INM705: Deep Learning for Image Analysis

# Instance Segmentation using Mask-RCNN

Coursework

2021-22

Alex Collins

Priyanka Velagala

# Table of Contents

# 1.0 Introduction

## 1.1 Background (Priyanka Velagala)

Instance segmentation is a problem in computer vision that comprises two subtasks - object detection and semantic segmentation. The first, object detection, focuses on classifying and localization of objects, whereas the second, semantic segmentation, assigns a category label to each pixel in an image(Hafiz & Bhat, 2020). Combining the two techniques simultaneously allows us to both isolate and classify each instance of an object in an image and perform instance segmentation(Hafiz & Bhat, 2020).

Some key challenges in this task are detection of small objects, where due to their size, the features extracted by a CNN in the early layers are already limited and can fully disappear in deeper layers of the network (Hafiz & Bhat, 2020). Another being occlusion of objects which results in loss of information thereby making instance segmentation inherently more difficult(Hafiz & Bhat, 2020). And finally class imbalance which is when there is a "*significant inequality*" in the number of training examples for each class (Hafiz & Bhat, 2020).

We can classify the current state of the art models that tackle instance segmentation into 3 broad categories (Di et al., 2021):

I. Methods based on detection - this technique finds the region of the instance of an object through object detection and then predicts a mask in the detection area (e.g. Mask R-CNN , PANet, HTC etc.) (Di et al., 2021).
II. Methods based on pixels - this technique predicts the class label for each pixel and then groups them via clustering technique to form an instance segmentation (e.g. SGN, SSAP, Discriminative etc. )(Di et al., 2021). Models using this method have a higher computational complexity in comparison.
III. Single shot instance segmentation - this technique first generates a set of "prototype" masks and a mask coefficient and then linearly combines the two to generate an instance mask (e.g. YOLOACT, PolarMask, SOLO etc.) (Di et al., 2021). Models that rely on this technique have a relatively low computational complexity.

For our coursework, with consideration of architectures in scope for this course, we have chosen to work with the Mask R-CNN.

## 1.2 Dataset

Our choice of dataset was motivated by one of the key problems discussed earlier - class imbalance. We chose to work with the Large Vocabulary Instance Segmentation (LVIS) dataset which is designed to encourage exploration of solutions to address this problem specifically. The dataset has ~2 million high-quality instance segmentation masks for 1203 object categories over ~160k images(Gupta et al., 2019). The images used in this dataset are the same as those from the benchmark COCO dataset

however, the segmentation masks improve on the quality of the masks in COCO which are relatively more coarse.

Additionally, the dataset also labels each category as frequent, common or rare based on the number of images with annotations of a specific category (see Table 1.2 for details).

| Category label | Number of annotated images |
|---|---|
| Frequent | >100 |
| Common | 11-100 |
| Rare | 1-10 |

Table 1.2: Classification of categories in LVIS by annotated images(Gupta et al., 2019)

Due to time constraints and limited computing resources we chose to work with a subset of the available data. We picked 2 categories from each of the frequent, common and rare categories. As the ground truths for the standard test set is not publicly available, we used a 9:1 split on the official train set to obtain a train and validation set for training purposes and used their official validation set for testing purposes.

## 1.3 Objectives

Our objective for this project is to use the Mask R-CNN for object detection in long-tailed categories in the LVIS dataset. Our motivation in using the Mask R-CNN lies in the improved performance it has over the Faster R-CNN due to preservation of spatial features through improved pixel-to-pixel alignment.

## 1.4 Code References

While the source code submitted was written from scratch, a full list of references that inspired our implementation and guided us in following best practices can be found in Appendix A of this document.

# 2.0 Data Preprocessing

In this section, we will briefly describe the steps taken in data preprocessing. We will begin by addressing the notion of a federated dataset introduced by LVIS to overcome the problem of having to exhaustively annotate every instance of each category present in each image and then discuss steps taken for image preprocessing.

## 2.1 Federated Dataset

LVIS introduces the concept of Federated Dataset due to two main issues(Gupta et al., 2019):

Issue #1: Due to the larger number of categories captured in LVIS, this presents some ambiguity during evaluation when an object can have more than one label. For example, an object can be both a book and a notebook. However, if the ground truth annotation doesn't have both (or all correct) labels, the model may incorrectly penalize the prediction.
Issue #2: To exhaustively annotate and produce a high quality segmentation of each instance of each category in each image can be challenging in cases when it's ambiguous

The federated dataset ($\mathcal{D}$) for a class, c is the union of two disjoint sets(Gupta et al., 2019):

1. Positive set ($P_c$) - the set of images containing labeled instances of a c.
2. Negative set ($N_c$) - the set of images where (it was verified) no instances of c exist

$$\mathcal{D} = \cup_c(\mathcal{P}_c \cup \mathcal{N}_c)$$

This overcomes issue 1 as for each object, the model's ability to detect that object are only evaluated on the positive and negative set for the class, c and detection of any of the other classes on that set are not. In defining disjoint sets this also ensures that we don't incorrectly penalize predictions of a class on an image that is neither part of the positive or the negative set and also disregard any cases when the segmentation mask is ambiguous.

