Do Zero ao Yolo

O presente documento tem a finalidade de instruir e detalhar sobre o funcionamento da rede neural Yolo com base na nossa participação e experiência na competição do site ” Zindi África”, podemos verificar mais detalhes da mesma neste link, podendo assim o usuário generalizar o conhecimento compreendido e resolver projetos futuros. Para este tutorial utilizaremos o google colab para criação dos códigos e a linguagem python 3+.

1. Introdução ao YOLO (O que é?)

YOLO, sigla para "You Only Look Once" , é um algoritmo extremamente preciso e veloz, capaz de detectar e classificar objetos em imagem ou até mesmo em câmeras em tempo real. O YOLO por sua vez , utiliza apenas uma rede neural para a imagem inteira , por esta razão , o desempenho resulta em uma alta taxa de processamento.

2. Instalação

Se por um acaso , o usuário estiver utilizando outro meio para estar trabalhando com o YOLO sem ser o google colab deixo este link para estar visualizando a instalação de forma adequada.
Se o caso do usuário for pelo google colab , temos uma maneira simples de estar instalando:

- Em uma célula qualquer do notebook deem o seguinte comando:

```
pip install ultralytics
```

- Logo podemos importar as funções que serão necessárias ao nosso estudo.

```
from ultralytics import YOLO
import yaml
```

Com isso , será instalado ao terminal do notebook a biblioteca ultralytics , onde está situado o YOLO. Logo , podemos começar a utilizar a ferramenta em nossos projetos.

3. Formato YOLO

Para utilizar o algoritmo YOLO, é necessário estabelecer alguns padrões,como por exemplo:

- Remover os valores Nulos
- As classes devem ser numeradas a partir do ‘0’.
- Normalizar , se necessário, os valores da coordenadas da Bbox
- Devem-se criar um arquivo yaml para as configurações .
- Devem-se criar duas pastas: uma de treino e uma de validação , a terceira pasta que é a de teste é opcional.Nestas pastas devem ser descritas mais outras duas com os nomes de ‘labels’ e ‘images’

Obs: as Pasta Images e a Pasta labels devem ter obrigatoriamente o nome ‘images’ e ‘labels’, caso contrário o algoritmo pode não reconhecer e ocorrer algum tipo de problema.

- Pasta images: Onde contem as imagens que serão utilizadas no modelo
- Pasta labels: Onde irá conter os arquivos no formato ‘txt’ com as informações da classe do objeto e suas bounding box devidamente normalizada, no formato exemplo descrito abaixo:

  [1 0.195 0.127 0.04 0.046]

  [2 0.42 0.392 0.084 0.076]

- Essa linha acima descreve a classe , o ponto x do centro da bbox , o ponto y do centro da bbox, o tamanho e a altura da bbox , respectivamente.

Aqui nesta sessão abordaremos dois tipos de caso.O caso 1 é se o usuário não tiver um dataset definido , e o segundo caso vamos abordar o que ocorreu em na competição que participamos(os dados abordados podem ser encontrados no repositório).

O **primeiro caso** é se o seu objetivo for treinar um modelo de detecção de objetos em que o usuário não possui informações sobre as Bounding Boxes (ou seja , não ter um arquivo onde contém as informações sobre o objeto), recomendamos o uso do aplicativo "LabelImage". Com ele, é possível criar labels manualmente, construindo as bounding boxes do zero. Embora esse processo possa ser demorado e estressante, é uma das melhores opções para a construção de labels do zero. No LabelImage é permitido que faça as suas labels no formato YOLO diretamente. Caso tenha interesse em saber mais sobre aqui está um vídeo ensinando a lidar com exatamente este caso.

O **segundo caso** é o que ocorreu na competição em que participamos.

Dando um contexto geral, foi passado para os usuários um único dataset que tinha as seguintes informações:

-Nome da imagem em que o objeto estava presente
-Categoria do objeto
-BBox do objeto

Também foi disponibilizado uma única pasta que continha as imagens de treino e validação.

