DISCIPLINARUM SCIENTIA *Disciplinarum Scientia.* **Série: Naturais e Tecnológicas**
Santa Maria, v. 25, n. 3, p. 69-85, 2024
Recebido em: 20.12.2024. Aprovado em: 20.12.2024.
ISSN 2176-462X | DOI: doi.org/10.37779/nt.v25i3.5211

DESENVOLVIMENTO DE UMA API COM TECNOLOGIA DE RECONHECIMENTO FACIAL PARA LOCALIZAÇÃO DE ANIMAIS

DEVELOPMENT OF AN API WITH FACIAL RECOGNITION TECHNOLOGY FOR ANIMAL LOCATION

Douglas Iracet dos Santos[1], Alexandre de Oliveira Zamberlan[2], Fabrício Tonetto Londero[3] e Ana Paula Canal[4].

RESUMO

Este trabalho descreve o desenvolvimento de uma API baseada em tecnologias de reconhecimento facial para identificação de animais. A API permite aos usuários comparar imagens de seus animais desaparecidos com fotos de animais semelhantes na base de dados. O projeto explora as vantagens e desafios do reconhecimento facial na localização de animais perdidos, focando na eficácia dessa tecnologia. Foi desenvolvido utilizando a metodologia FDD(*Feature Driven Development*), implementado em Python, a biblioteca OpenCV e os algoritmos YOLOv3 e Random Forest. O principal resultado A API gera uma lista de animais semelhantes com base na imagem enviada pelo usuário.

Palavras-chave: OpenCV; Random Forest; Python; YOLOv3.

ABSTRACT

This work describes the development of an API based on facial recognition technologies for animal identification. The API allows users to compare images of their missing pets with photos of similar animals in the database. The project explores the advantages and challenges of facial recognition in locating lost animals, focusing on the effectiveness of this technology. It was developed using the FDD (Feature Driven Development) methodology, implemented in Python, the OpenCV library, and the YOLOv3 and Random Forest algorithms. The main result is that the API generates a list of similar animals based on the image submitted by the user.

Keywords: *OpenCV; Random Forest; Python; YOLOv3.*

1 Autor. Universidade Franciscana (UFN). E-mail: douglas.iracet@ufn.edu.br. ORCID: https://orcid.org/0009-0007-6105-997X

2 Colaborador. Docente nos cursos de Ciência da Computação, Jogos Digitais e Sistemas de Informação na Universidade Franciscana (UFN). E-mail: alexz@ufn.edu.br. ORCID: https://orcid.org/0000-0002-9815-2031

3 Colaborador. Docente nos cursos de Ciência da Computação, Jogos Digitais e Sistemas de Informação na Universidade Franciscana (UFN). E-mail: fabricio.londero@ufn.edu.br. ORCID: https://orcid.org/0000-0002-4822-4981

4 Orientadora. Docente nos cursos de Ciência da Computação, Jogos Digitais e Sistemas de Informação na Universidade Franciscana (UFN). E-mail: apc@ufn.edu.br. ORCID: https://orcid.org/0000-0001-6360-1688

INTRODUÇÃO

Animais de estimação são importantes para muitas pessoas, oferecendo amor e companhia. A perda de um animal pode ser angustiante, tornando a busca urgente e exigindo recursos e ajuda comunitária. A tecnologia tem fornecido soluções para vários desafios, e o reconhecimento facial, já usado em segurança e medicina, agora ajuda a localizar animais perdidos. Este trabalho descreve o desenvolvimento de uma API de reconhecimento facial para ajudar donos a encontrar seus animais desaparecidos, permitindo que usuários façam o *upload* de imagens de seus animais para comparação com fotos de animais encontrados na base de dados.

Antes de os cães se tornarem os mais leais companheiros, os seres humanos enxergavam outros animais principalmente como fontes de alimento. A domesticação de animais, iniciada há 12 mil anos, visava a conveniência de ter uma fonte de alimento próxima. No entanto, a relação com os animais evoluiu significativamente. Conviver com animais de estimação traz benefícios diversos. Crianças que crescem com animais desenvolvem maior afetividade e senso de responsabilidade (TATIBANA e COSTA-VAL, 2009). Para os idosos, os animais oferecem conforto, aumentam a autoestima e promovem a interação social (Costa, 2006). Para muitos donos, os animais são vistos como membros da família (Walsh, 2009). Quando um animal desaparece, a busca torna-se uma prioridade urgente, utilizando métodos tradicionais e tecnologia avançada. Este projeto teve como objetivo implementar uma *Application Programming Interface* (API) de reconhecimento facial para a identificação de animais através de imagens faciais compartilhadas por usuários, baseado nos algoritmos YOLOV3, *Random Forest*.

1. REFERENCIAL TEÓRICO

O Referencial teórico do trabalho traz conceitos do reconhecimento facial, as principais tecnologias usadas para a implementação e trabalhos correlatos.

Existem dois tipos de abordagens fundamentais para o reconhecimento facial (LI e JAIN, 2011). A primeira consiste na extração de vetores de características de partes individuais do rosto, como olhos, nariz, boca e queixo, utilizando modelos deformáveis e análise matemática para isso. A segunda abordagem baseia-se na teoria da informação, derivando informações da imagem facial completa e caracterizando áreas importantes através de modelos de curvas e funções de energia (LI e JAIN, 2011; ZHANG e RUAN, 2006).

