Improve Model Inference Cost with Image Gridding

Shreyas Krishnaswamy

May 19, 2023

Improve Model Inference Cost with Image Gridding

by Shreyas Krishnaswamy

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Joseph Gonzalez
Research Advisor

5/18/2023

(Date)

* * * * * * *

[illegible]

Professor Trevor Darrell
Second Reader

(Date)

Improve Model Inference Cost with Image Gridding

by

Shreyas Krishnaswamy

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Joseph E. Gonzalez, Chair
Professor Trevor Darrell

Spring 2023

Abstract

Improve Model Inference Cost with Image Gridding

by

Shreyas Krishnaswamy

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Joseph E. Gonzalez, Chair

The overwhelming success of AI has spurred the rise of Machine Learning as a Service (MLaaS), where companies develop, maintain, and serve general-purpose models such as object detectors and image classifiers for users that pay a fixed rate per inference. As more organizations incorporate AI technologies into their operations, the MLaaS market is set to expand, necessitating cost optimization for these services, particularly in high-volume applications. We explore how a simple yet effective method of increasing model efficiency, aggregating multiple images into a grid before inference, can significantly reduce the required number of inferences for processing a batch of images with varying drops in accuracy. To counter the slight decrease in object detection accuracy, we introduce *ImGrid*, an innovative technique that decides when to reprocess gridded images at a higher resolution based on model confidence and bounding box area assessments. Experiments on open-source and commercial models show that *ImGrid* reduces inferences by 50%, while maintaining low impact on mean Average Precision (mAP) for the Pascal VOC object detection task.

Contents

List of Figures

List of Tables

Acknowledgments

This work is a joint effort between Shreyas Krishnaswamy, Lisa Dunlap, Lingjiao Chen, Matei Zaharia, James Zou, and Joseph E. Gonzalez.

I would like to express my gratitude to Professor Joseph Gonzalez for being an excellent advisor throughout the program. I would like to thank my research mentor, Lisa Dunlap, for giving fantastic guidance and advice. I really appreciate your deep technical expertise and continuous support, which made it possible to complete this work. I would also like to thank Lingjiao Chen for collaborating throughout the program and providing technical insights that helped advance this project.

I would like to thank my family. Amma, Appa, and Harsha– your continued support and confidence in me made my academic endeavors possible. Thank you for all your patience and guidance. I couldn't have done it without you.

Finally, thank you to all my friends, fellow students, colleagues, TAs, teachers, and professors that have helped me along the way.

Chapter 1

Introduction

Recently, large models such as DINO and ChatGPT have shown remarkable progress on a wide range of tasks in computer vision and natural language processing. However, developing, training, and serving these models requires significant up-front investment in hardware, engineering, and data. As a consequence, large organizations like Microsoft, Google, and AWS, have taken a lead in developing these technologies and are increasingly offering Inference-as-a-Service to provide smaller organizations access to these large-scale pre-trained models. By using these inference services, smaller organizations are able to bypass the costly up-front investments needed for state-of-the-art predictions and instead pay a fixed rate per inference request. Meanwhile, large organizations are able to leverage economies of scale to reduce the costs of maintaining and serving these models while also gaining access to usage data that allows them to further refine their models and services.

In order for inference services to be cost-effective on hardware accelerators, requests must be processed in batches. Batching inputs improves arithmetic intensity [16] and can significantly increase the overall throughput of inference hardware [5]. However, this also means that inference services must transform their inputs into fixed-size structures that can be batched together effectively. This presents a unique opportunity for users of inference services to reduce the number of prediction requests – and as a consequence their costs – by cleverly packaging multiple inputs into a single request.

For example, consider object detection services which take a single image and return the bounding boxes of detected objects in the image, such as the example boxes proposed in Figure 1.1. To leverage efficiencies from batching, companies providing these object detection services must scale input images to a fixed size, so multiple images can be packed together into a single tensor. To remain competitive, these companies must also ensure their models maintain high accuracy across diverse and challenging images with many objects at different scales. For complex tasks that require high accuracy, such as semantic segmentation, this means the user must provide high resolution images that contain information even when scaled to a smaller fixed size before batching. However, many common applications (such as doorbell cameras detecting a person at the door) don't require the full

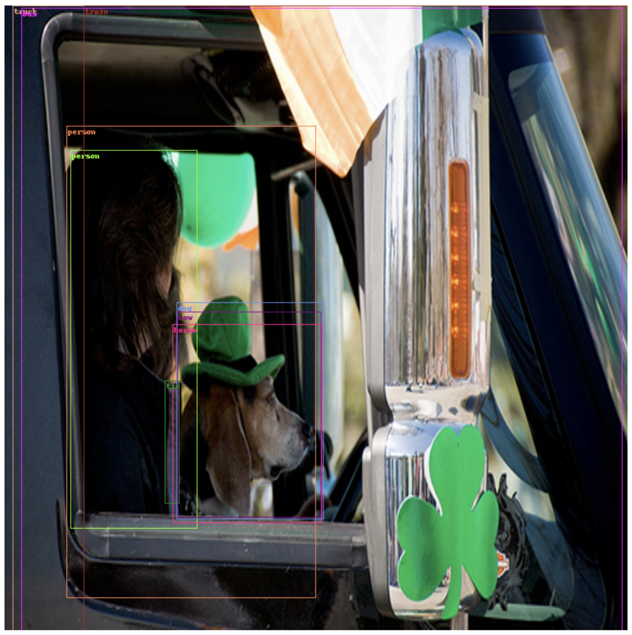


Figure 1.1: Object detection result on an image from the COCO dataset [11]. The object contains a person and a dog inside a truck. The ML model proposes different bounding boxes and labels for these objects.

resolution of the image and may be down-scaled without significant accuracy loss. This creates an opportunity for service users to down-scale multiple images and arrange them in one larger image to send as a single prediction request. In other words, spatially batching images makes better use of a single prediction request by packing more information into the same fixed input size of a model.

