Desenvolvimento de uma API com Tecnologia de Reconhecimento Facial para Localização de Animais

Douglas Iracet dos Santos, Ana Paula Canal
Curso de Ciência da Computação
UFN - Universidade Franciscana
Santa Maria - RS
douglas.iracet@ufn.edu.br, apc@ufn.edu.br

***Resumo*—Este trabalho propõe 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, enfocando na eficácia dessa tecnologia. Foi desenvolvido utilizando a metodologia FDD, implementado em Python, utilizando a biblioteca OpenCV e os algoritmos YOLOv3 e Random Forest. O principal resultado é a geração de uma lista de animais semelhantes com base na imagem enviada pelo usuário.**

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

I. 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 propõe desenvolver 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.

A. Justificativa

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 [1].

Conviver com animais de estimação traz benefícios diversos. Crianças que crescem com animais desenvolvem maior afetividade e senso de responsabilidade [2]. Para os idosos, os animais oferecem conforto, aumentam a autoestima e promovem a interação social [3]. Para muitos donos, os animais são vistos como membros da família [4].

Quando um animal desaparece, a busca torna-se uma prioridade urgente, utilizando métodos tradicionais e tecnologia avançada. Este projeto desenvolve uma API utilizando técnicas de reconhecimento facial para localizar animais perdidos, demonstrando como a tecnologia pode beneficiar tanto a sociedade quanto os animais queridos.

B. Objetivo Geral

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.

Os objetivos específicos foram:

- Estudar os algoritmos de reconhecimento facial, YOLOv3 e Random Forest.
- Implementar esses algoritmos usando Python e a biblioteca OpenCV.
- Testar API desenvolvida.

II. Referencial Teórico

Nessa seção são discutidas as ferramentas que foram utilizadas para o desenvolvimento da API que utiliza técnicas de reconhecimento facial.

A. Reconhecimento facial

Existem dois tipos de abordagens fundamentais para o reconhecimento facial [5]. 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 [5]. 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 [5] [6].

Existem diferentes tipoes de abordagens como por exemplo em Ghosal, Tikmani e Gupta [7], 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, Baviskare e Gaikwad [8], utiliziaram métodos de Análise dos Componentes Principais (PCA) e técnicas estatísticas. Stane Anil [5] implementou 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[9].

B. Python

Python é uma linguagem de programação de alto nível, interpretada, dinâmica e modular, que é multiplataforma e multiparadigma. Essa abordagem específica organiza o software de forma a submeter os procedimentos às classes, o que oferece um maior controle e estabilidade aos códigos, tornando-a uma escolha ideal para o projeto [10].

C. OpenCV

O OpenCV é uma biblioteca de código aberto criada pela Intel em 2000, projetada para aplicações de visão computacional em tempo real. Originalmente desenvolvida em C/C++, é altamente portável e suporta diversos sistemas operacionais, sendo amplamente utilizada em ambientes acadêmicos e comerciais [11].

O OpenCV é uma biblioteca abrangente que inclui módulos para processamento de imagens e vídeo, estruturas de dados, álgebra linear, GUI e controle de dispositivos de entrada. Com mais de 2500 algoritmos, muitos considerados padrões no campo da visão computacional, cobre uma ampla gama de áreas como segmentação, reconhecimento facial (incluindo o método Viola-Jones), aprendizado de máquina, filtragem de imagens, rastreamento de movimento e outras técnicas relevantes [11].

D. SQLite com Django Framework

O Django é um framework web em Python projetado para desenvolvimento rápido e eficiente de aplicativos web escaláveis e robustos [12]. Possui um sistema de ORM (Object-Relational Mapping), permitindo aos desenvolvedores interagir com o banco de dados usando objetos Python, em vez de consultas SQL diretas. Isso simplifica a manipulação de dados e aumenta a portabilidade do código entre diferentes bancos de dados relacionais [12].

O SQLite é frequentemente preferido para projetos menores ou em desenvolvimento devido à sua natureza leve e integrada. 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 [13].

O Django utiliza migrações para garantir a consistência entre o modelo de dados definido em classes Python e a estrutura do banco de dados. Todas as alterações no modelo são registradas em migrações que podem ser aplicadas ao banco de dados usando o comando `manage.py migrate`

E. API

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 [14].

F. Random Forest

O método de classificação Random Forest, proposto por Breiman [15], é uma técnica de agregação de classificadores do tipo árvore, onde cada árvore é construída de forma aleatória [7]. 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 [15].

G. YOLOv3

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. 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 [16].

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.

