PUC Minas

Aplicação de Machine Learning no Reconhecimento Por Imagem de Plantas

Inteligência Artificial/Visão Computacional

Visão Computacional e Realidade Aumentada
2º/2024

Alunos: Leonardo Moraes
Tuanne Assenço

O Projeto

A proposta envolve a coleta de imagens de plantas provenientes de diversas fontes, como redes sociais, Google, plataformas como o Kaggle e outros datasets especializados. Essas imagens serão processadas e analisadas com base em características como tipos de plantas, cores, formatos das folhas, tipos de flores, entre outros atributos visuais. Utilizando técnicas de machine learning e inteligência artificial, o sistema será treinado para reconhecer diferentes espécies de plantas com precisão, garantindo uma identificação eficaz e confiável.

Objetivo

- **Reconhecimento de Plantas por Foto:** O projeto visa a criação de um modelo de reconhecimento que pode ser utilizado como uma *feature* que dispositivos IoT integrem, como por exemplo acesso a câmeras para capturar imagens de plantas. Essas fotos são então processadas pelo modelo criado neste projeto de Visão Computacional, permitindo um reconhecimento da espécie. Esta abordagem auxilia em integrações em que se beneficiem deste tipo de proposta.

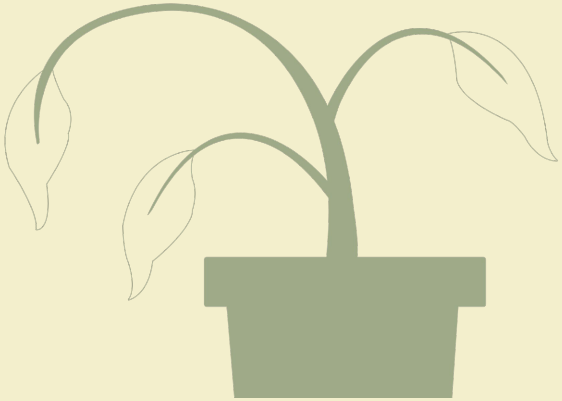
Coleta e Processamento de Dados

- **Source:** Definir e escolher uma fonte de dados com várias imagens de plantas para que o modelo seja treinado e testado.

- **Lendo e Pré-processando as Imagens:** Cada imagem é lida utilizando a biblioteca **OpenCV(‘cv2’)**, resultando em uma matriz numérica que o código pode manipular. As imagens são convertidas para o formato **RGB** e **redimensionadas** para 224x224 pixels. Este procedimento **garante dimensões consistentes para os dados de entrada**.

- **Armazenamento dos Dados:** As **imagens processadas, junto com seus rótulos**, são armazenadas como listas.

- **Salvar os Dados em um Arquivo Binário:** Por fim, os dados processados são salvos em um arquivo chamado 'data.pickle' usando o **módulo pickle**. Este arquivo pode ser facilmente carregado posteriormente para **treinamento do modelo** sem a necessidade de reprocessar os dados.



Desenvolvimento do Modelo de Reconhecimento

- **Carregamento e Divisão de Dados:** Os dados serão carregados e divididos em conjuntos de treinamento e teste, reservando 10% para teste.

- **Construção da Arquitetura do Modelo:** Através da utilização do TensorFlow, serão especificadas camadas de convolução e pooling para processar as imagens. A entrada do modelo é definida com dimensões fixas e levando em conta a acomodação de cores.

- **Compilação e Treino do Modelo:** O modelo será treinado e posteriormente salvo em arquivo, anulando a necessidade de reprocessamento.

Teste e Validação do Modelo

- **Testes:** O modelo treinado será carregado para prever imagens de teste. Seu desempenho será avaliado, exibindo **métricas** como **acurácia.** As previsões são usadas para criar um gráfico visual que compara as categorias reais e previstas das plantas no conjunto de teste, oferecendo uma representação rápida e intuitiva do desempenho do modelo.

