RECONHECIMENTO FACIAL DE BUGIOS-RUIVO ATRAVÉS DE REDES NEURAIS CONVOLUCIONAIS

Orlando Krause Junior, Andreza Sartori – Orientadora

Curso de Bacharel em Ciência da Computação
Departamento de Sistemas e Computação
Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brasil

orlando.krausejr@gmail.com, asartori@furb.br

Resumo: *Embora existam muitos estudos para a identificação facial automática em seres humanos, muito pouco é aplicado para a marcação e identificação de animais. Este trabalho apresenta o desenvolvimento de um protótipo utilizando Rede Neural Convolucional para o reconhecimento facial de bugios-ruivo. Com isso, pretende-se auxiliar os pesquisadores do Projeto Bugio - FURB no monitoramento dos animais, onde atualmente é realizado por meio de chips subcutâneos, que é invasivo e pode ser traumático, ou na experiência e conhecimento empírico do pesquisador. Para isso foram utilizadas as redes Inception-ResNet v2, ResNet50 e Xception. Para o treinamento destas redes foi utilizada a função de perda Triplet Loss para auxiliar o aprendizado da extração de características, a fim de permitir o One-Shot Learning, ou seja, aprender de um único exemplar. Dentre as redes testadas, a Xception foi a que apresentou melhor resultado, atingindo uma acurácia de 99,94%, enquanto a rede Inception-ResNet v2 atingiu 99,78% e a ResNet50 atingiu 75,98%.*

Palavras-chave*: Bugio-ruivo. Projeto Bugio-FURB. Rede Neural Convolucional. Triplet Loss. Xception. ResNet. Inception. One-Shot Learning.*

1 INTRODUÇÃO

O atual momento do Brasil impõe um grande desafio para a conservação da biodiversidade. O país cresce e, mesmo com a volatilidade momentânea no seu ritmo de desenvolvimento, expande os projetos de empreendimentos que, não raro, representam severas intervenções a ambientes naturais até então pouco ou nada alterados. Intervenções que podem tornar-se verdadeiros desastres se associadas às mudanças climáticas sobre as quais pouco se sabe (COSTA et al., 2013, p. 3).

Ainda segundo Costa et al. (2013, p. 19) a implantação de áreas protegidas é uma das estratégias mais eficientes para a conservação da biodiversidade. Contudo, além dos esforços para que as Unidades de Conservação (UCs) funcionem verdadeiramente para a conservação da biodiversidade em escala nacional, a gestão local dessas áreas deve garantir a conservação da biodiversidade e dos processos ecológicos naturais em escala regional. Nesse sentido, monitorar a integridade da biodiversidade local (in situ) em UCs ao longo do tempo é essencial para a tomada de decisão em uma gestão em nível local, regional e nacional.

Crouse et al. (2017, p. 1, tradução livre) apontam que a maioria das pesquisas sobre o comportamento e a ecologia de populações de animais selvagens requer que os sujeitos do estudo sejam individualmente reconhecíveis. Crouse et al. (2017, p. 2, tradução livre) indica ainda que os métodos atuais de identificação individual geralmente envolvem captura e marcação de animais com identificadores únicos, como combinações de colares e/ou marcas coloridas, ou aproveitando a variação natural em populações (por exemplo, cicatrizes, padrões de pele e pelagem) e confiando no conhecimento dos pesquisadores nas diferenças individuais.

O Projeto Bugio, criado pela FURB, tem como principal objetivo a preservação do bugio-ruivo. Dentre as atividades do projeto está o monitoramento de indivíduos, com o intuito de estudar e preservar a espécie. Atualmente este monitoramento é feito através de chips subcutâneos, porém para que seja aplicado, se faz necessário a captura e sedação do indivíduo, que pode acabar gerando sequelas.

Visto a importância da identificação dos indivíduos por um método não invasivo, e em parceria com o Projeto Bugio-FURB, este trabalho propõe o desenvolvimento de um protótipo para reconhecimento facial de bugios-ruivo utilizando Redes Neurais Convolucionais. Para isso foram testadas três Redes Neurais Convolucionais, sendo elas: Inception-ResNet v2, ResNet50 e Xception. A técnica Triplet Loss foi utilizada nas redes neurais para treiná-las na extração de características, enquanto para classificação, foram utilizados os algoritmos de Aprendizado de Máquina K-Nearest Neighbors (kNN), Radius Nearest Neighbors (rNN) e Support Vector Machine (SVM). Com isso, foi atingido o One-Shot Learning, ou seja, foi possível aprender novas classes a partir de poucos exemplos treinando apenas o algoritmo de classificação. Além disso, foi construído uma aplicação para demonstrar as funcionalidades desenvolvidas neste protótipo. Nesta aplicação é possível prever o nome do bugio a partir de uma imagem e adicionar novos imagens na base utilizada na previsão. Os objetivos específicos são: (i) construir uma base de dados de imagens de bugios-ruivo em conjunto com os profissionais do Projeto Bugio-FURB; (ii) extrair características relevantes utilizando os modelos de Redes Neurais Convolucionais Inception-ResNet v2, ResNet50 e Xception; (iii) utilizar os classificadores KNN, rNN e

SVM a fim de reconhecer os indivíduos a partir das características extraídas pela rede neural; (iv) permitir One-Shot Learning, ou seja, aprender a classificar um indivíduo a partir de um único exemplo; (v) desenvolver aplicação com interface para realizar o reconhecimento facial.

Este trabalho acelera o processo de reconhecimento de indivíduos, que atualmente são necessárias muitas horas de campo para que sejam coletadas estas informações. Além do tempo ganho, este trabalho busca reduzir os erros de amostragem e melhorar a qualidade dos dados coletados. O trabalho também permitirá a identificação de indivíduos que sobreviveram a febre amarela em casos de epizootias (doença que ocorre em populações de animais, similar a epidemia em seres humanos).

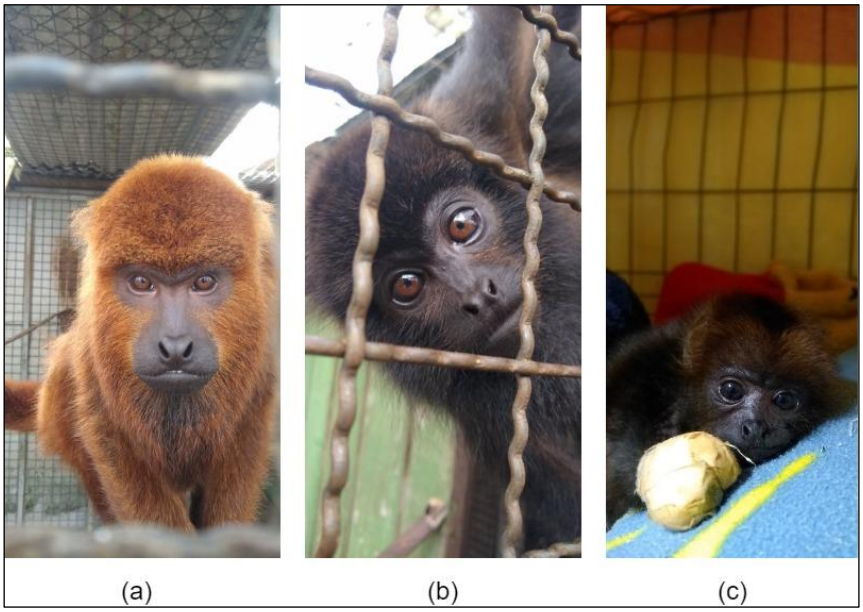
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo explorar os assuntos e técnicas abordadas neste trabalho. A seção 2.1 apresenta as características sobre o objeto de estudo deste trabalho, os bugios-ruivos. A seção 2.2 aborda os principais conceitos de Redes Neurais Artificiais, as características e conceitos de Redes Neurais Convolucionais e o conceito de *One Shot Learning*. Por fim, a seção 2.3 discute os trabalhos correlatos a este.

2.1 BUGIO-RUIVO E O PROJETO BUGIO-FURB

O bugio-ruivo (subespécie *Alouatta Guariba Clamitans*), exibido na Figura 1, está entre os maiores primatas da américa do sul, com um comprimento entre a cabeça e o corpo de 45 a 58 centímetros e um comprimento de cauda de 48 a 67 centímetros. Pesa entre 4 a 7 kg e possui dimorfismo sexual (diferenças entre machos e fêmeas) onde os machos são ruivos/avermelhados, conforme Figura 1(a) e as fêmeas castanho-escuro (Figura 1(b)). Eles vivem em grupos, em árvores de altura entre 10 a 20 metros, tendo um macho como líder do grupo. Possuem hábitos diurnos e se alimentam de folhas e frutos (BICCA-MARQUES et al., 2014).

Figura 1 – Imagem de Bugios-ruivos: da esquerda para a direita – macho, fêmea e bebê



Fonte: elaborado pelo autor.

A subespécie *Alouatta Guariba Clamitans* é o primata mais abundante no Estado de Santa Catarina e é considerado como próximo de ameaça, ou seja, com perigo de extinção em médio prazo, pela União Internacional de Conservação da Natureza (IUCN). A destruição do habitat natural e o tráfico são as principais causas da diminuição das populações livres e do aumento do número de animais cativos em centros de triagem, criadouros científicos e conservacionistas, e consequentemente, de um maior contato com o homem e seus animais domésticos (SOUZA JUNIOR, 2007).

