# NYCU Visual Recognition using Deep Learning
# HW3 report

109612019, 林伯偉

**Github Link** : https://github.com/Albert5865/NYCU-Computer-Vision-2025-Spring/tree/main/HW3

## 1. Introduction:

In this project, the task of instance segmentation was tackled by using deep learning, where the goal is to detect and segment cells in colored medical images. The model architecture is based on Mask R-CNN, a well-established approach for instance segmentation tasks. The dataset comprises colored images along with their corresponding masks, representing four distinct classes of cells. The primary objective of this work is to accurately segment these cells and produce a reliable submission for the competition on CodaBench.

## 2. Method:

To achieve the desired segmentation, we used the following approach:

### Data Pre-processing :

The dataset consists of medical images in TIFF format, each containing multiple cells represented by their respective segmentation masks. To prepare the data for training, I've performed the following pre-processing steps:

- **Image Loading and Augmentation:** The images were loaded using the `tifffile` library, and basic augmentations like random horizontal flips were applied to improve model generalization.
- **Mask Processing:** The masks are stored as individual TIFF images for each class, and for each instance within the class, the mask is processed and converted to the required format (Run-Length Encoding for submission).

### Training Strategy to Prevent Memory Overload: :

Due to the large size of the dataset and the model, training was split into smaller chunks to prevent out-of-memory (OOM) errors. Instead of loading the entire dataset into memory at once, each epoch was divided into 10 segments, with only 1/10th of the data loaded in each segment. This strategy effectively reduced memory usage while still allowing the model to train on the entire dataset.

### Model Architecture :

I employed a modified **Mask R-CNN** model architecture:

- **Backbone:** resnetrs200.tf_in1k, a high-performing ResNet variant with 93.4M parameters, was used as the backbone for feature extraction.

- **Neck:** FPN (Feature Pyramid Networks) was used to enhance multi-scale feature extraction.
- **Heads:** The heads include a Mask prediction head to generate the segmentation masks, a Box prediction head for bounding boxes, and a classification head for identifying the class of each instance.

- Model : resnetrs200.tf_in1k
- Learning rate : 5e-5
- Batch size : 1
- Data augmentation :
  - HorizontalFlip (p=0.5)
  - VerticalFlip (p=0.5)
  - RandomRotate90 (p=0.5)
  - ShiftScaleRotate (shift_limit=0.1, scale_limit=0.1, rotate_limit=15, p=0.5)
  - RandomBrightnessContrast (p=0.2)
  - GaussianBlur (p=0.2)
  - HueSaturationValue (p=0.2)
- Weight decay : 5e-4
- Optimizer : AdamW
- Epoch : < 150 (early stopped after 10 rounds without improvement)

## Why this Architecture:

**resnetrs200.tf_in1k** was chosen as my backbone because it is a highly efficient and robust architecture designed for extracting high-level features from images. This backbone is designed to capture multi-scale features, which are crucial for segmenting objects (cells) of different sizes and shapes in complex medical images. However, the model is still memory-intensive, requiring careful memory management techniques like segmenting the data during training. Nonetheless, the larger architecture requires significant computational resources for both training and inference, especially when dealing with large datasets.

