



# **Deteccção e Análise de Padrões Comportamentais em Animais através de Visão Computacional**



Raphael Lanzoni Fracarolli e Marcos Vinicius de Lima Silva

# Objetivos do Projeto



**Treinamento com dados de gatos**



**Reconhecimento de padrões de expressões**



**Identificação e classificação de expressões**

# Objetivos do Projeto: interpretação do comportamento dos gatos

---

## Compreensão da expressão facial

A interface deve interpretar expressões faciais, como dentes expostos, posicionamento de orelhas, entre outros e correlacionar com suas respectivas emoções.

## Análise de posturas corporais

Movimentos como a cauda levantada ou posicionamento do corpo (alguns exemplos são: costas arqueadas e barriga para cima) devem ser analisados para compreender as emoções e estados de ânimo dos gatos.

## Integração de comportamento e expressão

A combinação da análise de expressão facial com o comportamento geral do gato ajudará a criar um perfil emocional mais preciso.

# Objetivos do Projeto: Desenvolvimento da interface do usuário



## Criação de uma interface simples

A interface deve ser simples e responsiva, garantindo uma rápida resposta através do upload de uma imagem do gato, sem grande complexidade para o usuário.



## Feedback das emoções

Implementar feedback das emoções dos gatos de maneira clara, com respostas que façam sentido para os usuários.



# Ferramentas Utilizadas: Tecnologia de Análise de Imagem



## Manipulação de Dados

- Pandas
- Numpy



## Processamento de imagens

- Pillow
- OpenCV (cv2)
- TensorFlow



## Modelagem e Deep Learning

- TensorFlow
- Keras
- Além de classes como: Sequential, Dense, Dropout, entre outras...



## Visualização

- Matplot
- Seaborn



## Divisão e análise de dados

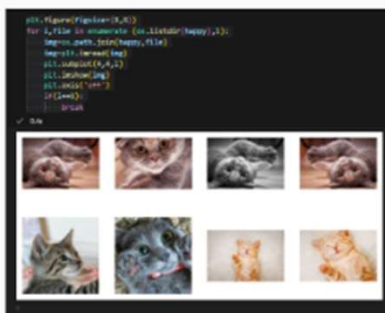
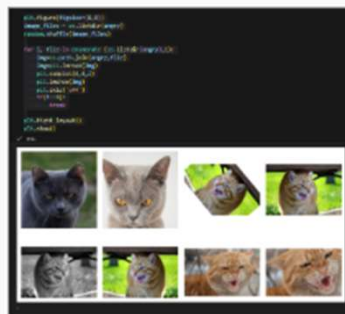
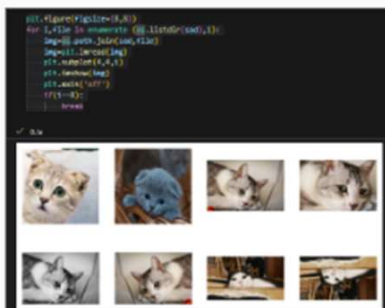
- Sklearn



# Compilação e treinamento

## Configuração do modelo

- Otimizador: Adam
- Função de perda: `categorical_crossentropy`
- Métrica: accuracy
- O treinamento é realizado em até 50 épocas com early stopping, interrompendo o treinamento caso a perda de validação não melhore por 10 épocas consecutivas.
- O desempenho do modelo no conjunto de teste dependerá da qualidade dos dados e das condições de treinamento



```
def plot_training_history(history):  
    acc = history.history['accuracy']  
    val_acc = history.history['val_accuracy']  
    loss = history.history['loss']  
    val_loss = history.history['val_loss']  
    epochs_range = range(len(acc))  
    plt.figure(figsize=(20, 5))  
    plt.subplot(1, 2, 1)  
    plt.plot(epochs_range, acc, label='Training Accuracy')  
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')  
    plt.legend(loc='lower right')  
    plt.title('Training and Validation Accuracy')  
    plt.subplot(1, 2, 2)  
    plt.plot(epochs_range, loss, label='Training Loss')  
    plt.plot(epochs_range, val_loss, label='Validation Loss')  
    plt.legend(loc='upper right')  
    plt.title('Training and Validation Loss')  
    plt.show()  
  
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])  
earlystop = EarlyStopping(monitor='val_loss',  
                           min_delta=0.001,  
                           patience=10,  
                           restore_best_weights=True)  
history = model.fit(  
    X_train, y_train_onehot,  
    epochs=50,  
    validation_data=(X_val, y_val_onehot),  
    callbacks=[earlystop])
```