Com esses dados podemos manipular eles para ficar no formato YOLO.

1. Foi necessário primeiramente remover os valores nulos do nosso dataset. Existem várias maneiras de resolver esse objetivo , sendo um deles usando a função '.dropna()' da biblioteca 'pandas'.Essa abordagem garante que nosso dataset não contenha valores faltantes.

2. Logo em seguida, observamos que os valores na coluna 'category_id' não estavam no formato esperado pelo YOLO. Especificamente, os valores estavam como 1, 2 e 3. No formato YOLO, as categorias de objetos devem começar do valor 0.

   Para resolver isso, podemos subtrair uma unidade de cada valor na coluna 'category_id'. Essa abordagem é simples e eficaz para ajustar os valores conforme necessário. Veja como podemos fazer isso:

- **Leitura dos Dados**: Primeiro, carregamos os dados que contêm a coluna 'category_id'.
- **Ajuste das Categorias**: Em seguida, subtraímos uma unidade de cada valor na coluna 'category_id' para que as categorias comecem em 0.

  Este método garante que os valores de 'category_id' estão no formato apropriado para o YOLO, começando em 0, 1, 2, etc. É uma solução direta e eficaz para preparar seus dados para o treinamento ou a inferência do modelo YOLO.

3. No que se diz a respeito as Bbox , temos que normalizar elas , ou seja os valores das coordenadas têm que estar entre 0 e 1, para o padrão YOLO , as coordenadas tem que ficar da seguinte maneira [x , y , t , a] onde,

- valores de x , y devem ser o ponto do centro das Bbox
- valores de t , a devem ser o tamanho e altura

4. Essa questão varia de acordo com o data set disponibilizado, como estamos levando em conta a competição da unicef basta visualizar na descrição da competição qual foram os dados. Pegando de exemplo , o dataset disponibilizado continha os valores da coordenadas da seguinte maneira [xc, yc , tc , ac] onde,

- **xc , yc** são os x e y mínimos respectivamente
- **tc , ac** são o tamanho e altura respectivamente

5. Para normalizar essas coordenadas de forma adequada bastava usar as seguintes fórmulas.

- **para a coordenada xc:**

$$(xc + \frac{tc}{2})/tamanho\_imagem$$

- **para a coordenada yc:**

$$(yc + \frac{ac}{2})/tamanho\_imagem$$

- **para a coordenada tc:**

$$(tc)/tamanho\_imagem$$

- **para a coordenada ac:**

$$(ac)/tamanho\_imagem$$

- Assim as coordenadas do exemplo estarão normalizadas no formato YOLO *neste exemplo em questão.*

Obs1: O tamanho da imagem pode variar, com isso pode-se ter em mente criar algum tipo de código condicional.
Obs2: O intuito é mostrar como normalizamos para este caso específico , cabe aos usuários futuros deste documento compreender o que deve ser feito no projeto que estão trabalhando.

6. Com isso agora precisamos criar um arquivo que vai conter as configurações do nosso modelo que será treinado , no repositório que foi compartilhado tem uma configuração padrão que pode ser visualizada e usada.Aqui abaixo tem especificado o que cada variável condiz na estrutura deste arquivo.

- **path:** o diretório onde fica armazenado o conjunto de dados geral.
- **train:** o diretório onde fica armazenado o conjunto de dados treino.
- **val:** o diretório onde fica armazenado o conjunto de dados validação.

- **test *(opcional)*:** o diretório onde fica armazenado o conjunto de dados teste.
- **names:** define os nomes das classes que o modelo deve detectar , começando do ‘0’.

- Devidamente configurado de acordo com o projeto temos os arquivos de configuração do nosso modelo.

7. Com isso basta apenas agora separar os dados de treino e de teste , e criar nossas labels em formato txt , no notebook no repositório do Nias está apresentado a função que pode ser utilizado para separar os arquivos e criar as labels , mas é importante que seja generalizado e encontre outras formas de resolver.Temos alguns pontos a ser ressaltados sobre estas pasta.