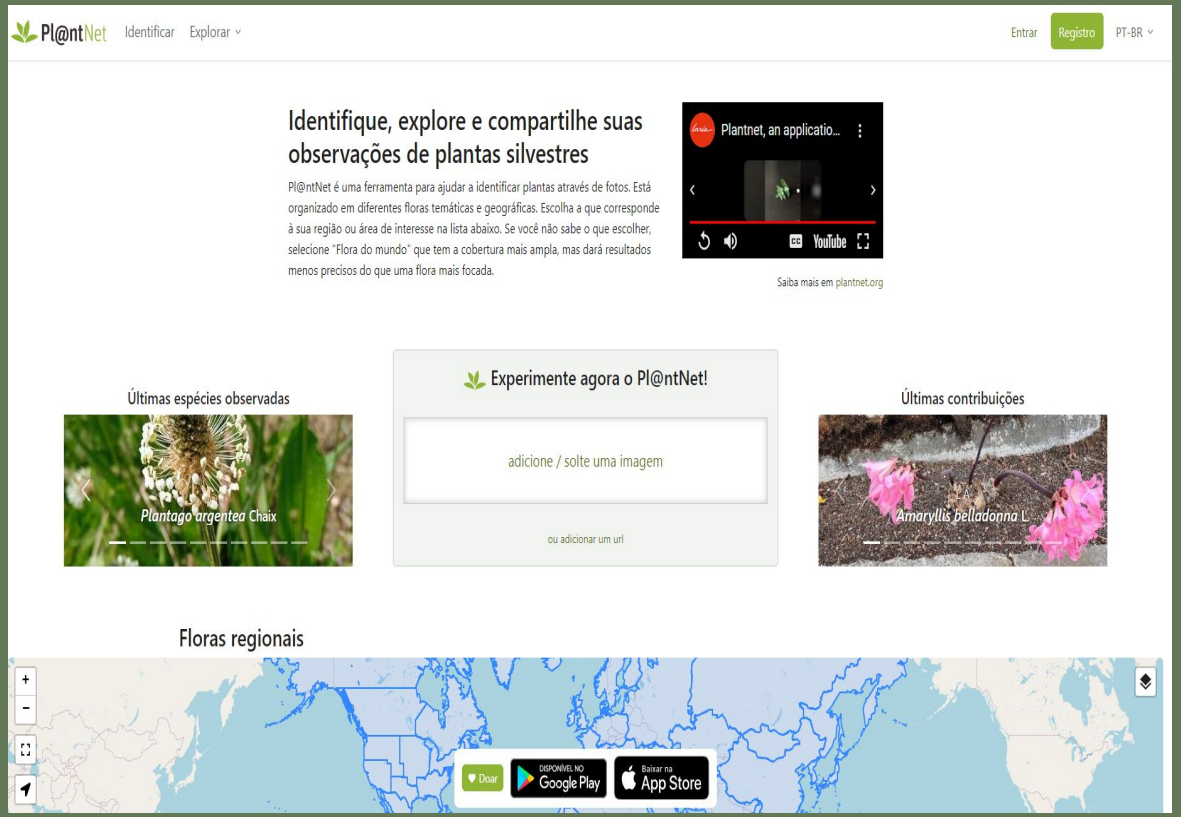
- **Análise de Aproveitamento:** Com a avaliação do conjunto de teste analisaremos a **precisão**, indicando a **eficácia** do modelo na identificação precisa das categorias de plantas definidas.

Estudo de Caso

Pl@ntnet

Estudo de caso: Pl@antnet

A Pl@ntNet é uma ferramenta de identificação de plantas baseada em uma aplicação móvel e web, desenvolvida para ajudar usuários a reconhecer e catalogar espécies vegetais. Utilizando inteligência artificial, ela permite identificar plantas através de fotos tiradas com o celular, analisando características como folhas, flores, frutos, cascas e outras partes da planta.