This paper investigates the utility of *image gridding*, a straightforward technique that consolidates multiple images into a single inference. Contrary to traditional batching, which stacks images in a tensor for computational efficiency, image gridding spatially arranges images into a grid. Instead of submitting a single image per query, the user can query the model on the grid of images and post-process the results to obtain the per-image predictions. By reducing the number of inferences needed to analyze a set of images, this technique enhances model efficiency and reduces user cost. For example, Figure 1.2 shows 25 images packed together into a single image grid. The resulting bounding boxes and labels were detected via a single inference.

The primary tradeoff for this technique is balancing efficiency against accuracy. Given that models have a maximum input size, images must be resized to lower resolutions to fit within the grid. As a result, the more images incorporated into the grid, the more likely it is that the accuracy for each individual image will decline. For instance, while the image grid in Figure 1.2 offers cost savings, some of the images contain objects that were missed or misclassified, reducing the model's accuracy.

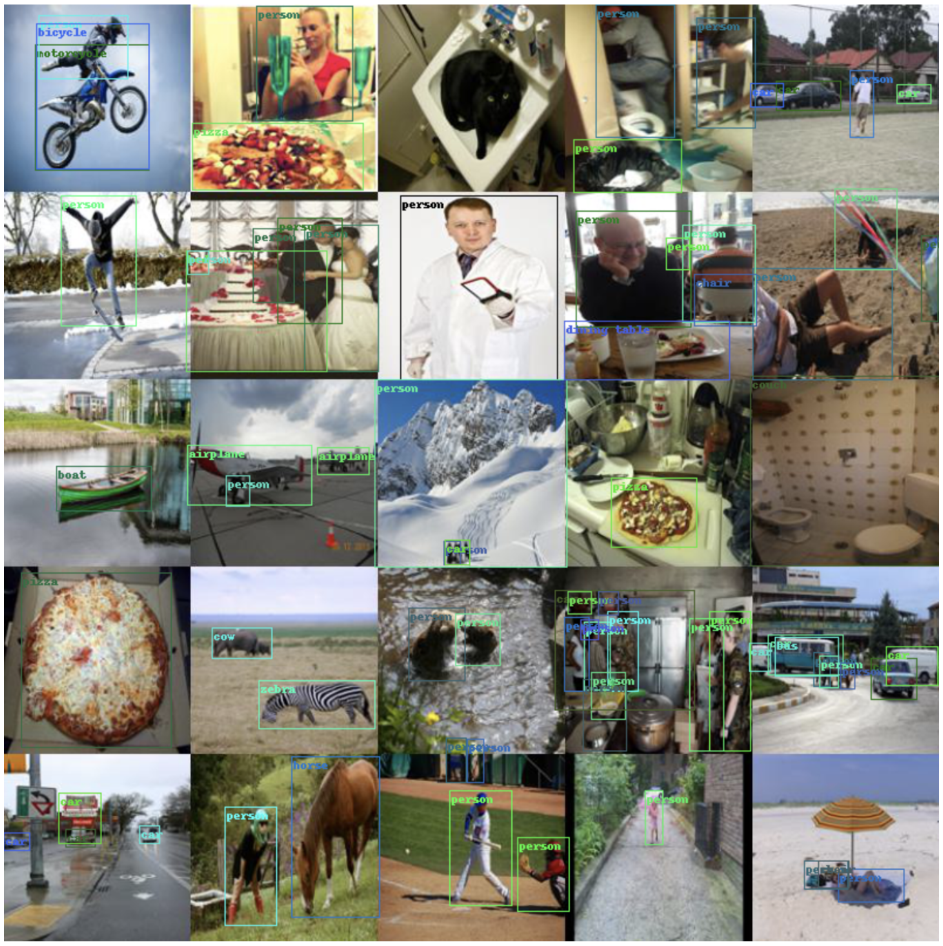


Figure 1.2: Object detection result on an image grid constructed from COCO dataset images [11]. Many of the images' objects are successfully detected, providing significant cost savings. However, some objects are missed or misclassified.

To address this accuracy tradeoff, we introduce ***ImGrid***, a method to determine which images likely produced inaccurate model inferences and should be retried at a higher resolution. With *ImGrid*, users can initially grid their images at a high grid size, such as a 5x5 grid containing 25 images, and submit the grid to the model. *ImGrid* uses the confidence scores and the bounding box sizes in the results to determine which images should then be retried at a lower grid size, such as a 2x1 or even a 1x1 (individual image).

Image gridding and *ImGrid* substantially reduce inferences needed to process a set of images. Across open source and commercial models, we observe that *ImGrid* can reduce inferences by 50% for object detection tasks on the Pascal VOC datasets with a typically low reduction in mAP.

Chapter 2

Background

2.1 Cost Model

MLaaS providers [8][1][2] typically charge by usage. Each inference incurs a small fee per task requested on the inference. For instance, suppose a user requests two tasks on a single image: face detection and object detection. The MLaaS provider will charge once for the face detection and once for the object detection. However, they typically will not charge for the *number* of faces or objects in the image; they'll only charge once for the overall task. Regardless of the number of objects detected, the user will pay the same amount. Table 2.1 contains sample prices for object detection across different cloud providers.