III. Trabalhos correlatos

Os trabalhos correlatos exploram o reconhecimento facial aplicado à localização de animais perdidos, detalhando abordagens, tecnologias e contribuições relevantes nesse campo específico.

A Pupz é uma plataforma pet brasileira que desenvolveu um aplicativo gratuito de reconhecimento facial para cães e gatos, com mais de 94% de precisão, para ajudar a encontrar animais perdidos. A plataforma permite que os usuários cadastrem seus pets gratuitamente e usem a câmera do celular para reconhecimento facial.[17].

A *startup* chinesa Megvii também utiliza reconhecimento facial para identificar cachorros perdidos, focando nas singularidades do focinho dos cães. O aplicativo alcança uma taxa de acerto de 95% em testes com 15 mil cachorros, ajudando a encontrar animais e multar tutores negligentes [18].

A Petvation desenvolveu uma porta inteligente para animais de estimação que utiliza reconhecimento facial para garantir a segurança dos pets. A porta é equipada com câmeras infravermelhas em ambos os lados, oferecendo um campo de visão de 120 graus. Ela identifica o animal de estimação com base na sua aparência utilizando iluminação infravermelha. Os tutores podem controlar os horários de entrada e saída do pet, eliminando a necessidade de verificar se a porta está trancada.[19].

Um sistema de monitoramento de gatos domésticos foi desenvolvido utilizando aprendizagem de máquina para reconhecimento individual. Esse sistema identifica gatos específicos em imagens e executa ações correspondentes. As tecnologias usadas incluem YOLOv8, BiT e OpenCV, com um enfoque distintivo na região dos olhos para o reconhecimento facial.[20]. Uma característica distintiva deste algoritmo é a utilização de estudos para realizar o reconhecimento facial com foco na região dos olhos, o que oferece várias vantagens em comparação com os métodos de reconhecimento atualmente utilizados [20].

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. Este trabalho se destaca pelo uso de algoritmos como Random Forest e YOLOv3 para desenvolver uma API eficiente de reconhecimento facial de animais perdidos, pois essas duas tecnologias se mostraram muito eficientes no reconhecimento de animais e na extração de características, sendo integráveis a diversas plataformas e serviços.

IV. Metodologia

O *Feature-Driven Development (FDD)* é uma metodologia ágil de desenvolvimento de software. Ela é composta por cinco processos distintos: Desenvolvimento de um Modelo Abrangente, Construção de uma Lista de Funcionalidades, Planejamento por Meio de Funcionalidades, Projeto por Meio de Funcionalidades e Construção por Meio de Funcionalidades [21]

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, conhecido como o dono do produto [21]

A. Desenvolvimento do modelo abrangente

Durante a elaboração do modelo abrangente, as enitdades de escopo são estabelecidas a partir das especificações do domínio de negócios, conforme descrito por [21]. O Diagrama de domínio é produzido, como mostrado na Figura 1, proporcionando uma visão panorâmica da aplicação ao apresentar suas principais entidades e associações.

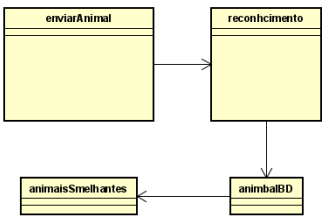

Figura 1. Diagrama de domínio.

A Figura 2 fornece uma visão detalhada do Diagrama de Atividade principal do sistema. Nesse ponto, 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 semalhantes 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 o usuário caso o animal seja encontrado

B. Lista de funcionalidades

Esta fase tem como finalidade identificar os requisitos funcionais da API, como descrito na Tabela 1.

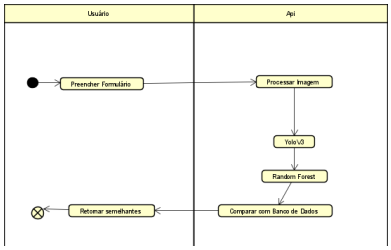

Figura 2. Diagrama de Atividade.

Tabela I
TABELA DE REQUISITOS FUNCIONAIS

Requisitos Funcionais		
Funcionalidade	**Descrição**	**Complexidade**
RF01	A API deve aceitar o registro de animais perdidos, permitindo informações detalhadas, como fotos e características.	Média
RF02	A API deve realizar reconhecimento facial para auxiliar na identificação de animais perdidos e correspondência com registros de animais.	Alta
RF03	A API deve permitir atualizações de dados de animais perdidos, como a adição de novas imagens ou informações.	Média

Na Tabela 2, são apresentados os requisitos não funcionais, que desempenham um papel crucial na manutenção da proposta de desenvolvimento da aplicação e estabelecem as principais diretrizes tecnológicas e caminhos a serem seguidos no projeto.