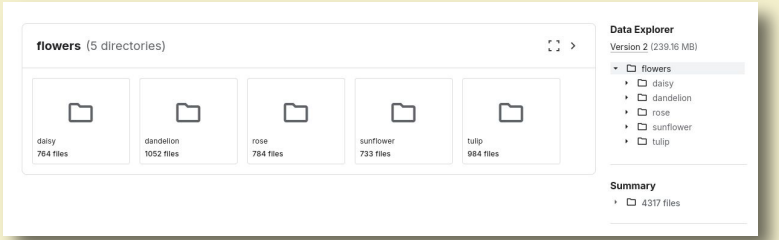
Segunda Etapa:
Desenvolvimento do modelo de reconhecimento

Dataset

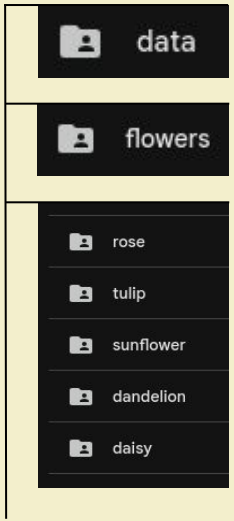
- **Base de dados:** Utilizando a plataforma **Kaggle** e por meio de uma busca manual dos participantes desse projeto na web, foi construído um **dataset** com dados coletados para o **treinamento do modelo**, contendo imagens de diversas espécies de flores.

- **Kaggle: Flowers Recognition**

  URL: https://www.kaggle.com/datasets/alxmamaev/flowers-recognition



- **Levantamento de base interno:**



Coleta e Processamento de Dados

- **Source:** Kaggle

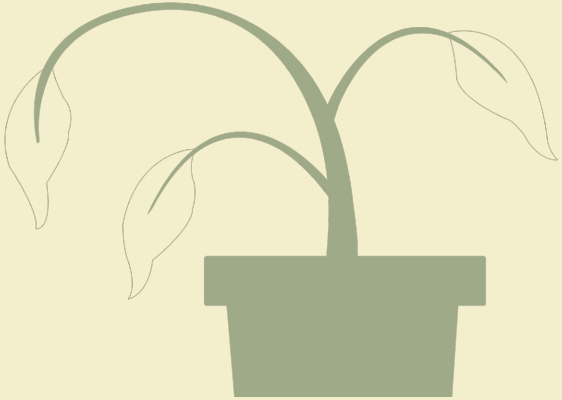
Nome do Arquivo	Tamanho do Arquivo
flowers.zip	236 MB

O dataset possui 5 pastas contendo entre 700 e 1052 arquivos **.jpg** de **5 espécies** de flores distintas: rosas, girassóis, tulipas, margaridas e dente-de-leão.

- **Lendo e Pré-processando as Imagens:** Cada imagem é lida utilizando a biblioteca **OpenCV('cv2')**, resultando em uma matriz numérica que o código pode manipular. As imagens são convertidas para o formato **RGB** e **redimensionadas** para 224x224 pixels. Este procedimento **garante dimensões consistentes para os dados de entrada**.

- **Armazenamento dos Dados:** As **imagens processadas, junto com seus rótulos**, são armazenadas como listas.

- **Salvar os Dados em um Arquivo Binário:** Por fim, os dados processados são salvos em um arquivo chamado 'data.pickle' usando o **módulo pickle**. Este arquivo pode ser facilmente carregado posteriormente para **treinamento do modelo** sem a necessidade de reprocessar os dados.



Dataset

- **Base de dados:** Utilizando a plataforma **Kaggle** e por meio de uma **busca manual dos participantes desse projeto na web**, foi construído um **dataset** com dados coletados para o **treinamento do modelo**, contendo imagens de diversas espécies de flores.