2.2 Size Limits

MLaaS providers do not typically impose a resolution limit on input images. Rather, they impose a moderate file size limit (about 4-5 MB per image). Users may submit images with large height and width dimension to the MLaaS provider, as long as the image's file size remains small. However, while particular details about the MLaaS provider's ML model is usually black-boxed, typical computer vision models resize images to a fixed maximum size before processing them. In other words large images are likely downscaled before processing.

Cloud Provider (Region)	Base Price per Image (USD)	Base Tier Limit
Google Cloud Vision	$0.0015	5,000,000
AWS Rekognition (us-east-2)	$0.001	1,000,000
Azure Cognitive Service (Central US)	$0.001	1,000,000

Table 2.1: Price per image for object detection task across different cloud providers. The prices provided are for the base tier, which is limited by a number of inferences per month. Once a user exceeds the base tier inference limit, the price changes.

Chapter 3

Method

3.1 Image Gridding

Given a constant stream of data to label, we aim to maximize *model utilization* by minimizing the number of model forward passes required to label a window of images. We assume a small labeled dataset of images to use for profiling.

A model has a fixed input image height H and input image width W, meaning the model takes in a total fixed input size of $H \times W$. We arrange r rows of images with c images each into an image grid, so the grid contains $r \cdot c$ total images, each with the same dimensions $h_{img} \times w_{img}$. We then resize each grid, so the total grid size is $H \times W$, which matches the model's input size. This image grid is then submitted to the model API as a single image and the model predictions are post-processed to obtain per-image labels. As an aside, fixing the input size is straightforward for open source models, since the model's input size is usually well-documented. However, details about the MLaaS providers' ML model architectures are rarely advertised. We emulate the effect of the model's fixed size by resizing all grids in our evaluation to the same fixed size. This should be reasonable since the MLaaS provider can then resize the fixed image sizes to the true input size, a transformation that's consistent across all image grids submitted to the API.

We sketch this process for a few sample tasks.

Object Detection: Running an object detection model over a grid of images returns a set of bounding boxes, labels, and confidence scores. Since the model treats the overall grid as a single image, the coordinates for each bounding box are mapped to the overall grid, rather than the specific image that it belongs to. Each bounding box can be mapped back to its image by converting its grid-based coordinates to the image coordinates. Given a bounding box's grid-based coordinates $(x_{g,min}, y_{g,min}, x_{g,max}, y_{g,max})$ and image size $h_{img} \times w_{img}$:

Definition 1: Converting gridded coordinates to image coordinates

$$\text{Image row index} = I_r = \left\lfloor \frac{y_{g,min} + y_{g,max}}{2 \cdot h_{img}} \right\rfloor$$

$$\text{Image col index} = I_c = \left\lfloor \frac{x_{g,min} + x_{g,max}}{2 \cdot w_{img}} \right\rfloor$$

$$x_{img} = x_g - (I_c \cdot w_{img})$$

$$y_{img} = y_g - (I_r \cdot h_{img})$$

x_{img} and y_{img} are the image coordinates for the bounding box. This formulation assumes that the bounding box is mapped to the image containing its midpoint, so there's no ambiguity when assigning a bounding box to an image when the box intersects multiple images. Alternatively, clients can simply discard a bounding box if it intersects multiple images.

Classification: Classification can be performed by running an object detection model on the grid, assigning each bounding box to an image, and aggregating the bounding boxes' labels for each image. Generally, running object detection inferences is more expensive than simple image classification; however, the gains from image gridding may outweigh the added cost, depending on the particular model and dataset.

Safe Search: Safe search models provide a likelihood that the image is unsafe. Since pure object detection models are rarely trained to detect unsafe images, users can adopt a divide-and-conquer strategy. If an image grid is determined unsafe, it likely contains at least one unsafe image. The grid can then be divided into two subgrids, and each subgrid can be rerun. This recurses until the specific unsafe images are found. If the number of unsafe images in the dataset is relatively small, this approach can save inferences by quickly accepting entire groups of images that lack unsafe material.

We focus on object detection in our experiments due to its popularity.

3.2 *ImGrid*

We describe *ImGrid*, a method that uses cascading grid sizes to preserve accuracy while still capturing image gridding efficiency gains. The key intuition behind *ImGrid* is that for typical object classes, most instances are either relatively hard to detect, meaning downscaling them would cause them to be missed or misclassified, or they're relatively easy to detect, meaning even when they're downscaled relatively aggressively (e.g. by being placed in a 5x5 grid), they're still classified correctly. We address this by running all images in a high grid size (e.g. 5x5) where they're relatively cheap to run, and then we use features in the model's response to determine whether to retry some images at a lower grid size (e.g. 2x1).

High grid sizes offer the greatest savings since they pack together many images in a single inference. However, resizing images to lower resolutions inherently removes information. Therefore, in general, as the grid size increases and individual image resolution decreases, model accuracy tends to diminish. We can group these detection failures at high grid sizes into two categories: misclassifications and missed detections. The challenge is detecting which images contain these errors. We thus rely on model confidence to detect potential false positives and average bounding box area to infer potential missed detections.