Tabela II
TABELA DE REQUISITOS NÃO FUNCIONAIS

Requisitos Não Funcionais	
Requisito	**Descrição**
RNF01	A API deve ser projetada para suportar a carga de solicitações simultâneas de reconhecimento facial de animais perdidos, implementando algoritmos eficientes como YoloV3, 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 3 representa a interação entre os atores e os casos de uso, exibindo suas relações e delineando as funcionalidades do sistema.

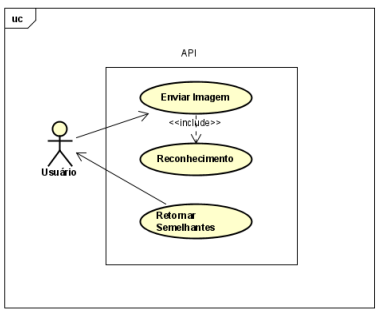

Figura 3. Diagrama de caso de uso.

C. Planejar por Funcionalidade

Durante o planejamento das funcionalidades, são definidas a ordem de implementação e um conjunto inicial de datas para a execução de cada funcionalidade [22].

Na Tabela 3, está a estimativa de tempo, considerando a complexidade de cada requisito. O tempo total estimado para a implementação das funcionalidades foi de 75 horas.

Tabela III
ESTIMATIVA DE TEMPO PARA IMPLEMENTAÇÃO DAS FUNCIONALIDADES

Requisitos Funcionais	
Funcionalidade	**Tempo (horas)**
RF01	8
RF02	10
RF03	25

D. Desenvolvimento da API

Está seção aborda a implementação da API com técnicas de reconhecimento facial para localização de animais perdidos. Destaca-se o treinamento do modelo Random Forest com a biblioteca *scikit-learn* em Python. São mencionadas as principais bibliotecas importadas, como *os*, *cv2*, *numpy* e *sklearn*, para suporte às operações matemáticas, processamento de imagens e aprendizado de máquina.

A seção também traz a documentação dos *endpoints* da API. E Algoritmos como YOLOv3 e Random Forest são destacados como fundamentais para o reconhecimento facial na API. A Figura 4 mostra o diagrama de sequência da API.

1) Treinamento do Random Forest: Para o treinamento do modelo Random Forest, foi utilizado as biblioteca scikit-learn, uma ferramenta utilizada para aprendizado de máquina em Python. As principais bibliotecas importadas foram:

- `import os`
  Interação com o sistema operacional.
- `import cv2`
  Processamento de imagens e visão computacional.
- `import numpy as np`
  Operações matemáticas e arrays.

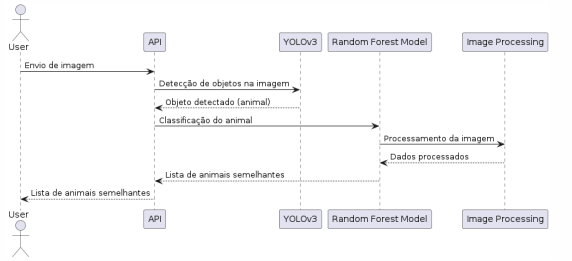

Figura 4. Diagrama de sequência .

- `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.
- `import joblib`
  Salvar e carregar modelos.

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) Código para Carregar e Treinar o Modelo: O Código 1 implementa uma 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.

```
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))
```

Código 1. Função para carregar imagens

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 como mostra o Código 2.

```
# 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)
```

Código 2. treinamento do random forest

4) Avaliação do Modelo: A performance do modelo foi avaliada usando métricas como precisão, revocação,recall e F1-score. No Código 3 mostra como foi feita essas avaliações.

```
# 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)
```

Código 3. avaliação do modelo

Na Figura 5 traz a precisão, revocação e F1-score. O modelo apresentou uma precisão de 100% para a classe "pastor-alemao". 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".

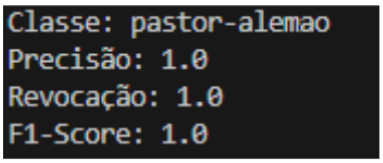

Figura 5. precisão do modelo .

Na Figura 6, está apresentada a quantidade de fotos utilizadas para o treinamento, totalizando 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.

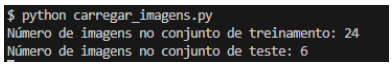

Figura 6. arquivos carregados.

