Let's Find Momo

Yipeng Su
yipengsu@stanford.edu

Yun Kun (Wilson) Zhang
wyzhang@stanford.edu

Abstract

Few-Shot object detection(FSOD) problems have attracted great research interest in the community due to traditional object detectors relying on large supervised object detection datasets. In this paper, we try to solve a real world FSOD problem - Let's find Momo [5]. This work explores the problem from a Generative Adversarial Network(GAN) perspective by building a generator generating candidate images to train the discriminator. We show that the generator alone helps to increase our top 1 accuracy performance from 0% to 25% and top 3 accuracy performance to 40%. In the end, we cropped and resized the test images to further improve the top1 accuracy to 34% and top 3 accuracy further to 50%.

1. Introduction

Let's find Momo is a series of children's hide-and-seek books. Momo, a lovely border collie, loves hiding in any places. Our job is to find Momo in each picture. This task becomes very challenging when Momo fades into a large landscape background or merges into a group of other objects. In this paper, we try to use Deep Learning for Computer Vision technologies to help to solve this problem. The formal description of this game belongs to object detection area where we learn to detect the position of target objects in given pictures. Three things make the problem more difficult are 1) In all pictures, Momo is hiding behind some objects, cars, desks, buildings or peoples. We only can identify the target by part of its body(face, in most of the time). 2) The target is very small in the images. Momo is never the main object in the image and actually in the most of images it takes less than 5% pixels. 3) We don't have large amount of training examples. We attach one example of Momo Challenges here. See how long will you use to find Momo. 1

The input to our algorithm is a batch of images containing Momo. We then use a Single-Shot Detector(SSD) to output predicted bounding boxes.

Figure 1. Find Momo Challenge Example. Can you identify MoMo?

2. Related Work

General Object Detection. General object detection algorithms can be divided into two categories: proposal-based(two stage) and proposal-free(one stage). R-CNN [2] series detectors fall into the first category. They first use the region-proposal network to propose the potential regions containing a targe object and then use the classification network to predict the object. While one stage detector like SSD [8], YOLO [11] and RetinaNet [6], they don't rely on the RPN network to select the regions. So they are conceptually simpler and significantly faster [1] [12] [7]. However both of them require thousands of examples to achieve decent performance.

Few-shot Obejct Detection. Few-shot object detection refers to training a model on novel (unseen) object classes with little data, it's usually done by performing transfer learning on deep models. Compared to the normal solution, this is often a preferable choice without extra data collection. More importantly, the source-domain knowledge is an effective supervision to generalize the learning procedure of target domain, when the training

set is scarce [4] [1]. Kang et al [4] proposed a method using a meta feature learner and a re-weighting module within a one-stage detection architecture. Fan et al [1] proposed a method using Attention-RPN, Multi-Relation Detector and Contrastive Training strategy, which exploits the similarity between the few shot support set and query set to detect novel objects while suppressing false detection in the background

Small Object Detection. Small-object detection is a challenging problem. Many object detectors struggle with effective features extraction for small-scale objects. Rabbi et al [10] applied a generative adversarial network (GAN)-based model to solve the problem. The model has three components: enhanced super-resolution GAN(ESRGAN) [14] that generates realistic textures from low-resolution pictures, Edge Enhancement Network (EEN) that removes noise and enhances the extracted edges from an image, and Detection network. Shamsolmoali et al [13] proposed integrating an image pyramid network into SSD to achieve more strong semantic features from the small objects.

Self Supervised Learning. Huang et al. [3] describes various self supervised learning methods for object detection. The methods he can be categorized into four types of losses: self-distillative (such as SoCo, EsViT and BYOL), predictive (such as DETReg and UP-DETR), constrastive (such as DETcon, MoCo and SimCLR) and clustering-based (such as SwAV).

3. Methods

As stated in the introduction, Finding MoMo falls into the problems of Few-shot Object Detection and Small Object Detection. We attempted both ways with Few-shot Object Detection as a baseline model and Small Object Detection as the final model.

