Project Report

MACHINE LEARNING

# Food Recognition with Deep Learning

Daniele Filippini - 953737
2020/2021
daniele.filippini2@studio.unibo.it

June 22, 2021


Alma Mater Studiorum - University of Bologna
Master's Degree in Computer Science

## Introduction

Recognizing food from images is an extremely useful tool for a variety of use cases. In particular, it would allow people to track their food intake by simply taking a picture of what they consume. Food tracking can be of personal interest, and can often be of medical relevance as well. Medical studies have for some time been interested in the food intake of study participants but had to rely on food frequency questionnaires that are known to be imprecise.

Image-based food recognition has in the past few years made substantial progress thanks to advances in deep learning. But food recognition remains a difficult problem for a variety of reasons.

For food recognition problems, it is common to use data sets such as Food101, composed of 101 food categories, with 101.000 images. However Food101 does not have images with various food classes, making it difficult to use in real-world settings [2].

For this reason a novel dataset was built for the *AICrowd Food Recognition Challenge*. This growing data set has been annotated - or automatic annotations have been verified - with respect to segmentation, classification (mapping the individual food items onto an ontology of Swiss Food items), and weight/volume estimation [1]. The dataset contains:

- Training set: with 24.120 food images, with their corresponding 39.328 annotations in MS-COCO format
- Validation set: with 1.269 food images, with their corresponding 2.053 annotations in MS-COCO format

Figure 1: An example of food image segmentation

This works aims to evaluate SOTA methods for segmentation and classification applied to AICrowd Food Recognition Dataset, with the objectives of:

- compare different approaches for image segmentation
- evaluate the efficiency of techniques such as Data Augmentation, Multi Scale Training, Test Time Augmentation
- propose a statistical analysis of the results

The reminder of the report is organized as follows:

- Section I Background: contains all the theoretical informations needed to understand the models and procedure adopted in the experimental study;
- In Section II Methodology: we describe the libraries used and the methodologies followed in the experimental study. Also, a visual exploration to understand what is in the dataset and the characteristics of the data is presented. Moreover, we present and motivate the preprocessing techniques applied to the data;
- Section III Experimental study: we explain which models have been tested, their hyperparameters and we analyse the results obtained in different experiments;
- Section IV concludes the report.

# Contents

# 1 Background

## 1.1 Object Detection with Deep Learning

Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs [3].

The first important task is called **localization**: Given an image with an object contained in a box GT (short for Ground Truth) we would like our model to predict a bounding box BB such that

$$IoU(BB, GT) > threshold$$

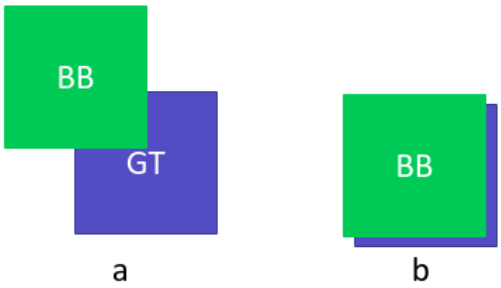



Figure 2: Intersection over union: (a) IoU=0.16, (b) IoU=0.9

Intersection over Union is an evaluation metric used to measure the accuracy of an object detector on a particular dataset.

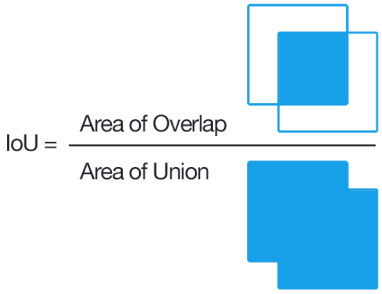



Figure 3: IoU equation

The second typical task is **classification**: Most detectors are also required to predict the class of the object in the bounding box.

*How are these tasks performed jointly?*

The input to a classification network is assumed to contain a single object. The classifier, except its last two layers, is composed of a structure called Feature Extractor (F.E.). There are many variations for the F.E. architecture (e.g. ResNet, DenseNet, Mobilenet etc.), but basically a F.E. is a sequence of convolution layers with activation functions producing multiple feature maps that flow down the network, and also there are spatial compression operations called max-pooling.

During training the F.E. learns to represent significant attributes in different feature maps. While the first layers are limited to representing only simple attributes like borders and simple shapes. The network learns to combine them in complex ways so that the final features of the F.E. can represent high-level attributes such as "this area has metallic texture", "this area has wheels", "this area has pointy ears", "this area has furry texture", etc.

The output of the feature extractor still preserves some spatial information, although at low resolution. The final segment of the classifier takes the final feature maps and replaces each map with its spatial-average value (an operation called global average-pool). This operation destroys the

spatial information and gives the average intensity of each attribute in the original image, regardless of location. The last layer is a fully-connected (F.C.) layer that learns the best coefficients for linearly combining the attribute intensities in a way that predicts the object class.

There are two main classes of problems:

- **Classification + Localization**: we want to classify the image and get the location of the object in the image. Here a fixed number of elements (typically 1) is assumed to be in the image.
- **Object detection**: it's a more general case of classification + localization, we don't know how many objects are in the image beforehand. We want to detect all the objects in the image and draw bounding boxes around them.

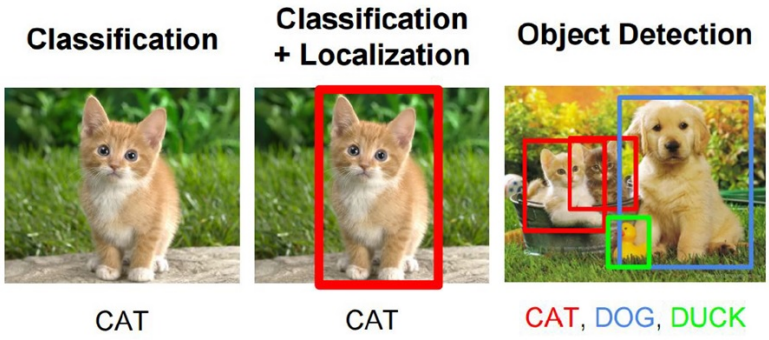



Figure 4: Classification, Localization and Detection

## 1.2 Detection Networks

The major reason why you cannot proceed with object detection by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable — not constant, this is because the number of occurrences of the objects of interest is not fixed [3].

A naive approach to solve this problem would be to take different regions of interest from the image, and use a CNN to classify the presence of the object within that region.

The problem with using this approach is that the objects in the image can have different aspect ratios and spatial locations. For instance, in some cases the object might be covering most of the image, while in others the object might only be covering a small percentage of the image.

As a result of these factors, we would require a very large number of regions resulting in a huge amount of computational time. So to solve this problem and reduce the number of regions, we can use Region-Based CNN, which selects the regions using a proposal method [4] [5].

### 1.2.1 Region Based CNN

To bypass the problem of selecting a huge number of regions, Ross Girshick et al. proposed a method where we use **selective search** to extract just 2.000 regions from the image and he called them **region proposals**, that are simply the smaller regions of the image that possibly contains the objects we are searching for in the input image.

Therefore, now, instead of trying to classify a huge number of regions, you can just work with 2.000 regions. These 2.000 region proposals are generated using the **selective search algorithm**. The steps of the algorithm are:

1. Generate initial sub-segmentation: we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals

These 2.000 candidate region proposals are warped into a square and fed into a convolutional neural network, that acts as a feature extractor. The extracted features are fed into an **SVM** to classify the presence of the object within the region proposal.

Finally, a **Bounding Box Regressor** (Bbox reg) is used to predict the bounding boxes for each identified region.

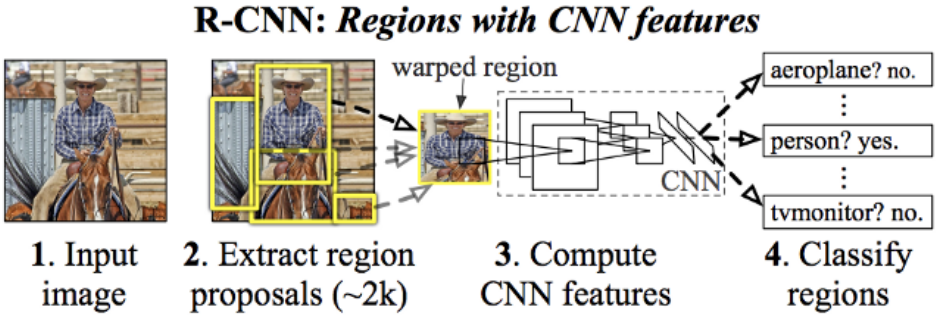



Figure 5: R-CNN model

The main problems of R-CNN are:

- It still takes a huge amount of time to train the network as you would have to classify 2.000 region proposals per image.
- It cannot be implemented real-time as it takes around 47 seconds for each test image.
- The selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage. This could lead to the generation of bad candidate region proposals.

### 1.2.2 Fast R-CNN

To solve the previous problems another object detection algorithm was developed, Fast R-CNN.

Instead of extracting CNN feature vectors independently for each region proposal, this model aggregates them into one CNN forward pass over the entire image and region proposals share this feature matrix. So, instead of feeding the region proposals to the CNN, we fed the input image to the CNN to generate a convolutional feature map.

For each region proposal we can identify the correspondingly part in the feature map and warp it into a square.

Using a RoI pooling layer we reshape them into a fixed size so that it can be fed into a fully connected layer.

**RoI pooling** is a type of max pooling to convert features in the projected region of the image of any size $h \times w$, into a small fixed window $H \times W$.

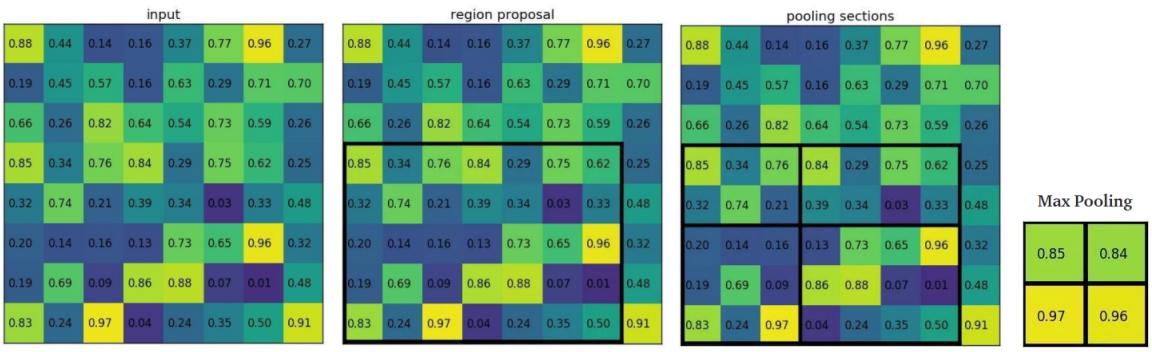



Figure 6: RoI pooling example

From the RoI feature vector a **softmax layer** is used to predict the class of the proposed region and also the offset values of the bounding box using a Bounding Box Regression model.

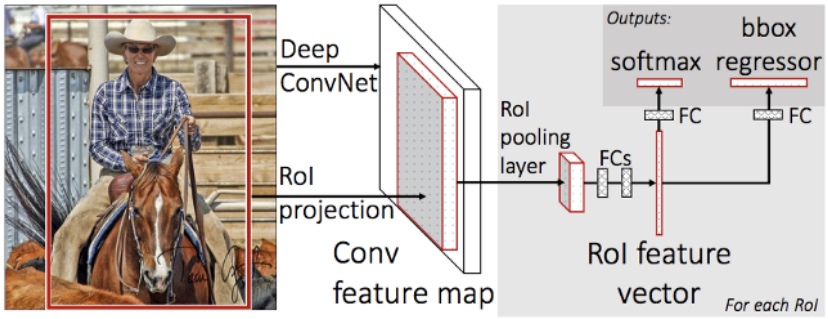



Figure 7: Fast R-CNN model

But even Fast-RCNN has certain problematic areas. It also uses selective search as a proposal method to find the Regions of Interest, which is a slow and time consuming process.

### 1.2.3 Faster R-CNN

Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. An intuitive solution is to eliminate the selective search algorithm and let the network learn the region proposals.

Similar to Fast R-CNN the image is provided as an input to a convolutional network which provides a convolutional feature map.

Then, instead of using the selective search algorithm, a separate network is used to predict the region proposals, this is called **Region Proposal Network** (RPN).

RPN uses a sliding window over these feature maps, and at each window, it generates K **anchor boxes** of different shapes and sizes. These boxes are defined to capture the scale and aspect ratio of the specific object classes to detect and are typically chosen based on object sizes in the training dataset.

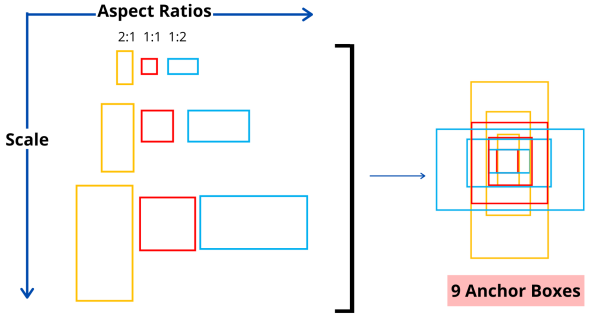



Figure 8: An example of anchor boxes

One drawback of using anchors is that each object gives rise to multiple bounding boxes from which we need to pick the best-fitting one.

An even more severe drawback is that since each spatial output cell is forced to predict bounding boxes even if the confidence is extremely low, we end up with hundreds or thousands of bounding boxes per image, by design.

For these reasons, for each anchor, RPN needs two things:

- First is the probability that an anchor is an object using a binary classifier

- Second is the Bounding Box Regressor for adjusting the anchors to better fit the object

**Non-Maximum Suppression** is typically used, it's a greedy algorithm that loops over all the classes, and for each class it checks for overlaps (IoU — Intersection over Union) between all the bounding boxes. If the IoU between two boxes of the same class is above a certain threshold (usually 0.7), the algorithm concludes that they refer to the same object, and discards the box with the lower confidence score (which is a product of the objectness score and the conditional class probability).

Finally, the predicted regions are reshaped using RoI pooling layer and the proposal are passed to a fully connected layer which has a softmax layer and a linear regression layer, to classify and output the bounding boxes.

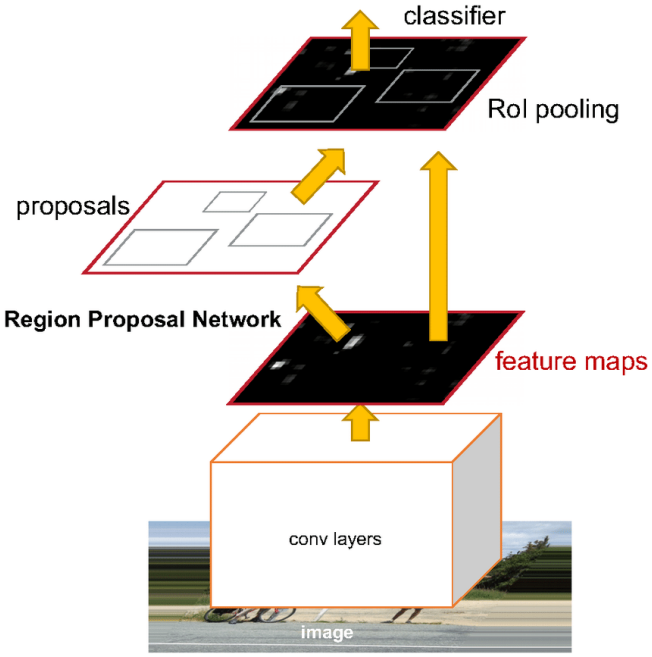



Figure 9: Faster R-CNN model

## 1.3 Segmentation

**Segmentation** is the task of classifying each pixel in an image from a predefined set of classes [7].

Two types of image segmentation exist:

- **Semantic segmentation** refers to the process of linking each pixel in the given image to a particular class label.
- **Instance segmentation** treats multiple objects of the same class as individual objects / separate entities.

In other words, semantic segmentation treats multiple objects within a single category as one entity. Instance segmentation, on the other hand, identifies individual objects within these categories.

To achieve the highest degree of accuracy, computer vision teams often build a dataset for instance segmentation.

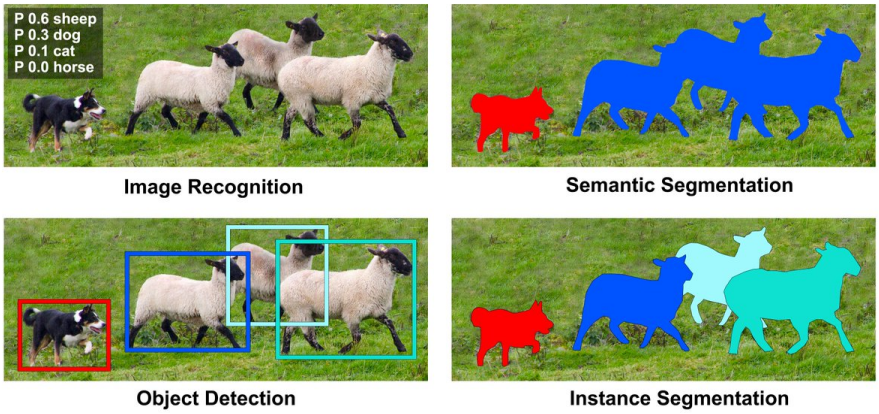



Figure 10: Different types of detection

### 1.3.1 Mask R-CNN

Mask R-CNN extends Faster R-CNN to pixel-level image segmentation. The key point is to decouple the classification and the pixel-level mask prediction tasks. Based on the framework of Faster R-CNN, it added a third branch for predicting an object mask in parallel with the existing branches for classification and localization.

Therefore, instance segmentation can essentially be solved in 2 steps:

1. Perform a version of object detection to draw bounding boxes around each instance of a class
2. Perform semantic segmentation on each of the bounding boxes

It works, because if we assume step 1 to have a high accuracy, then semantic segmentation in step 2 is provided a set of images which are guaranteed to have only 1 instance of the main class. The job of the model in step 2 is to just take in an image with exactly 1 main class, and predict which pixels correspond to the main class (cat/dog/human etc.), and which pixels correspond to the background of an image.

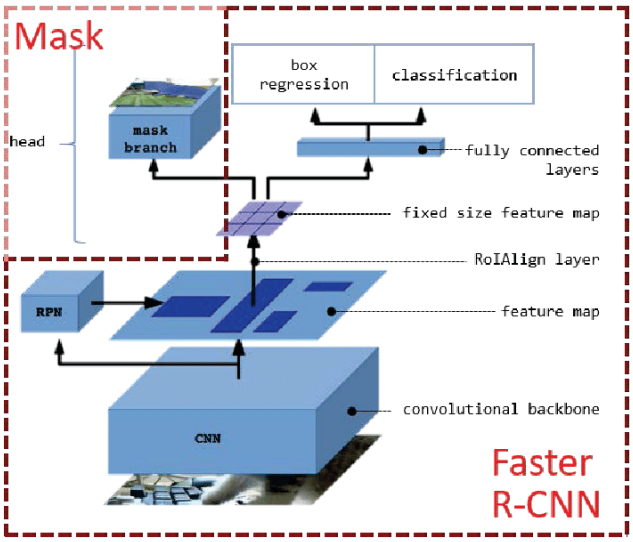



Figure 11: Mask R-CNN architecture

The **mask branch** is a small fully-connected network applied to each RoI, predicting a segmentation mask in a pixel-to-pixel manner.