Roses



Tulip



Sunflower



Dandelion



Daisy



Desenvolvimento do Modelo de Reconhecimento

myclassifier.py

myclassifier.py

scripts > myclassifier.py > ...

```python
from utils import load_data
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

(feature, labels) = load_data()

x_train, x_test, y_train, y_test = train_test_split(feature, labels, test_size = 0.1)

categories = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

input_layer = tf.keras.layers.Input([224,224,3])

conv1 = tf.keras.layers.Conv2D(filters = 32, kernel_size=(5, 5), padding='Same',
                               activation='relu')(input_layer)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool1)
pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv2)

conv3 = tf.keras.layers.Conv2D(filters=96, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool2)
pool3 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv3)

conv4 = tf.keras.layers.Conv2D(filters=96, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool3)
pool4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2))(conv4)

flt1 = tf.keras.layers.Flatten()(pool4)

dn1 = tf.keras.layers.Dense(512, activation='relu')(flt1)
out = tf.keras.layers.Dense(5, activation='softmax')(dn1)

model = tf.keras.Model(input_layer, out)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=100, epochs=10)

model.save('my_model.h5')
```

- **Carregamento e Divisão de Dados:** Os dados são carregados e divididos em conjuntos de treinamento (x_train e y_train) e teste (x_test e y_test) usando a função train_test_split, reservando 10% para teste.

- **Construção da Arquitetura do Modelo:** Através da utilização do TensorFlow, são especificadas camadas de convolução e pooling para processar as imagens. A entrada do modelo é definida com dimensões 224x224x3 para acomodar as imagens coloridas.

- **Convoluções e Pooling (Camadas 1-4):** Quatro camadas de convolução seguidas por camadas de pooling são implementadas para extrair características das imagens.

Desenvolvimento do Modelo de Reconhecimento

myclassifier.py

myclassifier.py

scripts > myclassifier.py > ...

```python
from utils import load_data
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

(feature, labels) = load_data()

x_train, x_test, y_train, y_test = train_test_split(feature, labels, test_size = 0.1)

categories = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

input_layer = tf.keras.layers.Input([224,224,3])

conv1 = tf.keras.layers.Conv2D(filters = 32, kernel_size=(5, 5), padding='Same',
                               activation='relu')(input_layer)
pool1 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool1)
pool2 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv2)

conv3 = tf.keras.layers.Conv2D(filters=96, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool2)
pool3 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2))(conv3)

conv4 = tf.keras.layers.Conv2D(filters=96, kernel_size=(3, 3), padding='Same',
                               activation='relu')(pool3)
pool4 = tf.keras.layers.MaxPooling2D(pool_size=(2, 2), strides=(2,2))(conv4)

flt1 = tf.keras.layers.Flatten()(pool4)

dn1 = tf.keras.layers.Dense(512, activation='relu')(flt1)
out = tf.keras.layers.Dense(5, activation='softmax')(dn1)

model = tf.keras.Model(input_layer, out)

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=100, epochs=10)

model.save('my_model.h5')
```

- **Camada Densa e Camada de Saída:** A saída do pooling é achatada (Flatten) e conectada a uma camada densa com 512 neurônios. A camada densa alimenta uma camada de saída com 5 neurônios, correspondendo às categorias de plantas, utilizando a função de ativação softmax.

- **Compilação e Treino do Modelo:** O modelo é treinado utilizando os conjuntos de treinamento, com um tamanho de lote de 100 e 10 épocas. Após o treinamento, o modelo é salvo no arquivo 'my_model.h5' para uso futuro sem a necessidade de reexecutar o treinamento.

Desenvolvimento do Modelo de Reconhecimento

utils.py

utils.py

scripts > utils.py > load_data

```python
import os
import numpy as np
import matplotlib.pyplot as plt
import cv2
import  pickle

data_dir = '../project/data/flowers'

categories = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']

data = []

def make_data():
    for category in  categories:
        path = os.path.join(data_dir, category)
        label = categories.index(category)

        for img_name in os.listdir(path):
            image_path = os.path.join(path, img_name)
            image = cv2.imread(image_path)

            try:
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                image = cv2.resize(image, (224, 224))

                image = np.array(image, dtype=np.float32)

                data.append([image, label])

            except Exception as e:
                pass

    print(len(data))

    pik = open('data.pickle', 'wb')
    pickle.dump(data, pik)
    pik.close()

make_data()
```

- **Definição do diretório e categorias:** A função começa especificando o diretório onde as imagens estão localizadas e as categorias das flores, como 'daisy', 'dandelion', 'rose', 'sunflower', e 'tulip'.
- **Processamento das imagens:** Para cada categoria, a função percorre as imagens, converte as cores de BGR para RGB, redimensiona para 224x224 pixels e as converte em arrays NumPy do tipo float32. Cada imagem é associada ao seu rótulo (índice da categoria) e armazenada em uma lista.
- **Salvamento em arquivo:** Ao final, a função salva a lista data (contendo as imagens e seus rótulos) em um arquivo .pickle para uso posterior, útil para treinamento de modelos de aprendizado de máquina.

