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

- Faz a importação da pasta ***dataset*** do driver permitindo que o ***detectron*** tenha acesso a pasta com imagens que servirão como base de leitura para que a inteligência artificial seja capaz de reconhecer e responder de acordo com o padrão que foi.

```
!pip install pyyaml==5.1
!pip install torch==1.8.0+cu101 torchvision==0.9.0+cu101 -f https://download.pytorch.org/whl/torch_stable.html

[ ] !pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu101/torch1.8/index.html

[ ] import torch
    assert torch.__version__.startswith("1.8")
    import torchvision
    import cv2
```

- Instalação do ***detectron*** no ***colab*** junto de algumas dependências que servem como suporte do programa.

```
[ ] import os
    import numpy as np
    import json
    import random
    import matplotlib.pyplot as plt
    %matplotlib inline

    from detectron2.structures import BoxMode
    from detectron2.data import DatasetCatalog, MetadataCatalog
```

- Ainda mais alguns ***imports*** do ***detectron***.

```python
def get_data_dicts(directory, classes):
    dataset_dicts = []
    for filename in [file for file in os.listdir(directory) if file.endswith('.json')]:
        json_file = os.path.join(directory, filename)
        with open(json_file) as f:
            img_anns = json.load(f)

        record = {}

        filename = os.path.join(directory, img_anns["imagePath"])

        record["file_name"] = filename
        record["height"] = 430
        record["width"] = 410

        annos = img_anns["shapes"]
        objs = []
        for anno in annos:
            px = [a[0] for a in anno['points']] # x coord
            py = [a[1] for a in anno['points']] # y-coord
            poly = [(x, y) for x, y in zip(px, py)] # poly for segmentation
            poly = [p for x in poly for p in x]
```

- Nessa classe ele utiliza todos os códigos que são necessários para a padronização de imagens tornando elas mais visíveis ao ***detectron***, permitindo que ele faça a identificação nas semelhanças da figura em todos os ângulos que são propostos ao programa. Vale salientar tem que as imagens que estão servindo como base precisam ser definidas e pré-estabelecidas para que a IA possa reconhecelas.

```python
[ ] from detectron2 import model_zoo
    from detectron2.engine import DefaultTrainer, DefaultPredictor
    from detectron2.config import get_cfg
    from detectron2.utils.visualizer import ColorMode, Visualizer

[ ] cfg = get_cfg()
    cfg.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_X_101_32x8d_FPN_3x.yaml"))
    cfg.DATASETS.TRAIN = ("category_train",)
    cfg.DATASETS.TEST = ()
    cfg.DATALOADER.NUM_WORKERS = 2
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_X_101_32x8d_FPN_3x.yaml")
    cfg.SOLVER.IMS_PER_BATCH = 2
    cfg.SOLVER.BASE_LR = 0.00025
    cfg.SOLVER.MAX_ITER = 1000
    cfg.MODEL.ROI_HEADS.NUM_CLASSES = 2
```

- Modelo de segmentação de instancia após o ***detectron*** obter acesso as imagens o código fara alguns ***imports*** e começara a salvar e treinar a IA.

```python
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)
trainer = DefaultTrainer(cfg)
trainer.resume_or_load(resume=False)

trainer.train()

[ ] from detectron2.checkpoint import DetectionCheckpointer, Checkpointer
    checkpointer = DetectionCheckpointer(trainer, save_dir=cfg.OUTPUT_DIR)
    torch.save(trainer.model.state_dict(), "/content/drive/MyDrive/datasets/skin detectron 2 data/data/mymodel.h5")
```

- Nesse trecho do código, o própio ***detectron*** solicita o download do ***import*** para ser implementado dentro do ***colab***, todas as pastas e imagens antes já inseridas para assim alocar espaço em nuvem e iniciar o processo de treinamento.

```
cfg.MODEL.WEIGHTS = os.path.join(cfg.OUTPUT_DIR, "model_final.pth")
cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.5
cfg.DATASETS.TEST = ("skin_test", )
predictor = DefaultPredictor(cfg)
```

```
test_dataset_dicts = get_data_dicts(data_path+'test', classes)
```

```
# len(test_dataset_dicts) = 7, 7 dictionaries in a list (7 images for test)
# test_dataset_dicts[0] = each dict containing 4 keys: 'file_name', 'height', 'width', 'annotations'
```

- Após o treinamento ser concluído, alguns arquivos de teste serão aplicados tornando o código mais eficiente uma vez que haverá um mínimo de tratamento de erro caso alguma imagem das muitas listadas não se encontre nos padrões da IA.

