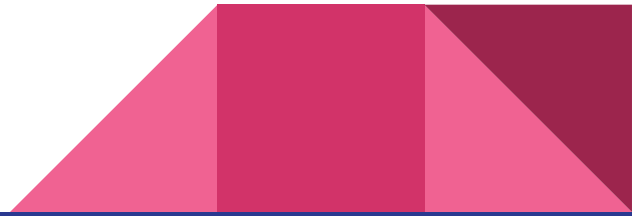
Image recognition using YOLOv5

Por Equipe dos Federated

Tiago Guerreiro - (201906840007)
João Fonseca - (201806840035)
Lucas Prazeres - (201906840005)
Devid Barboza - (201906840018)

Sumário

1. Instalação
2. Geração do dataset
3. Configuração da rede
4. Treino
5. Pós-processamento
6. Resultados



1. Instalação

Para preparar o ambiente do trabalho proposto, a equipe seguiu a documentação presente no GitHub da Yolo v5 e no site do CUDA, que inclui:

1) Criação de um ambiente virtual com virtualenv;
2) Clonar o repositório e instalar as dependências;
3) Clonar o repositório do Desafio Petrobras e instalar as dependências;

Obs: Para o CUDA, deve-se atualizar a chave GPG.

2. Geração do Dataset

O dataset da atividade foi gerado a partir do código customizado multithread_geradados.py, para utilizar várias threads da CPU no processo. Ele consiste em:

- Geração do número de imagens e seus respectivos labels;
- Determinação da resolução e tamanho das imagens;
- Determinação das bounding boxes dos objetos a serem detectados;

3. Configuração da rede

Nesta etapa ocorre as alterações nos arquivos que serão usados na fase de teste, originalmente fornecidos no próprio GitHub da Yolo V5.

- Formatação do dataset, definindo uma pasta ‘images’, responsável por armazenar as imagens geradas, e ‘labels’, responsável por guardar as labels das imagens (classe x y w h), ambas contidas na pasta ‘datasets/petrobras’.
- Alteração do arquivo “coco128.yaml”, mudando a informação do número de classes de 80 para 36 e a lista de nomes das classes para as do desafio fornecido. O nome do arquivo final foi alterado para “IC_final.yaml”.
- Alteração do arquivo “yolov5s.yaml”, definindo o número de classes no modelo que será usado na etapa de treino (arquivo de configuração do modelo). O nome do arquivo final foi alterado para “IC_final_yolov5s.yaml”.

4. Treino da rede

Para treinar a rede, basta passar como parâmetros do arquivo train.py, localizado na pasta raiz da Yolov5, os arquivos YAML de configuração do dataset e da rede. O resultado final seria algo como isso:

python train.py --data IC_final.yaml --cfg IC_final_yolov5s.yaml --epochs 6 --weights ‘ ’ --batch-size 16 --img 640

Ao passar o parâmetro weights vazio, treinamos nossos pesos do início (sem pesos pré-treinados).

5. Pós-processamento

O pós-processamento vai modelar a saída da rede para o resultado desejado na atividade. Basicamente, carrega-se o arquivo de pesos treinados (**best.pt**) e passa-se como entrada uma lista de caminhos das imagens, que são processados pelo modelo. As saídas da rede é salva na variável result, que é iterada para fornecer a saída desejada, salva no arquivo **output.txt**.

```python
import torch
import glob

model = torch.hub.load('ultralytics/yolov5', 'custom', path='./best.pt')

image_path = './test_images/'
imgs = glob.glob(image_path + "*.png")

result = model(image_path,size=640)

img_df = result.pandas().xyxy

linhas_txt = []
for idx,df in enumerate(img_df):
  if 'display' in df.values:
    inf_esq = df[df['name'].str.match(r'sinal_apenas|sinal_e_um|sinal_zero|sinal_mais')==True]
    if inf_esq['name'].values[0] == 'sinal_apenas':
      inf_esq='-'
    elif inf_esq['name'].values[0] == 'sinal_e_um':
      inf_esq='-1'
    elif inf_esq['name'].values[0] == 'sinal_zero':
      inf_esq='+'
    elif inf_esq['name'].values[0] == 'sinal_mais':
      inf_esq='+1'
    sup_esq=df[df['name'].str.match(r'sup_esq.*')==True].values[0][-1].split('_')[-1]
    sup_dir=df[df['name'].str.match(r'sup_dir.*')==True].values[0][-1].split('_')[-1]
    inf_dir=df[df['name'].str.match(r'inf_dir.*')==True].values[0][-1].split('_')[-1]

    linhas_txt.append(f"image_{idx} {imgs[idx].split('/')[-1]}\n{sup_esq}{sup_dir}\n{inf_esq}{inf_dir}\n")
  else:
    linhas_txt.append(f"image_{idx} {imgs[idx].split('/')[-1]}\n-9999\n-9999\n")

with open('output.txt', 'w+') as file:
  file.writelines(linhas_txt)
```

6. Resultados