Detecção de fases de minérios usando inteligência artificial

Ore Phase Detection Using Artificial Inteligence

Nancy Baygorrea
Bolsista PCI, Computer science, D.Sc.

Otávio Gomes
Supervisor, Material science, D. Sc.

Resumo

Neste trabalho, trata-se de um estudo sucinto de métodos de detecção de fases de minérios em imagens de eléctron retroespalhados, obtidas com microssonda eletrônica, com o uso de inteligência artificial. Utilizando as ferramentas da interface de programação de aplicativos de detecção de objetos do Tensorflow, a partir de um modelo *pré-treinado,* pretende-se fazer um estudo comparativo de dois modelos de detecção afim de identificar as fases de minérios nas imagens com o fim de, posteriormente, construir um modelo mais inteligente de detecção de fases nas imagens de minérios.

Palavras-chave: modelos de detecção de objetos; inteligência artificial; tensorflow; redes neurais.

Abstract

This work is a brief study of ore phase detection methods in backscattered electron images, obtained with scanning electron microscope (SEM), using artificial intelligence. Using the Tensorflow object detection with application programming interface tools, from pre-treined, we intend to make a comparative study of two detection models in order to identify the phases of ores in the images, to later, to build a smarter model of phases detection in ore images.

Key words: object detection models; artificial intelligence; tensorflow; neural networks.

1. Introdução

A detecção de objetos (OD), em visão computacional, é a tarefa de detectar instancias de objetos numa determinada classe presente em uma imagem. Essa detecção não só foca a classificação de diferentes instancias, mas também tenta estimar as características e localizações desses objetos contidos em cada imagem. A detecção de objetos (OD) tem crescido muito nos últimos anos, apresentando uma vasta quantidade de novos modelos de redes neurais desenvolvidos. De fato, a OD tem muitas aplicações em visão computacional e "*Deep Learning"* inclusive na mineralogia (CHEN et al, 2017; IU et al., 2021). Entretanto, encontra-se poucos "*datasets standar"* nessa área referente a minérios. Além disso, existem outras aplicações, como por exemplo, em testes de reconhecimento de face (SUNK: POGGIO, 2002; YANG: NEVATIA, 2016), detecção de pedestre (WOJEK et al., 2012), detecção de esqueleto (KOBATAKE;YOSHINAGA, 1996) condução automática (CHEN et al., 2015, 2017), condições de clima (MATHIAS et al., 2013; NEUMANN, 2019), entre outros.

O estado de arte das arquiteturas de OD geralmente consiste em dois estágios, muitos dos quais tem sido treinados em *dataset* COCO (TSUNG et al. 2014), o dataset mais comum para detecção de objetos usado para avaliar o desempenho de métodos de visão computacional). O primeiro estágio é o detetor de objetos de estado único (redes convolucionais de disparo único para reconhecimento de objetos) remove o processo de extração da região de interesse (RoI), classifica e faz regressão ao candidato de caixa anchor, sendo um exemplo dessa e modelo a família do YOLO, CornerNet, CenterNet entre outros); o detetor de objetos de dois estágios divide a tarefa de detecção de objetos em outros dois estágios: extrai a RoI, classifica e faz regressão nessa região. Eles também são chamados de classificadores de regiões associados a extratores de característica em CNN (por exemplo R-CNN (GIRSHICK:DONAHUE;DARREL;MALIK, 2014), Faster-RCNN [(GIRSHICK, 2015), Mask-RCNN (HE et al. 2017), entre outros)

O pipeline de modelos de detecção de objetos tradicionais podem ser divididos em três etapas: seleção de região de informação, extração de características e classificação. De fato, quando uma imagem é um input na CNN, o problema de classificar as classes correspondentes de cada instancia é conhecido como problema de classificação. A saída desta rede neural é representada como um valor de probabilidade para todas as classes. Logo vem uma tarefa de classificação nesses valores de probabilidade o que de fato entende-se como um problema de regressão para prever a posição do objeto usando uma caixa delimitantes retangular (bounding box) o que registra a região de maior confiança da presença da instancia detectada. Na previsão, os modelos são tipicamente medidos de acordo com a métrica Média da Interseção sobre a União (Mean IoU).