When run without modifications on the original Faster R-CNN architecture, the Mask R-CNN regions of the feature map selected by the RoI pooling were slightly misaligned from the regions of the original image, leading to inaccuracies in the final result.

Because pixel-level segmentation requires much more fine-grained alignment than bounding boxes, mask R-CNN provides a new pooling layer named **RoIAlign**. This layer fixes the location misalignments caused by RoI pooling using *bilinear interpolation* to compute the floating point location values in the input.

**Loss Function**

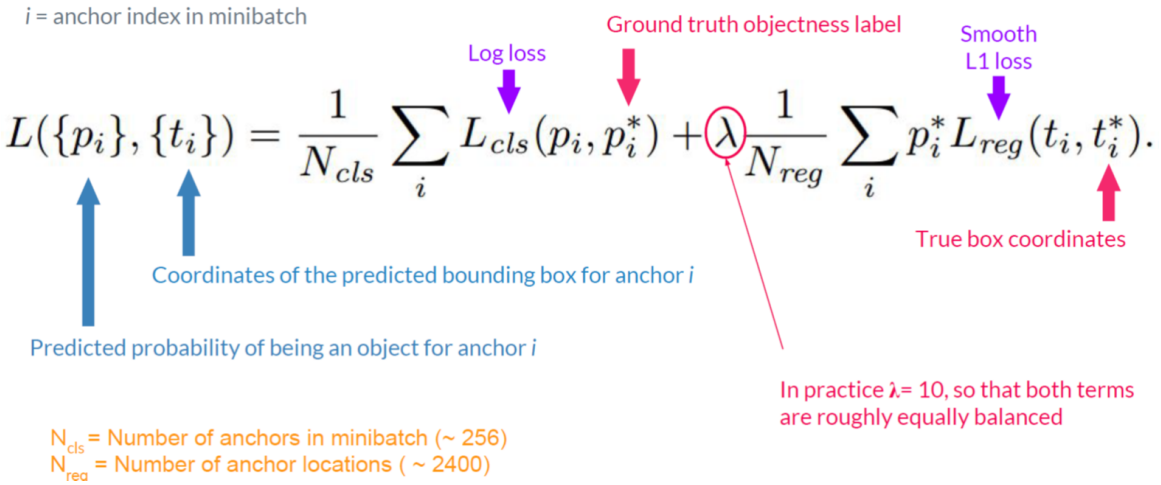



Figure 12: Faster RCNN Loss Function

where the classification loss is the log-loss over two classes (object vs non-object). $p_i$ is the output score from classification branch for anchor i, $p_i^*$ is the ground truth label (1 or 0).

$$L(p_i, p_i^*) = -log[p_i^* p_i + (1 - p_i^*)(p_i)]$$

The loss function of Mask-RCNN model is:

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

classification loss and box regression loss are the same as Faster-RCNN.

The mask branch generates a mask of dimension $m \times m$ for each RoI and each class. There are K classes in total, so the total output is $k \cdot m^2$. To this a per-pixel sigmoid is applied and the $L_{mask}$ is defined as the average binary cross-entropy loss, only including the k-th mask if the RoI is associated with ground truth class k (other mask outputs do not contribute to the loss).

$$L_{mask} = -\frac{1}{m^2}\sum_{1\le i,j\le m}[y_{ij} log\hat{y}_{ij}^k + (1 - y_{ij}) log(1 - \hat{y}_{ij}^k)]$$

where $y_{ij}$ is the label of a cell $(i, j)$ in the true mask fro the region of size $m \times m$, $\hat{y}_{ij}^k$ is the predicted value of the same cell in the mask learned for the ground truth class k.

### Hyperparameters in Mask R-CNN

There are several hyperparameters in Mask R-CNN models.

- **Backbone**: The Backbone is the Conv Net architecture that is to be used in the first step of Mask R-CNN. Options for Backbones include ResNet50, ResNet101, and ResNext 101. This choice should be based on the trade-off between training time and accuracy. ResNet50 would take relatively lesser time than the later ones, and has several open source pre-trained weights for huge data sets like coco, which can considerably reduce the training time for different instance segmentation projects. ResNet101 and ResNext101 will take more time for training (because of the number of layers), but they tend to be more accurate if there are no pre-trained weights involved and basic parameters like learning rate and number of epochs are well tuned.

- **Train ROIs per Image**: This is the maximum number of ROI's, the Region Proposal Network will generate for the image, which will further be processed for classification and masking in the next stage.
- **Max GT Instances**: This is the maximum number of instances that can be detected in one image. If the number of instances in the images are limited, this can be set to maximum number of instances that can occur in the image. This helps in reduction of false positives and reduces the training time.
- **Detection Min Confidence**: This is the confidence level threshold, beyond which the classification of an instance will happen. If detection of everything is important and false positives are fine, reduce the threshold to identify every possible instance. If accuracy of detection is important, increase the threshold to ensure that there are minimal false positive by guaranteeing that the model predicts only the instances with very high confidence.
- **Image Min and Max dimensions**: The image size is controlled by these settings. The default settings resize images to squares of size 1024x1024. Smaller images can be used (512x512) can be used to reduce memory requirements and training time. The ideal approach would be to train all the initial models on smaller image sizes for faster updating of weights and use higher sizes during final stage to fine tune the final model parameters.

Methods like GridSearch with cross validation might not be useful in cases of CNN because of huge computational requirements for the model and hence it is important to understand the hyper-parameters and their effect on the overall prediction.

### 1.3.2 Summary of models

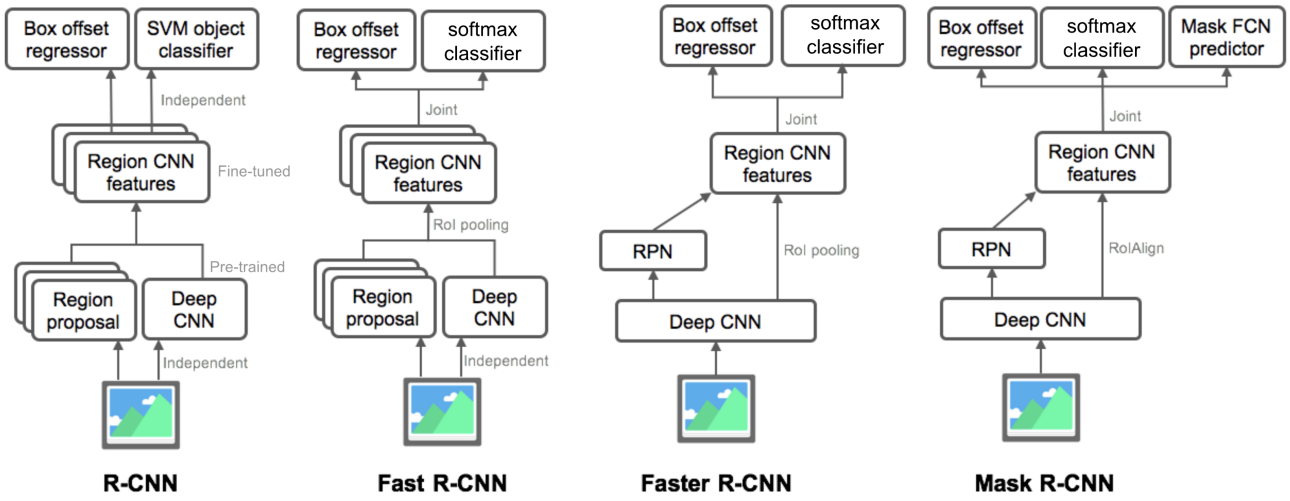



Figure 13: The different architectures of the Region Based Models

## 1.4 Cascade R-CNN

The two problems of recognition (distinguishing foreground objects from background and assigning proper class labels) and localization problem (assigning accurate bounding boxes) can be difficult to solve accurately, due to the fact that there are many "close" false positive, corresponding to "close but not correct" bounding boxes.

In object detection the difference between positives and negatives is not fine-grained. The boundary between them is defined by thresholding the IoU score between candidate and GT bboxes. Threshold is typically set to 0.5, but this is a very loose requirement for positives. The resulting detectors frequently produce noisy bounding boxes.

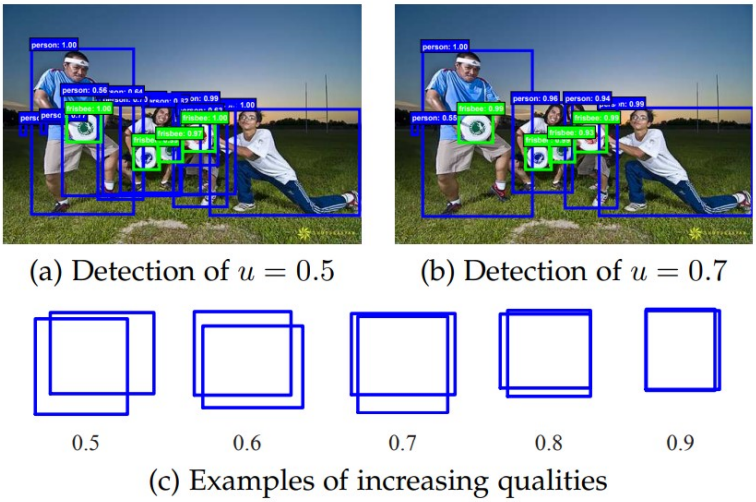



Figure 14: Example of detection with different thresholds

Prior deep learning object detectors' performance tends to degrade with increasing the IoU (Interaction over Union) thresholds [8] [12]. The detector trained with threshold score of 0.5 outperforms the detector trained with score 0.6 for low IoUs, underperforms it at higher IoUs. In general, a detector optimized for a single IoU value is not optimal for other values.

**Paradox of high quality detection**: the detector will only achieve high quality if presented with high quality proposals, but increasing the threshold score usually degrades the performance.

This problem has two causes:

- Overfitting during training, due to exponentially vanishing positive samples, i.e. lot of positive samples are gone when IoU threshold increases.
- Inference-time mismatch between the IoUs for which the detector is optimal and those of the input hypotheses. High quality detectors are only optimal for high quality hypotheses, detection performance can degrade substantially for hypotheses of lower quality. For example, training at higher(lower) IoU threshold but test at lower(higher) IoU threshold.

Cascade R-CNN is a multi-stage extension of Faster R-CNN proposed to solve the above problems, where detector stages deeper into the cascade are sequentially more selective against close false positives. The cascade stages are trained sequentially, using the output of one stage to train the next.

The output IoU of a Bbox Regressor is almost always better than its input IoU and the output of a detector trained with a certain IoU threshold is a good hypothesis distribution to train the next detector with higher IoU threshold.

Cascade R-CNN by adjusting bboxes at each stage aims to find a good set of false positive for training the next stage. With this resampling, the quality of detection hypothesis increases gradually and reduces the problems of overfitting and inference time mismatch.

### 1.4.1 Prior Architectures

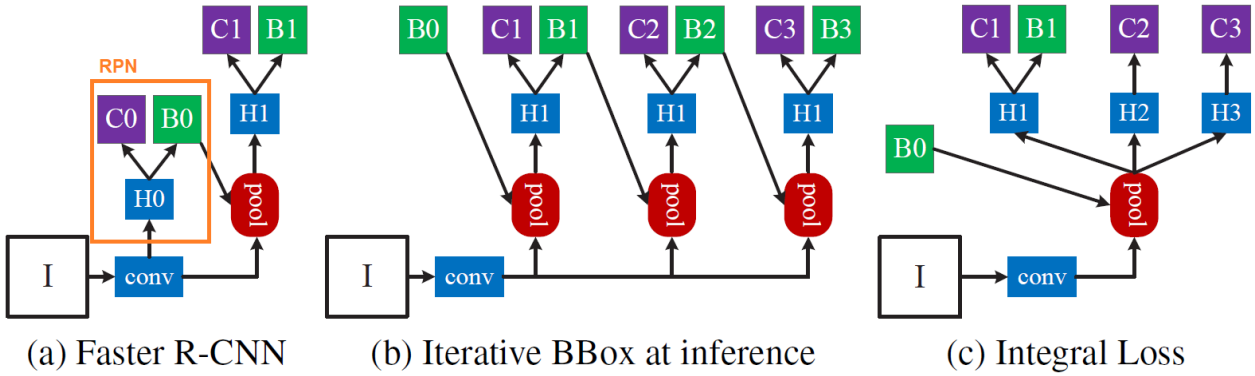



Figure 15: Prior Art Network Architectures

In Faster R-CNN the first stage is a proposal sub-network (H0), called Region Proposal Network, applied to the entire image to produce preliminary detection hypothesis, known as object proposals. In the second stage, these hypotheses are then processed by a region-of-interest detection sub-network (H1), denoted as detection head. A final classification score (C1) and a bounding box (B1) are assigned to each hypothesis.

The main problem is that B1 is not accurate enough.

Iterative Bbox Inference try to solve this problem by giving B1 in input to the same H1 multiple times, with this process it can regress the bounding box to obtain B2, and so on. This iterative approach attempts to gradually fine-tune the bounding box to obtain a more accurate one.

However, all heads (H1 in the figure) are the same, H1 is used again and again. The improved performance is limited. Usually no improvement beyond twice.

This is due to the fact that the distribution of bounding boxes changes significantly after each iteration. Therefore, it can be optimal for the initial distribution but sub-optimal after that.

Also, during training, there is only one H1, while during inference, there are multiple H1, which causes mismatch between training and testing.

For this reason, in Integral Loss different heads are used and the classifiers are ensembled during inference. However, high quality classifiers are prone to overfitting.

### 1.4.2 Cascade R-CNN Network

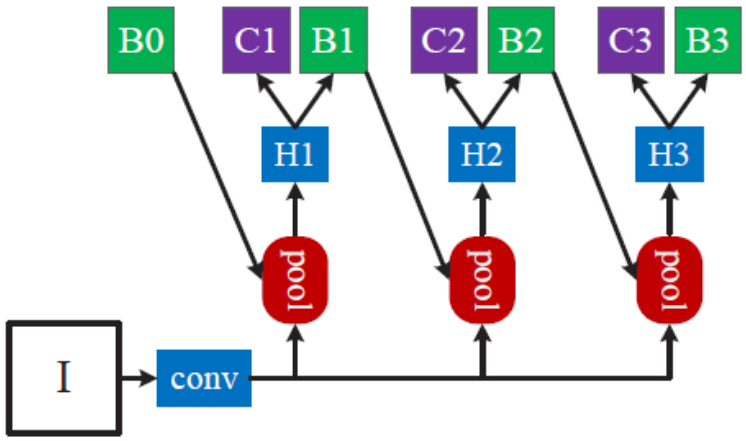



Figure 16: Cascade R-CNN Network Architecture

Different from iterative bbox that uses the same H1, different heads are used at different stages, i.e. H1, H2, H3 are used as shown in the figure above. Each of them is designed for one specific IoU threshold from small to large.

**Cascaded Bounding Box Regression**: regression task is decomposed in a sequence of simpler steps in which there is a cascade of specialized regressors optimized for the bbox distribution generated by the previous regressor. In this way the hypothesis are improved progressively.

**Cascaded Detection**: The initial hypotheses distribution produced by RPN is heavily tilted towards low quality. To solve the problem, cascade regression is used as a resampling mechanism that enables the sets of positive examples of successive stages to keep roughly constant size, even when the detector quality threshold is increased. The main advantage is that the potential for overfitting at large IoU thresholds is reduced, since positive examples become plentiful at all stages. At each stage the R-CNN head contains a classifier and a regressor optimized for the corresponding IoU threshold, this makes the detectors of deeper stages optimal for higher IoU thresholds, beating the paradox of high quality detection.

**Cascaded Segmentation**: in general, instance segmentation is implemented in addition to object detection, and a stronger object detector usually leads to improved instance segmentation. In the Mask R-CNN, the segmentation branch is inserted in parallel to the detection branch. However, the Cascade R-CNN has multiple detection branches. This raises the questions of 1) where to add the segmentation branch and 2) how many segmentation branches to add.

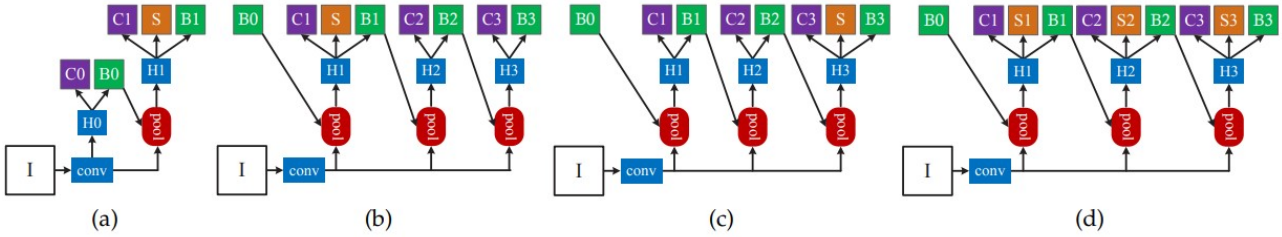



Fig. 6: Architectures of the Mask R-CNN (a) and three Cascade Mask R-CNN strategies for instance segmentation (b)-(d). Beyond the definitions of Fig. 3, "S" denotes a segmentation branch. Note that segmentations branches do not necessarily share heads with the detection branch.

Figure 17: Cascade Mask R-CNN architectures

An option could be adding a single mask prediction head at either the first or last stage of the Cascade R-CNN. The third strategy addresses the second question, adding a segmentation branch to each cascade stage.

All three strategies improve on baseline performance, although with smaller gains than object detection.

Comparing strategies, (c) outperforms (b). This is because (b) trains the mask head in the first stage but tests after the last stage, leading to a mask prediction mismatch. This mismatch is reduced by (c). The addition of a mask branch to each stage by strategy (d) does not have noticeable benefits over (c), but requires much more computation and memory. Strategy (b) has the best trade-off between cost and AP performance.

## 1.5 Hybrid Task Cascade

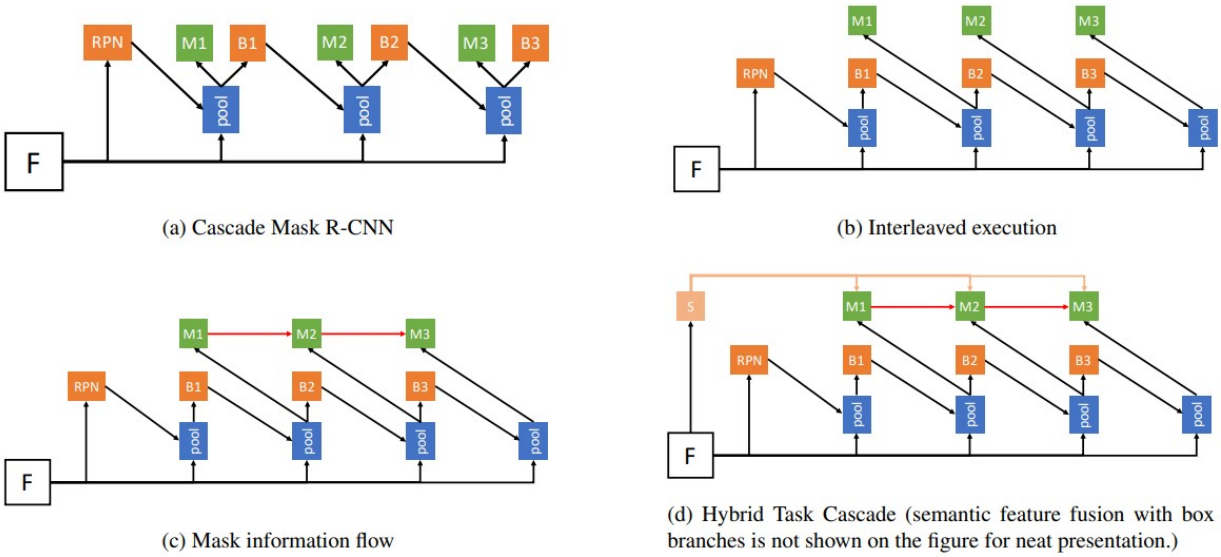
A combination of Cascade R-CNN and Mask R-CNN only brings limited gains in terms of mask AP compared to bbox AP.

