Collage Inference: Tolerating Stragglers in Distributed Neural Network Inference using Coding

Krishna Giri Narra
EE Department, University Of Southern California
Los Angeles, CA, USA

Zhifeng Lin
EE Department, University Of Southern California
Los Angeles, CA, USA

Ganesh Ananthanarayanan
Microsoft Research USA

Salman Avestimehr
EE Department, University Of Southern California
Los Angeles, CA, USA

Murali Annavaram
EE Department, University Of Southern California
Los Angeles, CA, USA

Abstract
MLaaS (ML-as-a-Service) offerings by cloud computing platforms are becoming increasingly popular these days. Pre-trained machine learning models are deployed on the cloud to support prediction based applications and services. For achieving higher throughput, incoming requests are served by running multiple replicas of the model on different machines concurrently. Incidence of straggler nodes in distributed inference is a significant concern since it can increase inference latency, violate SLOs of the service. In this paper, we propose a novel coded inference model to deal with stragglers in distributed image classification. We propose enhanced single shot object detection models, Collage-CNN models, to provide necessary resilience efficiently. A Collage-CNN model takes collage images formed by combining multiple images as its input and performs multi-image classification in one shot. We generate custom training collages using images from standard image classification datasets and train the model to achieve high classification accuracy. Deploying the Collage-CNN models in the cloud, we demonstrate that the 99th percentile latency can be reduced by 1.45X to 2.46X compared to replication based approaches and without compromising prediction accuracy.

CCS Concepts • Computer systems organization → Availability; Redundancy; Cloud computing; • Computing methodologies → Object recognition; Neural networks.

1 Introduction
Artificial Intelligence in its current form of deep learning is an enabler of advances across many fields ranging from autonomous driving, health care, wireless communications, data center management, machine translation and so on. The availability of data sets, accessible learning frameworks for rapid prototyping, and relatively inexpensive cloud infrastructure is accelerating the progress of deep learning related technologies. As a result, major cloud computing platforms are beginning to offer Machine Learning as a Service (MLaaS) [1, 4, 5]. MLaaS offerings include support for data pre-processing, model training, and model inference.

Supervised deep learning occurs in two phases: Training phase and Inference (also called Prediction Serving) phase. Training phase involves data collection and preprocessing, selection of model architecture, and training the parameters of the architecture to minimize a loss function. Model training is done through forward and backward propagation operations. The output from this process is a trained model(s) which stores the architecture and the values of parameters of the architecture. During inference, the trained model is deployed to make predictions on different inputs. Inference phase only performs forward propagation operation on the models. Given the immense computational demands of training many research efforts have tackled distributed training in the cloud [12, 25, 35, 36, 43]. Inference, on the other hand, poses a different set of constraints while deploying as a service.

Model deployment in cloud for inference is generally concerned with quality of service (QoS) guarantees provided to the user. From the user perspective one critical QoS metric is the query latency. From a service provider's perspective
In this paper we try to answer the following question: *is it possible to provide low variance inference latency with a tunable tradeoff in accuracy loss and computational cost?* To tackle this challenge we propose a technique called Collage Inference. As shown in figure 1 Collage Inference uses a unique convolutional neural network (CNN) based redundancy model that can perform multiple predictions in one shot during image processing. This redundant model is run concurrently as a single backup service for a collection of individual inference models. The redundant model takes as input an image collage formed from multiple images and predicts the object classes in all the included images. In the scenario when any of the nodes running single image inference models becomes a straggler, the prediction from the collage inference model can be used in the place of the prediction from straggler node(s). One can view Collage Inference as a coded inference model where the encoding is the collection of images that are spatially arranged into a collage, and the decoder simply selects the inference result of an image that is assigned to a straggler node. In this paper, we describe and demonstrate the effectiveness of this redundancy model. We present various ways to design the inputs to the redundancy model so that it incurs low computation overhead and does not compromise accuracy. We explore some of the design space and describe the trade-offs in terms of amount of redundancy provided by the model and model accuracy.

The main contributions of this paper are as follows:

- We propose a novel idea of creating collage images to do simultaneous multi-image classification and mitigate straggler effects in distributed image classification systems.
- We describe the architecture changes to the CNN models to perform Collage inference. We provide techniques to generate very large training datasets for training the Collage CNN models.
- We evaluate the Collage CNN models by deploying them in the cloud and show their effectiveness in mitigating stragglers without compromising prediction accuracy. We demonstrate that Collage CNN models can reduce 99-th percentile latency by 1.47X compared to replication based approach.

Rest of the paper is organized as follows. In section 2 we provide background on image classification and object detection, in section 3 we describe the collage inference techniques, in section 4 we describe architecture of models and our implementation, section 5 provides experimental evaluations and design space exploration, section 6 discusses related works and in section 7 we make our conclusions.

## 2 Background and Motivation

In this work, we transform the problem of straggler mitigation during image classification tasks into a multi-object detection problem. As such we briefly describe image classification and object detection. We then present an overview of Convolutional Neural Networks (CNNs), that are the current state of art in machine learning for image classification and object detection. After discussing CNNs we present few motivational experiments.

### 2.1 Background

**Image Classification**: Image classification is a fundamental task in computer vision. In image classification, the goal is to predict the main object present in a given input image. There are a variety of algorithms, large datasets, and challenges for this task. A widely known challenge is the Imagenet Large
Scale Visual Recognition Challenge (ILSVRC). It’s training dataset consists of 1.2 million images that are distributed across 1000 object categories. Since 2012 [30], the improvements in accuracy of image classification tasks have come from using Convolutional Neural Networks (CNNs). Some of the popular CNN architectures are: ResNet [26], Wide ResNet [54], Inception [51], MobileNet [27], VGGNet [50].