3.1. Transfer Learning

Both the baseline model and the final model use the transfer learning approach. We leverage a pre-trained deep neural network model RetinaNet. The model will be trained in the large MSCOCO object detection data-sets, and then we update the model to let the model only focus on one target class. We re-train the new model with corresponding MoMo datasets. We picked RetinaNet as a starting point because it utilizes Focal Loss function instead of cross entropy loss. This Focal Loss Function can handle class imbalance problem caused by many background objects. The Focal Loss function adds a modulating factor $(1 - p_t)^\gamma$ to the cross entropy loss, with tunable focusing parameter $\gamma \geq 0$. Specifically, Focal Loss Function is defined as [7]:

$$FL(p_t) = -(1 - p_t)^\gamma log(p_t)$$

The one-stage RetinaNet network architecture uses a Feature Pyramid Network (FPN) backbone on top of a feed-forward ResNet architecture to generate a rich, multi-scale convolutional feature pyramid. To this backbone RetinaNet attaches two subnetworks, one for classifying anchor boxes and one for regressing from anchor boxes to ground-truth object boxes. Our approach is to hold the box subnet unchanged and retrain the classification subnet. Specifically, we update the target class to 1 and restore the box regression weights, then we retrained the model on the corresponding MoMo Dataset.

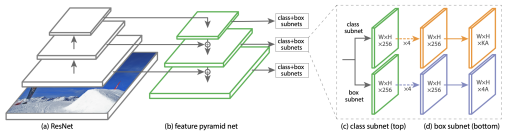


Figure 2. RetinaNet pipeline.

3.2. Baseline Model

We attempted a few-shot object detection model. Since in vast majority of the MoMO pictures, MoMo is in the only dog in the scene. We figured it will be helpful to incorporate other border collie's picture in the training. To overcome MoMo being small, we cropped the MoMo images to sub-images and only included the sub-image with MoMo in training datasets. To illustrate, we cropped the image into 16 images and the 10th image contained MoMo. We then resized the 10th image and included it in the training data.

Figure 3. Cropping Image.

Together 120 general border collie pictures and 40 cropped MoMo pictures are included in training. In test

time, we cropped the images and ran each sub-image into the baseline model. We then compared the score for the top one bounding box of each sub-image, kept the bounding box the has the highest score, and concatenated all sub-images back to one single image.

Figure 4. Model Output

The accuracy rate for this model is about 0.24. The shortcomings of the baseline model are 1) MoMO sometimes appears in the boarder line; 2) resizing made the training pcitures blurry; 3) the training samples are still limited. The details of experiment for this model is not described in section 4 and 5 since it's not the final model.

3.3. Final Model

To overcome the shortcomings of the baseline model, we experimented a GAN like model to train the detector with a generator. The generator generates the candidates similar with Momo images. And the discriminator(object detector) will finally solve the find Momo challenges by learning to find the Momo in the artificial images. This method is inspired by the GAN and self-supervising model in the lecture. The high level model training process is illustrated as below:

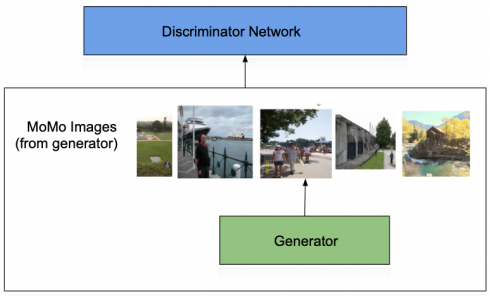


Figure 5. Model Training Pipeline

Note it's a little different from our initial models. In our previous attempts, we tried to mix labeled Momo Data into training data or add another fine-tuning stage after the model trained with generator. The reason we decide to move those components is we have too few Momo Data to test after splitting them into train, evaluation and test subsets. And the actual accuracy improvement is also trivial.

As for the loss function, the object detection loss consists of two parts: box regression loss and classification loss. Since we want the model weights are more sensitive to the classification error, so we decide to put a relative larger weight on the classification loss. Final total loss in our model = $0.3 \times localization\_loss + 0.7 \times classification\_loss$.

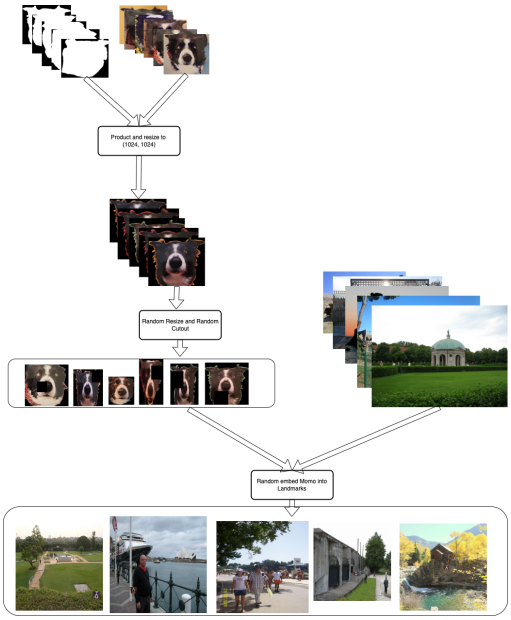


Figure 6. Generator Architecture

The generator is designed to provide Momo-Like challenges to the discriminator. Generator pre-processed the Momo Dataset by masking, resizing and cutting out then randomly embed them into background images 6. We talk more details about the Dataset processing in Section 4. With introducing this generator into our model, even we didn't use any original Momo images in our training data, our model still outperforms the baseline model.