2. Objetivos

- Construir um modelo pré-treinado para detecção de objetos multiclasses e fazer a inferência no nosso *dataset* particular;

- Construir dois modelos de detecção de objetos multiclasse com ferramentas de Tensorflow utilizando a informação de uma rede pré-treinada;
- Fazer um estudo comparativo dos resultados obtidos pelos dois modelos dados.

3. Material e Métodos

Treinar um modelo de aprendizado profundo (deep learning) para o problema de OD precisa de um conjunto de datas muito grande, além de recurso computacional braseados em processadores composto por muitos núcleos menores e mais especializados (GPU). Então, a abordagem mais simples para facilitar o desenvolvimento do experimento seria, de fato, começar com um modelo *pré-treinado,* re-treinar para detectar os objetos do dataset particular (processo chamado de "Transfer Learning") e treinar novos modelos no nosso *dataset*.

Nesta seção, vamos discutir o treinamento de um modelo de OD para um *dataset* particular com o uso do API de OD de TensorFlow e Google Colab. O TensorFlow OD é um software de recurso livre para construir modelos de detecção de objetos e segmentação de imagens que localizam múltiplas instancias na mesma imagem. Algumas das arquiteturas e modelos que Tensorflow OD API inclui: CenterNet, EfficientDet, SSD MobileNet, SSD ResNet, faster R-CNN, ExtremeNet, Mask RCNN. Para nossos experimentos, utilizaremos o EfficientDet e o Faster R-CNN.

O roteiro do nosso experimento é dado como segue:

a) Preparar o *dataset*;

b) Clonar as librerias do API OD Tensorflow, installa-o, e importar as suas funções.

c) Carregar um modelos pré-treinado com seus pesos e os últimos *checkpoint*s;

d) Fazer a *inferência* para os nossos dados;

e) Treinar dois modelos de detecção com nossos dados considerando a informação do *checkpoint* do modelo pré-treinado.

A seguir alguns detalhes do experimento em cada etapa:

a) Preparação de dados: Nosso conjunto de imagens minerais de elétron retro espalhados, obtidas com uma microssonda eletrônica, consiste de biotita (rotulo: 'B'), plagioclásio (rótulo: 'P'), óxidos (rótulo: 'O'), anfibólios (rótulo: 'Ap'), piroxênio(rótulo: 'px') e espaços vazios ou buracos rótulo:'hole'). Todas essas imagens são de tamanho 512x512, em escala de cinza e formato JPG.

