Segmentação de Imagem

Tarefa 3 de MAC6914

Renato Lui Geh, NUSP: 8536030

1 Introdução

Para a Tarefa 3, usamos modelos pré-treinados [1] do framework DeepLab [4]. As arquiteturas do framework usadas foram Xception [2] e MobileNetv2 [5]. O framework também usa a busca de arquitetura descrita em [3].

Para a arquitetura Xception, foram testados dois diferentes *output strides*. Este parâmetro será denotado por O. Nos experimentos, usamos $O = 8$ e $O = 16$.

O dataset escolhido para segmentar foi o Cat vs Dogs [1].

2 Código

O script anexado junto à tarefa (`segment.py`) foi escrito em Python3 e segmenta as imagens do dataset Cats vs Dogs, salvando as imagens em um diretório `out`. Para replicar os resultados deste relatório, precisamos primeiro preparar o dataset e o framework:

[1] `https://www.kaggle.com/c/dogs-vs-cats/data`

```
git clone https://github.com/bonlime/keras-deeplab-v3-plus
unzip cats-vs-dogs.zip # Dataset
cd keras-deeplab-v3-plus
cp /path/do/script/segment.py .
```

Após feito isso, podemos rodar o script, que tem como argumentos:

```
Usage: segment.py backbone [param] [weights]
  backbone - 'xception' or 'mobilenetv2'
  param    - OS for xception
  weights  - 'random' or 'pascal_voc' (default)
```

Onde o argumento `backbone` é ou `xception` ou `mobilenetv2`. O argumento `param` é o parâmetro O se Xception. O último parâmetro `weights` indica se a rede deve usar os pesos pré-treinados ou se deve utilizar pesos aleatorizados.

Ambas as redes foram treinadas com imagens $(512, 512, 3)$. Antes de segmentar as imagens, redimensionamos as imagens para 512×512 e normalizamos as intensidades dos pixels. A captura dos rótulos é feita buscando o $\arg\max$ dos *logits* da última camada da rede.

3 Resultados

Os resultados foram geralmente bons. As imagens originais, juntamente com suas segmentações podem ser encontradas nos subdiretórios deste relatório. Cada subdiretório é nomeado a partir do nome da arquitetura seguido de seu parâmetro. Neste relatório apenas mostraremos os resultados ruins ou surpreendentes.

Apresentaremos alguns resultados extraídos a partir do código usado.

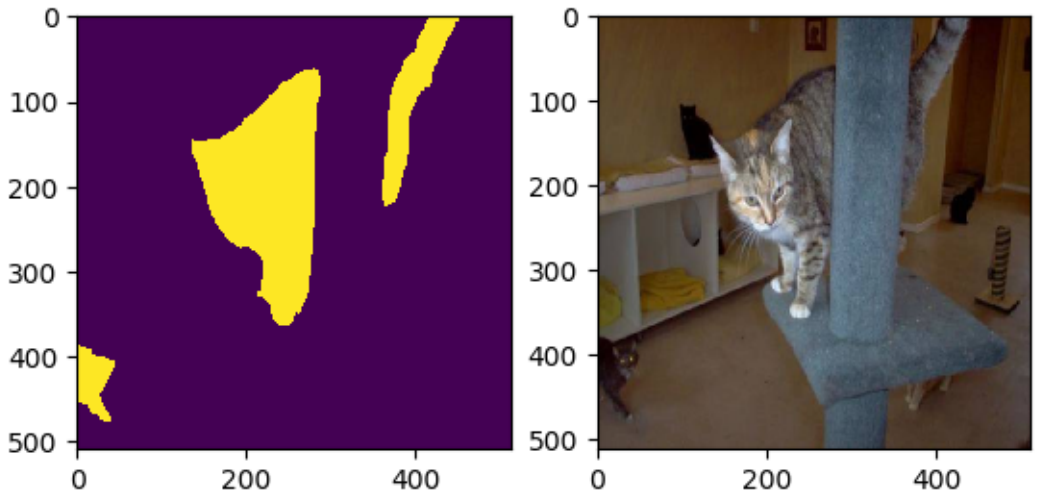

Figura 1: A imagem acima não conseguiu capturar todos os gatos, mas identificou corretamente o que está mais centralizado e o do canto inferior esquerdo. Apesar de estar parcialmente obstruído, a rede conseguiu identificar a parte posterior do gato.