Object Detection: Given an input image, the object detection task involves predicting the classes of all objects present in the image along with the locations of objects in the image. The location information is predicted as a rectangular bounding box within the image. Image Classification can be considered as a sub-task of object detection. There are two different methods to perform object detection using CNNs:

- Separating Region proposal and Classification: Region proposal methods are used to generate potential bounding boxes and an CNN based image classifier is used to extract features and classify the objects in each proposal. These methods can take multiple iterations, one for each region proposal, for each object detection. Examples of these models include: R-CNN [23], Fast R-CNN [22], Faster R-CNN [47], R-FCN [14].
- Unified or Single shot object detection: Single CNN is used to generate both the bounding boxes and the object class information. There is no separate region proposal network. Examples include: YOLO [44, 45], SSD [40], DSOD [49], DSSD [20].

During inference, the Single shot object detectors have lower latency while maintaining similar accuracy as that of the region proposal based detectors. Of course one may pose the natural question: why not use the multi-object detection models for single image classification task too? While using a multi-object inference model seems attractive, multi-object inference models suffer an accuracy loss of 4-5% (figure 6), which is a significant reduction that may not be acceptable to the end user. For a perspective, to achieve this 5% increase in accuracy when using ResNet models the number of layers needs to be increased by more than 3X. Hence, multi-object detection and image classification are generally considered distinct tasks that require different approaches.

Convolutional Neural Networks: Convolutional Neural Networks (CNNs) are a class of neural network architectures used for processing visual data i.e., images [30, 31]. A CNN is generally composed of 3 types of layers: convolution layers, pooling layers, fully connected layers and non-linear activation functions: Rectified Linear Unit (ReLU), Sigmoid etc. An input to the CNN is processed using a sequence of many layers to produce corresponding output. Each layer along with its activation function performs either a linear or non-linear transformation of its input. Convolution layer consists of multiple convolutional filters. During computation, each filter is slid across the input image to generate a corresponding activation map (or feature map) as an output. Each convolution filter learns to detect a specific pattern (or feature) in its input. After training it has been observed that, the convolution filters in the initial layers detect simpler features like edge, color etc. whereas convolution filters in the deeper layers detect complex features like nose, eye etc. Pooling layers are used for down-sampling the size of intermediate activation maps and are placed between sequences of convolution layers. Fully connected layers are located towards the end of the CNN and they generate values that are used to classify the object in the image.
2.2 Motivation

We wanted to measure the effect of stragglers during distributed image classification. For this purpose, we designed an image classification server that uses a ResNet-34 CNN model to provide inference and created 50 instances of this server on Digital Ocean cloud [2]. Each instance is running on a compute node that consists of 2 CPUs and 4GB memory. We created a client to generate requests to all these servers. We measured inference latency across these 50 nodes while performing single image classification and the probability density function of the latency is shown in figure 2. The average single image inference latency was 0.15 seconds whereas the 99-th percentile latency was 0.70 seconds. The 99-th percentile latency is significantly (4.67X) higher than the mean latency. We repeated this experiment many times and found that similar behavior occurs most of the time. These experiments led us to conclude that stragglers can significantly deteriorate the QoS in online cloud based image classification systems.

3 Collage Inference Technique

In this section we describe the challenges in straggler mitigation using existing techniques, then we describe our Collage Inference technique.

3.1 Existing Techniques and Challenges

MLaaS is currently being deployed for online inference, such as Amazon AWS SageMaker [1] or Google Cloud ML Engine [4] or Microsoft Azure ML Service [5]. Users can use pretrained models or can develop and deploy their own trained models. These systems provide some basic QoS guarantees such as 99th-percentile inference latency, and throughput for a given compute resource (including CPU, memory, and storage) allocation. In a production environment the service provider (for pre-trained models) or the users (for custom trained models) may instantiate a set of N compute nodes, each running a replica of the trained image classification CNN such as Resnet. We will refer to a single image classifier model as S-CNN. The input to S-CNN is a single image of resolution Width(W) x Height(H) and C-color channels. The output from the model is the class label of the main object in the image. Assuming a deployment scenario where the number of concurrent classification requests exceeds N (the number of replicas), a front-end load balancer in the cloud may direct each image to one of the N replicas for inference. If a machine running one of the N replicas does not provide the result within a stipulated time window then the customer will perceive a reduced QoS. For an end user facing application, not meeting QoS guarantees will adversely affect the interactiveness of the application [10, 15].

One option to improve QoS in the presence of a straggler is to add redundancy in the form of over-provisioning of compute nodes. Consider a system of 10 nodes over-provisioned by 1 node. This node would be running another replica of S-CNN. One challenge is it is difficult to know ahead of time which one of the N nodes will be a straggler. As a result, deciding which one of the input requests to replicate becomes difficult. Another strategy is to duplicate the inference request Image_i sent to Node i only when the node is detected as a straggler. This is a reactive approach that requires waiting for a straggler appearance before launching a redundant request. For instance, a request may be launched speculatively after waiting for an expected latency, similar to what is adopted in Hadoop MapReduce frameworks [6, 16]. There are practical challenges in implementing the reactive approach. First, from our measurements, shown in figure 2, the inference latency could be in 10’s to 100 milliseconds. As a
result, speculative relaunch techniques must be fast enough to adopt. Second, the image must be re-distributed to a new machine for replicated execution. As a result reactive approach may increase the service latency depending on how long the reactive approach waits for a response before speculating a job. To avoid the challenges, the system can be over provisioned by factor of 2. That is for every one of \( N \) nodes there will be a backup node and every input request will be duplicated. However, this approach increase the resource costs significantly by 2X.