The LVIS dataset also defines a flag when certain classes present in an image are not exhaustively annotated and dictates a procedure for handling those cases. For our purposes, we deliberately exclude images which have the flag set for our classes of interest.

## 2.2 Image Data Preprocessing

To allow training with a batch size > 1, we performed transformations on the image data to ensure all images are of the same height and width. In contrast to the the parameters used in training of the Mask R-CNN by He et al. (2018) where image as big as 800x1024 were used for training, with consideration of our limited computing resources, we chose to work with a much smaller and manageable size of 480 x640 which allowed us to train with a batch size of 10(Matterport, n.d.). This required that first the images be scaled up or down to the chosen dimensions preserving their aspect ratio and any remaining area padded on the bottom and right sides . The respective bounding boxes and masks of the training samples were also scaled in a similar manner.

# 3.0 Model Design

In this section, we provide an overview of model architecture for Mask R-CNN. We will also justify our choice of hyperparameters, loss functions and optimizers used and cover other pertinent aspects of model training.

## 3.1 Model Architecture

Mask R-CNN is an extension of the Faster R-CNN model with an additional branch that predicts segmentation masks. Here, we identify the main components of the model, define their inputs and outputs and describe their functionality.

### 3.1.1 Backbone Network with FPN

The first component of the model is the backbone network, where the preprocessed images are passed into the CNN (ResNet50) to extract key features from an image and generate feature maps. The early layers of the network which have access to a high resolution image detect broad representations of objects, as deeper layers of the network learn features that are semantically stronger(Lin et al., 2017). However, as image resolution declines with each successive layer, the semantically strong features are learned on low resolution images. In addition, as objects can vary in scale, the combination of two factors leave room for improvement that is addressed by the FPN network.

The FPN extends the backbone by introducing two pathways(Lin et al., 2017):

1. The bottom up pathway which is the feedforward CNN network
2. The top down pathway which generates higher resolution features that are semantically strong by using lateral connections from the bottom up path way where feature maps of the same size are merged from the bottom up pathway to the top down pathway

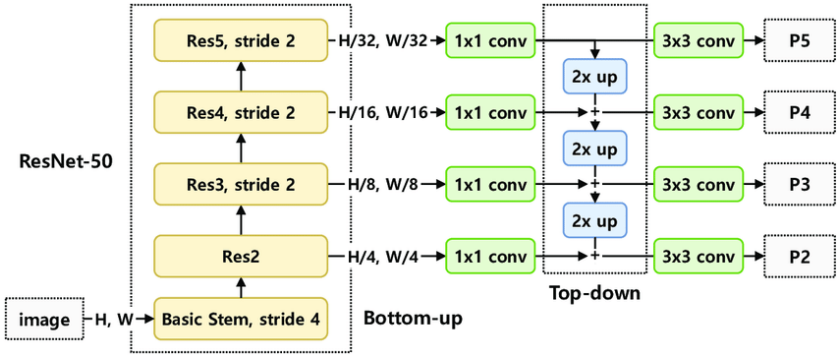


Figure 3.1 Architecture of backbone Network - ResNet50 with FPN (Lee et at, 2021)

For our implementation, we used ResNet50 with an FPN network where the backbone was pretrained on the ImageNet dataset and the FPN on COCO 2017 (Lee et at, 2021). The COCO 2017 dataset has 80 categories which are a subset of the larger 1200+ categories annotated in the LVIS set.

### 3.1.2 Region Proposal Network

The main purpose of the Region Proposal Network is to generate regions of interest (ROI). A set number of anchors are projected on an image where for each anchor, k boxes ("anchor boxes") are generated using varying scales and aspect ratios. In the case of Faster R-CNN, a common default is to use a total of 9 boxes generated for each position, with 3 scales and 3 aspect ratios (Ren et al., 2016). The purpose of

the RPN then remains that for each anchor, it predicts the objectness score associated with the anchor (i.e. how likely it is to be an object vs. the background) and the offsets (x,y,w,h) of the region proposal with respect to the position of the anchor.

