Documentação Notebook Detecção de Objetos em Vídeos com Darknet

Modelo Yolo v4

Etapa 1: Download do Darknet

O framework usada para a criação da arquitetura da rede neural profunda usada pelo Yolo é o Darknet, assim o primeiro passo é baixar o framework usando o repositório do GitHub https://github.com/AlexeyAB/darknet

Etapa 2: Compilando a Biblioteca

Uma vez que temos o framework darknet disponível passamos para ajustar o Makefile para habilitar OPENCV e GPU para darknet e então construir darknet usando os seguintes comandos:

```
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

Após isso testamos para ver se o GPU está habilitado:

```
import tensorflow as tf
device_name = tf.test.gpu_device_name()
print(device_name)
```

E então criamos o darknet para que seja possível usar o arquivo executável darknet para executar ou treinar detectores de objetos:

```
!make
```

Etapa 3: Carregando os Pesos do Modelo Pré-Treinado

Tendo o framework pronto para uso conectamos com o Google Drive e baixamos os pesos do modelo. O YOLOv4 já foi treinado no conjunto de dados COCO, que tem 80 classes que ele pode classificar, portanto pegamos esses pesos pré-treinados para que possamos executar o YOLOv4 nessas classes pré-treinadas e obter detecções.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
!wget
https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
```

Etapa 4: Carregando o vídeo

Para testar o modelo precisamos passar um vídeo em que será possível detectar as classes. Assim temos 2 formas de carregar o vídeo:

De uma URL

Para recuperar um vídeo de uma URL usamos o comando wget que é usado para recuperar conteúdos e arquivos de vários servidores na internet:

```
!wget
https://github.com/gabevr/yolo/raw/master/videos/video_teste02.mp4
```

Do Google Drive

Para recuperar um vídeo do Google Drive precisamos estar conectados ao Google Drive, mas como já fizemos isso na etapa 3 seguimos para o próximo passo, então copiamos o vídeo para o diretório em que o framework está:

```
!cp /content/gdrive/MyDrive/20210925_103150.mp4 .
```

Etapa 5: Realizando a Detecção em Vídeo

O modelo agora está construído e pronto para executar detecções usando YOLOv4 na nuvem. Para rodar a detecção o comando segue o seguinte padrão:

```
!./darknet detector demo <path to .data file> <path to config> <path to weights> -dont_show <path to vídeo> -i 0 -out_filename <output filename>
```

Significado das flags de customização:

- `-dont_show` - para não ter a imagem produzida após a execução do darknet
- `-i` - editar arquivo no lugar
- `-out_filename` - nome do arquivo de saída do modelo (i.e. arquivo rotulado)

Para o exemplo do notebook a execução fica:

```
!./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights
-dont_show 20210925_103150.mp4 -i 0 -out_filename resultado.avi
```

E após rodar a detecção rodamos o comando que mostra informações sobre o uso do disco rapidamente para mostrar informações sobre o arquivo de saída do modelo:

```
!du -h resultado.avi
```

Etapa 6: Visualizando o Resultado

Para visualizar o resultado só é preciso executar o arquivo de saída do modelo, para o nosso caso copiamos o arquivo para uma pasta no Google Drive para poder ter o resultado salvo:

```
!cp ./resultado.avi /content/gdrive/My\ Drive/Cursos\ -\
recursos/YOLO/videos/resultado1.avi
```

Etapa 7: Especificando um Threshold

Até essa etapa o modelo já está pronto, o objetivo agora é melhorar a qualidade do modelo. Então usamos a flag `-thresh`, ela é usada para adicionar um limite para confidências nas detecções. Apenas detecções com um nível de confiança acima do limite definido serão retornadas.

Essa flag é adicionada no mesmo comando de execução da detecção que rodamos na etapa 5, deixando o comando com o seguinte padrão:

```
!./darknet detector demo <path to .data file> <path to config> <path to
weights> -dont_show <path to vídeo> -i 0 -out_filename <output filename>
-thresh <threshold value>
```

Para o exemplo do notebook baixamos um novo vídeo para o teste do modelo e seguimos o mesmo fluxo:

```
!wget https://github.com/gabevr/yolo/raw/master/videos/video_rua01.mp4
```

Copiamos o vídeo para o diretório do framework:

```
!cp /content/gdrive/My\ Drive/Cursos\ -\
recursos/YOLO/videos/video_rua01.mp4 ./
```