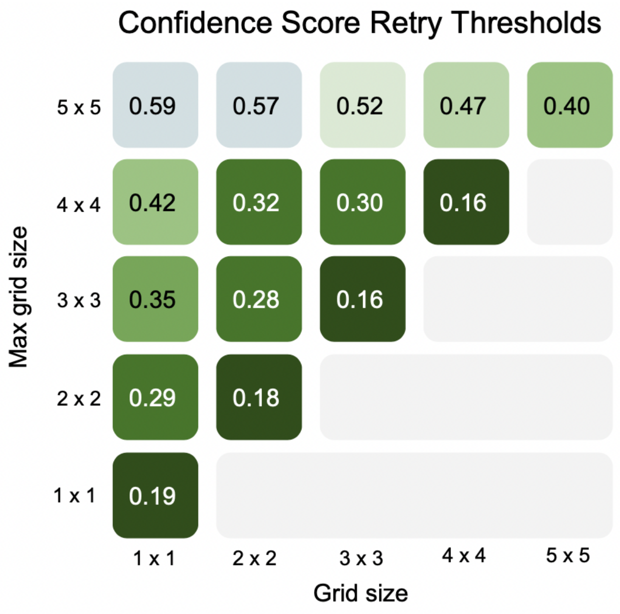


Figure 3.1: Heatmap of confidence score thresholds for the "motorcycle" object in the COCO dataset [11]. These scores were constructed using 100 images that contain motorcycles. The x-axis represents the grid size that the motorcycle was detected in, and the y-axis represents the maximum grid size in which the motorcycle can still be detected. For instance, on average, motorcycles that can be detected in a 4x4 grid have a confidence score of 0.42 when gridded in a 1x1. Only square grids (1x1 to 5x5) were profiled for these values, but a similar map could be constructed using rectangles. Note that having a max grid size of 5x5 in this figure means only that the object is still detectable in a 5x5. We did not profile higher grid sizes, so some of the motorcycles in this category may still be detected in higher grid sizes. During inference, if an object's confidence score is lower than the corresponding threshold, the image is retried.

***ImGrid*– Profiling:** The first step of *ImGrid* is profiling the dataset to collect the average bounding box area per image and the average model confidence score per class. If ground truth data is not available, the user can approximate these values by running the profiling set through their object detection model and using the predicted boxes instead.

To collect the average bounding box area per image, we average the ground truth bounding box areas from the profiling set. Any image area covered by multiple bounding boxes must be double counted (once for each bounding box covering the area).

Second, we run each image in the profiling set at each grid size that's feasible given the user's budget. The predicted bounding boxes during profiling must be grouped across three axes (1) their class label, (2) the grid size that their image used, and (3) the highest grid size in which they remain detectable. Then, each group's average model confidence score must be recorded. Figure 3.1 contains an example of these confidence scores for the "motorcycle" class in COCO [11].

Finally, we select a set of grid sizes from highest to lowest. During inference, all images will be sent to the highest grid size, and each image that remains under the confidence score or bounding box area threshold will be retried at higher grid sizes until the image is accepted, or the highest grid size has processed it. The particular grid sizes to use is a hyperparameter and can be chosen through trial-and-error.

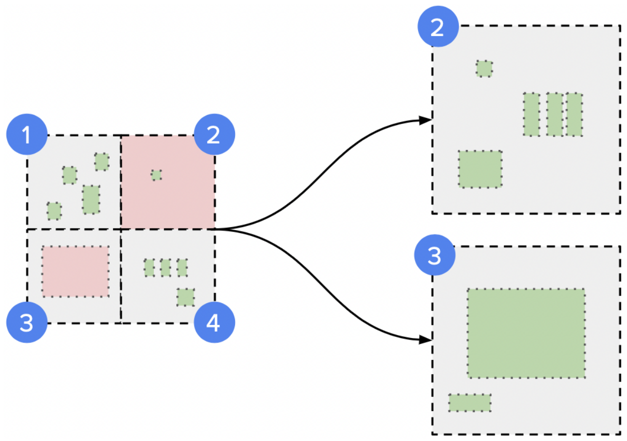


Figure 3.2: Diagram of the *ImGrid* method. In this example, all images are initially run using a 2x2 grid and then rerun individually as 1x1 grids. Images 1 and 4 contain a handful of valid detections (denoted as green rectangles) that cover a sufficiently high bounding box area, so they are not retried. However, image 2 contains one detection that covers a tiny area of the image. This is lower than the bounding box area threshold, so it's retried as a 1x1. Image 3 contains a detection with a lower-than-required confidence score (denoted as a red box), so it too is retried as a 1x1. The 1x1 results for images 2 and 3 are kept, and their 2x2 results are discarded.

***ImGrid*– Inference:** For inference, all images are first gridded at the highest grid size. Next, two conditions are checked for each image. First, if the bounding box area in the image (scaled up to the corresponding area in a 1x1) is smaller than the average bounding box area recorded in profiling, the image is retried at the next highest grid setting and the current results are discarded.

Second, if any bounding box's confidence score is lower than the profiling set's average confidence score for bounding boxes corresponding to (1) the same label, (2) the same grid size as the image, and (3) a max grid size equal to the image's grid size, then the image is retried, and its previous results are discarded.

Otherwise, the image's results are kept. This process repeats as images are retried at lower and lower grid settings, until they're either processed or tried at the lowest grid setting, at which point the results are kept regardless of bounding box area or confidence. An example of this process is depicted in Figure 3.2.

