

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

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 perdidos. 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

O objetivo primordial deste projeto consistiu em implementar uma *Application Programming Interface (API)* de reconhecimento facial. Esta API foi projetada para identificação de animais através de imagens faciais compartilhadas por usuários, baseado nos algoritmos YOLOV3, Random Forest. O propósito final foi criar uma ferramenta capaz de proporcionar uma busca rápida para reunir animais perdidos com seus proprietários.

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 duas 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].

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.

No trabalho de Kshirsagar, Baviskare e Gaikwad [8], foram utilizados 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].

O método de classificação Random Forest, proposto por Breiman [10], 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 [10].

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. [11].

Para implementar o YOLOv3 em qualquer linguagem de programação três arquivos principais são necessários:

- **yolo3.weights**: Contém os pesos pré-treinados do modelo.
- **yolo3.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.

Este trabalho se concentrou na concepção de uma API que utiliza classificadores como Random Forest para classificação facial. O processo inicia com o módulo de aquisição de

imagem, utilizando o algoritmo YOLOV3 para detectar um objeto na imagem, implementado pela biblioteca OpenCV juntamente com a linguagem Python.

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 projetos como está API [12].

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 portátil e suporta diversos sistemas operacionais, sendo amplamente utilizada em ambientes acadêmicos e comerciais.[13].

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.[13].

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. Focado em maximizar a produtividade dos desenvolvedores, Django segue o princípio do "batteries-included", oferecendo uma vasta gama de funcionalidades prontas para uso. [14].

O Django possui um sistema de ORM (Object-Relational Mapping) poderoso, 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. [14].

Embora o Django suporte vários bancos de dados como PostgreSQL, MySQL, Oracle, entre outros, 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.[15].

Ao utilizar o SQLite com o Django, os desenvolvedores podem aproveitar as vantagens do framework, incluindo sua administração automática de banco de dados, migrações de esquema simplificadas e uma biblioteca de recursos para desenvolvimento web [15].

III. TRABALHOS CORRELATOS

Esta seção apresenta os trabalhos correlatos que 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. Fundada por Carlos Fabbro, inspirado por sua Golden Retriever Chanel, a plataforma permite que os usuários cadastrem seus pets gratuitamente e usem a câmera do celular para reconhecimento facial. O aplicativo facilita a localização dos tutores, mantendo a segurança e a privacidade conforme a LGPD.[16].

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 [17].

A Petvation desenvolveu uma porta inteligente para animais de estimação que utiliza reconhecimento facial para garantir a segurança dos pets. Feita de liga de alumínio e operada por um motor que desliza verticalmente, 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. Para segurança adicional, sensores de movimento e uma câmera voltada para baixo garantem que a porta não se feche até que o animal tenha passado completamente, evitando acidentes. [18].

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.[19]. Entre as tecnologias investigadas, destacam-se o YOLOv8 (You Only Look Once, versão 8), o BiT (Big Transfer) e o OpenCV, que são frameworks pré-treinados em Python para detecção de objetos [19]. 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 [19].

Os estudos mencionados exploram o reconhecimento facial para localização e identificação de pets perdidos, além de monitorar sua presença em ambientes específicos. Utilizam abordagens variadas, desde algoritmos próprios focados em características biométricas até tecnologias como YOLOv8, 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, integrável a diversas plataformas e serviços.

IV. METODOLOGIA

O *Feature-Driven Development (FDD)* é uma metodologia ágil de desenvolvimento de software que combina aspectos rigorosos como modelagem, planejamento e controle com práticas ágeis. 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.[20]

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.[20]

A. Desenvolvimento do modelo abrangente

Durante a elaboração do modelo abrangente, as entidades de escopo são estabelecidas a partir das especificações do domínio de negócios, conforme descrito por [20]. O Diagrama de domínio é produzido, como mostrado na Figura 1, proporcionando uma visão panorâmica da aplicação ao apresentar suas principais entidades e associações.

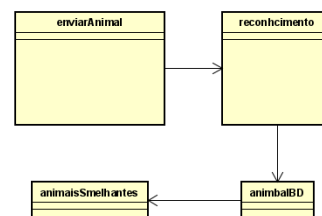


Figura 1. Diagrama de domínio.

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

```

graph TD
    subgraph "Usuário"
        A[Receber formulário]
        F[Retornar resultados]
    end
    subgraph "App"
        B[Processar Imagem]
        C[Validar]
        D[Random Forest]
        E[Comparar com Banco de Dados]
    end
    A --> B
    B --> C
    C --> D
    D --> E
    E --> F
  
```

B. Lista de funcionalidades

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

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.