Cropping the test images really helps and improves our accuracy by 10% in average. And it works much better on those hard challenges where the Momo is very small. See examples in Figure 15. Cropping also brings us some side affects when the Momo appears right on the cropping lines.

To reduce overfitting, we changed our detector model to RetinaNet50 from RetinaNet101, apply 0.5 dropout rate at detector network and add weights regularization. More discussion in the Experiment Overfitting Section 5.

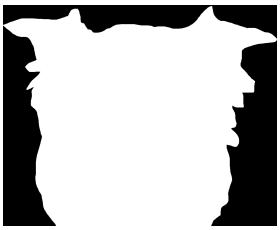
4. Dataset and Features

We used two datasets in our model, Google-Landmarks Dataset [9] and self-collected Momo Dataset.

4.1. Training Data

From our observation, in most of the Find Momo challenges, we only can find part of Momo's head in the picture. And the majority part of its body is hiding behind other objects. So in order to mimic those images, we also only use Momo heads in our generator. We call them Momo avatars 7a. To further mimic the Momo hiding in the picture, we apply a random cutout on the Momo avatar.

(a) Example of Momo Avatar.



(b) Example of Momo Mask.

Figure 7. We use the product of those images pairs as the target Momo in generated candidates.

For the background of candidates, we randomly select 10000 images from Google-Landmarks Dataset. The landmark dataset consists of a variety of landscapes and city views. We believe those are perfect backgrounds for the generator. See the example of training images generated by our generator 8b.

All of our training images have the shape as (1024, 1024, 3).

4.2. Eval & Test Data

We use the Momo dataset as the evaluation and test data. We also resize the images to (1024, 1024) and convert to RGB channels. In the final accuracy improvement, we apply the 4×4 cropping on the test images. We use 25 images of Momo as the evaluation dataset and left images as test dataset.

5. Experiments

We use the pre-trained SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50) as our discriminator. We update the predict number of class to 1 and restore all but the classification layer at the top weights. Then we use the training data from generator to train the SSD model.

(a) Example of Landmark Dataset.

(b) Example of generated candidate.

Figure 8. We selectively randomly resize the Momo avatar images and embed them into the landmark images. Can you find where is Momo in this generated image?

Learning rate Since we used a pre-trained model, we should not use a large step size to update the weights. But if the step size is too small, we will need longer training time to converge.

From Figure 9, we can tell when learning rate is too small(lr=1e-4), it takes very long time and still not able to converge. While learning rate is large(lr=1e-2, 1e-3), it can quickly converge to the optimal direction. So in the end we choose the lr=1e-3 as our final learning rate.

Optimizer We use the common SGD with momentum optimizer in our model.

Batch size We choose to use 30 as the batch size because this is the largest one we could use in the AWS server. Interestingly, we don't see much training speed improvement from large batch size. 10

We tried 10, 20, 30 batch size in our model training, and we don't see marginally difference between. Our assumption is that the object detection tasks created by generator are relatively easy tasks and the model

doesn't need too much training data to learn the target objects. 5

Figure 9. Loss history of different learning rate.

Figure 10. Loss history of different mini-batch size.

Accuracy for different Batch Size			
Metrics	Size = 10	Size = 20	Size = 30
Top1	23%	23%	25.6%
Top3	35.1%	37.8%	39.2%
Top3 crop16	47.3%	50%	54.1%

5.1. Final accuracy performance

Since we only dozens of unlabeled data, we use the accuracy for the main metric.

$$\text{accuracy} = \frac{\text{count of images with correct bounding boxes}}{\text{count of images}}$$

Below is a table showing final experiment results with selected hyper-parameters.

5.2. Hyperparameter Tuning

Hyper-Parameters Used In Model	
Parameter	Value
learning rate	1e-3
optimizer	SGD
momentum	0.9
mini-batch size	30
cropping size	9 or 16

Top1 We use the original test images as predict input and choose the bounding box with highest confidence score as the output.

