UNIVERSIDADE DO VALE DO TAQUARI – UNIVATES
CURSO DE ENGENHARIA DE SOFTWARE

CONSTRUÇÃO DE UM CLASSIFICADOR DE PELAGENS DE FELINOS BASEADO EM REDES NEURAIS CONVOLUCIONAIS

Giovani Luis Stein

Lajeado, novembro de 2019

Giovani Luis Stein

CONSTRUÇÃO DE UM CLASSIFICADOR DE PELAGENS DE FELINOS BASEADO EM REDES NEURAIS CONVOLUCIONAIS

Trabalho de conclusão de curso apresentado no Centro de Ciências Exatas e Tecnológicas, da Universidade do Vale do Taquari – Univates, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Dr. Marcelo de Gomensoro Malheiros.

Lajeado, novembro de 2019

Dedico este trabalho majoritariamente a minha família e amigos, que sempre me apoiaram em todos os momentos da graduação desde o início da mesma.

RESUMO

As redes neurais têm sido muito utilizadas em pesquisas e no desenvolvimento de produtos nos últimos anos, mesmo que tais técnicas já existam há décadas na literatura. Com o advento de hardware massivamente paralelo e de custo acessível, somente agora o emprego de redes neurais conseguiu atingir níveis de precisão acima de seres humanos. Redes neurais para a classificação de imagens tipicamente seguem a arquitetura chamada rede neural convolucional (RNC), sendo hoje a abordagem mais utilizada. o emprego de bibliotecas de alto nível como o Keras permite um ciclo rápido de experimentação, que é ancorado em bibliotecas de baixo nível como o Tensorflow, que garantem a eficiência dos processos de treino e avaliação. este trabalho tem como objetivo analisar o problema de classificação automática de fotos de algumas espécies de felinos, usando com base as diferenças de padrão em suas pelagens. uma possível aplicação seria o rastreamento de indivíduos em seu habitat natural, para auxiliar em sua preservação natural. Este estudo também pode auxiliar na compreensão de quais são os mecanismos biológicos que criam tais tipos característicos de pigmentação, pois permite construir uma medida quantitativa de similaridade entre padrões biológicos. Para tanto, este trabalho descreve a construção e treinamento de uma rede neural convolucional para a classificação de imagens de quatro espécies de grandes felinos, aplicado tanto em imagens reais quanto em imagens sintéticas.

Palavras chave: Inteligência Artificial. Aprendizado de Máquina. Redes Neurais. Redes Neurais Convolucionais. Classificação de Imagens.

ABSTRACT

Neural networks have been widely used in research and product development in recent years, even though such techniques have existed for decades in the literature. with the advent of massively parallel and affordable hardware, only now has the use of neural networks been able to reach levels of precision beyond humans. Neural networks for image classification typically follow the convolutional neural network (CNN) architecture, being today the most widely used approach. the use of high level libraries such as Keras allows for a rapid experimentation cycle, being anchored in low level libraries like Tensorflow, which ensures the efficiency of training and evaluation processes. This work aims to analyze the problem of automatic classification of photos of some feline species, based on the pattern differences in their fur. A possible application would be tracking individuals in their natural habitat ot help their natural preservation. this study can also help in understanding which biological mechanisms create those characteristic pigmentation types, as it allows the construction of a quantitative measure of similarity between biological patterns. Therefore, this work describes the construction and training of a convolutional neural network for the classification of images of four species of big cats, applied to both real and synthetic images.

Keywords: Artificial Intelligence. Machine Learning. Neural Networks. Convolutional Neural Networks. Classification of Images.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CG	Computer Graphics
CNN	Convolutional Neural Network
GPU	Graphics Processing Unit
IA	Inteligência Artificial
LSTM	Long Short-Term Memory
ML	Machine Learning
RAM	Random Access Memory
RCN	Recurrent Convolutional Networks
RD	Reação-Difusão
RL	Reinforcemente Learning
RNA	Rede Neural Artificial
RNC	Rede Neural Convolucional

SUMÁRIO

1 INTRODUÇÃO

Com a evolução da Inteligência Artificial (IA) e dos métodos de Aprendizagem de Máquina (em inglês, Machine Learning - ML), muitas pesquisas estão sendo feitas nessa área. Em consequência, aplicações dos mais variados tipos e finalidades estão em desenvolvimento.

Devido ao aumento de desempenho e barateamento de custo de hardware massivamente paralelo, os resultados obtidos têm melhorado significativamente, fazendo que aplicações anteriormente limitadas hoje ultrapassem a capacidade humana. Com isso, a implementação e o treino de aplicações de Aprendizado de Máquina visando a solução de um determinado problema se tornaram muito mais simples e viáveis, por conta da diversidade de ferramentas disponíveis e da relativa facilidade de acesso a hardware computacionalmente poderoso.

O Aprendizado de Máquina pode ser utilizado para a resolução dos mais variados problemas: desde a classificação de um conjunto de dados até a predição de novas informações através de séries históricas.

O uso de redes neurais para classificação de dados tem sido comum nos últimos anos, em especial pelo desenvolvimento de bibliotecas como Keras e TensorFlow, que facilitaram a utilização de várias técnicas de aprendizado automatizado. O grande fator que possibilitou o uso em larga escala deste tipo de aprendizagem foi o *deep learning*, ou aprendizado profundo, que faz o uso de redes neurais com um grande número de camadas. Isso permite construir modelos mais precisos, fazendo assim uma análise mais profunda dos dados. As técnicas de *deep*

learning são muito utilizadas, por exemplo, em aplicações como reconhecimento de fala, visão computacional e processamento de linguagem natural.

A classificação de imagens utilizando redes neurais é uma área que sofreu um grande crescimento nos últimos anos, em que muitos projetos de pesquisa são feitos visando a construção de redes neurais mais eficientes. Além disso, competições de criação e desenvolvimento destas redes com premiações acontecem em todo mundo, sendo patrocinadas por grandes empresas e atraindo novos talentos para o campo de Aprendizado de Máquina.

As Redes Neurais Convolucionais (em inglês, Convolutional Neural Networks - CNN) se organizam em torno da estrutura espacial de uma imagem bidimensional. Por conta disso, as aplicações baseadas em classificação de imagens tornam-se mais rápidas durante o processo de treinamento, possibilitando o uso em redes neurais com um número maior de camadas em relação à abordagem tradicional de redes neurais densas. Atualmente as CNNs são utilizadas na maioria das aplicações de reconhecimento de imagens.

A precisão com que as Redes Neurais Convolucionais conseguem classificar as imagens muitas vezes consegue alcançar a precisão humana, isso é um fator muito importante e demostra que hoje em dia muitas aplicações podem, de alguma maneira, utilizar desta tecnologia para realizar tarefas de reconhecimento e classificação.

O presente trabalho tem como objetivo utilizar uma Rede Neural Convolucional para fazer a classificação de imagens da pelagem de algumas espécies de grandes felinos. A solução desenvolvida utiliza a biblioteca de alto nível Keras para a implementação da rede neural, e um *dataset* de fotos reais para o processo de treinamento.

Para validação dos resultados, foram utilizadas novas imagens dos animais, fotos de detalhes da pelagem das mesmas espécies e imagens artificiais de pelagem sintetizadas por técnicas de Computação Gráfica (CG), para validar a eficiência e precisão da rede neural construída.

1.1 Objetivos

O objetivo geral deste trabalho é investigar se é possível utilizar uma rede neural para fazer a classificação de imagens de pelagem de animais, tanto de pelagens reais quanto de pelagens geradas artificialmente. Para isso foi implementada uma Rede Neural Convolucional multicamada, onde se deu o processo de treinamento utilizando um *dataset* de fotos reais.

Como objetivos específicos deste trabalho, podemos listar:

- Pesquisar e fazer a revisão bibliográfica dos conceitos fundamentais, também buscando por trabalhos relacionados.
- Pesquisar técnicas relacionadas ao desenvolvimento de Redes Neurais Convolucionais.
- Definir as tecnologias utilizadas no trabalho.
- Analisar arquiteturas de Redes Neurais Convolucionais.
- Criar uma base inicial de fotos e fazer uma classificação manual.
- Preparar a base de imagens para o treinamento da rede neural.
- Efetuar e avaliar o treinamento supervisionado da rede neural.
- Realizar testes e ajustes no modelo treinado de rede neural para identificar os parâmetros que levam a um melhor resultado.

1.2 Estrutura do trabalho

Este trabalho está estruturado em seis capítulos. O primeiro capítulo contém a introdução ao problema proposto e demais objetivos do trabalho. O segundo capítulo compreende as referências sobre as áreas de conhecimento abordados. O terceiro apresenta trabalhos relacionados, enquanto o quarto capítulo consiste no processo metodológico, na seleção das tecnologias utilizadas e na definição da arquitetura da rede neural. O quinto capítulo apresenta os principais parâmetros estudados e a

execução dos testes. O sexto capítulo analisa os resultados obtidos nos testes e traça as conclusões finais deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo são definidos os conceitos principais para a compreensão deste trabalho. Os tópicos a seguir abordam a Inteligência Artificial e suas definições, descrevem a estrutura e o funcionamento das redes neurais e também discorrem brevemente sobre o histórico e as tecnologias relacionadas a *deep learning*. Além disso, é feita uma introdução ao processamento de imagens e aos padrões de pigmentação de animais.

2.1 Inteligência Artificial

Segundo Russel e Norvig (2013), a Inteligência Artificial (IA) pode ter várias definições, podendo ser dividida em quatro abordagens:

- **Pensando como um humano:** “O novo e interessante esforço para fazer computadores pensar.” (Haugeland, 1985 apud Russel e Norvig, 2013).
- **Pensando racionalmente:** “O estudo das faculdades mentais pelo uso de modelos computacionais.” (Charniak e McDermott, 1985 apud Russel e Norvig, 2013).
- **Agindo como seres humanos:** “Arte de criar máquinas que executam funções que exigem inteligência quando executados por pessoas.” (Kurzweil, 1990 apud Russel e Norvig, 2013).

- **Agindo racionalmente:** “Inteligência Computacional é o estudo do projeto de agentes inteligentes.” (Poole et al., 1998 apud Russel e Norvig, 2013).

De acordo com Luger (2014), a Inteligência Artificial pode ser definida como a área da Ciência da Computação que estuda a automação do comportamento inteligente, enfatizando que a IA deve ser baseada em princípios teóricos e possuir aplicações sólidas. O autor também salienta que entre os princípios se incluem as estruturas de dados usados na exibição do conhecimento, os algoritmos utilizados para a aplicação do conhecimento obtido e as linguagens de programação utilizadas no seu desenvolvimento.

2.2 Aprendizado de Máquina

Segundo Artero (2009), a capacidade de aprendizado devido à experiência é importante para o desenvolvimento das espécies. Desse modo, a implementação de técnicas de aprendizado automático tem se tornado uma das áreas de grande interesse na IA. O autor complementa ainda que o Aprendizado de Máquina (em inglês, Machine Learning - ML) pode ser definido como a alteração do programa de controle com base em suas próprias experiências.

De acordo com Lopes, Santos e Pinheiro (2014), o Aprendizado de Máquina visa estudar e melhorar métodos computacionais para a construção de sistemas capazes de obter conhecimento automaticamente.

2.2.1 Aprendizado Supervisionado

Conforme Artero (2009), para executar um aprendizado supervisionado é utilizado um conjunto de valores (ou atributos de entrada) e sua respectiva saída. As entradas são inerentes aos objetos que se deseja conhecer, enquanto as saídas representam as classes que se espera obter.

Em conformidade com Coppin (2013), redes neurais que utilizam o aprendizado supervisionado aprendem quando recebem dados para o treinamento que já estejam pré-classificados.

Segundo Braga, De Carvalho e Ludermir (2011), o aprendizado supervisionado possui duas formas de implementação. A primeira é conhecida como offline, sendo usada quando os dados do conjunto não sofrem alteração. Ou seja, depois de se conseguir encontrar uma solução a mesma permanece fixa. Na segunda forma, conhecida como online, os dados do conjunto sofrem alteração, ou seja, a rede neural está em um processo contínuo de ajustes.

2.2.2 Aprendizado Não Supervisionado

Coppin (2013) enfatiza que o treinamento não supervisionado é capaz de fazer uma rede aprender sem nenhum tipo de intervenção humana, sendo utilizado em situações em que os dados precisam ser classificados ou agrupados em um determinado conjunto, mas onde as classes utilizadas no treinamento não são previamente conhecidas.

O método de aprendizado não supervisionado não será utilizado no trabalho, mas técnicas como essa são imprescindíveis para áreas como mineração de dados e análise de tendências.

2.3 Redes Neurais

Segundo Haykin (2001), uma rede neural pode ser definida como um processador massivamente paralelo, distribuído em muitas pequenas unidades elementares de processamento, tendo a capacidade de armazenar conhecimento empírico e tornar este conhecimento apto para o uso. Muitas vezes também se usa o termo equivalente Rede Neural Artificial (RNA).

As redes neurais possuem muitas semelhanças com o cérebro humano, uma vez que o conhecimento humano é obtido através de um procedimento de aprendizagem. De forma análoga, as ligações entre os neurônios artificiais são utilizadas para guardar o conhecimento obtido no processo de aprendizagem. Conforme Haykin (2001), as propriedades que caracterizam uma rede neural são:

- **Não linearidade:** neurônios podem ser lineares ou não lineares, ou seja, podem ter vários sentidos e apresentar várias conexões e caminhos possíveis em uma rede.

- **Mapeamento de entrada-saída:** a rede neural aprende através de um processo de treinamento, utilizando exemplos pré-classificados e assim mapeia os resultados de entrada com os resultados de saída.
- **Adaptabilidade:** a rede neural armazena o conhecimento obtido em seus pesos sinápticos, sendo capaz de se ajustar caso os dados de treinamento se modifiquem durante o processo de aprendizagem.
- **Resposta a evidências:** a rede neural pode dar conclusões sobre um determinado padrão e informar um grau de confiança sobre o mesmo.
- **Informação contextual:** o conhecimento proveniente do processo de aprendizagem fica armazenado na estrutura da rede.
- **Tolerância a falhas:** redes neurais podem possuir mecanismos que reduzam a quantidade de neurônios, caso aconteça algum tipo de falha na rede, mantendo assim sua performance.

2.3.1 Construção de uma Rede Neural

Consoante a De Medeiros (2018), a construção de uma rede neural depende de quatro fundamentos:

- **Número de camadas:** as redes neurais possuem no mínimo uma camada de entrada, por onde adquirem as informações das amostras, e uma camada de saída, que retorna valores mapeados para os grupos utilizados no treinamento da mesma.
- **Quantidade de neurônios em cada camada:** as redes neurais não possuem uma quantidade fixa de neurônios por camada. No caso da camada de entrada, esta tipicamente tem o mesmo número de neurônios que o número de entradas das amostras do conjunto de treinamento.
- **Tipo de função de transferência:** define qual a forma de ativação do neurônio.