- O nome do arquivo txt tem que ter o nome da sua respectiva imagem.
- Dentro do arquivo txt tem que conter o nome da classe e sua respectiva coordenadas , se por ventura tiver mais de uma Bbox , ela deve estar em outra linha no mesmo arquivo , como mostrado no arquivo de repositório compartilhado.
- É importante salientar que os labels e as imagens devem corresponder com os seus diretórios , não podemos uma imagem de treino e seu labels está no diretório de validação.

Pronto , feito este passo podemos finalmente começar a mexer com os modelos.

4. Criando modelo YOLO

Com isso pronto podemos importar e treinar nosso modelo.

Primeiramente temos que escolher que tipo de configuração queremos ter , para mais detalhes acesse este link da documentação YOLO , nele está apresentado com detalhes todos os tipos de configuração que os modelos apresentam , hiperparâmetros que podem ser mexidos e etc. O material é oficial da Ultralytics para quem tiver mais interesse sobre o assunto.

1. Para importar o modelo basta apenas criar uma variável com sua respectiva configuração, uma configuração de exemplo poderia ser o 'yolov8n.pt'.

```
modelo = YOLO('yolov8n.pt')
```

2. Agora será necessário mais um passo antes de treinarmos o modelo , que seria criar uma pasta chamada runs , nesta pasta que será salvo o nosso modelo treinado para futuramente realizarmos predições com o mesmos.

3. Depois de criado a pasta podemos colocar nosso modelo para treinar basta inserir o código abaixo:

```
modelo.train(data = 'diretorio do arquivo yaml aqui',
             epochs=15,
             project='pasta onde serão salvos',
             val=True)
```

Obs: existem muitos hiperparâmetros que podem ser configurados e ajustados para isso acesse o link da documentação para ter mais detalhes sobre essa parte.

4. Você irá poder acompanhar o desempenho do seu modelo durante as épocas de treinamento como mostrado na imagem abaixo.

```
Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
10/20         0G      1.915      1.668     0.3393         69        320: 100%|██████████| 120/120 [08:54<00:00,  4.46s/it]
           Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 16/16 [01:06<00:00,  4.14s/it]
             all        486       1004      0.333       0.32      0.198     0.0961
```

Significados dos predicts:

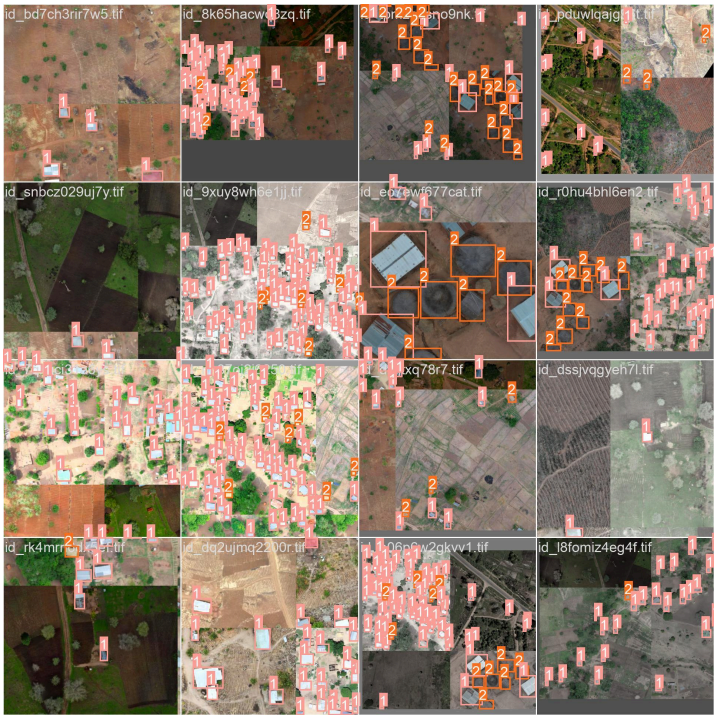
- **Box Loss** (Perda de Caixa): Esta perda mede a diferença entre as caixas delimitadoras (bounding boxes) previstas pelo modelo e as caixas reais. É usada para ajustar a posição, o tamanho e a forma das caixas delimitadoras geradas pelo

modelo para que correspondam o mais próximo possível às caixas delimitadoras verdadeiras nos dados.