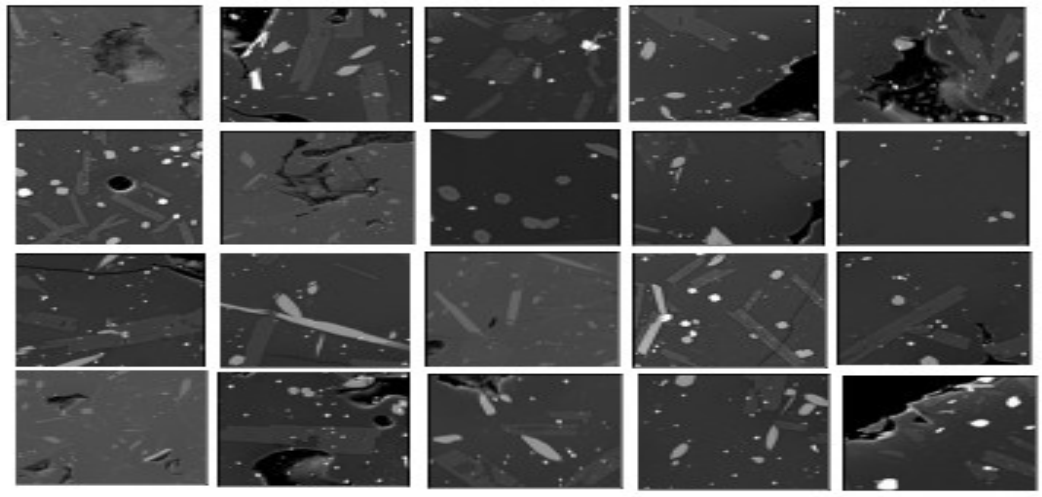


Figura 1. Conjunto de imagens que será utilizado na construção do nosso *dataset*.

Para a construção de um *dataset* para treinamento, é necessário fazer a rotulagem de cada uma das fases dos minérios nas imagens. Existem vários recursos para fazer essa rotulagem para modelos de detecção de objetos, por exemplo, LabelBox, Scale AI, SuperAnnotate, Dataloop, Playment, Supervise.ly, Hive Data, CVAT, Labelme, LabelImg, VoTT, Img Lab, entre outros. Para nosso propósito foi utilizado o LabelImg (*https://github.com/tzutalin/labelImg*), os quais salvam as anotações em extensão XML e formato PASCAL VOC. O formato mais comum utilizado por vários modelos de OD é o formato COCO, mas a conversão de qualquer formato para o COCO é feito em um comando simples. O Roboflow é um software muito útil porque ajuda no pré-processamento assim como também com o aumento de imagens com algumas operações (processo chamado de '*data augmentation*').

Além disso, muitos dos modelos pré-treinados disponíveis tem sido treinados utilizando o ImageNet (*dataset* que contém imagens de formato RGB). Portanto, se usamos esses modelos pré-treinados para imagens em escala de cinza, como é no nosso caso, são incompatíveis com modelos pré-treinados em imagens RGB. Nesse cenário, temos duas possíveis abordagens que podem ser consideradas: acrescentar canais individuais para cada imagem em escala de cinza para ter três ou mais canais ou modificar a primeira camada convolucional da rede neural pré-treinada. No nosso experimento apresentado nesta seção foi considerado a primeira abordagem pela simplicidade na manipulação de datas evitando assim modificar o modelo.

b) Clonar as librerias do API OD Tensorflow, installa-o, e importar as suas funções. Precisa-se também importar o módulo object_detection.utils do Tensorflow versão 2.2.0. O modelo pode ser baixado do repositório oficial: https://github.com/tensorflow/models.

c) Carregar um modelo pré-treinado com seus pesos e os últimos *checkpoint:* Para nosso propósito de construir um modelo pré-treinado, vai ser considerado o modelo model=download_model ('centernet_hg104_512x512_coco17_tpu-8'). Nesse repositório encontra-se o pipeline_config (informação sobre as configurações do modelo) para personalizar de acordo a nosso *dataset*, o caminho é o seguinte: centernet_hg104_512x512_coco17_tpu-8/saved_model/pipeline.conf. Esta é a parte mais importante do pré-treinamento.

```
from object_detection.builders import model_builder
pipeline_config = "./centernet_hg104_512x512_coco17_tpu-8/pipeline.config"
model_dir = "./centernet_hg104_512x512_coco17_tpu-8/checkpoint"
configs = config_util.get_configs_from_pipeline_file(pipeline_config)
model_config = configs['model']
detection_model = model_builder.build(model_config=model_config, is_training=False)
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(model_dir, 'ckpt-0')).expect_partial()
```

Figura 2. Construindo o modelo com pesos *pré-treinado*s.