- **Método de treinamento:** é o algoritmo que efetua a atualização dos pesos das sinapses, e, portanto, permite o treinamento da rede neural.

2.3.2 Neurônio Artificial

Segundo Haykin (2001), o neurônio é uma de muitas unidades de processamento encontradas em uma rede neural. O autor também salienta os três elementos básicos de um neurônio:

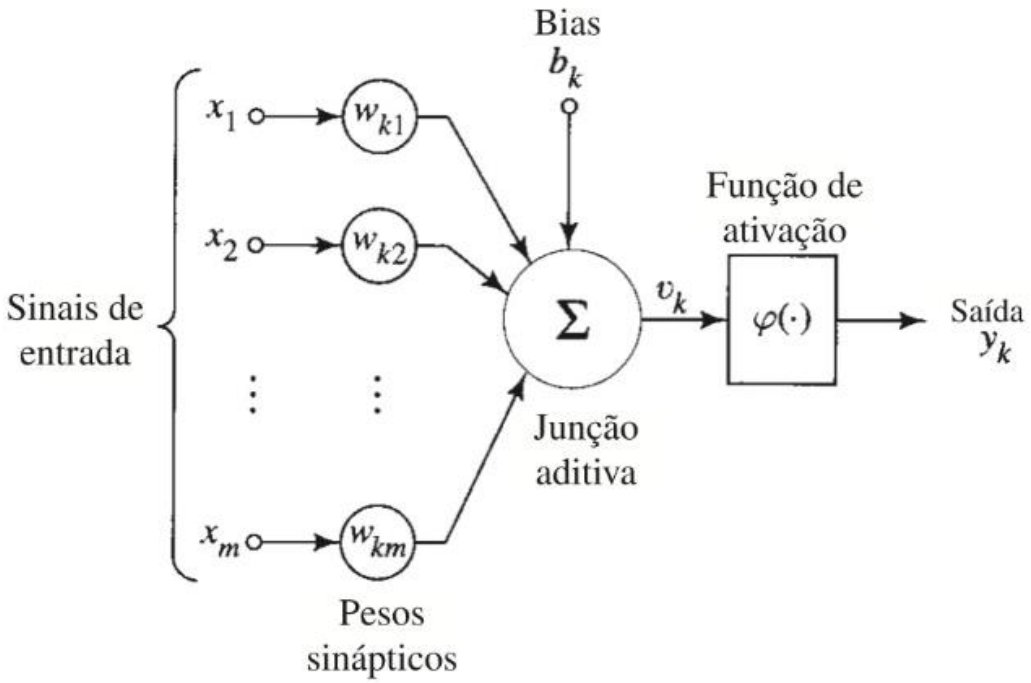
- **Conjunto de sinapses:** também conhecidos como elos de conexão, possuindo um peso ou força própria. Um sinal x_j na entrada da sinapse j que está conectada a um neurônio k é multiplicado pelo peso sináptico w_{kj}. O peso sináptico de um neurônio está em um intervalo que inclui tanto valores negativos como valores positivos.

- **Somador:** soma os sinais de entrada, equilibrados pelas respectivas sinapses do neurônio.

- **Função de ativação:** restringe a amplitude da saída de um neurônio. Esta limita o sinal de saída a um valor finito, sendo que o intervalo normalizado pode ser um intervalo unitário fechado [0,1] ou ainda o intervalo [-1,1].

A Figura 1 exibe o modelo de um neurônio não-linear, com suas respectivas entradas, funções e saídas.

Figura 1 - Modelo não-linear de um neurônio.



Fonte: Haykin (2001, p. 36).

Em conformidade com Coppin (2013), as redes neurais são construídas em analogia ao cérebro humano, e, portanto, contam com uma grande quantidade de neurônios artificiais. O autor enfatiza que um neurônio artificial geralmente possui um número menor de conexões se comparado a um neurônio biológico, de forma que as redes neurais são muito menores em número de neurônios do que o cérebro humano.

2.3.3 Perceptron

Segundo Coppin (2013), o *perceptron* é um neurônio elementar, utilizado para fazer a classificação de suas entradas em uma de duas categorias. O *perceptron* pode ter várias entradas que podem ser estruturadas em uma grade, podendo esta ser usada na representação de uma imagem ou campo de visão, e assim, os *perceptrons* podem ser usados para efetuar a classificação de imagens.

2.3.4 Redes Neurais Multicamadas

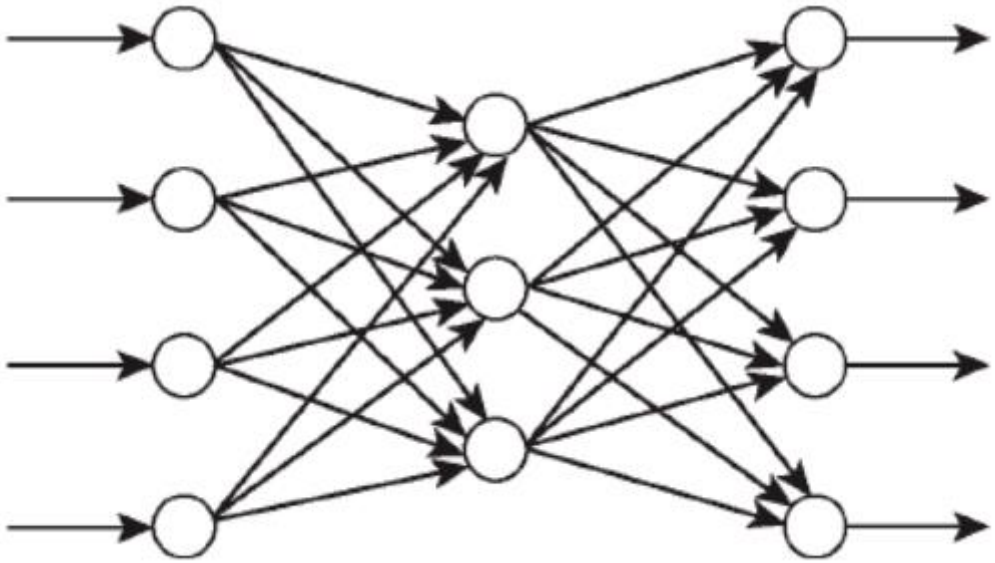
Coppin (2013) diz que os neurônios podem estar conectados e organizados em camadas. Estas redes multicamadas são capazes de modelar funções que não

sejam linearmente separáveis. Uma rede neural multicamada é tipicamente composta de:

- **Camada de entrada:** cada neurônio desta camada recebe somente um sinal de entrada. Em alguns casos os nós dessa camada não são neurônios, mas apenas levam os valores das entradas para a próxima camada.
- **Camada oculta:** uma rede neural pode conter uma ou várias camadas ocultas, onde estão localizados os neurônios que efetivamente realizam o trabalho.
- **Camada de saída**: responsável pela última etapa do processamento, que retorna os sinais de saída.

A Figura 2 demostra uma rede neural multicamada.

Figura 2 – Rede neural Multicamadas com três camadas



Fonte: Coppin (2013, p. 262).

2.3.5 Algoritmo de *backpropagation*

Fernandes (2003) diz que, ao utilizar o algoritmo de *backpropagation*, uma rede neural executa dois passos. Inicialmente uma entrada é fornecida para a camada de entrada da rede. Então o processo continua passando através das camadas ocultas da mesma rede, até que se chegue à camada de saída, onde é construída a resposta. No próximo passo, o resultado gerado na camada de saída é

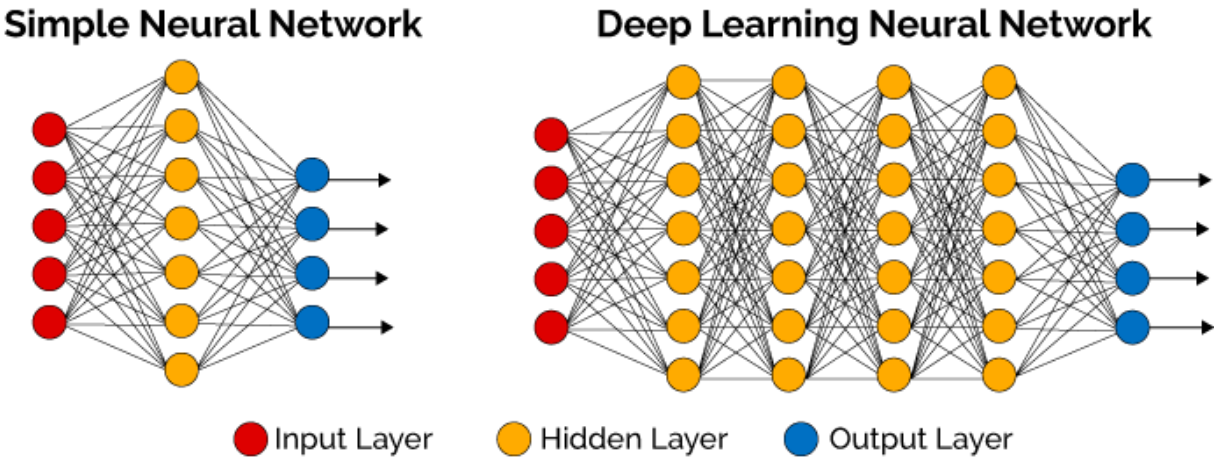
comparado com o resultado que se espera, de acordo a classe de validação fornecida, verificando se ele é válido ou não. Caso não seja válido, uma taxa de erro é calculada. Este erro é então propagado pela rede em sentido inverso, a partir da saída em direção à entrada, de forma que os pesos das conexões das camadas ocultas da rede sofram modificações.

De acordo com Rosa (2011), o algoritmo de *backpropagation* normalmente atualiza os valores dos pesos, depois de verificar todos os valores de entrada e saída. Após o ajuste nos pesos conclui-se o processo chamado de **época**. O autor também salienta que um treinamento utilizando o algoritmo requer muitas épocas para conseguir resultados satisfatórios.

2.4 Deep Learning

Consoante com Ponti e Da Costa (2017), métodos de *deep learning* são utilizados principalmente em problemas de classificação, que buscam descobrir um determinado modelo usando um certo grupo de dados e ainda um método para orientar o aprendizado. Os autores também complementam que ao concluir a fase de aprendizado, temos uma função que pode receber uma entrada e retornar a qual classe a mesma pertence. A Figura 3 compara uma rede neural multicamada simples e uma rede neural do tipo *deep learning*.

Figura 3 - Rede neural multicamada simples e rede neural profunda.



Fonte: Data Science Academy (2017).

2.4.1 Rede Neural Convolucional

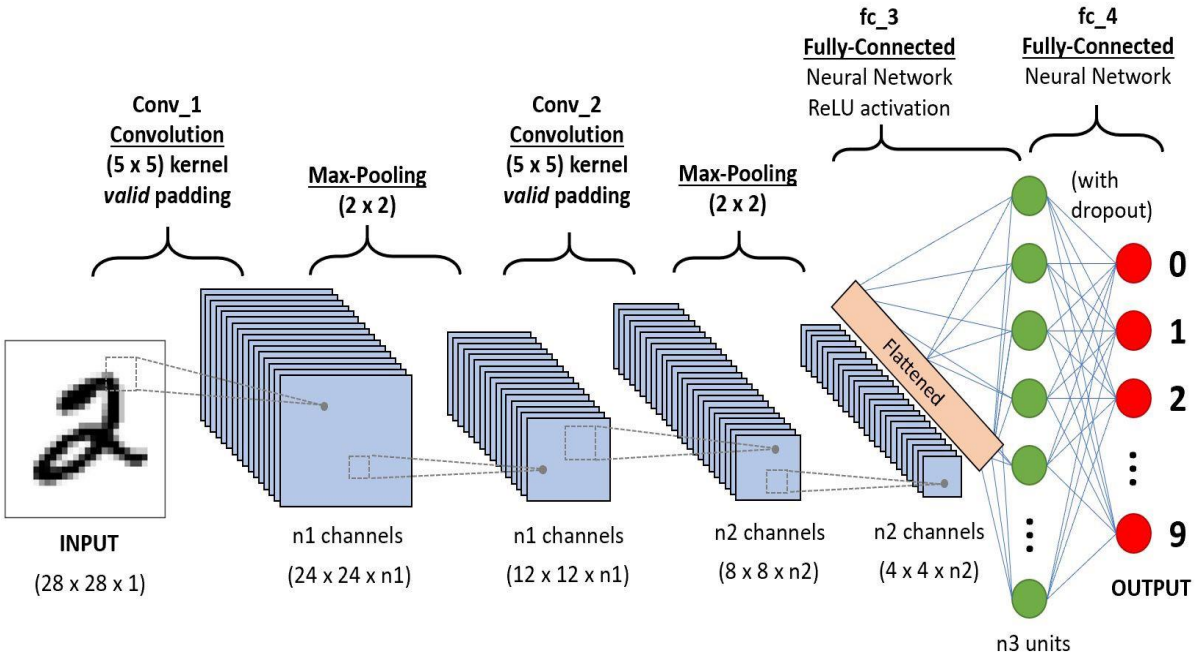
Em concordância com Ponti e Da Costa (2017), CNNs representam o modelo de rede neural para deep learning mais utilizado atualmente. Este modelo utiliza camadas convolucionais que processam os dados de entrada levando em conta campos receptivos locais. Essas camadas também possuem operações como o pooling, que reduz a dimensionalidade espacial das representações.

Em conformidade com Data Science Academy (2017) uma Rede Neural Convolucional é um algoritmo de aprendizado profundo que pode captar um input de entrada, atribuir importância (pesos) a vários aspectos desta entrada e ser capaz de diferenciar um do outro. O pré-processamento exigido em uma CNN é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos mais antigos os filtros são feitos manualmente e com treinamento suficiente, as CNNs têm a capacidade de aprender esses filtros ou características.

Ponti e Da Costa (2017) complementam que nas camadas convolucionais cada neurônio é um tipo de filtro local. O efeito combinado em toda a rede corresponde à aplicação desse filtro em toda a imagem.

A Figura 4 demostra o funcionamento de uma CNN genérica.

Figura 4 - Funcionamento de uma CNN



Fonte: A Comprehensive Guide to Convolutional Neural Networks. Disponível em: < https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Acesso em 12 dez. 2019

2.5 Processamento de Imagens

Nascimento, Guimarães e Shiguemori (2012) dizem que as técnicas para processamento de imagens são estudadas em várias áreas. Muitas dessas técnicas consistem em extrair determinadas informações e características de uma imagem através de sucessivas transformações, de forma que o resultado seja adequado para uma determinada aplicação.