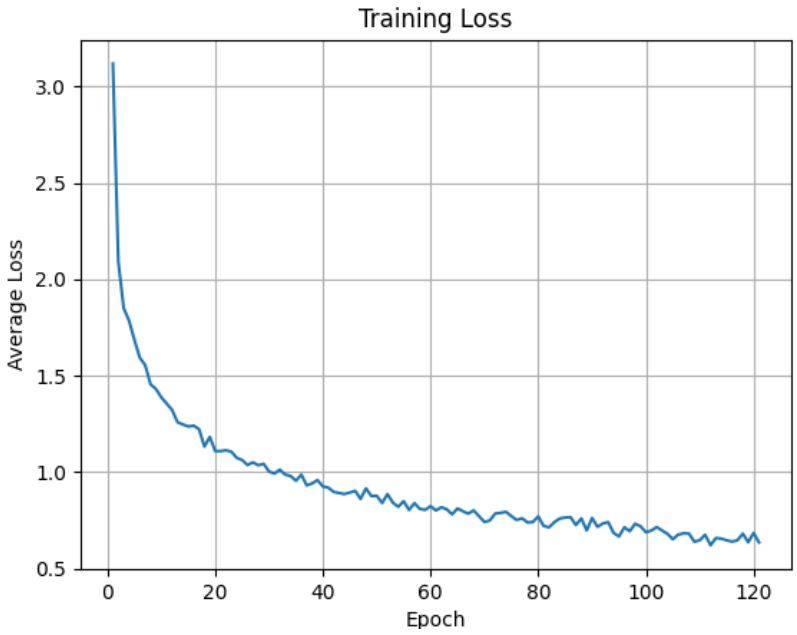
**Pros:**

- **Efficiency:** It is designed to efficiently capture multi-scale features, essential for detecting objects (cells) at various sizes.
- **Flexibility:** This backbone works well across different datasets and tasks, making it a solid choice for medical image segmentation.

**Cons:**

- **Memory Intensive:** While efficient, the model is still memory-intensive, requiring careful memory management techniques like segmenting the data during training.
- **Computational Cost:** The larger architecture requires significant computational resources for both training and inference, especially when dealing with large datasets

## 3. Results:

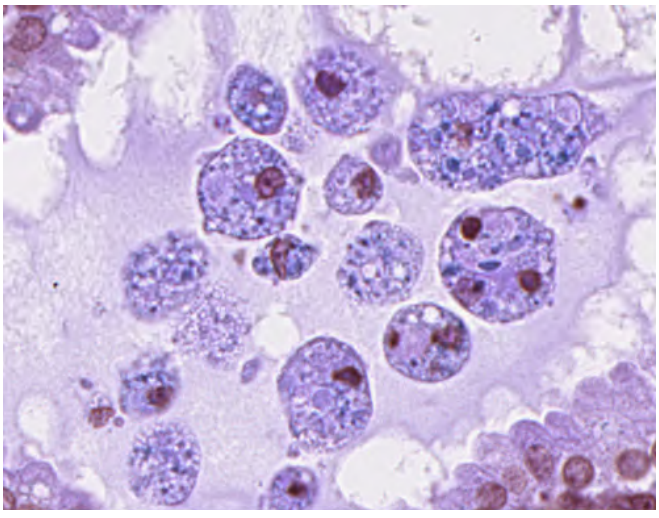


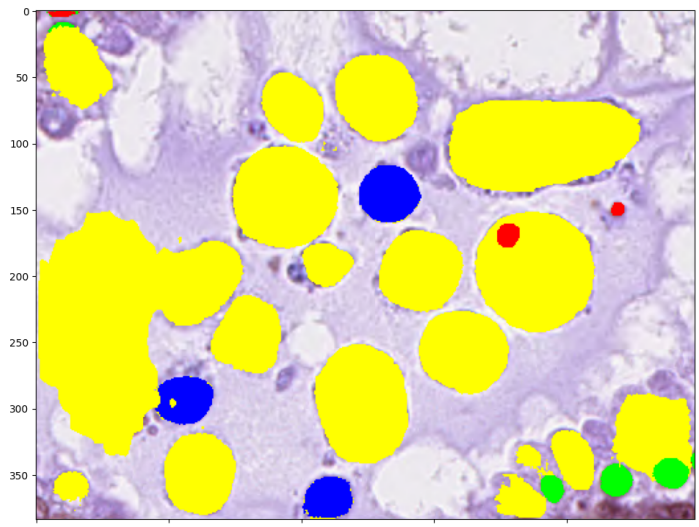

scoring_result-resnetrs200-dataaug_round047_loss0.8602-0.29 > {} scores.json > ...

