede MLP. Na parte superior, a figura exibe uma representação de um neurônio artificial. Além disso, sob as camadas ocultas, há uma observação sobre o fato que o número de nós dessas camadas pode ser diferente de camada para camada.



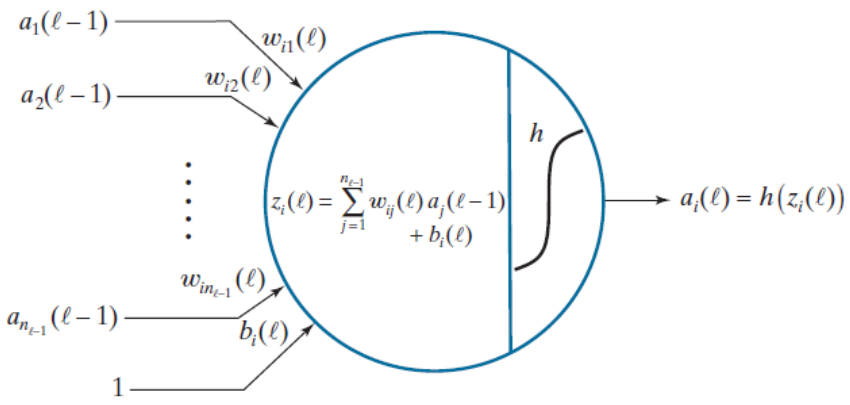
Fonte: Gonzalez e Woods (2017).

O treinamento de uma rede MLP está envolvido no contexto do aprendizado de máquina supervisionado, no qual se usa um conjunto de dados rotulado em que se conhece a saída correta. Nesse caso, para que a rede possa "aprender", é necessário que o conjunto de treinamento contenha amostras representativas de cada classe do problema (VIEIRA; BAUCHSPIESS, 1999).

Outra modificação em relação ao *perceptron* é que normalmente a rede MLP não utiliza uma função de ativação na forma de degrau e passa a usar uma função de ativação do tipo sigmoide:

$$h(z) = \frac{1}{1 + e^{-z}}, \tag{2.11}$$

Figura 5: Operações realizadas em um modelo de um neurônio artificial na rede MLP. O l representa uma camada específica na rede.



Fonte: Gonzalez e Woods (2017).

em que z representa o resultado do cálculo efetuado pelo neurônio. Além da sigmoide, outras funções de ativação podem ser utilizadas em uma rede MLP.

Outra diferença em relação ao *perceptron* é que na rede multicamadas, o erro obtido na saída da rede é transferido para as camadas intermediárias para que seja feito um ajuste dos pesos, que é o processo conhecido como retropropagação (*backpropagation*). Esse processo será detalhado a seguir. Antes disso, é necessário apresentar outro conceito importante, que é o de *forward pass*.

Nesse caso, as saídas da primeira camada são os componentes do vetor de entrada:

$$a_j(1) = x_j \qquad j = 1, 2, \ldots, n_1, \tag{2.12}$$

em que $n_1 = n$ é a dimensionalidade de $\mathbf{x}$.

Como é mostrado na Figura 5, o cálculo realizado pelo neurônio i na camada l é dado por:

$$z_i(l) = \sum_{j=1}^{n_{l-1}} w_{ij}(l) a_j(l-1) + b_i(l), \tag{2.13}$$

para $i = 1, 2, \ldots, n_l$ e $l = 2, \ldots, L$. A quantidade $z_i(l)$ é chamada de entrada total da rede para o neurônio i da camada l (GONZALEZ; WOODS, 2017).

A saída (valor de ativação) do neurônio i na camada l é dada por:

$$a_i(l) = h(z_i(l)) \qquad i = 1, 2, \ldots, n_l, \tag{2.14}$$

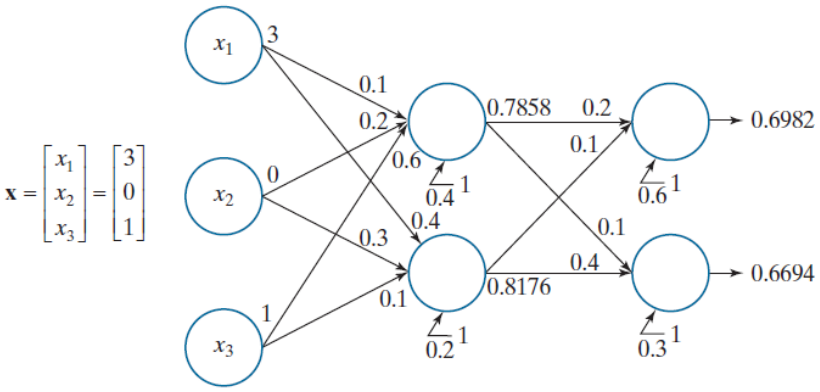
em que h é uma função de ativação. Já o valor do neurônio i na camada de saída da rede é:

$$a_i(L) = h(z_i(L)) \qquad i = 1, 2, \ldots, n_L. \tag{2.15}$$

As Equações 2.12 até 2.15 mostram todas as operações necessárias para mapear a entrada de uma rede *feedforward* totalmente conectada à sua saída (GONZALEZ; WOODS,

2017). Para compreender melhor o *forward pass*, é útil apresentar um exemplo numérico simples desse processo. A Figura 6 mostra uma rede neural totalmente conectada de três camadas (uma camada de entrada, uma camada oculta e uma camada de saída). Além disso, essa rede aceita três entradas, tem duas saídas e utiliza a função de ativação sigmoide (Equação (2.11)).

Figura 6: Exemplo de *forward pass*.



Fonte: Gonzalez e Woods (2017).

De acordo com Gonzalez e Woods (2017), uma rede neural pode ser definida completamente pelos seus pesos, *biases* e função de ativação. Dessa forma, torna-se necessário entender as equações da retropropagação, que é o processo de escolha para encontrar o valor dos pesos e dos *biases* em uma rede multicamadas.

O processo de retropropagação envolve quatro etapas básicas:

1. Atribuição de valores aleatórios aos pesos da rede;
2. Passagem para a frente (*forward pass*) através da rede para classificar todos os padrões do conjunto de treinamento e determinar o erro de classificação;
3. Passagem para trás (*backward pass*) que alimenta o os erros de volta, das saídas às entradas, para calcular as alterações necessárias e atualizar os pesos dos nós;
4. Atualização dos pesos e *biases* na rede.

Todas as etapas acima são repetidas até que o erro atinja um nível aceitável (GONZALEZ; WOODS, 2017). A seguir, serão apresentas as principais equações usadas durante o processo de retropropagação, em que a principal ferramenta matemática necessária para derivar essas equações é a regra da cadeia.

Toda a discussão envolvendo a retropropagação está relacionada com a busca de parâmetros na rede que minimizem a função de erro (perda). Dessa forma, considerando

que os valores de ativação do neurônio j na camada de saída são definidos por $a_j(L)$, é possível definir o erro desse neurônio da seguinte forma:

$$E_j = \frac{1}{2}\Big(r_j - a_j(L)\Big)^2, \tag{2.16}$$

para $j = 1, 2, \ldots, n_L$, em que r_j é a saída desejada para o neurônio j e $a_j(L)$ é a saída obtida pela rede para um dado padrão $\mathbf{x}$.

Com isso, o erro da saída da rede em relação a um único padrão de entrada $\mathbf{x}$ é a soma dos erros de todos os neurônios da camada de saída:

$$E = \sum_{j=1}^{n_L} E_j = \frac{1}{2}\sum_{j=1}^{n_L}\Big(r_j - a_j(L)\Big)^2 = \frac{1}{2}\|\mathbf{r} - \mathbf{a}(L)\|^2, \tag{2.17}$$

em que a expressão após a última igualdade (à direita) segue a definição da norma vetorial euclidiana. Assim, o erro total da saída da rede sobre todos os padrões de treinamento é dado como a soma dos erros dos padrões individuais (GONZALEZ; WOODS, 2017).

Como o objetivo é encontrar os pesos que minimizam o erro total, utiliza-se então o gradiente descendente. No entanto, Gonzalez e Woods (2017) destacam que, ao contrário do que ocorre no *perceptron*, não é possível calcular os gradientes dos pesos nos neurônios ocultos. A grande vantagem da retropropagação é que esse algoritmo garante que um resultado equivalente seja encontrado a partir da propagação do erro de saída em direção às primeiras camadas da rede.

Para ajustar todos os pesos na rede, é necessário saber como o E varia em relação a esses pesos. Como a retropropagação começa da saída e trabalha para trás a partir dessa saída, é preciso definir a variação do erro no neurônio j na camada l em relação a entrada z_j como sendo:

$$\delta_j(L) = \frac{\partial E}{\partial z_j(L)},$$

em que j é usado de forma genérica para significar qualquer neurônio da rede.

A partir disso, é possível expressar esta equação em termos da saída $a_j(L)$ usando a regra da cadeia:

$$\delta_j(L) = \frac{\partial E}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)}\frac{\partial a_j(L)}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)}\frac{\partial h\Big(z_j(L)\Big)}{\partial z_j(L)} = \frac{\partial E}{\partial a_j(L)}h'\Big(z_j(L)\Big),$$

em que a Equação (2.15) é usada para obter a expressão após a penúltima igualdade (GONZALEZ; WOODS, 2017).

Utilizando a Equação (2.16) como função de erro e a Equação (2.11) como função de ativação, é possível encontrar:

$$\delta_j(L) = h\Big(z_j(L)\Big)\Big[1 - h\Big(z_j(L)\Big)\Big]\left[a_j(L) - r_j\right].$$

Como a variação do erro $\delta_j(L)$ na camada L é conhecida, é possível calcular a variação do erro $\delta_j(l)$ em um neurônio j na uma camada l em função da variação na

camada seguinte $\delta_j(l+1)$ e, assim, essa variação será retropropagada até a segunda camada da rede. Dessa forma, é possível obter o resultado desejado utilizando novamente a regra da cadeia:

$$\begin{aligned}\partial_j(l) = \frac{\partial E}{\partial z_j(l)} &= \sum_i \frac{\partial E}{\partial z_i(l+1)} \frac{\delta z_i(l+1)}{\partial a_j(l)} \frac{\partial a_j(l)}{\partial z_j(l)} \\ &= \sum_i \delta_i(l+1) \frac{\partial z_i(l+1)}{\partial a_j(l)} h'\Big(z_j(l)\Big) \\ &= h'\Big(z_j(l)\Big) \sum_i w_{ij}(l+1)\delta_i(l+1),\end{aligned}$$

para $l = L-1, L-2, \ldots 2$.