Existem diferentes tipos de abordagens, como, por exemplo, em GHOSAL, TIKMANI e GUPTA (2009), a extração de características de imagens faciais é realizada utilizando a transformada de Wavelet Gabor, sendo que o algoritmo *Random Forest* é empregado como classificador dessas características. A transformada de Wavelet Gabor gera um grande número de características, muitas das quais são redundantes. Para lidar com esse problema, a técnica do Random Forest é utilizada para

identificar as características relevantes, reduzindo o espaço de características e acelerando o processo de classificação facial. Kshirsagar, Baviskar e Gaikwad(2011) utilizaram métodos de Análise dos Componentes Principais (PCA) e técnicas estatísticas. Stan e Anil(2011) implementaram um sistema de reconhecimento facial que empregou técnicas como *Eigenfaces*, PCA e redes neurais.

O algoritmo PCA foi utilizado para extrair características representativas que identificam de maneira única uma imagem facial, servindo como entradas para a rede neural classificadora de faces. O estudo demonstrou que os Eigenfaces podem fornecer características significativas, reduzindo o tamanho das entradas da rede neural e aumentando a eficiência do processo de reconhecimento. No entanto, essa abordagem é sensível a condições de iluminação não controlada (DINIZ *et al.*, 2013).

1.1 FERRAMENTAS E TECNOLOGIAS

Para a implementação, foi utilizada a linguagem Python, que é uma linguagem de programação de alto nível (SETTE e CARVALHO, 2021). O OpenCV é uma biblioteca de código aberto criada pela Intel em 2000, projetada para aplicações de visão computacional em tempo real. Portável, sendo amplamente utilizada em ambientes acadêmicos e comerciais (OPENCV, 2012). O Django é um *framework* web em Python projetado para desenvolvimento rápido e eficiente de aplicativos web escaláveis e robustos (Django Documentation, 2024).

O SQLite é um banco de dados SQL embutido que não necessita de configuração de servidor, sendo ideal para aplicações de *desktop* e web de pequena escala (SQLite Documentation, 2024). O Django utiliza migrações para garantir a consistência entre o modelo de dados definido em classes Python e a estrutura do banco de dados.

Uma API (*Application Programming Interface*) é um conjunto de definições e protocolos que permite a comunicação entre diferentes sistemas de software. Ela define como os desenvolvedores podem solicitar serviços de um sistema operacional, biblioteca ou serviço web, facilitando a integração entre aplicações. APIs podem ser públicas ou privadas e são amplamente usadas para conectar serviços, como pagamento online, redes sociais ou dados de mapas. Exemplos de APIs incluem as APIs web que utilizam HTTP/HTTPS e as especificações como OpenAPI para criar APIs RESTful (RED HAT, 2024).

1.2 MODELOS UTILIZADOS

O método de classificação Random Forest, proposto por Breiman (2001), é uma técnica de agregação de classificadores do tipo árvore, onde cada árvore é construída de forma aleatória (Ghosal, Tikmani e Gupta, 2009). Para determinar a classe de uma instância, o método combina os resultados de várias árvores de decisão por meio de um mecanismo de votação. Cada árvore contribui com uma

classificação ou um voto para uma classe específica. A classificação final é determinada pela classe que recebe o maior número de votos entre todas as árvores (BREIMAN, 2001). YOLOv3 (*You Only Look Once*, versão 3) é um avançado modelo de detecção de objetos em tempo real desenvolvido por Joseph Redmon e Ali Farhadi (REDMON e FARHADI, 2018).

Utiliza a rede neural convolucional Darknet-53, composta por 53 camadas, para detecções em três escalas diferentes, integrando características de vários níveis para maior precisão. Com o uso de caixas ancoradas, YOLOv3 prevê deslocamentos para otimizar a detecção de objetos de diferentes tamanhos. Sua função de custo abrange perdas de localização, confiança e classificação, fundamentais para o refinamento das previsões.

YOLOv3 é notável por seu equilíbrio entre precisão e velocidade, operando até 45 FPS em GPUs modernas, sendo ideal para aplicações que exigem processamento em tempo real, como vigilância e condução autônoma (REDMON e FARHADI, 2018). Para implementar o YOLOv3 em qualquer linguagem de programação três arquivos principais são necessários: yolov3.weights: Contém os pesos pré-treinados do modelo. yolov3.cfg: Arquivo de configuração que define a arquitetura da rede neural. coco.names: Arquivo de texto que lista os nomes das classes de objetos que o modelo pode detectar.

1.3 TRABALHOS CORRELATOS

Os trabalhos correlatos exploram o reconhecimento facial aplicado à localização de animais perdidos. A Pupz é uma plataforma pet brasileira que desenvolveu um aplicativo gratuito de reconhecimento facial para cães e gatos, para ajudar a encontrar animais perdidos. (EQUIPE CÃESGATOS, 2021).

A startup chinesa Megvii também utiliza reconhecimento facial para identificar cachorros perdidos, focando nas singularidades do focinho dos cães. (WAGNER WAKKA, 2019).

A Petvation desenvolveu uma porta inteligente para animais de estimação que utiliza reconhecimento facial para garantir a segurança dos pets. Os tutores podem controlar os horários de entrada e saída do pet, eliminando a necessidade de verificar se a porta está trancada (NUBIA DA CRUZ, 2022).

Um sistema de monitoramento de gatos domésticos foi desenvolvido utilizando aprendizagem de máquina para reconhecimento individual. As tecnologias usadas incluem YOLOv8, BiT e OpenCV, com um enfoque distintivo na região dos olhos para o reconhecimento facial (GABRIEL WHITACKER GEROTTI, 2023).

Os estudos mencionados exploram o reconhecimento facial para a localização e identificação de animais. Utilizam abordagens variadas, desde algoritmos próprios focados em características biométricas até tecnologias como YOLO, BiT e OpenCV. Todos têm como objetivo promover a segurança e o bem-estar animal, facilitando a reintegração com seus tutores e processos de adoção.