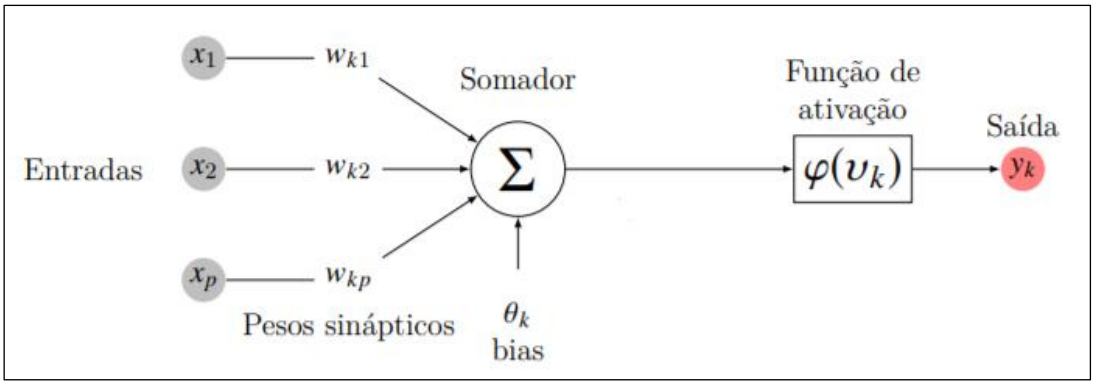
O Projeto Bugio foi criado em 1991 em uma parceria entre a Prefeitura Municipal de Indaial e Fundação Universidade Regional de Blumenau (FURB) e tem como principal objetivo a preservação da espécie *Alouatta Guariba Clamitans*, o bugio-ruivo. O local é destino de animais trazidos pelo Ibama, pela Polícia Ambiental ou ainda pelos moradores locais por apresentarem ferimentos tendo como casos mais comuns atropelamentos, choque elétrico e ataque por outros animais. O projeto também capacita estudantes e profissionais interessados nas áreas de primatologia, biologia e medicina da conservação. Além disso, busca sensibilizar a comunidade regional quanto à importância da preservação ambiental. Atualmente para realizar o monitoramento da espécie, o projeto utiliza microchips subcutâneos como técnica

de marcação dos indivíduos, este método depende da captura dos indivíduos. Em animais não capturados o reconhecimento depende de variações da cor da pelagem, cicatrizes ou ainda pelo conhecimento visual do pesquisador.

2.2 REDE NEURAL ARTIFICAL

Segundo Haykin (2003, p. 27), as Redes Neurais Artificiais (RNA) têm como inspiração o funcionamento do cérebro humano, visto que este processa informações de forma diferente em relação ao computador digital, sendo ele altamente complexo, não-linear e paralelo. Ainda segundo Haykin (2003, p. 28), uma RNA é uma máquina projetada para modelar a forma como o cérebro executa as tarefas. A unidade mais simples de uma RNA é chamada de neurônio, onde é realizado o processamento de informações. Cada neurônio pode estar conectado a vários outros e para cada conexão um peso é atribuído. Para calcular a saída de um neurônio é aplicada uma função de ativação sobre a soma de cada entrada multiplicada pelos seus pesos (CUSTÓDIO, 2019, p. 25). A Figura 2 representa um neurônio com suas entradas, pesos, função de ativação e saída.

Figura 2 - Representação de um neurônio simples



Fonte: Custódio (2019).

Uma RNA pode ser dividida em três partes, uma camada de entrada, uma ou mais camadas ocultas que são responsáveis por realizar o processamento e uma camada de saída, onde cada camada é composta por um ou mais neurônios. A arquitetura de uma RNA é definida, entre outros fatores, pelo número de camadas, pela quantidade de neurônios em cada camada e pelo tipo de conexão entre os neurônios, podendo ser Feedfoward, onde os dados fluem em uma única direção, ou Feedback (redes recorrentes), onde a rede neural é realimentada com a saída da mesma (AFFONSO, 2010, p. 36).

Para que uma RNA funcione é necessário definir seus pesos, onde este processo de "descoberta" dos pesos é chamado de treinamento. O Backpropagation (ou retro propagação) é um algoritmo utilizado para reajuste dos pesos, onde atualiza os pesos tendo como base o retorno da função de erro. A função de erro é responsável por calcular quão imprecisa foi a saída da rede, comparando o valor gerado pela rede com o valor esperado no treinamento. Com isso o processo de treinamento consiste em: alimentar a rede, calcular o erro e atualizar os pesos de forma a minimizar o valor do erro. Estes passos devem ser executados por várias iterações até que atinja algum critério de parada (CUSTÓDIO, 2019, p. 31).

2.2.1 Rede Neural Convolucional

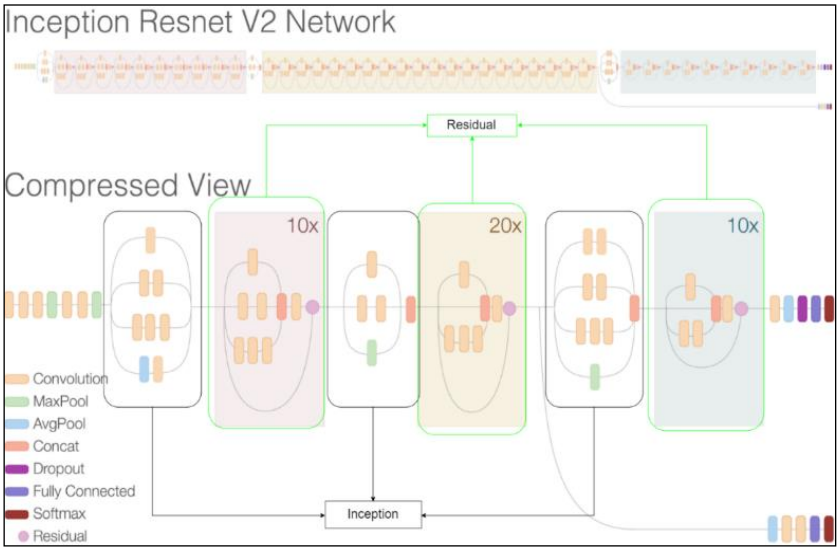
A habilidade de uma RNA de múltiplas camadas de aprender características complexas, torna ela uma forte candidata para tarefas de reconhecimento de imagens. Em uma abordagem tradicional, um extrator de características é construído manualmente para obter as características relevantes da imagem, estas características são a entrada para uma RNA de classificação. Uma abordagem mais interessante seria a própria RNA aprender a extrair as características relevantes, sendo assim, sua entrada seria a própria imagem. Utilizar um modelo padrão de RNA para realizar tarefas utilizando imagens como entrada pode acarretar alguns problemas. Por exemplo, se a entrada utilizar uma imagem 100x100 (10.000 pixels) e tiver uma camada completamente conectada com 100 neurônios, é necessário treinar 1 milhão (10.000 pixels x 100 neurônios) de pesos somente para esta camada, tornando o treinamento lento e aumentando a utilização de recursos de hardware. Para resolver este problema foi desenvolvida a Rede Neural Convolucional (LECUN et al., 1998, p. 5).

Uma Rede Neural Convolucional (Convolutional Neural Network - CNN), apresentada na Figura 3, possui uma ou mais camadas ocultas onde são executadas operações de convolução (*Convolution*), responsáveis por extrair as características da imagem, seguidas por camadas completamente conectadas (*Fully Connected* e *Softmax*), responsáveis pela classificação. Nas camadas de convolução os pesos treinados são na verdade matrizes, também chamada de máscaras, que são utilizadas na operação de convolução. Uma das vantagens desta abordagem é o uso de pesos compartilhados, pois

uma mesma máscara é utilizada em todos os neurônios de entrada. Desta forma, menos pesos são treinados melhorando o tempo de treinamento e utilização de recursos. Outra vantagem é que a operação consegue destacar uma mesma característica em qualquer lugar da imagem (VARGAS; PAES; VASCONCELOS, 2016, p. 2).

Neste trabalho foram utilizadas 3 modelos de Redes Neurais Convolucionais para executar a classificação, Inception-ResNet v2, ResNet50 e Xception. O primeiro foi a Rede Neural Inception-ResNet v2, que tem esse nome devido as duas técnicas que utiliza, Inception e Residual Network (SZEGEDY et al., 2017). A Figura 3 apresenta a arquitetura completa desta rede, tendo três blocos Inception (circulados na cor preta), e três blocos residuais diferentes (circulados em verde) onde o primeiro bloco é repetido 10 vezes, o segundo bloco 20 vezes e o terceiro bloco é repetido 10 vezes. As próximas camadas são responsáveis pela classificação, sendo uma camada completamente conectada e uma camada com a função de ativação Softmax que é a saída da rede.

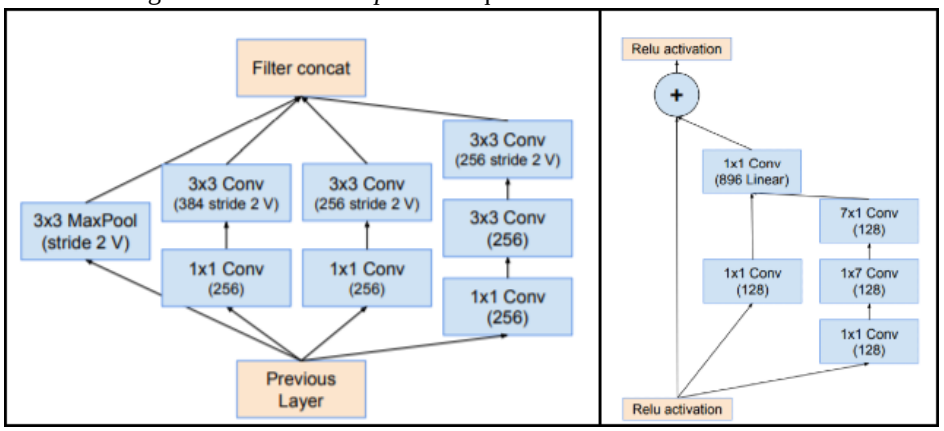
Figura 3 – Arquitetura da Rede Neural Convolucional Inception-ResNet v2



Fonte: Szegedy et al. (2017).

A técnica Inception consiste em utilizar uma rede neural dentro de outra rede neural. Na Figura 4, à esquerda, é exibido um módulo Inception. Uma das vantagens desta abordagem é que é possível utilizar filtros de vários tamanhos sobre uma mesma entrada, desta forma, podendo reconhecer características em escalas diferentes, e ao final enviar para a próxima camada as características extraídas de todos os filtros. Além disso o custo computacional desta operação é relativamente baixo (SZEGEDY et al. 2015). A Rede Residual (*Residual Network*) foi proposta por He et al. (2016) para resolver problemas de aprendizado em redes neurais muito profundas, ou seja, com muitas camadas. Na Figura 4 à direita, é exibido um bloco residual, que consiste em executar operações na entrada e ao final somar o resultado destas operações a entrada novamente, este processo é chamado de *Skip Connections* (pular conexões). Com isto, caso os pesos das camadas internas produzam um valor zerado, ao somar com a entrada o valor não mudará, e assim, não afeta as próximas camadas (HE et al. 2016).