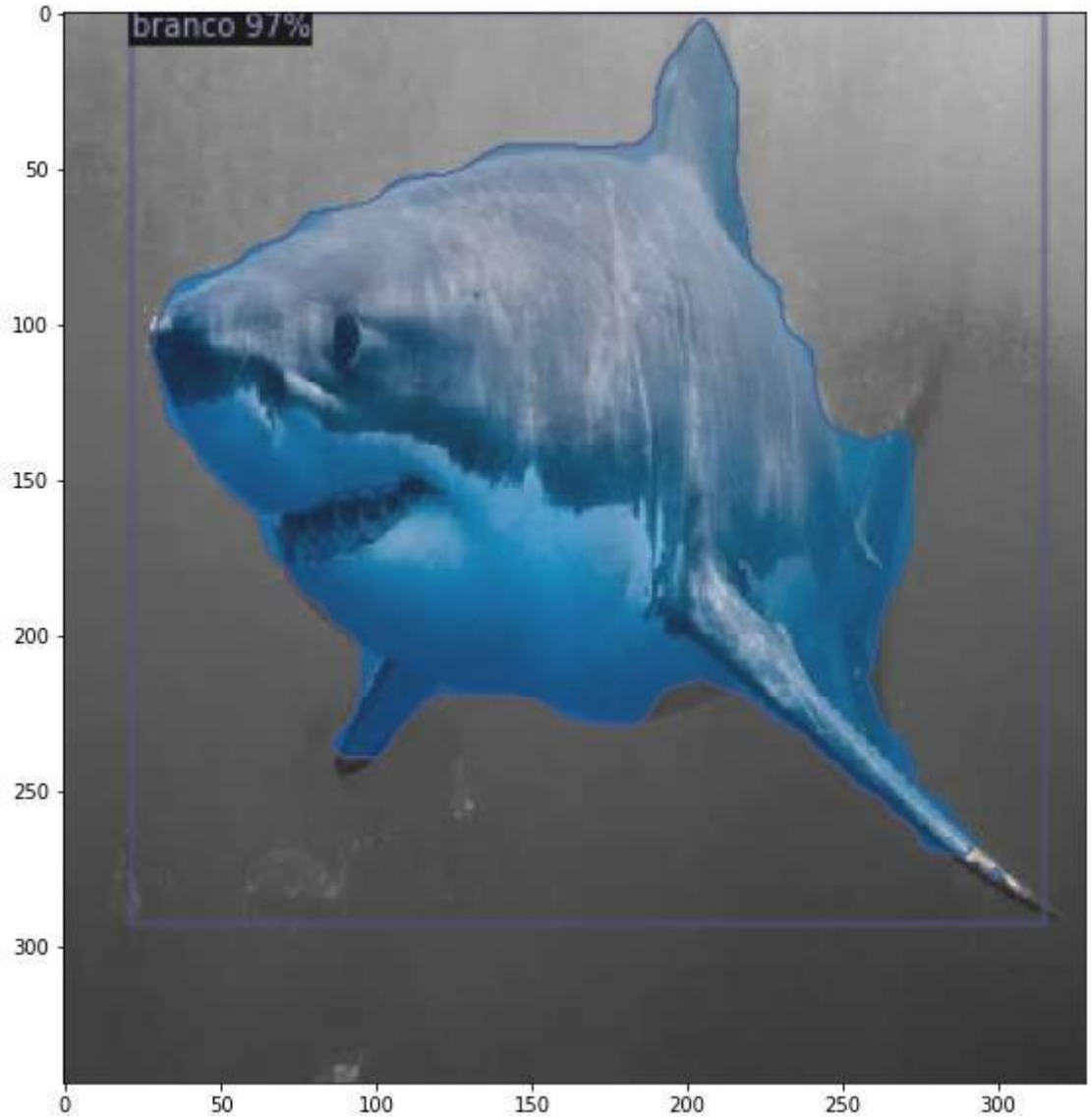
```
for d in random.sample(test_dataset_dicts, 1):
    img = cv2.imread(d["file_name"])
    outputs = predictor(img)
    v = Visualizer(img[:, :, ::-1],
                   metadata=microcontroller_metadata,
                   scale=0.8,
                   instance_mode=ColorMode.IMAGE_BW # removes the colors of unsegmented pixels
    )
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    plt.figure(figsize = (14, 10))
    plt.imshow(cv2.cvtColor(v.get_image()[:, :, ::-1], cv2.COLOR_BGR2RGB))
    plt.show()
```

- Este trecho do código de treinamento fara um teste randômico com uma das imagens listadas na pasta teste para conferir se tudo está funcionando corretamente, finalizando o trecho do código da parte de treinamento da IA.