Hybrid Task Cascade is another cascade architecture for instance segmentation used to bridge this gap. It had the best performance on the benchmark MS-COCO dataset as of 2019 [13].

Two important aspects of HTC are:

1. instead of performing cascaded refinement on detection and segmentation tasks separately, it interweaves them for a joint multi-stage processing;

2. it adopts a fully convolutional branch to provide spatial context, which can help distinguishing hard foreground from cluttered background;




(a) Cascade Mask R-CNN

(b) Interleaved execution

(c) Mask information flow

(d) Hybrid Task Cascade (semantic feature fusion with box branches is not shown on the figure for neat presentation.)

Figure 18: The architecture evolution from Cascade Mask R-CNN to Hybrid Task Cascade

One draw back of Cascade Mask R-CNN is that the two branches (bounding box regression and mask prediction) are executed in parallel during training, both taking the bounding box predictions from the preceding stage as input. Consequently, the two branches are not directly interacting within a stage.

The three key points of hybrid task cascade are:

- Interleaved bbox regression and mask prediction
- Information flow between mask branches
- Semantic segmentation branch to explore more contextual information

To improve the performance, the bbox and mask branches are **interleaved**, in this way the mask branch can take advantage of the updated bounding box predictions.

With only the interleaved execution, the mask prediction at each stage is based purely on the RoI features and bbox of the previous stage. There is no direct information flow between mask branches at different stages, which prevents improvements on mask prediction accuracy.

For this reason a direct path to reinforce the **information flow** between mask branches is added, by feeding the mask features of the preceding stage to the current stage. This information flow makes it possible for progressive refinement of masks, instead of predicting masks on progressively refined bounding boxes.

Finally to further help distinguishing the foreground from the cluttered background, the **spatial context** has been used. An additional branch to predict per-pixel semantic segmentation for the hole images has been added (S in figure 18). Semantic segmentation feature is a strong complement for the existing bbox and mask features, thus them are combined for obtain better performance. As a result, the bbox and mask heads of each stage take not only the RoI features extracted from the backbone (or neck) as input, but also exploit semantic features, which can be more discriminative on cluttered background.

**Loss Function**

The overall loss function takes the form of a multi-task learning:

$$L = \sum_{t=1}^{T} \alpha_t (L^t_{bbox} + L^t_{mask} + \beta L_{seg})$$

where:

- $L^t_{bbox}(c_i, r_t, \hat{c}_t, \hat{r}_t) = L_{cls}(c_t, \hat{c}_t) + L_{reg}(r_t, \hat{r}_t)$.

- $L^t_{mask}$ is the loss of mask prediction at stage $t$, which adopts the binary cross entropy as in Mask-RCNN.
- $L_{seg} = CE(s, \hat{s})$ is the semantic segmentation loss in the form of cross entropy.
- $\alpha_t$ and $\beta$ are used to balance the contribution of different stages and tasks.

### 1.6 Architecture

Although model architectures of different detectors are different, they have common components, which can be roughly summarized into the following classes [15]:

- Backbone: is the part that transforms an image to feature maps, such as a ResNet-50, without the last fully connected layer.
- Neck: is the part that connects the backbone and the heads. It performs some refinements or reconfigurations on the raw feature maps produced by the backbone. An example is Feature Pyramid Network.
- DenseHead: is the part that operates on dense locations of feature maps, including AnchorHead and AnchorFreeHead, e.g. RPNHead, RetinaHead, FCOSHead.
- RoIExtractor: is the part that extracts RoI-wise features from a single or multiple feature maps with RoIPooling-like operators. An example that extracts RoI features from the corresponding level of feature pyramids is SingleRoIExtractor.
- RoIHead: is the part that takes RoI features as input and make RoI-wise task-specific predictions, such as bounding box classification/regression, mask prediction.

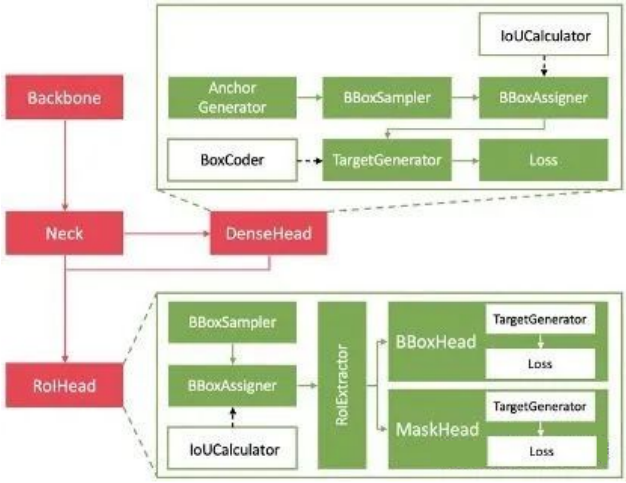



Figure 19: Model architecture example

#### 1.6.1 Backbone: ResNet

According to *Universal approximation theorem*, we know that, given enough capacity a feed-forward neural network with a single layer is sufficient to represent any function. However, the layer might be massive and the network is prone to overfitting the data. Therefore, there is a common trend in the research community that our network architecture needs to go deeper. For example, the VGG network and GoogleNet had 19 and 22 layers respectively.

However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient infinitely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

The core idea of ResNet is introducing a so-called **identity shortcut connection** that skips one or more layers. With these connections the gradients can flow directly through the skip connections backwards from later layers to initial filters [11].

The identity mapping does not have any parameters and is just used to add the output from the previous layer to the layer ahead.

Instead of learning a direct mapping $x \to y$ with a function $H(x)$, we can define a residual function $F(x) = H(x) - x$, which can be reframed into $H(x) = F(x) + x$, where $F(x)$ is the residual function and x represents the identity (input = output).

The results shows that is easy to optimize the residual function $F(x)$ than to optimize the original, unreferenced mapping $H(x)$.

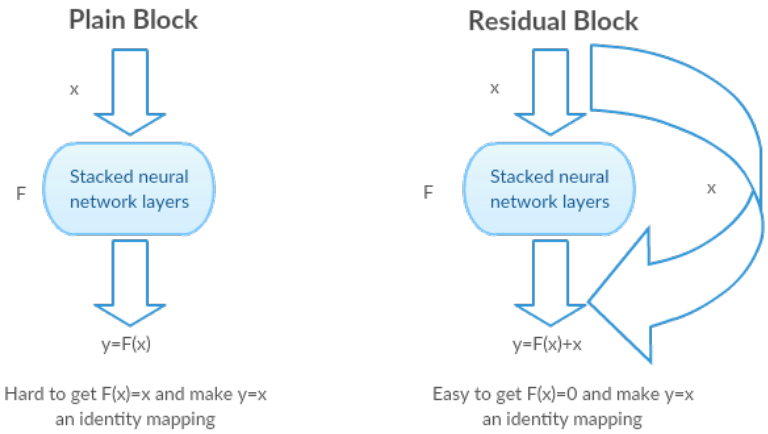



Figure 20: Identity mapping in residual blocks

Each residual unit is composed of two convolutional layers, with Batch Normalization (BN) and ReLU activation function, using $3 \times 3$ kernels and preserving spatial dimensions.

### 1.6.2 Backbone: ResNeXt

ResNeXt provides an interpretation of treating ResNet as an ensemble of many smaller networks.

The operation of a simple neuron in a network can be recast as a combination of splitting, transforming, and aggregating:

- split: $x \to x_i$
- transform: $w_i x_i$
- aggregate: $\sum_{i=1}^{n} w_i x_i$

The ResNeXt architecture follow the same paradigm, but the elementary transformation ($w_i x_i$) is replaced with a more generic function, which itself can also be a network.

Therefore, in contrast to previous "Network-in-Network" paradigm, that turns out to increase the dimension of depth, this new "Network-in-Neuron" paradigm expands along a new dimension. This means that instead of linear function in a simple neuron that do $w_i x_i$ in each path, a non-linear function $T_i(x)$ is performed for each path:

$$F(x) = \sum_{i=1}^{C} T_i(x)$$

A new dimension C is introduced, called *Cardinality*. The dimension of cardinality controls the number of independent paths, i.e. the number of more complex transformations.

Therefore, in this architecture the same transformations are applied 32 times, and the results are aggregated at the end, as shown in figure 21.

It is shown by experiments that cardinality is an essential dimension and can be more effective than the dimensions of width and depth, indeed, the authors note that ResNeXt architectures can achieve better performance with the same complexity as ResNet.

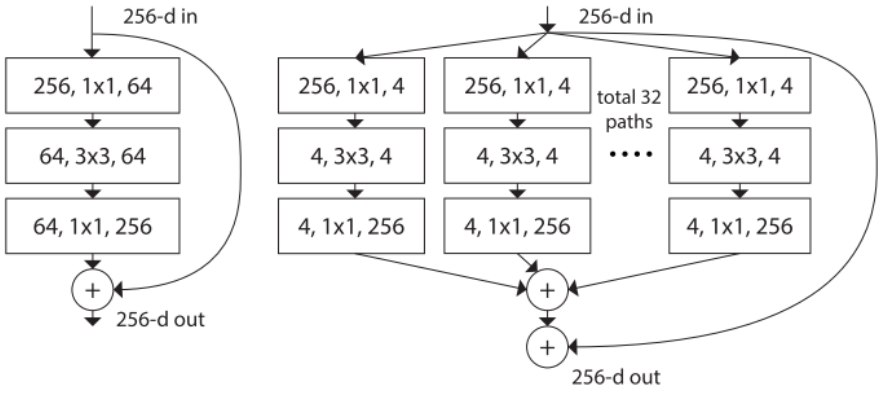



Figure 21: ResNeXt building blocks (the numbers in the blocks represents [# input channels, filter size, # output channels])

As we can see the right block is the block used in ResNeXt and instead of bottling the incoming 256 dimension vector to 64 channels and then bumping it back up, it drastically reduces the channel size to 4 and bumps back up to 256 but does this operation multiple times in parallel.

### 1.6.3 Neck: Feature Pyramid Network

Detecting objects in different scales is challenging in particular for small objects. We can use a pyramid of the same image at different scale to detect objects.

However, processing multiple scale images is time consuming and the memory demand is too high to be trained end-to-end simultaneously.

Hence, we can create a pyramid of feature and use them for object detection. However, feature maps closer to the image layer composed of low-level structures that are not effective for accurate object detection [9].

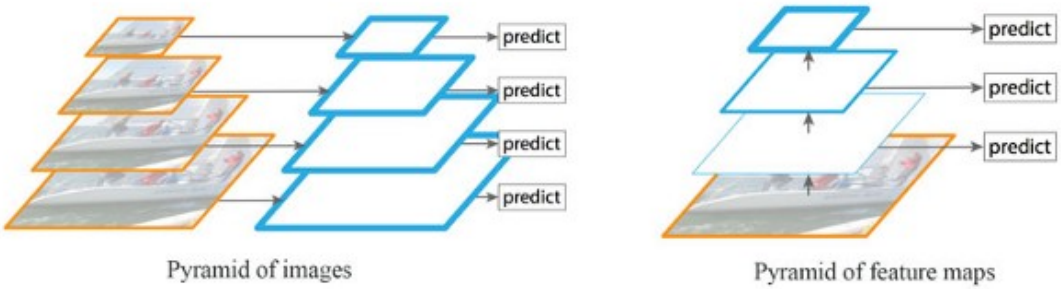



Figure 22: FPN

Feature Pyramid Network (FPN) is a feature extractor designed for such pyramid concept with accuracy and speed in mind. It replaces the feature extractor of detectors like Faster R-CNN and generates multiple feature map layers (multi-scale feature maps) with better quality information than the regular feature pyramid for object detection.

FPN composes of a bottom-up and a top-down pathway. The bottom-up pathway is the usual convolutional network for feature extraction. As we go up, the spatial resolution decreases. With more high-level structures detected, the *semantic value* for each layer increases.

FPN provides a top-down pathway to construct higher resolution layers from a semantic rich layer.

While the reconstructed layers are semantic strong but the locations of objects are not precise after all the downsampling and upsampling. We add lateral connections between reconstructed layers and the corresponding feature maps to help the detector to predict the location betters. It also acts as skip connections to make training easier (similar to what ResNet does).

*FPN is not an object detector by itself, it is a feature extractor that works with object detectors.*

FPN extracts feature maps and later feeds into a detector, says RPN, for object detection.

In the FPN framework, for each scale level, a $3 \times 3$ convolution filter is applied over the feature maps followed by separate $1 \times 1$ convolution for objectness predictions and boundary box regression. These $3 \times 3$ and $1 \times 1$ convolutional layers are called the RPN head. The same head is applied to all different scale levels of feature maps.

FPN is also good at extracting masks for image segmentation.

### 1.6.4 Example: FPN with Faster R-CNN

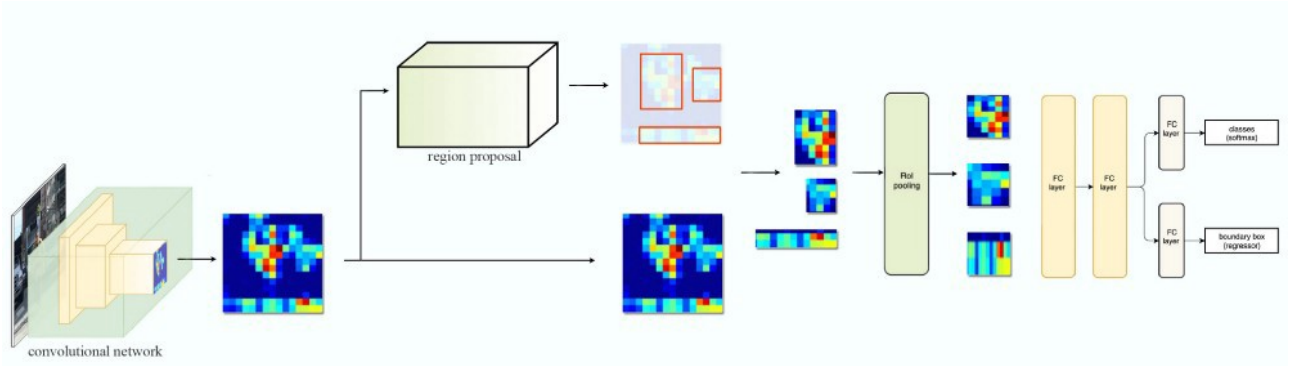



Figure 23: Faster RCNN architecture

In FPN, we generate a pyramid of feature maps. We apply the RPN (described in the previous section) to generate RoIs. Based on the size of the RoI, we select the feature map layer in the most proper scale to extract the feature patches.

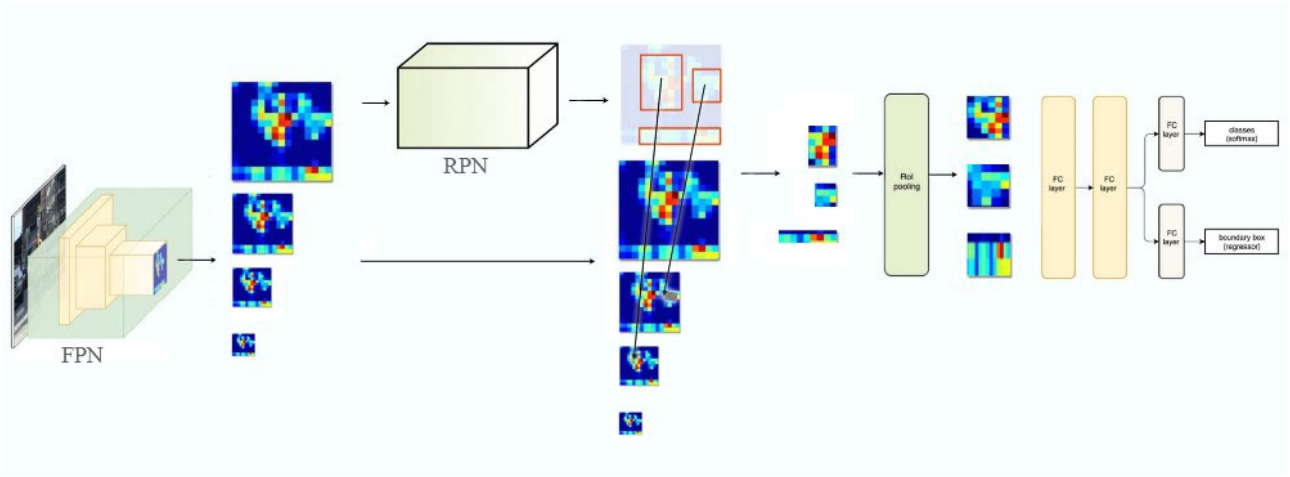



Figure 24: Faster RCNN architecture with FPN

## 1.7 Data Augmentation & Test Time Augmentation

There exists a lot of ways to improve the results of a neural network by changing the way we train it. In particular, Data Augmentation is a common practice to virtually increase the size of training dataset, and is also used as a regularization technique, making the model more robust to slight changes in the input data.

Data Augmentation is the process of randomly applying some operations (rotation, zoom, shift, flips,...) to the input data. By this mean, the model is never shown twice the exact same example and has to learn more general features about the classes he has to recognize.

This means, variations of the training set images that are likely to be seen by the model. For example, a horizontal flip of a picture of a cat may make sense, because the photo could have been taken from the left or right. A vertical flip of the photo of a cat does not make sense and would probably not be appropriate given that the model is very unlikely to see a photo of an upside down cat.

Image data augmentation is typically only applied to the training dataset, and not to the validation or test dataset.

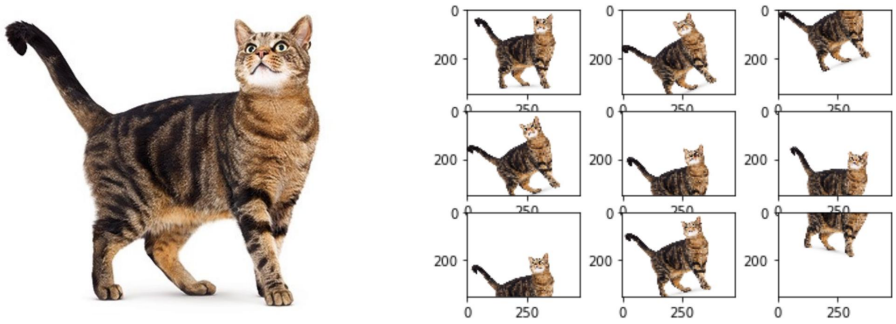


Figure 25: An example of image data augmentation

However, there also exists some ways to improve the results of the model by changing the way we test it, and this is where Test Time Augmentation (TTA) comes into play.

Similar to what Data Augmentation is doing to the training set, the purpose of Test Time Augmentation is to perform random modifications to the test images. Thus, instead of showing the regular, "clean" images, only once to the trained model, we will show it the augmented images several times. We will then average the predictions of each corresponding image and take that as our final guess.

