

**PSI3472 – Prof. Hae Yong Kim – Escola Politécnica da USP**  
Guilherme Marques da Silva Nicolino – N°USP: 5946710  
**Exercício Programa** – Detecção de Objetos

## Introdução:

### Resumo do Funcionamento do Programa e Resultados Obtidos

O programa utiliza arquitetura **R-CNN** (Region CNN) com segmentação de objetos em imagens usando o **Meta SAM 2.1** para propor **ROIs** e a **VGG16** para classificar essas regiões. Treinado com um conjunto de dados de **80 mil** imagens aumentadas artificialmente, o modelo demonstrou alta precisão com **100%** de acertos com **MAE 0.8** e **IoU médio de 0.937** no conjunto de teste de 100 imagens. Mostrou generalização robusta, apesar de tempo de processamento elevado (30 minutos para classificação de 100 imagens com ~20 **ROIs** cada).

## Funcionamento do Programa

### 1. Proposta de ROIs (Regions of Interest):

- O segmentador **Meta SAM 2.1** é responsável por gerar **ROIs**. Ele divide a imagem em áreas que possivelmente contêm objetos, propondo múltiplas regiões baseadas em características visuais como bordas e contrastes, sendo possível extrair **bounding boxes** dessas **regiões propostas**.

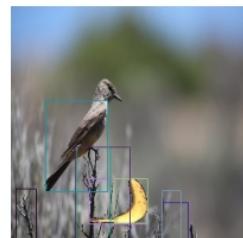
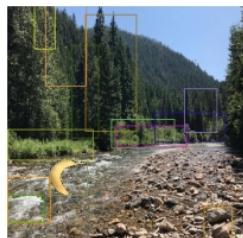
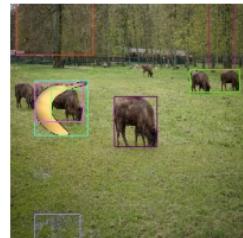
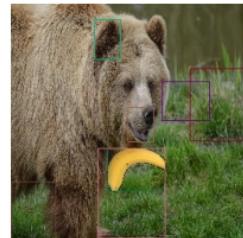
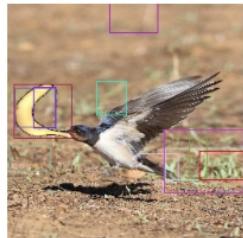
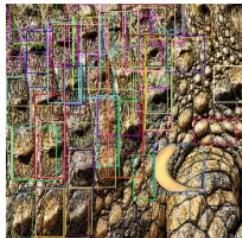
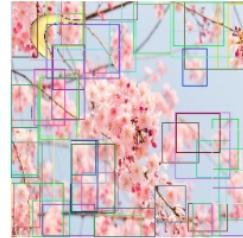
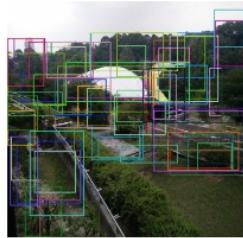
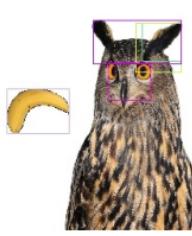
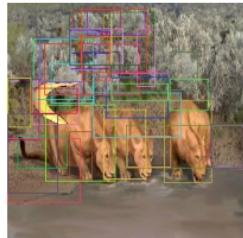
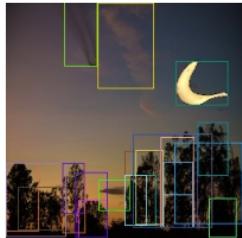


[facebookresearch/sam2: The repository provides code for running inference with the Meta Segment Anything Model 2 \(SAM 2\), links for downloading the trained model checkpoints, and example notebooks that show how to use the model. \(github.com\)](https://facebookresearch/sam2: The repository provides code for running inference with the Meta Segment Anything Model 2 (SAM 2), links for downloading the trained model checkpoints, and example notebooks that show how to use the model. (github.com))

Figure showing the original (left) and segmented (right) image.