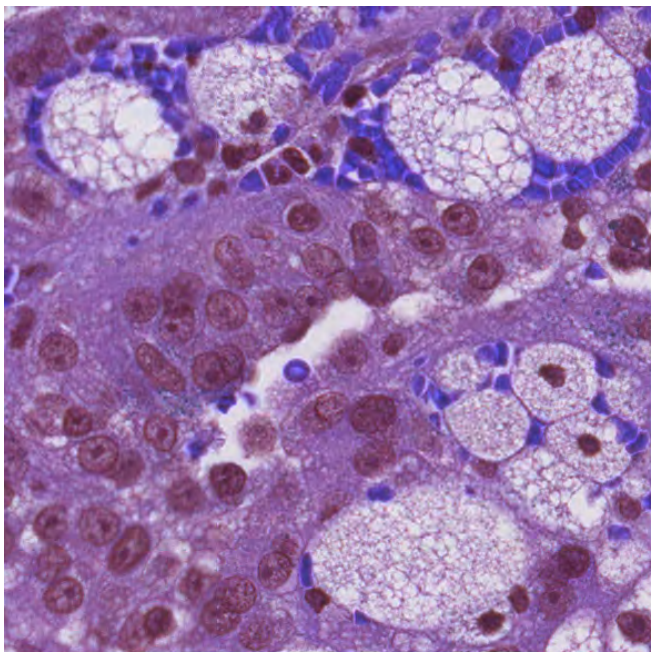
```
1   {"public_score": 0.287886122637029, "private_score": 0.32893856806963334}
```

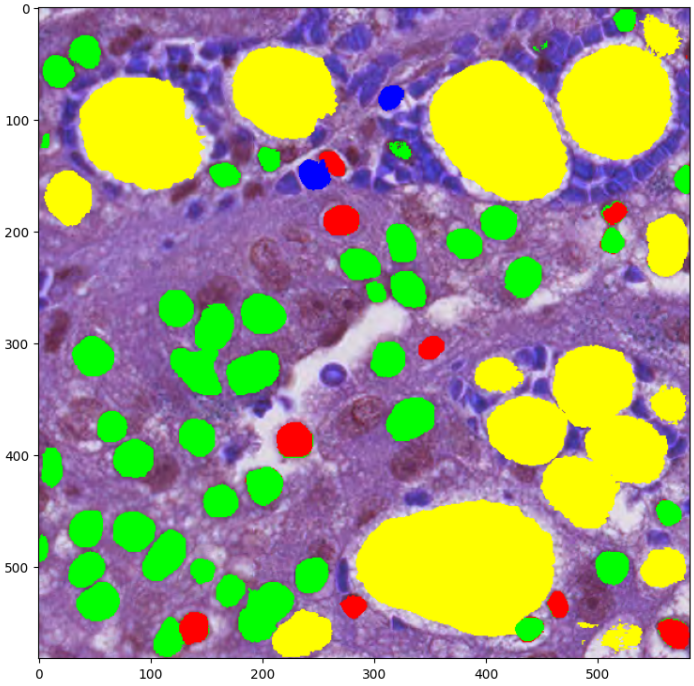
The loss plot above of the resnetrs200.tf_in1k model with data augmentation shows a sharp decline in training loss, stabilizing around 0.7 by the end of training, indicating good convergence. However, despite the model saved at **epoch 121** having a smaller loss, it didn't perform as well as the one saved at **epoch 47**. The **public score** of 0.2879 and the **private score** of 0.3289 were achieved with the model from epoch 47, which suggests that the model saved at epoch 121, although optimized, may have started to overfit, losing generalization ability on unseen data.