Em concordância com Marques Filho e Vieira Neto (1999), o processamento de imagens tem despertado grande interesse por viabilizar diversas aplicações que utilizam imagens para obter ou extrair informações, principalmente em análises feitas via computador.

3 TRABALHOS RELACIONADOS

Neste capítulo se encontram alguns trabalhos relacionados ao tema proposto, utilizando diferentes tecnologias para fazer classificação automática de imagens.

Crall et al. (2013) apresentam o HotSpotter, que é um algoritmo que faz a identificação das populações de animais de acordo com seu padrão da pelagem. O sistema utiliza um banco de dados, com imagens rotuladas onde cada rótulo identifica um animal individual presente na imagem. Para fazer a identificação dos animais o algoritmo combina as imagens utilizadas na consulta com as imagens do banco de dados, classificando as imagens por similaridade e atribuindo uma pontuação. Os autores salientam que o algoritmo calcula os padrões típicos da pelagem de animais, comparando estes padrões com as imagens do banco de dados, a fim de identificar um animal. De acordo com o artigo, os resultados finais estavam entre 95% a 99% de precisão.

Andrew, Greatwood e Burghardt (2017) falam sobre a utilização de drones para a identificação de gado holandês. Para isso os cientistas utilizaram uma rede neural e técnicas de *deep learning*. Dois *datasets* foram utilizados no treinamento da rede, um com 940 imagens coloridas do revestimento dorsal de 89 indivíduos distintos. Outro *dataset* consiste em 34 vídeos com 20 segundos cada do rebanho, capturados ao ar livre. Após um tratamento nos vídeos um novo conjunto foi gerado com um total de 23 indivíduos e 160 vídeos.

Naquele trabalho foram utilizados conceitos de redes neurais LSTM (Long Short-Term Memory) e técnicas de CNN para o processamento de vídeo. Além de imagens e técnicas de Redes Convolucionais Recorrentes (em inglês, Recurrent Convolutional Networks - RCN) para processamento espaço-temporal. Os *drones* são utilizados para capturar as imagens em tempo real do gado e depois estas imagens são utilizadas para a identificação individual dos animais ao ar livre. De acordo com os autores, após o processo de treinamento com 1000 épocas e eventuais ajustes na rede neural utilizada, a precisão da rede para reconhecimento de vídeo foi de mais de 98%.

Ge et al. (2015) falam sobre a classificação de imagens que possuem categoria refinada, ou seja, possuem diferenças sutis na aparência geral entre as várias classes (baixa variação entre classes) e grandes variações de pose e aparência na mesma classe (grande variação intraclasse). Os autores utilizaram uma CNN para primeiramente dividir as classes em subconjuntos visualmente semelhantes e depois aprender recursos específicos do domínio para cada subconjunto. No trabalho foi utilizado uma técnica de aprendizado de máquina chamada de transferência de aprendizado, que consiste em utilizar os vetores de características gerados por uma rede neural previamente treinada, com seus pesos sinápticos configurados para identificar e extrair padrões de um determinado conjunto de dados.

Para fazer a classificação dos subconjuntos de imagens visualmente semelhantes, foram comparadas a cor e a textura. Depois foi utilizado um algoritmo de agrupamento *k-means* para gerar os subconjuntos de imagens onde são extraídos os padrões. Após o treinamento a precisão alcançada pela rede foi de 77,5%.

Pacheco (2016) apresenta o desenvolvimento de um classificador de imagens de espécies de peixes baseados em redes neurais. Para realizar o trabalho o autor utilizou as bibliotecas TensorFlow e Keras para desenvolver uma rede neural convolucional, com o objetivo de fazer a classificação de seis espécies diferentes de peixes. O autor salienta que para o experimento foram utilizadas oito classes, contendo um total de 3.777 amostras divididas de forma desigual entre as classes. Para a realização dos testes foram utilizados três modelos de arquitetura com

configurações distintas, 100 épocas de treinamento e uma matriz de entrada com o tamanho de 48 x 48 pixels.

Foram utilizados dois cenários de teste. No primeiro cenário, as imagens do *dataset* foram balanceadas, ou seja, divididas de forma igual entre as classes. No segundo cenário foi mantida a divisão original das imagens. Os resultados ficaram entre 70% e 80% de precisão quando utilizado o primeiro cenário, e de 98% para o segundo cenário.

Trnovszky et al. (2017) falam sobre um sistema de reconhecimento de animais, que visa a avaliação de métodos de classificação de imagens quando comparados a uma CNN. Para o desenvolvimento foi utilizado um *dataset* contendo cinco classes de treinamento (raposa, lobo, urso, porco e veado) com 100 imagens para cada classe. Cada imagem tinha tamanho de 150 x 150 pixels. As imagens foram tratadas de forma que ficassem alinhadas e normalizadas, com base nas posições dos olhos dos animais.

Naquele trabalho foram comparadas algumas técnicas para a classificação e reconhecimento de imagens: Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA) e Support Vector Machine (SVM). Estas três técnicas foram comparadas a uma CNN, utilizando várias configurações e distribuições das imagens no *dataset*. Os resultados em todos os cenários foram melhores ao utilizar uma CNN, que produziu resultados em torno de 98% de precisão, em quanto as demais técnicas utilizadas ficaram abaixo de 90%.

Todos os trabalhos relacionados que foram citados neste capítulo possuem semelhanças ao trabalho proposto, pois utilizam na sua maioria redes neurais. Também empregam outras técnicas para fazer o reconhecimento de imagens de animais, tanto utilizando sua pelagem característica como pela imagem do animal propriamente dito.

4 MATERIAIS E MÉTODOS

Neste capítulo são discutidos os métodos utilizados no presente trabalho, incluindo as tecnologias utilizadas e o descrevendo o desenvolvimento do mesmo.

4.1 Métodos

Esta seção está subdividida em quatro partes: método de pesquisa, modo de abordagem, objetivos de pesquisa e procedimentos técnicos.

4.1.1 Métodos de Pesquisa

Em concordância com Marconi e Lakatos (2008), o conhecimento científico se distingue de outros conhecimentos por se tratar de fatos reais, pois suas suposições são testadas e sua autenticidade é colocada à prova através de experiências. Este trabalho utiliza esta abordagem para realizar procedimento de testes ou experimentação de métodos científicos sobre as amostras e com isso, obter resultados concretos.

4.1.2 Modo de Abordagem

O modo de abordagem deste trabalho é qualitativo. Segundo Dalfovo, Lana e Silveira (2008), na pesquisa qualitativa o problema tem sua complexidade descrita, e

torna-se necessário classificar e entender os processos para obter-se a compreensão das particularidades dos componentes dos grupos. Tal modo de abordagem aplica-se ao trabalho, pois existe a necessidade de validação dos objetivos apresentados no capítulo inicial, que ocorrerá por meio de testes durante o processo de desenvolvimento.

4.1.3 Objetivos da Pesquisa

A metodologia quanto ao objetivo será exploratória, pois para Cervo, Bervian e da Silva (2007), a pesquisa exploratória é comumente o primeiro passo na pesquisa por experiência. Consoante a isso, Castro (2006) ressalta que na abordagem exploratória não há revisão sistemática prévia e nem discussões de teorias alternativas. Devido a isso o objetivo de pesquisa será exploratório, pois o trabalho busca avaliar a possibilidade da classificação de determinados tipos de imagens.

4.1.4 Procedimentos Técnicos

A metodologia quanto ao procedimento técnico será a pesquisa bibliográfica, já que para Gil (2002), a pesquisa bibliográfica é composta por livros e artigos científicos, fundamentada em materiais já criados. O trabalho desenvolvido utiliza a pesquisa bibliográfica para a fundamentação da teoria e também no desenvolvimento do projeto, utilizando métodos e tecnologias encontrados durante o levantamento inicial.

4.2 Tecnologias

Nesta seção são apresentadas as tecnologias utilizadas, incluindo configurações e demais caraterísticas do processo de desenvolvimento.

4.2.1 Python

O Python é uma linguagem de programação criada por Guido van Rossum no início da década de 1990. Esta foi projetada para tornar programas bastante legíveis. Também possui uma biblioteca muito grande de funcionalidades, facilitando assim o desenvolvimento de aplicações mais complexas, ainda que com um código de aparência simples (PERKOVIC, 2016).

4.2.2 TensorFlow

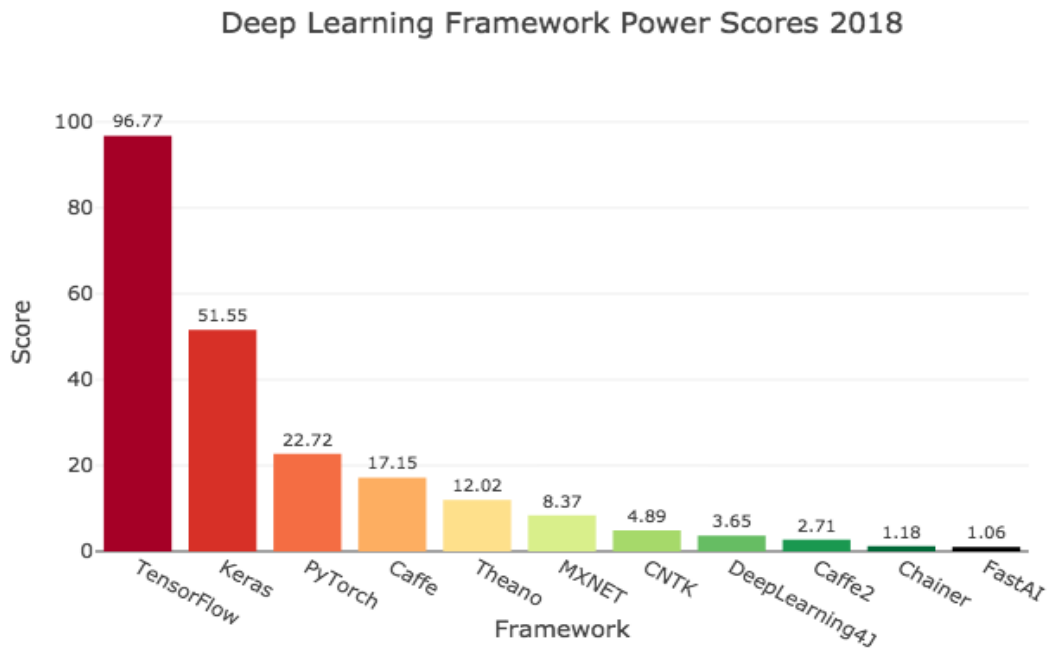
O TensorFlow é uma biblioteca de código aberto para computação numérica, utilizando grafos de fluxo de dados. Os nós dos grafos representam operações matemáticas e as arestas representam matrizes de dados multidimensionais. Esse tipo de arquitetura permite que ele rode em quase todas as plataformas, seja uma CPU ou uma GPU, em uma máquina desktop, um servidor ou um dispositivo móvel. Foi desenvolvido com o objetivo de realizar aprendizado de máquina e pesquisa em redes neurais profundas (TENSORFLOW, 2019).

4.2.3 Keras

O Keras é um *framework* de aprendizado profundo, que oferece várias APIs de uso simples que reduzem o número de ações do usuário, fornecendo feedback sobre erros e problemas comuns. Ele possui integração com a biblioteca TensorFlow, que opera em um nível mais baixo. Por conta da qualidade desta integração, o Keras é considerado o frontend oficial do TensorFlow (KERAS, 2018).

A Figura 5 mostra a classificação de vários frameworks de *deep learning*.

Figura 5 – Classificação de *frameworks* de aprendizagem profunda.



Fonte: Keras (2018).

4.2.4 Jupyter

O Jupyter é uma IDE de desenvolvimento que permite a criação e compartilhamento de documentos com código fonte dividido em blocos. Possui suporte a várias linguagens, incluindo Python, R, Julia e Scala. Permite também a integração com diversas bibliotecas de aprendizado de máquina, de *big data*, de análise estatística e de visualização científica (JUPYTER, 2019).

4.2.5 Google Colaboratory

O Google Colaboratory é um ambiente de desenvolvimento *online* baseado no Jupyter, sendo que é executado diretamente em servidores na nuvem. Possui aceleração por GPU, CPU ou TPU. Além disso, possui várias bibliotecas pré-instaladas para facilitar o desenvolvimento.

Esta ferramenta se integra ao ambiente Google Docs, permitindo compartilhamento e colaboração simultânea entre diversos usuários.

4.2.6 Google Images Download

O Google Images Download é um *script* em Python, que utiliza o pacote Selenium para simular a interação de um navegador com um *site*, possibilizando fazer a pesquisa e o *download* automático de um grande número de imagens. Nele é possível fazer pesquisas avançadas passando parâmetros como tamanho da imagem, proporção e formato.

4.2.7 Scikit-image

O Scikit-image é uma biblioteca em Python utilizada para o processamento de imagens. Ela possui uma coleção vasta de algoritmos para os mais variados tipos de tratamentos em imagens.

4.2.8 OpenCV

É uma biblioteca focada nas áreas de visão computacional e aprendizado de máquina, sendo de código aberto. Esta biblioteca possui um conjunto extenso de mais de 2500 algoritmos especializados (OPENCV, 2019).

4.2.9 Módulos Utilizados

Os principais módulos de Python utilizados na montagem do modelo de treinamento e dos algoritmos de tratamento de imagens estão listados a seguir:

- **matplotlib**: biblioteca gráfica para gerar os gráficos de treinamento.
- **skimage**: biblioteca para processamento de imagens.
- **numpy**: biblioteca que permite trabalhar com arranjos, vetores e matrizes de N dimensões.
- **os**: biblioteca utilizada para manipular arquivos.
- **ImageDataGenerator**: utilizado para configurar transformações aleatórias e operações de normalização, feitas nas imagens durante o período de treinamento.
- **Sequential**: cria um modelo que permite a configuração camada por camada.
- **RMSprop**: método de taxa de aprendizado adaptável.
- **Conv2D**: utilizado para a transformação da imagem em uma matriz bidimensional.
- **MaxPooling2D**: utilizado para fazer a redução do tamanho dos dados.
- **Activation**: função de ativação da rede neural.
- **Dropout**: método de regularização para modelos de redes neurais.
- **Flatten**: é usada para tornar uma entrada multidimensional em unidimensional.
- **Dense**: camada totalmente conectada.
- **CSVLogger**: utilizado para obter uma visão sobre estados internos e estatísticas do modelo durante o treinamento.