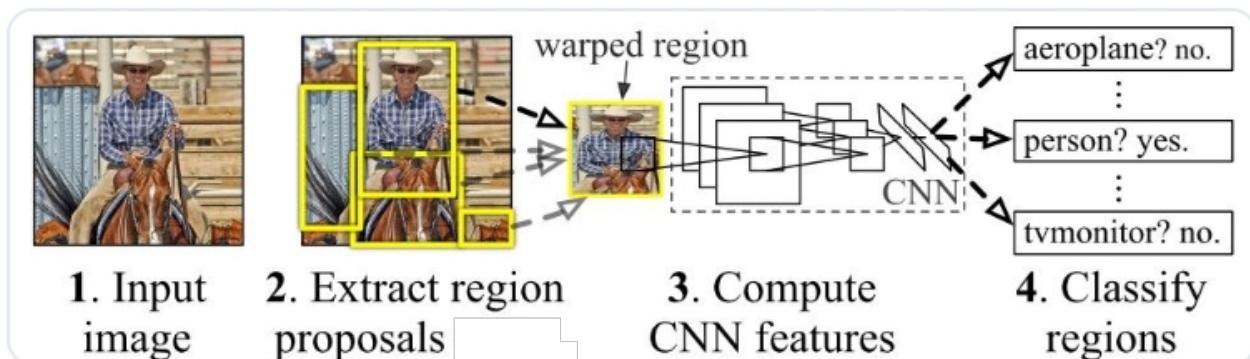
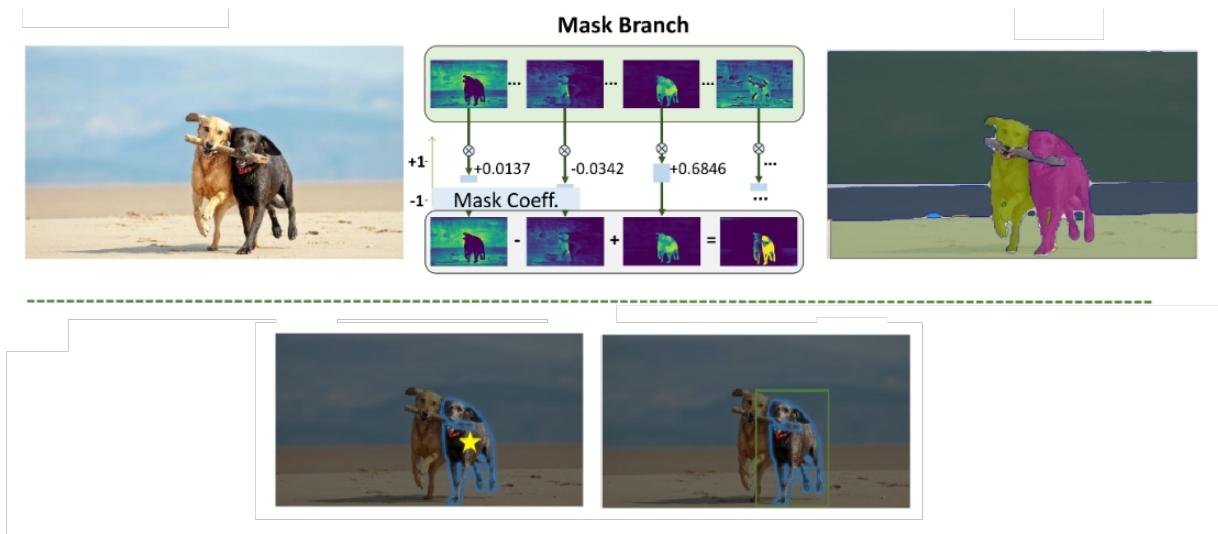
# ROIs Geradas pelo SAM 2.1



## 2. Classificação das ROIs:

- As **ROIs** propostas são alimentadas em uma **CNN (VGG16)** previamente treinada com **Transfer Learning** de **ImageNet**.
- O modelo foi treinado com um conjunto de dados contendo **40 mil** imagens do **objeto alvo** (aplicando técnicas de **Data Augmentation** para aumentar a diversidade e robustez do treinamento) e **40 mil** imagens de "**não objetos**" (imagens aleatórias que não possuem o objeto alvo).
- Para cada **ROI**, a CNN calcula a **probabilidade** dela conter o objeto desejado. Se essa probabilidade ultrapassar um determinado limiar, a **ROI** é considerada positiva para a presença do objeto.

# R-CNN: Segmentação e Classificação de ROIs



## Resultados Obtidos

### 3. Avaliação:

- O programa é capaz de detectar o objeto desejado com uma **alta precisão**, graças ao uso combinado de um **segmentador robusto** e de um **classificador** baseado em **Transfer Learning**.
- O uso de um grande conjunto de dados balanceado e técnicas de **Data Augmentation** ajudaram a minimizar overfitting e a melhorar a generalização do modelo em diferentes condições de luz, ângulos e tamanhos do objeto.