2. METODOLOGIA

O *Feature-Driven Development* (FDD) é uma metodologia ágil de desenvolvimento de software, composta por cinco processos distintos:

No primeiro processo, Desenvolvimento de um Modelo Abrangente, realiza-se um estudo detalhado do domínio de negócios e define-se o escopo do projeto. Em seguida, na Construção de uma Lista de Funcionalidades, todas as funcionalidades necessárias são identificadas, priorizando-as durante o Planejamento por Meio de Funcionalidades, incluindo a avaliação de sua viabilidade funcional.

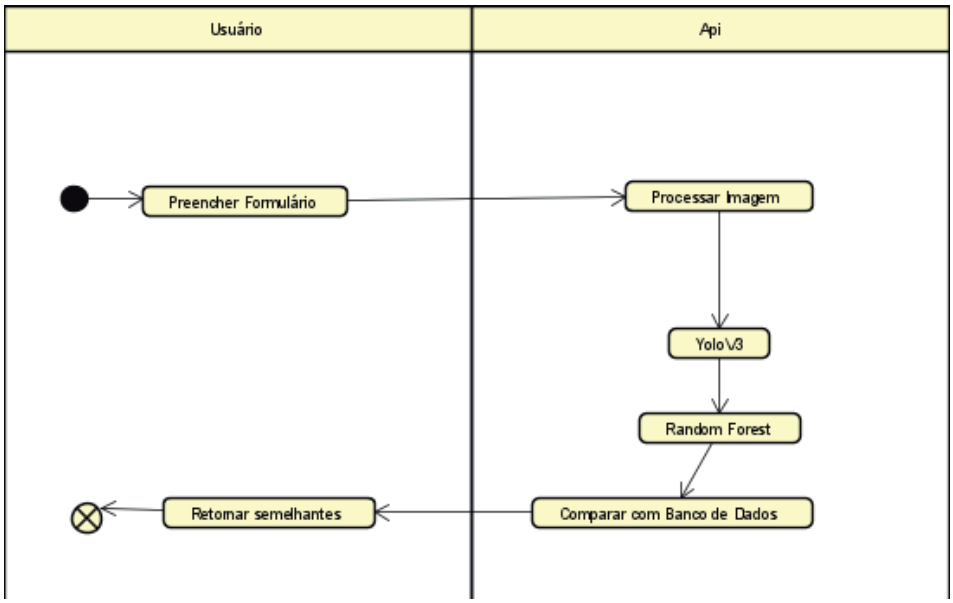
No processo de Projeto por Meio de Funcionalidades, cada funcionalidade é detalhada com atividades específicas, como o esboço do modelo de interface do usuário e a criação de diagramas de sequência e classe. Na etapa de Construção por Meio de Funcionalidades, o código é gerado iterativamente, entregando a cada ciclo uma funcionalidade que agrega valor ao cliente (SILVA *Et al.*, 2011).

2.1. DESENVOLVIMENTO DO MODELO ABRANGENTE

O Diagrama de Domínio é produzido, proporcionando uma visão panorâmica da aplicação ao apresentar suas principais entidades e associações.

A Figura 1 fornece uma visão do Diagrama de Atividade principal. O usuário irá utilizar a API para mandar os dados exigidos no formulário e fazer o *upload* da imagem desejada e a API irá retornar uma lista dos animais semelhantes ao da imagem enviada caso possua. Após esta fase, a aplicação verifica o usuário, possibilitando o cadastro ou busca de um animal perdido, e notifica caso o animal seja encontrado.

Figura 1 - Diagrama de Atividades.



Fonte: Construção do Autor.

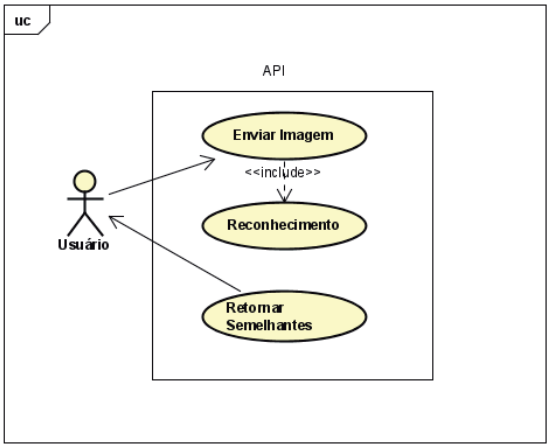
2.2. LISTA DE FUNCIONALIDADES E PLANEJAMENTO POR FUNCIONALIDADE

Foram identificados os requisitos funcionais e não funcionais da API. Os requisitos funcionais são:

- RF01: A API deve aceitar o registro de animais perdidos, permitindo informações detalhadas, como fotos e características. Complexidade: Média. Estimativa: 8 horas.
- RF02: A API deve realizar reconhecimento facial para auxiliar na identificação de animais perdidos e correspondência com registros de animais. Complexidade: Alta. Estimativa: 10 horas.
- RF03: A API deve permitir atualizações de dados de animais perdidos, como a adição de novas imagens ou informações. Complexidade: Média. Estimativa: 25 horas.
- Os requisitos não funcionais, que desempenham um papel crucial na manutenção da proposta de desenvolvimento da aplicação, foram os seguintes:
- RNF0**1**: A API deve ser projetada, implementando algoritmos eficientes como YOLOv3 e Random Forest para garantir a precisão e rapidez nas análises.
- RNF02: A documentação e os *endpoints* da API devem ser intuitivos e bem descritos para facilitar o uso por parte dos desenvolvedores.

Procede-se à elaboração do Diagrama de Caso de Uso. A Figura 2 representa a interação entre os atores e suas relações e delineando as funcionalidades.

Figura 2 - Diagrama de caso de uso.