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[21].

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 <i>endpoints</i> da API devem ser intuitivos e bem descritos para facilitar o uso por parte dos desenvolvedores.

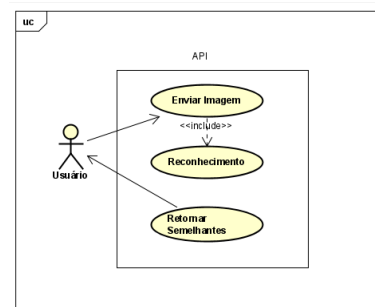


Figura 3. Diagrama de caso de uso.

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

Esta 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.

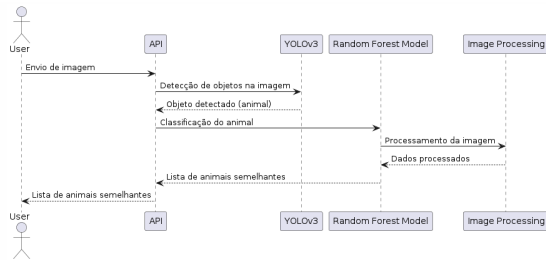


Figura 4. diagrama de sequência .

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.
- `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.

```

1 def carregar_imagens():
2     imagens = []
3     labels = []
4
5     # Percorre recursivamente os subdiretórios
6     for root, dirs, files in os.walk(IMAGES_DIR):
7         for filename in files:
8             # Verifica se o arquivo é uma imagem
9             if filename.endswith(".jpg") or
10                filename.endswith(".png"):
11                 # Carrega a imagem
12                 image_path = os.path.join(root,
13                    filename)
14                 image = cv2.imread(image_path)
15                 if image is not None:
16                     # Redimensiona a imagem para o
17                     tamanho desejado
18                     resized_image = cv2.resize(
19                         image, NEW_SIZE)
20                     # Adiciona a imagem e a classe
21                     ao conjunto de dados
22                     imagens.append(resized_image.
23                        flatten()) # Transforma a imagem em um vetor
24                     unidimensional
25                     label = os.path.basename(os.
26                        path.dirname(image_path))
27                     labels.append(label)
28
29     return imagens, labels
30
31 # Carregar imagens e labels
32 imagens, labels = carregar_imagens()
33
34 # Dividir os dados em conjuntos de treinamento e
35 teste
36 X_train, X_test, y_train, y_test =
37     train_test_split(imagens, labels, test_size
38                     =0.2, random_state=42)
39
40 # Exibir o número de imagens em cada conjunto
41 print("Número de imagens no conjunto de
42     treinamento:", len(X_train))
43 print("Número de imagens no conjunto de teste:",
44     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.


```

1 # Criar e treinar o modelo RandomForest
2 rf_classifier = RandomForestClassifier()
3 rf_classifier.fit(X_train, y_train)
4
5 # Salvar o modelo treinado
6 MODELOS_DIR = "C:/Users/dougl/myapi/modelos"
7 if not os.path.exists(MODELOS_DIR):
8     os.makedirs(MODELOS_DIR)
9 joblib.dump(rf_classifier, os.path.join(
10     MODELOS_DIR, "modelo_random_forest.pkl"))
11
12 # Prever as classes no conjunto de teste
13 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.

```

1 # Calcular a precisão para cada classe
2 precisao_por_classe = precision_score(y_test,
3     y_pred, average=None)
4
5 # Calcular a revocação para cada classe
6 revocacao_por_classe = recall_score(y_test, y_pred,
7     average=None)
8
9 # Calcular o F1-Score para cada classe
10 fl_por_classe = f1_score(y_test, y_pred, average=
11     None)

```

Código 3. avaliação do modelo

Na figura 5 traz a precisão, revocação e F1-score. O modelo apresentou uma precisão de 33.33% para a classe "animais", indicando que apenas um terço das previsões estavam corretas, revelando muitos falsos positivos. A revocação foi de 100%, mostrando que o modelo identificou corretamente todas as instâncias reais de "animais", sem falsos negativos. O F1-Score de 50% reflete um equilíbrio entre a alta revocação e a baixa precisão, sugerindo que, embora o modelo seja eficaz em encontrar todas as instâncias de "animais", ele também classifica incorretamente muitos outros como "animais".

```

Classe: animais
Precisão: 0.3333333333333333
Revocação: 1.0
F1-Score: 0.5

```

Figura 5. precisão do modelo.