- **OpenCV**: utilizada para realizar o tratamento nas imagens.

4.3 Desenvolvimento

Os testes de implementação foram iniciados com a avaliação de alguns *frameworks* e linguagens de programação. Entre as linguagens de programação avaliadas a escolhida foi o Python, pois possui uma maior gama de aplicações na área de Aprendizado de Máquina. O *framework* escolhido foi o Keras, pois demonstrou uma maior facilidade na implementação e uma documentação vasta, incluindo exemplos de uso.

Uma aplicação simplificada serviu como ponto de partida e também para testes de funcionamento da biblioteca.

4.3.1 Base de Dados

Um *dataset* com imagens de grandes felinos foi montado com o auxílio de uma biblioteca para a coleta das imagens. A classificação das mesmas foi feita manualmente para evitar erros na montagem das classes de treinamento.

Para o *download* das imagens, foi utilizado a biblioteca Google Images Download, onde foi usado um conjunto de parâmetros de busca com palavras-chave (nome do animal em várias línguas e nome científico da espécie). Após a conclusão do download foi feita manualmente a limpeza do *dataset*, removendo imagens erradas, com baixa resolução ou que possuíssem algum tipo de fator que pudesse comprometer os resultados do treinamento. Também foi utilizado um *script* para a remoção de imagens repetidas ou semelhantes.

Após a etapa de limpeza do *dataset* as imagens foram divididas em dois conjuntos: um conjunto de treinamento com 80% das imagens e um conjunto de validação com 20% das imagens. Também foi criado um conjunto menor com algumas imagens para testes após o processo de treinamento. O *dataset* completo ficou com 13.845 imagens, separados em quatro classes.

As quatro espécies de felinos escolhidas possuem pelagens distintas, que podem ser usadas como critério de classificação das imagens. As espécies escolhidas são: guepardo, leopardo, onça-pintada e jaguatirica. O Guepardo

(*Acinonyx jubatus*), ou *cheetah* em inglês, tem pelagem amarelada, com pequenos pontos espaçados de coloração preta. A Figura 6 demostra um guepardo e sua pelagem característica.

Figura 6 – Imagem de um guepardo.

Fonte: Foto por Sophia Hilmar (Pixabay, domínio público).

O Leopardo (*Panthera pardus*) possui pelagem de cor amarelada e que é coberta por manchas em forma de rosetas, que são em sua maioria fechadas e estreitamente espaçadas, de coloração preta. A Figura 7 demostra um leopardo e sua pelagem característica.

Figura 7 – Imagem de um leopardo

Fonte: Foto por hbieser (Pixabay, domínio público).

Onça-pintada (*Panthera onca*), ou *jaguar* em inglês, tem pelagem que pode ser da cor amarelo acastanhado (às vezes mais pálido), mas pode chegar ao avermelhado ou marrom. Possui rosetas grandes com pequenas manchas dentro, na coloração preta. A Figura 8 demostra uma onça-pintada e sua pelagem característica.

Figura 8 - Imagem de um onça-pintada

Fonte: Foto por Carlo Quinteros (Pixabay, domínio público).

Jaguatirica (*Leopardus pardalis*) ou *ocelot* em inglês: sua pelagem é curta e brilhante, com o fundo variando do amarelo claro ao avermelhado e cinza, com

manchas sólidas ou rosetas que podem se unir formando listras horizontais no corpo com coloração preta. A Figura 9 demostra uma jaguatirica e sua pelagem característica.

Figura 9 - Imagem de uma jaguatirica

Fonte: Foto por LucasFZ70 (Pixabay, domínio público).

4.3.2 Tratamento nas Imagens

Nesta parte descreve-se os tratamentos feitos nas imagens visando isolar a pelagem dos animais. Para isso foram implementados algoritmos usando as bibliotecas OpenCV e Scikit-image.

Para conseguir isolar a pelagem dos animais foram estudadas e implementadas algumas soluções. A primeira solução consistia em um algoritmo para recortar o animal da imagem e depois remover o fundo, mas que não teve um resultado adequado devido as imagens terem tamanhos e aspectos diferentes. Na maioria dos casos, não era possível recortar o animal de forma correta.

A segundo solução foi a implementação de um algoritmo que centraliza a imagem e aplicava um fator de *zoom* de 3.5 vezes. Esta solução teve resultados mais interessantes pois foi possível manter boa parte das imagens já que na maioria delas o animal se encontrava no centro.

A Figura 10 demostra a imagem original e a imagem após o tratamento.

Figura 10 - Tratamento na imagem

Fonte: Foto por Robert Feil (Pixabay, domínio público).

Após o tratamento e limpeza do novo *dataset* a distribuição das imagens ficou com 5226 imagens para o treinamento e 1120 imagens para validação.

4.3.3 Construção da Rede Neural

Para a construção da rede neural foram avaliados vários modelos de arquitetura presentes no *framework* Keras. Atualmente o Keras conta com 18 modelos pré-treinados utilizando imagens do *dataset* ImageNet. Após a avaliação dos modelos foram escolhidos três para utilização no desenvolvimento do trabalho:

- **MobileNet**: é uma das arquiteturas mais adequadas para aplicativos de visão baseados em dispositivos móveis. Usa convoluções separáveis em profundidade que reduzem significativamente o número de parâmetros quando comparados à uma rede com convoluções normais com a mesma profundidade. Isso resulta em redes neurais profundas e leves.

- **MobileNetV2**: utiliza os mesmos conceitos do primeiro modelo possuindo menos parâmetros, melhor velocidade de treinamento e um aumento na precisão.

- **DenseNet121**: buscam resolver os problemas de redes neurais muito profundas, simplificando o padrão de conectividade entre as camadas e explorando o potencial da rede através da reutilização de recursos.

Os dois primeiros modelos possuem respectivamente 4.253.864 e 3.538.984 parâmetros, sendo destinadas ao uso *mobile* e assim possuindo menos parâmetros de entrada quando comparadas com o terceiro modelo, que possui 8.062.504 parâmetros. Todos os modelos e as demais configurações foram avaliados na etapa de testes.

4.3.4 Modelo de Treinamento

O modelo de treinamento utilizado é o *Sequential*, um modelo que permite a configuração da rede neural camada por camada.

Para o modelo principal da rede neural foram testados três modelos predefinidos de redes neurais, disponíveis na biblioteca Keras. Os modelos utilizados são o MobileNet, MobileNetV2 e DenseNet121. Cada um deles possui uma arquitetura diferente, mas todos utilizam o modelo convolucional em sua estrutura.

A primeira camada deve receber o parâmetro **input_shape**, que possui o formato de entrada dos dados esperado. As demais camadas não necessitam receber esse parâmetro.

As redes neurais utilizadas foram originalmente projetadas para classificar imagens do *dataset* ImageNet. Na implementação das mesmas, disponível no Keras, estas recebem dois parâmetros a mais: **weights**, onde é especificado se a rede neural vai utilizar os pesos do ImageNet, e **include_top**, que é responsável por incluir as camadas totalmente conectadas na parte superior da rede.

A camada **Flatten** é responsável pelo achatamento dos dados, ou seja, transforma a matriz bidimensional em um vetor que é inserido na camada totalmente conectada da rede.

As camadas do tipo **Dense**, ou camadas totalmente conectadas, são as camadas mais comuns em redes neurais. Estas consomem as saídas de outras camadas, usando estas como entradas de seus neurônios. Os parâmetros utilizados foram o **units**, que recebe um número inteiro com a quantidade de neurônios da camada e o parâmetro **activation**, que indica qual função de ativação deve ser utilizada.

Após a definição das camadas da rede é preciso utilizar o método **compile** para compilar o modelo de treinamento. O método **compile** pode variar dependendo do tipo de classificação desejada. Seus parâmetros são:

- **loss**: é uma função de perda, calculando o valor objetivo que o modelo tentará minimizar. Pode ser o identificador em formato *string* de uma função de perda existente.
- **optimizer**: recebe uma função de otimização que pode impactar na taxa de aprendizado. Tipicamente é a função que implementa a estratégia de aprendizado a ser aplicada sobre o modelo de rede neural.
- **metrics**: define as métricas que serão avaliadas durante o processo de treinamento da rede neural.

O método **summary** é responsável por retornar as informações da construção da rede neural, como número de neurônios, tipo das camadas e dimensões dos dados das camadas.

A Figura 11 mostra um exemplo de estrutura do modelo de treinamento:

Figura 11 - Modelo de Treinamento

```
model = Sequential()
main_model = MobileNet(weights='imagenet', include_top=False, input_shape=input_shape)
model.add(main_model)
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))


model.compile(loss='binary_crossentropy',
            optimizer=SGD(lr=0.0001),
            metrics=['accuracy'])

with open(MODEL_SUMMARY_FILE,"w") as fh:
    model.summary(print_fn=lambda line: fh.write(line + "\n"))
```

Fonte: Do Autor (2019)

4.3.5 Configuração e Definição dos Dados

Nessa etapa é utilizado a classe **ImageDataGenerator** para fazer as transformações e ajustes nas imagens, preparando as mesmas para a etapa de treinamento. Dois conjuntos de imagens estão sendo utilizados nos testes, o conjunto de treinamento (**training_generator**) e o conjunto de validação (**validation_generator**), ambos contendo o mesmo número de classes.

Para a configuração das imagens, foram utilizados os seguintes parâmetros:

- **rescale**: fator de redimensionamento da imagem.
- **shear_range**: ângulo de cisalhamento no sentido anti-horário, medido em radianos.
- **zoom_range**: intervalo para *zoom* aleatório.
- **horizontal_flip**: inverte as entradas horizontalmente de forma aleatória.
- **rotation_range**: faixa de graus para rotações aleatórias.
- **width_shift_range**: deslocamento aleatório horizontal das imagens.
- **height_shift_range**: deslocamento aleatório vertical das imagens.
- **fill_mode**: preenche os pontos fora dos limites da entrada.
- **validation_split**: fração de imagens reservadas para validação.

Após a configuração das imagens é definido o local do diretório, as dimensões da imagem, o tamanho do lote de testes por época e o modelo de classe que determina o tipo de matrizes de rótulos que serão retornados. A Figura 12 exibe como a configuração foi feita.

Figura 12 - Configuração das imagens

```
training_data_generator = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rescale=1/255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2)

validation_data_generator = ImageDataGenerator(
    rescale=1./255)


training_generator = training_data_generator.flow_from_directory(
    training_data_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode="binary")
validation_generator = validation_data_generator.flow_from_directory(
    validation_data_dir,
    target_size=(IMAGE_WIDTH, IMAGE_HEIGHT),
    batch_size=BATCH_SIZE,
    class_mode="binary")
```

Fonte: Do Autor (2019)

4.3.6 Configuração do Treinamento

A função **fit_generator** é responsável por executar o treinamento da rede neural. Nela são definidos parâmetros como o conjunto de testes e de validação, o número de épocas de treinamento e a quantidade de entradas por época. Também pode ser utilizada uma função de *callback* que retorna um método ou função durante o final de cada época do treinamento. A Figura 13 exibe as configurações feitas nesta etapa.

Figura 13 - Configuração do treinamento

```
model.fit_generator(
    training_generator,
    steps_per_epoch=len(training_generator.filenames)//BATCH_SIZE,
    epochs=EPOCHS,
    validation_data=validation_generator,
    validation_steps=len(validation_generator.filenames)//BATCH_SIZE,
    callbacks=[PlotLossesKeras(), CSVLogger(TRAINING_LOGS_FILE,
                                            append=False,
                                            separator=";")],
    verbose=1)
model.save(MODEL_FILE)
```

Fonte: Do Autor (2019).

4.3.7 Estimativa de escala das pelagens dos animais

Foi implementado um algoritmo para fazer a estimativa da escala das pelagens dos animais. Este algoritmo recebe uma imagem e faz algumas etapas de processamento de imagens sobre a mesma.

Primeiramente, ao carregar uma imagem são utilizados os módulos **color** e **exposure** presentes na biblioteca Scikit-image, para remover a cor da imagem deixando a mesma em preto e branco, além de aprimorar o contraste da imagem.

Em seguida é aplicado o filtro **threshold_otsu** que utiliza o método de Otsu. A partir de uma imagem com tons de cinza, este método determina qual o melhor valor de *threshold* para separar os elementos do fundo e da parte frontal da imagem em dois grupos de pixels, atribuindo a cor branca ou preta para cada um deles. Tal tratamento faz com que os pontos de pelagem fiquem em destaque, facilitando o cálculo da sua escala.

No próximo passo foram utilizados os módulos **util** e **morphology** para inverter as cores da imagem, de modo que os pontos de pelagem fiquem na cor branca, além de remover pequenos ruídos das imagens. Em seguida são mapeados todos os pontos de pelagem utilizando o algoritmo **morphology.watershed**, onde os pixels são distribuídos em bacias marcadas e armazenados em filas, de acordo com suas propriedades.

Por fim é utilizado o algoritmo **measure.regionprops**, que mede as propriedades das regiões da imagem rotuladas e retorna informações como tamanho, área e intensidade. Então os dados de tamanho são agrupados utilizando um histograma por área, onde é possível verificar a variação de sua distribuição.

Logo após são mapeados todos os subconjuntos gerados que possuem uma quantidade de valores acima de 5% da quantidade do conjunto pai. Estes subconjuntos são utilizados para o cálculo da média. Tal método foi utilizado para eliminar valores que estavam fora de uma determinada faixa visando uma melhoria no resultado da estimativa de tamanho.

As imagens com sua escala calculada poderão ser utilizadas no processo de treinamento, onde será possível dizer a qual classe pertence e qual o tamanho da pelagem do animal.

5 TESTES E ANÁLISE DOS RESULTADOS

Neste capítulo são descritos os testes feitos sobre vários cenários, onde se variam a arquitetura da rede neural, o *dataset* de treinamento e também algumas configurações. Visando a melhoria e precisão dos resultados, cada teste foi executado cinco vezes, onde os resultados exibidos correspondem à média aritmética dos valores alcançados.

Para realizar os testes foram utilizadas duas máquinas distintas. A primeira foi o ambiente do Google Colaboratory com as seguintes configurações: processador dual core, 12,6 GB de memória RAM e uma GPU Nvidia Tesla K80 com 12 GB de memória RAM.

O segundo ambiente é uma máquina desktop com as seguintes configurações: processador Core i5 4590, 16 GB de memória RAM e uma GPU Nvidia GTX 1650 com 4GB de memória RAM. O segundo ambiente foi utilizado para testes mais simples e o primeiro para testes mais pesados.