### 4. Métricas:

(Conjunto de teste de 100 imagens)

- **100%** de classificações corretas
- IoU Médio **0.937** (Intersection over Union)
- IoU Mínimo **0.873**
- Foi obtido um MAE de **0.80** (Mean Absolute Error)

# **Desenvolvimento: Ambiente de desenvolvimento utilizado e Reprodução dos Programas**

## **Compilação do Programa**

Para que o professor ou monitor possa compilar e executar o programa, eles podem seguir os seguintes passos:

### **1. Configuração do Ambiente:**

- Abrir o **Google Colab** e garantir que as bibliotecas mencionadas (**Keras 3.x** e **Meta SAM 2.1**) estejam instaladas. (Inclui instrução nos arquivos e referências)
- Incluir os arquivos do dataset de teste **banana-detection** (da biblioteca **gdown**), arquivo **vit\_h** de pesos do segmentador do **Meta SAM 2.1**, e o arquivo **model.keras** da rede treinada por mim (Link nas referências)

### **2. Utilização do Programa:**

- O programa classificador pode ser executado diretamente no **Google Colab** com as bibliotecas e arquivos mencionados.
- Alternativamente, é possível gerar um novo **model.keras** usando o notebook **banana\_transfer.ipynb**, que treina o modelo com os dados preparados por **data\_aug.ipynb**

### **3. Geração de Dados e Treinamento:**

- O notebook **data\_aug.ipynb** pré-salva imagens geradas com técnicas de **Data Augmentation** a partir do conjunto de treino, facilitando o uso e reutilização desses dados no treinamento da rede com o arquivo **banana\_transfer.ipynb**.

Dessa forma, qualquer pessoa com acesso ao **Google Colab** e aos **arquivos mencionados** pode facilmente compilar e executar o programa.

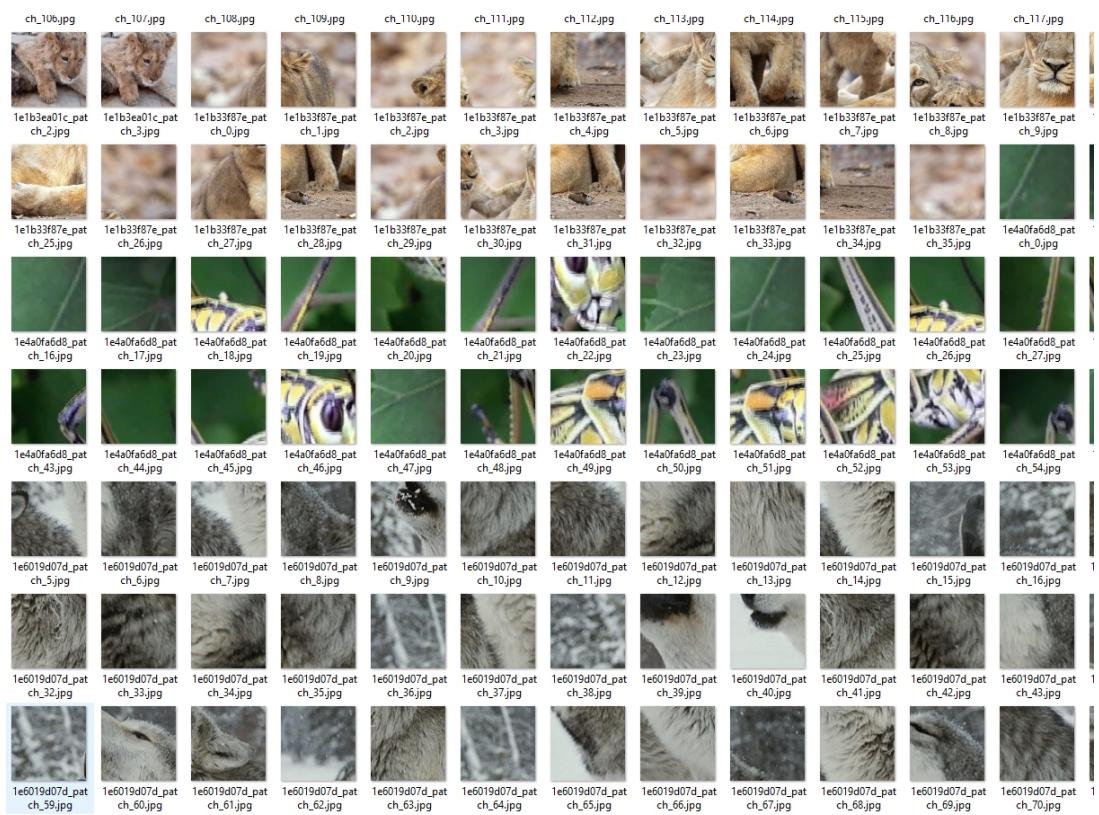
# Visualização de Parte dos Dados de Treino de Objetos e Não Objetos

Objetos: 40.000 itens



Não Objetos: 45.269 itens

- As imagens de não objetos foram geradas criando “patches” de diferentes tamanhos a partir de imagens grandes aleatórias de animais, alimentos, paisagens, etc.



# Modelo da Rede CNN

- A rede neural utiliza **Transfer Learning** de **VGG16** pré-treinada em **ImageNet** para extrair características visuais.
- A saída da **VGG16** passa por uma camada **Flatten**, que transforma as características extraídas em um vetor unidimensional, mantendo certa variância à tamanho (bananas maiores com **probabilidades** maiores), ao contrário de uma **GlobalAveragePooling2D** que tende a ser mais invariante à tamanho, comportamento observado em testes.