Na Figura 7 e Código 4 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.

```
# Calcular a matriz de confus o
conf_matrix = confusion_matrix(y_test, y_pred)

# Exibir a matriz de confus o como um heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap
    ='Blues', xticklabels=rf_classifier.classes_,
    yticklabels=rf_classifier.classes_)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

Código 4. avaliação do modelo

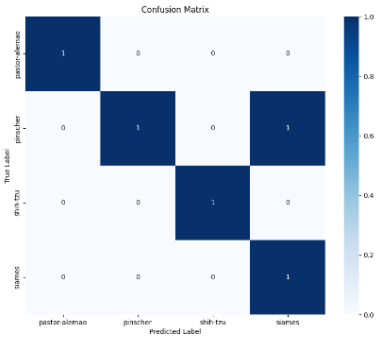

Figura 7. Matriz de Confusão.

5) Banco de Dados Utilizado: O banco de dados utilizado neste projeto foi construído 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 5 traz o Código 5 onde é implementada uma model com campos que vão preencher a tabela do banco de dados.

```
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
```

Código 5. Criação da tabela do banco

Para viabilizar a busca e filtragem de animais perdidos, foi utilizado os recursos do Django ORM. Foi implementado 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.

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. O Código 6 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.

```
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 evitar
    detec  es m ltiplas do mesmo objeto
    indexes = cv2.dnn.NMSBoxes(boxes, confidences,
     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])
```

Código 6. detecao_objetos_yolo.py

No Código 7 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.

```
# 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
```

Código 7. detecao_objetos_yolo.py

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` , 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.

```python
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})
```

Código 8. detecao_objetos_yolo.py

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 da seguinte forma.

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

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

Código 9. detecao_objetos_yolo.py

9) Teste da Api: No Código 8, 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.