For example, we can show to the model 5 different modifications of the same image (by flipping, rotate, etc.) and then the model makes a prediction for each image. It can happen that only a subset of predictions are correct, but the mean of them gives the correct answer.

By averaging our predictions, on randomly modified images, we are also averaging the errors. The error can be big in a single vector, leading to a wrong answer, but when averaged, only the correct answer stand out. This is particularly useful for test images that the model is pretty unsure

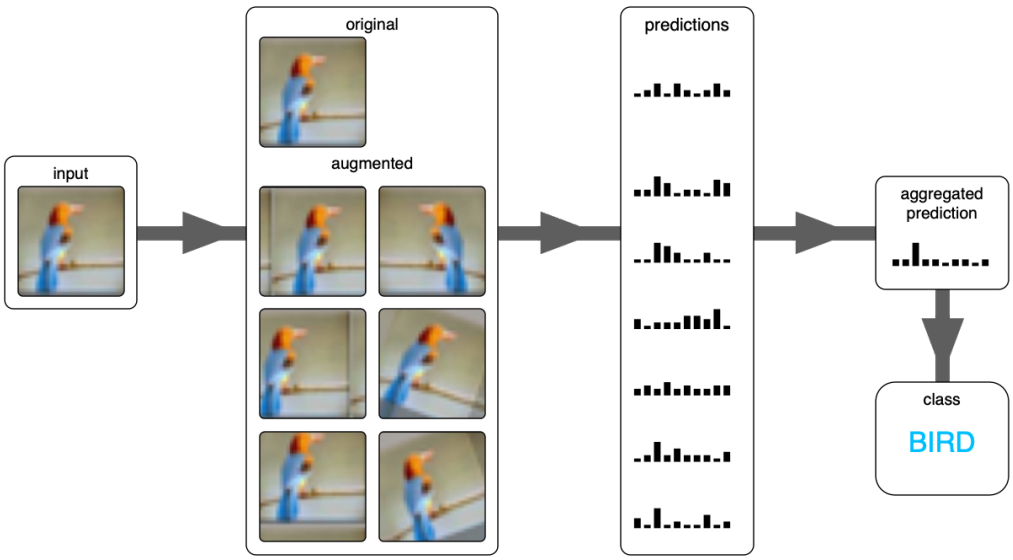



Figure 26: An example of test time augmentation

# 2 Methodology

## 2.1 Experimental settings and libraries

For this work Python was used with the MMDetection library which provides a simple way to create a variety of detection deep learning networks in an easy way.

MMDetection is an open source object detection toolbox based on PyTorch. It is a part of the OpenMMLab project. The toolbox directly supports popular and contemporary detection frameworks, e.g. Faster RCNN, Mask RCNN, RetinaNet, etc.

All basic bbox and mask operations run on GPUs. The training speed is faster than or comparable to other codebases, including Detectron2, maskrcnn-benchmark and SimpleDet.

The models implemented in this project were processed using the Google Colaboratory tool (Google Colab), which allows the use of the Jupyter Notebook service hosted on Google servers, with free access to computational resources, including Nvidia K80, T4, P4 and P100 GPUs and 12GB of RAM.

To evaluate the results of the experiments we will use the official cocoapi python library that assists in loading, parsing, visualizing and evaluating the annotations in COCO format. In the section 2.6 we will describe all the 12 common metrics used to evaluate coco like dataset [17].

In addition TIDE toolbox has been used. This is a general toolbox for identifying object detection and instance segmentation errors [18]. With this tool we are able to decompose the error in 6 categories:

- Classification Error: $IoU_{max} \geq t_f$ for GT of the incorrect class, i.e. localized correctly and classified incorrectly;
- Localization Error: $t_b \leq IoU_{max} \leq t_f$ for GT of the correct class, i.e. classified correctly but localized incorrectly;
- Both Cls and Loc Error: $t_b \leq IoU_{max} \leq t_f$ for GT of incorrect class, i.e. classified incorrectly and localized incorrectly;
- Duplicate Detection Error: $IoU_{max} \geq t_f$ for GT of the correct class, but another higher-scoring detection already matched that GT, i.e. would be correct if not for a higher scoring detection;
- Background Error: $IoU_{max} \leq t_b$ for all GT, i.e. detected background as foreground;
- Missed GT Error: All undetected ground truth (false negative) not already covered by classification or localization error;

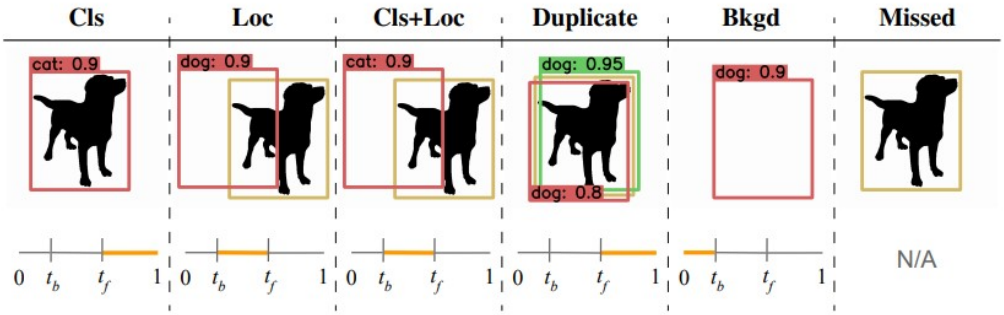



Fig. 1: **Error Type Definitions.** We define 6 error types, illustrated in the *top row*, where box colors are defined as: ■ = false positive detection; ■ = ground truth; ■ = true positive detection. The IoU with ground truth for each error type is indicated by an orange highlight and shown in the *bottom row*.

Figure 27: Tide types of errors

## 2.2 Dataset

As described in the introduction section the dataset used is the AICrowd Food Recognition Challenge Dataset composed of:

- Training set: with 24.120 food images, with their corresponding 39.328 annotations in MS-COCO format
- Validation set: with 1.269 food images, with their corresponding 2.053 annotations in MS-COCO format
- Test set: contains the same images as the validation set

The MS-COCO json format is specific for object detection and instance segmentation; COCO annotations are inspired by the Common Objects in Context (COCO) dataset, that is a large scale object detection dataset with million of object instances and 80 object category.

```
{
    "info": {...},
    "categories": [
        {
            'id': 2578,
            'name': 'water',
            'name_readable': 'Water',
            'supercategory': 'food'
        }, ..., {...}
    ],
    "images": [
        {
            'file_name': '065537.jpg',
            'height': 464,
            'id': 65537,
            'width': 464
        }, ..., {...}
    ],
    "annotations": [
        {
            'area': 44320.0,
            'bbox': [86.5, 127.49999999999999, 286.0, 170.0],
            'category_id': 2578,
            'id': 102434,
            'image_id': 65537,
            'iscrowd': 0,
            'segmentation': [[235.99999999999997,
                              372.5,
                              ...
                              264.0,
                              371.5]]
        }, ..., {...}
    ],
}
```

Listing 1: COCO annotations format

The main components of COCO annotations format are: *info* - contains high level information about the dataset, *categories* - contains a list of categories, *images* - contains the complete list of images in the dataset and *annotations* - contains a list of every individual object annotation from every image in the dataset.

## 2.3 Data Exploration

The dataset contains **273** classes, but the number of images is not balanced between them. Both the train and validation sets are imbalanced.

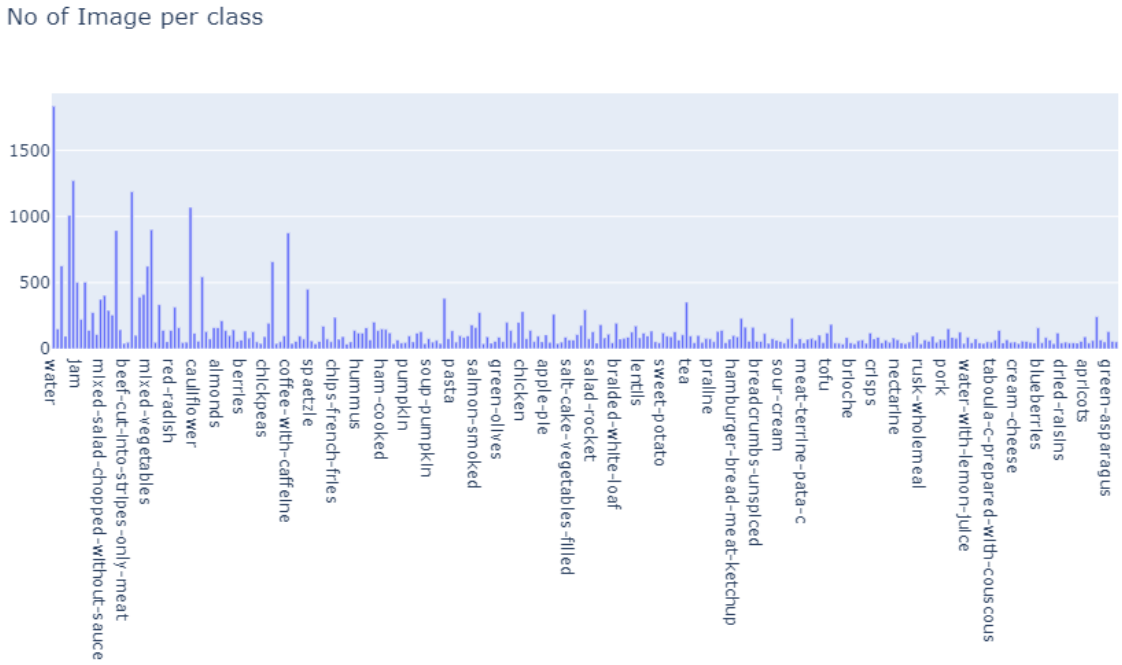



Figure 28: Number of images per class in the training set

The average number of images per class is 144, however there are classes with a higher number, such as "Water", that is the class with the greater number of images (1853) and classes with a smaller number, "Veggie Burger" is the class with less number of images, only 35.

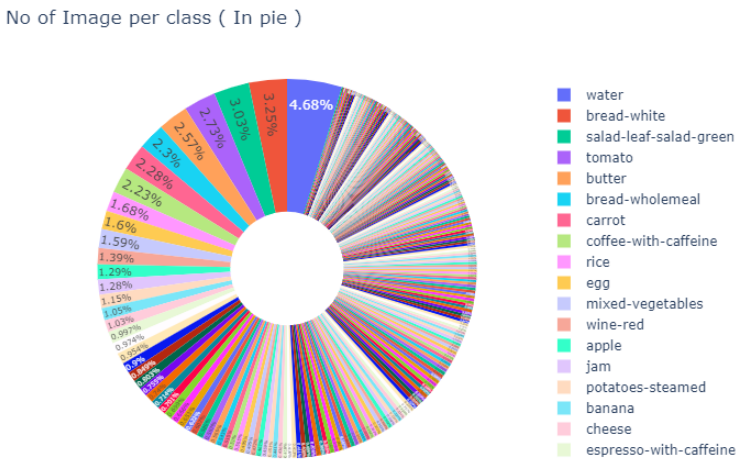



Figure 29: Number of images per class in the training set

Also the dimension of the images varies a lot, the average dimension is $645 \times 645$, but the distribution is skewed, so the median is $480 \times 480$.

The minimum width-height is $182 - 183$ and the maximum $4608 - 4096$.

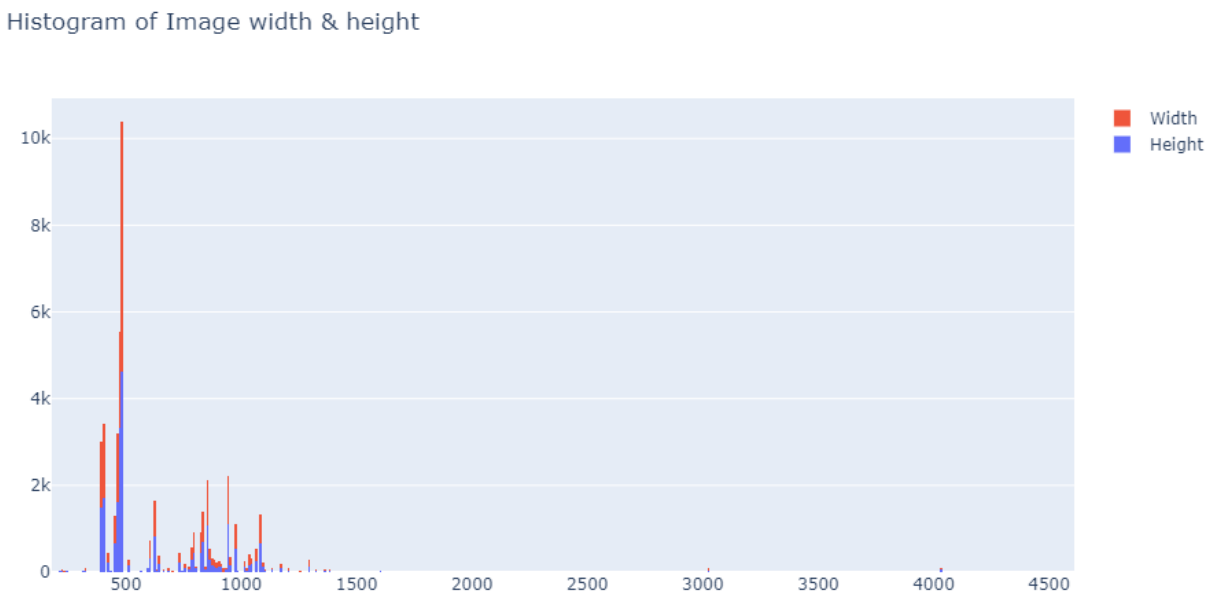



Figure 30: Distribution of with and height for training set images

Many of the images are square, the aspect ratio ($height \setminus width$) is equal to 1, some are horizontal rectangles, the minimum aspect ratio is 0.41 and some are vertical rectangles, 2.15 is the maximum aspect ratio.

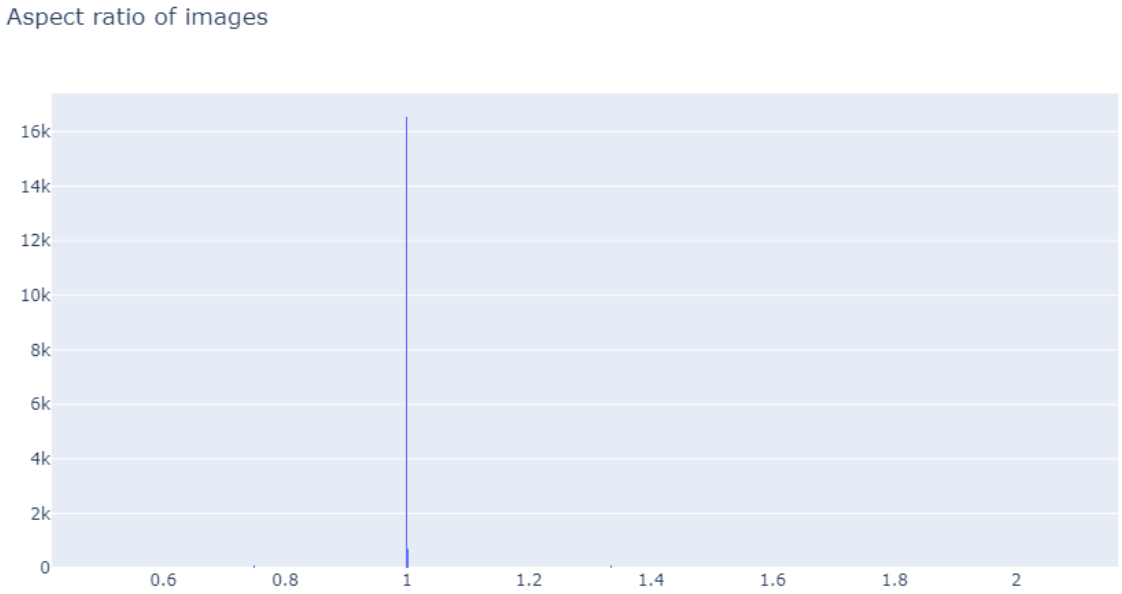



Figure 31: Aspect ratio distribution for training set images

The total number of annotations is 39.328, every one of them represents a single object. Therefore we have a mean of 1.65 instances per image. More than 15.000 images contains only one instance, only a few hundred contains more than 6 elements, the maximum is 12 in a single image.

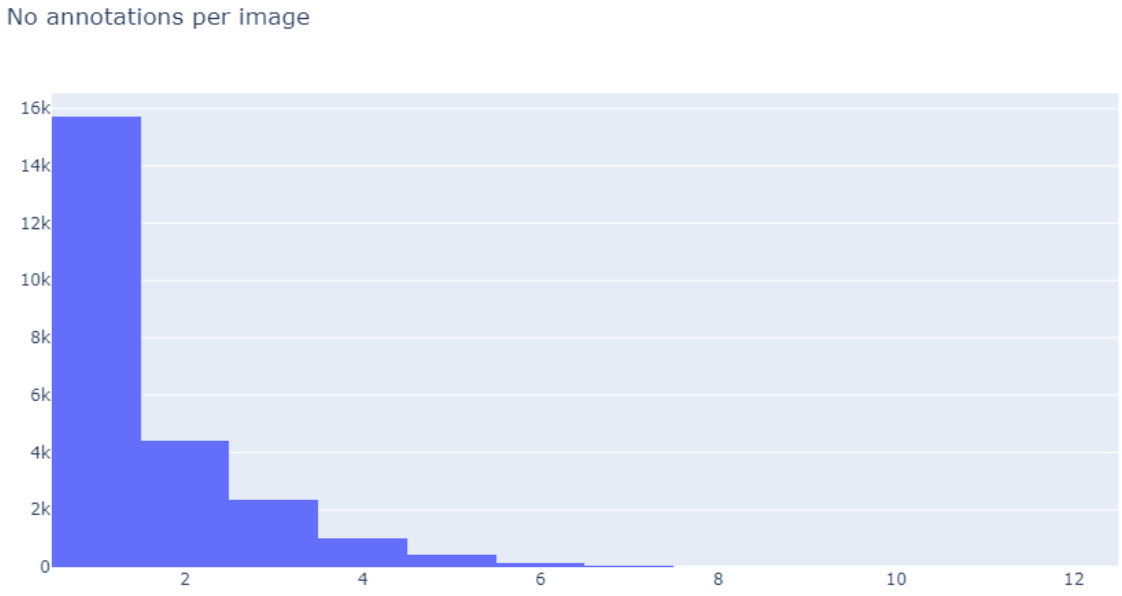
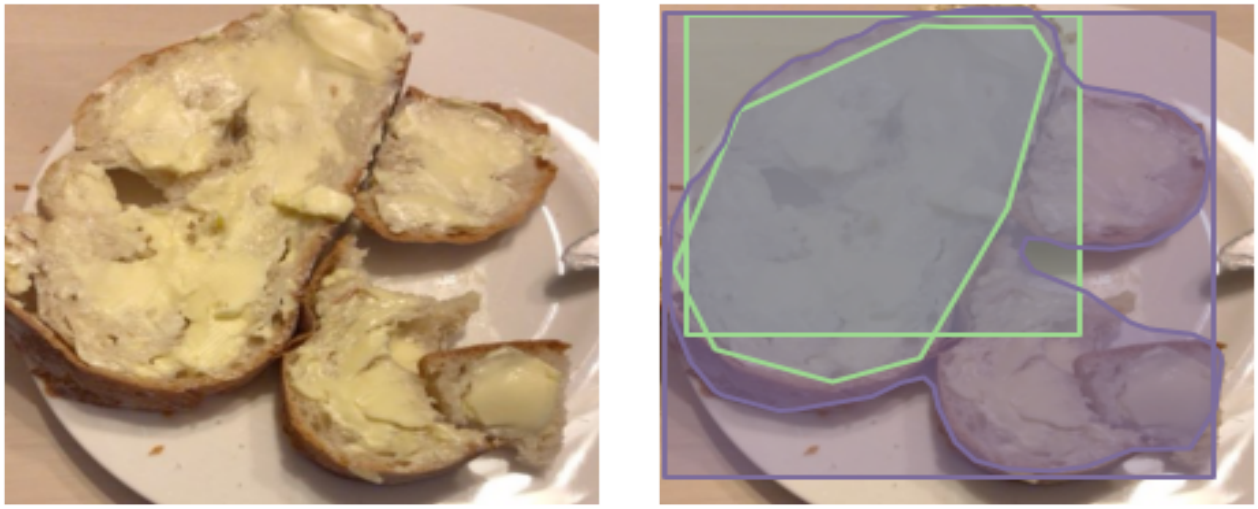



Figure 32: Number of instances per image (training set)



(a) Original image

(b) Image with bbox and mask

Figure 33: An example of an image with two annotations

By looking to various examples of images in the dataset we have noticed that the mask annotations have a very good quality, but the bboxes are not so precise, some of them are rotated or completely out of position with respect to the mask itself, see figure 34.