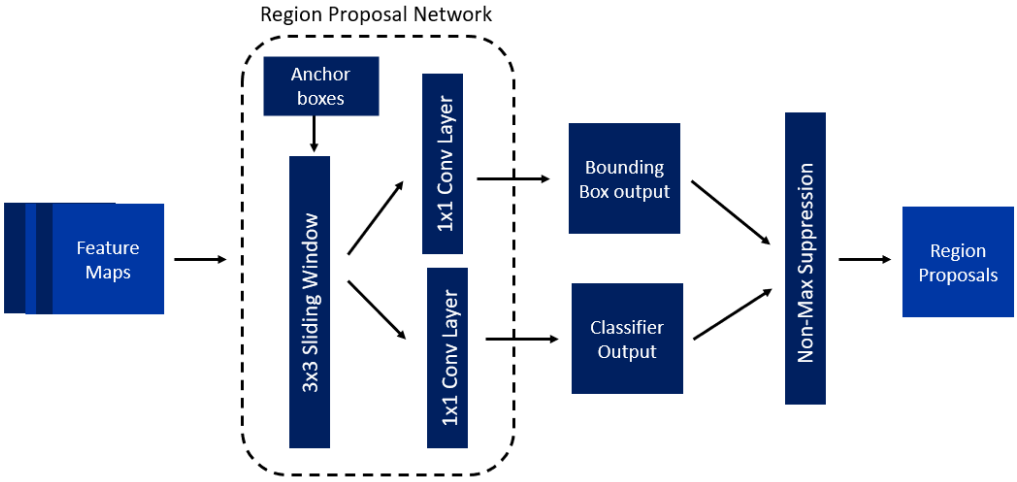


Figure 3.2 : Region Proposal Network (RPN)

To train the network using gradient descent, we need to pick some ROI and subsequent class predictions, bounding boxes and segmentations to calculate loss. Anchors are assigned either a positive label (i.e. contains an object) or a negative label (i.e. part of background) based on whether the max IOU of the ground truth bounding boxes and the anchor is greater than at least 0.7 or less than 0.3 respectively (0.7 and 0.3 are hyper-parameters so can be changed).

If the image feature map is 40 x 60, and we have k = 9 anchor boxes per anchor point, we would have 40 x 60 x 9, or ~20k predictions. As proposals at this stage can yield many overlapping region proposals, non-maximum suppression (NMS) is used to minimize the number of region proposals passed further along the network: the RPN network outputs binary classifications and we asses the top n classifications (another hyper parameter), with n = 2000 being a common value.

### 3.1.3 Faster R-CNN

The Faster R-CNN then uses the region of interest (ROI) proposals from the RPN and the feature maps from the backbone network to extract a fixed length feature vector which is used as input for the subsequent fully-connected layers. For this, as feature maps are much smaller in scale than region proposals, the proposals need to be scaled down to the feature map level. This is performed through ROI Align which in contrast to ROI Pooling has better performance as stride isn't quantized which means better alignment of ROI and extracted features and thus higher the quality of masks generated.

The output feature vector is then passed into the detection network composed of fully connected layers, where the network is jointly trained on:

1. A regression problem to predict the bounding boxes of objects in the image

2. A classification problem to accurately predict the class associated with the bounding box.

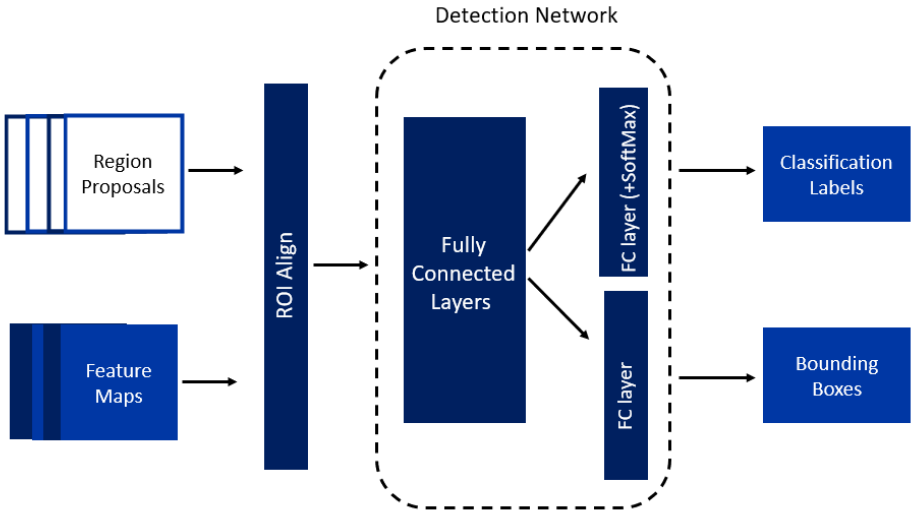


Figure 3.3: Detection Network in Faster R-CNN

### 3.1.4 Mask Head

The output of the ROIAlign Layer from Figure 3.3 which preserves the spatial locations of features is then passed into a Fully Convolutional Network (FCN). The FCN predicts a binary mask for each class independently. As the Faster-RCNN handles the class prediction, in the FCN there is no competition among the output classes for each pixel: masks are predicted for every class and the loss is only calculated for the mask corresponding to the correct class. The mask head predicts masks of a standard size: *m* x *m* x *K*, where *K* is the number of classes. After the masks have been generated they are resized to align with the true dimensions of the image.