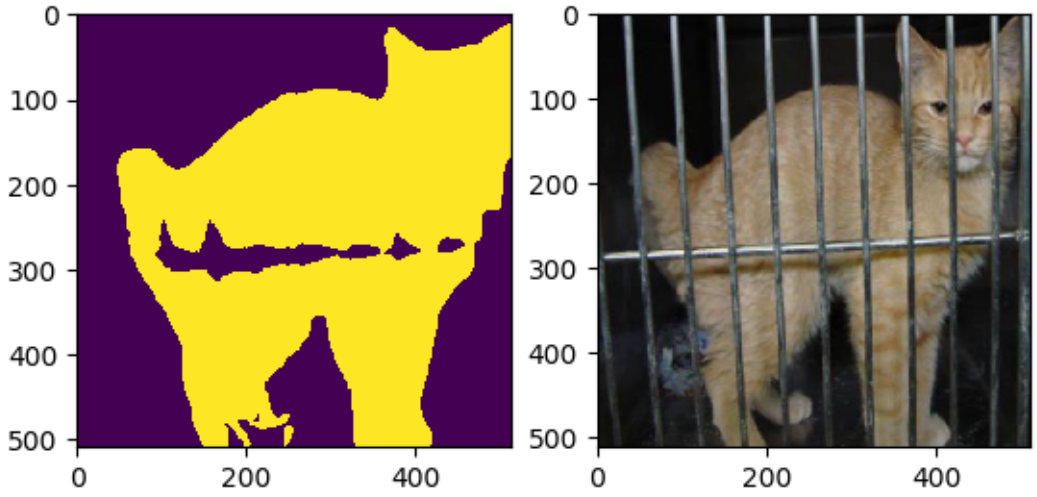

Figura 2: Quando há objetos finos obstruindo os animais, a rede teve dificuldade em distinguir o que é ou não gato ou cachorro.

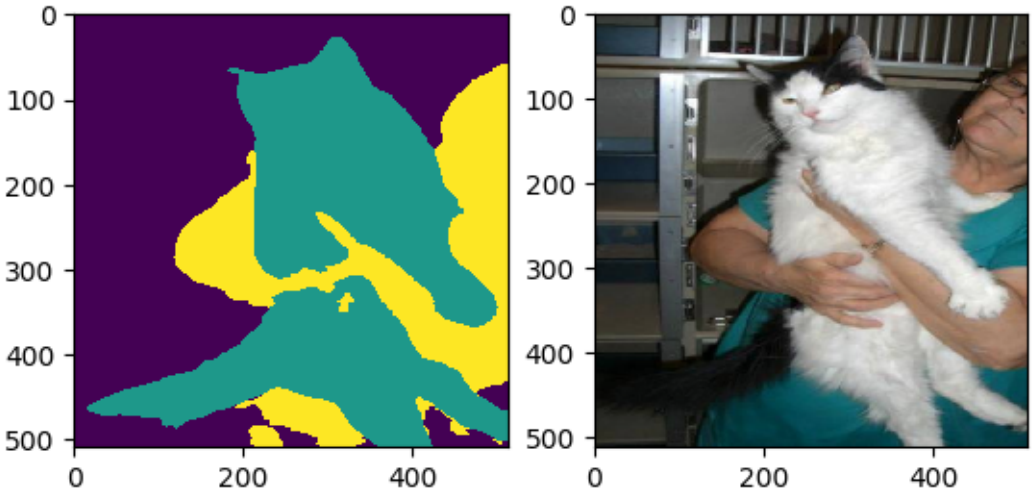

Figura 3: A presença de humanos foi capturada de forma correta na maior parte dos casos.