O desenvolvimento acima mostra como começar com o erro na saída e evidencia como esse erro muda em função das entradas da rede para cada neurônio (GONZALEZ; WOODS, 2017). O passo seguinte é a obtenção da variação do erro em função dos pesos através de uma nova aplicação da regra da cadeia:

$$\frac{\partial E}{\partial w_{ij}(l)} = \frac{\partial E}{\partial z_i(l)} \frac{\partial z_i(l)}{\partial w_{ij}(l)} = \delta_i(l) \frac{\partial z_i(l)}{\partial w_{ij}(l)} = a_j(l-1)\delta_i(l).$$

Similarmente, a variação do erro em função do *bias* pode ser encontrada a partir de:

$$\frac{\partial E}{\partial b_i(l)} = \delta_i(l).$$

Tendo a taxa de variação de E em relação aos pesos e *biases* da rede, é possível partir para o último passo que é a atualização dos pesos da rede usando o gradiente descendente:

$$w_{ij}(l) = w_{ij}(l) - \alpha \frac{\partial E(l)}{\partial w_{ij}(l)} = w_{ij}(l) - \alpha\delta_i(l)a(l-1),$$

e a atualização dos *biases* é dada por:

$$b_i(l) = b_i(l) - \alpha \frac{\partial E}{\partial b_i(l)} = b_i(l) - \alpha\delta_i(l),$$

para $l = L-1, L-2, \ldots 2$, em que os $a's$ são computados no processo de *forward pass* e os $\delta's$ são calculados durante a retropropagação (GONZALEZ; WOODS, 2017).

2.3 APRENDIZADO EM REDES NEURAIS CONVOLUCIONAIS

Uma característica importante das CNNs é que, além de usarem matrizes 2D (imagens) como entradas, essas redes são capazes de aprender diretamente dos dados de uma imagem bruta, o que é uma vantagem crucial. Apesar dessas e outras diferenças em relação as redes multicamadas (ou totalmente conectadas) vistas anteriormente, Gonzalez e Woods (2017) destacam que os cálculos realizados por ambas as redes durante o processo de aprendizagem são muito semelhantes.

Como os processos de *forward pass* e *backpropagation* já foram comentados anteriormente para as redes totalmente conectadas, o objetivo desta seção é apenas dar uma visão geral de como essas etapas acontecem em uma rede neural convolucional.

Em relação ao *forward pass*, é necessário destacar que o resultado de uma convolução entre o *kernel* w e uma matriz de entrada com valores a_{xy} pode ser expresso da seguinte maneira:

$$z_{x,y} = \sum_l \sum_k w_{l,k} a_{x-l,y-k} + b = w * a_{x,y} + b, \tag{2.18}$$

em que l e k abrangem as dimensões do *kernel*, x e y percorrem as dimensões da entrada e b é o *bias*. Dessa forma, o valor correspondente de $a_{x,y}$ é dado por:

$$a_{x,y} = h(z_{x,y}).$$

No caso, esse $a_{x,y}$ é diferente do que foi expressado na Equação (2.18), na qual $a_{x,y}$ representa os valores da camada anterior. Agora, é necessário alterar a notação para diferenciar as camadas. Assim como nas redes totalmente conectadas, o l é usado com esse propósito:

$$\begin{aligned} z_{x,y}(l) &= \sum_l \sum_k w_{l,k}(l) a_{x-l,y-k}(l-1) + b(l) \\ &= w(l) * a_{x,y}(l-1) + b(l) \end{aligned} \tag{2.19}$$

e

$$a_{x,y}(l) = h\Big(z_{x,y}(l)\Big), \tag{2.20}$$

para $l = 1, 2, \ldots, L_c$, em que L_c é o número de camadas convolucionais e $a_{x,y}(l)$ denota as características combinadas na camada convolucional l (GONZALEZ; WOODS, 2017). Quando $l = 1$, tem-se:

$$a_{x,y}(0) = \{\text{valores dos } \textit{pixels} \text{ nas imagem de entrada}\},$$

e quando $l = L_c$:

$$a_{x,y}(L_c) = \{\text{valores das características agrupadas na última camada da CNN}\}.$$

Esse resultado e a Equação 2.19 representam tudo que é necessário para o *forward pass* em uma seção convolucional de uma CNN. Os valores de todas as características agrupadas da última camada são vetorizados e alimentados em uma rede neural *feedforward* totalmente conectada, cuja propagação para frente é explicada pelas Equações 2.13 e 2.14 (GONZALEZ; WOODS, 2017).

As equações de alimentação de uma CNN são parecidas com as de uma rede neural totalmente conectada, mas, no caso das CNNs, a multiplicação é substituída pela operação de convolução. Para mostrar de modo geral como ocorre a aprendizagem em uma CNN,

Gonzalez e Woods (2017) fazem um resumo desse processo. Dessa forma, como primeiro passo, a rede é inicializada com um conjunto de pequenos pesos aleatórios e *biases*. Na retropropagação, o vetor que chega da rede totalmente conectada à camada de saída do *pooling* é convertido para matrizes 2D do mesmo tamanho dos mapas de características agrupados nessa camada. Dessa forma, cada mapa de características agrupado é ampliado para corresponder ao tamanho de seu mapa de características correspondente.

Sendo $a(0)$ o conjunto de *pixels* da imagem de entrada da primeira camada e passando pelas Equações 2.19 e 2.20 referentes ao *forward pass*, o próximo passo é justamente a etapa de retropropagação, em que, para cada neurônio em cada mapa de características da camada, ocorre o seguinte cálculo:

$$\delta_{x,y}(l) = h'\Big(z_{x,y}(l)\Big)[\delta_{x,y}(l+1) * rot180(w(l+1))]\,,$$

em que $l = L_c - 1, L_c - 2, \ldots, 1$ e $rot180$ representa uma rotação de 180° para cada *kernel* 2D em uma camada. Por último, existe uma atualização dos pesos e *biases* da seguinte forma:

$$w_{l,k}(l) = w_{l,k}(l) - \alpha\delta_{l,k}(l) * rot180(a(l-1))$$

e, além disso:

$$b(l) = b(l) - \alpha \sum_x \sum_y \delta_{x,y}(l),$$

em que $l = 1, 2, \ldots, L_c$.

É necessário destacar que os pesos em $w(l)$ e o valor do *bias* $b(l)$ são diferentes para cada mapa de características em cada camada. Todo esse procedimento é repetido para um número específico de épocas ou até que o erro de saída da rede neural seja um valor aceitável (GONZALEZ; WOODS, 2017).

2.4 REGULARIZAÇÃO E NORMALIZAÇÃO

Assim como em outras formas de aprendizado de máquina, a regularização e outras técnicas podem ser usadas para que as redes neurais possam generalizar melhor os dados não observados (SZELISKI, 2010).

Antes de comentar essas técnicas, é importante introduzir os conceitos de *overfitting* e *underfitting*. De modo geral, o primeiro ocorre quando um modelo gerado por uma rede neural aprende exageradamente o conjunto de treino, mas não é capaz de generalizar quando é apresentado a novos dados. Por outro lado, quando um modelo apresenta um erro alto tanto para os dados de treino, quanto para os dados de teste, há um indício de ocorrência de *underfitting* (BASGALUPP, 2007).

Dito isso, as penalidades sobre os pesos de uma rede podem ser usadas para melhorar o condicionamento do sistema e para reduzir o *overfitting*. Por exemplo, a utilização da regularização L2 torna os pesos grandes menores, enquanto que o operador

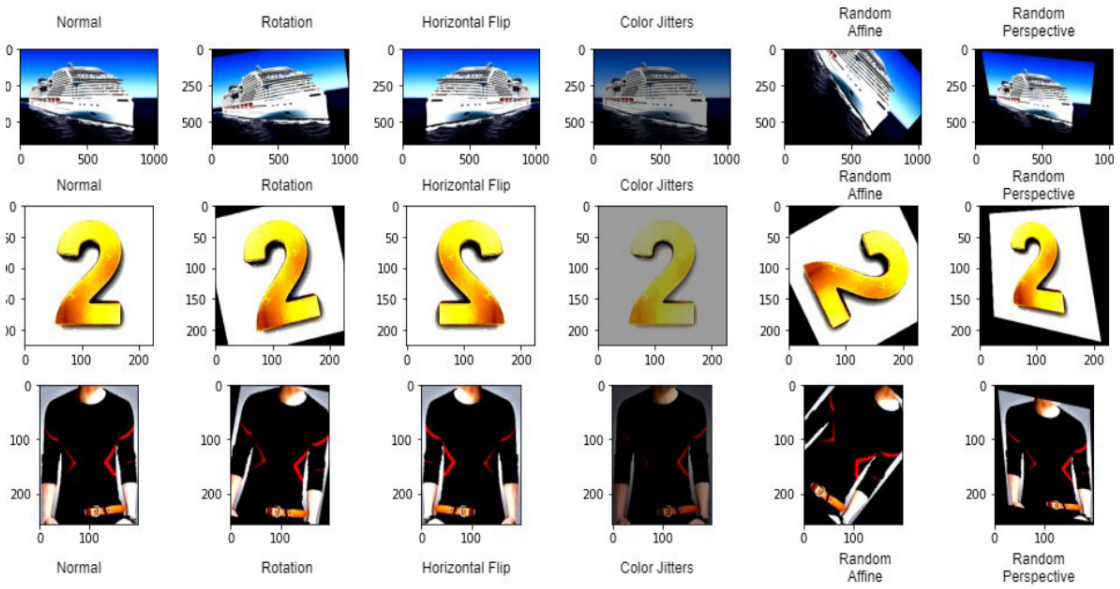
de seleção e contração menos absoluta (*Least Absolute Shrinkage and Selection Operator — LASSO*) pode levar alguns pesos até zero. Esses tipos de regularização também são conhecidos como decaimento dos pesos (*weight decay*) e eles tornam os pesos menores durante a otimização dentro da rede neural (SZELISKI, 2010). Existem diversas opções de algoritmos de otimização, como o SGD e Adam. O primeiro é a sigla para *Stochastic Gradient Descent*, que pode ser traduzido para Método do Gradiente Estocástico. O SGD é um algoritmo baseado no método do gradiente, mas possui uma aleatorização das componentes da função de custo. Já o Adam é uma varição do algoritmo do SGD, mas introduz uma inovação que é o cálculo do primeiro e segundo momentos do gradiente e não somente das taxas individuais de cada parâmetro (JERONYMO, 2019).