Na Figura 6 está a quantidade de fotos utilizadas para o treinamento e o número de conjuntos que foram testados por exemplo os conjuntos são divididos em tipo do animal gato ou cachorro e a raça desses animais pastor alemão e siames.

```

$ python carregar_imagens.py
Número de imagens no conjunto de treinamento: 24
Número de imagens no conjunto de teste: 6

```

Figura 6. arquivos carregados.

Na Figura 7 e Código 4 traz 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.

```

1 # Calcular a matriz de confusão
2 conf_matrix = confusion_matrix(y_test, y_pred)
3
4 # Exibir a matriz de confusão como um heatmap
5 plt.figure(figsize=(10, 8))
6 sns.heatmap(conf_matrix, annot=True, fmt='d', cmap
7     ='Blues', xticklabels=rf_classifier.classes_,
8     yticklabels=rf_classifier.classes_)
9 plt.xlabel('Predicted Label')
10 plt.ylabel('True Label')
11 plt.title('Confusion Matrix')
12 plt.show()

```

Código 4. avaliação do modelo

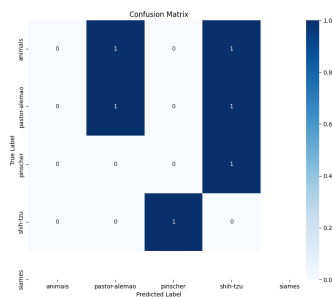


Figura 7. Matriz de Precisão.

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. SQLite é uma biblioteca em C que oferece um banco de dados SQL embutido, ideal para aplicações que necessitam de uma solução leve e fácil de administrar.

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.

```

1 from django.db import models
2 from django.utils import timezone

```

```

3
4 class AnimalPerdido(models.Model):
5     nome = models.CharField(max_length=100)
6     tipo = models.CharField(max_length=100)
7     raca = models.CharField(max_length=100)
8     descricao_objetos = models.CharField(
9         max_length=100, default='Descrição não
10         fornecida')
11     foto = models.ImageField(upload_to='animais/')
12     data_criacao = models.DateTimeField(default=
13         timezone.now)
14     endereco = models.CharField(max_length=100,
15         default='')
16
17 def __str__(self):
18     return self.nome

```

Código 5. carregar_imagens.py

Para viabilizar a busca e filtragem de animais perdidos, foi utilizado os recursos do Django ORM. Implementamos uma função na API que emprega consultas do tipo *filter* e *exclude* para encontrar animais perdidos com características semelhantes às fornecidas pelo usuário. 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`.

O banco de dados SQLite devido à sua configuração e gerenciamento simplificados. Durante o desenvolvimento do projeto, executamos 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 `detectar_objetos` aonde carrega a imagem que o usuário envia na API e o algoritmo identifica se na imagem possui um objeto no caso o animal e retorna o resultado para carregar a imagem ele usa a biblioteca OpenCV.

```

1 def detectar_objetos(imagem):
2     # Carrega os arquivos de configura o e
3     # pesos do YOLO
4     net = cv2.dnn.readNet(os.path.join(YOLO_DIR, "
5     yolov3.cfg"), os.path.join(YOLO_DIR, "yolov3.
6     weights"))
7     classes = []
8     with open(os.path.join(YOLO_DIR, "coco.names")
9     , "r") as f:
10         classes = [line.strip() for line in f.
11         readlines()]
12
13     # Convertendo o objeto InMemoryUploadedFile
14     # para uma imagem OpenCV
15     imagem_opencv = processar_imagem(imagem)
16
17     # Detecta objetos na imagem