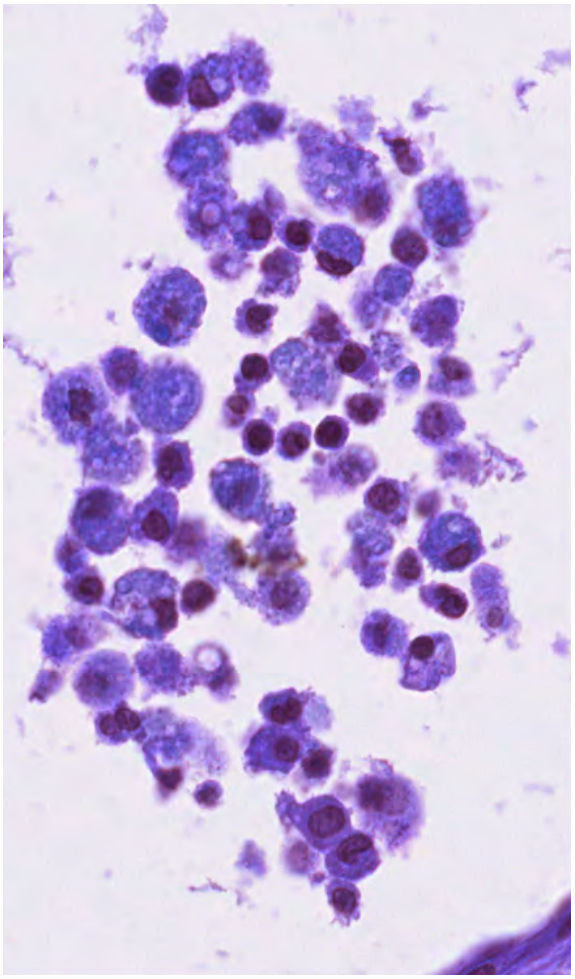
**Segmentation results :**

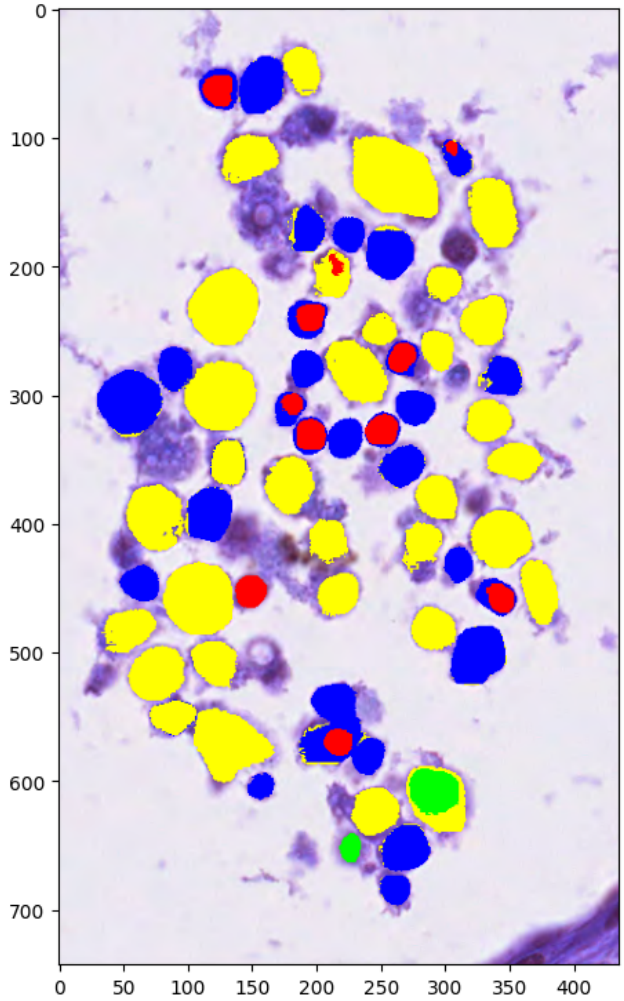







0
100
200
300
400
500
0
100
200
300
400
500

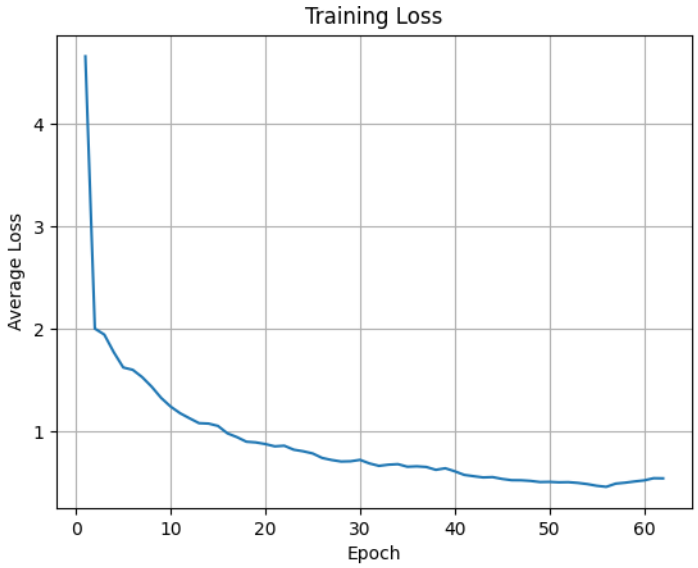



0
100
200
300
400
500
600
700
0
50
100
150
200
250
300
350
400

# 4. Additional experiments:

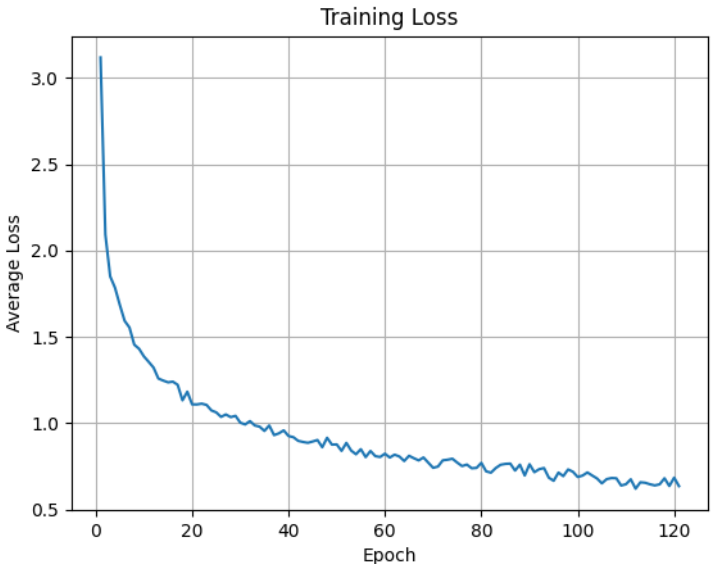
## 1. Different model architecture:

**seresnextaa101d_32x8d.sw_in12k_ft_in1k_288**




"private_score": 0.18711033514665062}

**resnetrs200.tf_in1k**




"private_score": 0.32893856806963334}

- **Hypothesis**: Testing different architectures could yield better performance depending on the model's ability to extract features and handle complex data like medical images.

- **How it works**: For this experiment, we explored several backbone architectures and their impact on the model's ability to segment cells. We chose to test seresnextaa101d_32x8d.sw_in12k_ft_in1k_288 and resnetrs200.tf_in1k.

### Result Analysis :

In this additional experiment, we tested two different model architectures: **resnetrs200.tf_in1k** (right) and **seresnextaa101d**(left). **resnetrs200.tf_in1k**, with its straightforward architecture of ReLU activations, a single 7x7 convolution layer with pooling, and a 1x1 convolution shortcut for downsampling, is optimized