Para os testes de definição dos parâmetros foram utilizadas duas configurações de dataset, uma utilizando duas classes (guepardo e leopardo) com 3.200 imagens para o treinamento e 800 imagens para a validação e a segunda, utiliza as quatro classes com 6400 imagens para o treinamento e 1600 imagens para a validação. Nas duas configurações as imagens foram divididas igualmente entre as classes de treinamento.

5.1 Definição da Arquitetura

Para a montagem da rede neural, foram escolhidas 3 arquiteturas, presentes no Keras. Os modelos escolhidos foram o MobileNet, o MobileNetV2 e o DenseNet121, devido a estes modelos possuírem uma arquitetura mais simples e assim facilitando o processo de treinamento.

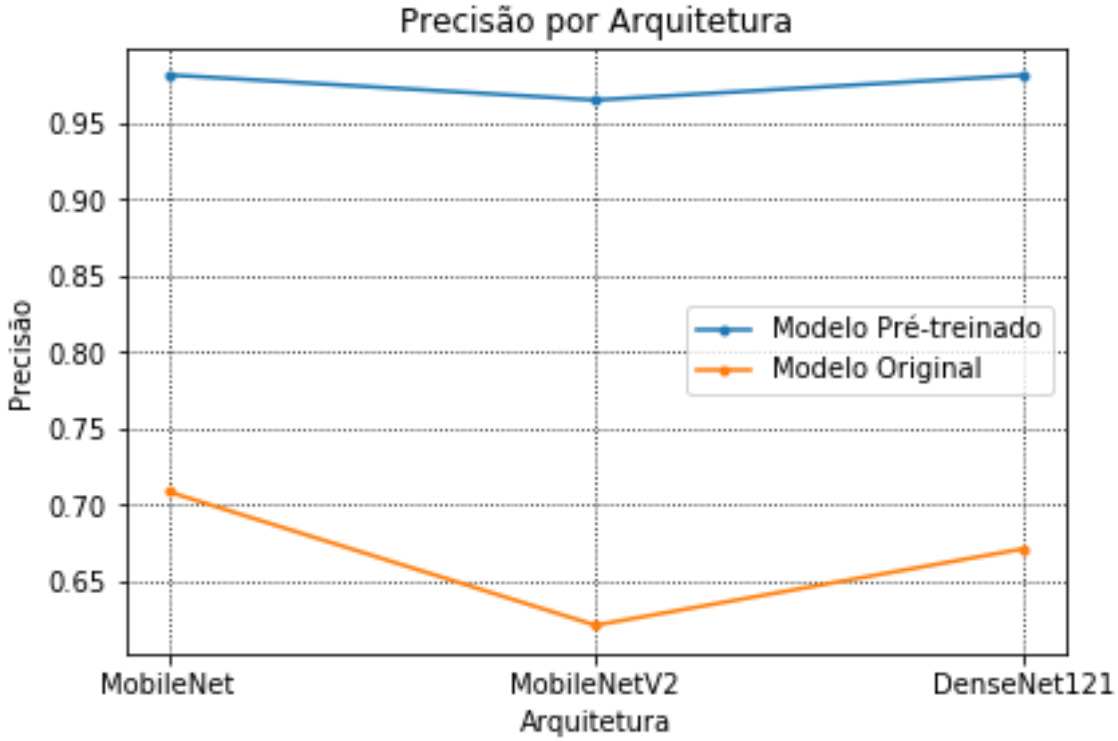
Para a definição da melhor arquitetura foi feita uma bateria de testes visando o melhor resultado possível, onde os melhores modelos foram comparados. Nesta etapa foram usadas várias configurações e ajustes finos para definir a melhor configuração para cada arquitetura, sendo que o número de épocas de treinamento foi de 10 e o *batchsize* foi de 5.

Outro parâmetro que foi utilizado nos testes de arquitetura foi o **weights,** que como explicado no último capítulo, define se a arquitetura irá começar o treinamento com os pesos pré-treinados ou se vai utilizar os pesos aleatórios. Para os testes serão utilizando dois modelos um sem este parâmetro (Modelo Original) e outro com o parâmetro (Modelo pré-treinado).

Para avaliação são considerados os seguintes atributos: maior precisão, a menor perda e o menor tempo de treinamento. Para cada configuração escolhida serão repetidos cinco vezes os testes, visando verificar se não há uma mudança muito grande de resultados.

A Figura 14 demostra os melhores resultados obtidos durante os testes.

Figura 14 - Teste de arquiteturas (Precisão)

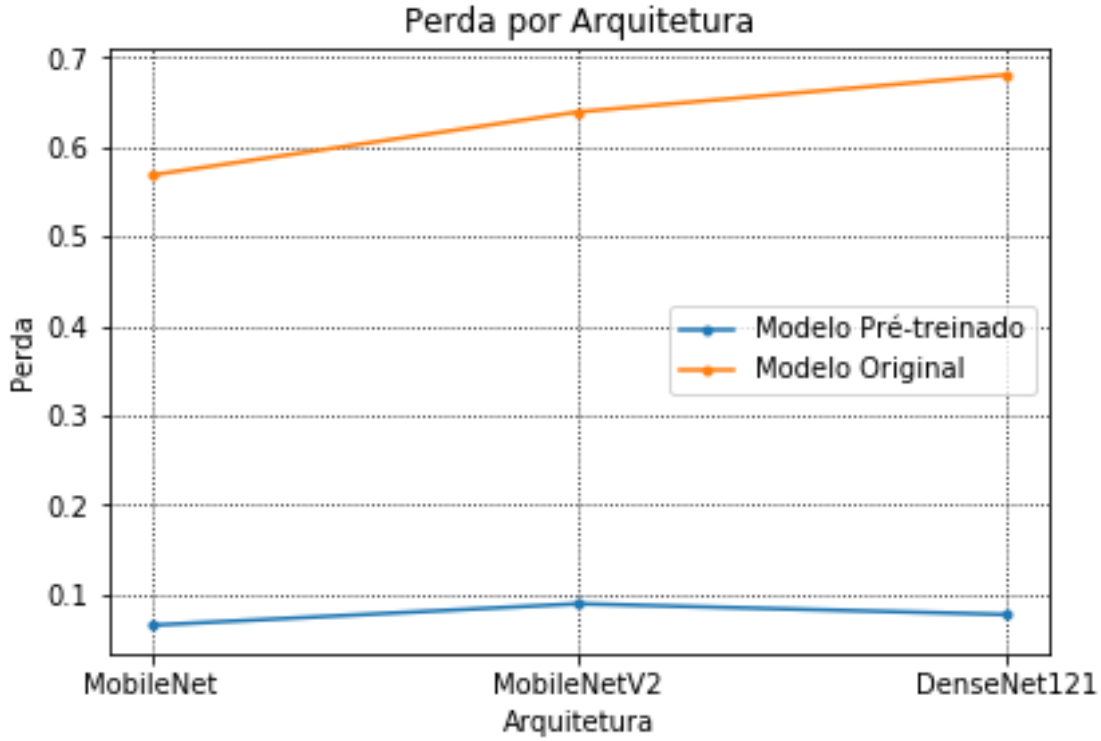


Fonte: Do Autor (2019)

Utilizando os pesos do *ImageNet*, todas as arquiteturas tiveram resultados acima dos 90%, os modelos MobileNet e DenseNet121 acabaram por ter resultados quase iguais com uma pequena margem de diferença já a MobileNetV2 teve um resultado um pouco abaixo das demais, mas se mantendo acima dos 95%. O modelo original teve resultados acima dos 70% para o MobileNet e resultados inferiores para as demais arquiteturas.

A Figura 15 demonstra o gráfico de perda para cada arquitetura.

Figura 15 - Teste de arquiteturas (Perda)



Fonte: Do Autor (2019)

Nó gráfico de perda quanto mais baixo o percentual melhor são os resultados, arquitetura MobileNet possui um valor de perda menor se comparado com as outras arquiteturas nos dois modelos de treinamento.

Observando os dois gráficos fica evidente a diferença na utilização dos dois modelos com variações de resultados muito grandes principalmente no gráfico de perda, em que a diferença chega a ser de mais de 50% em todos os casos e devido a isso o modelo pré-treinado será escolhido para o restante dos treinamentos.

Os tempos médios de treinamento para cada arquitetura é de 18,38 minutos para a MobileNet, de 22,23 minutos para a MobileNetV2 e de 41,38 minutos para a DenseNet121.

Após os testes foi possível perceber que a arquitetura MobileNet vem a ser uma boa escolha para a utilização nos demais testes do trabalho.

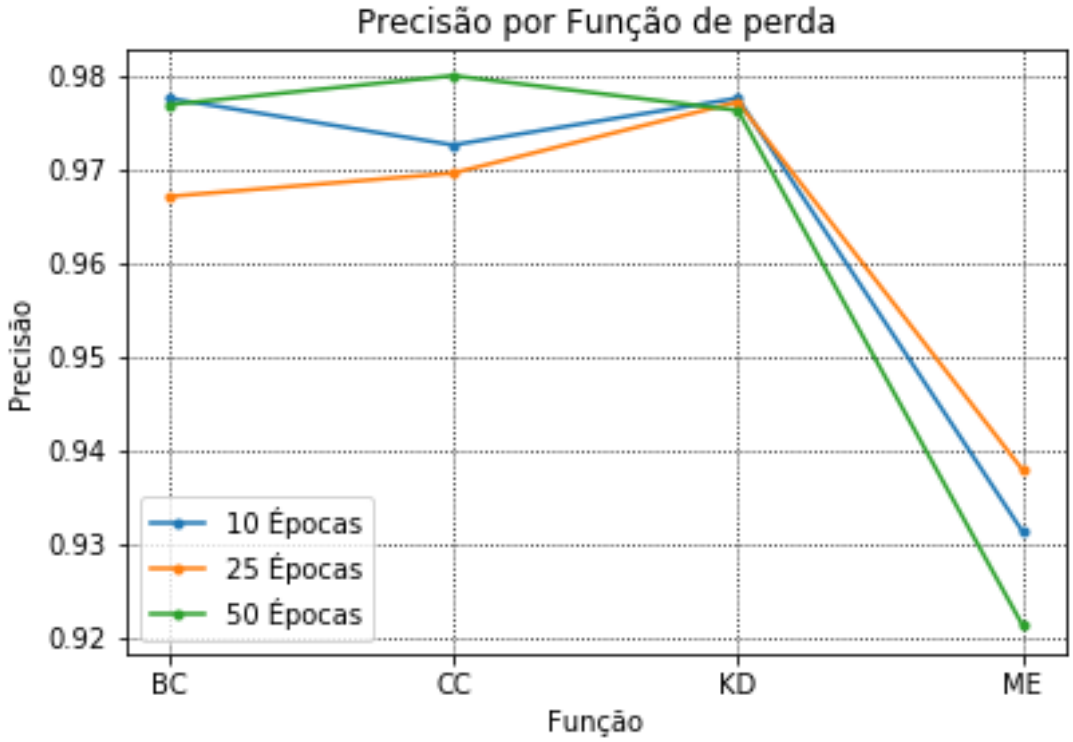
5.2 Definição da Função de Perda

A *loss function* tem por objetivo avaliar quão bem um algoritmo consegue modelar os dados de entrada. Caso o valor de perda for muito grande ele pode causar problemas no processo de previsão da classificação.

Atualmente existem várias funções de perda presentes na documentação do Keras. Para utilização neste trabalho foram avaliados todos os modelos e destes foram escolhidos quatro que obtiveram bons resultados. As funções escolhidas foram: *binary-crossentropy* (BC), *categorical-crossentropy* (CC), *kullback-leibler-divergence* (KD) e *mean-squared-error* (ME).

Para os demais testes, foi utilizada a arquitetura MobileNet, já que a mesma demostrou bons resultados nos testes iniciais. Cada uma das funções será testada em três cenários com 10, 25 e 50 épocas, variando também a quantidade de classes no treinamento. Os resultados para os testes com duas classes e 10, 25 e 50 épocas podem ser vistos na Figura 16.

Figura 16 - Função de perda para duas classes.

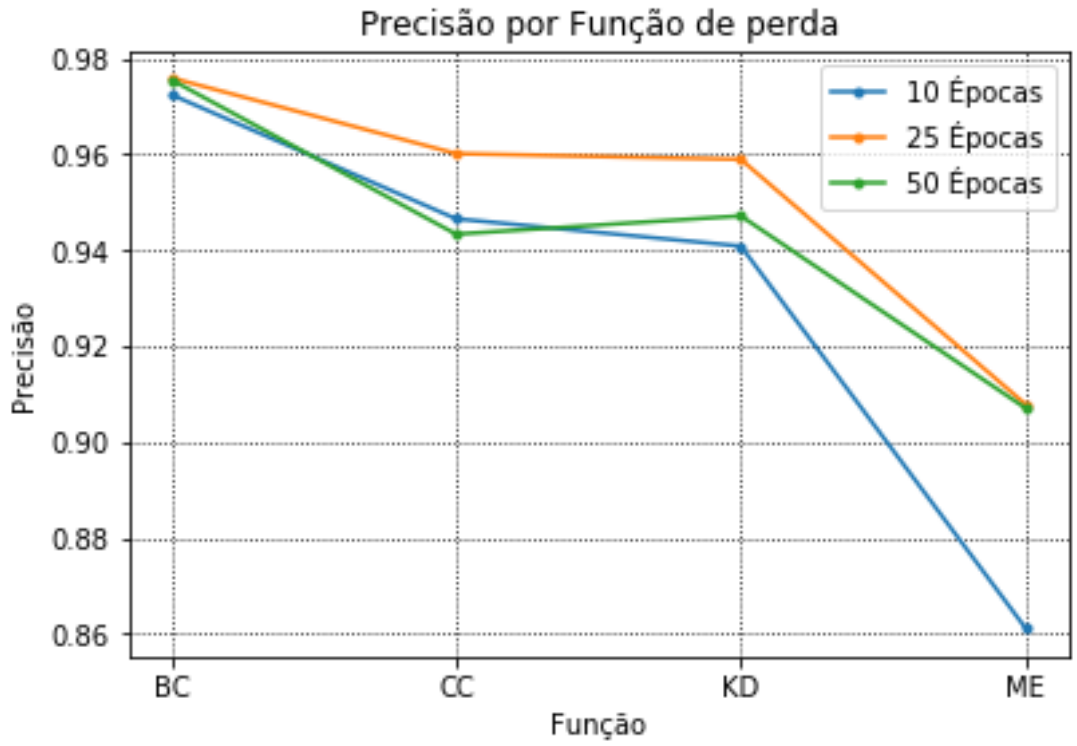


Fonte: Do Autor (2019)

Nos testes com duas classes foi possível identificar que as funções *binary-crossentropy* (BC), *categorical-crossentropy* (CC) e *kullback-leibler-divergence* (KD) tiveram resultados muito próximos a 98% de precisão. Já a função *mean-squared-error* (ME), em todos os casos, apareceu com resultados abaixo dos 95%.