```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})
```

```
26
27 # Exibindo a resposta da API
28 print('Resposta da API:', response.text)
```

Código 10. `detecao_objetos_yolo.py`

V. Resultados

Como resultado do arquivo `detecao_objetos_yolo.py` foi obtido um retângulo na imagem da foto testada com a classificação do animal como na Figura 8.

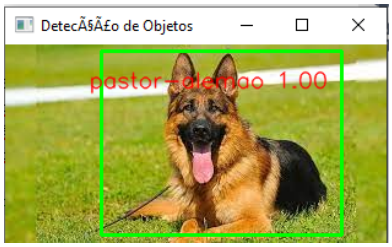

Figura 8. Resultado da foto enviada

A Figura 9 traz uma página web com o *endpoint* criado na API com uma lista inicial de animais que ja estão salvos no banco de dados.

Figura 9. Página web com lista de animais .

Na Figura 10 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.

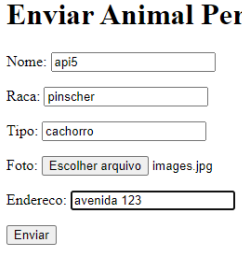

Figura 10. *endpoint* para enviar animal .

Na Figura 11, é mostrado o resultado do envio da imagem e a lista de possíveis animais semelhantes.

A Figura 12 mostra o resultado da simulação de um usuário utilizado a API no arquivo `test_apiṗy`, onde mostra o

Figura 11. Resultado da imagem enviada com animais semalhantes .

formulário sendo preenchido no console. Na Figura 13 traz uma lista em HTML dos animais perdidos semelhantes ao da foto que foi enviada.

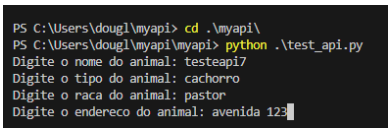

Figura 12. Dados enviados .

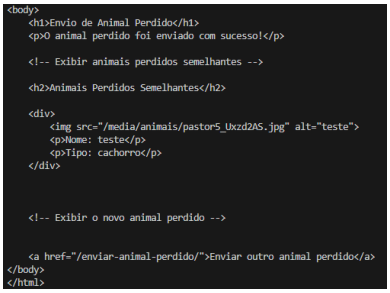

Figura 13. Retorno html da lista de animais semelhantes .

VI. 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, visite o repositório no GitHub:https://github.com/douglasisantos/api-animais-perdidos.

Referências

[1] Da Redação. "Animais domésticos". Em: (2006). URL: https://super.abril.com.br/ciencia/animais-domesticos4 (acesso em 05/11/2023).

[2] Tatibana L. S. e Costa-Val A. P. "Relação homem-animal de companhia e o papel do médico veterinário". Em: (2009). URL: https://crmvmg.gov.br/RevistaVZ/Revista03.pdf#page=11 (acesso em 05/11/2023).

[3] Costa E. C. "Animais de estimação: uma abordagem psico-sociológica da concepção dos idosos". Em: (2006). URL: https://www.uece.br/ppsacwp/wp-content/uploads/sites/37/2011/03/EDMARA-CHAVES-COSTA.pdf (acesso em 05/11/2023).

[4] Walsh F. "Human-Animal Bonds I: The Relational Significance of Companion Animals". Em: (2009). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1545-5300.2009.01296.x (acesso em 05/11/2023).

[5] Stan Z. Li. e Anil K. J. "Handbook of Face Recognition, 2nd Edition". Em: (2011). (Acesso em 06/11/2023).

[6] Zhang B. e Ruan Q. "Facial feature extraction using improved deformable templates, Signal Processing, 8th International Conference on, vol.4, no., 16-20". Em: (2006). (Acesso em 06/11/2023).

[7] Ghosal V., Tikmani P. e Gupta P. "Face Classification Using Gabor Wavelets and Random Forest, In Proceedings of the Canadian Conference on Computer and Robot Vision (CRV '09), IEEE Computer Society, Washington, DC, USA, 68-73". Em: (2009). (Acesso em 06/11/2023).

[8] Kshirsagar V. P., Baviskar M. R e Gaikwad M. E. "Face recognition using Eigenfaces, Computer Research and Development (ICCRD)". Em: (2011). (Acesso em 06/11/2023).

[9] Fabio Abrantes Diniz et al. "RedFace: um sistema de reconhecimento facial baseado em técnicas de análise de componentes principais e autofaces". Em: *Revista Brasileira de Computação Aplicada* 5.1 (maio de 2013), pp. 42–54. DOI: 10.5335/rbca.2013.2627. URL: https://seer.upf.br/index.php/rbca/article/view/2627.

[10] Luiz H. M. de Sette Guilherme A.;Carvalho. "Reconhecimento facial pela região dos olhos utilizando OpenCV". Em: (2021). URL: https://dspace.mackenzie.br/items/c388c23f-c733-4c36-a698-5ee4c6e10950 (acesso em 07/11/2023).

[11] OPENCV. "Open Source Computer Vision Library: biblioteca de visão computacional open source. Estados Unidos, Intel Corporation, 2001 [436 p.]." Em: (2012). URL: http://developer.intel.com (acesso em 06/11/2023).

[12] Django Documentation. "Django Project". Em: (2024). Disponível em https://docs.djangoproject.com/en/5.0/.

[13] SQLite Documentation. "SQLite". Em: (2024). Disponível em https://sqlite.org/docs.html.

[14] Red Hat. "YOLOv3: An Incremental Improvement". Em: (2024).

[15] Breiman L. "Random Forest.In Journal of Machine Learning, Vol.45, pages 5-32". Em: (2001). (Acesso em 01/12/2023).

[16] Redmon J. Farhadi A. "YOLOv3: An Incremental Improvement". Em: (2018). Disponível em https://arxiv.org/abs/1804.02767.

[17] Equipe CãesGatos. "Aplicativo de reconhecimento facial de cães e gatos ajuda a encontrar pets perdidos". Em: (2021). URL: https://caesegatos.com.br/aplicativo-de-reconhecimento-facial-de-caes-e-gatos-ajuda-a-encontrar-pets-perdidos/ (acesso em 07/11/2023).

[18] Wagner Wakka. "Startup usa reconhecimento facial para identificar cachorros pelo focinho". Em: (2019). URL: https://canaltech.com.br/apps/startup-usa-reconhecimento-facial-para-identificar-cachorros-pelo-focinho-144329/ (acesso em 07/11/2023).

[19] Nubia Da Cruz. "Startup usa reconhecimento facial para identificar cachorros pelo focinho". Em: (2022). URL: https://gizmodo.uol.com.br/empresa-cria-reconhecimento-facial-com-ia-para-cachorros-e-gatos/ (acesso em 07/11/2023).

[20] Gabriel Whitacker Gerotti. "Sistema de monitoramento de gatos domésticos feito por reconhecimento individual através de machine learning." Em: (2023). URL: https://repositorio.unesp.br/server/api/core/bitstreams/cb215e46-93c3-483e-8d51-24ee26f37731/content (acesso em 07/11/2023).

[21] F. G. Silva, S. C. P. Hoentsch e L. Silva. "Uma análise das Metodologias Ágeis FDD e Scrum sob a Perspectiva do Modelo de Qualidade MPS.BR". Em: *Scientia Plena* 5.12 (nov. de 2011). URL: https://www.scientiaplena.org.br/sp/article/view/678.

[22] Barbosa A. et al. "Metodologia ágil: Feature-Driven Development". Em: (2008). (Acesso em 07/11/2023).