To calculate pixel by pixel predictions on the mask, the ROI align layer replaced the ROI pooling layer in faster RCNN. ROIAlign avoids quantization and as a by-product improves estimation of object bounding boxes. (It is this feature of the model that chose us to choose mask-RCNN for object detection rather than faster-RCNN.

## 3.1.2 Loss Functions

As the network is jointly trained, the loss is the sum of 5 distinct losses: 2 for the RPN stage and 3 known as the multi-task loss.

$$L = L_{obj} + L_{obj\ box} + L_{cls} + L_{box} + L_{mask}$$

### 3.2.1 Loss Function for RPN

The RPN loss is comprised of two components:

$$L = L_{obj} + L_{obj\ box}$$

$L_{obj}$ is the class loss and is binary with predictions being either (any) object or background. If the anchor box is labeled as positive, then an $L_1$ loss is also calculated for the ROI object bounding box prediction. This is compared to the ground truth bounding box of the most likely object (the one with highest IOU with the anchor).

### 3.2.2 Loss Function for Classifier, Box Regressor and Mask Head in Faster R-CNN

The multi-task loss function for each ROI broken down as classification, box and mask loss function as follows:

$$L = L_{cls} + L_{box} + L_{mask}$$

$L_{cls}$ is the class loss, and is calculated for positively labeled ROIs. The model outputs an K+1 probabilities, for the K classes + background class, and a cross entropy loss is used to compare to ground truth, summarized by the equation below.

$$l_n = -w_{y_n} \log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^{C} \exp(x_{n,c})}$$

$L_{box}$ is the bounding box loss and is calculated as a regression loss given that the objective is to minimize the magnitude of error. It is calculated using a smooth L1 loss:

$$l_n = \begin{cases} 0.5(x_n - y_n)^2/beta, & \text{if } |x_n - y_n| < beta \\ |x_n - y_n| - 0.5 * beta, & \text{otherwise} \end{cases}$$

Finally, $L_{mask}$ is the mask loss. Masks will be generated as *m* x *m* pixels for *K* classes. Loss is only calculated for the ground truth class and is the average of per-pixel binary cross entropy loss.

$$l_n = -w_n \left[ y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right],$$

## 3.3 Optimizers

During training, for model optimization we used mini-batch stochastic gradient descent with momentum to minimize the multi-task loss specified in section 3.2. We use a learning rate scheduler to control the rate at which the network parameters are updated (see Table 3.4.3 for values used) and using mini-batches allows the network to update parameters more frequently.

## 3.4 Hyperparameters

The following tables list hyper-parameters values used by different components of the network . Only those explicitly set to a value different than the default value have been listed:

| Image Dimensions | Batch Size | FCN layers dimension |
|---|---|---|
| 480 x640 | 10 | 256 |

Table 3.4.1: Model Hyper-parameters

| Learning rate | Momentum | Weight Decay |
|---|---|---|
| 0.005 | 0.9 | 0.005 |

Table 3.4.2: Hyperparameters used for SGD Optimize

| Step Size | Gamma |
|---|---|
| 3 | 0.1 |

Table 3.4.3: Hyperparameters used for Learning Rate Scheduler

# 4.0 Model Evaluation

## 4.1 Evaluation Metrics: mAP

We use mAP as our principal evaluation measure. We calculated it by hand - see metric.py in our code - and adapted the standard method to fit with the LVIS approach to federated data sets. For each class, we measure AP on a *subset* of the test data: that subset being the union of the positive and negative data sets for that class. We do this for IoU thresholds of 0.5 to 0.95 in increments of 0.05. The overall mAP is the mean of average precisions, which we average across categories. It should be noted that it is hard to compare our mAP with benchmarks or even between our testing on different datasets. For instance, we start with a training set of 500 images comprising two frequent, two common and two classes. Adding an extra three rare classes, only increases the dataset size to 530. Rare classes are much harder to train

effectively and one would expect lower mAP, but when there is a higher proportion of rare classes, the mAP evaluation results will get worse.

To help address this, as well as the overall number, we present per category AP for different IoU thresholds, to gain a better understanding of where the model is performing well and where it is not.

Note that in line with our objective of object detection, our mAP metric is for bounding box precision. We do not calculate mask AP.

## 4.2 Results

We ran 2 experiments with multiple models each to evaluate how class imbalance affects model performance.