By thresholding on the average bounding box size, *ImGrid* forces images with potentially missed detections to be retried at a larger grid size. Similarly, by thresholding on the confidence scores, *ImGrid* forces images with potentially misclassified detections to also be retried. Note that if using the mean as the threshold is too conservative or liberal for a particular task and inference budget, the thresholds can be adjusted by a constant factor or be replaced with a threshold some number of standard deviations above or below the mean.

3.3 Other approaches

Throughout our work, we pursued two high-level approaches to boost image gridding accuracy: grid-level augmentations and content-aware grid sizing. We limited our studies to the domain of object detection since it's a popular use case and since it can support other gridding use cases, such as classification. We studied two grid-level augmentations: black borders, where black stripes are inserted between neighboring images, and blurred borders, where borders between images are blurred with a Gaussian filter. Both augmentations seemed to slightly lower accuracy on grids.

The second approach– content-aware grid sizing– was motivated by an empirical observation: some objects within images tend to remain detectable even when scaled down a lot while the rest tend to become undetectable or misclassified even when scaled down moderately. In other words, for object detection, some images are "easier" and others are "harder" since they contain easier-to-detect (or harder-to-detect) objects. Our motivating hypothesis was that easier images should be gridded at higher grid sizes to maximize cost savings, while harder images should be gridded at lower grid sizes to maximize accuracy. Choosing the most suitable grid size for a particular image requires leveraging info about the image's content to determine whether it's an easy or hard image.

A natural question would be if it's possible to learn what size an image could be gridded into by using some feature representation. To test if this was possible, we trained a classifier to predict the optimal grid size for a given image, meaning the smallest resolution at which objects in the image can still be correctly predicted. We annotated each image in our profiling set with its optimal grid size and trained the classifier on each image's CLIP embedding. Since our classifier was unable to achieve accuracy that surpassed chance, we concluded that more work needed to go towards improving the model architecture and image featurization, which are promising areas for future work.

Chapter 4

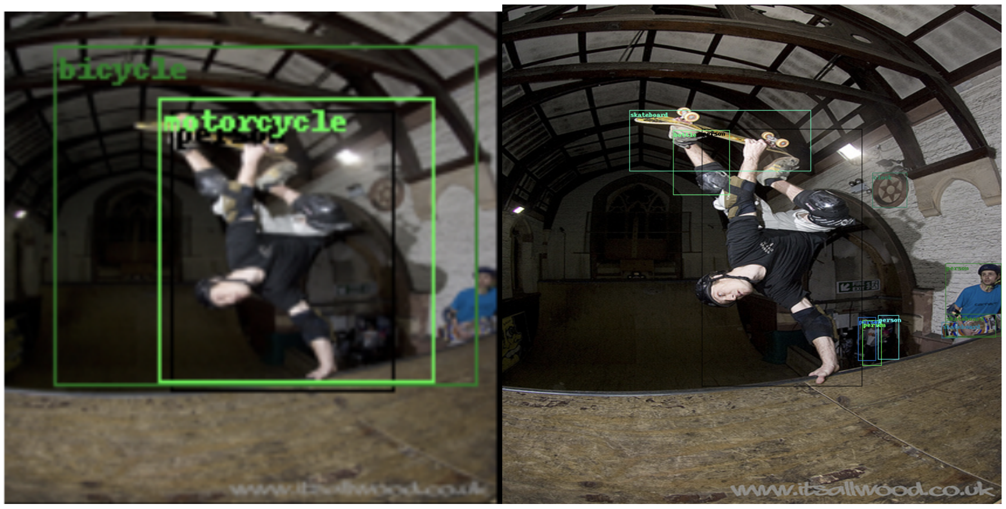
Experiments

We benchmark *ImGrid*'s cost savings and accuracy on object detection tasks across different open source and commercial models on the Pascal VOC and COCO datasets. Our goals are to: (1) explain how *ImGrid* can save cost while maintaining high accuracy, (2) evaluate *ImGrid*'s cost savings against sending individual images, (3) explore the image gridding cost-accuracy tradeoff, and (4) benchmark some grid-level augmentations.

Setup: We ran our experiments across three models and two datasets. The models were the Detection Transformer (DETR) [3], Faster Region-based Convolutional Neural Network (Faster-RCNN) [14] with ResNet 50, and AWS Rekognition [1]. We used open source implementations of Faster-RCNN [13] and DETR [6] that were trained on COCO. AWS Rekognition [1] offers a closed source vision model that charges per inference. We use two datasets: Pascal VOC 2012 [7] and COCO 2017. Unless otherwise stated, for Pascal the average bounding box threshold was collected by averaging over all the bounding box areas for all the images in the validation set, and the average confidence thresholds were collected by assessing 800 images from the validation set per class. Results were collected by running image gridding on the training set. Similarly, unless otherwise stated, for COCO the bounding box threshold was collected by averaging across all training images. The confidence thresholds were collected by assessing up to 800 images from the training set. Results were collected over the entire validation set. When performing benchmarks over Pascal, we manually remapped the output labels from the open source models to Pascal labels, instead of retraining the models altogether. This better emulates the setup for many commercial MLaaS models that cannot be retrained cheaply.

COCO Case Study: we start by investigating how *ImGrid* processes a sample COCO image on an object detection task using an RCNN. For this case study, we profile COCO using 100 images per class, and we run *ImGrid* using a two-layered cascade: first, all images are run through a 5x5 grid, and then *ImGrid* retries one at a time any images that don't meet the confidence score or bounding box area thresholds.