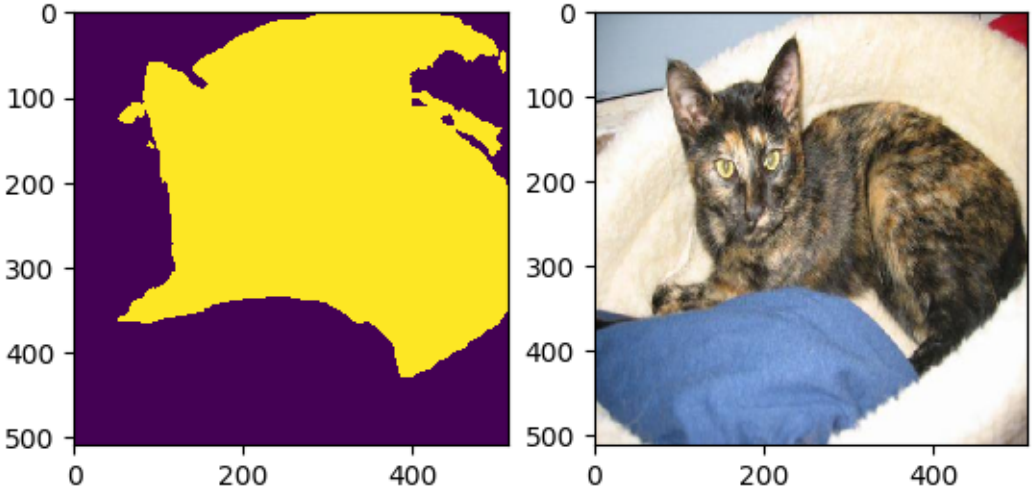

Figura 4: Em casos em que há objetos inanimados que contém pelugem ou características parecidas com a de animais, a rede confunde-se, classificando incorretamente como animal.

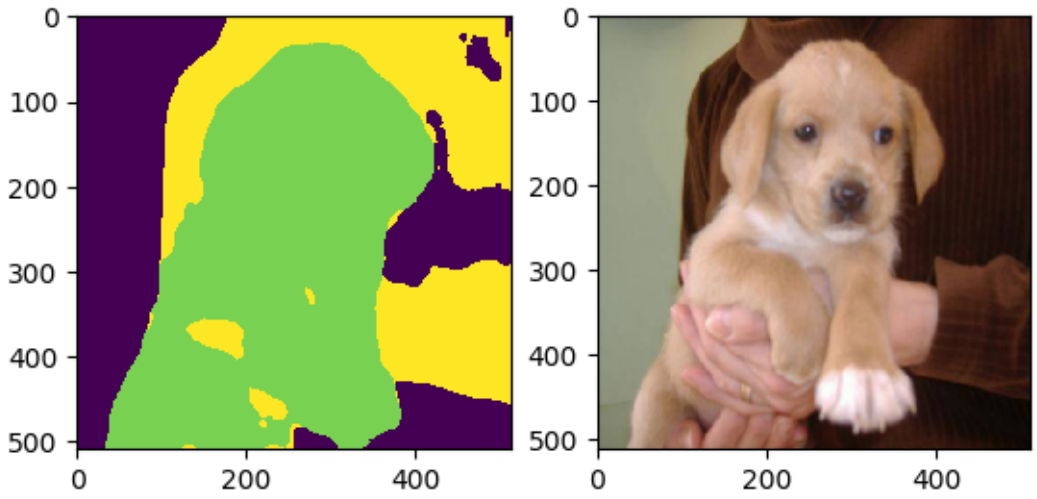

Figura 5: Quando regiões da imagem são muito escuras, a rede acha que trata-se de *background*, mesmo quando regiões adjacentes claramente não são.

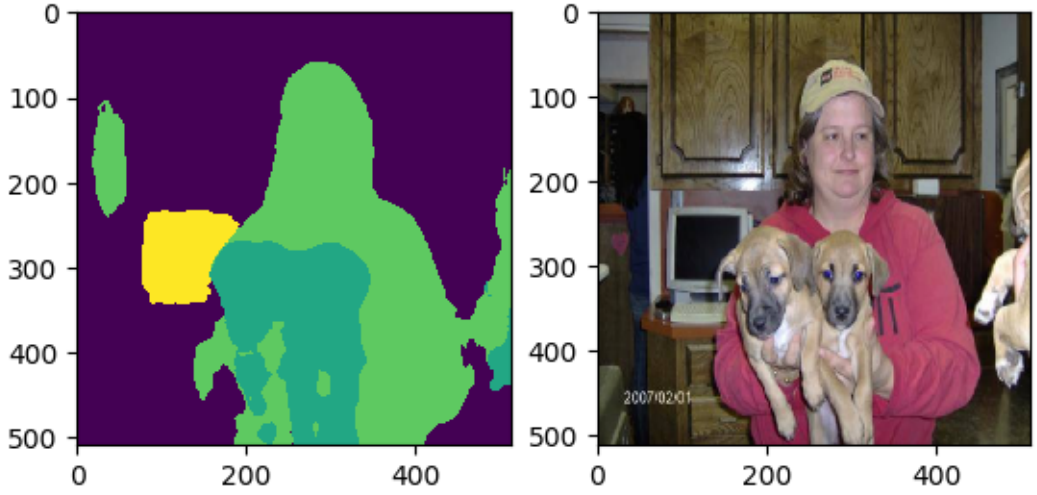

Figura 6: Como a rede foi pré-treinada em um dataset multi-classes, o modelo ainda consegue identificar objetos além de cachorros, gatos e humanos. No caso da figura acima, a rede identifica um computador.