Na Figura 2, mostra o código para carregar o último *checkpoint* do modelo. Além disso, carrega a configuração do *pipeline* e faz a construção do modelo de detecção. A última linha restaura os pesos desde o *checkpoint* para ser utilizado.

d) Fazer a i*nferência* para os nossos dados: Antes de começar com a inferência precisa-se colocar as imagens em modo tensor: Para isso, vamos utilizar a função A função load_image_into_numpy_array() o qual converte uma imagem em um arranjo compatível com um grafo tensorflow. Por convenção, coloca-se um arranjo de tamanho (altura, largura, 3), onde o valor de 3 representa o número de canais de uma imagem RGB.

Para a *inferência* de detecção de objetos, precisa-se de preparar as imagens, o dicionário das classes (label_map) e os dados de anotações tais como: arranjo de caminhos das imagens: train_image_filenames, um dicionário de índices e classes (o índice começa com valor de 1), número de classes: 6, arranjo de identificação das classes: por exemplo gt_labels=(np.array([1,4,65]), np.array([4,5,5,5,5])), as caixas delimitantes: um arranjo de [ymin, xmin, ymax, xmax]. Para esse fim, foi escrito uma função chamada get_boxes() que, lendo cada arquivo xml das imagens para treinamento, retorna um arranjo que contém a informação das caixas delimitantes e as classes dos rótulos para cada imagem: gt_boxes, get_labels= get_boxes(path_xml). Por exemplo, gt_labels=[np.array([1,1]), np.array([1,2,3])] e gt_boxes=p.array([[0.436, 0.591, 0.629, 0.712],[0.539, 0.583, 0.73, 0.71]], dtype=np.float32), np.array([[0.464, 0.414, 0.626, 0.548],[0.313, 0.308, 0.648, 0.526],[0.256, 0.444, 0.484, 0.629]], dtype=np.float32), onde cada arranjo em cada um deles representa informações, de pontuação para os label (1: 'P', 2: 'px', 3: 'B', 4: 'O', 5: 'Ap', 6:'hole') e suas coordenadas da caixa delimitante, de duas imagens para o processo de treinamento. O Category_index é construido como: category_index={1: {'id:1, 'name':'P'}, 2: {'id:2, 'name':'px'}, 3: {'id:3, 'name':'B'}, 4: {'id:4, 'name':'O'}, 5: {'id:5, 'name':'Ap'}, 6: {'id:6, 'name':'hole'}}. Ao fim, construir uma função detect_fn() que leia o modelo e retorne as detecções e um dicionario de predições como tensor.

e) Treinar dois modelos de detecção com nossos dados considerando a informação do *checkpoint* do modelo pré-treinado: Com todos esses inputs definidos, nós editamos o arquivo do pipeline para adaptar no nosso *dataset* particular, os *checkpoint* pré-treinados e também alguns parâmetros de treinamento específicos. Assim, para fazer um melhor treinamento, incrementa o batch_size de acordo com a capacidade do GPU.

No caso de treinamentos curtos, o número de passos tem que diminuir com o mesmo fator que foi incrementado o batch_size.

inferência Vamos utilizar *inferência* no método, para isso agregamos as imagens testes para a pasta localizado no tensorflow-object-detection/test. O resultado da inferência do modelo com nosso *dataset*, mostra três imagens com os rótulos certos e com a detecção resultante do modelo pré-treinado, para cada uma delas. Veja Figura 3.