Figure 4.1 compares a single image downscaled to fit inside a 5x5 grid against the original image



(a) Image in 5x5 grid (b) Image in 1x1 grid

Figure 4.1: This is an image from the COCO dataset. The left hand side is an image that has been downscaled to fit inside a 5x5 grid, and the image on the right is the same image scaled to fit inside a 1x1 grid. The image in a 5x5 contains misclassifications (e.g. “motorcycle”) that are corrected in the 1x1.

Label	Confidence	Threshold
person	0.8302	0.415
bicycle	0.6745	0.447
motorcycle	**0.3998**	**0.402**
bbox area	29,909	14,580

Table 4.1: Confidence and bounding box area thresholds from profiling COCO. The middle column contains the confidence scores (and total bounding box area) of the predicted bounding boxes when the image from Figure 4.1 is placed in a 5x5. Since the motorcycle bounding box’s confidence score is lower than the threshold, the image is retried.

scaled to run individually. The 5x5 version places a misclassified bounding box (“motorcycle”) on the person. This bounding box disappears in the 1x1 grid. *ImGrid* successfully retries the image in a 1x1 due to the thresholds set during profiling. As shown in Table 4.1, the motorcycle bounding box in the 5x5 has a lower than required confidence score, signifying that the model is unconfident about the prediction. Indeed when *ImGrid* retries the image, this box is removed altogether.

Cost tradeoffs: We now show the cost tradeoffs for *ImGrid* by benchmarking different configurations against one-at-a-time performance. Table 4.2 compares accuracy and cost metrics for

ML Model	Grid Strategy	Inferences per Image	% Inf. Saved	mAP	Δ mAP
Faster R-CNN					
	One-at-a-time (1x1)	1	0%	0.706	0
	$ImGrid_{\{(5,5),(1,1)\}}$	0.881	11.9%	0.684	−0.022
	$ImGrid_{\{(5,5),(2,1)\}}$	0.461	53.9%	0.659	−0.047
DETR					
	One-at-a-time (1x1)	1	0%	0.753	0
	$ImGrid_{\{(5,5),(1,1)\}}$	0.862	13.8%	0.732	−0.019
	$ImGrid_{\{(5,5),(2,1)\}}$	0.451	54.9%	0.676	−0.077
AWS Rekognition					
	One-at-a-time (1x1)	1	0%	0.782	0
	$ImGrid_{\{(5,5),(1,1)\}}$	0.830	17.0%	0.760	−0.022
	$ImGrid_{\{(5,5),(2,1)\}}$	0.434	56.6%	0.735	−0.047

Table 4.2: ML model and gridding strategy performance on the Pascal dataset[7]. The ΔmAP figures are calculated using the 1x1 baseline.

different *ImGrid* configurations on the Pascal dataset. Generally speaking, cost and accuracy are inversely related, with more cost savings resulting in a higher accuracy drop. However, large cost savings can still be achieved with relatively little drop in accuracy. For instance, by cascading a 5x5 grid to a 2x1 grid, AWS Rekognition can process the Pascal validation set in less than half as many inferences as one-at-a-time, incurring a relatively minor mAP cost of 0.047.

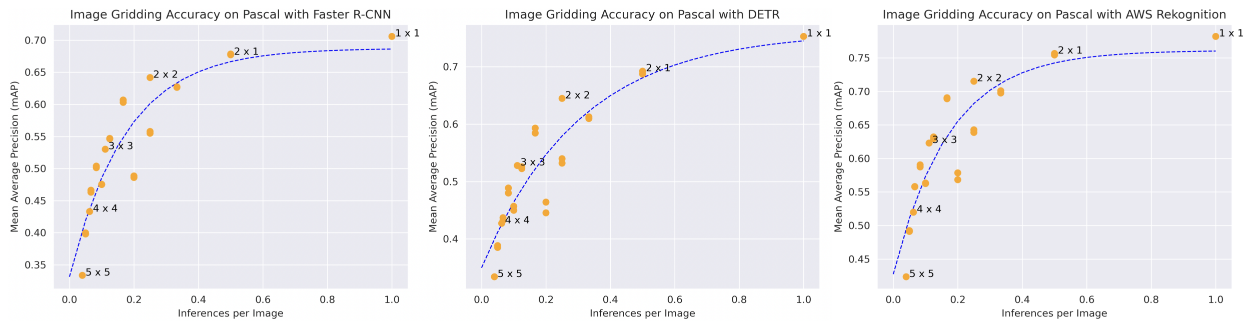


Figure 4.2: These are image gridding mAP scores on Pascal using different models. Each dot represents a rectangular grid configuration that the dataset was profiled with. The square grid configurations, as well as the 2x1 configuration, are labeled. Each scatterplot has a logistic trendline overlayed.

Image gridding: We also show results for gridding images without retrying. The scatter plots in Figure 4.2 and Figure 4.3 shows the cost-accuracy tradeoff across different grid settings. Simple image gridding can also provide large cost savings with moderate accuracy loss. However, as the

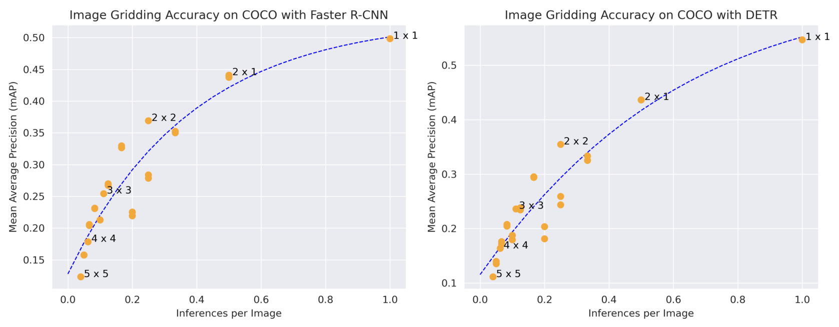


Figure 4.3: These are image gridding mAP scores on COCO using different models.