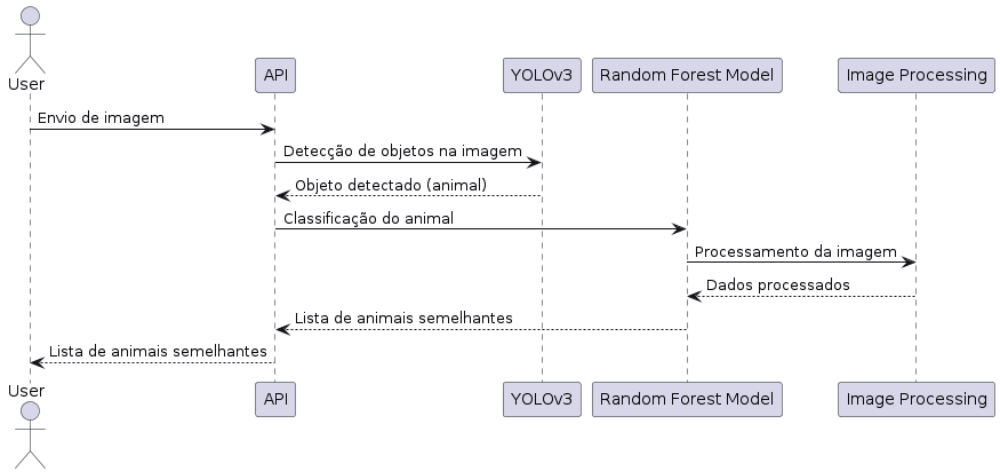


Fonte: Construção do Autor.

3. RESULTADOS E DISCUSSÕES

Esta seção aborda a implementação da API com técnicas de reconhecimento facial para localização de animais perdidos. Primeiramente é colocado o Diagrama de sequência (figura 3) onde destaca-se o resultado principal do trabalho utilizando modelo Random Forest com a biblioteca scikit-learn em Python as demais fases da construção da API são apresentadas nas próximas seções.

Figura 3 - Diagrama de sequência.



Fonte: Construção do Autor.

3.1 TREINAMENTO DO RANDOM FOREST

Para o treinamento do modelo Random Forest, foram utilizadas as bibliotecas scikitlearn, uma ferramenta utilizada para aprendizado de máquina em Python. As principais bibliotecas importadas foram:import os que é responsável pela Interação com o sistema operacional. import cv2 Processamento de imagens e visão computacional import numpy as np Operações matemáticas e arrays. from sklearn.model_selection import train_test_split Divisão de dados. from sklearn.ensemble import RandomForestClassifier Modelo de floresta aleatória, from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score Métricas de avaliação, import seaborn as sns Gráficos estatísticos, import matplotlib.pyplot as plt Criação de gráficos e import joblib Salvar e carregar modelos.

3.2 PREPARAÇÃO DE DADOS

Os dados foram preparados através das seguintes etapas: Coleta de Dados: Os dados foram coletados de várias fontes, contendo imagens de diferentes raças de cães e gatos; Pré-processamento: As imagens foram convertidas para arrays NumPy e rotuladas de acordo com suas classes (cães e gatos). Divisão dos Dados: Os dados foram divididos em conjuntos de treinamento e teste utilizando a função train_test_split da biblioteca scikit-learn.

3.3. CÓDIGO PARA CARREGAR E TREINAR O MODELO

Na figura 4 função para carregar imagens de um diretório específico, redimensioná-las e rotulá-las com base nas pastas onde estão armazenadas. A função carregar_imagens_yolo percorre recursivamente os subdiretórios do diretório principal, verificando arquivos com extensões válidas

como .jpg ou .png. Utilizando a biblioteca OpenCV, as imagens são carregadas e redimensionadas para um tamanho predefinido. Cada imagem redimensionada é convertida em um vetor unidimensional e adicionada a uma lista de imagens, enquanto seus rótulos, obtidos dos nomes dos subdiretórios pais, são armazenados em uma lista separada. Após o carregamento, os dados são divididos em conjuntos de treinamento e teste usando a função train_test_split do scikit-learn, permitindo a avaliação do modelo de aprendizado de máquina que será treinado posteriormente. O código também exibe o número de imagens em cada conjunto, proporcionando uma visão geral da distribuição dos dados. Após carregar as imagens e trazer a quantidade é feito o treinamento do modelo Random Forest com a classificação da raça ou tipo do animal e salvando em um arquivo .pk1.

Figura 4 - Código da Função Para Carregar Imagens.

```
def carregar_imagens():
    imagens = []
    labels = []

    # Percorre recursivamente os subdiret rios
    for root, dirs, files in os.walk(IMAGES_DIR):
        for filename in files:
            # Verifica se o arquivo    uma imagem
            if filename.endswith(".jpg") or
    filename.endswith(".png"):
                # Carrega a imagem
                image_path = os.path.join(root,
    filename)
                image = cv2.imread(image_path)
                if image is not None:
                    # Redimensiona a imagem para o
     tamanho desejado
                    resized_image = cv2.resize(
    image, NEW_SIZE)
                    # Adiciona a imagem e a classe
     ao conjunto de dados
                    imagens.append(resized_image.
    flatten())  # Transforma a imagem em um vetor
    unidimensional
                    label = os.path.basename(os.
    path.dirname(image_path))
                    labels.append(label)

    return imagens, labels

# Carregar imagens e labels
imagens, labels = carregar_imagens()

# Dividir os dados em conjuntos de treinamento e
    teste
X_train, X_test, y_train, y_test =
    train_test_split(imagens, labels, test_size
    =0.2, random_state=42)

# Exibir o n mero de imagens em cada conjunto
print("N mero de imagens no conjunto de
    treinamento:", len(X_train))
print("N mero de imagens no conjunto de teste:",
    len(X_test))
```

```
# Criar e treinar o modelo RandomForest
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)

# Salvar o modelo treinado
MODELOS_DIR = "C:/Users/dougl/myapi/modelos"
if not os.path.exists(MODELOS_DIR):
    os.makedirs(MODELOS_DIR)
joblib.dump(rf_classifier, os.path.join(
    MODELOS_DIR, "modelo_random_forest.pkl"))

# Prever as classes no conjunto de teste

y_pred = rf_classifier.predict(X_test)
```

Fonte: Construção do Autor.