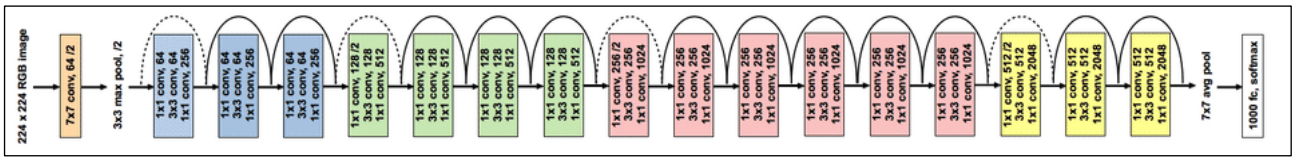
Figura 4 - Módulo *Inception* à esquerda e bloco residual à direita



Fonte: Adaptado de Szegedy et al. (2017).

A segunda Rede Neural Convolucional utilizada neste trabalho foi a ResNet50 (HE et al., 2016), foi nomeada com a abreviação de *Residual Network* e o número de camadas que utiliza (neste caso, 50). Assim como a rede anterior, esta rede utiliza a técnica blocos residuais, porém esta rede foi desenvolvida para validar a técnica de blocos residuais, sendo a primeira a utilizar esta técnica. A Figura 5 mostra a arquitetura desta rede. As linhas contínuas sobre os blocos representam os blocos residuais, enquanto a linha pontilhada além de ser um bloco residual, realiza uma redução de dimensionalidade dos mapas de características.

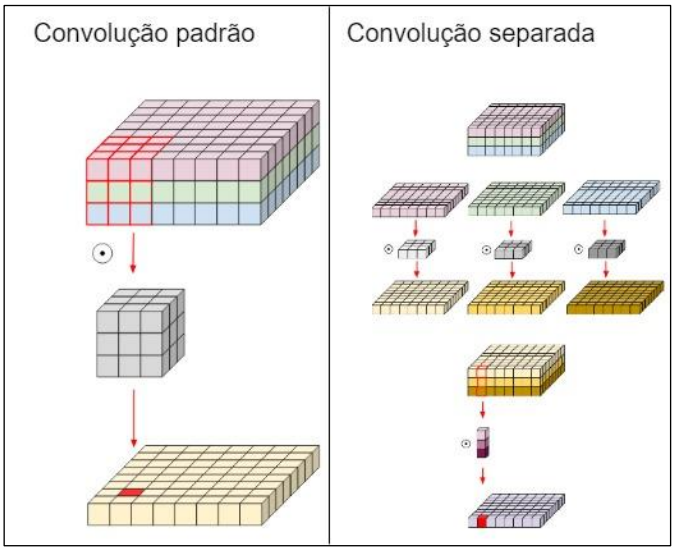
Figura 5 - Arquitetura da Rede Neural Convolucional ResNet50



Fonte: adaptado de He et al. (2016).

Por fim, a terceira Rede Neural Convolucional utilizada foi a Xception (CHOLETT, 2017), seu nome vem da técnica utilizada, chamada de Extreme Inception. Esta técnica baseia-se na rede neural Inception (SZEGEDY et al., 2015), mas utiliza a técnica Depthwise Separable Convolution (convolução separada em profundidade, tradução livre). Este tipo de convolução consiste em utilizar filtros diferentes para cada canal da imagem, ou seja, ao invés de utilizar um único filtro de três dimensões, utiliza um filtro para cada canal e em seguida aplica uma convolução padrão com um filtro de tamanho 1x1. A Figura 6 mostra a diferença entre as duas convoluções.

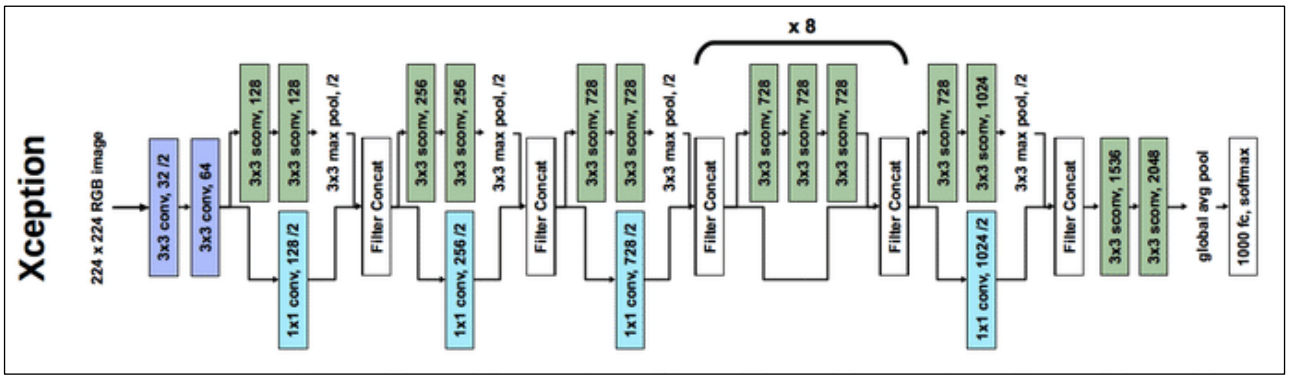
Figura 6 - Diferença entre convolução padrão e convolução separada



Fonte: adaptado de Pandey (2018).

Além da técnica de convolução separada, a rede Xception também utiliza a técnica de blocos residuais. Na Figura 7 é apresentada a arquitetura da rede Xception, onde retângulos verdes significam uma camada de convolução separada. Na Figura 7 é possível observar que a rede inicia com duas camadas que utilizam convolução padrão, seguido de três blocos residuais com convolução separada. Em seguida, a rede apresenta mais oito blocos residuais que geram mapas de características através de convolução separada. Por fim, é composta por mais um bloco residual com convolução separada e mais duas camadas de convolução separada e uma camada completamente conectada responsável por realizar a classificação.

Figura 7 - Arquitetura da Rede Neural Convolucional Xception



Fonte: adaptado de Cholett (2017).

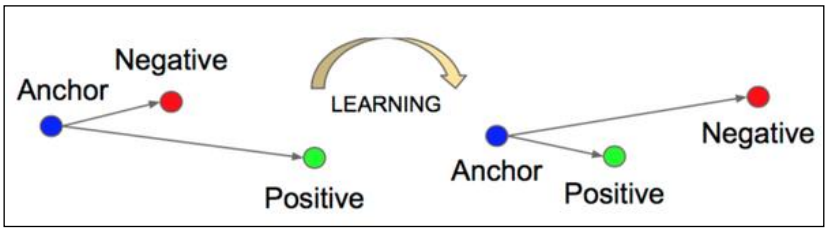
2.2.2 One-Shot Learning

Segundo Vinyals et al. (2016), aprender algo com poucos dados é um desafio em aprendizado de máquina. Nós humanos conseguimos aprender com pouca supervisão, por exemplo, uma criança consegue generalizar o conceito de girafa a partir de uma simples figura em um livro, enquanto uma rede neural artificial precisa de centenas ou milhares de exemplos. Este problema motivou o conceito de One-Shot Learning (aprendizado em um tiro), que consiste em aprender a identificar uma classe a partir de um ou poucos exemplos (VINYALS et al., 2016, p. 1). Para permitir o One-Shot Learning, este trabalho utilizou a função de perda Triplet Loss.

Schroff, Kalenichenko e Philbin (2015) utilizaram a técnica Triplet Loss (Perda Tripla) para treinar uma rede neural a fim de aprender como extrair características de uma face (no artigo é chamado de *face embeddings*). A rede neural desenvolvida pelos autores tem como saída um vetor com 128 características, onde, para classificação é verificado a distância euclidiana entre as características de duas faces. Com isso, a classificação pode ser feita com algoritmos mais simples como KNN ou SVM. Desta forma, quando uma nova classe for adicionada, a extração de características continua igual e o algoritmo de classificação consegue aprender a partir de um único exemplar.

A Figura 8 mostra o resultado de um treinamento utilizando a técnica Triplet Loss. Para calcular a perda, esta técnica utiliza três faces sendo uma delas a âncora (*anchor*) que é utilizado como base, outra é um exemplo negativo (*negative*), ou seja, de classe diferente da âncora e, por fim, um exemplo positivo (*positive*) de mesma classe da âncora. O erro gerado pela função Triplet Loss se baseia na distância euclidiana das características geradas, com o objetivo de distanciar exemplos negativos e aproximar exemplos positivos (SCHROFF; KALENICHENKO; PHILBIN, 2015).

Figura 8 - Treinamento utilizando *triplet loss*



Fonte: Schroff, Kalenichenko e Philbin (2015).

2.3 TRABALHOS CORRELATOS

Neste capítulo serão apresentados três trabalho correlatos. O primeiro trabalho, apresentado no Quadro 1, foi desenvolvido por Crouse et al. (2017) e tem como objetivo reconhecer lêmures através de características faciais extraídas com técnicas manuais. No Quadro 2 é apresentado o trabalho desenvolvido por Loos e Ernst (2013), que tem como objetivo reconhecer chimpanzés através de características faciais. A extração de características também é feita por meio de técnicas manuais, porém diferente do trabalho anterior, utiliza algoritmos de Aprendizado de Máquina para a classificação. Por fim, o trabalho de Schofield et al. (2019) no Quadro 3, também realiza o reconhecimento facial de chimpanzés, porém utiliza modelos de CNN para a classificação, além da localização e classificação de gênero. O trabalho de Schofield et al. (2019) substituiu o trabalho de Bolger et al. (2012), citado na proposta deste trabalho, pois mostra uma correlação maior por utilizar técnicas semelhantes e ter como objeto de estudo um primata.