```

```

12 blob = cv2.dnn.blobFromImage(imagem_opencv,
13     0.00392, (416, 416), (0, 0, 0), True, crop=
14     False)
15 net.setInput(blob)
16 outs = net.forward(get_output_layers(net))
17
18 # Processa as detec es e retorna os
19 resultados
20 class_ids = []
21 confidences = []
22 boxes = []
23 for out in outs:
24     for detection in out:
25         scores = detection[5:]
26         class_id = np.argmax(scores)
27         confidence = scores[class_id]
28         if confidence > 0.5:
29             # Objeto detectado com confian a
30             center_x = int(detection[0] *
31             imagem_opencv.shape[1])
32             center_y = int(detection[1] *
33             imagem_opencv.shape[0])
34             w = int(detection[2] *
35             imagem_opencv.shape[1])
36             h = int(detection[3] *
37             imagem_opencv.shape[0])
38             x = int(center_x - w / 2)
39             y = int(center_y - h / 2)
40             boxes.append([x, y, w, h])
41             confidences.append(float(
42             confidence))
43             class_ids.append(class_id)
44
45 # Aplica supress o n o-m xima para evitar
46 detec es mltiplas do mesmo objeto
47 indexes = cv2.dnn.NMSBoxes(boxes, confidences,
48     0.5, 0.4)
49
50 # Prepara os resultados
51 resultados = {
52     'objetos_detectados': [],
53     'scores': [],
54     'descricao_objetos': []
55 }
56 for i in range(len(boxes)):
57     if i in indexes:
58         x, y, w, h = boxes[i]
59         label = str(classes[class_ids[i]])
60         resultados['objetos_detectados'].
61         append(label)
62         resultados['scores'].append(
63         confidences[i])

```

Código 6. detecao_objetos_yolo.py

No Código 7 é feita a classificação do resultado da função `detectar_objetos` e utiliza o random forest o resultado do random forest que foi salvo um arquivo .pkl e faz a classificação do animal que está na foto e retorna o resultado.

```

1 # Classifica os animais detectados na imagem com o
2 # modelo Random Forest
3 for i, objeto in enumerate(resultados['
4     objetos_detectados']):
5     if objeto == 'dog' or objeto == 'cat':
6         # Regi o de interesse para
7         classifica o
8         x, y, w, h = boxes[i]
9         roi = imagem_opencv[y:y+h, x:x+w]
10        # Classifica o animal na regi o de
11        interesse

```

```

8         print("Classificando animal...")
9         classificacao = classificar_animal(roi
10     )
11     print("Resultado da classifica o:",
12         classificacao)
13     resultados['objetos_detectados'][i] =
14     classificacao['objetos_detectados'][0]
15     resultados['scores'][i] =
16     classificacao['scores'][0]
17
18     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.

```

1 from django.shortcuts import render, redirect
2 from django.http import HttpResponse
3 from .detecao_objetos_yolo import detectar_objetos
4 # Importe a fun o detectar_objetos
5 from .models import AnimalPerdido
6 from .forms import AnimalPerdidoForm
7 from rest_framework.views import APIView
8 from rest_framework.response import Response
9 from rest_framework import status
10 from django.db.models import Q
11
12 class EnviarAnimalPerdidoAPIView(APIView):
13     def post(self, request, format=None):
14         form = AnimalPerdidoForm(request.POST,
15             request.FILES)
16         if form.is_valid():
17             nome = form.cleaned_data['nome']
18             tipo = form.cleaned_data['tipo']
19             foto = form.cleaned_data['foto']
20             descricao = form.cleaned_data['
21             descricao']
22
23             # Processar a imagem enviada para
24             detectar_objetos
25             imagem_cv2 = processar_imagem(foto) #
26             Fun o para processar a imagem (defina essa
27             fun o)
28             resultados = detectar_objetos(
29             imagem_cv2) # Fun o para detectar objetos
30
31             # Salvar o novo animal no banco de
32             dados
33             novo_animal = AnimalPerdido(nome=nome,
34             tipo=tipo, descricao=descricao, foto=foto)
35             novo_animal.save()
36
37             # Filtrar animais semelhantes (
38             considerando a l gica desejada)
39             animais_perdidos_semelhantes =
40             AnimalPerdido.objects.filter(

```

```

30         Q(nome__icontains=nome) |
31         Q(tipo__icontains=tipo) |
32         Q(descricao__icontains=descricao)
33     ).exclude(id=novo_animal.id)
34
35     # Serializar os dados dos animais
36     perdidos semelhantes
37     serializer = AnimalPerdidoSerializer(
38     animais_perdidos_semelhantes, many=True)
39
40     return Response({
41         'novo_animal':
42         AnimalPerdidoSerializer(novo_animal).data,
43         'animais_perdidos_semelhantes':
44         serializer.data,
45         'objetos_detectados': resultados['
46         objetos_detectados']
47     })
48     return Response(form.errors, status=status
49     .HTTP_400_BAD_REQUEST)

```

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.

```

1 from django.urls import path
2 from .views import EnviarAnimalPerdidoAPIView
3
4 urlpatterns = [
5     path('enviar-animal-perdido-api/',
6         EnviarAnimalPerdidoAPIView.as_view(), name='
7         enviar_animal_perdido_api'),
8 ]

```

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.