Another recent technique from coded computing, referred to as learning a code [29], uses trained encoding and decoding neural networks to provide redundancy. Briefly the technique is as follows:

- In a system of \( N = 5 \) compute nodes, provide redundancy of 1 node \( O \). Each of the \( N \) nodes executes a replica of S-CNN.
- The model in node \( O \) takes as input all the 5 input images. These images are passed through a convolutional encoder network, composed of convolutional layers, and the outputs are then passed onto the S-CNN model.
- The outputs from the \( N + 1 \) models are fed to a decoder network, composed of fully-connected layers. The outputs from any straggler node is represented as a vector of zeros. The final output from the decoder network is trained to generate the missing prediction.
- Both the encoder and decoder networks are learned through back-propagation. The training data consists of images and also the expected predictions under different straggler scenarios.

The technique when evaluated on CIFAR-10 dataset, consisting of 60000 images equally distributed among 10 different classes, showed a recovery accuracy of 80.74% for \( N = 2 \) nodes, but the recovery accuracy drops to 64.31% for \( N = 5 \) nodes, when any one of the \( N \) nodes is a straggler. CIFAR-10 dataset is considered a small dataset and such large accuracy losses even for small datasets may not be acceptable.

### 3.2 Collage Inference

A key insight behind Collage inference is that the spatial information within an input image is critical for CNNs achieving high accuracy and it should be maintained. This places restrictions on how to combine multiple images into one. Another key observation is that image classification is a narrowly scoped object detection task where only a single object within a well defined boundary must be detected. Based on these insights we designed Collage inference to be a hybrid multi-object detection and image classification scheme where the multi-object detection model serves the purpose of a low cost backup classifier to deal with stragglers. The multi-object classifier is a trained CNN model, which we refer to as Collage-CNN. It takes a collage composed from all the images \([Image_1, ..., Image_i, ..., Image_N]\), where each image is given as an input to one of the \( N \) single image classifiers. The Collage-CNN provides the predictions for all the objects in the collage along with bounding box locations of each image in the collage. By smartly composing the collages and using the location information from the Collage-CNN, the serving system can replace the missing predictions from any straggler node(s).

The input and output requirements of the Collage-CNN lend themselves to the object detection algorithms in computer vision. As discussed in section 2, object detection algorithms take as input an image and provide as output the locations (bounding boxes) of all the objects in the image along with the class label of the object in the location. Some object detection algorithms use multiple iterations on the CNN extracted features to predict objects and their locations. Some algorithms perform single pass (single-shot) on a CNN to both extract features, predict objects and their locations. Since the goal of our work is to mitigate stragglers using a single Collage-CNN model, it is imperative that the Collage-CNN which acts as a redundant classification model to be as fast as the single image classification task latency. We explored using both iterative and single-shot CNNs as a Collage-CNN and determined that single-shot CNNs are better suited due to their fast response time, comparable to single image classification models [56]. We then explored using YOLO [44] algorithm based object detection models as base for Collage-CNNs. There are many YOLO models varying in their architectures and presenting different latency and accuracy tradeoffs. Models with higher depth have higher accuracy in predictions but also significantly more latency and computational resource cost. The primary design constraint for doing Collage Inference is that the Collage-CNN model architecture should have latency smaller than or equal to the S-CNN model. Based on several empirical experiments we decided to use YOLOv3-Tiny model [45] where the inference latency is smaller than or similar to the S-CNN models. We present latency measurements in section 5.

Some more key design decisions in the Collage-CNN are that of organization of images in the input collage and the individual image resolution within the collage. Most of the deep learning image classifiers tend to use square shaped images as input. S-CNN models trained on Imagenet dataset have input images with 224x224 resolution. To maintain reasonable accuracy, the resolution of input images to the Collage-CNN model has to be larger than a single input image resolution. This is because collage images are a combination of many single images. Lowering resolution of an image can also lower the accuracy of the model. However, increasing the resolution of input will increase the computation requirements for Collage-CNN. So, a fine balance needs to be made when choosing input resolution. Our Collage-CNN model, for Imagenet dataset, takes input collages of fixed size 416x416 pixel resolution. As such the size of any
collage created from multiple images cannot exceed this overall input image size. Thus to fit every \( N \) images into a collage in our Collage-CNN model we first create a square grid consisting of \( \lceil \sqrt{N} \rceil \) images. The number of images in the grid \( N \) determines the number of concurrent images that are classified by Collage-CNN alongside the \( N \) S-CNN models. The larger the value of \( N \) the smaller is the overhead for running the Collage-CNN inference. However, as the size of \( N \) grows more images must be packed into a collage which then reduces the resolution of each single image. This lowering of resolution also results in lower accuracy of predictions on each image. In evaluation section, we explore this trade off between lowering single image resolution, and corresponding Collage-CNN detection accuracy.

Figure 3 summarizes a Collage Inference system for \( N = 10 \) nodes with one of the nodes providing redundancy for the remaining 9 nodes. Each of the 9 nodes running S-CNN model takes a normal image as input. The 10th node takes the collage image as input. Each of the 9 input images is lowered in resolution and inserted into a specific location to form the collage image. Input image to node \( i \) goes into location \( i \) in the collage image. This placement order is useful later on to map different classes predicted by the Collage-CNN to corresponding input images. This collage image is provided as input to node 10. The predictions from the Collage-CNN are further processed using the algorithm described in section 4.4. The output predictions from all the 10 nodes are sent to the decode process in the system. This decode process uses the predictions from the redundant node to fill in any missing predictions from the 9 nodes and return the final predictions to the user.

Figure 4 contains few examples of collage images created from Imagenet and the predictions of Collage-CNNs on these images.

Next we discuss the architecture of Collage-CNN models, generation of collage images for training and validation, the Collage-CNN output decoding algorithm and the final decoder algorithm.