Top3 We use the original test images as predict input and choose the bounding boxes with top 3 highest confidence scores as the output. If any one among these boxes have Momo inside, we consider it's a right prediction.

Top3 crop16 We first crop each test image into 16 sub-images. We use the sub-images as predict input and only keep the bounding boxes with highest score for each sub-image. Then we pick top 3 among 16 bounding boxes as output.

Top3 crop9 We first crop each test image into 9 sub-images. We use the sub-images as predict input and only keep the bounding boxes with highest score for each sub-image. Then we pick top 3 among 9 bounding boxes as output.

Final Model Performance	
Metric	Accuracy
Top1	25.6%
Top3	39.2%
Top3 crop9	48.6%
Top3 crop16	54%

5.3. Overfitting

We do see signals showing overfitting signals from the evaluation loss history 11. As we can see from the chart, the training loss converges to the minimal value and stays there for the remaining epochs. While the evaluation loss stops decreasing after it hits 0.8.

The first thing we do to mitigate the overfitting is to reduce the model complexity by replacing the ResNet101 with ResNet50. Then we apply the drop out with keep ratio as 0.5 and weights regularization to further combat against overfitting. However, a combination of all of them doesn't make it better. 12 We examine the prediction results of those generated images, and we confirm the 100% accuracy. Then we realize the problem is from the training data. To clarify

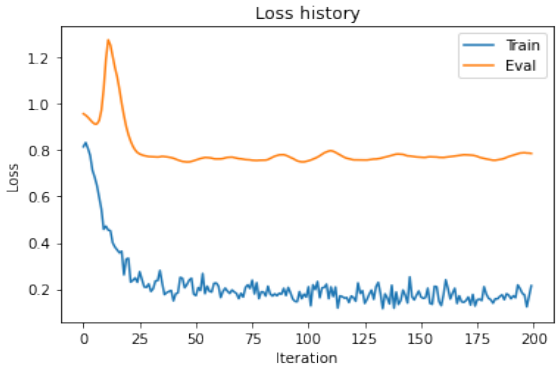


Figure 11. Loss history of Training Data and Eval Data of SSD Resnet101.

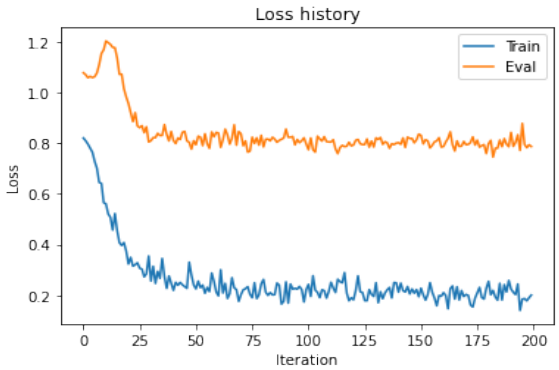


Figure 12. Loss history of Training Data and Eval Data of SSD Resnet50 with dropout and regularization.

here, the training images are all from generator and evaluation images all come from Momo Dataset.

Our assumption of this gap is that although we have as much training data as we want, those artificial images from the generator are still notably different from original Momo test images. So even if the model performs very well on the training data, we are unable to get the same results on the test data.

We try to mix some labeled Momo images into the training data, but still we see the same overfitting evaluation loss curve. Our explanation to this is that the proportion of labeled Momo data is too small in the whole training data (25 vs 6000), so in the end the model learns how to solve the problem given by the generator instead of the real Momo challenges. This is also a very common issue existing in every GAN model which we need to sovle in future work.

5.4. Prediction Results

Have you found Momo in the beginning picture? Our model does!

Figure 14a 14b shows that we are able to find Momo in

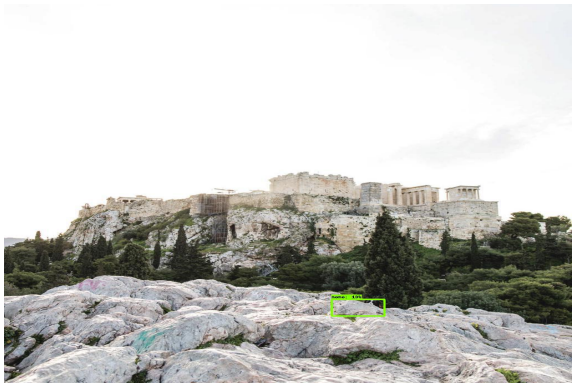
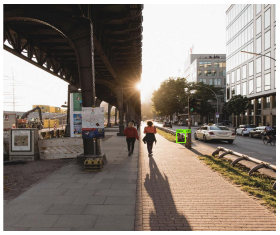
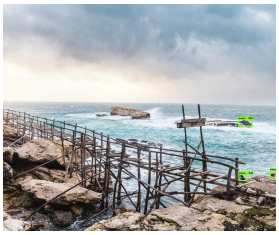


Figure 13. Predict bounding box with Momo.