Uma outra técnica importante e que pode reduzir o *overfitting* consiste na ampliação artificial do conjunto de treinamento através de perturbações feitas nas imagens originais. Esta técnica é chamada de *data augmentation*, que pode ser traduzido como aumento de dados. Existem diversas operações que podem ser feitas para aumentar os dados, como por exemplo, o *flipping*, *cropping*, *random perspective*, *random affine*, *color jitter* e *rotation*. O *flipping* é uma técnica em que ocorre uma inversão na imagem original, sendo que esta inversão pode ser na horizontal ou na vertical. De acordo com Claro et al. (2020), essa técnica é de fácil implementação e provou ser útil em conjuntos de dados como CIFAR-10 (KRIZHEVSKY; HINTON et al., 2009) e ImageNet (DENG et al., 2009). O *cropping* aplica um corte na imagem em um local e tamanho de saída especificados como parâmetro. Já as técnicas de *random perspective* e *random affine* realizam, respectivamente, uma transformação perspectiva ou afim aleatória da imagem de entrada. O *color jitter* manipula o brilho, contraste e saturação da imagem (TAKAHASHI; MATSUBARA; UEHARA, 2018). Por fim, a rotação é uma técnica em que a imagem pode ser girada para a direita ou esquerda em um ângulo entre 1° e 359°. Nesse caso, o parâmetro do grau de rotação é determinante para o resultado final e, se for aplicado de forma indevida, em alguns casos o rótulo dos dados pode não ser mais preservado após a transformação (CLARO et al., 2020). A Figura 7 mostra uma série de técnicas de aumento de dados que pode ser aplicada a uma imagem.

O *dropout* é outra técnica de regularização apresentada por Srivastava et al. (2014), em que a cada *mini-batch* (um lote que contém parte das amostras) durante o treinamento, uma porcentagem das unidades em cada camada é fixada em zero. Dessa forma, de acordo com Szeliski (2010), esse ajuste aleatório das unidades em zero introduz um ruído no processo de treinamento e evita que a rede se especialize demais em unidades ou tarefas específicas. Dessa forma, o *overfitting* pode ser reduzido e a generalização da rede pode melhorar.

A Figura 8 mostra um exemplo de aplicação do *dropout*. Observando a figura, fica evidente como uma parte das unidades da rede é removida (o mesmo que fixadas em zero). Como isso é feito aleatoriamente em cada *mini-batch*, o ruído injetado no processo

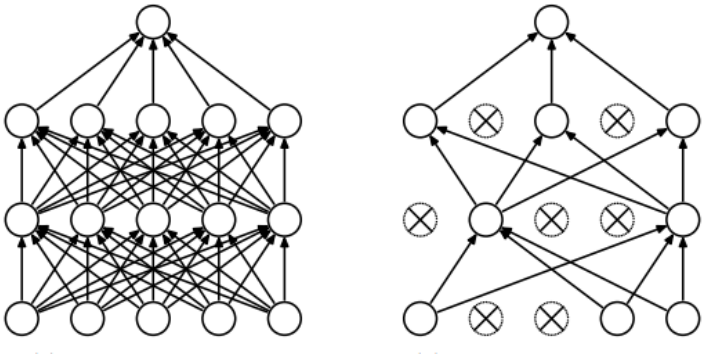
Figura 7: Representação de várias técnicas de *data augmentation*.



Fonte: Ethiraj e Bolla (2022).

de treinamento faz com que a rede não dependa excessivamente de unidades específicas (SZELISKI, 2010).

Figura 8: À esquerda, é possível observar uma rede neural padrão com 2 camadas ocultas. Já à direita, é possível notar o efeito da aplicação do *dropout*, em que as unidades cruzadas foram removidas.



Fonte: Srivastava et al. (2014).

A otimização dos pesos em uma rede neural pode ser considerada um processo complicado e costuma ser lento para a convergência. Desse modo, utilizar uma outra técnica chamada de *batch normalization* também pode ser vantajoso. A ideia por trás dessa técnica é redimensionar e recentralizar as ativações em uma determinada unidade para que elas tenham variância de uma unidade e média zero. Nesse caso, para a função de ativação ReLU, significa que a unidade estará ativa metade do tempo. A normalização é realizada considerando todas as amostras de treinamento n em um determinado *mini-batch*

B e calculando a média e as estatísticas de variância para a unidade i da seguinte forma:

$$\mu_i = \frac{1}{|B|} \sum_{n \in B} s_i^{(n)},$$

$$\sigma_i^2 = \frac{1}{|B|} \sum_{n \in B} (s_i^{(n)} - \mu_i)^2,$$

$$\hat{s}_i^{(n)} = \frac{s_i^{(n)} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}},$$

em que $s_i^{(n)}$ é a soma ponderada da unidade i para a amostra de treinamento n, $\hat{s}_i^{(n)}$ é a soma do lote normalizado correspondente e ϵ é uma pequena constante usada para evitar a divisão por zero. Após a normalização dos lotes, as $\hat{s}_i^{(n)}$ ativações agora possuem média zero e variância unitária. No entanto, essa normalização pode ser feita também com o objetivo de minimizar a função de perda durante o treinamento. Para isso, adiciona-se o parâmetro de ganho extra γ_i e o parâmetro *bias* β_i para cada unidade i e define-se a saída de um estágio da normalização do lote como:

$$y_i = \gamma_i \hat{s}_i + \beta_i.$$

Esses parâmetros agem como pesos regulares, ou seja, são modificados utilizando a descida de gradiente estocástico durante o treinamento com o intuito de reduzir a perda total (SZELISKI, 2010).

Santurkar et al. (2018) publicaram um trabalho sobre como o *batch normalization* ajuda na otimização de uma rede neural. Eles destacam que essa técnica permite um treinamento mais rápido e estável de redes neurais profundas e destacam um impacto fundamental dessa normalização dos dados no processo de treinamento: a técnica torna o cenário de otimização significativamente mais suave e essa suavidade induz um comportamento mais preditivo e estável dos gradientes, o que torna o treinamento mais rápido.

3 MATERIAIS E MÉTODOS

O presente capítulo detalha as características do conjunto de dados utilizado neste trabalho e apresenta a estrutura da rede neural usada para a geração do modelo de classificação, assim como os parâmetros de treinamento adotados.

3.1 CONJUNTO DE DADOS

Em problemas de classificação de imagens utilizando redes neurais convolucionais, os conjuntos de dados utilizados para o treinamento têm um grande impacto sobre os resultados. Luo et al. (2018) fazem uma discussão sobre como um conjunto de dados afeta o desempenho da classificação de imagens em uma CNN e afirmam que, quanto maior for o conjunto de treinamento, melhor será o desempenho da classificação. No entanto, eles destacam que, caso o conjunto de dados seja insuficiente, ainda assim é possível obter bons resultados a partir de um número maior de experimentos.

Este trabalho utiliza um *dataset* desenvolvido no contexto do projeto *Happy Cow ID*, que fez parte do projeto/ação gerencial "Residência Zootécnica Digital", uma iniciativa da Emprapa Gado de Leite em parceria com a Universidade Federal de Juiz de Fora (UFJF) e outras Instituições Federais de Educação Superior (IFES). O conjunto de dados utilizado possui 796 imagens de bovinos coletadas por um especialista da Embrapa Gado de Leite em diferentes ambientes e com informações sobre o estado de saúde, categoria, pose e raça de cada imagem. Ao todo, foram coletadas imagens de 151 indivíduos em 2 fazendas no período de junho de 2021 a maio de 2022.

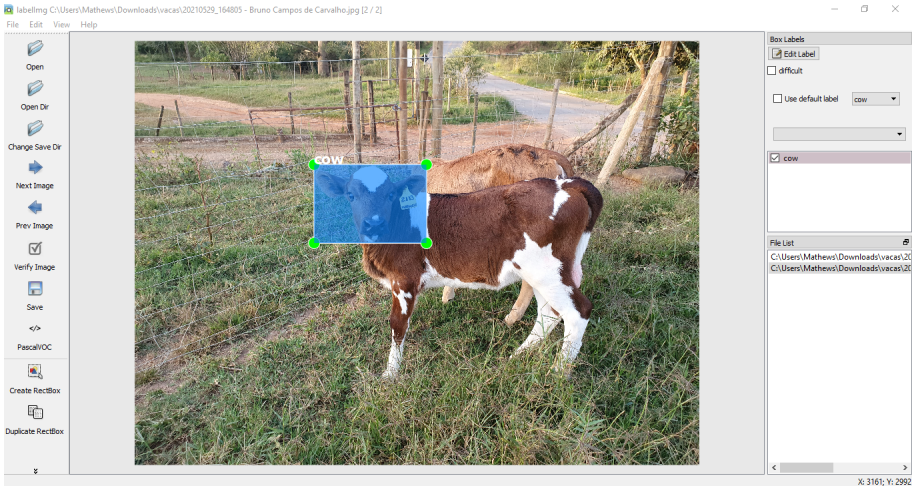
As imagens capturadas são coloridas e possuem diferentes resoluções, como 4624×3468, 4032×3024, 3468×4624 e 3024×4032 *pixels*. Como o propósito deste trabalho foi realizar classificações relacionadas a face do animal, concluiu-se que seria vantajoso fazer um recorte facial dos bovinos, a fim de descartar informações desnecessárias para o treinamento dos modelos de aprendizado profunda. Por exemplo, para a classificação da pose de um animal, não é necessário observar o corpo do bovino e o ambiente em que ele está inserido. Dessa forma, fazer uma detecção facial seguida de um recorte da face detectada foi uma estratégia que se mostrou interessante para o desenvolvimento do projeto.

Inicialmente, para fazer a detecção e recorte facial de todas as imagens capturadas sem um modelo de detecção, um integrante do projeto realizou a rotulação das imagens manualmente através da ferramenta *LabelImg* [1], que possibilita o enquadramento do objeto a ser detectado e gera para cada imagem um arquivo em formato XML com as coordenadas do objeto de interesse. Dessa forma, com as coordenadas da face do animal de interesse em cada imagem, foi possível realizar o recorte facial utilizando um *script* desenvolvido com a

[1] https://github.com/heartexlabs/labelImg

linguagem *Python*. A Figura 9 mostra o processo de rotulação de uma imagem através do *LabelImg* e a Figura 10 apresenta um exemplo de arquivo em formato XML gerado após a rotulação. A resolução das imagens resultantes é variada, visto que o processo de recorte gera imagens de diversos tamanhos por conta do enquadramento da face do animal. Um fator que explica essa diferença é que houve uma variação na distância de captura dos animais. Dessa forma, a maior imagem resultante ficou com resolução de 3004×2141 *pixels*, enquanto a menor imagem ficou com resolução de 117×102 *pixels*. A média de largura das imagens recortadas foi de 998 *pixels* e a média de altura das imagens recortadas foi de 993 *pixels*.