4 Collage-CNN Architecture and Implementation

4.1 Collage-CNN Architecture

A Collage-CNN model takes an image file as input. The outputs from the model are the list of objects detected in the image and their bounding boxes within the image. Bounding box for an object is the predicted location of the object in a rectangular form as shown in figures 4d, 4e, 4f. The model outputs the co-ordinates of the four vertices of each box. Each bounding box corresponds to one of the detected objects. If the model detects \( k \) objects in the image, it also predicts corresponding \( k \) bounding boxes. Using these bounding boxes, we can reverse map the object predictions to individual image classification result. As shown in figure 3 each image that is assigned to a S-CNN model running on node \( i \) is placed in a predefined square box within the collage. Specifically, in the collage each node \( i \) is assigned a box location \( l = L(i) \). We process the model outputs and extract the best possible bounding box information \( l \) to get \( L^{-1}(l) \) which would in turn correspond to the node \( i \).

We use YOLOv3-tiny model [45] as the base for building the Collage-CNN model for both datasets. YOLOv3-tiny model architecture consists of 10 convolution layers along with 6 Max pooling layers for extracting features from the input image. These features are then processed using a combination of 3 convolution layers and 1 upsampling layer to predict the locations and classes of the objects in the image. Considering the requirements of collage inference we modify the YOLOv3-tiny model architecture appropriately for each dataset to build the corresponding Collage-CNN. These changes are:

- First architecture change is the number of convolution filters in the Convolution layers 10 and 13. The number of filters is dependent on the number of classes present in the dataset [3]. For 100 classes of Imagenet-1k dataset the number of filters is 315 and for the CIFAR-10 dataset with 10 classes it is 45.

- Second architecture change is the resolution of input image to the model. This is modified depending on the dataset and the architecture of the collage. For CIFAR-10 dataset: a collage of 9 images would have a resolution of 96 x 96, a collage of 16 images would have a resolution of 128x128 etc. For collages from Imagenet-1k dataset the resolution is always fixed at 416 x 416.

- Third parameter change is the detection threshold of the model. We use a threshold of 0.15 during inference. Any object detected by the model in the image for which the confidence value is more than this threshold value is considered as a potential prediction.

- Fourth parameter change is customizing the anchor boxes provided to the model during training. Model predicts the final bounding boxes by modifying the dimensions of the anchor boxes provided to it. In all collage images used in training the bounding boxes are same and known a priori from building the collage. Providing these bounding boxes as the anchor boxes increases the test accuracy of the model. More on this in section 5.

4.2 S-CNN Architecture

We used a pre-trained ResNet-34 model as the single image S-CNN model for Imagenet-1k dataset. Input to the model is an image of resolution 224 x 224 and the output from the model is one of the 1000 possible class labels. This model has 33 convolutional layers with a fully connected layer at the end to provide class predictions. This model is
 Collage Inference: Tolerating Stragglers in Distributed Neural Network Inference using Coding

(a) Input: 2x2 Collage of 4 images
(b) Input: 3x3 Collage of 9 images
(c) Input: 4x4 Collage of 16 images

(d) Output: Collage with final predictions
(e) Output: Collage with final predictions
(f) Output: Collage with final predictions

Figure 4. Input and outputs of different collage CNN models

4.3 Training Data for Collage models

The datasets we used in our experiments are CIFAR-10 and Imagenet-1k (ILSVRC-2012) dataset. CIFAR-10 dataset consists of 60000 images divided into 10 object classes. 50000 images are provided for training and 10000 images for validation. Imagenet-1k dataset consists of 1.2 million images divided across 1000 object classes for training and 50000 images are provided for model validation. To train the Collage-CNN collages generated from images training datasets are used. To validate Collage-CNN collages generated from images in validation datasets are used. In our experiments with Imagenet-1k dataset, we picked all the training and validation images belonging to 100 of the 1000 classes for evaluations. The selected 100 classes correspond to 100 different objects. One of the reasons for this selection is that we are not aware of any pre-trained YOLO models for the complete Imagenet-1k dataset. Hence, to train the YOLO model in a reasonable time, and to experiment with various design spaces using limited compute resources, we selected the 100 class classification from Imagenet.

Let us look at the different collage architectures and the methods for generation of collage training and validation datasets. As explained in section 3.2 a collage of \(N\) images would be composed of \([\sqrt{N}, \sqrt{N}]\) square boxes. The image assigned to compute node 0 goes to square box indexed 0, image from node 1 goes to square box 1 and so on. We experimented with \(N = 4, 9, 16, 25\) images. Examples of different collage architectures are shown in figure 4. The 2X2 collage depicts the case where the four individual images are sent to
four compute nodes each running a S-CNN model for single image classification, while this single 2X2 collage created from the four images is sent to a single collage node for redundant detection. In this particular example the overhead of running the collage is approximately 25% more compute resources. This overhead can be reduced by using a 3X3 collage where one collage node acts as a redundant node for 9 single image classifier nodes (approximately 11% overhead). But note that the size of the collage is fixed and hence as more images are packed into a single collage the resolution of each image has to be reduced. As we show later in our results section this reduction in resolution has an adverse impact on the accuracy when the size of the collage exceeds 4X4.

During training, the target output given to the Collage-CNN model consists of $N \ast 5$ values. For each of the $N$ images in the collage there are 5 values: [class label, X-coordinate of center of the bounding box, Y-coordinate of center of bounding box, Width of bounding box, Height of bounding box]. In order to generate this training data, we wrote a python script that generates a square grid of images from a given dataset by appropriately scaling down the image size as needed to fit within the collage size. The script also generates the bounding box coordinates for each image in the collage along with the box dimensions as required for training.