**Em seguida, a rede tem:**

- Uma camada densa com **64 neurônios**, com **regularização L2** para controle de overfitting, e ativação **ReLU**
- **Dropout de 50%** para generalização.
- Outras camadas densas com **128, 256 e 512 neurônios**, cada uma ativada por **ReLU** e seguidas por dropout, ajudando a rede a generalizar melhor e evitar overfitting.
- Essa estrutura de “**pirâmide**” nas camadas densas, com a ideia de “**expandir o mapa de atributos**” mostrou resultados melhores do que apenas uma camada densa.

```
input_shape = (224, 224, 3) # Assuming input shape
num_classes = 1 # For binary classification

base_model = tf.keras.applications.VGG16(
    weights='imagenet',
    include_top=False,
    input_shape=input_shape
)

base_model.trainable = False

inputs = tf.keras.Input(shape=input_shape)

x = base_model(inputs, training=False) # Ensure the base model is in inference mode

x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dense(64,
                        kernel_regularizer=l2(l2=0.05),
                        activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(128,
                        activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(256,
                        activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.Dense(512,
                        activation='relu')(x)
x = tf.keras.layers.Dropout(0.5)(x)

# Output layer for binary classification
outputs = tf.keras.layers.Dense(num_classes,
                                activation='sigmoid')(x)

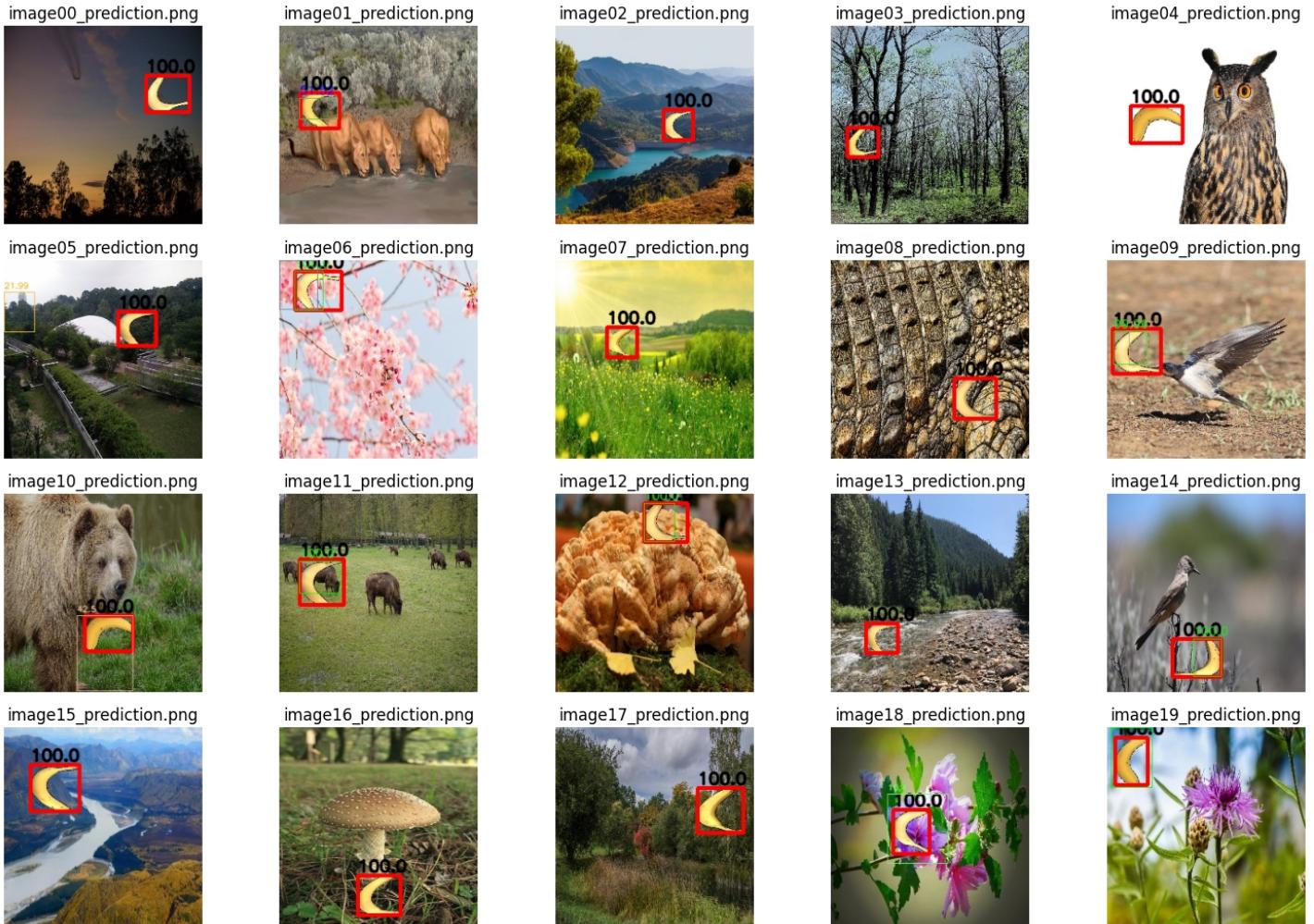
# Create the final model
model = tf.keras.models.Model(inputs, outputs)

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss='binary_focal_crossentropy', # Use binary crossentropy for binary
              metrics=['accuracy'])
```