Figura 9: Tela da ferramenta *LabelImg* durante o processo de rotulação da face de um bezerro em uma imagem capturada.



Fonte: Elaborada pelo autor (2023).

Figura 10: Exemplo de um arquivo XML gerado após a anotação de uma face através da ferramenta *LabelImg*.

```
<annotation>
    <folder>test</folder>
    <filename>2114_Hol_Sadio_2.jpg</filename>
    <path>C:\Users\Mathews\TFODCourse-original\Tensorflow\workspace\images\test\2114_Hol_Sadio_2.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>3468</width>
        <height>4624</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>cow</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>800</xmin>
            <ymin>1987</ymin>
            <xmax>2081</xmax>
            <ymax>4068</ymax>
        </bndbox>
    </object>
</annotation>
```

Fonte: Elaborada pelo autor (2023).

A Figura 11 mostra o exemplo de duas imagens de categorias diferentes presentes no conjunto de dados, em que é possível observar um bezerro (Figura 11a) e uma vaca

(Figura 11b). Observando a figura, nota-se que existe uma diferença fundamental entre a morfologia da cabeça dos bezerros e vacas. A Figura 12 apresenta exemplos de imagens de uma mesma vaca em duas poses diferentes. É necessário destacar que algumas imagens do *dataset* podem conter uma visão mais diagonalizada da face do animal, o que torna a definição do rótulo de pose um pouco mais difícil, como pode ser visto na Figura 13, que possui um animal rotulado com a pose frontal (Figura 13a) e um animal rotulado com a pose lateral (Figura 13b).

Figura 11: Exemplos de imagens de categorias diferentes.

(a) Bezerro. (b) Vaca.

Fonte: Elaborada pelo autor (2023).

Figura 12: Exemplos de imagens de poses diferentes.

(a) Frontal. (b) Lateral.

Fonte: Elaborada pelo autor (2023).

A Figura 14 exibe duas imagens de animais de raças diferentes. Nesse caso, como pode ser observado na figura, diferenciar a raça de alguns bovinos pode ser uma tarefa difícil para observadores leigos, pois as diferenças entre os animais são mais sutis. Esse aspecto pode ser explicado pela composição genética (ou grau de sangue) dos animais, fazendo com que certos indivíduos carreguem características de mais de uma raça. Por exemplo, o Girolando é uma raça considerada sintética, visto que é obtida a partir do

cruzamento entre as raças Holandesa e Gir. Baseado nisso, é possível avaliar a possibilidade de, futuramente, dividir o *dataset* em classes que consideram a composição genética dos animais.

Figura 13: Exemplos de imagens desafiadoras para o problema de pose.

(a) Frontal.

(b) Lateral.

Fonte: Elaborada pelo autor (2023).

Figura 14: Exemplos de imagens de raças diferentes.

(a) Girolando.

(b) Holandês.

Fonte: Elaborada pelo autor (2023).

A Tabela 1 mostra a distribuição do *dataset* de acordo com os rótulos de categoria e raça, sendo que é possível perceber que existe um balanceamento em relação a quantidade de imagens das classes bezerro e vaca. Nesta tabela, também é possível notar que, em relação à distribuição do *dataset* de acordo com os rótulos de raça, há um desbalanceamento maior entre as classes, já que a classe Holandês possui mais que o dobro de imagens em relação à classe Girolando.

A Tabela 2 apresenta a distribuição do *dataset* de acordo com os rótulos de pose e raça. Ao contrário do caso da distribuição de categoria, é possível notar que nesse caso existe um leve desbalanceamento, visto que há uma quantidade relativamente maior de imagens para a classe frontal.

Tabela 1 – Distribuição do *dataset* para os rótulos de categoria e raça.

	Girolando	Holandês	Total
Bezerro	93	316	409
Vaca	133	254	387
Total	226	570	796

Fonte: Elaborada pelo autor (2023).

Tabela 2 – Distribuição do *dataset* para os rótulos de pose e raça.

	Girolando	Holandês	Total
Frontal	99	393	492
Lateral	127	177	304
Total	226	570	796

Fonte: Elaborada pelo autor (2023).

Buscando ver a correlação entre as características das classes do conjunto de dados, foram feitas divisões no *dataset* combinando as seguintes características: raça com categoria e raça com pose. Em relação à distribuição do *dataset* de acordo com os rótulos de raça, mas levando-se em conta apenas as imagens de bezerros, é possível observar na Tabela 1 que existe um desbalanceamento severo entre as classes, em que a classe Girolando possui apenas $22,74\%$ do total de imagens, contra $77,26\%$ da classe Holandês. No caso da distribuição do *dataset* de acordo com os rótulos de raça, levando-se em conta apenas as imagens de vacas, nota-se que o desbalanceamento entre as classes é menor, mas ainda assim, a classe Girolando possui uma quantidade consideravelmente menor de imagens.

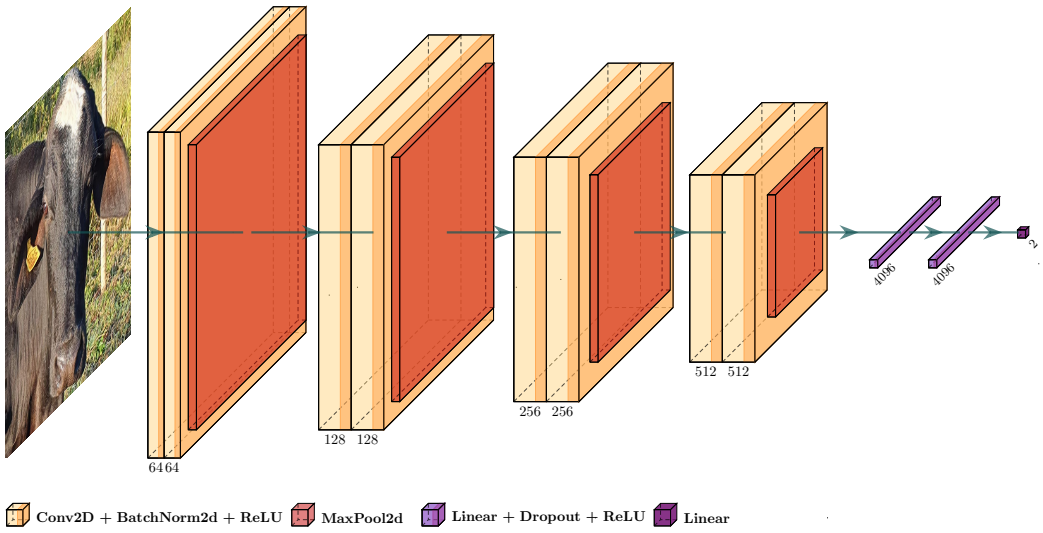
A Tabela 2 detalha a combinação entre a informação da raça com a pose do animal. Considerando-se apenas as imagens em que o animal está em uma pose frontal, percebe-se novamente um grande desbalanceamento entre as classes, em que a classe Girolando possui apenas $20,12\%$ do total de imagens, contra $79,88\%$ da classe Holandês. Levando-se em conta apenas imagens em que o animal está em uma pose lateral, o problema do desbalanceamento entre as classes é um pouco menor quando comparado com a combinação de raça com poses frontais.

3.2 MODELO DE REDE NEURAL

O presente trabalho utilizou uma variação da rede VGGNet (SIMONYAN; ZISSERMAN, 2014) proposta no trabalho de Khaireddin e Chen (2021) e, com base nos experimentos e na avaliação dos resultados obtidos, foram feitas modificações em alguns elementos da rede e nos parâmetros de treinamento. Como este trabalho envolve diferentes tipos de classificação de imagem, a rede foi alterada para atender as especificidades de

cada problema. Elaborada através da ferramenta *PlotNeuralNet*[2], a Figura 15 apresenta uma visão geral da arquitetura de rede neural convolucional utilizada para abordar todos os problemas de classificação discutidos neste trabalho.

Figura 15: Representação da arquitetura da rede utilizada.



Fonte: Elaborada pelo autor (2023).

Para a classificação de categoria, as imagens de entrada são coloridas e possuem resolução de 168×168 *pixels*. A rede utilizada é composta por quatro estágios convolucionais e três camadas totalmente conectadas. Cada estágio convolucional possui dois blocos convolucionais e uma camada de *max-pooling*. Cada bloco convolucional consiste em uma camada convolucional, seguida de uma camada de *batch normalization* e uma camada de ativação do tipo ReLU. Cada uma das duas primeiras camadas totalmente conectadas é seguida pelo método de regularização *dropout* e uma camada de ativação do tipo ReLU. A terceira camada totalmente conectada tem dois nós de saída, cada um representando uma classe do problema abordado. Todas as camadas convolucionais utilizam *stride* igual a 1, *padding* de 0, *kernel* de tamanho 5×5 e cada estágio convolucional possui, respectivamente, 64, 128, 256 e 512 *feature maps*. Dessa forma, ao final do último estágio convolucional, são gerados 512 parâmetros ($1 \times 1 \times 512$) que entram na primeira camada totalmente conectada da rede. É preciso destacar também que é usada uma distribuição normal para fazer a inicialização dos pesos na primeira camada convolucional da rede.

A rede utilizada para o problema da classificação de pose é muito semelhante à rede usada para a classificação da categoria, exceto pela alteração na resolução da imagem de entrada para 116×116 *pixels* e pela adição de um *padding* de 1 em todas as camadas convolucionais. Além disso, ao final do último estágio convolucional, são gerados 2048 parâmetros ($2 \times 2 \times 512$) que entram na primeira camada totalmente conectada dessa rede.