for efficiency in terms of parameters (93.2M) and GMACs (20.2). Despite its simpler design, it achieved a **private score** of 0.3289, indicating strong performance in the image classification task. The model's architecture is well-suited for generalization, allowing it to perform consistently well, even though its loss curve showed a slower decline during training.

On the other hand, **seresnextaa101d** utilizes a more complex architecture featuring Squeeze-and-Excitation (SE) channel attention, 3-layer stem convolutions, and grouped 3x3 bottleneck convolutions. While it has a similar number of parameters (93.6M) to **resnetrs200.tf_in1k**, its

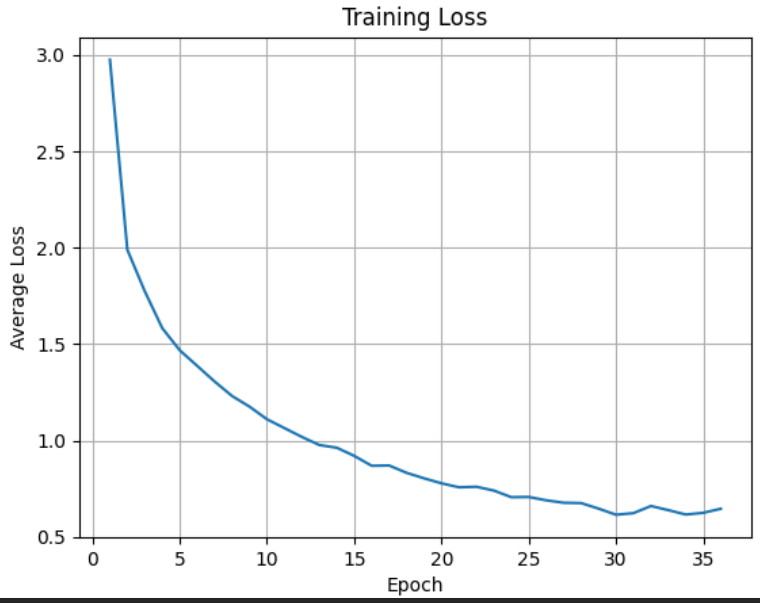
architecture is more computationally intensive, with higher GMACs (28.5) and activations (56.4M). Trained on a larger image size (288x288), the model's design includes a more sophisticated feature extraction mechanism through SE attention and bottleneck convolutions. Despite these advanced features, the **private score** of 0.1871 suggests that it didn't generalize as well as **resnetrs200.tf_in1k**, likely due to the increased complexity leading to overfitting. The faster loss convergence observed during training for this model didn't result in better performance on the private leaderboard.