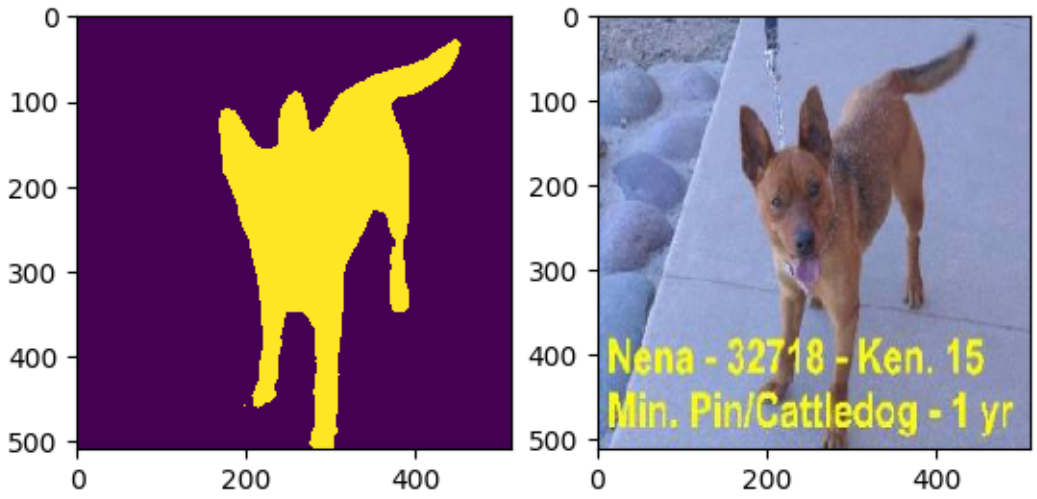

Figura 7: A presença de legendas ou letras não impactou na performance da rede.

4 Comparação entre arquiteturas

O modelo MobileNetv2 é claramente mais rápido de se aplicar segmentação, já que a rede foi construída com o intuito de ser executada em aparelhos de pouco poder de processamento. Já que a arquitetura Xception contém o dobro do número de parâmetros da MobileNetv2, é de se esperar que a primeira tenha melhor resultados do que a segunda. De fato a Xception proporcionou melhores segmentações.

Com relação à mudança de parâmetros, para a estrutura Xception, um parâmetro de $O = 8$ teve melhores resultados em geral do que $O = 16$, como era de se esperar. Não houve diferença significativa no tempo de segmentação com diferentes O. No entanto, supomos que a diferença foi significativa no treino.

Um experimento interessante foi, ao invés de usar pesos treinados, aleatorizar todos os filtros. Com isso foi possível observar como a estrutura é importante na identificação de objetos. Mesmo com pesos aleatórios, a estrutura Xception conseguiu identificar o contorno dos animais, enquanto que a estrutura MobileNetv2, que é significantemente menor e menos complexa teve resultados mais esperados de um modelo aleatório.