For the CIFAR-10 based collage dataset we uniformly and at random pick $N$ images from the 50000 training images to create each collage in the training dataset. For the imitation-based collage dataset we first pick all the training images from the 100 classes. Then, we uniformly and at random pick $N$ classes from the 100 classes. We pick one image from each of these $N$ classes and generate the collage image. Similar procedure is followed for generating collages for validation.

One challenge in creation of collage images is that the total possible number of collage images can be exponentially larger than the training data from the raw dataset images. There are many possible permutations to choose from while combining many different images into collages. While this selection is a challenge, it also provides more training samples because the task being performed by the Collage-CNN is more challenging than the single image S-CNN models. The larger training data allows for improving the model and can help increase it’s validation accuracy as we show in evaluation section.

The second challenge is that traditionally object detection models such as YOLO not only detect objects, but they might also infer and learn relationships between objects based on where objects commonly appear in an image [44]. This might be useful in standard object detection datasets. In collage-CNN based detection this learning is clearly not a good assumption because objects belonging to any class can be present in any location in the image. So, by generating lot more permutations of images for training we try to prevent the model from learning any absent inter-object correlations.

The input resolution to our Collage-CNN model is set to a fixed 416 x 416 pixels. So, while forming collages each single image resolution is set to $\sqrt{\frac{416}{\sqrt{N}}} \times \sqrt{\frac{416}{\sqrt{N}}}$ pixels. For the CIFAR-10 dataset since each image is of resolution 32 x 32 pixels we do not have to lower resolution even for a very large $N$ value. However, for Imagenet-1k dataset, we have to lower the resolution even for the smallest 2X2 collage. We use the python imaging library to lower the resolution of each image before forming the collage.

4.4 Decoding predictions of the Collage model

For each input collage image, the Collage-CNN model provides several object predictions. Each prediction is of the form: [class label, confidence of the class, X-coordinate of left edge of the bounding box, X-coordinate of right edge, Y-coordinate of top edge, Y-coordinate of bottom edge]. For providing information useful to the decoder, we need to extract the $N$ classes in the collage from these predictions. This presented us with few challenges because:

- The model may predict multiple classes within the same bounding box. This issue occurs particularly with imagenet-1k based images since each image can have multiple objects.
- The bounding boxes predicted might be significantly different from the Ground Truth bounding boxes. The number of bounding boxes could be more or less than $N$ that was used in creating the collage.

We implement a processing algorithm that extracts the best possible values for the $N$ image classes from all the object predictions. We refer to this as the collage decoding algorithm. First, all the predictions with confidence values less than detection threshold are ignored by the algorithm. Then the decoding algorithm calculates the Jaccard similarity coefficient, also known as Intersection over Union (IoU), of each predicted bounding box with each of the $N$ ground truth bounding boxes that are used in creating the collages. Let area of ground truth bounding box be $A_{gt}$, area of predicted bounding box be $A_{pred}$ and area of intersection between both the boxes be $A_i$. Then Jaccard similarity coefficient can be computed using the formula: $\frac{A_i}{A_{gt} + A_{pred} - A_i}$.

The ground truth bounding box with the largest similarity coefficient is assigned the class label of the predicted bounding box. As a result the image present in this ground truth bounding box is predicted as having an object belonging to this class label. This is repeated for all the object predictions. If there are multiple class labels predicted for a ground truth bounding box, then ties are broken based on the confidence associated with the prediction. Class label with the largest confidence is chosen for the bounding box.

Examples of object predictions are shown in figure 5. The ground truth collage is a 2x2 collage and is formed from 4 images. It consists of 4 bounding boxes G1, G2, G3, and G4 which contain objects belonging to classes A, B, C, and
D respectively. In scenario 1, the collage model predicts 4 bounding boxes P1, P2, P3 and P4 that are non-overlapping with each other. The collar decoding algorithm computes the values of Jaccard similarity coefficient for each predicted box with the ground truth bounding boxes. In this scenario: P1 would have largest similarity value with G1, P2 with G2, P3 with G3 and P4 with G4. So collar decoding algorithm predicts G1 as belonging to class A, G2 to class E, G3 to class C, G4 to class D. Next consider scenario 2. In this scenario, three bounding boxes are predicted by the model. Predicted box P1 is spread over G1 and G2. Jaccard similarity coefficient value of P1 with box G1 is: \( \frac{1}{5} \), G2 is: \( \frac{1}{5} \), G3 is: \( \frac{1}{7} \) and G4 is: \( \frac{1}{7} \). So, collar decoding algorithm predicts G1 as containing class A, G2 as empty prediction, G3 as class C, G4 as class D. In scenario 3, collar model predicts 5 different bounding boxes. Assigning classes A, C, D to boxes G1, G3, G4 respectively is straightforward. But both box P2 and box P3 have highest Jaccard similarity coefficient values with ground truth box G2. Since class B has higher confidence (80%) than class E (70%), collar decoding algorithm predicts G2 as containing class B. In scenario 4, four bounding boxes are predicted. Assigning class B, class C, class D to boxes G2, G3 and G4 respectively is straightforward. Box P1 has highest similarity with ground truth box G1. Since two classes are predicted within box P1 the tie between both classes A and E is broken by using confidence. Since the confidence of predicting class A is larger than that of class E, G1 is assigned class A.

4.5 Decoding and Providing final predictions
The outputs from collar decoding algorithm along with predictions from all the S-CNN models are provided as inputs to the final decoder process. The decoder process provides the final system predictions as shown in figure 3. If the predictions from all the S-CNN models are available, the decoder just provides these predictions as the final predictions and discards the Collage-CNN outputs, since there were no stragglers in MLaaS. In the case where predictions from any of the S-CNN models is not available i.e., there is a straggler node, then the prediction from the Collage-CNN corresponding to that model is used instead. It can be observed that the outputs from Collage-CNN model can be used to tolerate more than one straggler. The predictions from the Collage-CNN model can be used instead in place of any missing S-CNN model predictions. In the rare scenario, when there is a straggler S-CNN model and the corresponding prediction from Collage-CNN is empty the image replicated to another S-CNN model. Prediction from this replicated request is used by the decoder process.