(a) Momo is hiding inside the bushes.



(b) Momo is hiding behind the block.



(c) Momo is hiding under the bridge.

(d) Momo is hiding inside the cherry yard.

Figure 14. More predict results.

the highest confidence score bounding box. Figure 14c 14d shows that we are able to find Momo in the top 3 highest confidence score bounding box.

Figure 15 shows that we fail to find Momo in the original test images, however, we are able to find it in the cropped images.

To justify a little bit why we only use the accuracy for evaluation metrics, we don't have perfect labeled train/test Dataset to use and our data are either created by generator or manually collected. Since we are targeting to solve the Find Momo task, it doesn't make much sense to evaluate our model on other dataset. We don't use Intersection Over Union(IOU) either. The first reason is our first goal is to find the Momo instead of drawing a perfect bounding box which is also why we give larger weights on the classification loss than localization/regression loss. The second reason is for some challenges we are not able to label the data perfectly.

(a) Fail to find Momo in the original image.

(b) Find Momo in one of cropped images.

(c) Fail to find Momo in the original image.

(d) Find Momo in one of cropped images.

Figure 15. More predict results in cropped test images.

Compared to the regular object detection images, our targets are very small and meant to not be found. Please see some samples from our results section.

6. Future Work

We believe that introducing the GAN concept into object detection tasks is a very promising approach to solve the problem of lack of training data. In our experiment, although the generator is not perfect and the generated candidates are very far from the test Dataset, we still could get a large improvement with the same model/algorithm. Couple of things in our model we can improve in the future.

- Current generator needs some domain knowledge about the target and backgrounds. We selectively collect the Momo avatars and choose the Landmark Dataset, so it's hard to generalize our current model.

- The current generator is a statistic which cannot be updated via gradients of loss. We need to update the generator network to generate better candidates in each time step just like the real GAN model.

7. Contributions & Acknowledgements

7.1. Acknowledgements

We use the Tensorflow Object Detection API in our project. The public Colab: https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/eager_few_shot_od_training_tf2_colab.ipynb

7.2. Contributions

Author Information: Yipeng Su and Yun Kun (Wilson) Zhang

Yipeng did the literature review, designed the model framework, trained the model, evaluated the performance and wrote the paper.

Yun Kun performed the literature review, tested baseline model and participated in drafting the final paper.

References

[1] Qi Fan, Wei Zhuo, Chi-Keung Tang, and Yu-Wing Tai. Few-shot object detection with attention-rpn and multi-relation detector, 2019. 1, 2

[2] J.; Darrell T.; Malik J. Girshick, R.; Donahue. Feature hierarchies for accurate object detection and semantic segmentation, 2014. 1

[3] Gabriel Huang, Issam Laradji, David Vazquez, Simon Lacoste-Julien, and Pau Rodriguez. A survey of self-supervised and few-shot object detection, 2021. 2

[4] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. Few-shot object detection via feature reweighting, 2018. 2

[5] Andrew Knapp. Let's find momo website. http://www.letsfindmomo.com/. 1

[6] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2999–3007, 2017. 1

[7] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2017. 1, 2

[8] D.; Erhan D.; Szegedy C.; Reed S.; Fu C.Y.; Berg A.C Liu, W.; Anguelov. Ssd: Single shot multibox detector, 2015. 1

[9] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465, 2017. 4

[10] N.; Schubert M.; Chowdhury S.; Chao D. Rabbi, J.; Ray. Small-object detection in remote sensing images with end-to-end edge-enhanced gan and object detector network, 2020. 2

[11] S.; Girshick R.; Farhadi A. Redmon, J.; Divvala. You only look once: Unified, real-time object detection, 2016. 1

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. 1

[13] M.; Granger E.; Chanussot J.; Yang J. Shamsolmoali, P.; Zarepoor. Enhanced single-shot detector for small object detection in remote sensing images, 2022. 2

[14] K..; Wu S.; Gu J.; Liu Y. Wang, X.; Yu. Esrgan: Enhanced super-resolution generative adversarial networks, 2018. 2