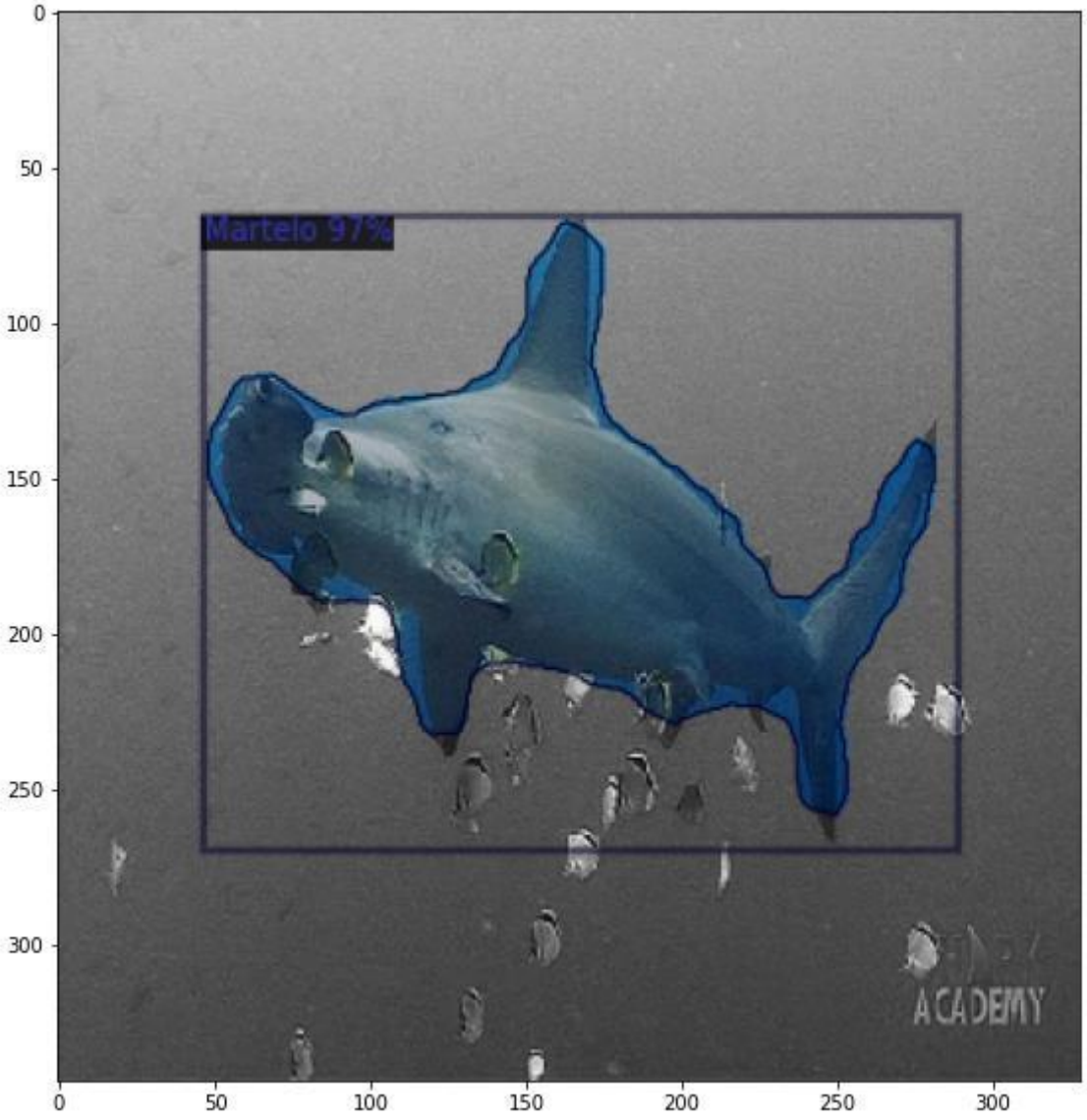
Teste de Execução

```
for d in random.sample(test_dataset_dicts, 5):
    img = cv2.imread(d["file_name"])
    outputs = predictor(img)
    v = Visualizer(img[:, :, ::-1],
                   metadata=microcontroller_metadata,
                   scale=0.8,
                   instance_mode=ColorMode.IMAGE_BW # removes the colors of unsegmented pixels
    )
    v = v.draw_instance_predictions(outputs["instances"].to("cpu"))
    plt.figure(figsize = (14, 10))
    plt.imshow(cv2.cvtColor(v.get_image()[:, :, ::-1], cv2.COLOR_BGR2RGB))
    plt.show()
```

Uma segunda classe irá executar tudo que foi executado antes, porém finalizará com esse trecho que irá fazer um teste com um número de 5 imagens encaminhadas da pasta ***dataset*** para aplicar a funcionalidade da IA.



Retirando das pastas de teste uma imagem para gerar um teste aleatório e comprovar se a IA está sendo efetiva.



Link GIT

https://github.com/StefanSalla/IAFINAL-STEFAN-NETO.git