### 4.2.1: Experiment 1

The purpose of this experiment was to evaluate how using a pretrained model vs. untrained model performs on a dataset with class imbalances. For testing and evaluation, we choose to work with a subset of the full LVIS dataset and used the following categories from each frequency class:

| **Frequency Class** | **Frequent** | **Common** | **Rare** |
|---|---|---|---|
| **Categories** | Coffee-maker,<br>Cowboy hat,<br>(Tennis racket,<br>skateboard)* | Monkey,<br>cappuccino | Chessboard,<br>Drumstick,<br>( horse_buggy, armor,<br>smoothie)** |

* an additional two (very) frequent categories were added to the dataset for Baseline+++ only
* an additional three rare categories were added to the dataset for Baseline++ only

Table 4.2.1.1: Categories used by Frequency Class in Experiment 1

For this experiment, the models differed in that where the baseline model only had a pre-trained feature extractor, the baseline+, baseline++, baseline +++ models had a fully pre-trained backbone.

| | **Feature extractor** | **Feature Pyramid Network** |
|---|---|---|
| **Baseline** | Pre-trained ResNet50 on ImageNet | No |
| **Baseline+,<br>Baseline++,<br>Baseline+++** | Pre-trained ResNet50 on ImageNet | Pre-trained FPN on COCO2017 |

Table 4.2.1.2 : Description of models evaluated in Experiment 1

The models were trained until validation loss was non-decreasing. The performance of the Baseline and Baseline+ model on each of the frequent, common and rare categories can be seen below in Figure 4.2.1-1.

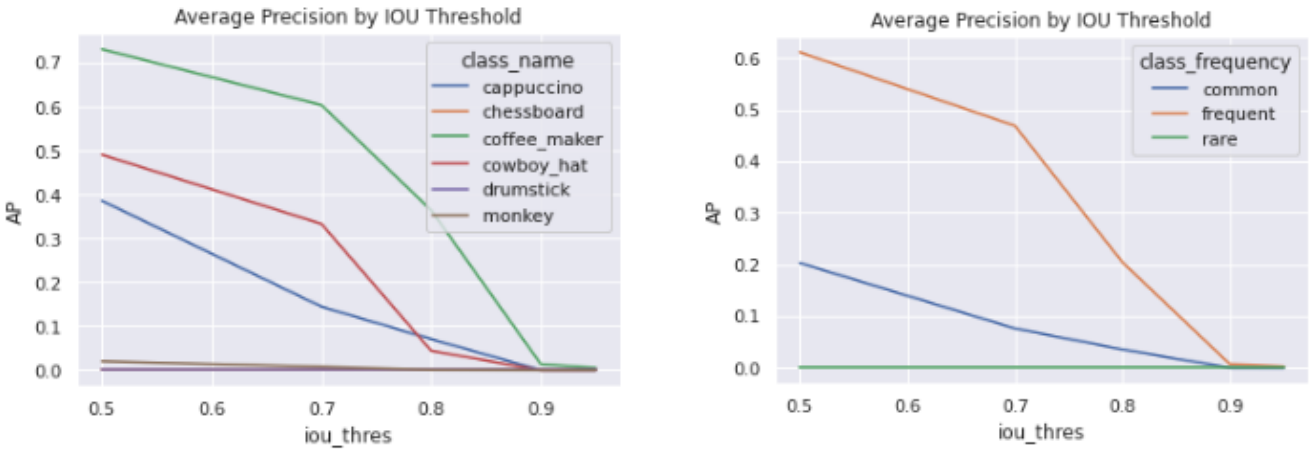


Figure 4.2.1-1: Average Precision by IOU threshold of **Baseline** model by Frequency Class

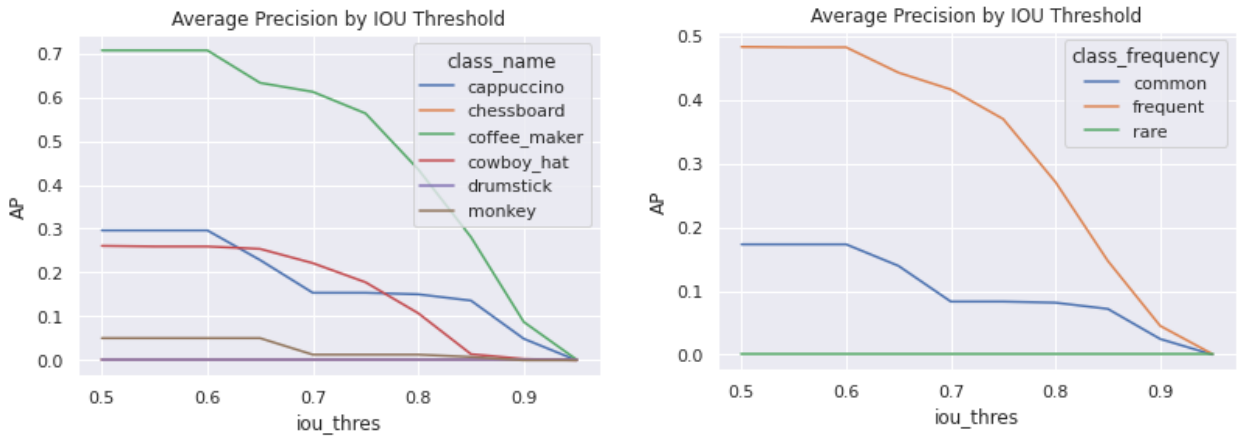


Figure 4.2.1-2: Average Precision by IOU threshold of **Baseline+** model by Frequency Class