Quadro 1 – Trabalho Correlato 1

Referência	Crouse et al. (2017)
Objetivos	Identificar lêmures através de reconhecimento facial
Principais funcionalidades	Reconhecimento facial de lêmures da espécie *Eulemur Rubriventer* com extração de características através de técnicas de Processamento de Imagens e classificação baseado na distância das características. A base de dados utilizada possui 462 imagens de 80 lêmures diferentes.
Ferramentas de desenvolvimento	Extração de características através das técnicas diferença de Gaussianas e Multi-scale Local Binary Pattern (MLBP) e classificação utilizando distância entre as características.
Resultados e conclusões	Foi necessária a intervenção manual para marcação da posição dos olhos, com isso atingiu um resultado de 93.3% de precisão

Fonte: elaborado pelo autor.

O trabalho de Crouse et al. (2017) se relaciona com o objetivo do presente trabalho, pois utiliza de características faciais para identificação de indivíduo, neste caso, o lêmure da espécie *Eulemur Rubriventer*. Porém o trabalho de Crouse et al. (2017) aplicou somente técnicas de Processamento de Imagens, isto é, utilizou para o reconhecimento facial a diferença de Gaussianas e MLBP, ou seja, não utilizou de aprendizado de máquina para realização desta identificação.

Quadro 2 - Trabalho Correlato 2

Referência	Loos e Ernst (2013)
Objetivos	Reconhecimento facial para chimpanzés
Principais funcionalidades	Detecção da posição da face e dos olhos através de técnicas de Processamento de Imagens. Também utilizou de técnicas de Processamento de Imagens para extração de características e, por fim, classificação utilizando Aprendizado de Máquina. Utilizou duas bases diferentes, a base Zoo com 2617 imagens de 24 indivíduos e a base Taï com 3905 imagens de 71 indivíduos.
Ferramentas de desenvolvimento	Extração de pontos-chave (face e olhos) utilizando de filtro Sobel e Local Binary Patter (LBP), extração de características com filtro Gabor e Speed Up Robust Features (SURF) e classificação utilizando Sparse Representation Classifier (SRC) e Support Vector Machine (SVM).
Resultados e conclusões	Atingiu 88.11% de acurácia na base Zoo e 73.31% na base Taï. Utilizando a detecção de face e olhos a acurácia caiu para 84.10% e 68.97% respectivamente

Fonte: elaborado pelo autor.

O trabalho de Loss e Ernst (2013) também se relaciona com este pelo seu objetivo, que é reconhecer o indivíduo através de características faciais, neste caso, os chimpanzés. Diferente do trabalho anterior, os autores propõem localizar a face e os olhos na imagem automaticamente, e, a partir desta, realizar a rotação e recorte da imagem. A localização foi implementada utilizando filtro Sobel e a técnica LBP. Para a extração de características foram usadas as técnicas SURF e filtro Gabor. Por fim, para classificação os autores utilizaram os modelos SVM e SRC. O trabalho atual, diferente deste correlato, utiliza Redes Neurais Artificiais para a extração de características, não sendo necessário a aplicação de técnicas manuais.

Quadro 3 - Trabalho Correlato 3

Referência	Schofield et al. (2019)
Objetivos	Reconhecimento facial de chimpanzés
Principais funcionalidades	Detecção da posição da face utilizando Redes Neurais Convolucionais, assim como para o reconhecimento facial e identificação de gênero. A base de dados utilizada possui 50 horas de vídeo contendo ao todo 23 indivíduos.
Ferramentas de desenvolvimento	Detecção de face foi feita utilizando CNN com base na rede do grupo Visual Geometry Group, VGG-16, do tipo Single Shot Detector (SSD). Para o reconhecimento facial e para identificação de gênero também foi utilizada uma CNN, em ambas foram utilizadas como base a rede VGG-M.
Resultados e conclusões	Acurácia de 81% na detecção de face, 92.47% no reconhecimento facial e 96.16% na identificação de gênero.

Fonte: elaborado pelo autor.

O trabalho de Schofield et al. (2019) também realiza o reconhecimento facial de chimpanzés, porém, utiliza uma abordagem diferente para extração das características. Assim como este trabalho, Schofield et al. (2019) utiliza um modelo de CNN para as tarefas de extração de características. A detecção de face foi feita através de uma CNN com base na VGG-16 do tipo SSD. Com a face localizada, a mesma é extraída e enviada para as próximas redes neurais. Tanto o reconhecimento facial quanto a identificação de gênero têm como entrada a face extraída na etapa anterior, a classificação nestes passos é feita através de uma CNN, que tem como base a rede VGG-M. Além disso, neste correlato a classificação do indivíduo é feita pela própria rede neural que extrai as características, enquanto no presente trabalho a classificação

foi separada da extração de características, onde a rede neural é responsável pela extração de características e a classificação por um algoritmo de aprendizado de máquina. Além disso, diferente do presente trabalho, Schofield et al. (2019) realiza a detecção da face e identificação de gênero.

3 ESPECIFICAÇÃO E DESENVOLVIMENTO DO PROTÓTIPO

Nesta seção serão apresentados os aspectos mais relevantes relacionados à especificação e desenvolvimento do protótipo para reconhecimento facial de bugios-ruivo. São abordados os requisitos do protótipo, seguidos de sua especificação e desenvolvimento da rede neural artificial.

3.1 REQUISITOS

Nesta seção são apresentados os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) do protótipo desenvolvido, na qual são apresentados nos quadros Quadro 4 e Quadro 5 respectivamente.

Quadro 4 - Requisitos Funcionais

RF01 – Permitir envio de imagem pelo usuário.
RF02 – Extrair características relevantes da imagem informada.
RF03 – Realizar classificação da imagem baseada nas características extraídas.
RF04 – Exibir resultado da classificação para o usuário.
RF05 – Permitir ao usuário adicionar novo indivíduo a base de dados existente.

Fonte: elaborado pelo autor.

Quadro 5 - Requisitos Não Funcionais

RNF01 – A aplicação deve ser construída utilizando a linguagem Python.
RNF02 – A aplicação deve utilizar o framework Keras para criação e treinamento do modelo.
RNF03 – A extração de características deve ser feita utilizando Redes Neurais Convolucionais.
RNF04 – Utilizar algoritmos k-NN, SVM e r-NN para classificação do indivíduo

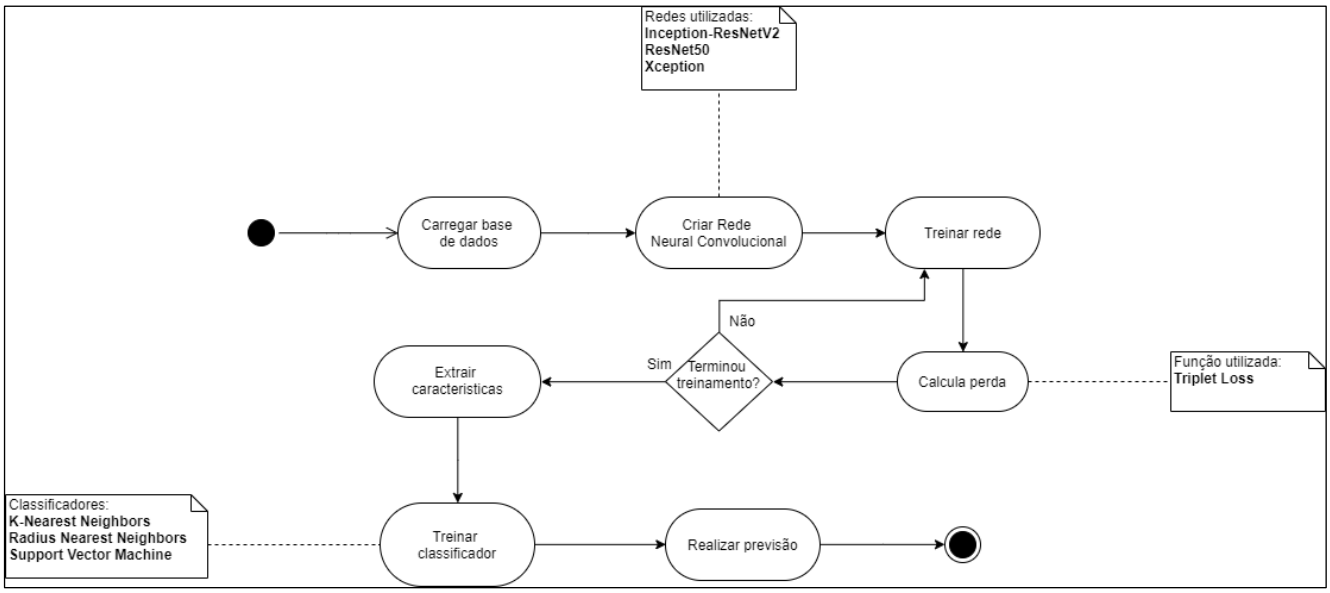
Fonte: elaborado pelo autor.

3.2 DESENVOLVIMENTO DO PROTÓTIPO

Esta seção apresenta o protótipo implementado. A primeira seção traz informações sobre a base de dados utilizada para o treinamento. A seção 3.2.2 apresenta a arquitetura da rede neural desenvolvida. Na seção 3.2.3 contém detalhes sobre o treinamento da rede neural. A seção 3.2.4 explica sobre o processo de inferência da rede neural. Por fim, a seção 3.2.5 apresenta a aplicação desenvolvida neste trabalho.

A Figura 9 apresenta o diagrama de atividades do processo de treinamento do protótipo desenvolvido. A primeira etapa do processo é o carregamento das imagens. Neste passo as imagens são redimensionadas para que sejam utilizadas no treinamento da rede. Em seguida é construída a Rede Neural Convolucional, na qual foram usadas três redes Inception-ResNet V2, ResNet50 e Xception. Então, é iniciado o treinamento da rede neural, onde é utilizada a função Triplet Loss para calcular a perda. Com a rede treinada, o próximo passo é extrair as características com a rede treinada, a fim de treinar o classificador. Para classificação foram testados três algoritmos k-NN, r-NN e SVM. Com o classificador treinado é possível então realizar a previsão.