3.4. AVALIAÇÃO DO MODELO

A performance do modelo foi avaliada usando métricas como precisão, revocação,recall e F1-score. Na figura 5 mostra como foi feita essas avaliações.

Figura 5 - avaliação do modelo.

```
# Calcular a precis o para cada classe
precisao_por_classe = precision_score(y_test,
    y_pred, average=None)

# Calcular a revoca  o para cada classe
revocacao_por_classe = recall_score(y_test, y_pred
    , average=None)

# Calcular o F1-Score para cada classe
f1_por_classe = f1_score(y_test, y_pred, average=
    None)
```

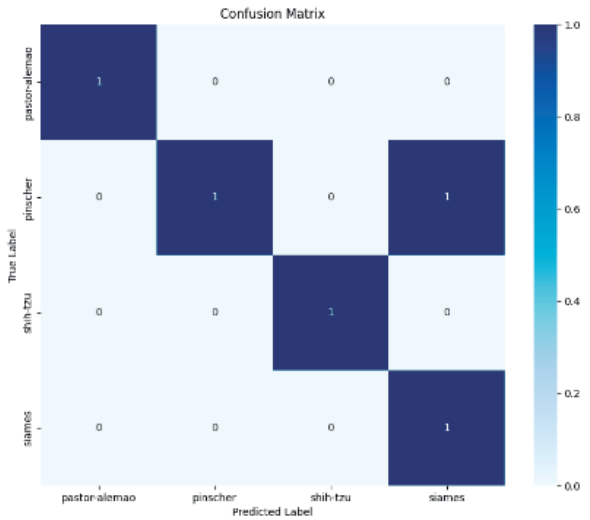
Fonte: Construção do Autor.

Como resultados da avaliação o modelo apresentou uma precisão de 100% para a classe “pastoralemao”. A Revocação foi de 100%, mostrando que o modelo identificou corretamente todas as instâncias reais de “pastor-alemao”, sem falsos negativos. O F1-Score de 100% reflete um equilíbrio entre a revocação e a precisão, mostrando que o modelo é eficaz em encontrar todas as instâncias de “pastor-alemao.

A quantidade de fotos utilizadas para o treinamento, totalizou 24 imagens. Esses dados foram divididos em 6 conjuntos. Cada conjunto inclui tanto o tipo de animal, indicando se é um cachorro ou um gato, quanto a raça desses animais, como um pastor alemão ou um gato siamês.Na Figura 6 trazem uma matriz de confusão, que é uma ferramenta usada para avaliar a performance do modelo Random Forest. Permite visualizar a relação entre as classes reais e previstas para o modelo, ajudando a identificar onde o modelo está acertando e onde está errando. As linhas representam as classes reais e as colunas representam as classes previstas.

Os elementos na diagonal principal (da célula superior esquerda para a célula inferior direita) representam o número de acertos para cada classe. Essas são as instâncias que foram corretamente classificadas. Os elementos fora da diagonal representam erros de classificação. Eles mostram quantas instâncias de uma classe foram incorretamente classificadas como outra classe.

Figura 6 - Matriz de Confusão.



Fonte: Construção do Autor.

3.6 IMPLEMENTAÇÃO DA DETECÇÃO DE OBJETOS COM YOLOV3

O ambiente foi configurado para utilizar o YOLOv3 com a biblioteca OpenCV no arquivo chamado detecao_objetos_yolo.py. Os arquivos de configuração (yolov3.cfg), pesos do modelo (yolov3.weights) e nomes das classes (coco.names) foram obtidos do repositório oficial do YOLO. A Figura 7 traz a função detecar_objetos aonde carrega a imagem que o usuario envia na API e o algoritmo identifica se na imagem possui o animal e retorna o resultado.

Figura 7 - Código da Função detecao_objetos_yolo.py.

```
def detectar_objetos(imagem):
    # Carrega os arquivos de configura o e
    pesos do YOLO
    net = cv2.dnn.readNet(os.path.join(YOLO_DIR, "
    yolov3.cfg"), os.path.join(YOLO_DIR, "yolov3.
    weights"))
    classes = []
    with open(os.path.join(YOLO_DIR, "coco.names")
    , "r") as f:
        classes = [line.strip() for line in f.
    readlines()]

    # Convertendo o objeto InMemoryUploadedFile
    para uma imagem OpenCV
    imagem_opencv = processar_imagem(imagem)

    # Detecta objetos na imagem
    blob = cv2.dnn.blobFromImage(imagem_opencv,
    0.00392, (416, 416), (0, 0, 0), True, crop=
    False)
    net.setInput(blob)
    outs = net.forward(get_output_layers(net))

    # Processa as detec  es e retorna os
    resultados
    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                # Objeto detectado com confian a
                center_x = int(detection[0] *
    imagem_opencv.shape[1])
                center_y = int(detection[1] *
    imagem_opencv.shape[0])
                w = int(detection[2] *
    imagem_opencv.shape[1])
                h = int(detection[3] *
    imagem_opencv.shape[0])
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(
    confidence))
                class_ids.append(class_id)

    # Aplica supress o n o-m xima para evi
    detec  es m ltiplas do mesmo objeto
    indexes = cv2.dnn.NMSBoxes(boxes, confide
     0.5, 0.4)

    # Prepara os resultados
    resultados = {
        'objetos_detectados': [],
        'scores': [],
        'descricao_objetos': []
    }
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]]
            resultados['objetos_detectados'].
    append(label)
            resultados['scores'].append(
    confidences[i])
```

Fonte: Construção do Autor.