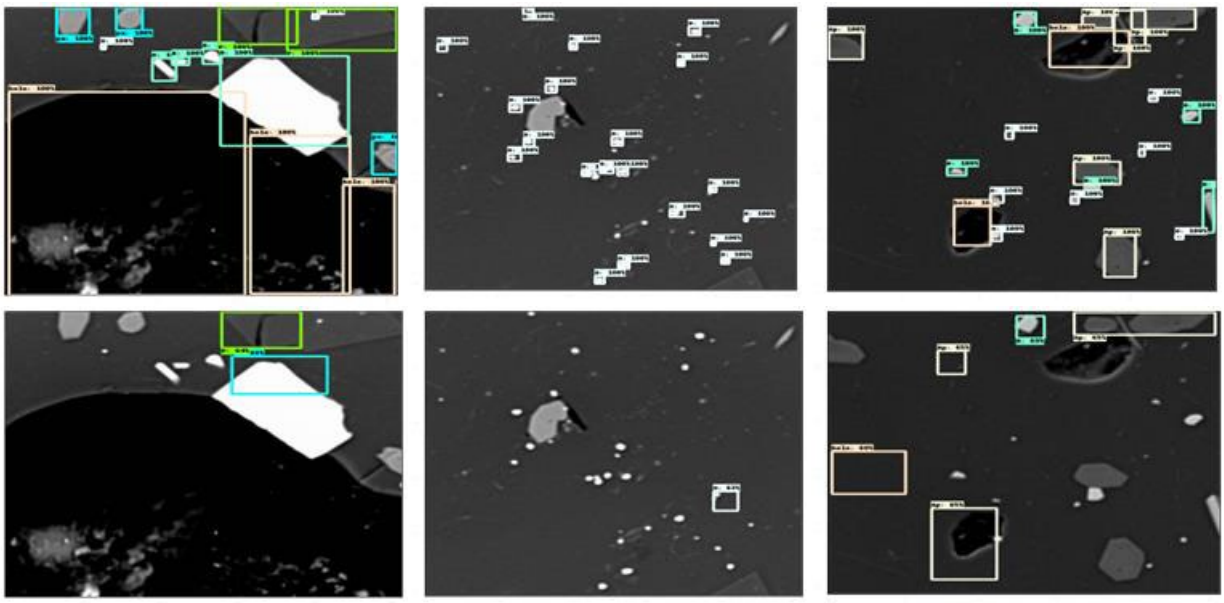


Figura 3. Superior: imagens com caixas delimitantes corretas.
Inferior: Imagens de saída do modelo pré-treinado Centernet_hg_104_512x512_coco17_tpu-8.

Nesta segunda parte do experimento, a fim de melhorar os resultados obtidos do modelo pré-treinado, vai-se treinar dois modelos de OD do TensorFlow2 (EfficientDet e o Faster-RCNN) com nosso próprio *dataset* utilizando a informação do *checkpoint* do pré-treinamento anterior. A determinação da escolha desses modelos foi determinada porque o EfficientDet é um dos algoritmos mais estáveis e eficientes para problemas OD e o Faster R-CNN (detector de dois estágios) é um algoritmo de reconhecimento de objeto de tipo de rede neural convolucional baseado em regiões.

Cada modelo de OD do Tensorflow2 tem pastas que podem ser personalizadas para nosso interesse: model_name, um arquivo base_pipeline_file (informações de treinamento de configuração específica que podemos adaptar para nosso *dataset*), um pretreined_*checkpoint* (localiza-se os pesos pré-treinados salvos desde que o modelo foi pré-treinado utilizando o *dataset* COCO) e um batch_size. Com os inputs definidos, editamos o base_pipeline_file com as informações do nosso *dataset*, o pretrained_*checkpoint* e também alguns parâmetros específicos. O modelo escolhido (EfficientDet e Faster R-CNN) vai começar com pesos salvos no pretreined_*checkpoint*, e será realizado um ajuste (chamado de '*fine tuning*'). Assim, o modelo começa identificando quais as características que devem ser úteis para a detecção de objetos.

Para treinar por mais tempo incrementamos o num_steps, para um treinamento rápido incrementamos o batch_size para um nível que o nosso GPU consiga trabalhar. Vale a pena ressaltar que para acessar às

imagens rotuladas o FASTER-RCNN precisa de arquivos de extensão XML em formato PASCAL VOC, entretanto, a família do EfficentDet utiliza o formato COCO JSON. A conversão de um para o outro é feito com simples linhas de comando ou softwares de rotulagens.