Desenvolvimento do Modelo de Reconhecimento

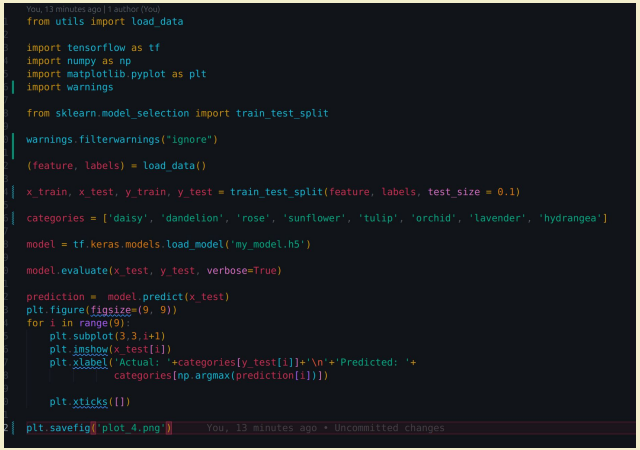
utils.py

```python
def load_data():
    pick = open('data.pickle', 'rb')
    data =  pickle.load(pick)
    pick.close()

    feature = []
    labels = []

    for img, label in data:
        feature.append(img)
        labels.append(label)

    feature = np.array(feature, dtype=np.float32)
    labels = np.array(labels)

    feature = feature/255.0

    return [feature, labels]
```

- **Carregamento do arquivo .pickle:** A função começa abrindo o arquivo data.pickle em modo leitura binária e carregando os dados contidos nele.
- **Separação e conversão dos dados:** Os dados são divididos em duas listas: uma para as imagens (feature) e outra para os rótulos (labels). As imagens são convertidas para arrays NumPy com tipo float32, e os rótulos são convertidos para arrays NumPy padrão.
- **Normalização e retorno dos dados:** As imagens são normalizadas dividindo os valores dos pixels por 255.0, garantindo que fiquem no intervalo [0, 1]. Em seguida, a função retorna as listas feature e labels como um único array contendo ambos.