Na Figura 9 é feita a classificação do resultado da função detecar_objetos e utiliza o resultado do modelo Rand Forest que foi salvo um arquivo .pk1 e faz a classificação do animal que está na foto e retorna o resultado.

3.5. BANCO DE DADOS UTILIZADO

O banco de dados utilizado neste projeto com o framework Django, que emprega SQLite como seu banco de dados padrão para projetos de pequeno e médio porte. O modelo de dados foi definido utilizando classes do Django ORM *(Object-Relational Mapping*), que mapeia as classes Python para tabelas no banco de dados. Para este projeto específico, foi criado um modelo simples para representar animais perdidos. Esse modelo inclui campos como nome, tipo, raça, descrição dos objetos

detectados na imagem e a própria imagem do animal. Na figura 8 traz o código onde é implementada uma model com campos que vão preencher a tabela do banco de dados.

Figura 8 - Código para a Criação da tabela do banco.

```
from django.db import models
from django.utils import timezone

class AnimalPerdido(models.Model):
    nome = models.CharField(max_length=100)
    tipo = models.CharField(max_length=100)
    raca = models.CharField(max_length=100)
    descricao_objetos = models.CharField(
    max_length=100, default='Descri  o n o
    fornecida')
    foto = models.ImageField(upload_to='animais/')
    data_criacao = models.DateTimeField(default=
    timezone.now)

    endereco = models.CharField(max_length=100,
    default='')

    def __str__(self):
        return self.nome
```

Fonte: Construção do Autor.

Para viabilizar a busca e filtragem de animais perdidos, foi utilizado os recursos do Django ORM. Foi implementada uma função na API que emprega consultas do tipo *filter* e *exclude* para encontrar animais perdidos com características semelhantes às fornecidas pelo usuário. Durante o desenvolvimento do projeto, foi executado migrações para criar as tabelas necessárias e inserimos dados manualmente ou através da aplicação para fins de teste e demonstração.

Figura 9 - Código do *Random Forest.*

```
# Classifica os animais detectados na imagem com o
    modelo Random Forest
    for i, objeto in enumerate(resultados['
    objetos_detectados']):
        if objeto == 'dog' or objeto == 'cat':
            # Regi o de interesse para
    classifica o
            x, y, w, h = boxes[i]
            roi = imagem_opencv[y:y+h, x:x+w]
            # Classifica o animal na regi o de
    interesse
            print("Classificando animal...")
            classificacao = classificar_animal(roi
    )
            print("Resultado da classifica o:",
     classificacao)
            resultados['objetos_detectados'][i] =
    classificacao['objetos_detectados'][0]
            resultados['scores'][i] =
    classificacao['scores'][0]

    return resultados
```

Fonte: Construção do Autor.

3.7. IMPLEMENTAÇÃO DA API

A API foi desenvolvida utilizando Django e Django REST Framework, com endpoints para enviar informações de animais perdidos e limpar a base de dados. A função detectar_objetos do arquivo detecao_objetos_yolo.py foi integrada para processar imagens e identificar objetos como cães e gatos. No arquivo views.py mostrado na figura 10, o método enviar _animal _perdido

solicita dados como nome, tipo, endereço e foto do animal, que são salvos no banco de dados. Após o salvamento, o sistema verifica a existência de animais semelhantes e retorna uma lista com raça e tipo para auxiliar na busca especialmente se a precisão não for de 100%, com uma resposta em HTML exibindo essa lista.

Figura 10 - Código da função enviar animal perdidos.

```
from django.shortcuts import render, redirect
from django.http import HttpResponse
from .detecao_objetos_yolo import detectar_objetos  # Importe a fun  o detectar_objetos
from .models import AnimalPerdido
from .forms import AnimalPerdidoForm
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from django.db.models import Q

def enviar_animal_perdido(request):
    if request.method == 'POST':
        form = AnimalPerdidoForm(request.POST, request.FILES)
        if form.is_valid():
            nome = form.cleaned_data['nome']
            tipo = form.cleaned_data['tipo']
            foto = form.cleaned_data['foto']

            # Chama a fun  o para detectar objetos na imagem
            resultados = detectar_objetos(foto)

            # Salva o novo animal perdido no banco de dados
            novo_animal = AnimalPerdido(nome=nome, raca=resultados['objetos_detectados'][0], tipo=tipo, foto=foto)
            novo_animal.save()

            # Verifica se h  animais detectados na imagem e se h  animais perdidos semelhantes
            if 'objetos_detectados' in resultados and resultados['objetos_detectados']:
                animais_perdidos_semelhantes = []
                for animal_detectado in resultados['objetos_detectados']:
                    # Busca animais perdidos semelhantes com base no tipo de animal detectado
                    animais_encontrados = AnimalPerdido.objects.filter(raca=animal_detectado, tipo=tipo)
                    animais_perdidos_semelhantes.extend(animais_encontrados)

                # Remove o novo animal da lista de animais semelhantes, se estiver presente
                animais_perdidos_semelhantes = [animal for animal in animais_perdidos_semelhantes if animal != novo_animal]

                # Retorna para a p gina de sucesso com a lista de animais semelhantes
                return render(request, 'envio_sucesso.html', {'animais_perdidos_semelhantes': animais_perdidos_semelhantes})
            # Retorna para a p gina de sucesso sem a lista de animais semelhantes
            return render(request, 'envio_sucesso.html', {'novo_animal': novo_animal})
    else:
        form = AnimalPerdidoForm()
    return render(request, 'enviar_animal_perdido.html', {'form': form})
```

Fonte: Construção do Autor.