[2] https://github.com/HarisIqbal88/PlotNeuralNet

Como já foi comentado na seção anterior, para o problema da classificação de raça, além do caso que utiliza todas as imagens do conjunto de dados, foram feitas divisões no *dataset* combinando algumas características. Nesse caso, cada uma dessas divisões foi abordada com modificações específicas na rede de treinamento. Para o problema mais geral, isto é, aquele que envolve o *dataset* completo, a rede utilizada é semelhante à rede para a classificação da categoria, porém utiliza um *kernel* de tamanho 3 em todas as camadas convolucionais e inicializa os pesos de todas as camadas convolucionais com os pesos da rede VGG16 pré-treinada a partir do conjunto de dados ImageNet (DENG et al., 2009). Além disso, as imagens de entrada para a rede utilizada neste problema possuem resolução de 190×190 *pixels*. Ao final do último estágio convolucional, são gerados 18432 parâmetros ($6 \times 6 \times 512$) que entram na primeira camada totalmente conectada dessa rede. Especificamente nesse problema, não ocorre a utilização da distribuição normal responsável por fazer a inicialização dos pesos na primeira camada convolucional da rede.

Para os problemas da classificação de raça de bezerros e vacas, os blocos convolucionais da VGGNet são similares aos que foram utilizados na rede de classificação de categoria, exceto pela alteração na resolução da imagem de entrada para 65×65 *pixels* e pela adição de um *padding* de 1 e *kernel* de tamanho 5×5 em todas as camadas convolucionais. Nesse caso, ao final do último estágio convolucional, são gerados 8192 parâmetros ($4 \times 4 \times 512$) que entram na primeira camada totalmente conectada dessa rede.

Por fim, os casos de classificação de raça em poses frontais e laterais utilizam redes semelhantes com aquelas que foram usadas para os problemas da classificação de raça de bezerros e vacas, exceto por uma alteração no tamanho das imagens de entrada e no tamanho do *padding* nas camadas convolucionais. Para a classificação de raça em poses frontais, as imagens possuem uma resolução de 91×91 *pixels*, enquanto que para a classificação de raça em poses laterais, as imagens possuem resolução de 96×96 *pixels* e a rede conta com um *padding* de tamanho 2. Sobre a rede desenvolvida para o problema de raça em poses frontais, ao final do último estágio convolucional, são gerados 512 parâmetros ($1 \times 1 \times 512$) que entram na primeira camada totalmente conectada dessa rede. Já para o problema de raça em poses laterais, ao final do último estágio convolucional, são gerados 2048 parâmetros ($2 \times 2 \times 512$).

4 EXPERIMENTOS COMPUTACIONAIS E RESULTADOS

Este capítulo apresenta os detalhes do ambiente e os protocolos utilizados para realizar os experimentos computacionais. Além disso, o capítulo discorre sobre os resultados obtidos para cada um dos problemas abordados pelo presente trabalho.

4.1 PROTOCOLO DE TREINAMENTO

Todos os experimentos relatados neste trabalho foram realizados em um computador com processador Intel(R) Core(TM) i5-10400 CPU @ 2.90GHz, com 8.00 GB de memória RAM (7.61 GB disponíveis) e GPU NVIDIA GeForce RTX 3060 com 12 GB de memória RAM. Os modelos foram desenvolvidos com a linguagem *Python* 3.6.5 e a biblioteca de código-fonte aberto *Pytorch* 1.10.2.

Para a definição dos conjuntos de treinamento e teste do *dataset*, a técnica de validação cruzada *k-fold* foi utilizada. Dessa forma, ocorre uma divisão aleatória em k partes de tamanhos iguais, sendo que $k-1$ compõem os dados de treinamento e a outra parte fica reservada para os dados de teste. Esse processo é feito até que todas as partes tenham integrado tanto o treinamento quanto o teste do modelo, o que resulta em k estimativas (SANTOS et al., 2019). Neste trabalho, os dados foram estratificados por classe e a validação cruzada *k-fold* com $k = 3$ foi empregada 3 vezes para cada experimento. Para cada divisão, 2/3 dos dados foram destinados para o treinamento e 1/3 destinados para o teste, sendo que, dos dados de treinamento, 1/4 foram usados como dados de validação. É preciso destacar que, para cada divisão dos conjuntos de treinamento, validação e teste, houve um tratamento para impedir que um determinado indivíduo estivesse presente em mais de um conjunto. Visto que um indivíduo possui mais de uma imagem no *dataset*, a presença de imagens do mesmo animal em conjuntos diferentes para uma determinada distribuição poderia imputar viés na rede.

Em relação ao aumento de dados realizado durante os treinamentos, todos os experimentos relatados na Seção 4.2 utilizaram as operações de *flipping* horizontal, rotação aleatória entre as faixas de $-40°$ e $40°$, perspectiva aleatória com um grau de distorção de $0,4$ e o *TenCrop*, que, dada uma dimensão específica, gera 10 novas imagens a partir de recortes da imagem original[3].

Além disso, todos os experimentos utilizaram a função de perda entropia cruzada e o otimizador SGD com os parâmetros de momento igual a $0,9$, decaimento de pesos com fator igual a $0,0004$ e *nesterov* definido como verdadeiro. A taxa de aprendizagem utilizada pelo otimizador varia de acordo com o problema e, por isso, será especificada nas subseções referentes a cada caso. Todas essas características e os parâmetros que são relatados a seguir foram definidos empiricamente.

[3] https://pytorch.org/vision/main/generated/torchvision.transforms.TenCrop.html

4.2 RESULTADOS

As subseções a seguir apresentam os melhores resultados alcançados para cada problema de classificação abordado neste trabalho, assim como uma análise feita partir da avaliação de determinadas métricas. Como o número de experimentos feitos e a quantidade de problemas abordados são grandes, decidiu-se apresentar apenas a configuração dos experimentos que alcançaram os melhores resultados. No entanto, vários modelos e parâmetros foram testados para cada problema.

A Tabela 3 mostra um exemplo de matriz de confusão gerada para comentar como os valores de *true positive* (TP), *true negative* (TN), *false positive* (FP) e *false negative* (FN) podem ser utilizados para calcular algumas métricas de avaliação de resultados. A acurácia é uma métrica importante para observar, dentre todas as predições feitas pelo modelo, quantas foram classificadas corretamente. O cálculo da acurácia pode ser feito a partir do seguinte cálculo:

$$\text{Acurácia} = \frac{TP + TN}{TP + TN + FP + FN}.$$

A precisão indica, dentre todas as predições que que o modelo fez para classe positivo, quantas estão corretas. A precisão pode ser calculada a partir da seguinte fórmula:

$$\text{Precisão} = \frac{TP}{TP + FP}.$$

O *recall* (também chamado de revocação ou sensibilidade) define, dentre todas as situações em que a classe positivo é esperada, quantas estão corretas. Essa métrica pode ser encontrada da seguinte forma:

$$\textit{Recall} = \frac{TP}{TP + FN}.$$

O F_1 *score* representa uma forma de observar a precisão e o *recall* através de apenas um número. Ele pode se encontrado a partir da seguinte fórmula:

$$F_1 \ \textit{score} = \frac{2 \cdot \text{Precisão} \cdot \textit{Recall}}{\text{Precisão} + \textit{Recall}}.$$

Já o F_β *score* é uma generalização do F_1 *score*, em que é introduzido um termo *Beta* responsável por ponderar o quanto o *recall* é mais importante que a precisão para o cálculo da métrica. A fórmula dessa métrica é dada por:

$$F_\beta \ \textit{score} = \frac{((1 + \text{Beta}^2) \cdot \text{Precisão} \cdot \textit{Recall})}{\text{Beta}^2 \cdot \text{Precisão} + \textit{Recall}}.$$

Dessa forma, no F_2 *score*, o *recall* passa a ter um peso maior que a precisão. Para avaliar os resultados deste trabalho, as métricas de acurácia, precisão, *recall*, F_1 *score* e F_2 *score* foram geradas a partir do conjunto de teste. Para realizar os cálculos das métricas de cada

Tabela 3 – Elementos que compõem uma matriz de confusão.

		Predição	
		Negative	***Positive***
Classes	***Negative***	*True Negative*	*False Positive*
	Positive	*False Negative*	*True Positive*

Fonte: Elaborada pelo autor (2023).

classe, foi necessário considerar a classe de interesse como a classe positiva, enquanto a outra classe foi considerada como negativa.

Além disso, as próximas subseções apresentam, para cada problema, a matriz de confusão da validação cruzada que atingiu a maior acurácia geral a partir do melhor modelo. Uma matriz de confusão é um esquema que mostra os erros e acertos do modelo e comparando os resultados obtidos com os esperados.

É necessário destacar que, para alguns problemas de classificação, foi necessário reduzir as dimensões das imagens de entrada. Isso foi feito com vista a diminuir o custo computacional dos processos de treinamento das redes, já que imagens de entrada com resoluções maiores tendem a demandar um gasto de memória maior na máquina em que ocorrem os treinamentos. Outros fatores que podem influenciar no gasto de memória são o *batch size*, a quantidade de imagens nos conjuntos de treinamento e validação e o tamanho do *kernel* utilizado nas camadas convolucionais da rede. Dessa forma, este trabalho conta com imagens de entrada em resoluções diferentes para cada problema abordado, visto que cada problema possui uma configuração específica de parâmetros.

4.2.1 Classificação de categoria

Para o problema de classificação de categoria, o experimento com o melhor resultado utilizou *batch size* igual a 16, taxa de aprendizagem igual a $0,005$ e o treinamento durou 300 épocas. Além disso, para a normalização das imagens de entrada, foram utilizados uma média e um desvio padrão de $0,5$.

A Tabela 4 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, contendo o desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 5 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a terceira validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 16 apresenta a matriz de confusão gerada a partir da terceira validação cruzada.

Os valores das acurácias por classe da Tabela 4 mostram que as classes bezerro e vaca atingiram acurácias próximas, o que deixou os desvios-padrões pequenos. Os valores da acurácia geral de cada validação cruzada também ficaram próximos uns dos outros. Esses comportamentos mostram como o modelo atingiu acurácias altas para as duas classes. Além disso, o modelo apresentou pouca variação de acurácia geral entre as validações

Tabela 4 – Resultados obtidos para o problema de categoria.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Bezerro	**Vaca**		
1	93,15	91,73	0,58	92,46
2	88,75	91,99	1,32	90,33
3	93,15	93,02	0,05	93,09
Média	91,68	92,25	0,23	91,96
dp	2,07	0,56	0,62	1,18

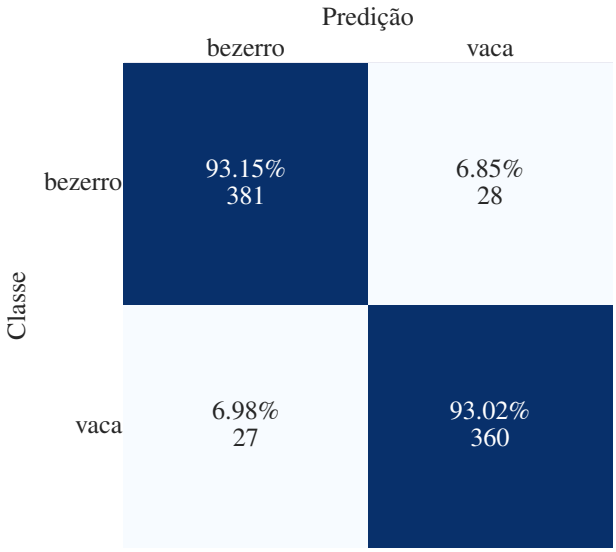
Fonte: Elaborada pelo autor (2023).