# Resultados Detalhados

## Classificação:

- Exibição das Primeiras Imagens e **ROIs** com **probabilidade** maior do que **0.1**



- Detalhe para Classificação de Imagens com **ROIs** sobrepostos: A **CNN** com **Flatten** conseguiu atribuir corretamente **probabilidades** maiores à bananas maiores e mais centradas no **ROI** classificado (probabilidades exibidas estão arredondadas com 2 casas). Estas classificações sobrepostas poderiam ser eliminadas com uma camada **NMS** (Non-Max Supression) do **TensorFlow**, porém não foi necessário ou empregado no programa desenvolvido.

[tf.image.non\\_max\\_suppression | TensorFlow v2.16.1](#)



# Métricas Alcançadas

- Mean Absolute Error **0.80**
- ▼ Mean Absolute Error (MAE)

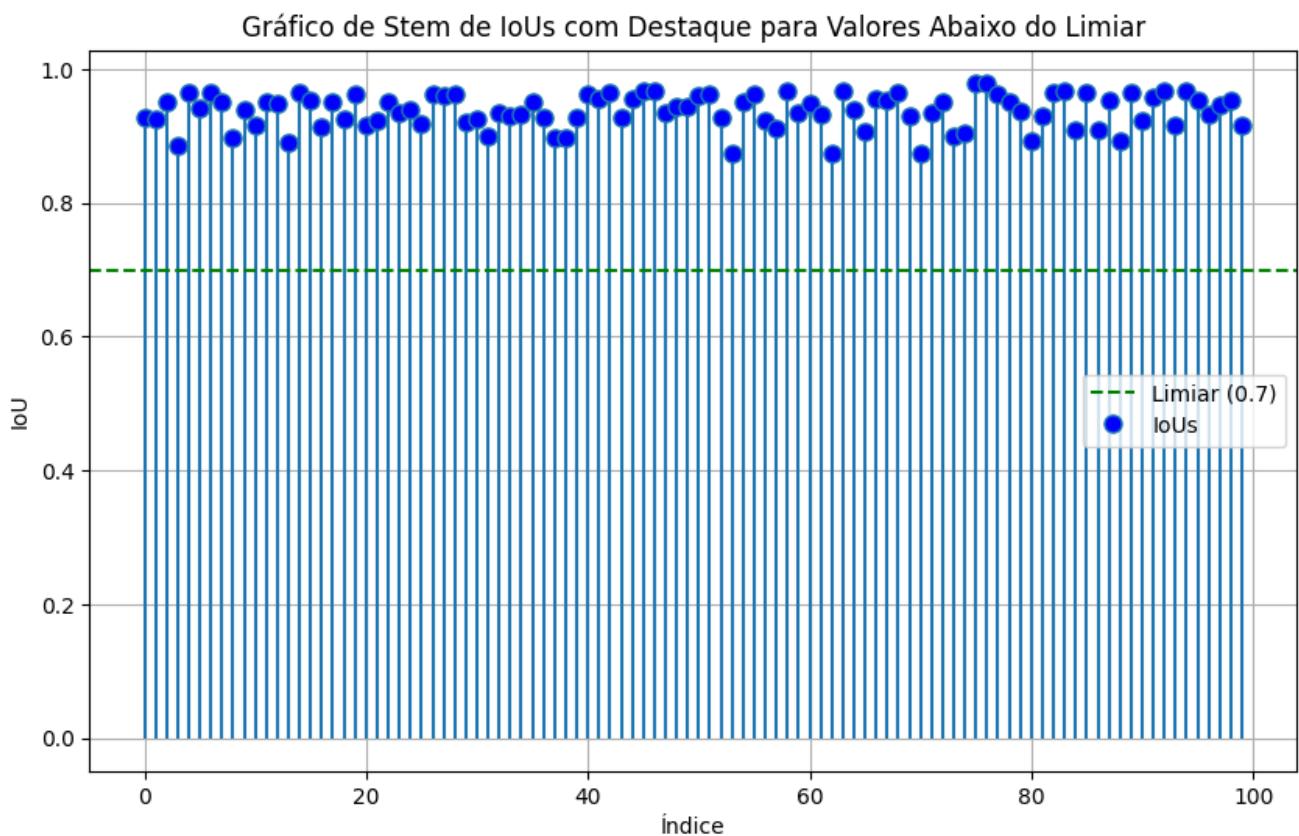
```
[ ] y_pred = np.array([x for x in np.array(p_test, dtype=object)[:,1]])
mae = mean_absolute_error(y_test[0:len(y_pred)], y_pred)
print(f"Mean Absolute Error (MAE): {mae:.4f}")
```