5 Experimental Evaluation
5.1 Training Parameters
The models are generally trained for 130K iterations using Stochastic Gradient Descent (SGD) with the following hyper parameters: learning rate of 0.001, momentum of 0.9, decay of 0.0005, and batch size of 64. While training Collage-CNN on Imagenet collages of shapes 4x4 and 5x5, the learning rate of 0.001 caused divergence in SGD. When the learning rate is reduced to 0.0005 SGD converged. Each model training is performed on a single compute node consisting of an AMD Ryzen 3 1200 Quad-Core Processor with 4 CPUs and 32GB of memory and a GeForce Titan 1080 GPU equipped with 11 GB of GDDRAM. The training run time is ~26 hours for 130K iterations.

As described in section 4, size of the Collage-CNN training data is increased using the different permutations possible when generating collages. We performed few experiments to verify that using more collage images in training helps to improve validation accuracy. We observed consistent improvements in validation accuracy. Some of the experimental results are listed below.

- While training a Collage-CNN model using 4x4 Imagenet based collages, as the training set size is doubled from 52K to 104K images validation accuracy increased by 6.95%.
- While training a Collage-CNN model using 3x3 Imagenet based collages, as the training set size is doubled from 26K to 52K images the validation accuracy increased by 1%.
- While training a Collage-CNN model using 3x3 CIFAR-10 based collages, as the training set size is increased...
from 10K to 50K images the validation accuracy increased by 1.38%.

Following from these experiments while training the Collage-CNN models on Imagenet dataset we used a larger number of collage images than the total number of single training images present in the Imagenet dataset. The total number of single training images in the selected 100 classes is 120K. For Collage-CNN model training we generated and used 208K collages.

5.2 Accuracy of the Collage-CNN models
We measured the top-1 accuracy of Collage-CNN and S-CNN models using validation images from CIFAR-10. The resolution of each validation image is unchanged while forming the collages. The size of CIFAR-10 dataset is 50000 training images, 10000 validation images. Let \( N \) be the number of images per collage. The collage set for each architecture consisted of \( \frac{50000}{N} \) training collages and \( \frac{10000}{N} \) test collages. The accuracy results are plotted in figure 6. The baseline S-CNN model has a accuracy of 92.2% whereas the 2x2 Collage-CNN models has a accuracy of 88.91%. Further, it can be seen that the accuracy of Collage-CNN models decreases gradually as the number of images per collage increases. As we discussed earlier in section 4.3, increasing the number of single images per collage can increase the redundancy that the Collage-CNN provides but it comes at a cost of decreasing accuracy. However, with more single images per collage the number of objects to be detected by the Collage-CNN is also higher. As a result, having more images per collages makes the task of Collage-CNN model more challenging.

Next we measured the top-1 accuracy of Collage-CNN and S-CNN models using validation images from Imagenet. Test collages are generated using the validation images from the selected 100 Imagenet classes. The resolution of each validation image is lowered to fit into the collage. This is because each validation image has a resolution of 224x224 and the collage image resolution is 416x416. The top-1 accuracy results are plotted in figure 7. Surprisingly, Collage-CNN model demonstrated slightly higher accuracy than baseline S-CNN model on single image inputs i.e., 1x1 collage and also 2x2 collage images. Again, as the number of images per collage increases the accuracy of Collage-CNN model decreases gradually. It can be observed that the rate of decrease in accuracy of Collage-CNN model on Imagenet is higher than on CIFAR-10. As the number of single images per Imagenet collage is increased, resolution of each image gets reduced significantly unlike with CIFAR-10 collages. This is likely the cause for higher loss in accuracy.

5.3 Inference Latency of the models
The inference latency of Collage-CNN and S-CNN on an AMD Ryzen 3 1200 Quad-Core Processor with 4 CPUs and 32GB of memory are shown in table 1. We measured latency on a batch size of 1 image because there would be no batching of images during online or real-time inference. Both models have similar inference latency on images from Imagenet dataset.

While performing inference using Collage-CNN and S-CNN there is a latency overhead associated with pre-processing each input image. For S-CNN the pre-processing includes resizing each image to 256x256 and then cropping it down to 224x224. For Collage-CNN the pre-processing includes creating the 416x416 collage by lowering the resolution of single images. The mean latencies for creating different Imagenet collages are shown in table 2. As the number of images in a collage increases, its creation latency increases proportionally. For a 3x3 collage, the creation time is ~13% of the mean of the corresponding inference latency.
We implemented an online image classification system and written in Python. The front node generates and sends http requests with the images for prediction. Flask is a micro web framework deployed it on the Digital Ocean cloud [2]. The system consists of a Load Balancer front node, multiple server nodes running S-CNN and Collage-CNN models. The Load Balancer front node performs multiple tasks. It collects requests from clients and generates single image classification requests to the S-CNN models. It also creates a collage from these single images and sends collage classification request to the Collage-CNN. It can replicate any single image requests if necessary. It also performs the decoding process described in section 4.5. We use one Virtual Machine (VM) to host the front node and N additional VMs to serve requests using the S-CNN and Collage-CNN models. Each of the server nodes runs a Flask based http server to serve incoming http requests with images for prediction. Flask is a micro web framework written in Python. The front node generates and sends http requests with the images. We performed experiments with N = 9, 16 S-CNN server nodes and 1 Collage-CNN server node. Validation images from Imagenet-1k dataset are used to generate inference requests.