Figure 34: Some examples of images with annotations from the training set

By exploring the dataset and by looking to many images we can say that it contains high quality annotations, but there are some difficulties that can affect the learning process:

- Drinks are annotated in many different ways (figure 36)
- Images not completely annotated (figure 37)
- Some categories are difficult to recognize also for humans (figure 38), for example there are:
  - 20 types of bread (Bread, Bread-Black, Gluten Free Bread, 5 Grain Bread, ...);
  - 3 types of water (Water, Water with lemon juice, Water mineral);
  - Many different types of tea, as shown in figure 38;
  - And so on;

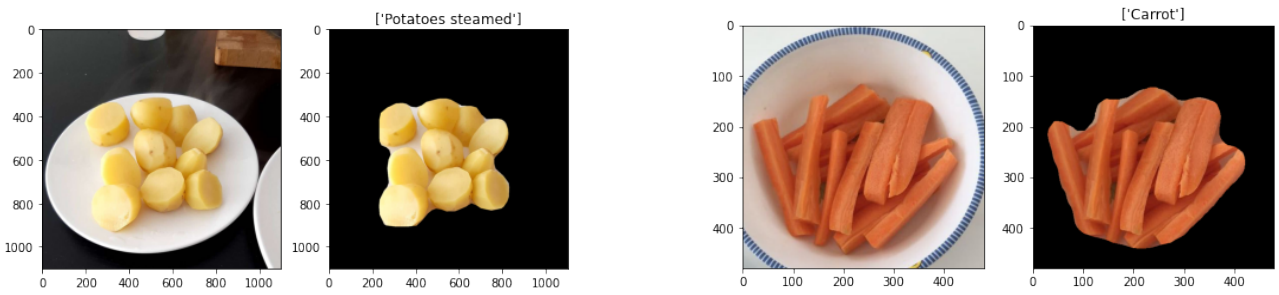



Figure 35: An example of two images with correct annotations

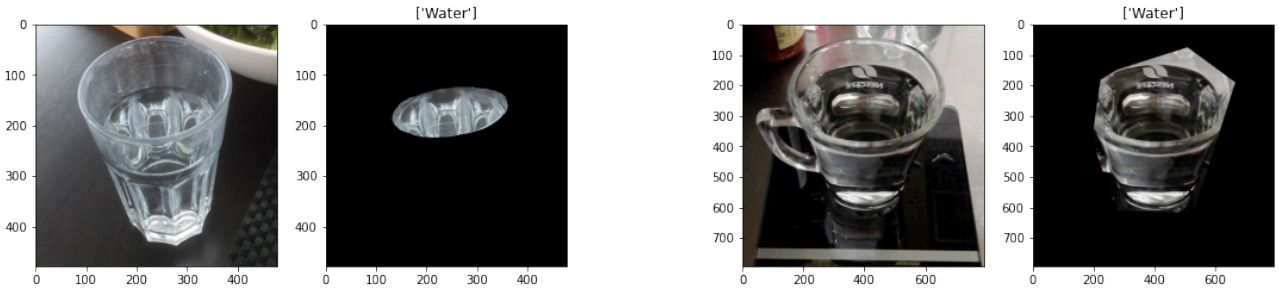



Figure 36: An example of different annotations style for drinks

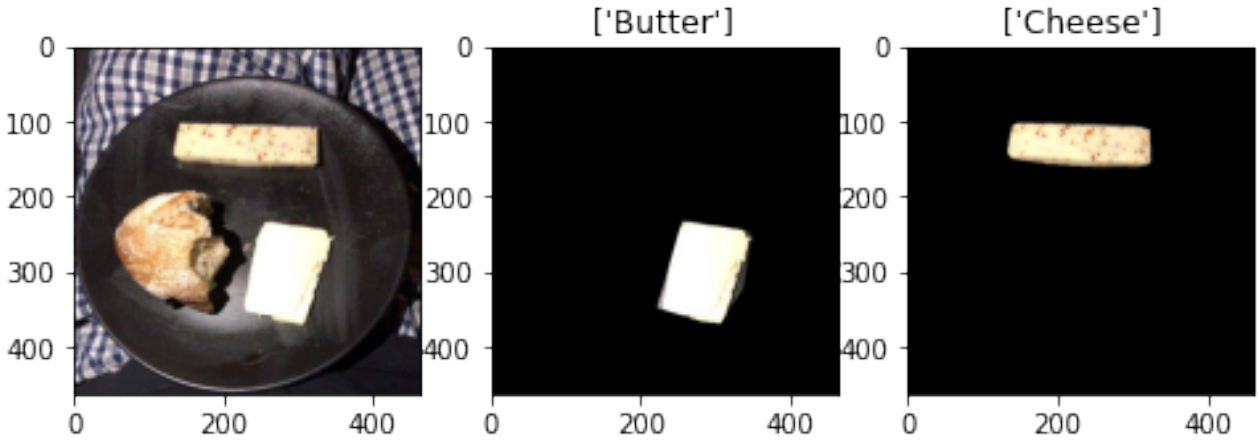



Figure 37: An example of image with incomplete annotation

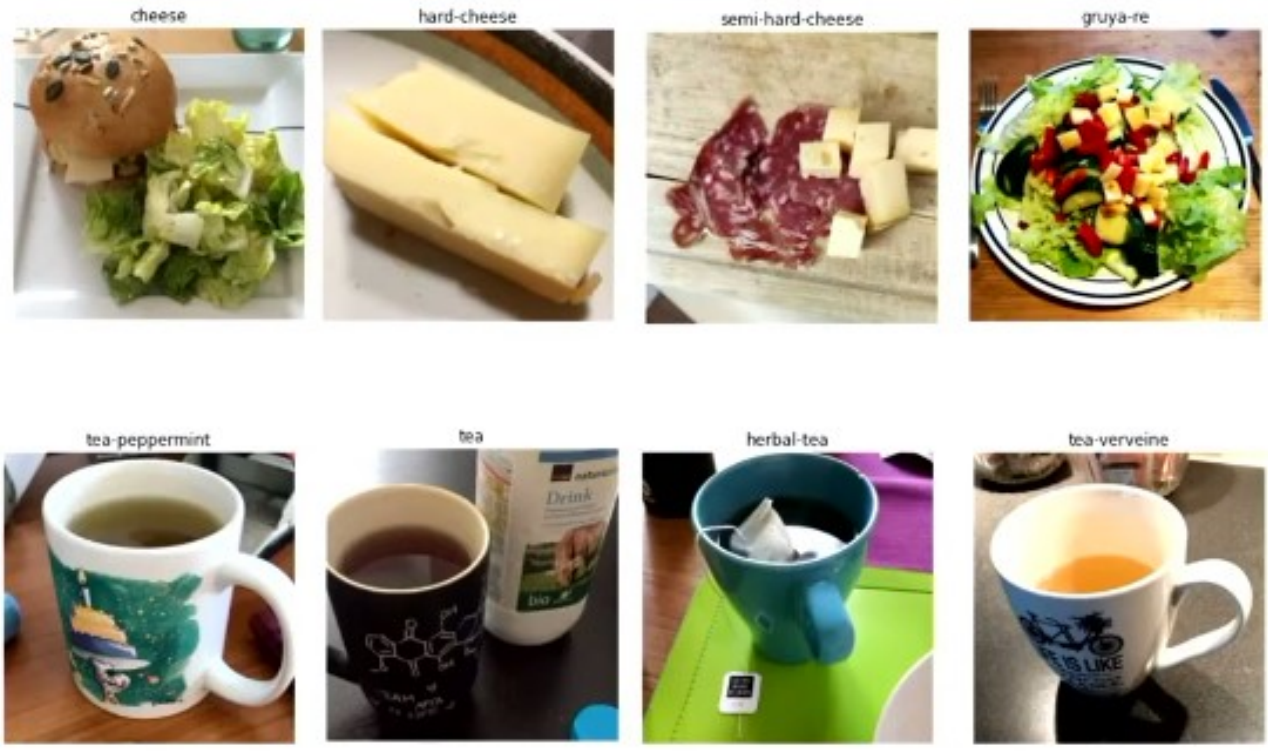



Figure 38: Some examples of classes that are difficult to distinguish

## 2.4 Data Preprocessing

### 2.4.1 Incorrect Annotations

Some of the images have annotations which are rotated with respect to image with and height. In other words, there are images that have a mismatch between image size and the sizes (for masks and bboxes) recorded in the COCO annotation files.

This problem derive from the fact that some images were taken with a cellphone and the orientation was not taken into account, meaning the with and height are swapped. For example, lets say an image has a height of 400 and width of 200, but it is recorded as height = 200 and width = 400.

These incorrect images make impossible to train the networks because they cause errors and raise exception during training, therefore we had to decide how to prevent the problem. Two solutions were possible:

- Remove the references to that images from annotation files, therefore ignore the images
- Modify the annotations by swapping image with and height in annotations data

Both of the modes where tested, the first was chosen because it preserves all the images in the dataset.

The number of incorrect images found is 28, of which 27 in the training set and 1 in the validation set.

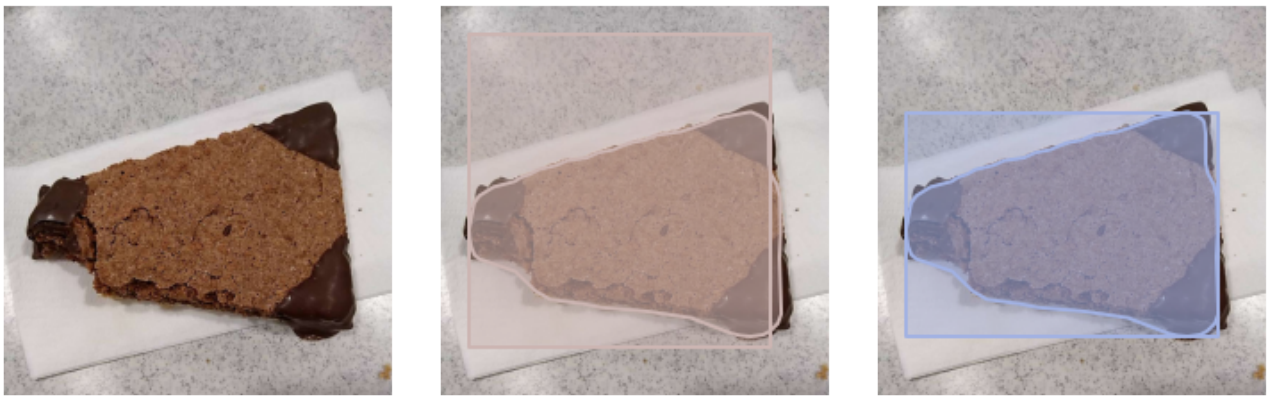
Figure 39: Some examples of problematic images

### 2.4.2 Fix Bboxes

While the masks seems to be correct, many of the images in the training set have bboxes that do not perfectly match the masks. Some of the bboxes are merely slightly off, but many are drastically off, as we can see in the examples in figure 34.

If these bboxes were used for training they could cause problems as the models will be attempting to learn using incorrect bboxes.

The idea to fix this problem was to recreate the bboxes coordinates based on the masks and modify the annotation files.



(a) Original image (b) Original bbox (c) Bbox calculated from mask

Figure 40: Bbox based on mask example

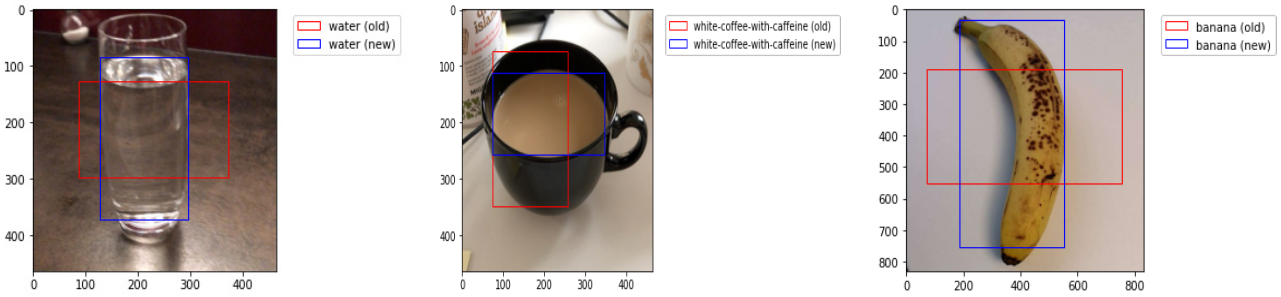



Figure 41: Examples of bboxes before and after update

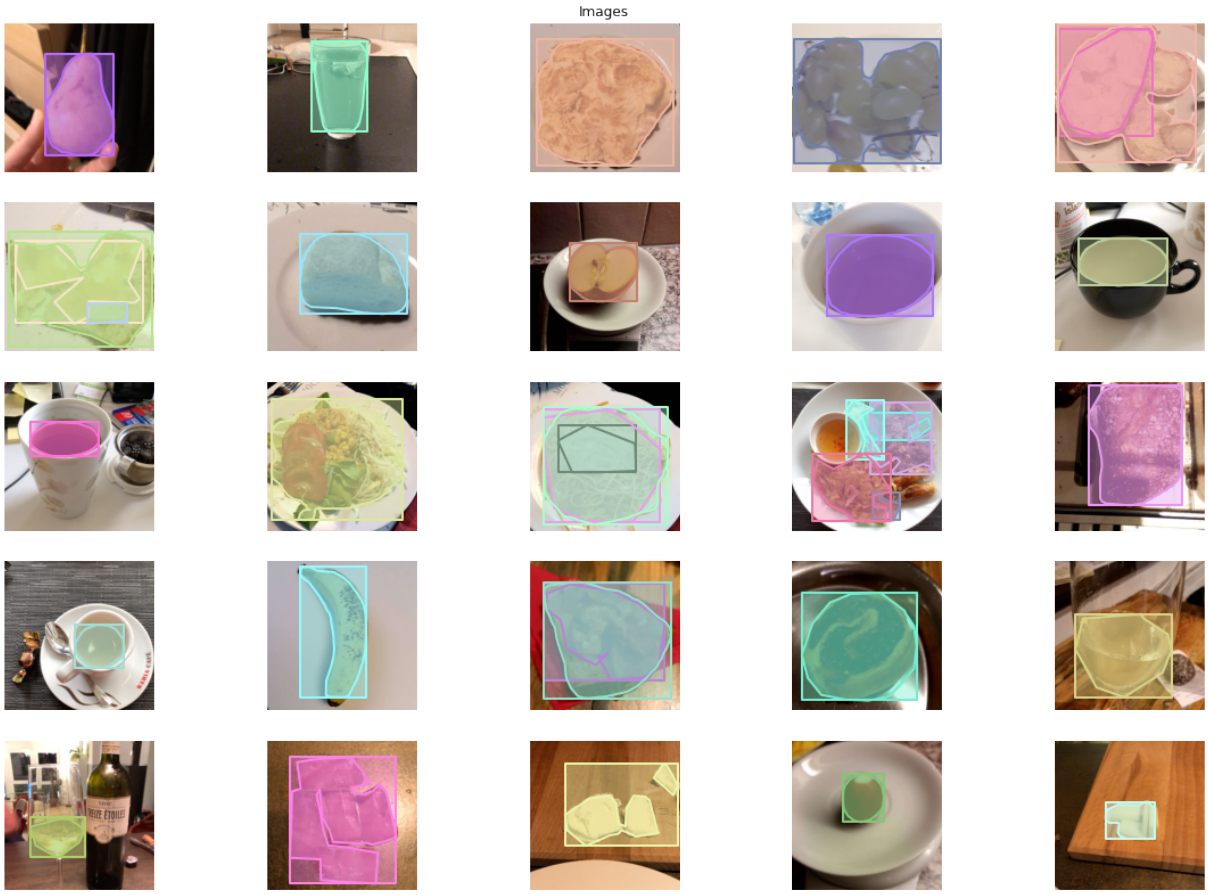



Figure 42: Some examples of images from the training set with fixed annotations

## 2.5 Data Augmentation

Different augmentation techniques have been used during training, these are shown in table 1. The probabilities proposed in the subsequent table are not precisely the same used in the experimental study. Indeed, the probabilities have been adjusted dynamically during the training phase looking to the progress of the loss function. When the loss start decreasing more slowly we incremented the probabilities of the different data augmentation techniques to try to speed up the process.

The figure 43 shows examples of the application of different augmentation techniques on an image of the dataset. We decided to not use channel shuffling because it seems to be difficult that something similar to the augmented image appear in the real world and also, in our dataset, colour seems to be an important information to recognize similar categories. For these reasons we removed channel shuffling and set a very small probability to multiplicative noise augmentation.