→ Mean Absolute Error (MAE): 0.8000

- IoU Médio e Mínimo

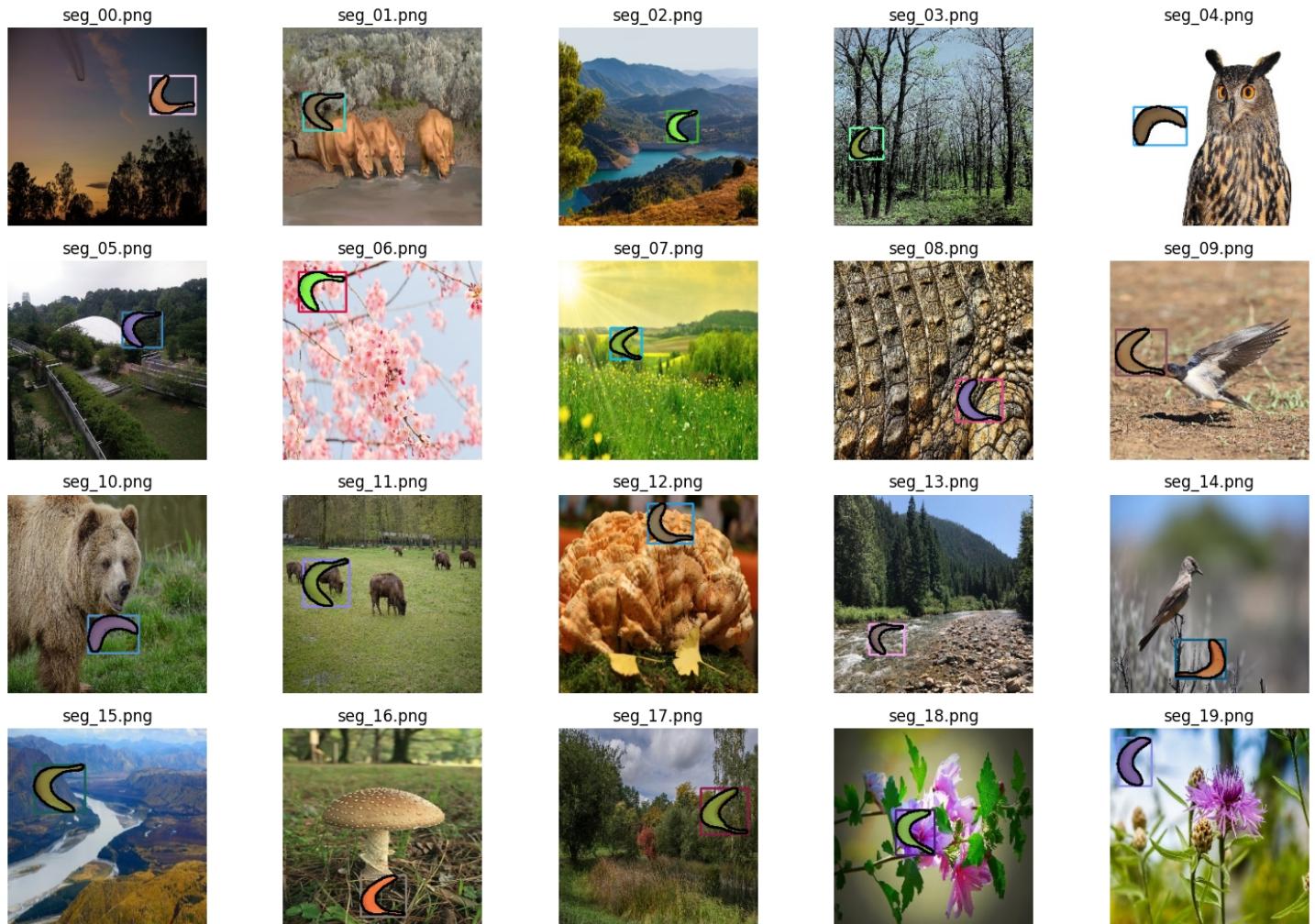
→ IoU média: 0.9377117447082414  
IoU mínima: 0.8735689397710303

- Gráfico de IoUs



# Segmentação dos Objetos Identificados

- Aproveitando que a **API do Meta SAM 2.1** também fornece **segmentação** a partir de **bounding boxes** de regiões, podemos adicionalmente desenhar a **segmentação semântica** dos **objetos detectados**, enaltecendo seu contorno e preenchimento.