Along with Collage Inference, we implemented two more methods for comparison. Each method is briefly described below:

- First method is where the front node sends requests to the S-CNN servers and waits till all of them respond. The front node does not replicate any slow and pending requests. This is the No replication method.
- In the second method, the front node sends requests to the S-CNN servers with a fixed timeout on all requests. If a server is a straggler and does not provide prediction before the timeout, the request is replicated. This is the replication method.
- During Collage Inference, the front node sends requests to the S-CNN and Collage-CNN servers with a fixed timeout on all requests. If one or more S-CNN servers are stragglers and do not provide predictions before the time out and the Collage-CNN provides predictions, the predictions from the Collage-CNN are used in the place of missing S-CNN predictions. If one or more S-CNN servers do not provide predictions before the timeout and the Collage-CNN also does not provide predictions before the time out, the requests sent to stragglings S-CNN nodes are replicated to the S-CNN nodes that already finished their requests.

During the experiments, we make each of the N server nodes follow the same inference latency distribution shown in figure 2 such that for every image inference, the mean of the inference latency is ~0.15 second and the 99-percentile inference latency is ~0.70 seconds. Following the measured distribution is to allow for comparing the latencies between the three different methods. The front node measures and logs the end-to-end latency for each request from the time it is sent to the time predictions for it are received. For requests to Collage-CNN model the end-to-end latency also includes time spent in forming the collage image.

### Table 1. Measured Inference Latencies

| BatchSize | Model    | Mean(ms) | Std. Deviation(ms) |
|-----------|----------|----------|--------------------|
| 1         | S-CNN    | 96       | 8.4                |
| 1         | Collage-CNN | 95      | 11                 |

### Table 2. Collage creation Latencies

| Collage architecture | Mean Collage creation latency(ms) |
|----------------------|----------------------------------|
| 1x1                  | 3                                |
| 2x2                  | 6                                |
| 3x3                  | 12                               |
| 4x4                  | 22                               |
| 5x5                  | 32                               |

#### 5.4 Collage Inference on the Cloud

We implemented an online image classification system and deployed it on the Digital Ocean cloud [2]. The system consists of a Load Balancer front node, multiple server nodes running S-CNN and Collage-CNN models. The Load Balancer front node performs multiple tasks. It collects requests from clients and generates single image classification requests to the S-CNN models. It also creates a collage from these single images and sends collage classification request to the Collage-CNN. It can replicate any single image requests if necessary. It also performs the decoding process described in section 4.5. We use one Virtual Machine (VM) to host the front node and N additional VMs to serve requests using the S-CNN and Collage-CNN models. Each of the server nodes runs a Flask based http server to serve incoming http requests with images for prediction. Flask is a micro web framework written in Python. The front node generates and sends http requests with the images. We performed experiments with N = 9, 16 S-CNN server nodes and 1 Collage-CNN server node. Validation images from Imagenet-1k dataset are used to generate inference requests.

Along with Collage Inference, we implemented two more methods for comparison. Each method is briefly described below:

- First method is where the front node sends requests to the S-CNN servers and waits till all of them respond. The front node does not replicate any slow and pending requests. This is the No replication method.
- In the second method, the front node sends requests to the S-CNN servers with a fixed timeout on all requests. If a server is a straggler and does not provide prediction before the timeout, the request is replicated. This is the replication method.
- During Collage Inference, the front node sends requests to the S-CNN and Collage-CNN servers with a fixed timeout on all requests. If one or more S-CNN servers are stragglers and do not provide predictions before the time out and the Collage-CNN provides predictions, the predictions from the Collage-CNN are used in the place of missing S-CNN predictions. If one or more S-CNN servers do not provide predictions before the timeout and the Collage-CNN also does not provide predictions before the time out, the requests sent to stragglings S-CNN nodes are replicated to the S-CNN nodes that already finished their requests.

During the experiments, we make each of the N server nodes follow the same inference latency distribution shown in figure 2 such that for every image inference, the mean of the inference latency is ~0.15 second and the 99-percentile inference latency is ~0.70 seconds. Following the measured distribution is to allow for comparing the latencies between the three different methods. The front node measures and logs the end-to-end latency for each request from the time it is sent to the time predictions for it are received. For requests to Collage-CNN model the end-to-end latency also includes time spent in forming the collage image.

#### 9 S-CNN + 1 Collage-CNN server nodes:

The end to end latency distribution observed when the image classification system consists of 9 S-CNN models with no replication method is shown in figure 8; 9 S-CNN models with replication method is shown in figure 9; 9 SNN models and 1=one 3x3 Collage-CNN model with Collage Inference is shown in figure 10. X-axis in each figure is the latency in seconds. The histograms along Y-axis are the probability density values for the latency distribution. The blue curve line along Y-axis shows the estimated Probability Density Function (PDF) of the end to end latency using Kernel Density Estimation. Let’s compare the methods across 4 metrics.