Tabela 5 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de categoria.

Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Bezerro	0,934	0,931	0,933	0,932
Vaca	0,928	0,930	0,929	0,930

Fonte: Elaborada pelo autor (2023).

Figura 17: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de categoria.



Fonte: Elaborada pelo autor (2023).

cruzadas. A Figura 16 evidencia como há um equilíbrio entre as acurácias de cada classe, sendo que os números de amostras preditas de forma incorreta em casa classe também são muito próximos.

Observando a Tabela 5, nota-se que todos os valores estão muito próximos a 1, indicando que o modelo teve bons resultados para ambas as classes, o que pode ser constatado a partir dos resultados da Tabela 4, já que ambas as classes apresentam acurácia

média acima de 91%.

4.2.2 Classificação de pose

Em relação ao problema de classificação de pose, o experimento com o melhor resultado utilizou *batch size* igual a 32 e o treinamento durou 200 épocas. Para a normalização das imagens de entrada, foram utilizados uma média de $0,5$ e um desvio padrão de 255. Além disso, o experimento contou com um programador da taxa de aprendizagem, o *ReduceLROnPlateau*[4], com uma taxa de aprendizagem inicial de $0,1$, fator de redução de $0,75$ e paciência de 5. Dessa forma, o programador verifica a acurácia de validação ao longo do treinamento e, se nenhuma melhora for observada dentro de 5 épocas, a taxa de aprendizado é reduzida.

A Tabela 6 apresenta os resultados obtidos com as 3 validações cruzadas do melhor modelo, incluindo o desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 7 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a segunda validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 17 exibe a matriz de confusão gerada a partir da segunda validação cruzada.

Tabela 6 – Resultados obtidos para o problema de pose.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Frontal	**Lateral**		
1	96,14	90,79	2,20	94,10
2	94,51	93,75	0,31	94,22
3	95,73	89,80	2,44	93,47
Média	95,46	91,45	1,65	93,93
dp	0,69	1,68	0,57	0,33

Fonte: Elaborada pelo autor (2023).

Tabela 7 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de pose.

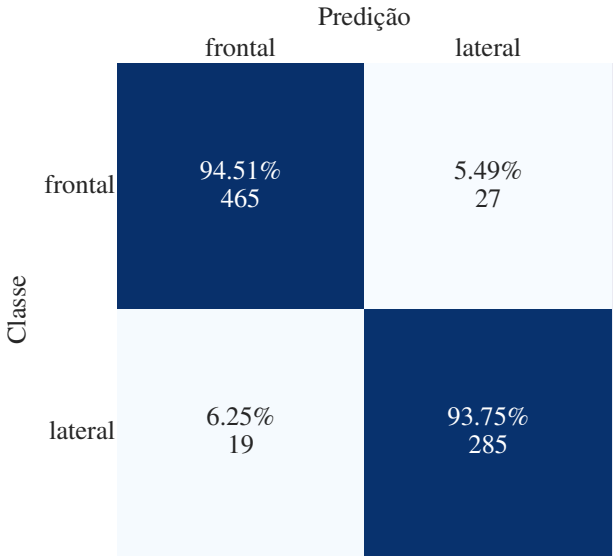
Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Frontal	0,961	0,945	0,953	0,948
Lateral	0,913	0,937	0,925	0,932

Fonte: Elaborada pelo autor (2023).

Assim como para o problema de classificação de categoria, o classificador de pose também atingiu resultados satisfatórios, com acurácia média de $93,93\%$. Uma característica que pode ser destacada a partir da Tabela 6 é que todas as acurácias por classe atingiram

[4] https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.ReduceLROnPlateau.html

Figura 18: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de pose.



Fonte: Elaborada pelo autor (2023).

valores acima de 89%, sendo que em todas as validações cruzadas, a classe frontal teve acurácias maiores que a classe lateral. Assim como aconteceu para o problema de categoria, a Figura 17 mostra como há um equilíbrio entre as acurácias de casa classe, sendo que os números de amostras preditas de forma incorreta em casa classe também são próximos.

A Tabela 7 mostra que a classe frontal possui resultados melhores que a classe lateral para as métricas F_1 *score* e F_2 *score*. Em relação ao *recall*, as duas classe atingiram o mesmo valor de $0,94$. Assim como para o problema de categoria, os valores altos em todas as métricas também indicam um comportamento satisfatório e um bom potencial para o modelo.

4.2.3 Classificação de raça

Como foi destacado na Tabela 1, há um desbalanceamento entre as classes Girolando e holandês, o que gerou, ao longo dos experimentos realizados, resultados que mostravam uma acurácia muito maior para a classe majoritária. Nesse sentido, especificamente para esse problema, foram feitas manipulações no *dataset* com a finalidade de observar o impacto que o desbalanceamento entre as classes poderia causar nos resultados. Desse modo, notou-se que a melhor manipulação foi aquela que deixou o conjunto de dados com a quantidade de indivíduos da raça Girolando exatamente igual à quantidade de indivíduos da raça holandesa. Essa alteração resultou em um número menor de imagens, mas diminuiu o desbalanceamento entre as classes.

Juntamente com a alteração citada acima, o experimento com melhor resultado utilizou *batch size* igual a 32, taxa de aprendizagem igual a $0,005$ e o treinamento durou 300 épocas. Além disso, para a normalização das imagens de entrada, foram utilizados

uma média e um desvio padrão de $0,5$.

A Tabela 8 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, além do desvio padrão (dp) e média das acurácias gerais e por classe, enquanto que a Tabela 9 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a segunda validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Por fim, a Figura 18 apresenta a matriz de confusão gerada a partir da segunda validação cruzada.

Tabela 8 – Resultados obtidos para o problema de raça.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Girolando	**Holandês**		
1	87,61	75,86	5,88	82,05
2	88,94	78,23	5,36	84,72
3	88,05	76,84	5,61	83,13
Média	88,20	76,98	5,61	83,30
dp	0,55	0,97	0,21	1,10

Fonte: Elaborada pelo autor (2023).

Tabela 9 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça.

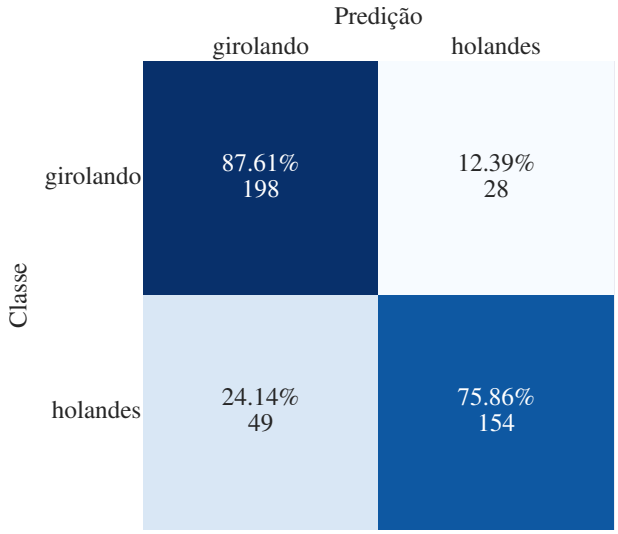
Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Girolando	0,802	0,876	0,837	0,860
Holandês	0,846	0,759	0,800	0,775

Fonte: Elaborada pelo autor (2023).

Observando a Tabela 8 e a Figura 18, é possível notar que a acurácia da classe Girolando é maior que a acurácia da classe Holandês em todas as validações cruzadas. Esse comportamento é bem diferente do que era observado antes da manipulação feita no conjunto de dados comentada anteriormente. Desse modo, ao igualar o número de indivíduos das duas raças presentes no *dataset*, o modelo passou a favorecer a classe girolando, ao contrário do que ocorria anteriormente.

A Tabela 9 evidencia que, apesar de possuir uma acurácia menor, a classe Holandês obteve uma precisão ligeiramente maior que a classe Girolando. Isso mostra que, apesar de ter acurácia menor, a classe Holandês consegue acertar com precisão que 85% dos animais classificados como sendo holandeses realmente são dessa raça. Já em relação ao *recall*, F_1 *score* e F_2 *score*, a classe Girolando possui melhores resultados, evidenciando que essa classe consegue acertar 88% dos animais dessa raça presentes no *dataset*. De modo geral, os valores altos em todas as métricas indicam um comportamento satisfatório e um

Figura 19: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça.



Fonte: Elaborada pelo autor (2023).

bom potencial para o modelo. Apesar de ter obtido resultados com acurácia média acima de 76% para as duas classes, esse problema se mostrou mais difícil que os demais. Isso motivou a análise da classificação de raça separada por categoria e pose.

4.2.4 Classificação de raça de bezerros

Para o problema da classificação de raça a partir de imagens somente de bezerros, o experimento com o melhor resultado utilizou *batch size* igual a 32, taxa de aprendizagem igual a $0,005$ e o treinamento durou 200 épocas. Para a normalização das imagens de entrada, foram utilizados uma média e um desvio padrão de $0,5$. Para os experimentos desta e das próximas subseções, não houve uma manipulação do *dataset* de modo a deixar as duas classes com a mesma quantidade de indivíduos.

A Tabela 10 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, contendo o desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 11 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a terceira validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 19 apresenta a matriz de confusão gerada a partir da terceira validação cruzada.

A Tabela 10 deixa evidente como há um certo desequilíbrio entre as classes em relação aos resultados, já que a classe Holandês atinge acurácias bem maiores que a classe Girolando. Essa diferença fica exposta pelo valor maior de desvio padrão para cada validação cruzada. Observando especificamente a Figura 19, é possível constatar que a quantidade amostras da classe Holandês é muito maior quando comparada com a classe Girolando. Dessa forma, é possível constatar o impacto que o desequilíbrio do *dataset* gera nos resultados.