Os resultados para os testes com quatro classes e 10, 25 e 50 épocas podem ser vistos na Figura 17.

Figura 17 – Função de perda para quatro classes.



Fonte: Do Autor (2019)

Durante os testes com quatro classes foi possível notar que ocorreu uma pequena diferença em relação aos testes com duas classes, uma vez que as funções *categorical-crossentropy* (CC) e *kullback-leibler-divergence* (KD) tiveram resultados muito próximos em todos os cenários, perdendo apenas para a função *binary-crossentropy* (BC), que manteve um resultado maior quando aumentado a complexidade dos testes e atingindo quase 98% de precisão. A função *mean-*

squared-error (ME) não obteve resultados tão altos, mas apresentou aumento de precisão, quando o número de épocas aumenta.

Devido a estes resultados a função *binary-crossentropy* demostrou ter um bom desempenho com complexidades mais altas de treinamento, sendo escolhida como base para os demais testes.

5.3 Definição do Otimizador

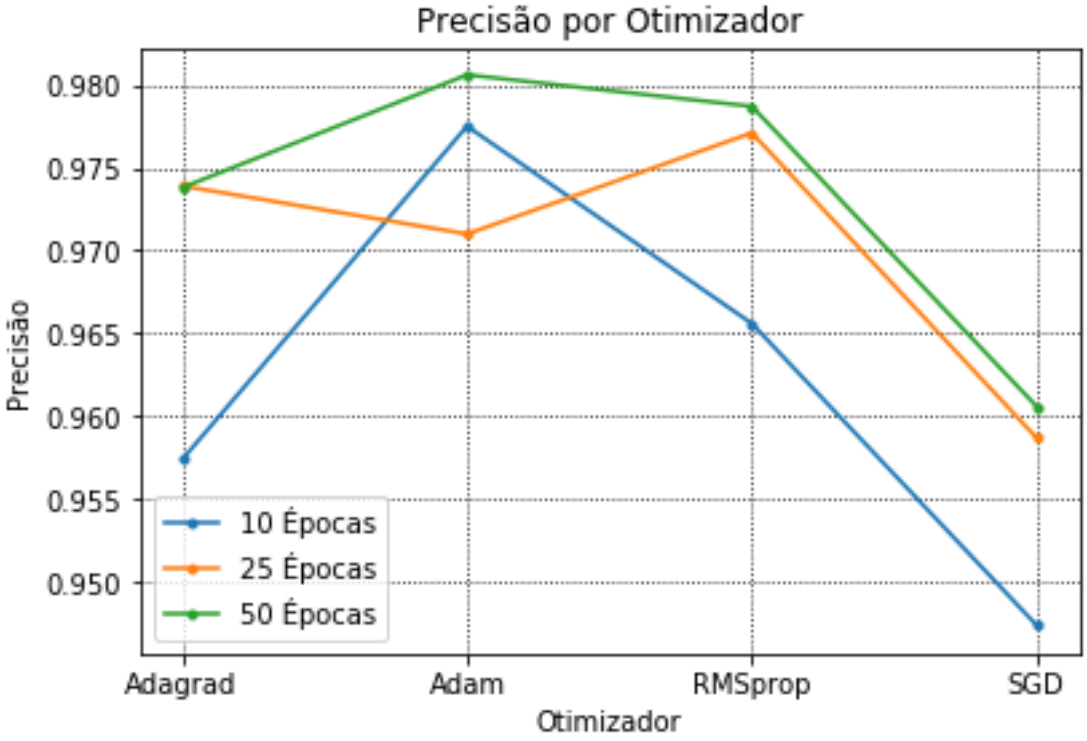
O *optimizer* tem como objetivo minimizar ou maximizar uma determinada função de perda (*loss function*), atualizando o modelo em resposta à saída da função. Isso sempre acontece da forma mais precisa possível, assim melhorando os resultados obtidos em um processo de treinamento.

Para a realização dos testes foram testados quatro *optimizers*. Estes tiveram os melhores resultados nos testes após a avaliação de todos os modelos presentes na documentação do Keras. Os escolhidos foram: Adam, Adagrad, SGD e RMSprop.

Os parâmetros utilizados serão os mesmos para todos os *optimizers*, sendo que as demais configurações se mantêm como nos últimos testes.

Os resultados para os testes com duas classes e 10, 25 e 50 épocas podem ser vistos na Figura 18.

Figura 18 – Otimizador para duas classes.

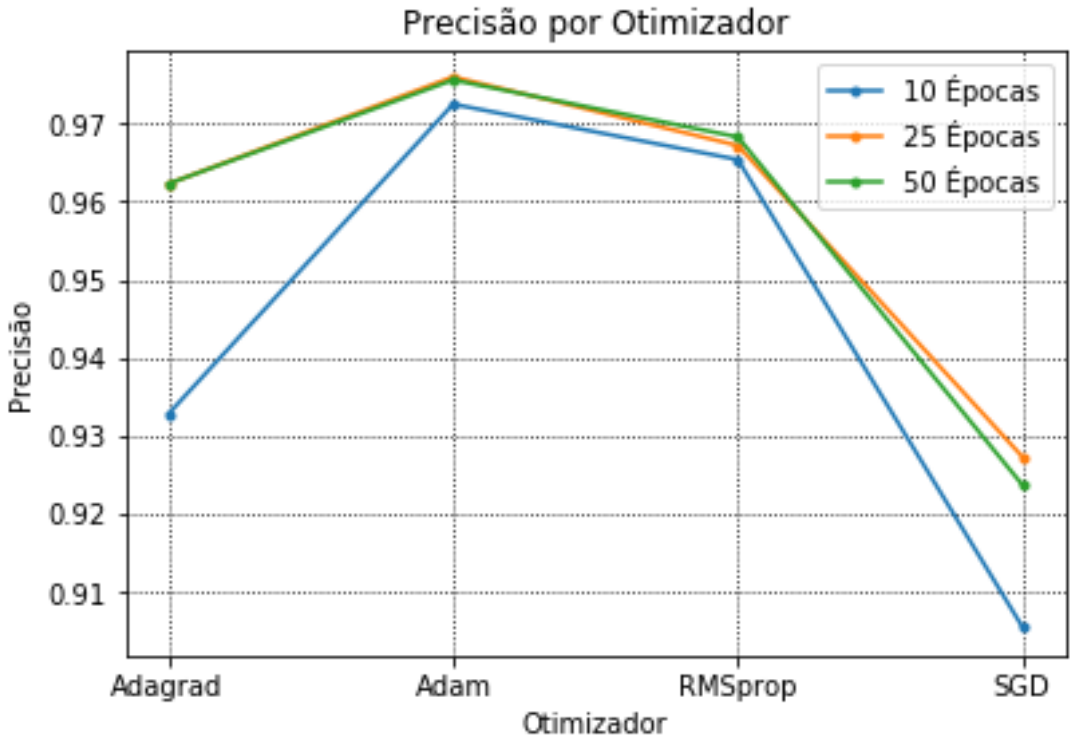


Fonte: Do Autor (2019)

Observando o gráfico é possível perceber que, com o aumento do número de épocas de treinamento, a precisão de todos os otimizadores aumenta. Os resultados para duas classes se mantiveram muito próximos, com exceção do otimizador SGD, que ficou um pouco abaixo dos demais.

Os resultados para os testes com quatro classes e 10, 25 e 50 épocas podem ser vistos na Figura 19.

Figura 19 – Otimizador para quatro classes



Fonte: Do Autor (2019)

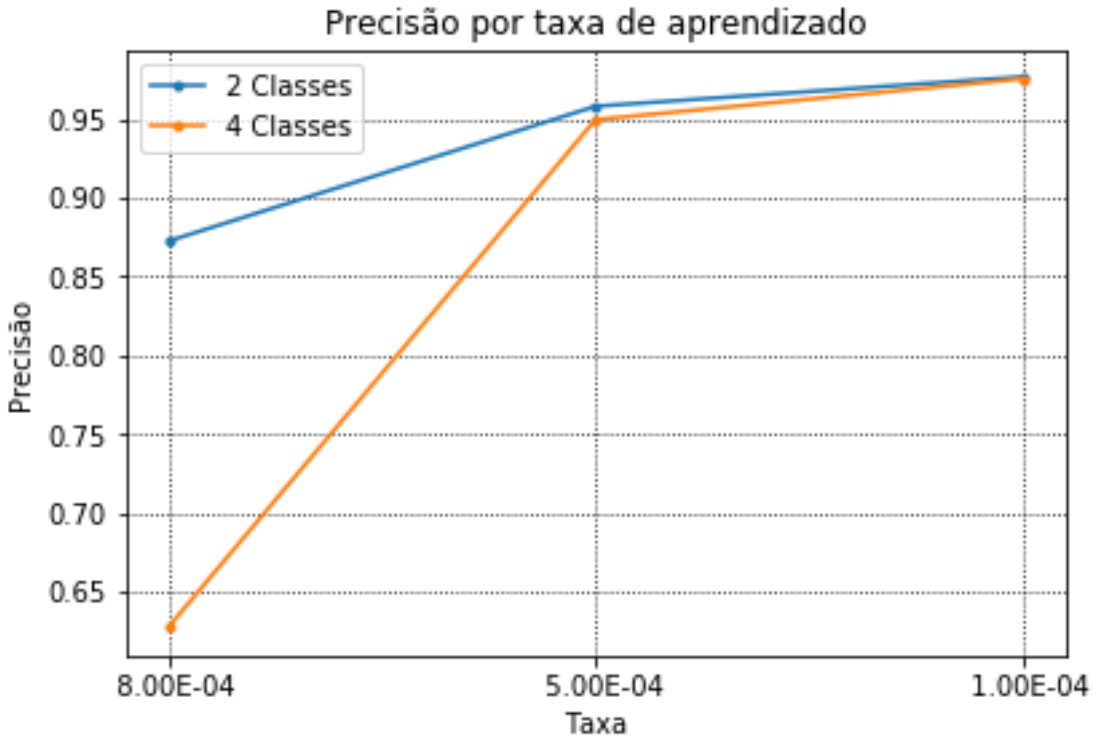
Comparando os testes com duas e quatro classes é possível notar que não houve mudanças bruscas de resultados e percentuais se mantiveram praticamente os mesmos. O otimizador que teve os melhores resultados foi o Adam, atingindo 97,56% de precisão, porém os demais também mantiveram bons números.

5.4 Configuração da Taxa de Aprendizado

Outra configuração que pode impactar no aprendizado da rede neural é a taxa de aprendizado (em inglês, *learning rate*). Este valor é responsável por controlar o quanto estamos ajustando os pesos da rede neural em relação ao gradiente de perda, ou seja, é o valor com que os pesos são atualizados durante o treinamento. Por padrão ele possui valores muito baixos, como por exemplo 0,001, mas pode ser configurado para valores maiores ou menores dependendo da aplicação.

Para a realização dos testes utilizando este parâmetro, foram escolhidos os valores de 0,0001, 0,0005 e 0,0008. Estes foram testados com variação de duas e quatro classes, e também com 50 épocas de treinamento. A Figura 20 exibe os resultados dos testes.

Figura 20 - Taxa de aprendizado para duas e quatro classes.



Fonte: Do Autor (2019)

No gráfico foi possível verificar que se a taxa de aprendizado possuir valores mais baixos a precisão da rede neural aumenta, independentemente do número de classes utilizadas no treinamento. Outro valor que foi testado foi 0,001, que como dito anteriormente é o valor *default* para todos os otimizadores, entretanto não foi possível obter bons resultados utilizando este valor, fazendo com que a precisão da rede neural não variasse durante os testes. Por isso este último valor foi descartado, não sendo utilizado nos testes.

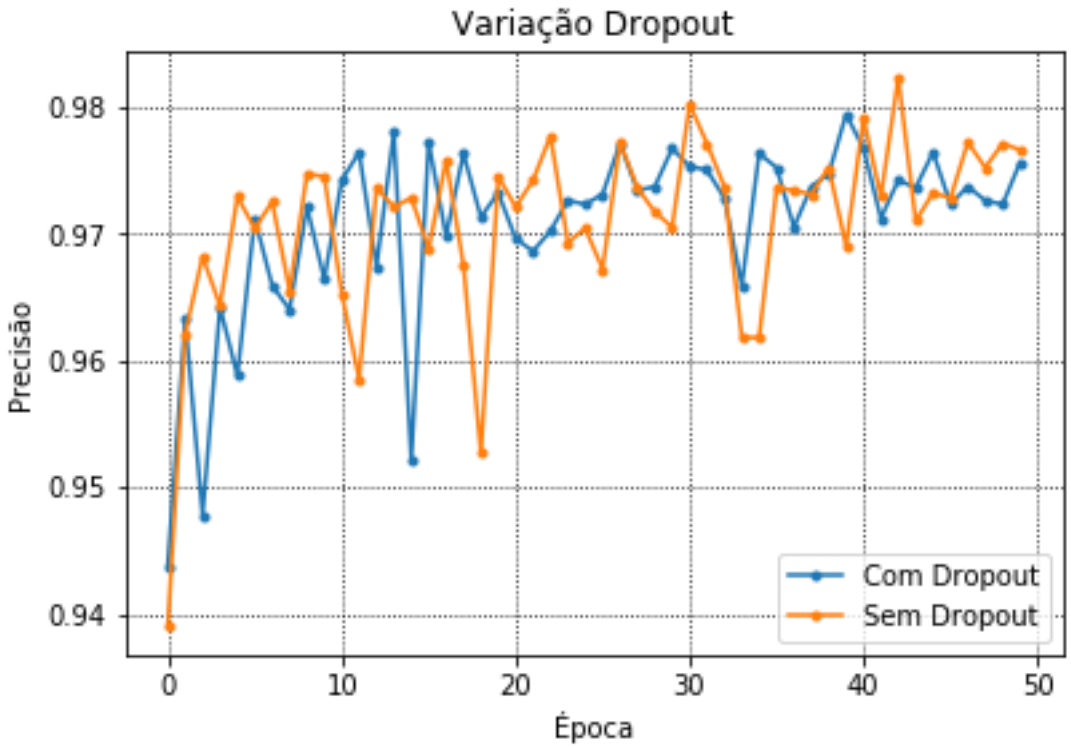
5.5 Camadas de *Dropout*

O *dropout* é uma técnica em que neurônios são selecionados aleatoriamente e ignorados durante o treinamento. Isso significa que sua contribuição para a ativação dos neurônios das próximas camadas é removida temporariamente, tanto no processo de cálculo do peso sináptico quanto no processo de atualização dos pesos (*backpropagation*).