```

1 import requests
2
3 # URL da sua API
4 url = 'http://localhost:8000/enviar-animal-perdido/'
5
6 # Solicitar ao usu rio os dados do animal perdido
7 nome = input("Digite o nome do animal: ")
8 tipo = input("Digite o tipo do animal: ")
9 raca = input("Digite o raca do animal: ")
10 endereco = input("Digite o endereco do animal: ")
11
12 # Caminho do arquivo de imagem
13 caminho_imagem = 'C:/Users/dougl/myapi/media/
14     animais/cachorro/pastor-alemao/pastor.jpg'

```



```

15 # Dados do animal perdido
16 data = {
17     'nome': nome,
18     'tipo': tipo,
19     'raca': raca,
20     'endereço': endereço,
21 }
22
23 # Fazendo uma solicitação POST sua API para
    enviar um animal perdido
24 with open(caminho_imagem, 'rb') as arquivo_imagem:
25     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.

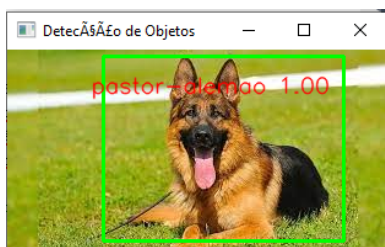


Figura 8. Resultado da foto enviada

A figura 9 traz uma página web com o *endpoint* criado na API com uma lista inicial de animais que já 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.

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 utilizando a API no arquivo `test_api.py`, onde mostra o formulário sendo preenchido no console. Na Figura 13 traz uma lista em HTML dos animais perdidos semelhantes ao da foto que foi enviada.

Enviar Animal Perdido

Nome:

Raca:

Tipo:

Foto:

Endereço:

Figura 10. *endpoint* para enviar animal .

Envio de Animal Perdido

O animal perdido foi enviado com sucesso!

Animais Perdidos Semelhantes



Figura 11. resultado da imagem enviada com animais semelhantes .

```

PS C:\Users\dougl\myapi> cd .\myapi\
PS C:\Users\dougl\myapi\myapi> python .\test_api.py
Digite o nome do animal: testeapi7
Digite o tipo do animal: cachorro
Digite o raca do animal: pastor
Digite o endereço do animal: avenida 123

```

Figura 12. Dados enviados .

```

<body>
  <h1>Envio de Animal Perdido</h1>
  <p>O animal perdido foi enviado com sucesso!</p>
  <!-- Exibir animais perdidos semelhantes -->
  <h2>Animais Perdidos Semelhantes</h2>
  <div>
    
    <p>Nome: testeapi7</p>
    <p>Tipo: cachorro</p>
  </div>
  <!-- Exibir o novo animal perdido -->
  <a href="/enviar-animal-perdido/">Enviar outro animal perdido</a>
</body>
</html>

```

Figura 13. Retorno html da lista de animais semelhantes .

VI. CONCLUSÃO

O desenvolvimento desta API *auxilia* na busca por animais de estimação perdidos, usando técnicas de reconhecimento facial para dar esperança aos proprietários. Contudo, desafios como a complexidade do reconhecimento em diferentes raças, expressões faciais e condições de imagem foram encontrados. A precisão em imagens de baixa qualidade e em áreas com poucos dados também foi um obstáculo. Perspectivas futuras incluem melhorar os algoritmos de reconhecimento facial, ampliar a base de dados e usar técnicas avançadas de processamento de imagem. Como

Possíveis soluções, aumentar o conjunto de imagens para o treinamento do random forest e aprimorar ainda mais o código YOLO.

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] Breiman L. “Random Forest. In Journal of Machine Learning, Vol.45, pages 5-32”. Em: (2001). (Acesso em 01/12/2023).
- [11] Redmon J. Farhadi A. “YOLOv3: An Incremental Improvement”. Em: (2018). Disponível em <https://arxiv.org/abs/1804.02767>.
- [12] 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).
- [13] 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).
- [14] Django Documentation. “Django Project”. Em: (2024). Disponível em <https://docs.djangoproject.com/en/5.0/>.
- [15] SQLite Documentation. “SQLite”. Em: (2024). Disponível em <https://sqlite.org/docs.html>.
- [16] 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).
- [17] 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).
- [18] 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).
- [19] 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).
- [20] 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.scientiaplenu.org.br/sp/article/view/678>.
- [21] Barbosa A. et al. “Metodologia ágil: Feature-Driven Development”. Em: (2008). (Acesso em 07/11/2023).