## Tempo de Processamento

- Treino VGG16 – Batch 10 – Epoch 10 – 80 mil imagens

```
Epoch 7: val_loss improved from 0.12852 to 0.04390, saving model to model.keras
6822/6822 551s 80ms/step - accuracy: 0.9946 - loss: 0.1855 - val_accuracy: 0.9999 - val_loss: 0.0439 - learning_rate: 1.0000e-05
Epoch 8/10
6821/6822 0s 59ms/step - accuracy: 0.9974 - loss: 0.0444
Epoch 8: val_loss improved from 0.04390 to 0.02270, saving model to model.keras
6822/6822 548s 80ms/step - accuracy: 0.9974 - loss: 0.0444 - val_accuracy: 1.0000 - val_loss: 0.0227 - learning_rate: 1.0000e-05
Epoch 9/10
6821/6822 0s 59ms/step - accuracy: 0.9977 - loss: 0.0307
Epoch 9: val_loss improved from 0.02270 to 0.01976, saving model to model.keras
6822/6822 561s 80ms/step - accuracy: 0.9977 - loss: 0.0307 - val_accuracy: 0.9999 - val_loss: 0.0198 - learning_rate: 1.0000e-05
Epoch 10/10
6821/6822 0s 59ms/step - accuracy: 0.9981 - loss: 0.0250
Epoch 10: val_loss improved from 0.01976 to 0.01937, saving model to model.keras
6822/6822 560s 80ms/step - accuracy: 0.9981 - loss: 0.0250 - val_accuracy: 0.9999 - val_loss: 0.0194 - learning_rate: 1.0000e-05
```

5000 segundos (~1 hora e 30 minutos)

# Tempo de Processamento

- Segmentador Meta SAM 2.1 – 100 imagens

```
Imagen 97
Total de máscaras geradas: 102
Imagen 98
Total de máscaras geradas: 98
Imagen 99
Total de máscaras geradas: 176
Primeira etapa de aquisição de ROIs levou 748 segundos
Dados salvos em roi_data21_25_full.zip
```

748 segundos (~13 minutos)

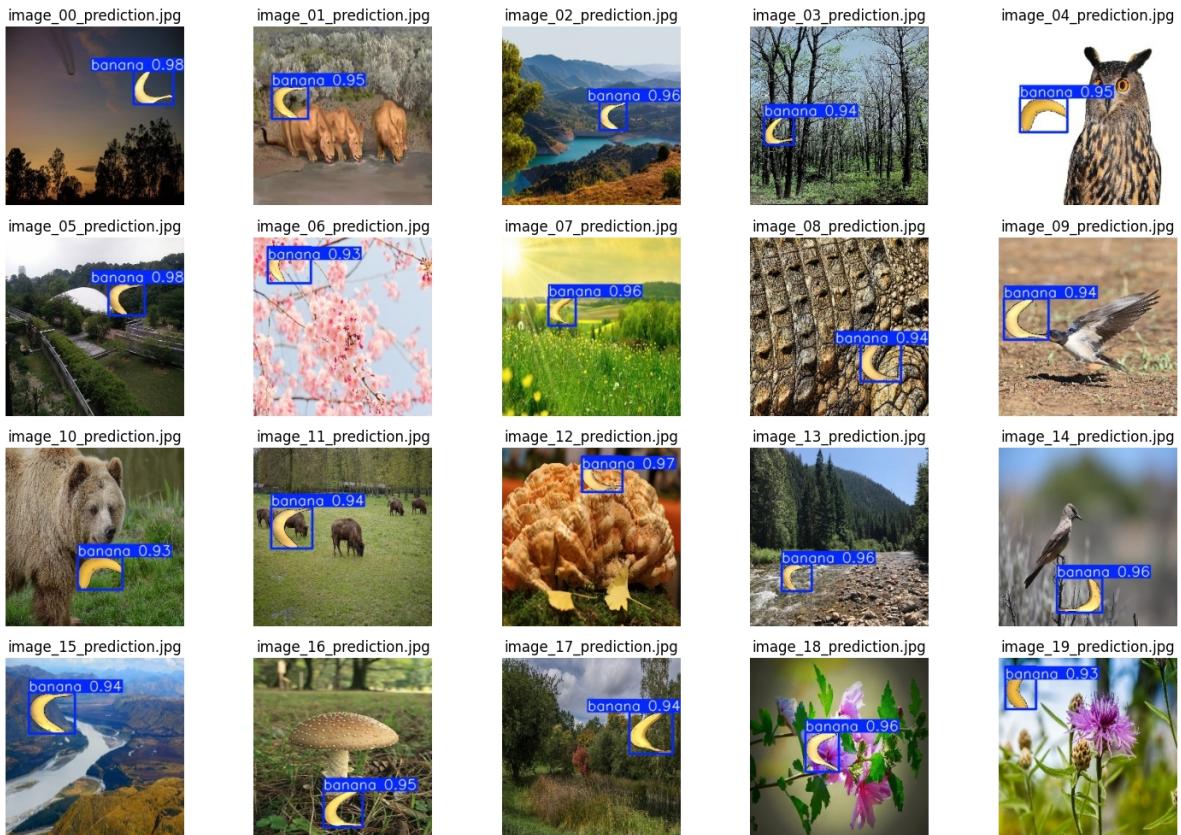
- VGG16 Inferência – 100 imagens (~20 ROIs por imagem)

```
Classificação dos ROIs levou 26 segundos
Classificando banana-detection/bananas_val/images/97.png ROIs: 33
Box Rede: (80, 59, 128, 122) - Prob. 100.0 - IoU: 0.9460
Box Original: [ 79 59 129 123]
Box Final: (80, 59, 128, 122)
Classificação dos ROIs levou 12 segundos
Classificando banana-detection/bananas_val/images/98.png ROIs: 35
Box Rede: (167, 42, 204, 88) - Prob. 99.9999771118164 - IoU: 0.9541
Box Original: [167 42 205 89]
Box Final: (167, 42, 204, 88)
Classificação dos ROIs levou 12 segundos
Classificando banana-detection/bananas_val/images/99.png ROIs: 40
Box Rede: (190, 76, 237, 116) - Prob. 100.0 - IoU: 0.9153
Box Original: [190 75 239 117]
Box Final: (190, 76, 237, 116)
Classificação dos ROIs levou 14 segundos
Tempo total: 1771 segundos
```