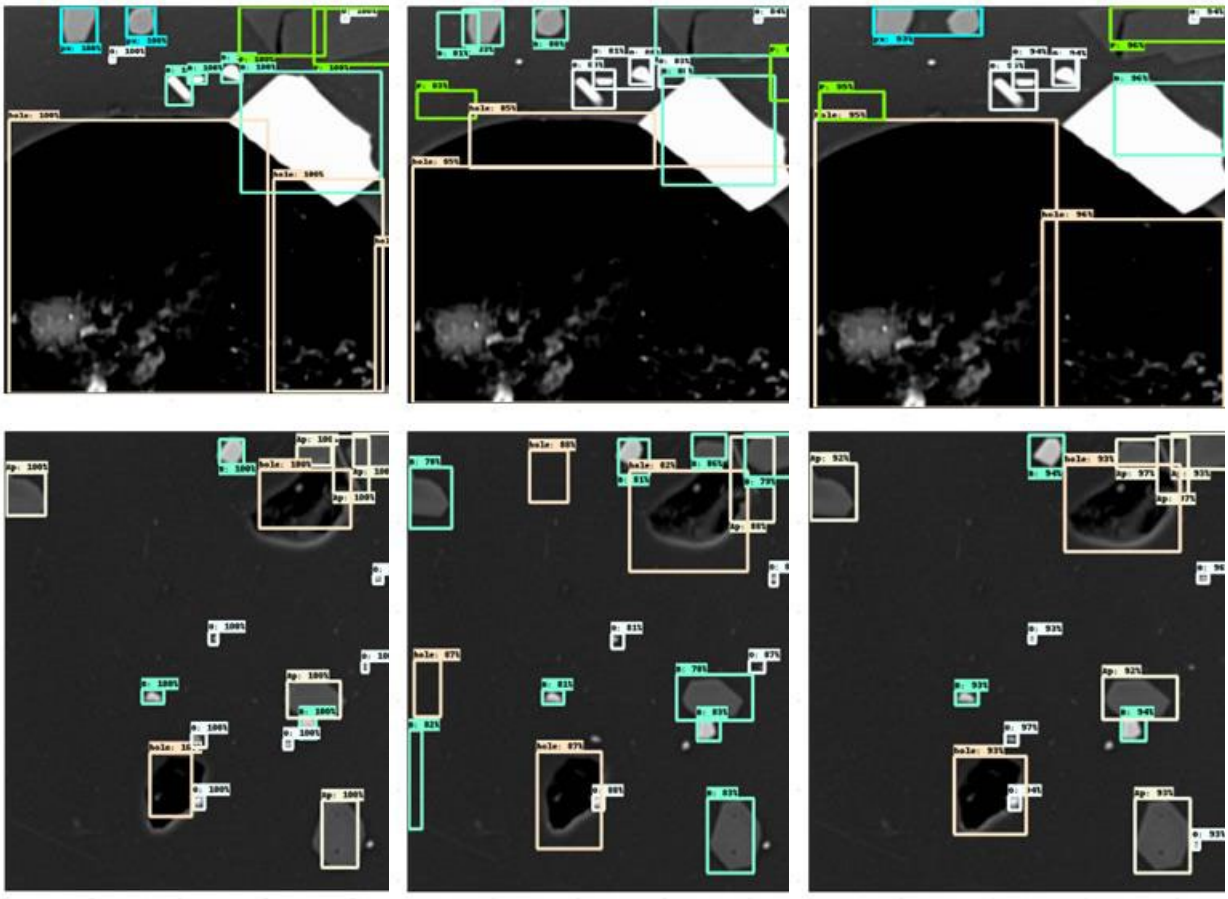
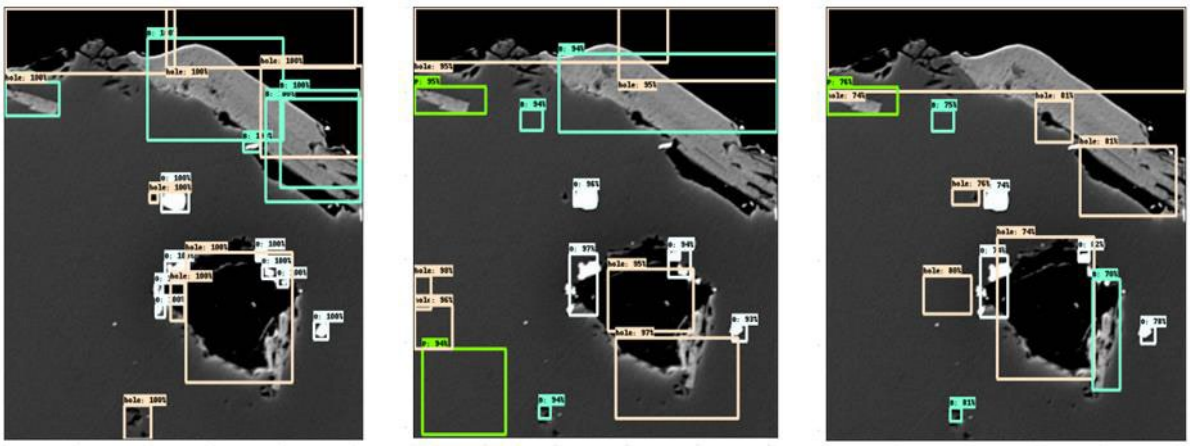