In summary, **resnetrs200.tf_in1k** outperforms **seresnextaa101d** in this task, likely due to its more efficient architecture, which is better suited for generalization with the dataset at hand. Despite the sophisticated features of **seresnextaa101d**, its complexity may have been too high for this specific problem, leading to overfitting.

## 2. Data augmentation - Gaussian Blur, RandomRotation etc :

- **Hypothesis**: Incorporating multiple augmentation techniques would encourage the model to generalize better by simulating more diverse training conditions.
- **How it works**: Gaussian blur smooths image textures, random rotation alters the orientation, and random horizontal flip simulates mirrored views. These augmentations prevent the model from overfitting to specific spatial patterns.

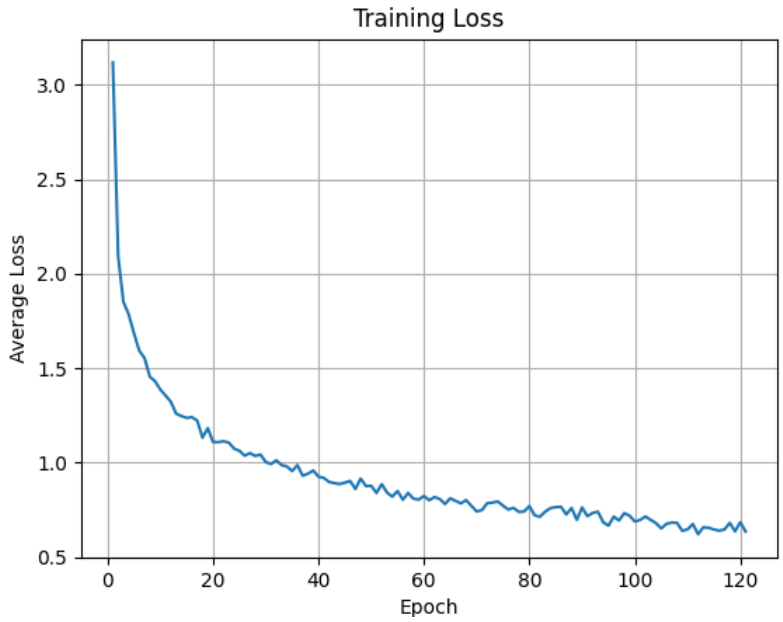
**Experiment results:**




public_score": 0.2611700503364249, "private_score": 0.3211018617853009}

The first learning curve above using model resnetrs200.tf_in1k "without" applying any of the techniques mentioned above reveals ends at epoch 36, corresponds to the model trained without data augmentation. The training loss shows a steep drop at the beginning, indicating that the model quickly

adapted to the data, but the reduction in loss becomes slower as it reaches a stable point, suggesting that further improvement may be limited




{"public_score": 0.287886122637029, "private_score": 0.32893856806963334}

The second learning curve obtained using the resnetrs200.tf_in1k model enhanced "with" all the techniques including dropout, data augmentation ends at epoch 121, shows the training process with data augmentation. It starts with a similar steep drop, but the loss decreases at a more gradual pace compared to the first plot. This indicates that data augmentation improved the model's generalization capability, allowing it to continue refining its weights over a longer training period, eventually reaching a more stable and lower loss. The results suggest that data augmentation had a positive impact on the model's ability to generalize to unseen data, as seen in the improvement of both public and private scores.

## References:


1. **Mask R-CNN:** He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*. (https://arxiv.org/abs/1703.06870)
2. **FPN (Feature Pyramid Networks):** Lin, T.-Y., et al. (2017). Feature Pyramid Networks for Object Detection. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR). (https://ieeexplore.ieee.org/document/8099589)
3. **Run-Length Encoding (RLE):** COCO dataset documentation. (https://cocodataset.org/#format-results)