- **Cls Loss** (Perda de Classificação): Esta perda avalia a precisão das previsões de classe do modelo. Em outras palavras, mede a diferença entre as classes previstas para cada objeto e as classes reais. O objetivo é minimizar essa perda para melhorar a precisão na identificação correta das classes dos objetos.
- **Dfl Loss** (Perda de Distribuição Focalizada): Esta perda é usada para melhorar a precisão da localização das caixas delimitadoras. DFL (Distribution Focal Loss) normalmente refere-se a uma técnica que ajusta a distribuição da perda, focando mais nas previsões difíceis ou mal classificadas, ajudando o modelo a aprender melhor essas previsões desafiadoras.
- **P (Precisão)**: A precisão mede a proporção de verdadeiros positivos (objetos corretamente identificados e classificados) em relação ao total de predições positivas feitas pelo modelo. Em outras palavras, é a quantidade de acertos entre as predições feitas. Uma alta precisão indica que poucos dos objetos identificados pelo modelo são falsos positivos.
- **R (Recall ou Revocação)**: O recall mede a proporção de verdadeiros positivos em relação ao total de objetos reais presentes nos dados. Em outras palavras, é a quantidade de acertos em relação ao total de objetos que deveriam ter sido identificados. Um alto recall indica que o modelo está conseguindo encontrar a maioria dos objetos reais, com poucos falsos negativos.
- **mAP50 (Mean Average Precision at IoU=0.50)**: O mAP50 é a média da precisão média (AP) calculada com um limiar de Intersection over Union (IoU) de 0,50. O IoU é uma métrica que avalia a sobreposição entre a caixa delimitadora prevista e a caixa real. Um IoU de 0,50 significa que a sobreposição entre as caixas deve ser de pelo menos 50% para ser considerada um acerto. O mAP50 é uma medida comum para avaliar a precisão de detecção de objetos.
- **mAP50-95 (Mean Average Precision at IoU=0.50 to 0.95)**: O mAP50-95 é a média da precisão média (AP) calculada em múltiplos limiares de IoU, variando de 0,50 a 0,95 (normalmente em incrementos de 0,05). Esta métrica fornece uma avaliação mais rigorosa e detalhada da precisão de detecção, considerando diferentes níveis de sobreposição entre as caixas previstas e as caixas reais. É uma métrica mais completa e frequentemente usada em competições e benchmarks de detecção de objetos.

5. Também podemos visualizar algumas outras coisas importantes. Na pasta “runs” que criamos anteriormente para salvar nossos treinos, vai ser criado uma pasta dentro dela chamada ‘train’ , e dentro desta pasta trein podemos tirar algumas informações , as mais importantes são:

- **Imagens de predição:** Uma imagem que mostrará como o modelo está reconhecendo os objetos , nessa imagem você consegue visualizar se as bbox estão delimitadas corretamente. A imagem abaixo ilustra isso.



- **A pasta ‘weights’:** Criada durante o treino , nela se encontra onde o modelo gerou a melhor performance durante as épocas de treinamento , e a última predição.Importante para fazer as predições.

Referencia Bibliografica.

AiNSTEiNSbr. COMO CRIAR UMA REDE YOLO DO ZERO. YouTube, 7 de fevereiro de 2022.
Disponível em: https://www.youtube.com/watch?v=8L3PCqADFPo&t=1238s

Ultralytics. Quickstart - Ultralytics TOLO Docs. Ultralytics ,2024.
Disponível em: https://docs.ultralytics.com/quickstart/#understanding-settings