Tabela 10 – Resultados obtidos para o problema de raça de bezerros.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Girolando	**Holandês**		
1	72,04	97,78	11,02	91,93
2	63,44	94,94	13,48	87,78
3	74,19	97,78	10,10	92,42
Média	69,89	96,83	11,53	90,71
dp	4,64	1,34	1,42	2,08

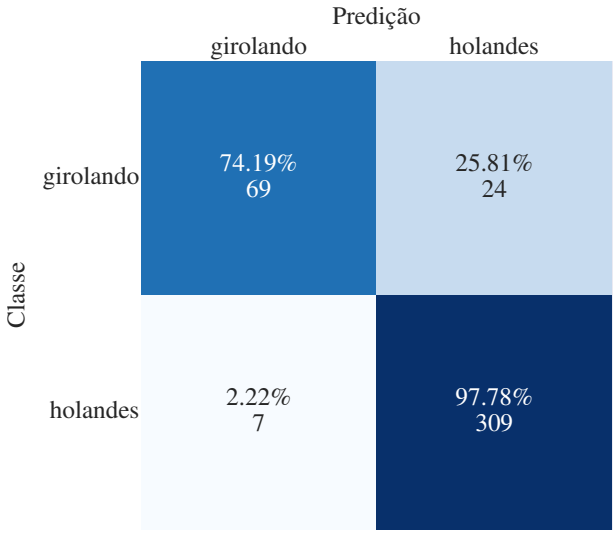
Fonte: Elaborada pelo autor (2023).

Tabela 11 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça de bezerros.

Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Girolando	0,908	0,742	0,817	0,770
Holandês	0,928	0,978	0,952	0,967

Fonte: Elaborada pelo autor (2023).

Figura 20: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça de bezerro.



Fonte: Elaborada pelo autor (2023).

Em relação a Tabela 11, nota-se que a classe Holandês possui resultados satisfatórios para todas as métricas, com destaque para o *recall*, que atinge $0,98$. Já a classe Girolando possui bons resultados para as métricas de precisão e F_1 *score*. Esses dados trazem algumas constatações interessantes, como o fato que a classe Girolando conseguiu acertar com precisão que 91% dos animais classificados como Girolando realmente são dessa raça.

4.2.5 Classificação de raça de vacas

Sobre o problema da classificação de raça a partir de imagens somente de vacas, o experimento com o melhor resultado utilizou *batch size* igual a 32, taxa de aprendizagem igual a $0,005$ e o treinamento durou 200 épocas. Para a normalização das imagens de entrada, foram utilizados uma média e um desvio padrão de $0,5$.

A Tabela 12 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, incluindo o desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 13 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a segunda validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 20 apresenta a matriz de confusão gerada a partir da segunda validação cruzada.

Tabela 12 – Resultados obtidos para o problema de raça de vacas.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Girolando	**Holandês**		
1	64,66	80,31	6,49	74,94
2	60,15	86,22	10,82	77,26
3	50,38	84,25	14,05	72,61
Média	58,40	83,59	10,45	74,94
dp	5,96	2,46	1,80	1,90

Fonte: Elaborada pelo autor (2023).

Tabela 13 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça de vacas.

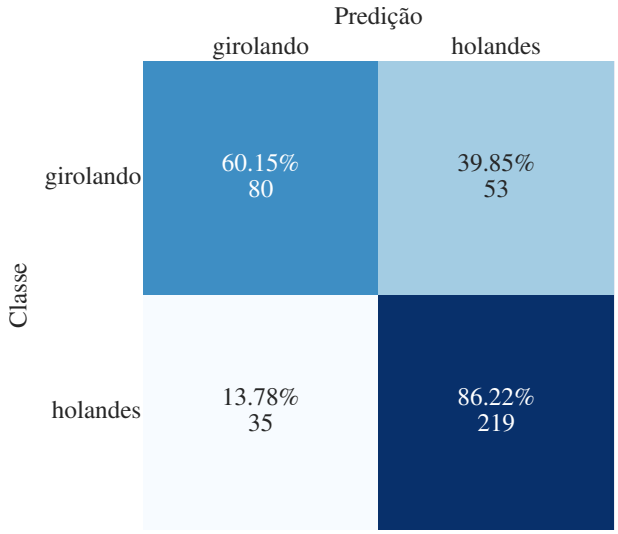
Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Girolando	0,696	0,601	0,645	0,618
Holandês	0,805	0,862	0,832	0,850

Fonte: Elaborada pelo autor (2023).

Com relação aos resultados da Tabela 12, é possível afirmar que, assim como para o problema de raça de bezerros, a classe Holandês possui acurácias bem maiores que a classe Girolando em todas as validações cruzadas. No entanto, as acurácias atingidas pela classe Holandês não ultrapassam $86,24\%$. Destaca-se negativamente a acurácia da classe Girolando da terceira validação cruzada, que é de apenas $50,38\%$, um valor baixo para um problema de classificação binária. Esse comportamento é representado pelo alto valor de desvio padrão entre as acurácias das classes de cada validação cruzada.

Sobre a Figura 20, nota-se, assim como para o problema de raça de bezerros, que há um desequilíbrio em relação ao número de amostras para cada classe no conjunto de teste, sendo que a Tabela 1 mostra que essa é uma característica de todo o conjunto de

Figura 21: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça de vaca.



Fonte: Elaborada pelo autor (2023).

dados. Dito isso, os resultados evidenciam novamente o impacto desse desequilíbrio nos resultados obtidos, já que a acurácia da classe Girolando é bem menor e muitas amostras dessa classe são preditas como sendo da classe Holandês.

Observando a Tabela 13, o fato da classe Girolando ter valores mais baixos para as métricas de *recall*, F_1 *Score* e F-2 *Score* chama a atenção negativamente. Até mesmo a precisão da classe Girolando é baixa, visto que o modelo consegue acertar com precisão que 70% dos animais classificados como sendo da classe Girolando realmente são dessa classe. Já a classe Holandês possui resultados melhores em relação a essas métricas. Comparando esses resultados com os valores da Tabela 11, fica evidente como a classificação de raça é mais difícil para vacas do que bezerros.

4.2.6 Classificação de raça em poses frontais

Para o problema da classificação de raça a partir de imagens somente de animais em poses frontais, o experimento com o melhor resultado utilizou *batch size* igual a 32, taxa de aprendizagem igual a $0,005$, o treinamento durou 200 épocas e, para a normalização das imagens de entrada, foram utilizados uma média e um desvio padrão de $0,5$.

A Tabela 14 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, além do desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 15 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a segunda validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 21 apresenta a matriz de confusão gerada a partir da segunda validação cruzada.

Assim como para os problemas que envolvem a combinação de raça com a característica de categoria, para o caso da combinação de raça de animais em poses frontais,

Tabela 14 – Resultados obtidos para o problema de raça em pose frontal.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Girolando	**Holandês**		
1	59,60	95,93	15,69	88,62
2	66,67	94,40	11,98	88,82
3	61,62	95,17	14,49	88,41
Média	62,63	95,17	14,05	88,62
dp	2,97	0,62	1,23	0,17

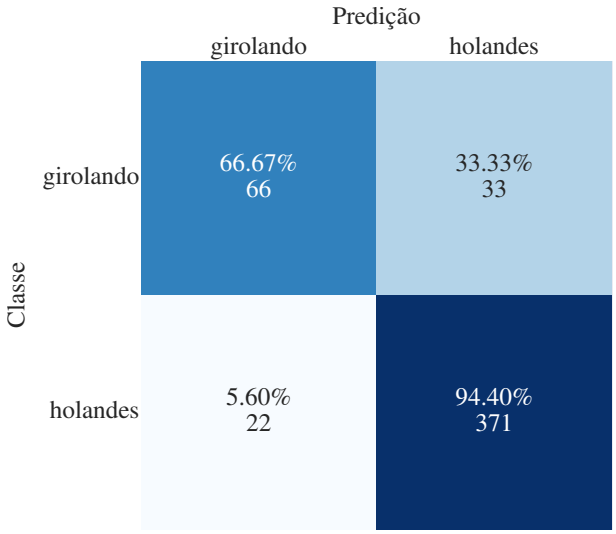
Fonte: Elaborada pelo autor (2023).

Tabela 15 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça em poses frontais.

Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Girolando	0,750	0,667	0,706	0,681
Holandês	0,918	0,944	0,931	0,939

Fonte: Elaborada pelo autor (2023).

Figura 22: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça em poses frontais.



Fonte: Elaborada pelo autor (2023).

ainda existe um problema de desbalanceamento de dados. Dessa forma, em todas as validações cruzadas, a classe Holandês possui acurácias melhores que a classe girolando, o que é representado pelos valores altos para o desvio padrão dessas acurácias por classe. A Figura 21 evidencia essas características e mostra como existe um número considerável de amostras da classe Girolando que são preditas incorretamente como sendo da classe Holandês.

Sobre a Tabela 15, nota-se que a classe Holandês tem resultados bem superiores em comparação com a classe Girolando. Por exemplo, a diferença dos valores de *recall* e F-2 *Score* entre as duas classes é consideravelmente grande. Além disso, chama a atenção o fato da classe Holandês possuir uma precisão e um *recall* tão altos.

4.2.7 Classificação de raça em poses laterais

Por fim, para o problema da classificação de raça a partir de imagens somente de animais em poses laterais, o experimento com o melhor resultado utilizou *batch size* igual a 32, taxa de aprendizagem igual a $0,01$, o treinamento durou 200 épocas e, para a normalização das imagens de entrada, foram utilizados uma média e um desvio padrão de $0,5$.

A Tabela 16 mostra os resultados obtidos com as 3 validações cruzadas do melhor modelo, contendo o desvio padrão (dp) e média das acurácias gerais e por classe. A Tabela 17 exibe as métricas de precisão, *recall*, F_1 *score* e F_2 *score* para a segunda validação cruzada do experimento, pois foi aquela que atingiu a maior acurácia geral. Já a Figura 22 apresenta a matriz de confusão gerada a partir da segunda validação cruzada.

Tabela 16 – Resultados obtidos para o problema de raça em poses laterais.

V.C.	**Acurácia das Classes (%)**		**dp**	**Acurácia geral (%)**
	Girolando	**Holandês**		
1	62,99	75,14	4,98	70,07
2	65,35	83,05	7,26	75,66
3	61,42	76,84	6,32	70,39
Média	63,25	78,34	6,19	72,04
dp	1,62	3,40	0,73	2,56

Fonte: Elaborada pelo autor (2023).