Testando o modelo

detec.py



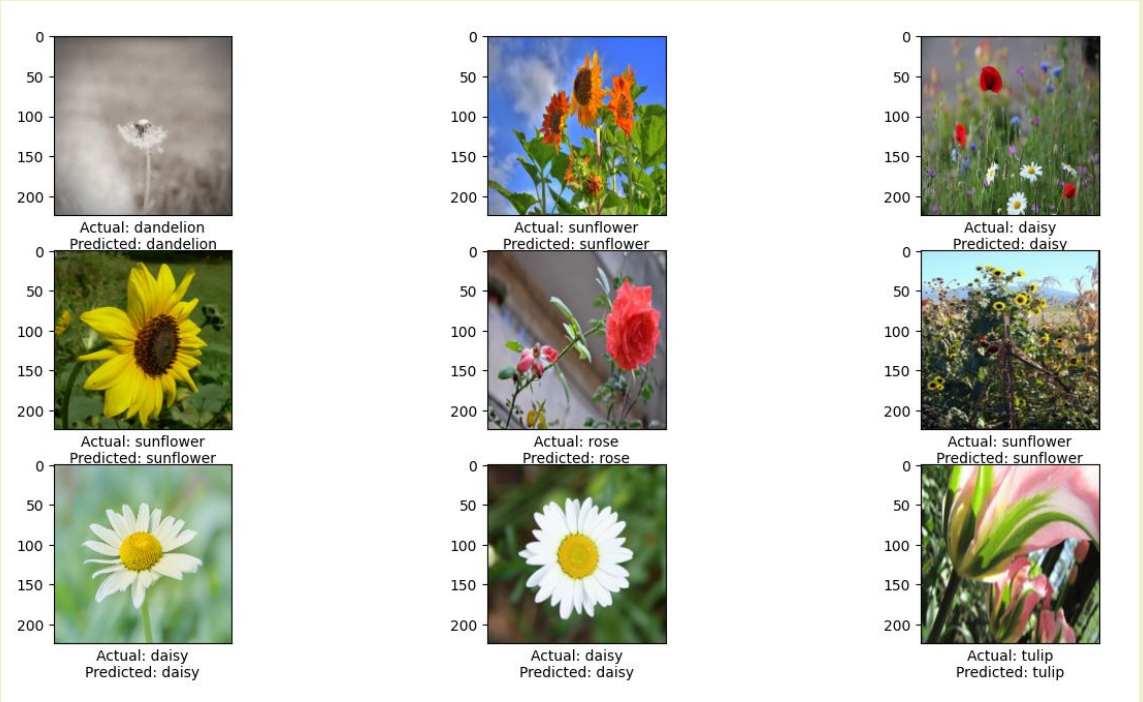
```
You, 13 minutes ago | 1 author (You)
from utils import load_data

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import warnings

from sklearn.model_selection import train_test_split

warnings.filterwarnings("ignore")

(feature, labels) = load_data()

x_train, x_test, y_train, y_test = train_test_split(feature, labels, test_size = 0.1)

categories = ['daisy', 'dandelion', 'rose', 'sunflower', 'tulip', 'orchid', 'lavender', 'hydrangea']

model = tf.keras.models.load_model('my_model.h5')

model.evaluate(x_test, y_test, verbose=True)

prediction =  model.predict(x_test)
plt.figure(figsize=(9, 9))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_test[i])
    plt.xlabel('Actual: '+categories[y_test[i]]+'\n'+'Predicted: '+
               categories[np.argmax(prediction[i])])

    plt.xticks([])

plt.savefig('plot_4.png')      You, 13 minutes ago • Uncommitted changes
```

- **Carregamento de Dados:** Utiliza load_data() para carregar features e labels.
- **Divisão de Dados:** Separa o teste com 10% das imagens presentes no dataset `train_test_split()`.
- **Categorias:** Classes predefinidas: daisy, dandelion, rose, sunflower, tulip, orchid, lavender, hydrangea.
- **Modelo Pré-treinado:** Carrega o modelo salvo my_model.h5 e avalia no conjunto de teste.
- **Predições e Visualização:**
  - Faz previsões no conjunto de teste.
  - Mostra 9 imagens com os rótulos reais e previstos.
  - Salva o gráfico.