Figura 9 - Diagrama de atividades



Fonte: elaborado pelo autor.

3.2.1 Base de Dados

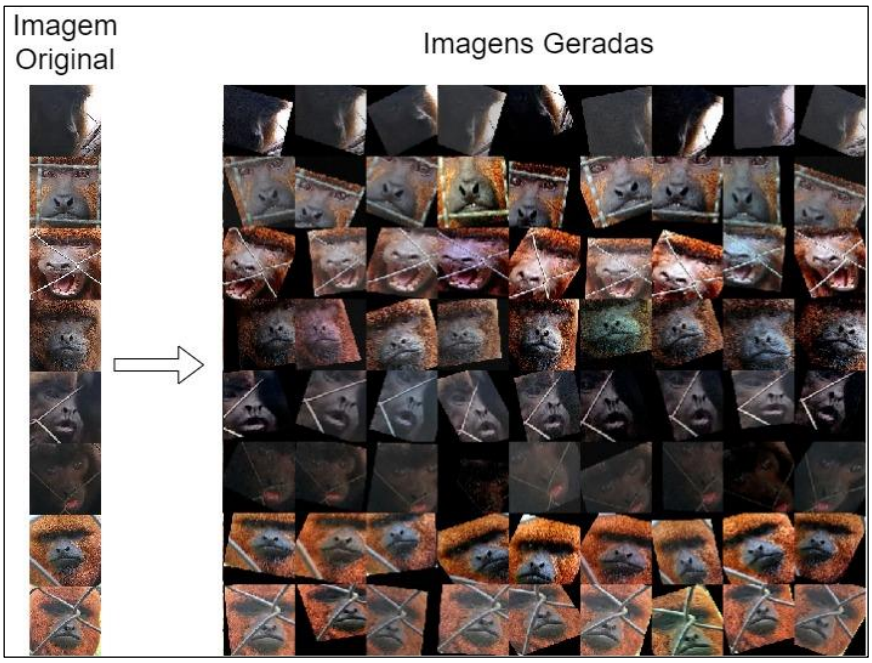
Para a construção da base de dados, foram coletadas 179 fotos de 20 bugios diferentes, obtidas no Projeto Bugio-FURB. Algumas destas imagens possuem variação de rotação, pose, iluminação e obstrução, conforme apresentado na Figura 10. As imagens foram recortadas manualmente para que contenham somente a face do animal, a fim de treinar o algoritmo para realizar o reconhecimento facial.

Figura 10 - Imagens da base de dados utilizada

Fonte: elaborado pelo autor.

Além destas imagens, foi utilizada a estratégia *data augmentation* a fim de aumentar a quantidade de imagens para a base de dados. Isto é, para cada imagem na base de dados, foram aplicadas transformações (iluminação, rotação, escala, embaçamento) e gerado mais 9 imagens, totalizando uma base de 1790 imagens. A Figura 11 apresenta um exemplo da imagem original (coluna à esquerda) e suas respectivas imagens geradas.

Figura 11 - *Data augmentation* aplicado a base de dados



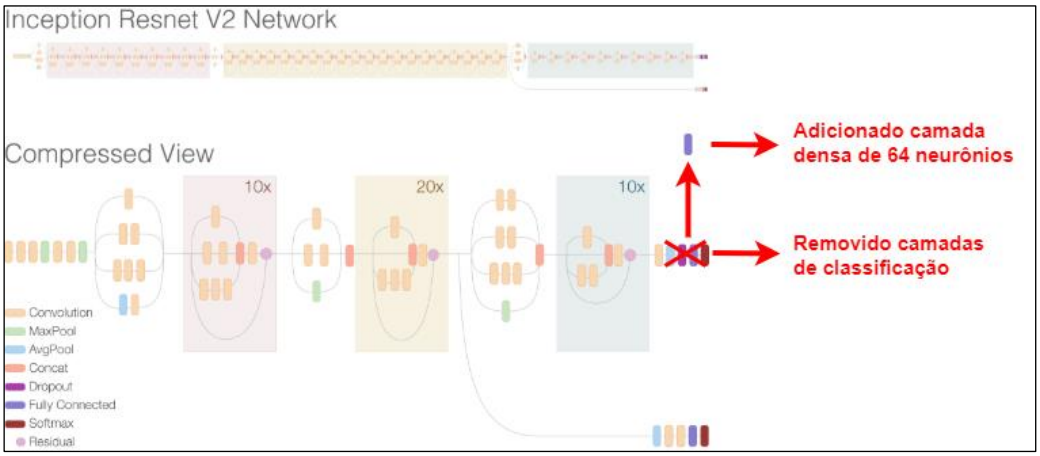
Fonte: elaborado pelo autor.

3.2.2 Arquitetura das Redes Neurais Convolucionais

Para este trabalho foram utilizadas três diferentes arquiteturas de CNN, são elas: Inception-ResNet v2 (SZEGEDY et al., 2017), ResNet50 (HE et al., 2016) e Xception (CHOLETT, 2017). Estas arquiteturas em seu desenvolvimento original, foram feitas e testadas com o objetivo de classificação da base de dados ImageNet (RUSSAKOVSKY et al., 2015). Neste trabalho as redes neurais não realizam classificação, mas sim realizam a extração de características, desta forma foram necessários alguns ajustes na arquitetura original.

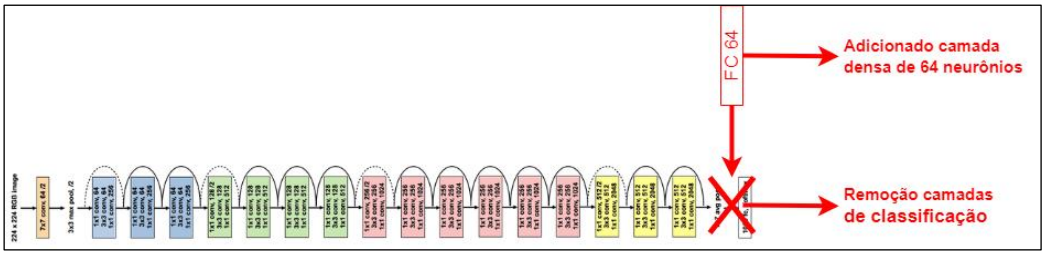
Para este trabalho a camada de entrada foi alterada para receber uma imagem de tamanho 96x96, bem como foram removidas as camadas finais da rede que são responsáveis pela classificação. Foi adicionada também uma camada completamente conectada com 64 neurônios, desta forma a saída da rede é um vetor de 64 características e não mais a classe prevista. Uma ilustração desta alteração para as três arquiteturas escolhidas, Inception-ResNet v2, ResNet50 e Xception é apresentada na Figura 12, Figura 13 e Figura 14 respectivamente. O Quadro 6 apresenta o código fonte em Python desta alteração utilizando o framework Keras.

Figura 12 - Alteração na arquitetura Inception-ResNet v2



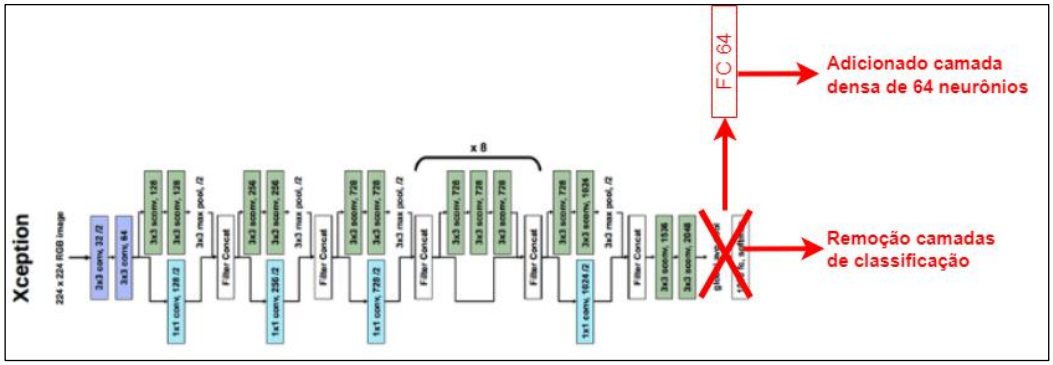
Fonte: adaptado de Szegedy et al. (2017).

Figura 13 - Alteração na arquitetura ResNet50



Fonte: adaptado de He et al. (2016).

Figura 14 - Alteração na arquitetura Xception



Fonte: adaptado de Cholett (2017).

Quadro 6 - Código fonte da criação da rede

```
1  inceptionResNetV2 =
   tf.keras.applications.inception_resnet_v2.InceptionResNetV2(include_top=False,
   input_shape=input_shape)
2  last_layer = inceptionResNetV2.output
3  x = tf.keras.layers.Flatten()(last_layer)
4  x = tf.keras.layers.Dense(64)(x)
5  x = tf.keras.layers.Lambda(lambda embeddings:
   tf.keras.backend.l2_normalize(embeddings, axis=1))(x)
6  model = tf.keras.models.Model(inceptionResNetV2.input, x)
```

Fonte: elaborado pelo autor.

O código apresentado no Quadro 6 é responsável por criar o modelo de rede neural baseado no modelo Inception-ResNet v2. O método `tf.keras.applications.inception_resnet_v2.InceptionResNetV2` utilizado na linha 1 é responsável por criar a rede Inception-ResNet v2. O parâmetro `include_top` com valor falso indica ao Keras que ele não deve inserir as camadas finais da rede, que são responsáveis pela classificação, enquanto o parâmetro `input_shape` informa o tamanho da imagem de entrada, que neste trabalho foi definido como 96x96. O método `tf.keras.layers.Flatten`, utilizado na linha 3, adiciona uma camada ao final do modelo criado anteriormente. Esta camada é responsável por transformar uma matriz em vetor, por exemplo, uma matriz 3x3 se transforma em um vetor de 9 elementos. O método `tf.keras.layers.Dense(64)`, na linha 4, adiciona uma camada completamente conectada com 64 neurônios, que é responsável por gerar as características que serão previstas. O método `tf.keras.layers.Lambda(lambda embeddings: tf.keras.backend.l2_normalize (embeddings, axis=1))`, na linha 5, adiciona uma camada responsável por aplicar uma normalização no vetor de características. A função `tf.keras.models.Model`, na linha 6, instancia a nova arquitetura com as alterações descritas. Para as outras arquiteturas testadas, foi alterada a linha 1 para utilizar o método `tf.keras.applications.resnet50.ResNet50` para a arquitetura ResNet50 e `tf.keras.applications.xception.Xception` para a arquitetura Xception.