3.8. *ENDPOINT* DA API

A API foi configurada para receber requisições POST com informações sobre animais perdidos, incluindo uma imagem. A imagem é processada pelo YOLOv3 para identificar objetos, e os resultados são utilizados para buscar animais semelhantes na base de dados. O endpoint foi configurado no urls.py como mostra na figura 11 da seguinte forma.

Figura 11 - Código do urls.py.

```python
from django.urls import path
from .views import EnviarAnimalPerdidoAPIView

urlpatterns = [
    path('enviar-animal-perdido-api/',
    EnviarAnimalPerdidoAPIView.as_view(), name='
    enviar_animal_perdido_api'),
]
```

Fonte: Construção do Autor.

3.9 TESTE DA API

Na Figura 12, um script test_api.py simula a interação de um usuário com a API. O endpoint enviar-animal-perdido/ é definido para receber dados como nome, tipo, raça e endereço de um animal perdido, além de um arquivo de imagem. Utilizando a biblioteca requests, os dados são estruturados em um dicionário e enviados via requisição POST para a API como um formulário multipart. Após a execução, a resposta da API é exibida no console, fornecendo feedback sobre o sucesso da operação de envio do animal perdido.

Figura 12 - Código d**o** test_api.py.

```python
import requests

# URL da sua API
url = 'http://localhost:8000/enviar-animal-perdido
    /'

# Solicitar ao usu rio os dados do animal perdido
nome = input("Digite o nome do animal: ")
tipo = input("Digite o tipo do animal: ")
raca = input("Digite o raca do animal: ")
endereco = input("Digite o endereco do animal: ")

# Caminho do arquivo de imagem
caminho_imagem = 'C:/Users/dougl/myapi/media/
    animais/cachorro/pastor-alemao/pastor.jpg'

# Dados do animal perdido
data = {
    'nome': nome,
    'tipo': tipo,
    'raca': raca,
    'endereco': endereco,
}

# Fazendo uma solicita  o POST    sua API para
    enviar um animal perdido
with open(caminho_imagem, 'rb') as arquivo_imagem:
    response = requests.post(url, data=data, files
    ={'foto': arquivo_imagem})

# Exibindo a resposta da API
print('Resposta da API:', response.text)
```

Fonte: Construção do Autor.

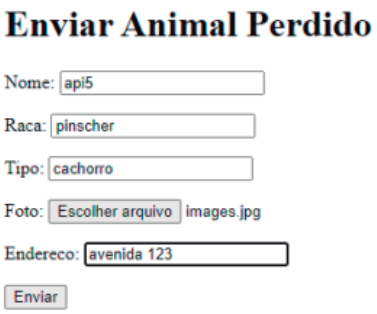
Como resultado do arquivo detecao_objetos_yolo.py foi obtido um retângulo na imagem da foto testada com a classificação do animal traz uma página web com o endpoint criado na API com uma lista inicial de animais que já estão salvos no banco de dados , após como na Figura 13.

Figura 13 - Resultado da foto enviada.

Fonte: Construção do Autor.

Na Figura 14 traz um endpoint com um formulário para enviar a foto do animal que foi perdido ou achado junto com os dados nome, raça, tipo e endereço.

Figura 14 - *endpoint* para enviar animal.



Fonte: Construção do Autor.

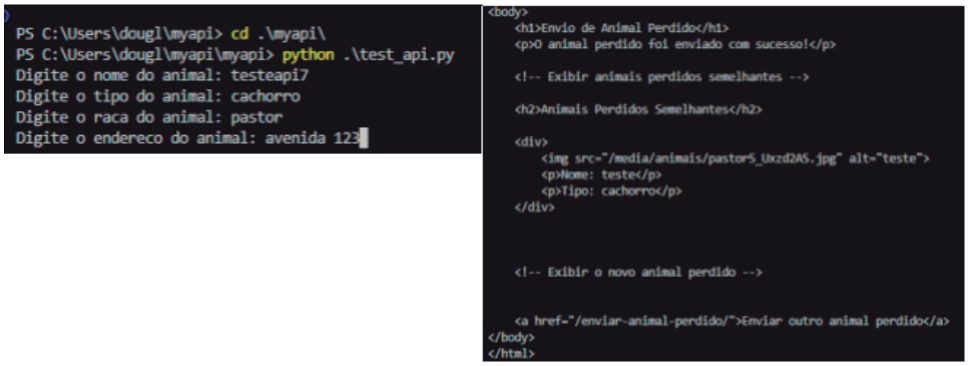
Na Figura 15, é mostrado o resultado do envio da imagem e a lista de possíveis animais semelhantes.

Figura 15 - Dados enviados.

Fonte: Construção do Autor.

A Figura 16 mostra o resultado da simulação de um usuário utilizado a API no arquivo test_ api˙py, onde mostra o formulário sendo preenchido no console e logo ap[os traz uma lista em HTML dos animais perdidos semelhantes ao da foto que foi enviada.

Figura 16 - Dados enviados.



Fonte: Construção do Autor**.**

CONCLUSÃO

O desenvolvimento desta API auxilia na busca por animais de estimação perdidos, utilizando técnicas de reconhecimento facial para dar esperança aos proprietários. O sistema conseguiu reconhecer imagens de animais comparando-as com as imagens armazenadas na base de dados, além de identificar as raças e o tipo de animal, demonstrando que os objetivos propostos foram cumpridos. Contudo, desafios como a complexidade do reconhecimento em diferentes raças, expressões faciais e condições de imagem foram encontrados. Uma base de dados com apenas 26 imagens não se mostrou eficiente para todos os animais. A precisão em imagens de baixa qualidade e em áreas com poucos dados também foi um obstáculo, além de o software não conseguir processar muitas imagens devido à pouca memória disponível.

