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

- Faz a importação da pasta *dataset* do driver permitindo que o <u>detectron</u> 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.

- 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*.

```
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.

- 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.

```
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)

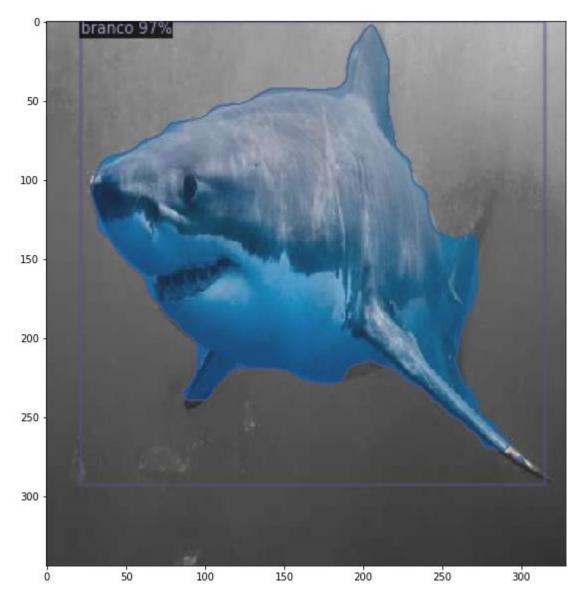
[ ] # len(test_dataset_dicts) = 7, 7 dictionaries in a list (7 images for test)
# test_dataset_dicts[0] = each dict containing 4 keys: 'file_name', 'height', 'width', 'annotations'
```

- 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.

- 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

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.