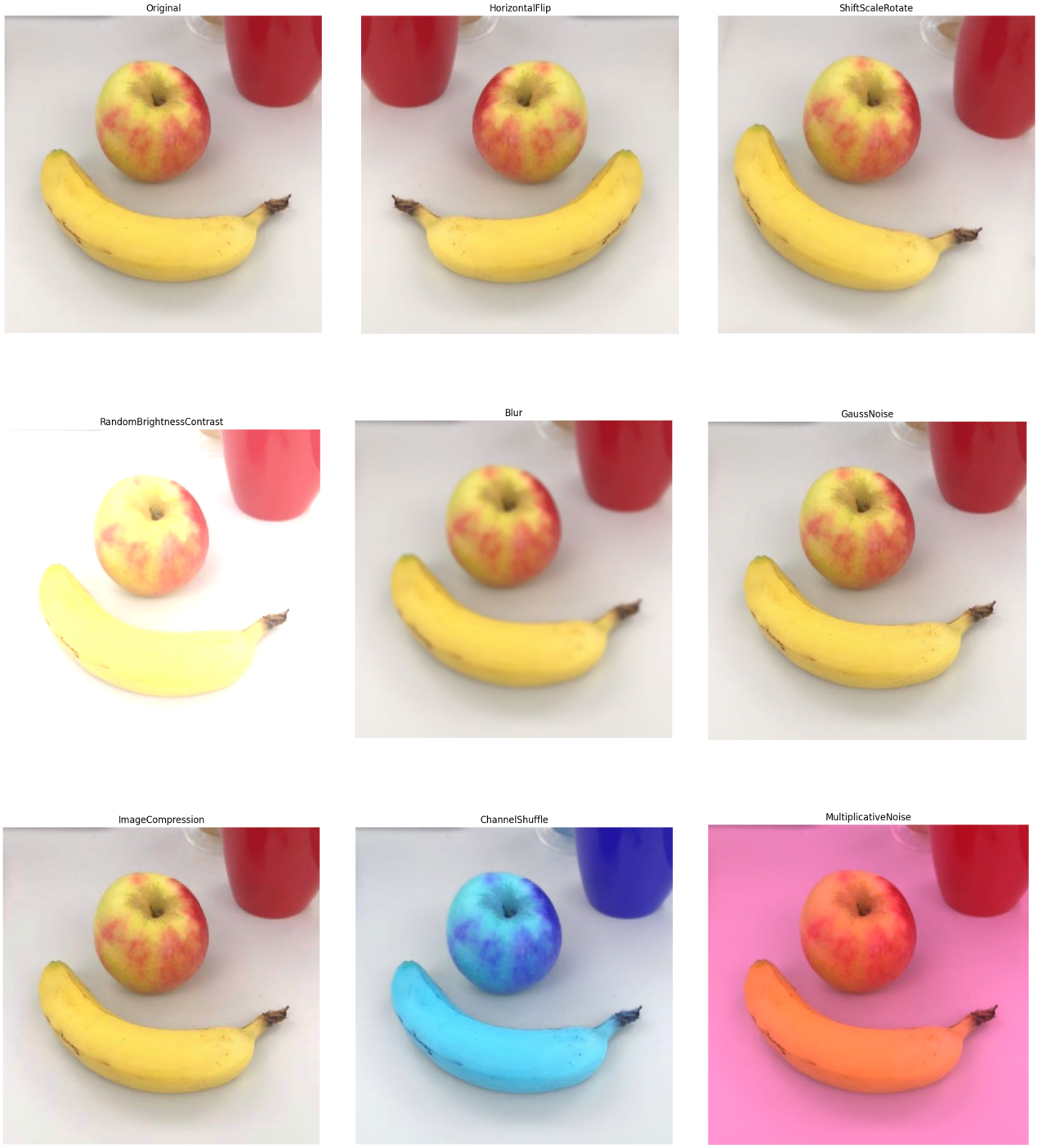



Figure 43: Data augmentation example

| Type | Params | Prob. | Description |
|---|---|---|---|
| Resize | img_scale=[(480, 480), (960, 960)]<br>keep_ratio=True | 1.0 | Resizes the images to different scales at each iteration, the scale is between the [min , max] |
| ShiftScaleRotate | shift_limit=0.0625<br>scale_limit=0.1<br>rotate_limit=20 | 0.7 | Randomly apply affine transforms: translate, scale and rotate the input |
| RandomBrightnessContrast | brightness_limit=[0.1, 0.3]<br>contrast_limit=[0.1, 0.3] | 0.3 | Randomly change brightness and contrast of the input image |
| Blur* | blur_limit=5 | 0.4 | Blur the input image using a random-sized kernel |
| MotionBlur* | blur_limit=5 | 0.4 | Motion blur the input image, that is the blur seen in moving objects in a photograph |
| GaussianNoise* | blur_limit=25 | 0.4 | Add gaussian noise to the input image |
| ImageCompression* | quality_lower=75 | 0.4 | Decreases jpeg with lower bound on the image quality |
| HorizontalFlip | | 0.5 | Flips the image horizontally |
| MultiplicativeNoise | multiplier=[0.5, 1.5]<br>per_channel=True | 0.1 | Multiply image to random number or array of numbers |

Table 1: Data Augmentation Types Adopted

* means that only one of the four types of augmentation is applied each time.

### 2.6 Evaluation criteria

For a known ground truth mask A, the model propose a mask B, therefore we can compute the IoU (Intersection over Union).

$$IoU = \frac{target \cap prediction}{target \cup prediction}$$

IoU measures the overall overlap between the true region and the proposed region. Then we consider it a True detection, when there is at least half an overlap, or when $IoU > 0.5$.

Instance segmentation models produce a collection of local segmentation masks describing each object detected in the image, therefore we need to calculate IoU of each mask.

To evaluate our collection of predicted masks, we'll compare each of our predicted masks with each of the available target masks for a given input.

- A **true positive** is observed when a prediction-target mask pair has an IoU score which exceeds some predefined threshold.
- A **false positive** indicates a predicted object mask had no associated ground truth object mask.
- A **false negative** indicates a ground truth object mask had no associated predicted object mask.

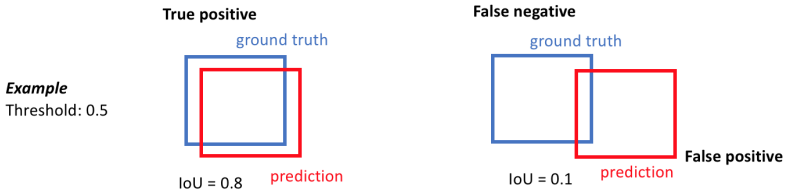



Figure 44: Example of True positive and False negative

- Precision: effectively describes the purity of our positive detections relative to the ground truth

$$precision = \frac{TP}{TP + FP}$$

- Recall: effectively describes the completeness of our positive predictions relative to the ground truth

$$recall = \frac{TP}{TP + FN}$$

In order to calculate the precision and recall of a model output, we'll need to define what constitutes a **positive detection**. To do this, we'll calculate the IoU score between each (prediction, target) mask pair and then determine which mask pairs have an IoU score exceeding a defined threshold value, setted to 0.5.

Typical scoring parameters $AP_{IoU>0.5}$ and $AR_{IoU>0.5}$ are computed by averaging over all the precision and recall values for all known annotations in the ground truth.

In COCO evaluation, the IoU threshold ranges from 0.50 to 0.95 with a step size of 0.05 represented as $AP_{[.5:.05:.95]}$.

The AP at fixed IoUs such as IoU=0.5 and IoU=0.75 is written as AP50 and AP75 respectively.

Unless otherwise specified, AP and AR are averaged over multiple Intersection over Union (IoU) values, specifically, 10 IoU thresholds of .50:.05:.95. This is a break from tradition, where AP is computed at a single IoU of .50 (which corresponds to our metric $AP_{IoU=0.50}$). Averaging over IoUs rewards detectors with better localization.

$$mAP = \frac{mAP_{0.50} + mAP_{0.55} + ... + mAP_{0.95}}{10}$$

AP is averaged over all categories. Traditionally, this is called *mean average precision* (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from context.

Metrics used in COCO evaluation are:

- $mAP_{[.5:.05:.95]}$: which is mAP averaged over 10 thresholds
- $mAP_{IoU=0.50}$: which is standard mean average precision
- $mAP_{IoU=0.75}$: which is a strict metric
- $mAP_{small}$: for small objects that cover areas less than $32^2$
- $mAP_{medium}$: for medium objects that cover areas greater than $32^2$ but less than $96^2$
- $mAP_{large}$: for large objects that cover areas greater than $96^2$
- $mAR_{max=1}$: which is mAR given 1 detection per image
- $mAR_{max=10}$: which is mAR given 10 detection per image
- $mAR_{max=100}$: which is mAR given 100 detection per image

- $mAR_{small}$: for small objects that cover areas less than $32^2$
- $mAR_{medium}$: for medium objects that cover areas greater than $32^2$ but less than $96^2$
- $mAR_{large}$: for large objects that cover areas greater than $96^2$

# 3 Experimental Study

In the experimental study we have chosen to test three different state-of-the-art models for instance segmentation:

- Mask RCNN
- Cascade Mask RCNN
- Hybrid Task Cascade

The first step was to configure the backbone network, the available possibilities were ResNet50, ResNet101 and ResNext101, we decided to use ResNet50 motivated by the fact that the computational cost was lower than the other two architectures, even if deeper models are able to achieve better training result as compared to the shallow networks [11]. In the reminder of the report R50 stands for ResNet50 and X101 stands for ResNext101.

Transfer learning is adopted for the backbone and pretrained weights on imageNet were used for the fist stage of the net (suggested in MMDetection as it is a common practice in object detection codebase such as Detectron) because we know that layers of convolutional networks compute feature maps of growing complexity. The weights learned in the first layers are likely to be independent from the particular kind of images they have been trained on. So it's a good idea to reuse them given the fact that they have been trained on a huge amount of data and are probably very good.

The second step was to define a training pipeline used to load the images, the annotations and doing preprocessing operations. Multiscale training was adopted, in which images were resized to different scales at each iteration and data augmentation process was used to making the model more robust to slight changes in the input data (as explained in section 2.5). The multiscale range of scales was set to $[(480, 480), (960, 960)]$, 480 is the minimum average dimension of images and 960 is a dimension in which most of the images in the dataset fall and has a reasonable computational cost.

The third step was to define a testing pipeline in which we used the test time augmentation technique that is used to improve the predictions of the model by averaging results. A fixed scale is used here ($800 \times 800$) that is a commonly used scale in AICrowd Food Recognition Challenge examples.

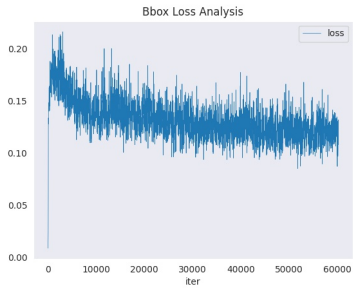
Then we defined a training procedure:

- Batch size: we set the batch size as bigger as possible given our computational power, 4 was the higher possible dimension.
- Learning rate: we used the learning rate indicated in the default model configurations, 0.0025 without lr-decay.
- Warmup: is a way to reduce the primacy effect of the early training examples. The learning rate is increased linearly over the warm-up period. Therefore, the learning rate started from 0.0001 and was increased of 0.0001 in the first 500 iterations, until it reached the value of 0.0025.
- Optimizer: momentum optimizer was used to increase the learning speed. Instead of using only the gradient of the current step to guide the search, momentum also accumulates the gradient of the past steps to determine the direction to go. In this way we are able to reduce the oscillations in the gradient steps.

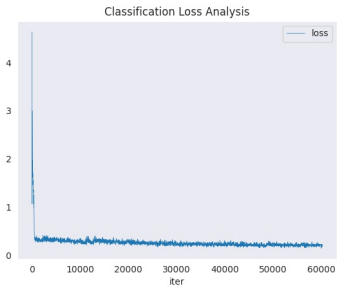
We trained all the three models for only 10 epochs because of the huge computational cost, instead of the 12 default epochs defined in MMDetection library and the 20 epochs used in AICrowd Challenge examples.

### 3.1 Mask RCNN R50 Results

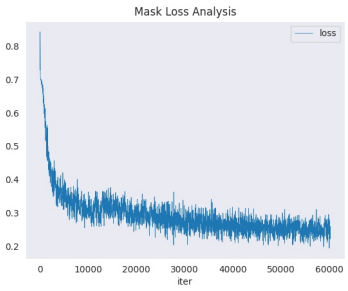
In this first experiment we tested the mask-RCNN model.



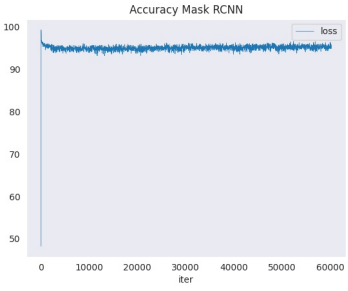

(a) Mask-RCNN Loss Bbox



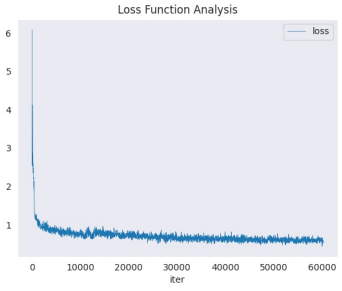

(b) Mask-RCNN Loss Classification




(c) Mask-RCNN Loss Mask




(d) Mask-RCNN accuracy




(e) Mask-RCNN Loss

Figure 45: Mask-RCNN Training curves

From the training curves we can see that the loss of the mask-RCNN model is about 0.50 after ten epochs, the mask loss is around 0.25, the bbox loss 0.15 and the classification loss 0.20.

By looking to the two different tasks of *object detection* and *instance segmentation*, the model performance is slightly the same, with $mAP_{IoU=0.50} \approx 0.25$.

The model performs better on instance segmentation than object detection for the object of medium size $mAP^{segm}_{medium} = 0.219 > mAP^{bbox}_{medium} = 0.173$, but in both of the cases the performance are better for object of medium size than on those of bigger or smaller size.

The recall related measures are all better for segmentation task, this can be due to the fact that the amount of false positive is greater in the case of detection task, as we can see in figure 47.

```
                                                                    bbox    segm
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.137 | 0.167
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.242 | 0.236
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.139 | 0.183
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.173 | 0.219
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.146 | 0.177
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.275 | 0.338
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.293 | 0.353
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.293 | 0.353
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.200 | 0.255
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.307 | 0.369
```

Figure 46: Mask-RCNN evaluation coco metrics summary

By looking to the detailed results in the figure 47 below, we can see that the classification is the

main error made by the model, than instead performs well in the other tasks, such as localization, background/foreground recognition, etc. Also, classification error is higher in detection task than in segmentation, that instead has a higher *Missing GT Error*, maybe due to the fact that segmentation requires more precise (pixel level) classification.

AP @ [50-95]

| Thresh | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|---|---|---|---|---|---|---|---|---|---|---|
| bbox AP | 23.60 | 23.05 | 21.56 | 19.75 | 16.95 | 13.59 | 9.32 | 4.41 | 1.28 | 0.18 |
| mask AP | 22.97 | 22.24 | 21.83 | 20.67 | 19.81 | 17.79 | 14.99 | 12.33 | 7.98 | 2.13 |

Main Errors

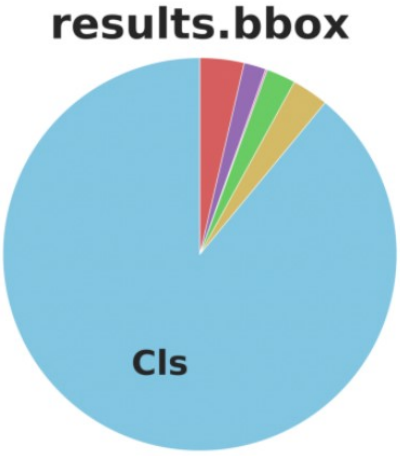
| Type | Cls | Loc | Both | Dupe | Bkg | Miss |
|---|---|---|---|---|---|---|
| bbox dAP | 44.20 | 1.49 | 1.19 | 0.07 | 0.89 | 1.80 |
| mask dAP | 39.60 | 2.16 | 1.07 | 0.13 | 1.21 | 2.33 |

Special Error

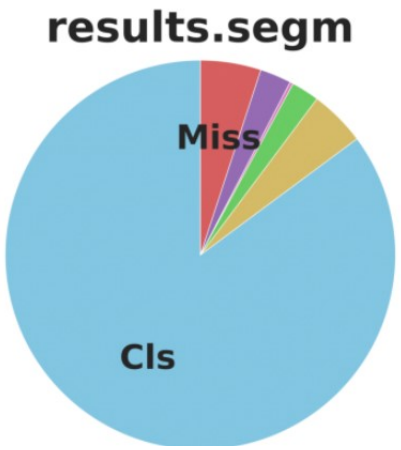
| Type | FalsePos | FalseNeg |
|---|---|---|
| bbox dAP | 24.56 | 13.39 |
| mask dAP | 23.95 | 14.03 |

Figure 47: Mask-RCNN evaluation detailed results

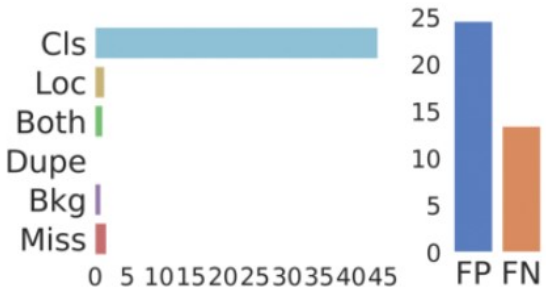
Also from the pie hart and barplot we can see that classification is the main source of errors and that in this case we have more false positives than negatives.



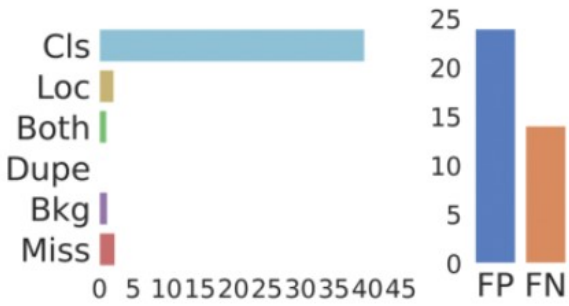

(a) Mask-RCNN Bbox error decomposition




(b) Mask-RCNN Segm error decomposition




(c) Mask-RCNN Bbox error types

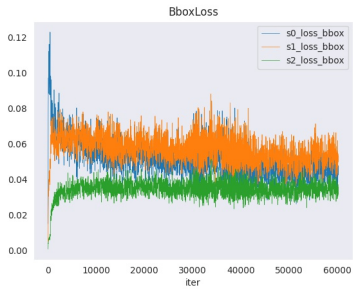



(d) Mask-RCNN Segm error types

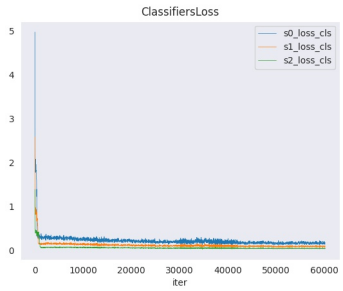
Figure 48: Mask-RCNN errors decomposition

### 3.2 Cascade Mask RCNN R50 Results

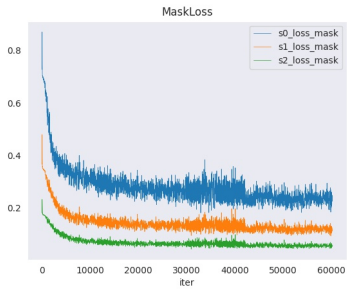
The second experiment we tested the cascade model, presented in section 1.4.



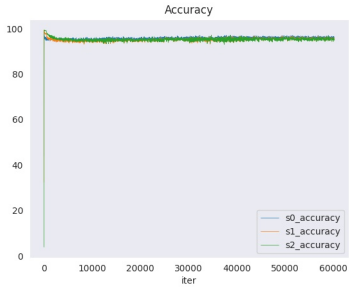

(a) Cascade Mask-RCNN Loss Bbox



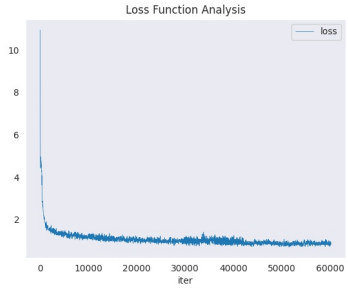

(b) Cascade Mask-RCNN Loss Classification




(c) Cascade Mask-RCNN Loss Mask




(d) Cascade Mask-RCNN accuracy




(e) Cascade Mask-RCNN Loss

Figure 49: Cascade Mask-RCNN Training curves

The Cascade R-CNN has multiple detection heads, each with its own loss. The stages differ only in the hypotheses at their input. Each stage is trained with the corresponding hypotheses, i.e. accounting for the input distribution changes. Stages can be optimized for the corresponding sample distributions and the performance improve across the stages. The detector becomes more selective against close false positives and specialized to the more precise hypotheses. For this reason the loss of the last stage is lower than the losses of previous stages.