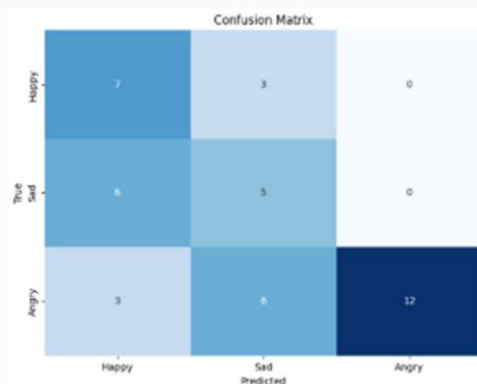
# Problemas X Soluções

A precisão do treinamento alcança valores próximos de 100%, mas a precisão da validação permanece baixa (60% ou menor).

Nas últimas epochs, a perda explode (val\_loss: 606.8511), enquanto a precisão de validação cai drasticamente. Isso pode ser causado por:

- Dados de validação inconsistentes.
- Superajuste extremo ao conjunto de treinamento.
- Erros na configuração do modelo ou dos dados.

```
Epoch 3/50 - accuracy: 0.9161 - loss: 1.4016 - val_accuracy: 0.5082 - val_loss: 1.7080
Epoch 4/50 - accuracy: 0.9110 - loss: 0.4317 - val_accuracy: 0.6276 - val_loss: 2.2010
Epoch 5/50 - accuracy: 0.9024 - loss: 0.3138 - val_accuracy: 0.5294 - val_loss: 3.1025
Epoch 6/50 - accuracy: 0.9363 - loss: 0.1828 - val_accuracy: 0.5294 - val_loss: 4.4558
Epoch 7/50 - accuracy: 0.9304 - loss: 0.2227 - val_accuracy: 0.5294 - val_loss: 3.3076
Epoch 8/50 - accuracy: 0.9387 - loss: 0.4637 - val_accuracy: 0.4671 - val_loss: 3.4234
Epoch 9/50 - accuracy: 0.9385 - loss: 0.1872 - val_accuracy: 0.5588 - val_loss: 3.4942
Epoch 10/50 - accuracy: 0.9401 - loss: 0.4487 - val_accuracy: 0.5588 - val_loss: 3.7203
Epoch 11/50 - accuracy: 0.9445 - loss: 0.4272 - val_accuracy: 0.5294 - val_loss: 3.9393
Epoch 12/50 - accuracy: 0.9445 - loss: 0.4248 - val_accuracy: 0.4671 - val_loss: 3.5808
Epoch 13/50 - accuracy: 0.9453 - loss: 0.4484 - val_accuracy: 0.4276 - val_loss: 3.3984
Epoch 14/50 - accuracy: 0.9475 - loss: 0.4225 - val_accuracy: 0.5588 - val_loss: 3.4234
Epoch 15/50 - accuracy: 0.9429 - loss: 0.2793 - val_accuracy: 0.2941 - val_loss: 606.8511
Epoch 16/50 - accuracy: 0.8138 - loss: 0.3138 - val_accuracy: 0.2941 - val_loss: 225.7788
Output is truncated. View on a scrollable dataset or open in a text editor. Adjust cell output settings.
```



## Algumas das Soluções Pensadas:

- Aumentar e melhorar o conjunto de dados
- Ajustar a taxa de aprendizado do otimizador. Ou até mesmo mudar para o SGD com momentum.
- Reavaliar os dados de validação
- Usar regularização como dropout

# Mudanças

Tivemos que realizar algumas mudanças para que o projeto funcionasse:

- Utilização do Database do Kaggle
- Criação de uma nova interface gráfica (Tkinter)