Tal técnica pode vir a melhorar o desempenho da rede neural em determinados casos, mas deve ser usada com cuidado, pois uma taxa elevada de

dropout pode desestabilizar a rede, uma vez que os neurônios são desativados de forma aleatória. A Figura 21 possui os resultados dos testes com e sem *dropout*.

Figura 21 – Efeito da Variação do *dropout*.



Fonte: Do Autor (2019)

Analisando o gráfico é possível ver que não há uma grande diferença nos resultados, em nenhum dos casos, tendo resultados muito semelhantes. Por padrão, os testes desde o início utilizam dropout nas camadas densas adicionais da rede, usando uma taxa de 0,2 de desativação de neurônios. Devido aos resultados não demostrarem nenhuma grande variação sem seu uso, a configuração foi mantida para os demais testes.

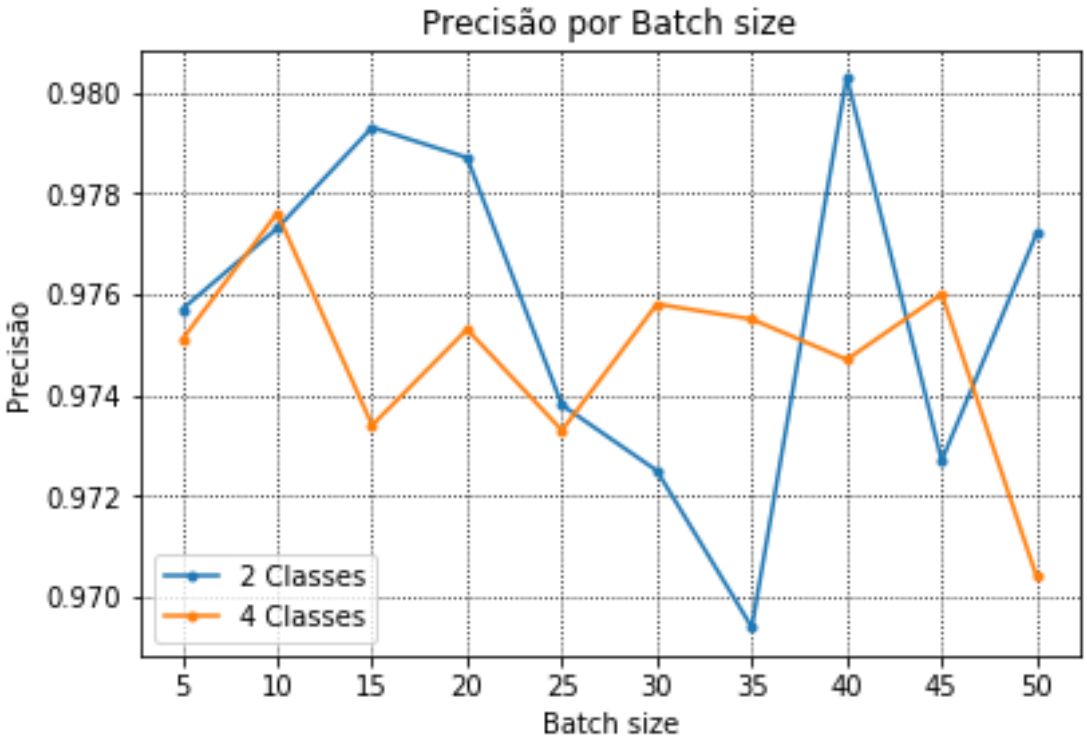
5.6 Definição do *Batch size*

O *batch size* é o parâmetro responsável pelo tamanho do lote de amostras que usado no treinamento da rede neural. Quanto maior o valor deste parâmetro, maior será o número de lotes de imagens utilizados por época. Deve haver um certo cuidado, pois muitos lotes de imagens podem causar consumo excessivo de memória RAM.

Por padrão, nos testes realizados nas seções anteriores, a configuração utilizada para este parâmetro foi a metade do número de épocas. Para os testes deste parâmetro foram avaliados os valores de 5 a 50, utilizando duas e quatro classes e 50 épocas.

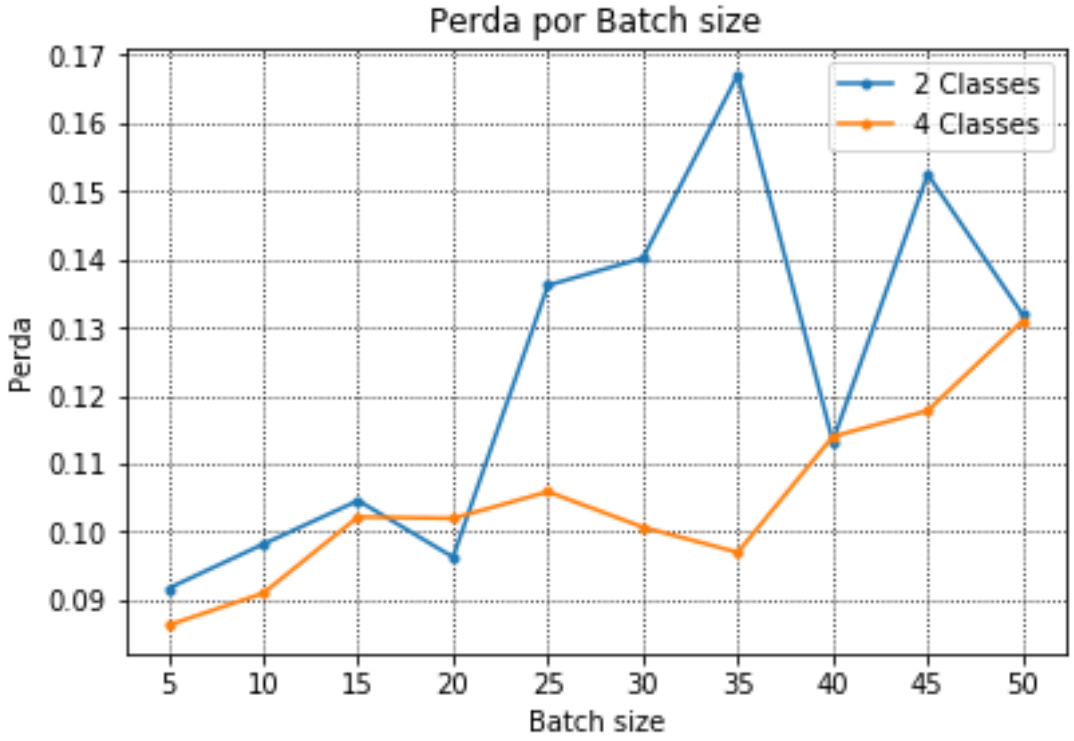
A Figura 22 mostra os resultados de precisão e a Figura 23 mostra a perda por batch size.

Figura 22 - Precisão por variação do *batch size*.



Fonte: Do Autor (2019)

Figura 23 - Perda por variação de *batch size.*



Fonte: Do Autor (2019)

Ao analisar os dois gráficos é possível ver algumas variações em comparação a duas e quatro classes, quando valores que ficaram ótimos em duas classes não tiveram o mesmo desempenho em quatro classes. Por isso, para a definição deste parâmetro será considerado o valor que possui uma boa média nos dois gráficos.

O valor que obteve resultados muito semelhantes nos dois cenários foi o de 10. Este não obteve os melhores resultados, mas alcançou médias próximas a 97,7% e perdas próximas a 9%. Tal valor se manteve mais estável durante estes testes, e por isso foi escolhido como parâmetro para os demais testes.

5.7 Testes com imagens da pelagem dos animais

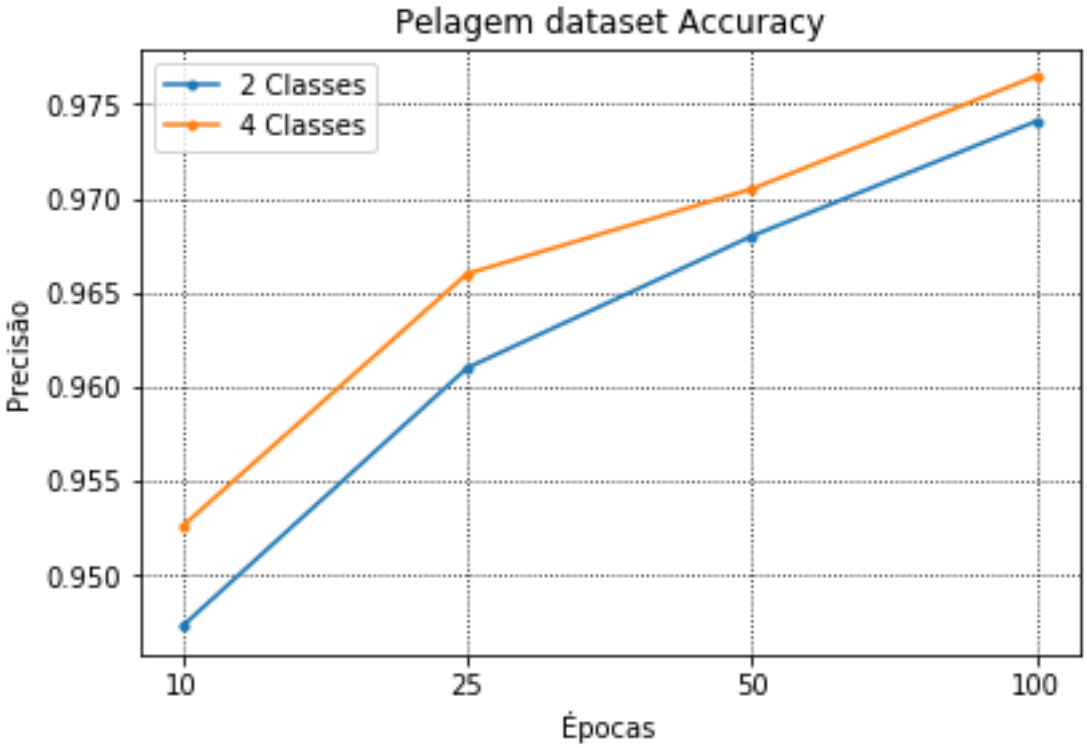
Por fim, no último conjunto de testes após a definição de todos os parâmetros, serão avaliadas as imagens da pelagem dos animais, para verificar se a rede neural é capaz de identificar estes padrões conforme o objetivo do trabalho.

Nesta parte será utilizado o *dataset* criado após o tratamento de isolar a pelagem dos animais. Já que as demais configurações da rede neural já foram

definidas, os testes utilizando o novo *dataset* utilizaram 10, 25, 50 e 100 épocas e tiveram variação de duas e quatro classes de treinamento.

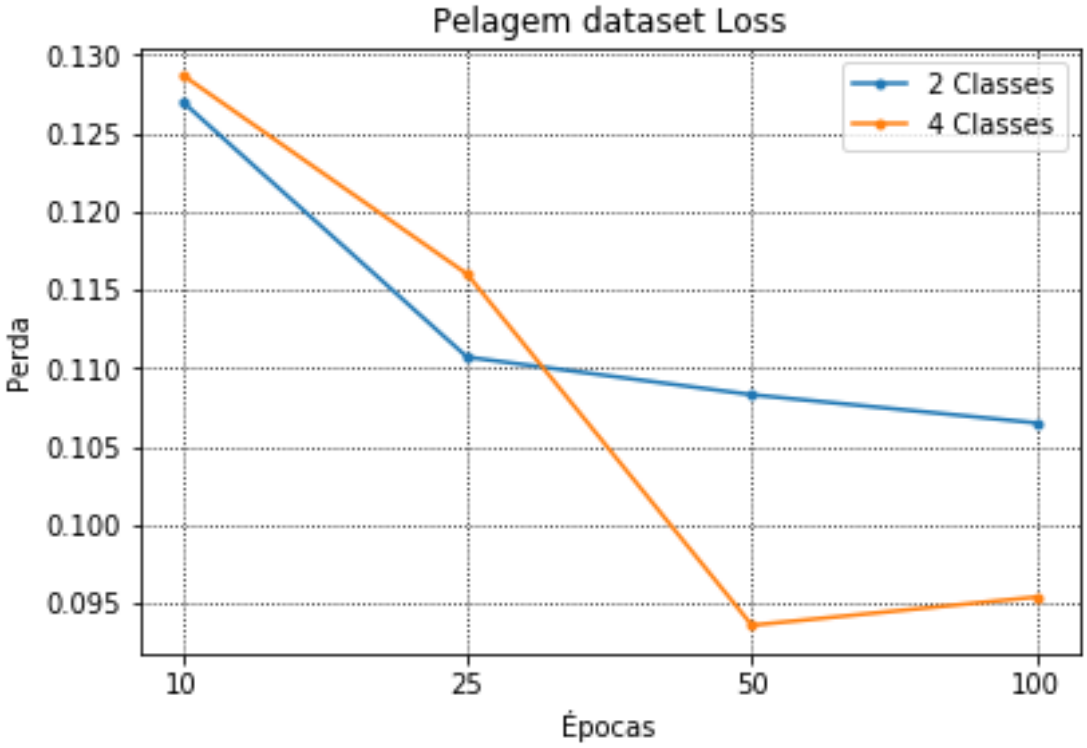
Os resultados dos testes com este *dataset* de pelagem dos animais pode ser visto nas Figuras 24 e 25.

Figura 24 - Dataset de pelagens accuracy



Fonte: Do Autor (2019)

Figura 25 - Dataset de pelagens Loss



Fonte: Do Autor (2019)

Os gráficos demostram que houve ganhos de precisão ao aumentar o número de épocas, chegando a 97,65% para 100 épocas de treinamento. Mesmo com números de épocas mais baixos já era possível observar resultados perto ou acima dos 95%.

Ao analisar o gráfico de perda é possível observar que não houve uma variação muito grande entre 50 e 100 épocas, o que demostra que a rede neural ficou estável entre estes períodos de teste. Os demais resultados e testes de previsão serão discutidos no próximo capítulo.

6 RESULTADOS

Neste capítulo serão discutidos os resultados do trabalho e demais atividades executadas durante o desenvolvimento.

6.1 Classificação das imagens

Para avaliar o desempenho da rede neural no processo de classificação serão feitos três cenários de testes. O primeiro irá utilizar as imagens originais das quatro espécies, o segundo irá utilizar as imagens das pelagens dos animais e o último cenário irá utilizar algumas imagens artificiais da pelagem dos animais.