1771 segundos (~ 30 minutos)

# Comparação Adicional – Rede YOLOV8

- Adicionalmente foi feita a comparação com os resultados obtidos por rede YOLOV8 usando biblioteca da **Ultralytics**



➡ Média Absoluta do Erro das Bboxes (MAE): 0.5700000000000001

➡ IoU média: 0.9556177740341992  
IoU mínima: 0.8898305084745762

- **Tempo de Treino: 218 segundos (~4 minutos) – 2000 Imagens**

```
plot: True
results_dict: {'metrics/precision(B)': 0.9984199658241136, 'metrics/recall':
save_dir: WindowsPath('C:/banana-detection/yolo-banana3')
speed: {'preprocess': 0.23995399475097656, 'inference': 2.6408743858337402,
task: 'detect'
training: {'epochs': 20, 'seconds': 218.98014545440674}
```

- **Tempo de Inferência: 50.0 ms – 100 imagens (0.5ms por imagem)**

```
94: 256x256 1 banana, 0.5ms
95: 256x256 1 banana, 0.5ms
96: 256x256 1 banana, 0.5ms
97: 256x256 1 banana, 0.5ms
98: 256x256 1 banana, 0.5ms
99: 256x256 1 banana, 0.5ms
Speed: 0.2ms preprocess, 0.5ms inference, 1.2ms postprocess per image at shape (1, 3, 256, 256)
```

## Referências

- [Faster R-CNN Explained for Object Detection Tasks | DigitalOcean](#)
- [Image classification via fine-tuning with EfficientNet \(keras.io\)](#)
- [VGG16 and VGG19 \(keras.io\)](#)
- [facebookresearch/sam2: The repository provides code for running inference with the Meta Segment Anything Model 2 \(SAM 2\), links for downloading the trained model checkpoints, and example notebooks that show how to use the model. \(github.com\)](#)
- [How to Use the Segment Anything Model \(SAM\) \(roboflow.com\)](#)
- [notebooks/notebooks/how-to-segment-images-with-sam-2.ipynb at main · roboflow/notebooks \(github.com\)](#)
- [notebooks/notebooks/how-to-segment-anything-with-sam.ipynb at main · roboflow/notebooks \(github.com\)](#)
- [notebooks/notebooks/how-to-segment-anything-with-fast-sam.ipynb at main · roboflow/notebooks \(github.com\)](#)
- [Efficient Object Detection with YOLOV8 and KerasCV](#)

## Repositório do Código

- [Hefero/Keras-SAM-VGG16-Object-Detection: Object Detection Implementation using Meta SAM 2.1 for ROI proposal and Transfer Learning from VGG16 using Keras 3.x for Image Classification \(github.com\)](#)

## model.keras treinado para o programa banana-detection

- <https://www.mediafire.com/file/xpeufdr1rf8wm9l/model.keras/file>