A nova interface conta com uma porcentagem de confiança e reestruturação visual

- Integração com uma API





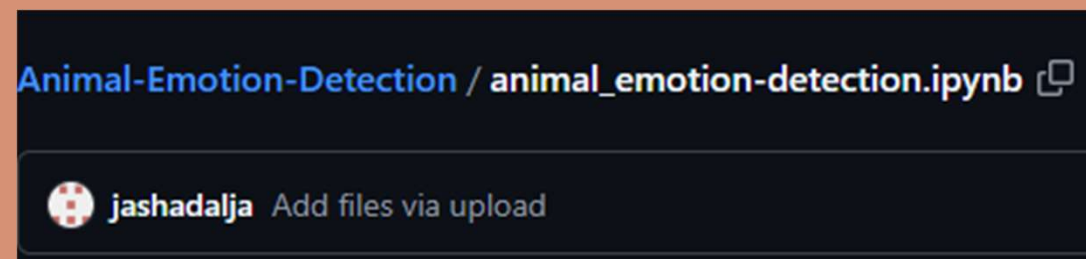
# Utilização do Database do Kaggle



- Embora o Database apresente imagens irrelevantes, sua implementação se mostrou essencial para o funcionamento do código e resolução do problema de overfitting no treinamento.

# Treinamento

Para o treinamento do modelo, foi utilizado um trabalho semelhante encontrado no GitHub



## Classificador de Emoções de Pets

Escolha uma imagem do seu Pet:

Carregue seu arquivo



Detectar Emoção

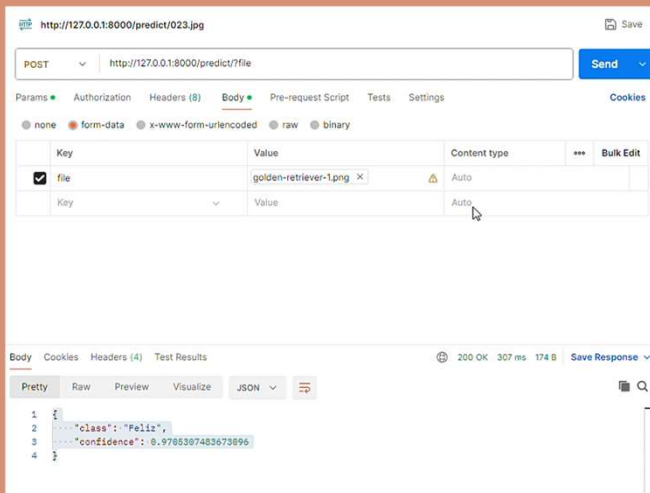
Emoção Detectada: Triste  
Confiança: 91.49%

## Interface Gráfica

- Foi realizada também uma reestruturação completa da interface gráfica desde a última apresentação. Alguns exemplos são:
- Modificações visuais
- Retorno de confiança da predição
- Modificações no código para integração com a API

# Implementação da API

Será mostrado no vídeo de demonstração, a execução do código integrado com a API



```
from fastapi import FastAPI, UploadFile, File
from fastapi.responses import JSONResponse
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
import os

# Cria a instância da API
app = FastAPI(title="API de Classificação de Emoções", description="API para classificar emoções em imagens", version="1.0")

# Carrega o modelo de Machine Learning
MODEL_PATH = "pets_detection.keras"
model = load_model(MODEL_PATH)

# Mapeia as classes do modelo
CLASS_NAMES = ["Happy", "Sad", "Angry"]

@app.get("/")
def root():
    """
    # Log da requisição recebida
    print(f"Recebendo request: {file.filename}, content_type: {file.content_type}")

    # Salva a imagem temporariamente
    temp_file = f"temp_{file.filename}"
    with open(temp_file, "wb") as f:
        f.write(await file.read())

    # Carrega e pré-processa a imagem
    image = load_img(temp_file, target_size=(224, 224)) # Redimensiona para 224x224
    image_array = img_to_array(image) / 255.0 # Normaliza os pixels

    # Faz a previsão
    predictions = model.predict(image_array)
    predicted_class = CLASS_NAMES[np.argmax(predictions)]
    confidence = float(np.max(predictions))

    # Remove o arquivo temporário
    os.remove(temp_file)

    # Retorna a previsão
    response = {"class": predicted_class, "confidence": confidence}
    print(f"Resposta gerada: {response}") # Log da resposta
    return JSONResponse(content=response)

except Exception as e:
    error_message = {"error": str(e)}
    print(f"Erro ao processar a requisição: {error_message}")
    return JSONResponse(content=error_message, status_code=500)
```

Body Cookies Headers (4) Test Results

200 OK 307 ms 174 B Save Response

```
1 {
2   "class": "Feliz",
3   "confidence": 0.9785397483673896
4 }
```