The loss at the end of the tenth epoch is approximately 0.80, therefore higher than the loss of the mask model.

```
                                                                    bbox    segm
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.148 | 0.164
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.226 | 0.223
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.166 | 0.177
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.165 | 0.173
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.155 | 0.172
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.283 | 0.308
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.296 | 0.320
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.296 | 0.320
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.193 | 0.198
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.310 | 0.335
```

Figure 50: Cascade Mask-RCNN evaluation coco metrics summary

The results for the two tasks of *object detection* and *instance segmentation* are, as before, similar;

the $mAP_{ioU=0.50}$ is 0.226 for detection and 0.223 for segmentation. Instead we have higher performance in segmentation if we look to the average recall metrics.

However, the final results for the mask-RCNN model outperform those for cascade-RCNN model, e.g. segmentation results are: $mAP^{maskRCNN}_{IoU=0.50} = 0.236 > mAP^{cascadeRCNN}_{IoU=0.50} = 0.223$

As before the main source of error is classification, where we have similar values to those obtained in Mask-RCNN experiment. In this case the *Miss GT Error* is higher than before, so more object are not recognized at all by the network.

In this second experiment we have less false positives, but more false negatives that are correlated with the *Miss GT Error*.

AP @ [50-95]

| Thresh | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|---|---|---|---|---|---|---|---|---|---|---|
| bbox AP | 22.02 | 21.60 | 20.92 | 19.94 | 18.50 | 16.16 | 13.14 | 8.52 | 3.12 | 0.19 |
| mask AP | 21.76 | 21.44 | 20.73 | 20.25 | 18.69 | 17.21 | 15.20 | 12.82 | 8.81 | 2.89 |

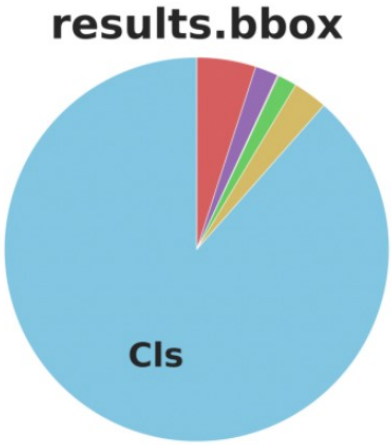
Main Errors

| Type | Cls | Loc | Both | Dupe | Bkg | Miss |
|---|---|---|---|---|---|---|
| bbox dAP | 44.11 | 1.44 | 0.80 | 0.05 | 0.97 | 2.47 |
| mask dAP | 40.87 | 1.78 | 0.75 | 0.06 | 1.14 | 3.37 |

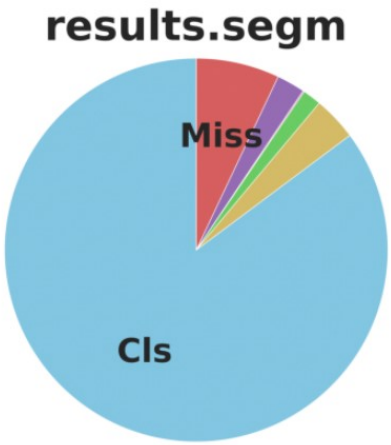
Special Error

| Type | FalsePos | FalseNeg |
|---|---|---|
| bbox dAP | 19.40 | 16.91 |
| mask dAP | 19.13 | 17.22 |

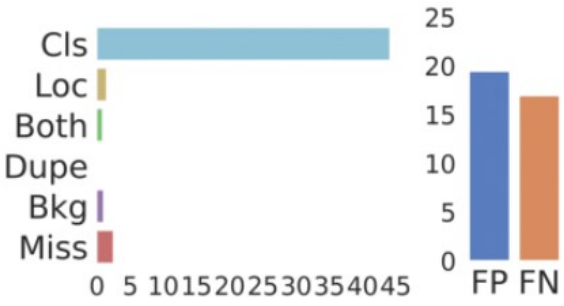
Figure 51: Cascade Mask-RCNN evaluation detailed results



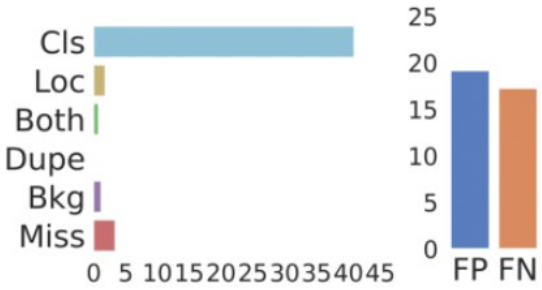

(a) Cascade Mask-RCNN Bbox error decomposition




(b) Cascade Mask-RCNN Segm error decomposition




(c) Cascade Mask-RCNN Bbox error types

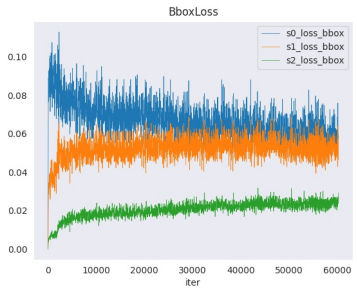



(d) Cascade Mask-RCNN Segm error types

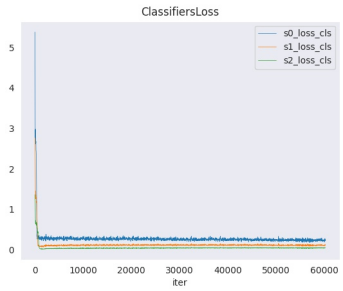
Figure 52: Cascade Mask-RCNN errors decomposition

### 3.3 HybridTaskCascade R50 Results

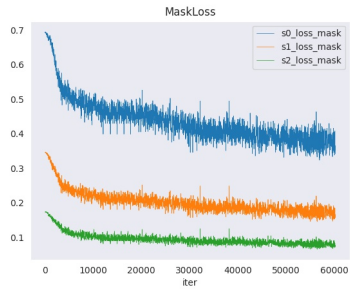
In the third experiment we tested the Hybrid task cascade model.



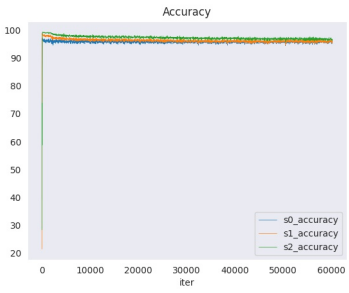

(a) Htc Loss Bbox



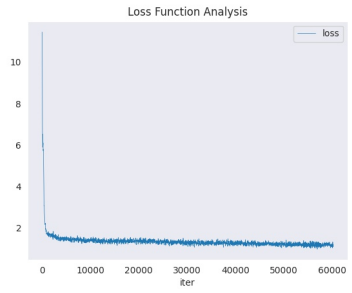

(b) Htc Loss Classification




(c) Htc Loss Mask




(d) Htc accuracy




(e) Htc Loss

Figure 53: Htc Training curves

The Hybrid Task Cascade model has poor performance compared to the other models. The mean average precision (IoU=0.50) for the segmentation task is 0.059 compared to 0.236 that we got for the mask-RCNN model. But also all the other metrics show very low scores which are not satisfactory at all.

```
                                                                    bbox    segm
Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.029 | 0.041
Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 0.058 | 0.059
Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 0.028 | 0.044
Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.026 | 0.051
Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.031 | 0.043
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.158 | 0.212
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.176 | 0.229
Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.176 | 0.229
Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = 0.000 | 0.000
Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.058 | 0.094
Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.188 | 0.242
```

Figure 54: Htc evaluation coco metrics summary

The classification is, also in this case, the main type of error and it's much more higher than the previous two cases (see also graphs in figure 56).

The average recall metrics obtain discrete results if compared to precision ones, this is due to the fact that the number of false negative is very low, instead the number of false positive is quite high, as in the case of mask-RCNN model.

AP @ [50-95]

| Thresh | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|---|---|---|---|---|---|---|---|---|---|---|
| bbox AP | 5.66 | 5.32 | 4.89 | 4.48 | 3.65 | 2.73 | 1.33 | 0.33 | 0.08 | 0.00 |
| mask AP | 5.77 | 5.57 | 5.43 | 5.14 | 4.83 | 4.27 | 3.44 | 2.77 | 2.09 | 0.37 |

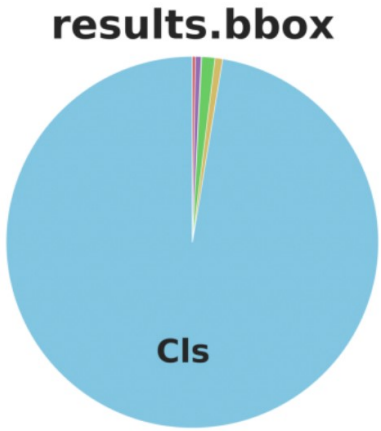
Main Errors

| Type | Cls | Loc | Both | Dupe | Bkg | Miss |
|---|---|---|---|---|---|---|
| bbox dAP | 58.77 | 0.40 | 0.70 | 0.02 | 0.30 | 0.16 |
| mask dAP | 51.60 | 0.41 | 0.55 | 0.01 | 0.43 | 0.31 |

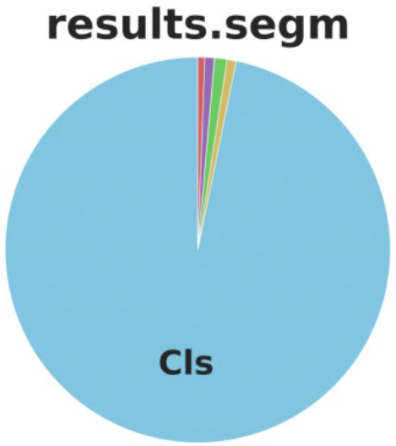
Special Error

| Type | FalsePos | FalseNeg |
|---|---|---|
| bbox dAP | 27.54 | 3.73 |
| mask dAP | 26.94 | 3.97 |

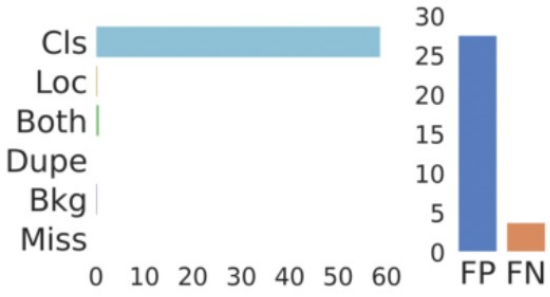
Figure 55: Htc evaluation detailed results



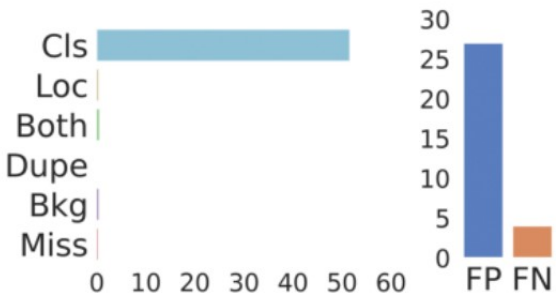

(a) Htc Bbox error decomposition




(b) Htc Segm error decomposition




(c) Htc Bbox error types




(d) Htc Segm error types

Figure 56: Htc errors decomposition

The fact that the HTC model get worse results than the others two models, even if it should be the most powerful model, can derive from the fact that the number of parameters for this model is higher and therefore it requires a longer training time. Instead, we trained all three models for the same amount of time (ten epochs). We can see this phenomenon also by comparing Mask-RCNN model and Cascade Mask-RCNN model, the second is more powerful then the first, but it does not overcome the other given the same amount of training time.

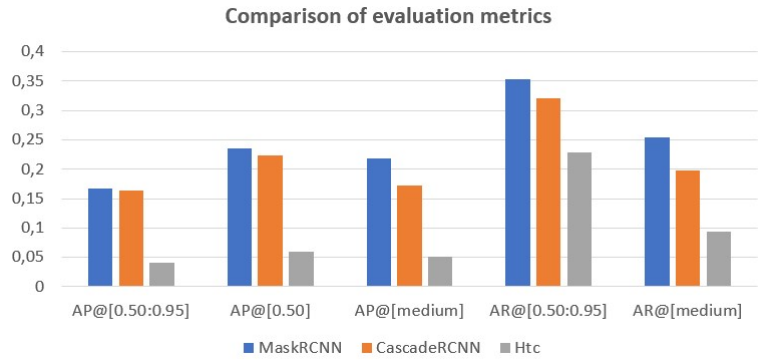



Figure 57: Comparison of some evaluation metrics of the three tested models

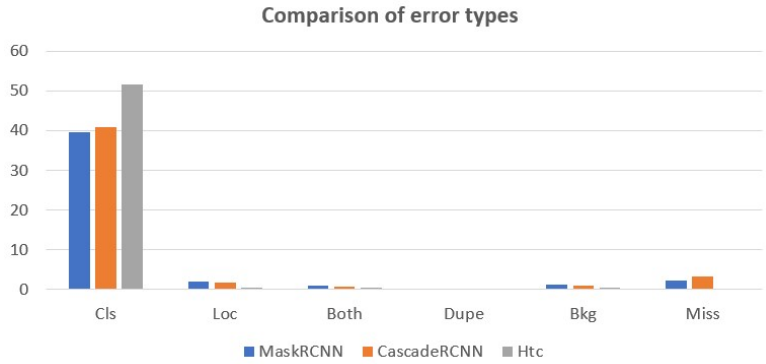



Figure 58: Comparison of the types of errors of the three tested models

By looking to the subsequent table we can see that the number of parameters for the cascaded models is extremely high if compared to mask-RCNN model. This means that the amount of training time needed to tune the parameters is much more higher.

We can see this also from the number of Floating Point operations, FLOPs are often used to describe how many operations are required to run a single instance of a given model. FLOPs can be also used to describe the computing power of given hardware like GPUs and conversely to know how long it may take to train a model on that hardware. GFLOPs means GigaFlops, that is billion of FLOPs.

However, the number of operations and parameters of Cascade Mask-RCNN and HTC is quite similar, but the second get results that are much worse than the first. This could be due to the random weights initialization, the models start learning from different points in the search space and the first achieve a better result.

| **Model** | **Number of parameters** | **GFLOPs** |
|---|---:|---:|
| Mask-RCNN R50 | 45.21 M | 265.01 |
| Cascade Mask-RCNN R50 | 77.84 M | 406.18 |
| Htc R50 | 77.97 M | 408.77 |

Table 2: Models complexity

More on the parameters, ResNet50 (the backbone of the models) has 23.283 M of parameters, this represents the 51.49% in the case of Mask-RCNN model and approximately 30% in cascaded models.

The FPN, that is the neck of the models, has 3.344 M of parameters and represents the about the 7,38% in the Mask-RCNN model and the 4,25% in cascaded models.

The big difference is, obviously, in the RoIHead where the Mask-RCNN model has 17.989 M of parameters (39,78%) while the other two models have triple, since they consists of 3 consecutive stages

similar to that of mask-RCNN model. The RoIHead in cascaded models has more or less the 65% of the parameters of the networks.

Therefore, not surprisingly, the difference between cascaded and non-cascaded models is huge, there is a trade-off between power and complexity, a more powerful model means greater complexity.

### 3.4 HybridTaskCascade R50 Pretrained Results

In this fourth experiment, we tried to train again the Htc model, but this time we started by using available weights from Round 2 of AICrowd food recognition challenge. In the second round of the challenge only 60 categories were available, compared to the 273 of the last round. The weights refers to a model trained for 20 epochs with ResNet50 as backbone.

We also made some modifications to try to make the learning process more effective:

- we have changed the weight associated to the classification loss, we increased it from 1.0 to 2.0, to try to correct the problems highlighted previously;
- we lowered the maximum number of detections per image from 100 to 30, this because the maximum number of items in a single image is 12, therefore it makes not much sense to have a so big number of possible detections;

| | | | | | bbox | segm |
|---|---|---|---|---|---|---|
| Average Precision | (AP) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] | = | 0.182 | 0.228 |
| Average Precision | (AP) @[ IoU=0.50 | area= all | maxDets=100 ] | = | 0.309 | 0.302 |
| Average Precision | (AP) @[ IoU=0.75 | area= all | maxDets=100 ] | = | 0.193 | 0.247 |
| Average Precision | (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] | = | 0.000 | 0.000 |
| Average Precision | (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] | = | 0.172 | 0.193 |
| Average Precision | (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] | = | 0.191 | 0.240 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] | = | 0.376 | 0.462 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] | = | 0.394 | 0.481 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area= all | maxDets=100 ] | = | 0.394 | 0.481 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] | = | 0.000 | 0.000 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] | = | 0.209 | 0.254 |
| Average Recall | (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] | = | 0.416 | 0.504 |

Figure 59: Htc pretrained evaluation coco metrics summary

AP @ [50-95]

| Thresh | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|---|---|---|---|---|---|---|---|---|---|---|
| bbox AP | 29.90 | 29.12 | 27.90 | 26.17 | 23.02 | 18.69 | 12.85 | 6.10 | 1.89 | 0.23 |
| mask AP | 29.21 | 28.80 | 28.13 | 27.10 | 25.78 | 23.93 | 21.74 | 18.21 | 13.21 | 4.87 |

Main Errors

| Type | Cls | Loc | Both | Dupe | Bkg | Miss |
|---|---|---|---|---|---|---|
| bbox dAP | 31.09 | 1.66 | 1.58 | 0.04 | 1.18 | 2.06 |
| mask dAP | 26.89 | 2.90 | 1.06 | 0.07 | 1.67 | 2.45 |

Special Error

| Type | FalsePos | FalseNeg |
|---|---|---|
| bbox dAP | 33.29 | 7.86 |
| mask dAP | 32.32 | 8.79 |

Figure 60: Htc pretrained evaluation detailed results

The results show that the difference between this experiment and the previous is huge, the $mAP_{IoU=0.50}$ is equal to 0.309 against the 0.059 obtained previously. This is also the best result obtained among all the tested models.

We can see that the classification error dropped down to 26.89 for instance segmentations, this shows that:

- transfer learning is very powerful, especially in this case where we used weights of a model trained on very similar dataset for initialization;
- increasing the weights for classification loss could be useful to focus the model more on this type of errors;
- decreasing classification error leads to improvements for the final mAP;

However we can also see that there is still a lot of room for improvement, indeed the $mAP_{IoU=[0.50:0.95]}$ is only 0.182. More training time and/or more data are needed to improve the AP and AR scores.

### 3.5 HybridTaskCascade X101 Results

In this last experiment we show, analyse and compare results obtained from Hybrid Task Cascade model with ResNext101 as backbone of the network.