Figura 4. Esquerda: Imagem com rotulagem correta. Centro:
imagem de saída do modelo EfficientDet. Direita: imagem de saída do modelo Faster-RCNN.

4. Resultados e Discussão

A partir dos resultados obtidos, de acordo com as Figuras 4 e 5, podemos fazer algumas observações: 1- o modelo Faster R-CNN não detecta fases muito pequenas, tais como os óxidos (caixa delimitante de cor branca) de forma individual mas se ocorrerem com aglomerações são detectáveis. Por outro lado, identificou-se de forma errônea a fase da biotita (observe a caixa delimitante verde que identifica a biotita como plagioclásio (P)). Podem ser observadas também algumas fases que não estão na imagem testada (ver Figura 5-b). 2- Nos resultados obtidos pelo modelo EfficientDet, Figura 5-c, observa-se que muitas das fases foram identificadas de forma errônea, a biotita também foi identificada como plagioclásio, uma parte da fronteira do espaço vazio foi identificado como biotita (B). Houve identificações de pequenos espaços vazios (hole) onde na imagem à esquerda não aparece. Detectou-se também, algumas fases que não estão na imagem testada. Na Figura 4 podem ser observados alguns anfibólios (Ap) que foram identificados como biotita (B).



(a) (b) (c)

Figura 5. Esquerda: imagem com rótulo correto que contém espaços vazios(caixa delimitante de cor laranja) e biotita (caixa delimitante de cor verde). Centro: imagem obtida pelo modelo Faster-RCNN. Direita: Imagem obtida pelo modelo EfficientDet.

5. Conclusões

Podemos concluir que os modelos de Tensorflow para detecção de fases de minérios funciona razoavelmente bem. É importante apontar que para conseguir todos esses resultados obtidos pelos dois modelos (Efficient e Faster R-CNN) foram necessários pré-classificar manualmente alguns tipos de imagens de maneira proposital para a construção do *dataset*, de fato, foi verificado que imagens com muita quantidade de fase de plagioclásio, fases muito pequenas ou imagens de baixa resolução que eram treinadas com esses modelos retornavam imagens sem ou com poucas detecções erradas. Então, esses tipos de imagens foram descartadas para formar o *dataset* deste experimentos. Vale a pena explorar outros modelos para a detecção dessas fases minerais. Por fim, é importante continuar com esses estudos afim de construir um modelo mais inteligente de detecção de fases de minérios, sobretudo para imagens que possuem fases muito complexas de serem identificadas.