By observing the AP between each frequency class in the Baseline+ model, we notice that the less represented a class is during training, the less likely the model is able to detect it during inference. This is consistent with expectations as the object representations of the other classes would have more heavily influenced model parameters during training and therefore during inference the model will have learned a poor representation of the categories that are rare. It is interesting to note on the graph on the left, cowboy hat, which is a frequent category and monkey, which is a common category, have a roughly equal AP at all thresholds, from which we could infer that the size of an object could also influence this metric.

| | mAP (AP@0.5): **frequent** | mAP (AP@0.5): **common** | mAP (AP@0.5): **rare** |
|---|---|---|---|
| Baseline Model | 0.213 (0.54) | 0.006 (0.02) | 0.000 (0.000) |
| Baseline+ Model | 0.370 (0.593) | 0.164 (0.287) | 0.000 (0.000) |
| Baseline++ Model | 0.384 (0.603) | 0.142 (0.237) | 0.012 (0.017) |
| Baseline+++ Model | 0.441 (0.707) | 0.067 (0.222) | 0.000 (0.000) |

Table 4.2.1-21: mAP and AP@IOU =0.5 for models

The table above shows the mAP and AP@IOU=0.5 of the Baseline, Baseline+, Baseline++ and Baseline+++ broken down by frequency class. It can be noted that unlike the first two models, the Baseline++ model is able to detect rare categories with roughly 1% accuracy and generally tends to perform better across all frequency classes.

We included Baseline+++ to see how a much bigger training set affected results. The previous models had ~450 images in the train/val datasets, and including tennis racket and skateboard categories increased the training set to 4500. mAP for the frequent category did improve substantially in Baseline+++ indicating that volume of training data is important. (However we cannot discount the possibility that skateboard and tennis racket performed so well because they are also MS COCO categories and one would expect pre-trained weights to be highly effective already.) Baseline +++ also had the effect of making common and rare categories even rarer relative to the entire dataset, and performance for common categories deteriorated as a result.

## 4.2.2: Experiment 2

The purpose of this experiment was to evaluate how changing the sizes of the anchor boxes in the RPN affects performance on a dataset with class imbalances. For this experiment, we chose to work with a different set of categories, where the category objects are small in size. We hypothesized that by changing the size of the anchor boxes to be smaller, model performance would improve across all classes.

For this experiment we choose to work with the categories listed in Table 4.2.2.1. The anchor sizes and backbone settings that were used for each of the models is listed in Table 4.2.2.2 , followed by the results obtained during evaluation in Table 4.2.1.3.

| **Frequency Class** | **Frequent** | **Common** | **Rare** |
|---|---|---|---|
| **Categories** | Grape, crumb | Pea, peach | Apricot, crouton |

Table 4.2.2.1: Categories used by Frequency Class in Experiment 2

| | **Anchor Sizes in RPN** | **Backbone** |
|---|---|---|
| **Baseline** | (128, 256, 512) | Pretrained ResNet50_FPN |
| **Baseline+** | (2,4,8,16, 32, 64) | Pre-trained ResNet50_FPN |

Table 4.2.2.2 : Description of models evaluated in Experiment 2

| | mAP (AP@0.5): **frequent** | mAP (AP@0.5): **common** | mAP (AP@0.5): **rare** |
|---|---|---|---|
| Baseline Model | 0.159 (0.275) | 0.158 (0.275) | 0.008 (0.010) |
| Baseline+ Model | 0.01 (0.025) | 0.03 (0.093) | 0.000 (0.000) |

Table 4.2.2.1: mAP and AP@IOU =0.5 for Baseline and Baseline+ model for Experiment 2

In this experiment as we intentionally picked small objects, the baseline+ model's anchor sizes were drastically scaled down. However based on the results obtained, this yielded a worse accuracy across all frequency classes proving our hypothesis was incorrect.

### 4.2.3 Overfitting

An issue common to all our model variations was over-fitting. We consistently observed that validation performance would plateau after 5-15 epochs, despite continual improvement on the training set. We hypothesized that small training sets of ~500 images led to more overfitting risk, but when we added two very frequent categories (which increased the training set to ~4500 images) we still experienced over-fitting.