A função de perda utilizada na rede neural proposta é a Triplet Loss, esta função penaliza características de classes diferentes que estão próximas e penaliza características da mesma classe que estão distantes. Para o cálculo da perda, a função Triplet Loss utiliza todas as imagens enviadas no lote (*batch*) de treinamento. Neste trabalho foi utilizada a implementação de Triplet Loss da biblioteca TensorFlow. Com a utilização da Triplet Loss, a Rede Neural não precisará ser alterada quando for necessário adicionar uma nova classe, pois ela realiza somente a extração de características, sendo responsabilidade do classificador aprender a nova classe. Desta forma podemos obter o aprendizado de um único exemplar, atingindo o One-Shot Learning.

3.2.3 Treinamento

Para realizar o treinamento da rede, a base de dados foi dividida em duas partes, sendo 85% para treinamento e 15% para validação. Foram utilizadas 20 épocas (*epochs*) para este treinamento, onde cada época possui 40 lotes, e cada lote contém 64 imagens. O algoritmo de otimização utilizado, responsável pela atualização dos pesos na rede neural, foi o Adam, com uma taxa de aprendizado de 0,001. Tanto o algoritmo de otimização quanto a taxa de aprendizado foram escolhidos empiricamente.

Durante os testes com a função Triplet Loss foi identificado um problema no treinamento. Em alguns lotes de treinamento as imagens estavam desbalanceadas, e, em alguns casos havia uma única imagem para uma classe. Isto se tornou um problema pois a função Triplet Loss necessita de pelo menos duas imagens da mesma classe (âncora e positivo), além de uma de classe distinta (negativo) para que a mesma consiga contribuir efetivamente para o aprendizado.

Para resolver este problema foi implementado um gerador de lotes de forma que as classes ficassem balanceadas dentro de cada lote. Esta função consiste em selecionar aleatoriamente as classes do lote e então selecionar aleatoriamente as imagens de cada classe, na qual possui dois parâmetros principais: quantidade de classes por lote e quantidade de imagens por classe. Nos testes realizados foi atribuído o valor 8 para estes dois parâmetros, tendo assim um lote de 64 imagens.

No Quadro 7 é exibido o código implementado para o gerador de lotes. Na linha 3, a função `def generator` é responsável por selecionar as imagens. Na linha 5 é escolhida aleatoriamente as classes que serão utilizadas neste lote. A quantidade de classes é definida pela variável `num_classes_per_batch` que foi escolhida empiricamente com o valor 8. Na linha 6 é iterado sobre cada classe selecionada anteriormente e na linha 7 são obtidas todas as imagens referentes a classe que está selecionada. Na linha 8 começa a seleção de imagens da classe, onde a quantidade de imagens por classe é definida pela variável `num_images_per_class` que também foi escolhida empiricamente como 8. Para seleção é escolhido aleatoriamente uma imagem entre todas as disponíveis para aquela imagem (Linha 9, 10). A imagem é carregada e retornada junto com sua respectiva classe (Linha 11), na qual encerra o gerador de imagens. A função `batch_generator`, na linha 12, utiliza o gerador criado anteriormente (linha 14) para criar o lote de imagens. Em seguida são criados os `arrays` que irão armazenar as imagens com valores zerados (linhas 15, 16 e 17). Então, são geradas imagens até atingir o tamanho do lote desejado (linhas 18, 19, 20 e 21). Por fim, a função retorna o lote gerado (linha 22).

Quadro 7 - Código fonte gerador de lotes

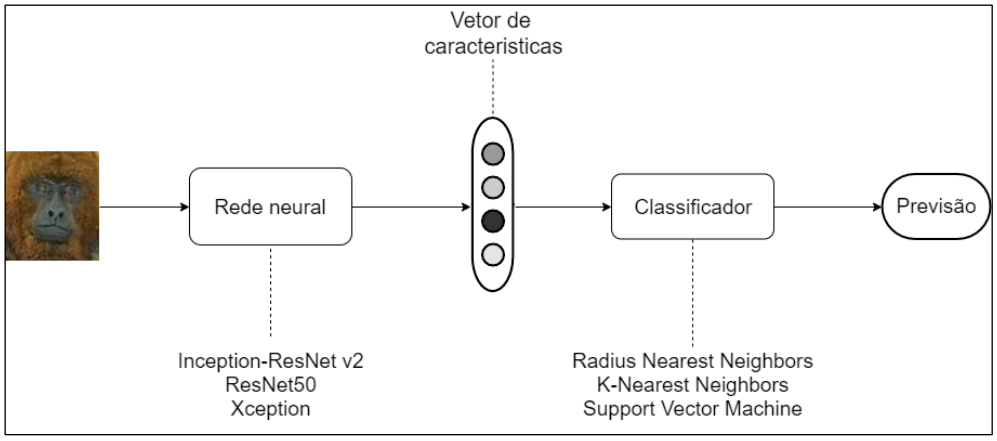
```
1  def get_generator(labelWithImages, input_shape, num_images_per_class=8,
   num_classes_per_batch=8):    return batch_generator()
2      allLabels = list(labelWithImages.keys())
3      def generator():
4          while True:
5              labels = np.random.choice(allLabels, num_classes_per_batch,
   replace=False)
6              for label in labels:
7                  images = labelWithImages.get(label)
8                  for _ in range(num_images_per_class):
9                      choice = np.random.choice(range(len(images)), 1)
10                     filepath = images[choice[0]]
11                     yield load_image(filepath, input_shape), label
12     def batch_generator():
13         while True:
14             g = generator()
15             batch_size = num_classes_per_batch * num_images_per_class
16             batch_x = np.zeros((batch_size,) + (input_shape[0], input_shape[1], 3))
17             batch_y = np.empty([batch_size], dtype="S20")
18             for i in range(batch_size):
19                 generated = next(g)
20                 batch_y[i] = generated[1]
21                 batch_x[i] = generated[0]
22             yield batch_x, batch_y
23     return batch_generator()
```

Fonte: elaborador pelo autor.

3.2.4 Inferência

O processo de inferência foi dividido em duas partes, extração de características e classificação. A extração de características é feita pela rede neural, tendo como saída um vetor de 64 características, enquanto a classificação foi realizada por três algoritmos de Aprendizado de Máquina: k-Nearest Neighbors (k-NN), Support Vector Machine (SVM) e Radius Nearest Neighbors (r-NN). A Figura 15 ilustra o processo de inferência com um classificador r-NN.

Figura 15 – Diagrama de fluxo do processo de inferência



Fonte: elaborado pelo autor.

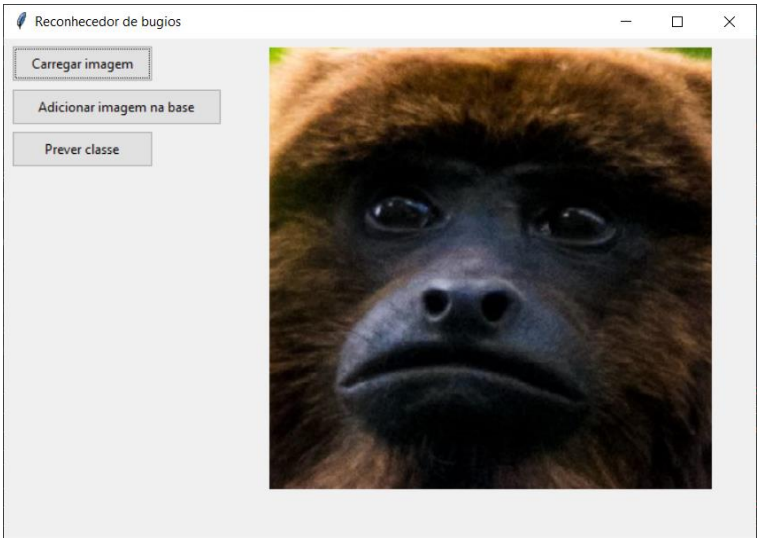
Na Figura 15, a imagem de entrada é redimensionada e enviada para Rede Neural. A Rede Neural, é responsável pela extração de características do bugio que são enviadas para um classificador. O classificador então identifica através

das características com base nas imagens de treinamento. Desta forma, a rede neural consegue extrair as características importantes mesmo de imagens de classes não utilizadas no treinamento, ou seja, quando uma nova classe for adicionada não é necessário retreinar a rede neural, somente a classificação precisa ser atualizada. Como o algoritmo de classificação se baseia em distância, a partir de um único exemplo inserido, foi possível prever novos elementos de uma mesma classe.

3.2.5 Aplicação

Foi desenvolvido uma aplicação desktop para apresentar as funcionalidades do protótipo desenvolvido. Nesta aplicação há três funcionalidades principais: carregar imagem, prever classe e adicionar nova imagem na base. A Figura 16 apresenta a tela da aplicação criada.

Figura 16 - Tela da aplicação com imagem carregada.