6. Agradecimentos

Agradece-se ao CNPq pelo auxilio financeiro para a realização deste trabalho. Ao Prof. Dr. Otávio Gomes pela parceria, oportunidade, motivação e apoio na elaboração deste trabalho.

7. Referências Bibliográficas

CAI, Z.; LEI, S.; XIAOJUAN, L. Deep Learning Based Granularity Detection Network for Mine Dump Materials, 2022, **Journal Minerals.** Key Laboratory of Metallurgical Equipment and Control Technology, Wuhan University of Science and Technology, Wuhan, China. 2022, v.4, p.2075-163X.

CHEN, X.; MA, H.; WAN, B; XIA, T. Multi-view 3d object detection network for autonomous driving. In INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 2017.

CHEN, C.; SEFF, A.; KORNHAUSER, A.; XIAO, J. Deepdriving: Learning affordance for direct perception in autonomous driving. In INTERNATIONAL CONFERENCE ON COMPUTER VISION, 2015.

GIRSHICK, R. Fast R-CNN.In: IEEE international conference on computer vision. **Proceedings of the Computer Science, Environmental Science**. Santiago. 2015. p.1440-1448.

GIRSHICK, R.; DONAHUE, J.; DARREL, T.; MALIK, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In: COMPUTER VISION AND PATTERN RECOGNITION, Columbus, 2014, p.80-587.

HE, K.; GKIOXARI, G.; DOLLAR, P. GIRSHICK, R. MASK RCNN. **Proceedings of the IEEE international conference on computer vision.** 2017. p. 2961-2969.

IU, Y.; ZHANG, Z.; LIU, X.; LEI, W.; XIA, X. Deep Learning Based Mineral Image Classification Combined With Visual Attention Mechanism. **In *IEEE Access*, 2021.** v. 9, p.98091-98109.

KOBATAKE, H.; YOSHINAGA, Y. Detection of spicules on mammogram based on skeleton analysis. **IEEE Trans. Med. Imag.,** v.15, nº.3, p. 235-245, 1996.

MATHIAS, M.; BENENSON, R.; VAN GOOL, L. Traffic sign recognition - how far are we from the solution?. **Proceeding of the 2013 INTERNATIONAL JOINT CONFERENCE on NEURAL NETWORKS (IJCNN).** 2013, p. 1-8.

NEUMANN, L. et al. NightOwls: A Pedestrians at Night Dataset. **Asian Conference on Computer Vision.** v. 11361, Springer, Cham, p. 691-705, 2019.

NGUYEN, N.D.; DO, T.; NGO,TD.; LE, DD. An evaluation of deep learning methods for small object detection. **Journal of Electrical and Computer Engineering.** V.2020:1-18. DOI: 10.1155/2020/3189691, 2020.

SUNG K.; POGGIO, T. Example-based learning for view-based human face detection. **IEEE Trans. Pattern Anal. Mach. Intell.,** v. 20, nº. 1, pp. 39-51, 2002.

SUSANTO, E.R.; ANALIA, R.; SUTOPO, P.D.; SOEBAKTI, H. The deep learning development for real-time ball and goal detection of barelang-FC, **INTERNATIONAL ELECTRONICS SYMPOSIUM ON ENGINEERING TECHNOLOGY AND APPLICATIONS (IES-ETA) IEEE,** p.146-151.

TSUNG, Y. et al. Microsoft COCO:Common Objects in Context. In ECCV. Science Computer. 2014. p. 1405.0312.

YANG, Z.; NEVATIA, R. A multi-scale cascade fully convolutional network face detector. In INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION, 2016.

WOJEK, P.; DOLLAR, B. SCHIELE, B. PERONA, P. Pedestrian detection: An evaluation of the state of the art. **IEEE Trans. Pattern Anal.Mach. Intell.,** v. 34, nº. 4, p.743, 2012.