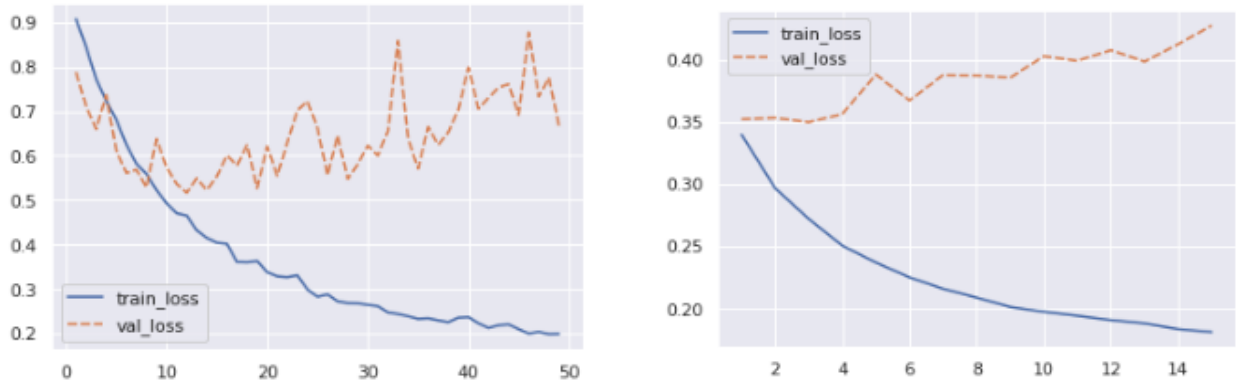


Figure 4.2.1-3: Training and Validation loss vs epoch. Left hand chart shows **Baseline** from experiment 1 (~450 images). Right hand chart shows **Baseline+++** from experiment 1 (~4500 images)

# 5.0 Conclusion (Alex Collins)

In this work, we explore a challenging long-tailed object detection and instance segmentation task. We evaluated model performance using mAP on object bounding boxes.

To make transfer from MS COCO pre-trained models, we selected classes that generally do not appear in the MS COCO class list. Despite this we found that transfer of FPN weights pre-trained on the COCO dataset was an effective way to improve performance.

Given the compute and time constraints, we trained on a very small subset of the data. Charts of train and val loss demonstrate evidence of over-fitting. When we increased the train/val dataset size from ~500 to ~4500 images we saw the same evidence of overfitting on train data, but did see an increase in performance at evaluation.

Presence of rare classes in the test set was very sparse. This made it hard to evaluate performance on rare classes, so we increased rare classes from 2 to 5. This gave us a better estimate of performance and demonstrated just how challenging detection is for rare classes.

Objects in the LVIS vocabulary can be very numerous and very small in images. We experimented with using very small anchor boxes, with the smallest scale being 2. This was a failure. A limitation could be that resolution of anchor points was not fine enough to allow the smallest anchor boxes to cover the image: if there wasn't overlap between where the small anchor boxes happened to be and where small objects were, the model would lack training experience on positive object matches.

Our results show evidence that unbalanced data creates a significant challenge to effective inference on rare classes, and our model is not effective at dealing with the problem. There are many potential extensions. To name just a few:

- Upsampling of rare points, using image transforms such as flip, rotate and jitter.
- Another upsampling approach is data augmentation using Mosaic augmentation introduced by Bochkovskiy et al, 2020. Training images are combined in such a way so that familiar objects are seen in surprising contexts.
- When an object is detected, the sigmoid cross entropy loss for classification can lead to a "negative" signal for categories that are not present. This leads to suppression of rare categories. The loss function could be modified to avoid this such as in (Tang et. al, 2019).

# 6.0 Reflections

## 6.1 Dataset

Our principal challenge was working with a more complex dataset. The images are the same as coco, and we just needed to merge the train and validation sets: LVIS defines the data splits in the annotation files.

The Federated Dataset approach in LVIS is reasonably simple in concept, but created design challenges which we had to overcome:

- How to deal with the non-exhaustive flag?
  The model should not evaluate loss for a class when that class is non-exhaustively annotated in that image. We decided to exclude all images where any of our chosen classes were not exhaustively annotated. This meant we didn't have to write custom code within the loss function.
- How to deal with negative sets in training and evaluation?
  When training on the full dataset, all images have class annotations and all images are in the negative set for some classes. In our case, if we included the full negative set for all our classes of interest, many images would have no annotations. The Mask RCNN PyTorch model cannot cope with this as it expects at least one annotation in the ground truth. Initially we created a dummy background bounding box and mask, but then realized that we could probably exclude images with no annotations for our chosen classes: there was still plenty of data between and inside images where the model could train on predicting background.
- Calculated mAP
  We had to modify the mAP calculation so that each class was evaluated against its custom dataset: the union of positive and negative images. We included negative sets in the test data, and so many test images did not have any annotations.