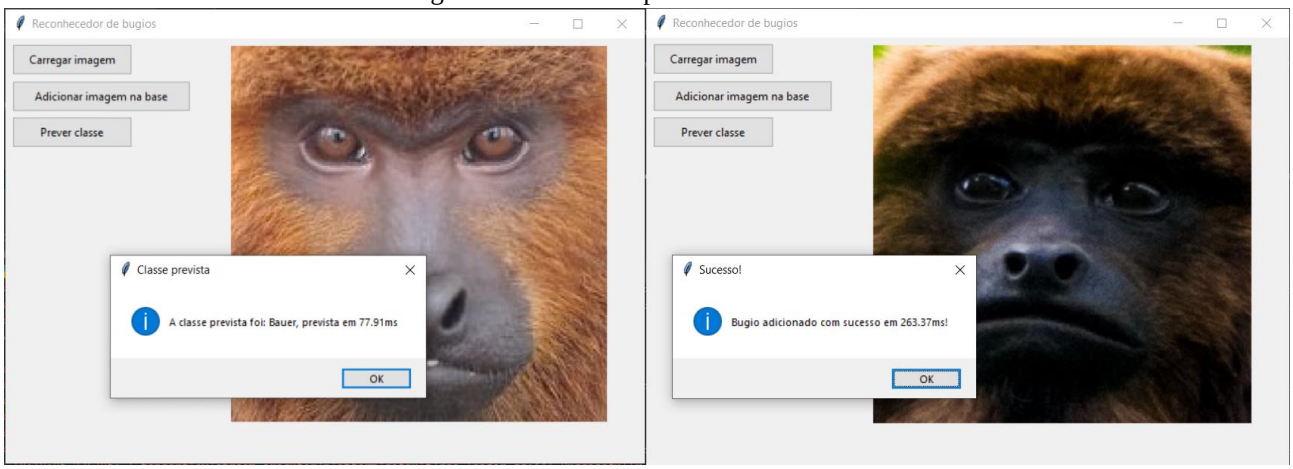


Fonte: elaborado pelo autor.

Ao carregar uma imagem a mesma é exibida ao lado dos botões. Com uma imagem selecionada é possível prever sua classe, ou adicionar ela a base de dados. Ao prever, será exibida uma mensagem de informação com o nome do bugio previsto e o tempo (em milissegundos) que levou para realizar a previsão (à esquerda na Figura 17). Quando utilizado a funcionalidade de adicionar a base de dados, é aberto uma tela pedindo que seja informado o nome do bugio (à direita na Figura 17). Após adicionar uma imagem é exibida uma mensagem de sucesso informando que o bugio foi adicionado e quanto tempo (em milissegundos) levou para adicionar o mesmo. A partir deste momento a aplicação pode reconhecer o novo bugio inserido.

A funcionalidade de adicionar imagem na base permite que o usuário insira inclusive imagens de bugios ainda não vistos. Desta forma caso não exista nenhum exemplar daquele indivíduo, ao adicioná-lo, a aplicação passa a considerar esta nova classe para realizar a previsão. Com isso é possível aprender a reconhecer um novo indivíduo a partir de uma única imagem, ou seja, é possível realizar o One-Shot Learning.

Figura 17 - Telas do aplicativo desenvolvido

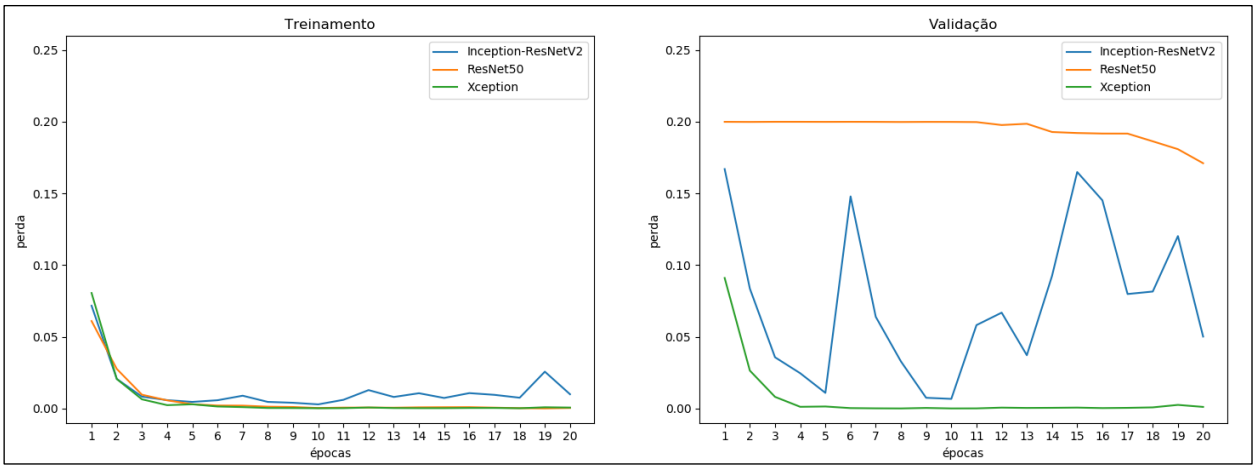


Fonte: elaborado pelo autor.

4 RESULTADOS

Nesta seção são apresentados os resultados obtidos nos testes da rede neural e do classificador desenvolvidos. A Figura 18 apresenta uma comparação entre os resultados das redes neurais Inception-ResNet v2 (SZEGEDY et al., 2017), ResNet50 (HE et al., 2016) e Xception (CHOLLET, 2017), comparando-as pela perda gerada pela função Triplet Loss. A Tabela 1 apresenta os melhores valores para cada rede neural testada. Os classificadores r-NN, k-KNN e SVM também foram testados, comparando-os pela acurácia atingida. Todas as arquiteturas foram treinadas com a mesma configuração, isto é a função de perda Triplet Loss e otimizador Adam com taxa de aprendizagem 0,001, a fim de extrair características faciais.

Figura 18 - Comparação entre as arquiteturas



Fonte: elaborado pelo autor.

A Figura 18 apresenta a perda conforme as épocas prosseguem para cada uma das redes testadas. No gráfico à esquerda é apresentado o valor da perda, gerado pela Triplet Loss de treinamento, enquanto à direita é apresentado o valor da perda de validação. Para comparação foram utilizadas 20 épocas. É possível observar que o valor de perda da rede Inception-ResNet v2 oscilou, principalmente no gráfico de validação, isto pode indicar que a rede está com problema no treinamento.

Tabela 1 - Valores de perda por arquitetura

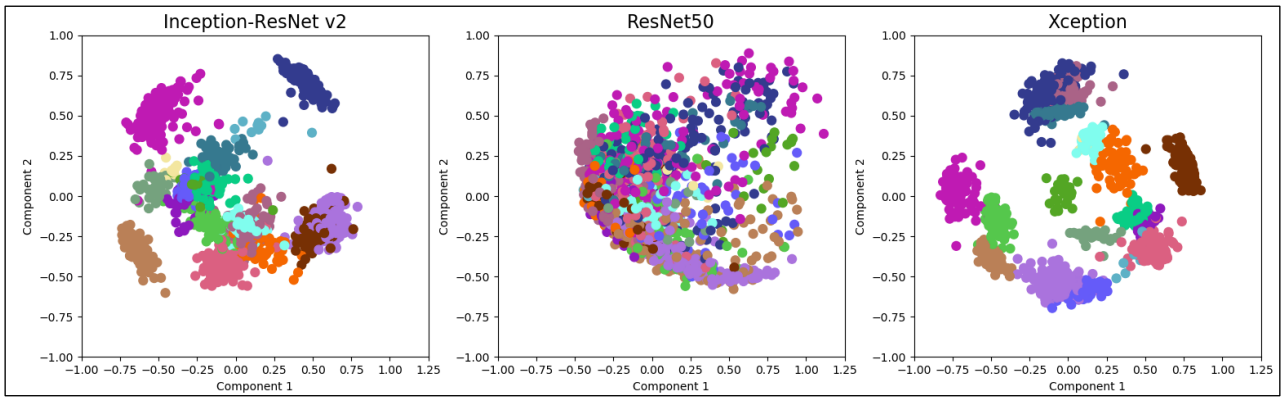
Redes	Treinamento (Menor valor de perda)	Validação (Menor valor de perda)
Inception-ResNet v2	0,002845 na época 10	0,006672 na época 10
ResNet50	**0,000022** na época 19	0,171088 na época 20
Xception	0,000136 na época 10	**0,000003** na época 8

Fonte: elaborador pelo autor.

Na Tabela 1 é possível observar que a rede ResNet50 foi a que atingiu o melhor resultado no treinamento, porém, no erro de validação a rede ResNet50 possui um erro mais alto que todas as outras redes. Este resultado pode indicar *overfitting*, isto é, o modelo se adaptou bem aos dados de treinamento, mas se tornou ineficaz na predição de novos resultados. A rede Xception apresentou uma melhor performance em relação as outras redes com uma perda de 0,000003 na validação, além de ser a menor em quantidade de pesos treinados. A rede Xception não apresentou variações após a época 4, indicando que a mesma pode ser treinada com menos épocas.

Para apresentar melhor a diferença entre as redes testadas foi utilizada a técnica Principal Component Analysis (PCA) a fim de reduzir a dimensionalidade das características geradas pelas redes de 64 dimensões para 2 dimensões de forma que possa ser visualizado num plano de duas dimensões. O resultado é apresentado na Figura 19 onde pontos de mesma cor representam imagens diferentes do mesmo indivíduo. Na figura é possível perceber que a rede ResNet50 não conseguiu deixar próximo imagens de um mesmo indivíduo, ou seja, gerando características difíceis de serem classificadas. A rede Xception mostra um agrupamento melhor que a rede Inception ResNet v2, evidenciando o resultado exibido na Figura 18.

Figura 19 - Visualização 2D das características de rede



Fonte: elaborador pelo autor.

Para avaliar os resultados de classificação, foram testadas cinco diferentes abordagens: três Radius Nearest Neighbor (r-NN) onde foi variado o valor do raio (r), um k-Nearest Neighbor (k-NN) com o valor de k definido empiricamente como 5 e um Support Vector Machine (SVM). Para aplicação dos algoritmos foi usada a implementação da biblioteca Scikit-Learn. Para cada classificação foi realizada a validação cruzada com a técnica k-Fold com k=10 e calculada a média da acurácia para cada algoritmo. O resultado dos testes, assim como a quantidade de pesos treináveis para cada rede utilizada, é apresentado na Tabela 2.