trend shows, accuracy falls extremely quickly for high grid sizes, such as 5x5. Intuitively, this makes sense since the downscaling imposed by high grid sizes results in missed or misclassified objects. This does pose a lost opportunity, though, since some images can actually benefit from high grid sizes. Capturing this optimization requires a mechanism to differentiate between harder images that should be gridded at low sizes and easier images that should use high sizes. *ImGrid*'s retry mechanism allows it to save results for easy images at high grid sizes while retrying harder images at low grid sizes, attaining the best of both worlds.

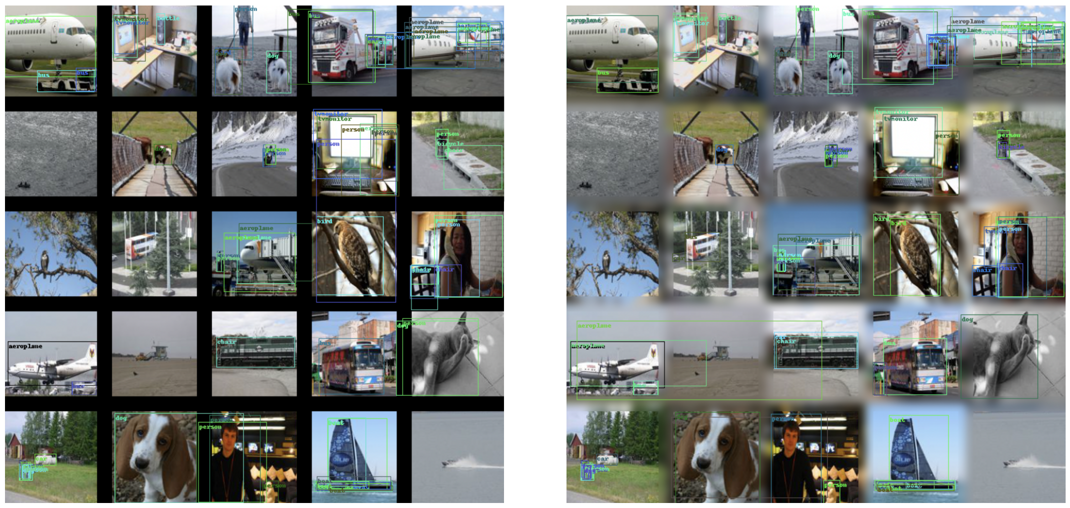


Figure 4.4: Sample grids from the Pascal dataset with black borders and blurred borders augmentations. 3.75% of each image is blacked out/blurred out at each side.

Augmentations: This section contains results from two grid-level augmentations that we assessed: black borders and blurred borders. See Figure 4.4 for example grids with these augmentations. The

ML Model	Grid Size	Baseline mAP	Augmented mAP	Δ mAP
Faster R-CNN				
	(1x1)	0.706	0.706	0
	(2x2)	0.642	0.617	−0.025
	(3x3)	0.530	0.513	−0.017
	(4x4)	0.433	0.414	−0.019
	(5x5)	0.334	0.320	−0.014
DETR				
	(1x1)	0.753	0.753	0
	(2x2)	0.645	0.617	−0.028
	(3x3)	0.528	0.497	−0.031
	(4x4)	0.427	0.393	−0.034
	(5x5)	0.334	0.307	−0.03

Table 4.3: mAP scores over Pascal for grids with the black borders augmentation. Each image had 3.75% of each side blacked out.

ML Model	Grid Size	Baseline mAP	Augmented mAP	Δ mAP
Faster R-CNN				
	(1x1)	0.706	0.706	0
	(2x2)	0.642	0.614	−0.028
	(3x3)	0.530	0.503	−0.027
	(4x4)	0.433	0.404	−0.029
	(5x5)	0.334	0.314	−0.03
DETR				
	(1x1)	0.753	0.753	0
	(2x2)	0.645	0.608	−0.037
	(3x3)	0.528	0.500	−0.028
	(4x4)	0.427	0.402	−0.025
	(5x5)	0.334	0.316	−0.018

Table 4.4: Blurred borders augmentation. 3.75% of each side in each image was blurred.

black borders augmentation inserts black stripes between consecutive images. We hypothesized that this could signal to the model that each image in the grid is unrelated. However, as shown by Table 4.3, this led to slightly worse results.

We performed the blurred borders augmentation by smoothing the seams between consecutive images using a symmetric Gaussian filter with a standard deviation of 7. We hypothesized that this would remove sharp changes in pattern between consecutive images, making the grid seem more like a natural image that better matched the model's training set. However, as shown in 4.4, this also generated slightly worse results than the baseline.

Chapter 5

Related Works

Optimizing MLaaS usage: Recent work for optimizing MLaaS usage has focused primarily on cascading models across different MLaaS providers or on aggregating results from different providers. For instance, FrugalML [4] optimizes MLaaS usage by providing a strategy to pick which MLaaS provider to send a particular request to. It leverages a multi-step process where a request is first sent to one model and then, based on a profiling step, sent to follow-on providers if necessary. Similarly, Armol [17] uses RL to aggregate results across different providers while still minimizing cost. This work, however, optimizes computer vision usage even with a single MLaaS provider.