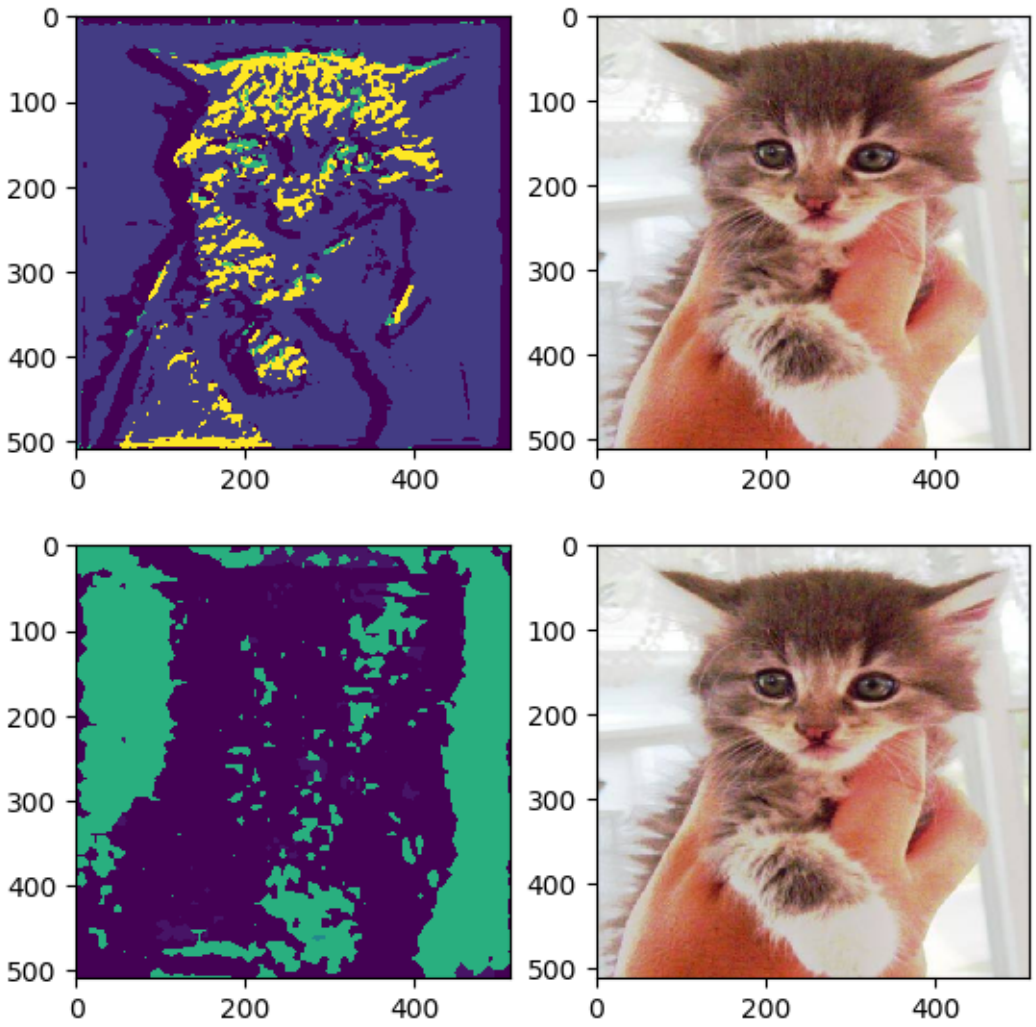

Figura 8: A imagem de cima mostra a segmentação feita em uma Xception com $O = 8$ aleatória, enquanto que a de baixo foi feita numa MobileNetv2 aleatória.

Em comparação com a Xception com $O = 16$, é possível ver que o *stride* tem grande impacto no nível de detalhe da segmentação.

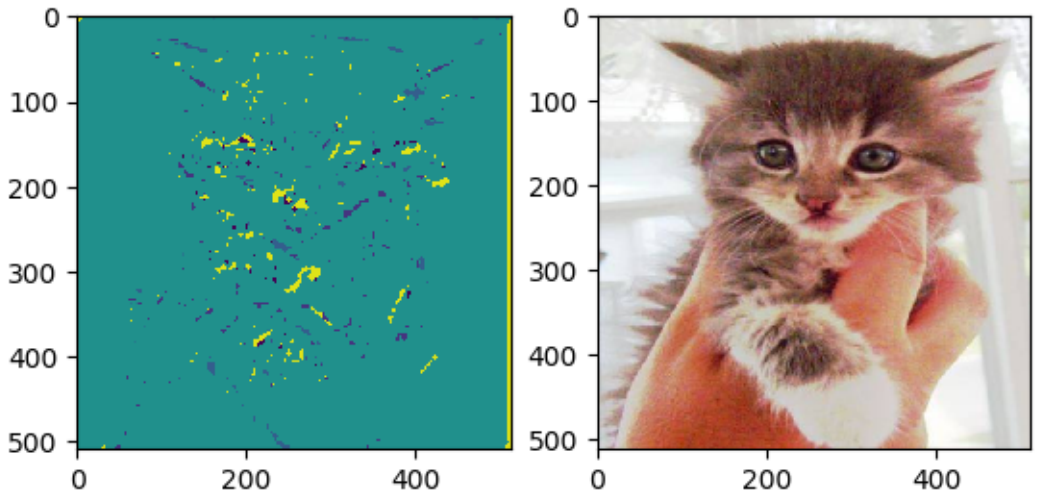

Figura 9: Xception aleatória com $O = 16$.

Referências

[1] bonlime. *Keras implementation of Deeplabv3+*. URL: `https://github.com/bonlime/keras-deeplab-v3-plus`.

[2] Liang-Chieh Chen et al. "Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation". Em: *ECCV*. 2018.

[3] Liang-Chieh Chen et al. "Searching for Efficient Multi-Scale Architectures for Dense Image Prediction". Em: *NIPS*. 2018.

[4] Google Inc. *DeepLab: Deep Labelling for Semantic Image Segmentation*. URL: `https://github.com/tensorflow/models/tree/master/research/deeplab`.

[5] Mark Sandler et al. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". Em: *CVPR*. 2018.