E então executamos a detecção:

```
!./darknet detector demo cfg/coco.data cfg/yolov4.cfg yolov4.weights
-dont_show video_rua01.mp4 -out_filename resultado02.avi -thresh 0.3
```

E por fim copiamos arquivo de saída para a pasta do Google Drive:

```
!cp ./resultado02.avi /content/gdrive/My\ Drive/Cursos\ -\
recursos/YOLO/videos/resultado2.avi
```

Modelo Yolo v4 com Tf Lite

Etapa 1: Download Tf Lite

Como o objetivo do nosso projeto é aplicar o modelo para dispositivos móveis, precisamos usar o TensorFlow Lite, ele é a versão leve projetada especificamente para plataformas móveis e dispositivos incorporados.

Assim baixamos o framework usando o repositório do GitHub https://github.com/hunglc007/tensorflow-yolov4-tflite**.** O desempenho do treinamento do repositório ainda não foi totalmente reproduzido, então usamos o Darknet do Alex usado anteriormente para treinar seus próprios dados e, em seguida, converter os pesos em tflite.

Etapa 2: Carregando os Pesos do Modelo Pré-Treinado

Essa etapa segue o mesmo fluxo que a etapa 3 do Model Yolo v4 citada anteriormente.

Etapa 3: Carregando os requisitos

Para executar o modelo precisamos instalar os pré requisitos necessários para o modelo e para a GPU:

```
!pip install -r requirements.txt
!pip install -r requirements-gpu.txt
```

Etapa 4: Salvando os Pesos do Darknet no Tensorflow

Treinar o modelo tem 2 partes:

Treinar modelo

Para gerar o modelo precisamos carregar os pesos para o modelo e treiná-lo, o script `save_model.py` faz exatamente isso então executamos ele:

```
!python save_model.py --weights ./yolov4.weights --output
./checkpoints/yolov4-416 --input_size 416 --model yolov4 --framework
tflite
```

Significado das flags de customização:

- `--weights` - pesos do modelo pré treinado
- `--output` - arquivo de saída do modelo (i.e. arquivo rotulado)
- `--input_size` - tamanho da entrada
- `--model` - o modelo usado
- `--framework` - o framework usado

Converter para tflite

Com o modelo treinado precisamos converter o arquivo de saída do modelo para o formato TensorFlow Lite, para que possamos usar no dispositivo móvel:

```
# no quantization
!python convert_tflite.py --weights ./checkpoints/yolov4-416 --output
./checkpoints/yolov4-416.tflite

# yolov4 quantize float16
!python convert_tflite.py --weights ./checkpoints/yolov4-416 --output
./checkpoints/yolov4-416-fp16.tflite --quantize_mode float16

# yolov4 quantize int8
!python convert_tflite.py --weights ./checkpoints/yolov4-416 --output
./checkpoints/yolov4-416-int8.tflite --quantize_mode int8 --dataset
./coco_dataset/coco/val207.txt

# Run demo tflite model
!python detect.py --weights ./checkpoints/yolov4-416-fp16.tflite --size
416 --model yolov4 --image ./data/kite.jpg --framework tflite
```

No aprendizado profundo, os pesos são armazenados como números de ponto flutuante de 32 bits e podem ser reduzidos para inteiros de 8 bits, o que eventualmente significa uma redução do tamanho do modelo, assim quantizamos para redução do tamanho.

Etapa 5: Download do Modelo

Assim só é preciso baixar os arquivos gerados pelo script `detect.py`, para isso geramos uma função que recebe um diretório e baixa os arquivos do mesmo:

```
from google.colab import files

def downloadModels( models_dir_path):
   for file_name in os.listdir( models_dir_path):
       file_path = models_dir_path+file_name
       files.download( file_path)
```

E então passamos o diretório com os arquivos de saída do modelo:

```
downloadModels( "/content/tensorflow-yolov4-tflite/checkpoints/")
```

Etapa 6: Copiar Modelo para Pasta

Após baixar o modelo copiamos para uma pasta no Google Drive para poder ter o resultado salvo:

```
from google.colab import drive
drive.mount('/content/drive')

def copyToFolderRecursively( source, dest):
   #cuidado
   !cp -r $source $dest

dest_path = "/content/drive/MyDrive"
source_path = "/content/tensorflow-yolov4-tflite/checkpoints"
copyToFolderRecursively( source_path, dest_path)
```