The model has been in training for 20 epochs, therefore much longer than all previous cases.

| | | | | | bbox | segm |
|---|---|---|---|---|---|---|
| Average Precision | (AP) | @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = | 0.505 | 0.471 |
| Average Precision | (AP) | @[ IoU=0.50 | area= all | maxDets=100 ] = | 0.619 | 0.609 |
| Average Precision | (AP) | @[ IoU=0.75 | area= all | maxDets=100 ] = | 0.578 | 0.537 |
| Average Precision | (AP) | @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = | 0.500 | 0.600 |
| Average Precision | (AP) | @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = | 0.482 | 0.425 |
| Average Precision | (AP) | @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = | 0.520 | 0.482 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area= all | maxDets= 1 ] = | 0.740 | 0.676 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area= all | maxDets= 10 ] = | 0.757 | 0.688 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area= all | maxDets=100 ] = | 0.757 | 0.688 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = | 0.500 | 0.600 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = | 0.538 | 0.464 |
| Average Recall | (AR) | @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = | 0.779 | 0.705 |

Figure 61: Htc ResNext101 evaluation coco metrics summary

AP @ [50-95]

| Thresh | 50 | 55 | 60 | 65 | 70 | 75 | 80 | 85 | 90 | 95 |
|---|---|---|---|---|---|---|---|---|---|---|
| bbox AP | 59.86 | 59.25 | 58.90 | 58.30 | 57.30 | 55.91 | 52.74 | 45.40 | 30.13 | 10.36 |
| mask AP | 58.92 | 57.79 | 57.15 | 55.82 | 54.34 | 51.93 | 47.03 | 37.79 | 25.29 | 9.45 |

Main Errors

| Type | Cls | Loc | Both | Dupe | Bkg | Miss |
|---|---|---|---|---|---|---|
| bbox dAP | 15.80 | 0.72 | 1.36 | 0.00 | 0.96 | 0.57 |
| mask dAP | 13.61 | 1.99 | 0.95 | 0.02 | 1.60 | 1.16 |

Special Error

| Type | FalsePos | FalseNeg |
|---|---|---|
| bbox dAP | 31.05 | 2.75 |
| mask dAP | 26.93 | 5.53 |

Figure 62: Htc ResNext101 evaluation detailed results

The difference is clear by looking to the scores, the model far exceeds previous results, getting a $mAP_{IoU=0.5}$ of 0.609 in instance segmentation task and 0.619 in detection task. So it doubled the scores obtained in the previous experiment.

The difference between the two models is given by the backbone, the higher number of layers of the X101 net result in a much more greater number of parameters for the entire model. We also know that more layers means more powerful model because deeper ResNet achieve better training result if compared to the shallow networks [11]. The number of parameters of ResNet50 is 23.283 M and that of ResNext101 is 80.838 M. This means also a higher number of GigaFlops needed.

| Model | Number of parameters | GFLOPs |
|---|---|---|
| Htc R50 | 77.97 M | 408.77 |
| Htc X101 | 140.84 M | 645.94 |

Table 3: HybridTaskCascade models complexity

It can be seen from figure 62 that the classification error has more than halved, however it still remains the main source of error.

We now are going to analyse class-wise metrics to try to determine if some of these categories mostly affect the results.

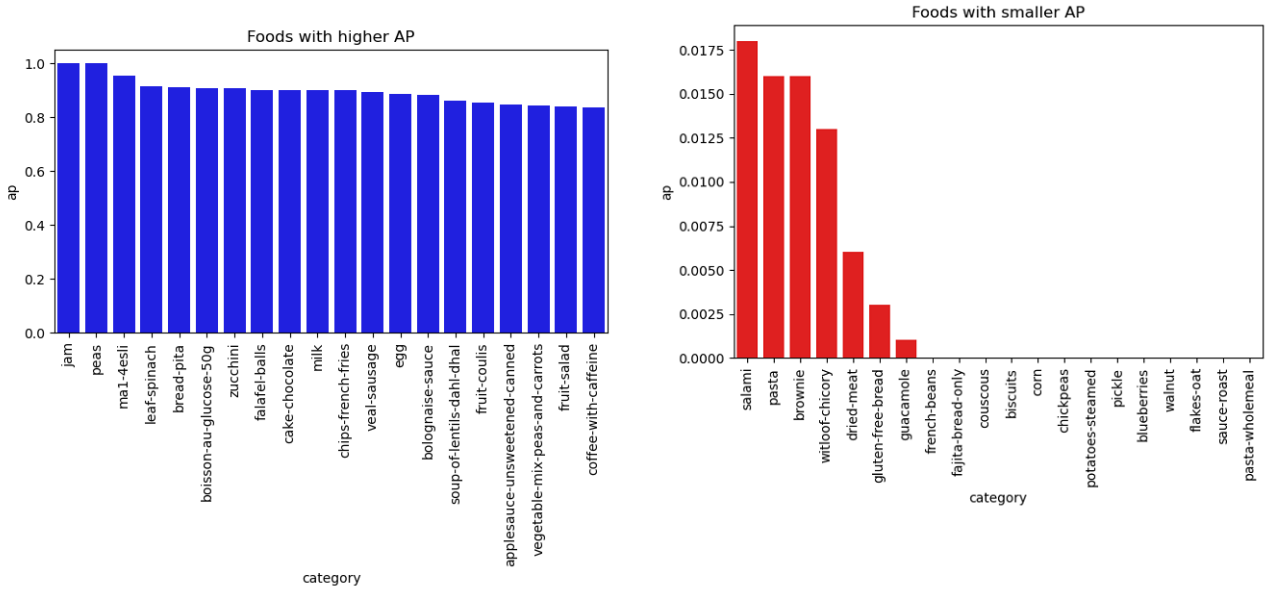



Figure 63: Twenty food items with higher and smallest mAP

By analysing class-wise results for average precision, we can see that only for two categories we get 100% of precision. For 12 categories we get an average precision of zero. Therefore we can notice a big difference among the various classes.

However, there is not a direct correlation between the number of images per class in the training set and the average precision values obtained for the categories. Indeed, there are categories with many images and categories with a small number of images that obtain good and bad AP scores.

From the figure 64 we can see that there are many classes with AP score near zero, but the higher number of food categories get a score between 0.45 and 0.75.

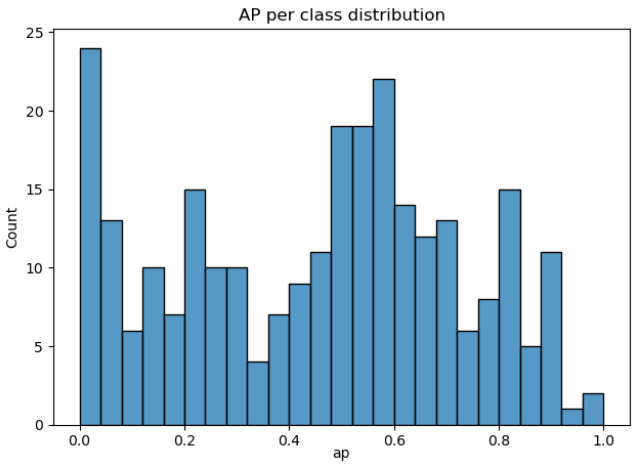



Figure 64: Mean average precision per class distribution of values

We have also analysed the types of errors for the classes with smallest AP scores (figure 63), only for some of them the main source of error is classification, which however is very high, greater than 40 for 12 of them.

For the other classes, especially for 'corn', a high source of error is Miss GT Error, meaning that in many cases the class is not recognized. This is due to the fact that in many cases there are this corn seeds with other foods, such as salad or tomatoes, and the networks tend to categorize them in other classes, such as 'Mixed Vegetables'.

In one case, 'salami', the main source of error is Localization, this could be due to the fact that typically the salami is cut into slices that are in different positions in the image and perhaps not all of them are recognized by the algorithm.

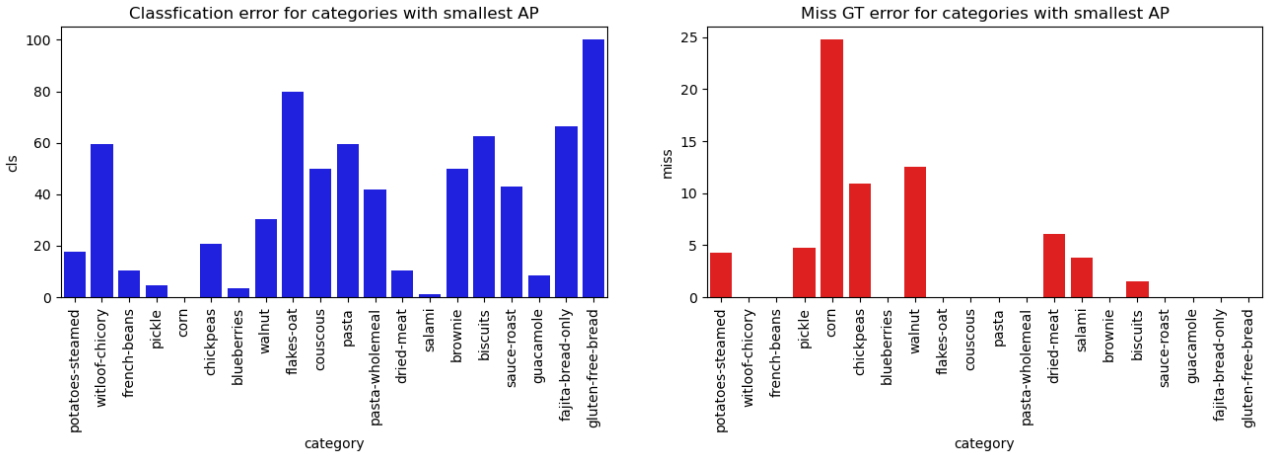



Figure 65: Two main errors for the food items with smallest mAP

The figure 66 highlight that the main source of error is Classification, but we can see that more than 35 classes does not have this type of error. There are also many classes with very high classification error.

For the other types of error we can see that a very small number of classes have high values for them, meaning that the algorithm is quite good in localization of items and distinction between background and foreground.

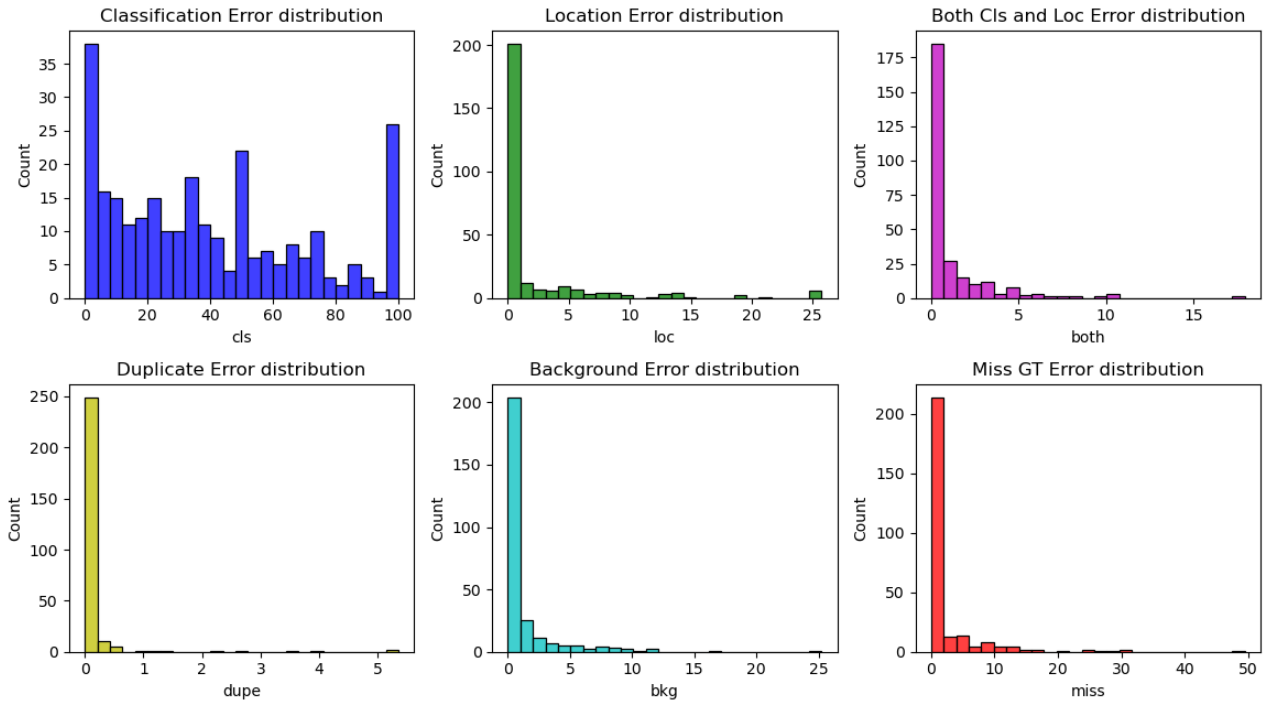
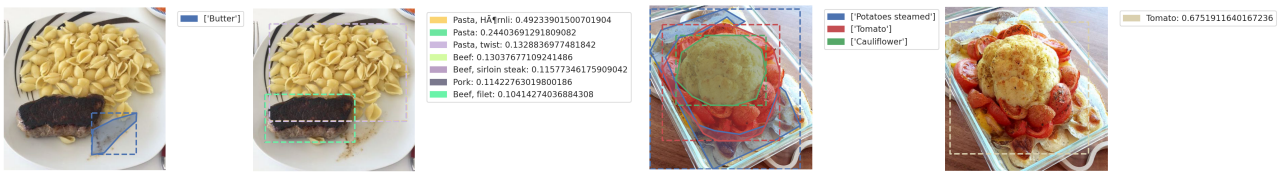



Figure 66: Distributions of values for 6 different type of errors

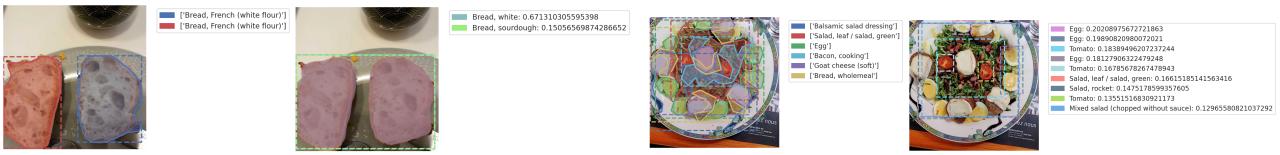
These errors in some cases are caused by the problems already mentioned in Data Exploration (section 2.3), i.e. images partially annotated or not perfectly annotated or again categories very similar among them.

Some examples of errors:



(a) Image erroneously annotated

(b) Overlapped items



(c) Similar Categories

(d) Difficult image

Figure 67: Ground Truth image VS predicted detection

Moreover, by analysing the confusion matrix we saw that there are some categories for which we have many common mistakes (Table 68). Water is sometimes confused with other light coloured drinks, like syrup or white wine; Coffee, Bread and Tea are confused with similar categories (as also show in figure 67); Butter is typically confused with cheese and Tomato with other red food items, like Strawberry and Pepper.

| | top1 | top2 | top3 | top4 | top5 |
|---|---|---|---|---|---|
| **Water** | Water, mineral | Syrup (diluted, ready to drink) | Boisson au glucose 50g | Water with lemon juice | Wine, white |
| **Butter** | Honey | Cheese | Fresh cheese | Soft cheese | Jam |
| **Bread, white** | Bread, French (white flour) | Bread, half white | Braided white loaf | Bread | Bread, sourdough |
| **Banana** | Almonds | Pear | Apple | Biscuits | Kiwi |
| **Salad, leaf / salad, green** | Mixed salad (chopped without sauce) | Salad dressing | Balsamic salad dressing | French salad dressing | Salad, rocket |
| **Tomato** | Sweet pepper | Carrot | Caprese salad (Tomato Mozzarella) | Strawberries | Red radish |
| **Coffee, with caffeine** | Espresso, with caffeine | Ristretto, with caffeine | White coffee, with caffeine | Tea, green | Sauce, soya |
| **Tea** | Coffee, with caffeine | Tea, black | Tea, green | Herbal tea | Tea, peppermint |

Figure 68: Categories with main errors

# 4 Conclusions

Food image segmentation is a very interesting, but also difficult problem. We saw that SOTA models perform very well on segmentation and detection tasks, they output bounding boxes and masks with no problem and low error. However, the models have difficulties in the classification task, likely due to the large number of very similar classes.

The dataset is quite large and well annotated, but there are still some problems that need to be fixed and that can affect the learning process of the models, such as images not completely annotated.

We saw that training require a very long time, especially for more complex models, and that transfer learning, not surprisingly, is very useful to help the models converge faster and to increase the models performance.

Concluding, deep neural networks are capable to recognize with discrete quality food items in real images taken by real people. This models could be used as a base tool for interesting applications, such as mobile apps that track users food consumption, estimate food weights, calculate food calories, or also for the automatic creation of diets etc.

We obtained good results with the tested convolutional neural networks, but we can also think to some other pre/post process that could be useful to improve the current results. Possible future work could be:

- *reformat the classes*: remove similar categories that are difficult to recognized also for humans and which we have proven to be the major source of error;
- *reduce the class imbalance*: this makes the problem very interesting from the perspective of machine learning, but also very hard because this affects the learning process in a negative way;
- *oversampling techniques*: these techniques could be used to try to solve the imbalance problem by oversampling the classes with less images;
- *ensemble learning techniques*: to ensemble predictions from different models based on confidence scores;

# References


[1] Food Recognition Challenge. AICrowd. (2020)..

[2] FREITAS, Charles NC; CORDEIRO, Filipe R.; MACARIO, Valmir. MyFood: A Food Segmentation and Classification System to Aid Nutritional Monitoring. In: 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI). IEEE, 2020. p. 234-239..

[3] Uri Almog. Object Detection With Deep Learning: RCNN, Anchors, Non-Maximum-Suppression. The Startup - Medium. (2020).

[4] Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. Towards Data Science. (2018).

[5] PULKIT SHARMA. A Step-by-Step Introduction to the Basic Object Detection Algorithms (Part 1). Analytics Vighya. (2018).

[6] Lilian Weng. Object Detection for Dummies Part 3: R-CNN Family. Lil'Log. (2017).

[7] Shaunak Halbe. Object Detection and Instance Segmentation: A detailed overview. The Startup - Medium. (2020).

[8] Sik-Ho Tang. Reading: Cascade R-CNN — Delving into High Quality Object Detection (Object Detection). Medium. (2020).

[9] Jonathan Hui. Understanding Feature Pyramid Networks for object detection (FPN). Medium. (2018).

[10] Nathan Hubens. Test Time Augmentation (TTA) and how to perform it with Keras. Towards Data Science. (2019).

[11] Neuro Hive. ResNet (34, 50, 101): Residual CNNs for Image Classification Tasks. (2019).

[12] CAI, Zhaowei; VASCONCELOS, Nuno. Cascade R-CNN: high quality object detection and instance segmentation. IEEE transactions on pattern analysis and machine intelligence, 2019.

[13] CHEN, Kai, et al. Hybrid task cascade for instance segmentation. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019. p. 4974-4983.

[14] CHEN, Kai, et al. MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155, 2019.

[15] CHEN, Kai, et al. MMDetection: Open mmlab detection toolbox and benchmark. arXiv preprint arXiv:1906.07155, 2019.

[16] Google Colaboratory, 2020, available at https://research.google.com/colaboratory/faq.html..

[17] Lin, Tsung-Yi, et al. "Microsoft coco: Common objects in context." European conference on computer vision. Springer, Cham, 2014..

[18] BOLYA, Daniel, et al. Tide: A general toolbox for identifying object detection errors. arXiv preprint arXiv:2008.08115, 2020..