O primeiro cenário de teste utilizará um modelo treinado com base nas imagens originais das quatro espécies. Já o segundo e terceiro modelo vão utilizar o modelo treinado com base nas imagens das pelagens dos animais (obtidas com a ampliação do centro das imagens).

A funções utilizadas para os testes de classificação foram a **predict_classes** e a **predict_proba**. A primeira indica a qual classe a imagem de entrada pertence, já a segunda retorna uma lista com as probabilidades de cada classe para a imagem de entrada.

As imagens dos animais e da pelagem utilizadas como amostras são provenientes do conjunto de testes do *dataset* já que o mesmo não foi utilizado em sua totalidade algumas imagens foram removidas para serem utilizadas nesta etapa.

Para as imagens sintéticas, foram utilizados três tipos de amostras: a primeira com modelos 3D sintetizados no editor Blender, na segunda com imagens 2D sintetizadas utilizando um algoritmo em Python e a terceira com as mesmas imagens sintetizadas do segundo modelo só que neste, foi feito um tratamento para gerar modificações aleatórias nas imagens. O objetivo disso é avaliar como a rede neural se comporta com imagens deste tipo. A Figura 26 mostra alguns exemplos de pelagens sintéticas.

Figura 26 - Exemplos de pelagens sintéticas

Fonte: Imagens disponibilizadas por Marcelo de Gomensoro Malheiros.

A Tabela 1 contêm os resultados dos testes de classificação.

Tabela 1 - Resultados

Cenário	Amostras	Acertos	Erros	Percentual
Imagens dos Animais	109	108	1	99,08%
Imagens da Pelagem	109	108	1	99,08%
Imagens Sintéticas 3D (CG)	4	1	3	25%
Imagens Sintéticas 2D (CG)	198	0	198	0%
Imagens Sintéticas 2D com variação (CG)	198	21	177	10,60%

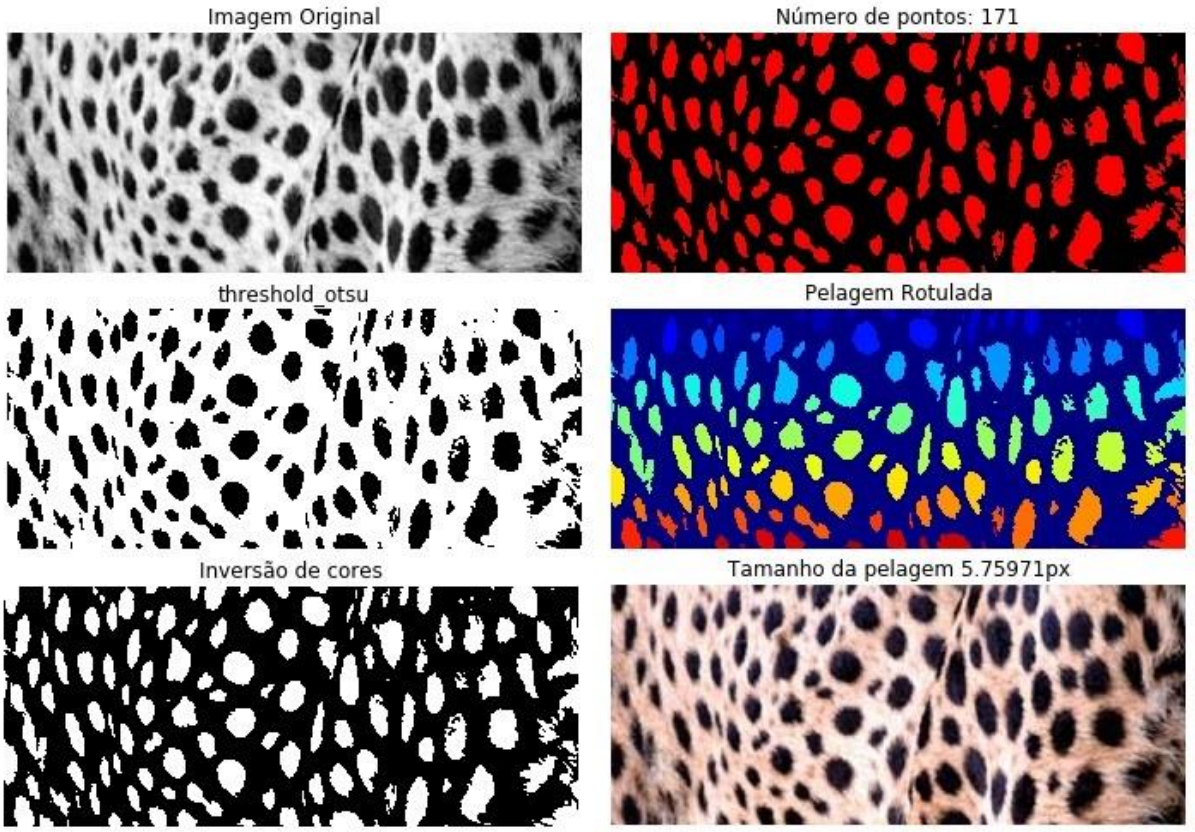
Fonte: Do Autor (2019).

Para as imagens dos animais e da pelagem foram obtidos ótimos resultados com um pouco mais de 99% de acerto, já as os testes com pelagens sintéticas não obtiveram bons números.

6.2 Estimativa da escala da pelagem.

No processo de cálculo de escala da pelagem foram utilizadas somente as imagens da pelagem dos animais pois as imagens contidas no *dataset* original possuíam muitas diferenças na posição dos animais e em alguns casos, não era possível visualizar as manchas na imagem. A Figura 27 demostra os tratamentos feitos na imagem até a definição do tamanho da pelagem.

Figura 27 - Etapas da definição do tamanho das pelagens



Fonte: Do Autor (2019).

Durante o processo de cálculo foi possível notar que a estimativa do tamanho da pelagem varia muito da proximidade da pelagem e do tamanho da imagem. Por motivos tempo, não foi possível separar as imagens em subclasses de acordo com o tamanho da pelagem e utilizar o tamanho da pelagem no processo de classificação estes procedimentos serão sugeridos como proposta para um possível futuro trabalho.

6.3 Considerações finais

A proposta central do trabalho foi verificar se era possível utilizar redes neurais para fazer a classificação de imagens de grandes felinos utilizando sua pelagem característica.

No trabalho foram abordados como temas Redes Neurais, Classificação de imagens, Deep learning e Processamento de imagens. Todos os temas tiveram sua base teórica e prática explicadas e utilizadas no processo de desenvolvimento do trabalho.

Avaliando os resultados obtidos nos últimos testes foi possível comprovar que uma rede neural convolucional é capaz de fazer a classificação das imagens de grandes felinos, utilizadas imagens reais do animal propriamente dito e da sua pelagem característica, mas para imagens sintéticas não foi possível obter resultados bons. Uma possível solução para melhorar os resultados com este tipo de imagem seria adicionar imagens sintéticas no *dataset* de pelagens dos animais para que estas sejam usados no processo de treinamento.

6.4 Trabalhos Futuros

Como possíveis trabalhos futuros seguem algumas implementações de acordo com o tema proposto:

- Utilizar Support Vector Machine como técnica de classificação, avaliando a performance e sua utilização para a classificação das imagens de grandes felinos.
- Adicionar mais classes de treinamento, verificando o impacto no desempenho ao se tentar identificar outras espécies de grandes felinos.
- Separar as imagens em subclasses de acordo com o tamanho da pelagem e utilizar as mesmas no processo de treinamento.

- Utilizar outras imagens para o treinamento, avaliando o desempenho da rede neural ao não somente se utilizar imagens de animais, mas incluindo, por exemplo, veículos, pessoas e objetos.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDREW, William; GREATWOOD, Colin; BURGHARDT, Tilo. **Visual Localisation and Individual Identification of Holstein Friesian Cattle via Deep Learning**, Venice, Italy, 29 out. 2017. Disponível em: < http://openaccess.thecvf.com/content_ICCV_2017_workshops/papers/w41/Andrew_Visual_Localisation_and_ICCV_2017_paper.pdf>. Acesso em: 02 jun. 2019

ARTERO, Almir Olivette. **Inteligência Artificial Teoria e Prática.** 1. ed. São Paulo: Livraria da Física, 2009.

BRAGA, Antônio; DE CARVALHO, André; LUDERMIR, Tereza. **Redes Neurais Artificiais Teoria e Aplicação.** 2. ed. Rio de Janeiro: LTC, 2011

CASTRO, Claudio de Moura. **A Prática da Pesquisa.** 2. ed. São Paulo: Pearson Prentice Hall, 2006.

CERVO, Amado L.; BERVIAN, Pedro A.; DA SILVA, Roberto. **Metodologia Científica.** 6. ed. São Paulo: Pearson Prentice Hall, 2007.

COPPIN, Ben. **Inteligência Artificial.** 1. ed. Rio de Janeiro: LTC, 2013.

CRALL, Jonathan P; STEWART, Charles V; BERGER-WOLF, Tanya Y; RUBENSTEIN, Daniel I; SUNDARESAN, Siva R. **HotSpotter — Patterned species instance recognition**, Tampa, FL, USA, 17 jan. 2013. Disponível em: < http://cs.rpi.edu/hotspotter/crall-hotspotter-wacv-2013.pdf >. Acesso em: 09 mai. 2019

DALFOVO, Michael Samir; LANA, Rogério Adilson; SILVEIRA, Amélia. **Métodos quantitativos e qualitativos: um resgate teórico.** Revista Interdisciplinar Científica Aplicada, Blumenau, v.2, n.4, p.0113, Sem II. 2008.

Data Science Academy. Deep Learning Book. 2017. Disponível em: < http://deeplearningbook.com.br/>. Acesso em: 09 mai. 2019.

DE MEDEIROS, Luciano Frontino. **Inteligência Artificial Aplicada Uma Abordagem Introdutória.** 1. ed. Curitiba: Intersaberes, 2018.

FERNANDES, Anita Maria da Rocha. **Inteligência Artificial noções gerais.** 1. ed. Florianópolis – SC: Visual Books Editora, 2003

GE, ZongYuan; MCCOOL, Christopher; SANDERSON, Conrad; CORKE, Peter. **Subset Feature Learning for Fine-Grained Category Classificationl**, Boston, MA, USA, 12 jun. 2015. Disponível em: < https://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W03/papers/Ge_Subset_Feature_Learning_2015_CVPR_paper.pdf>. Acesso em: 02 jun. 2019

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas S.A., 2002.

HAYKIN, Simon. **Redes Neurais Princípios e práticas.** 2. ed**.** Porto Alegre, 2001

JUPYTER. **Jupyter Documentation.** 2019. Disponível em: < https://jupyter.org/documentation>. Acesso em: 24 abr. 2019.

KERAS. **Why use Keras?** 2018. Disponível em: < https://keras.io/why-use-keras/ >. Acesso em: 20 abr. 2019.

LOPES, Isaias Lima; SANTOS, Flávia Aparecida Oliveira; PINHEIRO, Carlos Alberto Murari. **Inteligência Artificial.** 1. ed. Rio de Janeiro: Elsevier, 2014.

LUGER, George F. **Inteligência Artificial.** 6. ed. São Paulo: Person Education, 2014.

MARCONI, Marina de Andrade.; LAKATOS, Eva Maria. **Técnicas de pesquisa: planejamento e execução de pesquisas, amostragens e técnicas de pesquisas, elaboração, análise e interpretação de dados**. 7.ed. São Paulo: Atlas, 2008.

MARQUES FILHO, Ogê; VIERIA NETO, Hugo. **Processamento Digital de Imagens.** 1. ed. Rio de Janeiro: Brasport, 1999.

MESQUITA, Davi Padilha**. Estudo e modelagem dos efeitos da forma e do crescimento em processos de formação de padrões de pelagem via reação-difusão.** 2014. Trabalho de conclusão de curso (Biotecnologia). Instituto de Biociências, Universidade Federal do Rio Grande do Sul - UFRGS, Porto Alegre. Disponível em:<https://lume.ufrgs.br/handle/10183/117656>. Acesso em: 20 mai. 2019.

NASCIMENTO, Marina Laís da Silva; GUIMARÃES, Lamartine Nogueira Frutuoso; SHIGUEMORI, Elcio Hideiti. **Aplicação de técnicas de processamento de imagens no reconhecimento de marcos em rodovias para treinamento de redes**

neurais artificiais. São Paulo, v. 1, 2012. Disponível em:< http://mtc-m16c.sid.inpe.br/col/sid.inpe.br/mtc-m18/2013/01.07.12.55/doc/worcap2012_submission_15%20-%20Marina%20La%EDs%20da%20Silva%20Nascimento.pdf>. Acesso em: 19 abr. 2019.

OPENCV. **OpenCv Documentation.** 2019. Disponível em: < https://opencv.org/about/>. Acesso em: 10 nov. 2019.

PACHECO, G. C. Andre. **Classificação de espécies de peixe utilizando redes neurais convolucional**, Universidade Federal do Espírito Santo (UFES), Vitória - ES, Brasil, nov. 2016. Disponível em: < https://arxiv.org/pdf/1905.03642.pdf>. Acesso em: 31 out. 2019

PERKOVIC, Ljubomir. **Introdução à computação usando Python: um foco no desenvolvimento de aplicações.** 1. ed. Rio de Janeiro: LTC, 2016.

PONTI, Moacir A.; DA COSTA, Gabriel B. Paranhos. Como funciona o Deep Learning. In: VIERIA, Vaninha; RAZENTE, Humberto L.; BARIONI, Maria Camila N. (Org). **Tópicos em Gerenciamento de Dados e Informações.** 1. ed. São Paulo, 2017. p. 63-93 Disponível em:< http://sbbd.org.br/2017/wp-content/uploads/sites/3/2017/10/topicos-em-gerenciamento-de-dados-e-informacoes-2017.pdf >. Acesso em: 14 abr. 2019.

ROSA, João Luís Garcia. **Fundamentos da Inteligência Artificial.** 1. ed. Rio de Janeiro: LTC, 2011

RUSSEL, Stuart; NORVIG, Peter. **Inteligência Artificial.** Rio de Janeiro: Elsevier, 2013.

TENSORFLOW. **TensorFlow Documentation.** 2019. Disponível em: < https://www.tensorflow.org/about>. Acesso em: 20 abr. 2019.

TRNOVSZKY, Tibor; KAMENCAY, Patrik; ORJESEK, Richard; BENCO, Miroslav; SYKORA, Peter. **Animal Recognition System Based on Convolutional Neural Network**, University of Zilina, Slovakia, set. 2017. Disponível em: < https://pdfs.semanticscholar.org/68ce/c9c6572abae2fddde2ed47785df24eba1713.pdf>. Acesso em: 31 out. 2019