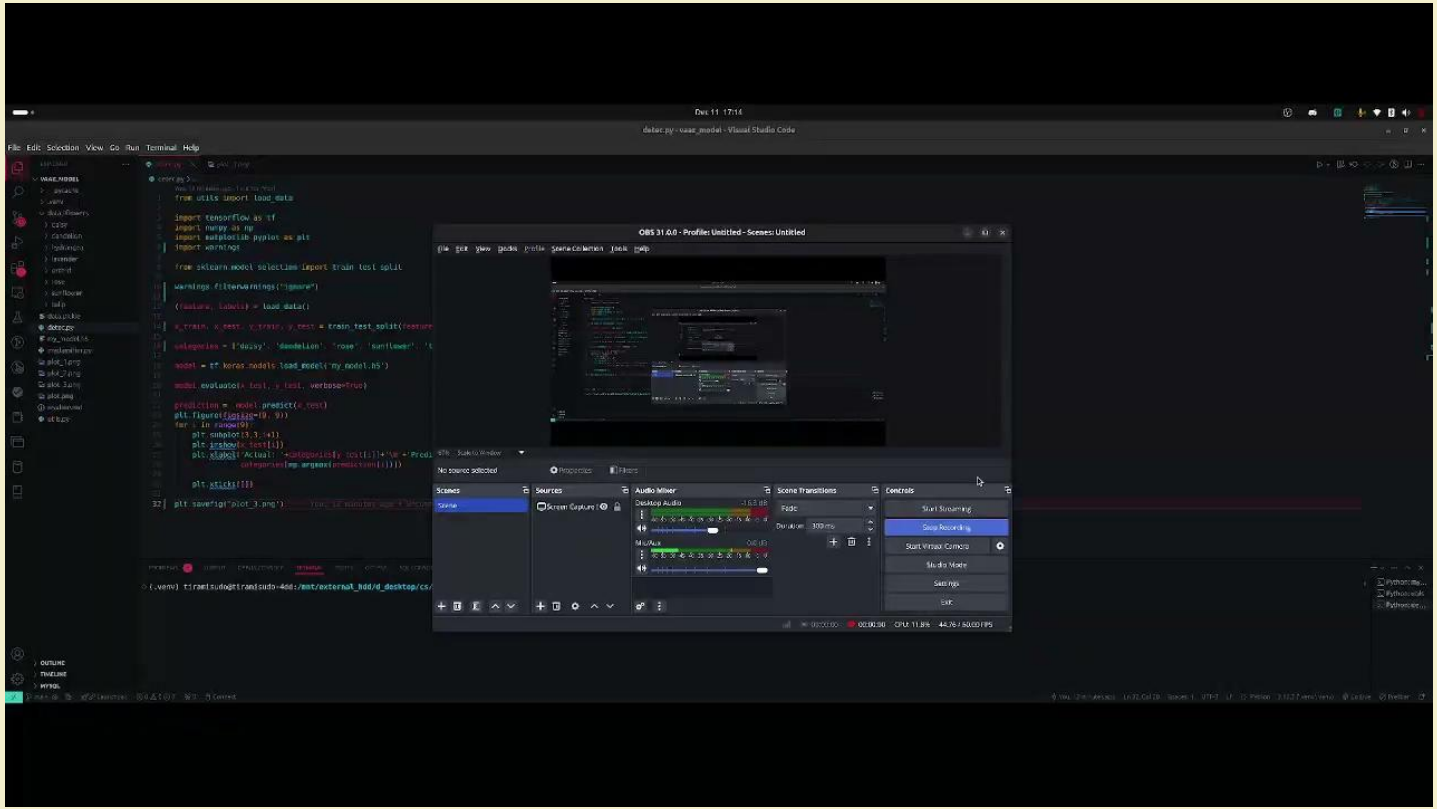
Teste e Validação do Modelo

- **Testes:** O modelo treinado é carregado para prever imagens de teste. Seu desempenho é avaliado, exibindo **métricas** como **acurácia.** As previsões são usadas para criar um gráfico visual que compara as categorias reais e previstas das plantas no conjunto de teste, oferecendo uma representação rápida e intuitiva do desempenho do modelo.

- **Aproveitamento:** A avaliação do conjunto de teste revelou uma precisão de **96.3%**, com **52 acertos em 54 testes realizados**, indicando a eficácia do modelo na identificação precisa das categorias de plantas definidas..



Demonstração da Aplicabilidade do Modelo



Bibliografia

MORAES, Leonardo, ASSENÇO, Tuanne. *Vaaz Visão*. Disponível em: https://github.com/developerleomoraes/vaaz_visao. Acesso em: 10 nov. 2024.

MAMAEV, Alexey. *Flowers Recognition*. Disponível em: https://www.kaggle.com/datasets/alxmamaev/flowers-recognition. Acesso em: 10 nov. 2024.

GOOGLE. *Pasta do Google Drive - Flowers Dataset*. Disponível em: https://drive.google.com/drive/folders/1SYiojb4YMEUzlXLH9fY_vS9THKKS-IVo?usp=sharing. Acesso em: 10 nov. 2024.

Obrigado!