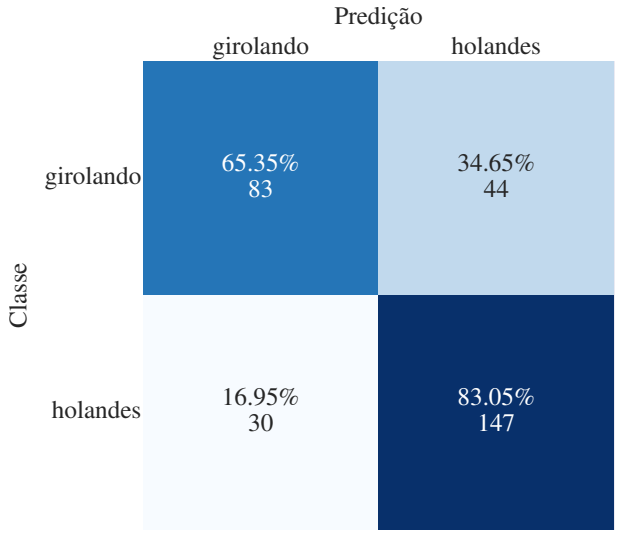
Tabela 17 – Métricas de precisão, *recall*, F_1 *score* e F_2 *score* da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça em poses laterais.

Classe	**Precisão**	***Recall***	F_1 ***score***	F_2 ***score***
Girolando	0,734	0,653	0,691	0,668
Holandês	0,770	0,830	0,799	0,818

Fonte: Elaborada pelo autor (2023).

A partir da Tabela 16, é possível constatar que a classe Girolando possui uma acurácia menor que a classe Holandês para todas as validações cruzadas e que existe um desvio padrão considerável das acurácias por classe de cada validação cruzada. Os valores das acurácias gerais são menores quando comparados com os resultados de outros

Figura 23: Matriz de confusão da validação cruzada que gerou a maior acurácia geral para o problema de classificação de raça em poses laterais.



Fonte: Elaborada pelo autor (2023).

problemas, visto que a melhor validação cruzada atinge apenas $75,66\%$. Dessa forma, nota-se que, assim como para o problema de raça em vacas, a classificação de raça em animais com pose lateral pode ser mais difícil dada a atual distribuição do *dataset*. A Figura 22 mostra que existe um número considerável de amostras da classe Girolando que são preditas como sendo da classe Holandês.

A Tabela 17 evidencia que as duas classes possuem precisões inferiores a 80% e o *recall* da classe Girolando é consideravelmente menor que o *recall* da classe Holandês. As métricas de F_1 *Score* e F-2 *Score* mostram como a classe Holandês tem resultados melhores que a classe Girolando.

5 CONCLUSÕES

Existem muitos casos de sucesso de métodos que utilizam aprendizado profundo e são aplicados a problemas de Visão Computacional na pecuária. Nesse contexto, este trabalho apresentou três modelos para classificação de faces de bovinos por categoria, pose e raça. Com isso, abrem-se muitas possibilidades de utilização de tais modelos para o uso em fazendas inteligentes através da integração com outras aplicações como, por exemplo, um sistema de reconhecimento facial em bovinos ou um sistema que avalia o estado de saúde dos animais da fazenda.

Destaca-se que neste trabalho foi criado um *dataset* que contém diversas imagens de bovinos com características variadas e em diferentes ambientes. As imagens foram coletadas por um especialista e possuem rótulos com algumas informações, o que pode ser muito útil para a utilização desse conjunto de dados em outros trabalhos de aprendizado de máquina.

A partir de uma rede conhecida, foram feitos diversos experimentos e os parâmetros foram ajustados para cada modelo de rede neural desenvolvido. Os modelos gerados para os problemas de classificação de categoria e pose alcançaram bons resultados, tendo ambos atingido acurácias gerais acima de 90%. Já o problema de classificação de raça se mostrou mais desafiador. Por isso, analisou-se também a combinação entre raça e outros atributos. A partir dessas combinações, verificou-se que os casos das classificações de raça de vacas e de animais em pose lateral não atingiram acurácias gerais tão altas, o que possivelmente está relacionado a características do próprio conjunto de dados, como foi discutido na Seção 4.2. Outras causas que podem explicar esse comportamento são o pequeno número de imagens no conjunto de treinamento, uma baixa representatividade das imagens em geral e a utilização de uma rede que pode ser simples para tratar problemas desse tipo.

De toda forma, este trabalho mostra evidências que os problemas de categoria e pose podem ser resolvidos por um modelo simples de rede neural convolucional e, com isso, cada um deles pode ser combinado para a solução de problemas mais complexos que conseguem se beneficiar dessas informações.

Ao longo do desenvolvimento do trabalho, notou-se também como as características do *dataset* impactam significativamente os resultados obtidos. Dessa forma, é muito importante trabalhar para uma ampliação do conjunto de dados atual e buscar novas imagens que tragam um maior equilíbrio entre todas as classes. Especificamente para o problema de classificação de raça, observou-se que existe um elemento de composição genética nos bovinos que pode dificultar a classificação correta da raça de certos animais. Isso abre uma oportunidade de separar o *dataset* em classes que consideram o grau de sangue dos bovinos e propor modelos de classificação com essa finalidade em trabalhos futuros.

Além disso, os modelos de classificação desenvolvidos neste trabalho podem ser aperfeiçoados a partir da variação de diversos parâmetros de treinamento e até mesmo através da utilização de outras redes neurais convolucionais presentes na literatura. Uma outra possibilidade que não pôde ser totalmente explorada para todos os problemas abordados é o aumento na resolução das imagens de entrada.

REFERÊNCIAS

BASGALUPP, M. P. *Algoritmos genéticos para seleção de atributos em problemas de classificação de processos de negócio*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio Grande do Sul, 2007.

BERNARDI, A. d. C. et al. Potencial de uso das tecnologias de agricultura e pecuária de precisão e automação. *Embrapa Pecuária Sudeste-Documentos (INFOTECA-E)*, São Carlos, SP: Embrapa Pecuária Sudeste, 2017.

BEZERRA, E. Introdução à aprendizagem profunda. *Artigo–31º Simpósio Brasileiro de Banco de Dados–SBBD2016–Salvador*, 2016.

CLARO, M. et al. Utilização de técnicas de data augmentation em imagens: Teoria e prática. *Sociedade Brasileira de Computação*, 2020.

DENG, J. et al. Imagenet: A large-scale hierarchical image database. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2009. p. 248–255.

ETHIRAJ, S.; BOLLA, B. K. Augmentations: An insight into their effectiveness on convolution neural networks. In: SINGH, M. et al. (Ed.). *Advances in Computing and Data Sciences*. Cham: Springer International Publishing, 2022. p. 309–322. ISBN 978-3-031-12638-3.

FERREIRA, F. C.; SIQUEIRA, K. B.; PEREIRA, L. G. R. A pecuária leiteira de precisão sob a ótica econômica. *Embrapa Gado de Leite-Artigo em periódico indexado (ALICE)*, Cadernos Técnicos de Veterinária e Zootecnia, n. 79, p. 137-145, 2015.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. [S.l.]: Editora Blucher, 2017.

GRIGORESCU, S. et al. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, Wiley Online Library, v. 37, n. 3, p. 362–386, 2020.

HU, H. et al. Cow identification based on fusion of deep parts features. *Biosystems Engineering*, Elsevier, v. 192, p. 245–256, 2020.

JERONYMO, V. *Implementando métodos de otimização para treinamento de redes neurais com pytorch*. Monografia (Trabalho apresentado como parte dos requisitos para obtenção de créditos na disciplina Projeto Supervisionado) — Universidade Estadual de Campinas (Unicamp), Campinas, 2019. Disponível em https://www. ime. unicamp. br/˜ mac/db/2019-2S-157490. Acesso em: 20 jan. 2023.

KHAIREDDIN, Y.; CHEN, Z. Facial emotion recognition: State of the art performance on fer2013. *arXiv preprint arXiv:2105.03588*, 2021.

KRIZHEVSKY, A.; HINTON, G. et al. Learning multiple layers of features from tiny images. Toronto, ON, Canada, 2009.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Ieee, v. 86, n. 11, p. 2278–2324, 1998.

LUO, C. et al. How does the data set affect cnn-based image classification performance? In: *2018 5th International Conference on Systems and Informatics (ICSAI)*. [S.l.: s.n.], 2018. p. 361–366.

MARENGONI, M.; STRINGHINI, S. Tutorial: Introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009.

PICCIALLI, F. et al. A survey on deep learning in medicine: Why, how and when? *Information Fusion*, Elsevier, v. 66, p. 111–137, 2021.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958.

SANTOS, H. G. d. et al. Machine learning para análises preditivas em saúde: exemplo de aplicação para predizer óbito em idosos de são paulo, brasil. *Cadernos de Saúde Pública*, SciELO Public Health, v. 35, p. e00050818, 2019.

SANTURKAR, S. et al. How does batch normalization help optimization? In: BENGIO, S. et al. (Ed.). *Advances in Neural Information Processing Systems*. [S.l.]: Curran Associates, Inc., 2018. v. 31.

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

SOUSA, J. R. de et al. Python e predição de dados usando redes neurais multicamadas. *Brazilian Journal of Development*, v. 6, n. 7, p. 54181–54185, 2020.

SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

SZELISKI, R. *Computer vision: algorithms and applications*. [S.l.]: Springer Science & Business Media, 2010.

TAKAHASHI, R.; MATSUBARA, T.; UEHARA, K. Ricap: Random image cropping and patching data augmentation for deep cnns. In: PMLR. *Asian conference on machine learning*. [S.l.], 2018. p. 786–798.

TIAN, M. et al. Automated pig counting using deep learning. *Computers and Electronics in Agriculture*, Elsevier, v. 163, p. 104840, 2019.

VIEIRA, Z. P.; BAUCHSPIESS, A. Implementação do servocontrole auto-sintonizado em tempo-real utilizando rede perceptron multicamadas. In: *Anais do 4 Congresso Brasileiro de Redes Neurais*. São José dos Campos, SP: CNRN, 1999. p. 308–313.

WATHES, C. M. et al. Is precision livestock farming an engineer's daydream or nightmare, an animal's friend or foe, and a farmer's panacea or pitfall? *Computers and electronics in agriculture*, Elsevier, v. 64, n. 1, p. 2–10, 2008.

WEBER, F. de L. et al. Recognition of pantaneira cattle breed using computer vision and convolutional neural networks. *Computers and Electronics in Agriculture*, Elsevier, v. 175, p. 105548, 2020.