Tabela 2 - Acurácia classificação por arquitetura e algoritmo

Redes	Pesos treináveis	r-NN, r=0.4	r-NN, r=0.5	r-NN, r=0.7	k-NN, k=5	SVM
Inception-ResNet v2	54374560	98,55%	99,39%	99,72%	**99,78%**	99,72%
ResNet50	24714304	53,46%	50,67%	46,76%	70,0%	**75,98%**
Xception	21986664	99,83%	99,89%	99,78%	**99,94%**	99,94%

Fonte: elaborado pelo autor.

Com os resultados é possível observar que a rede ResNet50 obteve uma acurácia bem abaixo das outras redes testadas, tendo seu melhor resultado uma acurácia de 75,98% com o algoritmo SVM. A rede Xception novamente atingiu o melhor resultado, superando as outras redes em todos os algoritmos de classificação, seu melhor resultado foi 99,94% nos algoritmos k-NN e SVM. A rede Inception-ResNet v2 atingiu um resultado próximo ao Xception tendo como seu melhor resultado o algoritmo k-NN com uma acurácia de 99,78%, porém a rede Inception-ResNet v2 possui mais que o dobro de pesos do que a rede Xception, tornado ela mais custosa computacionalmente no processo de treinamento e inferência. A rede Xception ainda obteve melhor resultado em todos os classificadores testados.

5 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um protótipo para reconhecimento o facial de bugios-ruivo utilizando Redes Neurais Convolucionais. O objetivo de classificar um bugio através de reconhecimento foi atingido. Utilizando a rede Xception com o algoritmo k-NN foi possível classificar um bugio com 99,94% de acurácia. Além disso, com a divisão entre a extração de características e a classificação, foi possível aprender a classificar uma nova classe a partir de um único exemplo, treinando apenas o classificador utilizado. A função de perda Triplet Loss desempenhou um papel fundamental para o treinamento da rede neural, pelo fato de comparar a âncora com um exemplo positivo e outro negativo faz com que a rede neural aprenda em menos iterações.

Apesar de conseguir um bom resultado de classificação, este protótipo possui algumas limitações. Atualmente para realizar a classificação é necessário que a imagem utilizada já esteja corretamente recortada contendo apenas a face do bugio, ou seja, precisa de uma ação prévia que é recortar a face da imagem original. Uma possível solução para este problema seria implementar outra rede neural responsável pela detecção da posição da face na imagem, de forma que seja possível extrair a face e enviar para a rede construída neste trabalho.

Como trabalhos futuros, poderia ser adicionado novas funcionalidades para o aplicativo, como classificação de mais imagens ao mesmo tempo, além de permitir inserir informações do bugio, como altura, peso, cumprimento da cauda, entre outras. Seria possível também criar uma aplicação mobile para que os pesquisadores consigam realizar a classificação diretamente na mata, onde estão os bugios. Outra possibilidade seria modelar uma arquitetura de Rede Neural Convolucional específica para a tarefa de reconhecimento facial de bugio, pois as redes utilizadas neste trabalho foram criadas para outros propósitos.

REFERÊNCIAS

AFFONSO, Carlos de Oliveira. **Aplicação de redes neuro fuzzy ao processamento de polímeros na indústria automotiva.** 2010. 157 f. Dissertação (Mestrado em Engenharia) - Universidade Nove de Julho, São Paulo, 2010. Disponível em: < http://bibliotecatede.uninove.br/handle/tede/153>. Acesso em: 22 nov. 2019.

BICCA-MARQUES, Júlio César et al. **Avaliação do Risco de Extinção de Alouatta guariba clamitans**. 2014. Disponível em: <http://www.icmbio.gov.br/portal/faunabrasileira/estado-de-conservacao/7179-mamiferos-alouatta-guariba-clamitans-guariba-ruivo>. Acesso em: 22 nov. 2019.

BOLGER, Douglas T. et al. A computer-assisted system for photographic mark-recapture analysis. **Methods In Ecology And Evolution**, [s.l.], v. 3, n. 5, p.813-822, 29 maio 2012. Wiley. http://dx.doi.org/10.1111/j.2041-210x.2012.00212.x. Disponível em: <https://doi.org/10.1111/j.2041-210X.2012.00212.x>. Acesso em: 22 nov. 2019.

CHOLLET, Francois. Xception: Deep Learning with Depthwise Separable Convolutions. **2017 Ieee Conference On Computer Vision And Pattern Recognition (cvpr)**, [s.l.], p.1251-1258, jul. 2017. IEEE. http://dx.doi.org/10.1109/cvpr.2017.195.

COSTA, Raul et al. **Monitoramento in situ da biodiversidade**: Uma proposta para a composição de um Sistema Brasileiro de Monitoramento da Biodiversidade. 2. ed. Brasília/DF: ICMBio, 2013.

CROUSE, David et al. LemurFaceID: a face recognition system to facilitate individual identification of lemurs. **Bmc Zoology**, [s.l.], v. 2, n. 1, p.1-14, 17 fev. 2017. Springer Nature. Disponível em: < https://bmczool.biomedcentral.com/track/pdf/10.1186/s40850-016-0011-9>. Acesso em: 04 nov.2018. http://dx.doi.org/10.1186/s40850-016-0011-9.

CUSTÓDIO, Caio Amaral. **Redes neurais artificiais e teoria do funcional da densidade:** otimização de funcionais para modelagem de nanomateriais. 2019. 49 f. Dissertação (Mestrado) - Curso de Química, Universidade Estadual Paulista (UNESP), Araraquara, 2019. Disponível em: <http://hdl.handle.net/11449/190855>. Acesso em: 22 nov. 2019.

HAYKIN, Simon. **Redes Neurais**: Princípios e Prática. 2. ed. Porto Alegre, Rs: Bookman, 2003. 898 p.

HE, Kaiming et al. Deep Residual Learning for Image Recognition. **2016 Ieee Conference On Computer Vision And Pattern Recognition (cvpr)**, [s.l.], p.770-778, jun. 2016. IEEE. http://dx.doi.org/10.1109/cvpr.2016.90. Disponível em: <https://doi.org/10.1109/cvpr.2016.90>. Acesso em: 22 nov. 2019.

LECUN, Y. et al. Gradient-based learning applied to document recognition. **Proceedings Of The Ieee**, [s.l.], v. 86, n. 11, p.2278-2324, nov. 1998. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/5.726791.

LOOS, Alexander; ERNST, Andreas. An automated chimpanzee identification system using face detection and recognition. **Eurasip Journal On Image And Video Processing**, [s.l.], v. 2013, n. 1, p.1-17, 19 ago. 2013. Springer Nature. Disponível em: < https://jivp-eurasipjournals.springeropen.com/track/pdf/10.1186/1687-5281-2013-49>. Acesso em: 04 nov.2018. http://dx.doi.org/10.1186/1687-5281-2013-49.

MASI, Iacopo et al. Deep Face Recognition: A Survey. **2018 31st Sibgrapi Conference On Graphics, Patterns And Images (sibgrapi)**, [s.l.], p.1-26, out. 2018. IEEE. http://dx.doi.org/10.1109/sibgrapi.2018.00067.

PANDEY, Atul. **Depth-wise Convolution and Depth-wise Separable Convolution.** 2018. Disponível em: <https://medium.com/@zurister/depth-wise-convolution-and-depth-wise-separable-convolution-37346565d4ec>. Acesso em: 1 dez. 2019.

RUSSAKOVSKY, Olga et al. ImageNet Large Scale Visual Recognition Challenge. **International Journal Of Computer Vision**, [s.l.], v. 115, n. 3, p.211-252, 11 abr. 2015. Springer Science and Business Media LLC. http://dx.doi.org/10.1007/s11263-015-0816-y.

SCHOFIELD, Daniel et al. Chimpanzee face recognition from videos in the wild using deep learning. **Science Advances**, [s.l.], v. 5, n. 9, p.1-9, set. 2019. American Association for the Advancement of Science (AAAS). http://dx.doi.org/10.1126/sciadv.aaw0736. Disponível em: <https://advances.sciencemag.org/content/5/9/eaaw0736>. Acesso em: 22 nov. 2019.

SCHROFF, Florian; KALENICHENKO, Dmitry; PHILBIN, James. FaceNet: A unified embedding for face recognition and clustering. **2015 Ieee Conference On Computer Vision And Pattern Recognition (cvpr)**, [s.l.], p.1-10, jun. 2015. IEEE. http://dx.doi.org/10.1109/cvpr.2015.7298682.

SOUZA JUNIOR, Julio César de. **Perfil sanitário de bugios ruivos, Alouatta guariba clamitans (Cabrera, 1940) (Primates: Atelidae): um estudo com animais recepcionados e mantidos em perímetro urbano no município de Indaial, Santa Catarina - BRASIL.** 2007. 109 f. Dissertação (Mestrado) - Curso de Saúde Pública, Universidade Federal de Santa Catarina, Florianópolis, 2007. Disponível em: <http://repositorio.ufsc.br/xmlui/handle/123456789/90014>. Acesso em: 04 nov. 2018

SZEGEDY, Christian et al. Going deeper with convolutions. **2015 Ieee Conference On Computer Vision And Pattern Recognition (cvpr)**, [s.l.], p.1-9, jun. 2015. IEEE. http://dx.doi.org/10.1109/cvpr.2015.7298594. Disponível em: <https://doi.org/10.1109/cvpr.2015.7298594>. Acesso em: 22 nov. 2019.

SZEGEDY, Christian et al. Inception-v4, inception-resnet and the impact of residual connections on learning. **Thirty-first Aaai Conference On Artificial Intelligence**, San Francisco, p.4278-4284, fev. 2017. Disponível em: <https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14806/14311>. Acesso em: 22 nov. 2019.

VARGAS, Ana Caroline Gomes; PAES, Aline; VASCONCELOS, Cristina Nader. Um estudo sobre redes neurais convolucionais e sua aplicação em detecção de pedestres. **Proceedings Of The Xxix Conference On Graphics, Patterns And Images**, São José dos Campos, out. 2016.

VINYALS, Oriol et al. Matching networks for one shot learning. **Advances In Neural Information Processing Systems**, [s.l.], p.3630-3638, 2016.