## 6.2 Sampling of the dataset

We are aware that resampling of rare classes could improve inference on these classes at test time. We did explore transforming images containing rare classes to create more training data. Given time constraints we did not implement transforms such as Jitter and Horizontal Flip: we were not confident that we could transform the annotations consistently in the time available.

## 6.3 Understanding and implementing mask RCNN framework

Understanding the concepts and working with the torchvision was a bit like peeling an onion. Every time we thought we understood it, we realized that there were holes in our understanding. However, by the end of the project we have a pretty good understanding of all aspects of the network, from the

backbone to the FPN, to the class, box and mask heads. We spent time navigating the source code and understanding how to view and change aspects of it such as the backbone, and ROI parameters. Many papers such as the 2019 LVIS competition winner (Tang et. al, 2019) and 2020 LVIS competition winner (Tan et. al, 2022) modified loss functions and we need to learn how to do that to improve models for a specific task.

## 6.4 Evaluation

We enjoyed writing the custom mAP calculation and are proud of that. Given that we chose a model that is capable of both object detection and instance segmentation, we would have liked to write code to calculate mask mAP as well. We ran out of time, and would do that as a next step for future work.

# 7.0 References


Bochkovskiy, A., Wang, C.Y. and Liao, H.Y.M., 2020. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*.

Hafiz, A. M., & Bhat, G. M. (2020, June 28). *A survey on instance segmentation: State of the art*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/2007.00047

Oksuz, K., Cam, B. C., Kalkan, S., & Akbas, E. (2020, March 11). *Imbalance problems in object detection: A Review*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/1909.00169

Di Tian, Yi Han, Biyao Wang, Tian Guan, Hengzhi Gu, and Wei Wei "Review of object instance segmentation based on deep learning," Journal of Electronic Imaging 31(4), 041205 (6 December 2021). https://doi.org/10.1117/1.JEI.31.4.041205

Gupta, A., Dollár, P., & Girshick, R. (2019, September 15). *LVIS: A dataset for large vocabulary instance segmentation*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/1908.03195

Matterport. (n.d.). *Matterport/MASK_RCNN: Mask R-CNN for object detection and instance segmentation on Keras and tensorflow*. GitHub. Retrieved May 22, 2022, from https://github.com/matterport/Mask_RCNN

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2017, April 19). *Feature Pyramid Networks for Object Detection*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/1612.03144

Lee, nwoong , Do*young, K., Wee, D., An efficient human instance-guided framework for Video action recognition*. (n.d.). Retrieved May 22, 2022, from https://www.researchgate.net/publication/356995563_An_Efficient_Human_Instance-Guided_Framework_for_Video_Action_Recognition

*Models and pre-trained weights* Models and pre-trained weights - Torchvision 0.12 documentation. (n.d.). Retrieved May 22, 2022, from https://pytorch.org/vision/stable/models.html

Ren, S., He, K., Girshick, R., & Sun, J. (2016, January 6). *Faster R-CNN: Towards real-time object detection with region proposal networks*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/1506.01497

Pytorch. (n.d.). *Vision/roi_heads.py at main · Pytorch/Vision*. GitHub. Retrieved May 22, 2022, from https://github.com/pytorch/vision/blob/main/torchvision/models/detection/roi_heads.py

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2018, January 24). *Mask R-CNN*. arXiv.org. Retrieved May 22, 2022, from https://arxiv.org/abs/1703.06870

*Gonzales, S., Arellano, C., Farias, J. DeepBlueBerry: Quantification of blueberries in the wild using instance ...* (n.d.). Retrieved May 22, 2022, from https://www.researchgate.net/publication/334998795_DeepBlueBerry_Quantification_of_Blueberries_in_the_Wild_Using_Instance_Segmentation

Tan J., Wang, C. Li, Q. “Joint COCO and Mapillary Workshop at ICCV 2019: LVIS Challenge Track “ (2019) Retrieved May 22, 2022 from https://www.lvisdataset.org/assets/challenge_reports/2019/strangeturtle_equalization_loss.pdf

Tan J., Zhang, G. Deng, H, “Joint COCO and LVIS workshop at ECCV 2020: LVIS Challenge Track” (2020) . Retrieved May 22, 2022 from https://www.lvisdataset.org/assets/challenge_reports/2020/lvisTraveler.pdf

# 8.0 Appendix

## Appendix A - Code References

Dataset: https://www.lvisdataset.org/dataset

Code references:

[1] https://pytorch.org/tutorials/intermediate/torchvision_tutorial.html
[2] https://github.com/cocodataset/cocoapi/tree/master/PythonAPI/pycocotools