Modifying the model: Some recent work, as well as traditional techniques, has focused on reducing ML inference cost by modifying the model itself. For instance, Nvidia's A-ViT model architecture adaptively modifies inference cost based on an image's complexity, reducing cost for easier images while accepting higher cost for difficult images. *ImGrid* also reduces inference cost based on image complexity; however, it doesn't require changing the model itself making it better suited for applications that rely on MLaaS. As another example, neural group testing [10] also recognizes the potential to save inference cost by combining multiple images for content moderation. However, it achieves this by proposing a model architecture that combines multiple images during the inference, rather than taking in an image grid. Unlike *ImGrid*, this requires a new model architecture altogether. In the same vein, traditional techniques like quantization also require modifying the model itself to reduce inference cost, making them incompatible with MLaaS APIs.

Combining images: Combining images to improve ML has been a common tactic used throughout the ML development process. For instance, one common data augmentation during training is to combine training images [15] when passing them into the model. This can be achieved by gridding training images together or by interpolating between them. Image gridding and *ImGrid*, on the other hand, combine images spatially during inference rather than training in order to drive down inference cost. One common practice in inference is stacking images into a microbatch and running a model on the batch to leverage computational efficiencies to reduce inference cost.

Image gridding is entirely compatible with the pattern. Image grids can be batched together allowing any efficiency savings across the two techniques to be multiplied.

Hardware inference speedup: To expedite model inference, some organizations have focused on building custom hardware. Some examples include Google Cloud's TPUs [9] or Tensorflow Lite microcontrollers [12]. These help reduce inference latency, but by the nature of hardware, they're more challenging and expensive to deploy and upgrade quickly. *ImGrid* can be performed immediately on existing models, and it's hardware agnostic, so once these hardware innovations are deployed, their gains can be amplified by *ImGrid*.

Chapter 6

Conclusion

In this work, we introduce image gridding and *ImGrid*, which lets MLaaS users pack multiple images into a single inference for tasks such as object detection. This helps improve model efficiency and reduces the cost of commercial vision models offered as a service. Empirical results demonstrate that *ImGrid* can achieve significant cost savings while keeping accuracy loss low. Interesting areas of future work include extending *ImGrid* to more computer vision tasks, studying more image features that could better determine what grid sizes they should use, and researching potential grid augmentations that could improve the accuracy of grids overall.

Bibliography

[1] *AWS Rekognition*. `https://aws.amazon.com/rekognition/`. Accessed: 2023-05-10.

[2] *Azure Cognitive Service for Vision*. `https://azure.microsoft.com/en-us/products/cognitive-services/vision-services`. Accessed: 2023-05-10.

[3] Nicolas Carion et al. "End-to-End Object Detection with Transformers". In: *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I*. Glasgow, United Kingdom: Springer-Verlag, 2020, pp. 213–229. ISBN: 978-3-030-58451-1. DOI: `10.1007/978-3-030-58452-8_13`. URL: `https://doi.org/10.1007/978-3-030-58452-8_13`.

[4] Lingjiao Chen, Matei Zaharia, and James Y Zou. "FrugalML: How to use ML Prediction APIs more accurately and cheaply". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., 2020, pp. 10685–10696. URL: `https://proceedings.neurips.cc/paper_files/paper/2020/file/789ba2ae4d335e8a2ad283a3f7effced-Paper.pdf`.

[5] Daniel Crankshaw et al. "Clipper: A Low-Latency Online Prediction Serving System". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, Mar. 2017, pp. 613–627. ISBN: 978-1-931971-37-9. URL: `https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/crankshaw`.

[6] *DETR HuggingFace Documentation*. `https://huggingface.co/docs/transformers/model_doc/detr`. Accessed: 2023-05-10.

[7] M. Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (June 2010), pp. 303–338.

[8] *Google Cloud Vision API*. `https://cloud.google.com/vision`. Accessed: 2023-05-10.

[9] Norman P. Jouppi et al. *TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings*. 2023. arXiv: `2304.01433 [cs.AR]`.

[10] Weixin Liang and James Zou. *Neural Group Testing to Accelerate Deep Learning*. 2021. arXiv: `2011.10704 [cs.LG]`.

[11] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: `1405.0312`. URL: `http://arxiv.org/abs/1405.0312`.

[12] Erez Manor and Shlomo Greenberg. "Custom Hardware Inference Accelerator for TensorFlow Lite for Microcontrollers". In: *IEEE Access* 10 (2022), pp. 73484–73493. DOI: `10.1109/ACCESS.2022.3189776`.

[13] *PyTorch Faster R-CNN models*. `https://pytorch.org/vision/0.13/models.html`. Accessed: 2023-05-10.

[14] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *CoRR* abs/1506.01497 (2015). arXiv: `1506.01497`. URL: `http://arxiv.org/abs/1506.01497`.

[15] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (2019), pp. 1–48.

[16] Samuel Williams, Andrew Waterman, and David Patterson. "Roofline: An Insightful Visual Performance Model for Multicore Architectures". In: *Commun. ACM* 52.4 (Apr. 2009), pp. 65–76. ISSN: 0001-0782. DOI: `10.1145/1498765.1498785`. URL: `https://doi.org/10.1145/1498765.1498785`.

[17] Shuzhao Xie et al. *Cost Effective MLaaS Federation: A Combinatorial Reinforcement Learning Approach*. 2022. arXiv: `2204.13971 [cs.LG]`.