Perspectivas futuras incluem melhorar os algoritmos de reconhecimento facial, ampliar a base de dados e usar técnicas avançadas de processamento de imagem. Possíveis soluções incluem aumentar o conjunto de imagens para o treinamento do random forest, aprimorar ainda mais o código YOLO e adicionar outras variáveis, como padrão de pelo, para o treinamento, pois foram utilizadas apenas imagens, tipo e raça do animal. Para acessar o código e os dados utilizados neste estudo, estão no repositório no GitHub.[5]

REFERÊNCIAS

BARBOSA, A. *et al.* **Metodologia ágil: Feature-Driven Development**. 2008. Acesso em: 07 nov. 2023.

BREIMAN, L. **Random Forest**. Journal of Machine Learning, v. 45, p. 5-32, 2001. Acesso em: 01 dez. 2023.

5 https://github.com/douglasisantos/apianimais-perdidos

COSTA, E. C. **Animais de estimação: uma abordagem psico-sociológica da concepção dos idosos**. 2006. Disponível em: https://www.uece.br/ppsacwp/wp-content/uploads/sites/37/2011/03/EDMARA-CHAVES-COSTA.pdf. Acesso em: 05 nov. 2023.

CRUZ, N. **Startup usa reconhecimento facial para identificar cachorros pelo focinho**. 2022. Disponível em: https://gizmodo.uol.com.br/empresa-cria-reconhecimento-facial-com-ia-para-cachorros-e-gatos/. Acesso em: 07 nov. 2023.

DINIZ, F. A. *et al.* **RedFace: um sistema de reconhecimento facial baseado em técnicas de análise de componentes principais e autofaces**. Revista Brasileira de Computação Aplicada, v. 5, n. 1, p. 42-54, maio 2013. DOI: 10.5335/rbca.2013.2627. Disponível em: https://seer.upf.br/index.php/rbca/article/view/2627. Acesso em: 06 nov. 2023.

DJANGO DOCUMENTATION. **Django Project**. 2024. Disponível em: https://docs.djangoproject.com/en/5.0/.

EQUIPE CÃESGATOS. **Aplicativo de reconhecimento facial de cães e gatos ajuda a encontrar pets perdidos**. 2021. Disponível em: https://caesegatos.com.br/aplicativo-de-reconhecimento-facial-de-caes-e-gatos-ajuda-a-encontrar-pets-perdidos/. Acesso em: 07 nov. 2023.

GEROTTI, G. W. **Sistema de monitoramento de gatos domésticos feito por reconhecimento individual através de machine learning**. 2023. Disponível em: https://repositorio.unesp.br/bitstreams/cb215e46-93c3-483e-8d51-24ee26f37731/content. Acesso em: 07 nov. 2023.

GHOSAL, V.; TIKMANI, P.; GUPTA, P. **Face Classification Using Gabor Wavelets and Random Forest**. In: CANADIAN CONFERENCE ON COMPUTER AND ROBOT VISION (CRV '09). IEEE Computer Society, Washington, DC, USA, 2009. p. 68-73. Acesso em: 06 nov. 2023.

KSHIRSAGAR, V. P.; BAVISKAR, M. R.; GAIKWAD, M. E. **Face recognition using Eigenfaces**. In: INTERNATIONAL CONFERENCE ON COMPUTER RESEARCH AND DEVELOPMENT (ICCRD), 2011. Acesso em: 06 nov. 2023.

LI, S. Z.; JAIN, A. K. **Handbook of Face Recognition**. 2. ed. 2011. Acesso em: 06 nov. 2023.

OPENCV. **Open Source Computer Vision Library: biblioteca de visão computacional open source**. Estados Unidos, Intel Corporation, 2001. 436 p. Acesso em: 06 nov. 2023.

RED HAT. **YOLOv3: An Incremental Improvement**. 2024.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**. 2018. Disponível em: https://arxiv.org/abs/1804.02767.

SETTE, L. H. M.; GUILHERME, A.; CARVALHO. **Reconhecimento facial pela região dos olhos utilizando OpenCV.** 2021. Disponível em: https://dspace.mackenzie.br/items/c388c23f-c733-4c36-a-698-5ee4c6e10950. Acesso em: 07 nov. 2023.

SILVA, F. G.; HOENTSCH, S. C. P.; SILVA, L. **Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS**.BR. Scientia Plena, v. 5, n. 12, nov. 2011. Disponível em: https://www.scientiaplena.org.br/sp/article/view/678. Acesso em: 07 nov. 2023.

SQLITE DOCUMENTATION. **SQLite**. 2024. Disponível em: https://sqlite.org/docs.html.

TATIBANA, L. S.; COSTA-VAL, A. P. **Relação homem-animal de companhia e o papel do médico veterinário**. 2009. Disponível em: https://crmvmg.gov.br/RevistaVZ/Revista03.pdf#page=11. Acesso em: 05 nov. 2023.

WAKKA, W. **Startup usa reconhecimento facial para identificar cachorros pelo focinho**. 2019. Disponível em: https://canaltech.com.br/apps/startup-usa-reconhecimento-facial-para-identificar-cachorros-pelo-focinho-144329/. Acesso em: 07 nov. 2023.

WALSH, F. **Human-Animal Bonds I: The Relational Significance of Companion Animals**. 2009. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1545-5300.2009.01296.x. Acesso em: 05 nov. 2023.

ZHANG, B.; RUAN, Q. **Facial feature extraction using improved deformable templates**. In: SIGNAL PROCESSING, 8th International Conference, 2006. Acesso em: 06 nov. 2023.