- Mean latency: No replication method has the lowest mean latency (0.40s), followed by Replication (0.42s) and Collage inference (0.48s). Collage inference has a slightly higher mean mainly owing to the collage creation overhead in the Collage-CNN model.
- Standard Deviation: Both No replication and Replication methods have a similar standard deviation of 0.18s whereas Collage inference has a significantly lower standard deviation of 0.06s. Using Collage-CNN model reduces the standard deviation in latency by 3X.
- 99-th percentile latency: The 99-th percentile latency of Collage inference (0.60s) and is 1.47X lower than both No replication and Replication methods.
- Accuracy: Accuracy related statistics for Collage inference method are shown in table 3. Each request mentioned in this table refers to a distributed inference request. That is, it consists of 9 concurrent requests to S-CNN models and 1 concurrent request to the Collage-CNN model. When the Collage-CNN is not a straggler and its predictions are used by the decoder, the accuracy of its predictions is 87.86%. This is significantly better than the top1-accuracy of 3x3 Collage-CNN on
Table 3. Accuracy statistics for 3x3 Collage-CNN

|                                |            |
|--------------------------------|------------|
| Total requests                 | 8310       |
| Total requests that encountered stragglers | 1478       |
| Collage-CNN is one of the stragglers | 201        |
| Collage-CNN is not a straggler  | 1277       |
| Collage-CNN predicted accurately | 1122       |
| Accuracy of Collage-CNN when it is used | 87.86%     |

Table 4. Accuracy statistics for 4x4 Collage-CNN

|                                |            |
|--------------------------------|------------|
| Total requests                 | 4680       |
| Total requests that encountered stragglers | 1002       |
| Collage-CNN is one of the stragglers | 313        |
| Collage-CNN is not a straggler  | 689        |
| Collage-CNN predicted accurately | 563        |
| Accuracy of Collage-CNN when it is used | 81.71%     |

16 S-CNN + 1 Collage-CNN server nodes: The end to end latency distribution observed when the image classification system consists of 16 S-CNN models with no replication method is shown in figure 11; 16-S-CNN models with replication method is shown in figure 12; 16-SNN models and one 4x4 Collage-CNN model with Collage Inference is shown in figure 13. Let’s compare the methods across 4 metrics.

- Mean latency: Replication method has the lowest mean latency (0.54s), followed by No replication method (0.61s) and Collage inference (0.67s). The higher mean latency of Collage inference is due to the collage creation overhead in the Collage-CNN model.
- Standard Deviation: Again Collage inference has the lowest standard deviation of 0.07s. This is 4X lower than Replication method and 9X lower than the No Replication method.
- 99-th percentile latency: The 99-th percentile latency of Collage inference (0.96s) is 1.45X lower than Replication (1.39s) and 2.46X lower than the No Replication method (3.32s).
- Accuracy: Accuracy related statistics for Collage inference method are shown in table 4. When the 4x4 Collage-CNN is not a straggler and it’s predictions are used by the decoder, the accuracy of its predictions is 81.71%. This prediction accuracy is lower than that provided by 3x3 Collage-CNN (87.86%). This is expected because 4x4 model is providing redundancy for 16 S-CNN models. However, this prediction accuracy is significantly higher than the top1-accuracy of 4x4 Collage-CNN on Imagenet (72.38%) shown in figure 7. The difference arises because when the Collage-CNN is used only a subset of it’s predictions corresponding to the straggler nodes are taken into account. Again, the Collage-CNN provides a significantly lowers variation in latency without decreasing inference accuracy.

6 Related Work

For earlier discussion on related work please refer section 3.1.

Tail latency in Distributed systems: Paragon [17] presents a QOS aware online heterogenous datacenter scheduler. Adrenaline [28] identifies and selectively speeds up long queries by quick voltage boosting. Prior works like [18, 34, 41, 57] focus on
Another strategy used for straggler mitigation is arriving server B. This approach is similar to the reactive approach \[35\]\[36\]. In this replication when there is failure of a parameter server B, the parameter server A, \(B\), would send a request for the new data to the datacenter. Using replicated tasks to improve the response times needs multiple replicas of all the data which adds overheads. Another strategy used for straggler mitigation is arriving at an approximate result without waiting on the stragglers. In this work we present Collage Inference and a novel redundancy model called Collage-CNN that provides redundancy in the presence of straggler nodes, maintains high model accuracy and low resource overheads. Collage Inference uses lightweight object detection models to provide recovery from straggler nodes at runtime. Collage-CNN model provides good tradeoffs between accuracy and latency. Our experiments on CIFAR-10 dataset demonstrate a ~26% higher top-1 accuracy than the existing works. In addition, we demonstrate excellent results on Imagenet-1k dataset. Deploying the Collage-CNN models in the cloud we demonstrate that the 99-th percentile latency can be reduced by 1.45X to 2.46X compared to replication based approaches without compromising prediction accuracy. We conclude that Collage Inference is a new and promising approach to mitigate stragglers in distributed inference.

### 7 Conclusion

Online prediction serving systems are becoming increasingly used to serve image processing based requests. Serving requests at low latency, with high accuracy and low resource cost becomes very important; especially in the presence of stragglers. In this work we present Collage Inference and a novel redundancy model called Collage-CNN that provides redundancy in the presence of straggler nodes, maintains high model accuracy and low resource overheads. Collage Inference uses lightweight object detection models to provide recovery from straggler nodes at runtime. Collage-CNN model provides good tradeoffs between accuracy and latency. Our experiments on CIFAR-10 dataset demonstrate a ~26% higher top-1 accuracy than the existing works. In addition, we demonstrate excellent results on Imagenet-1k dataset. Deploying the Collage-CNN models in the cloud we demonstrate that the 99-th percentile latency can be reduced by 1.45X to 2.46X compared to replication based approaches without compromising prediction accuracy. We conclude that Collage Inference is a new and promising approach to mitigate stragglers in distributed inference.

### References

1. [n. d.]. AWS Sage Maker. https://aws.amazon.com/sagemaker/.
2. [n. d.]. Digital Ocean. https://www.digitalocean.com.
3. [n. d.]. Github AlexeyAB Darknet. https://github.com/AlexeyAB/darknet.
4. [n. d.]. Google Cloud ML Engine. https://cloud.google.com/ml-engine/.
5. [n. d.]. Microsoft Azure ML Service. https://azure.microsoft.com/en-us/services/machine-learning-service/.
6. 2014. Apache Hadoop. http://hadoop.apache.org/.
7. Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A System for Large-Scale Machine Learning. In 12th {USENIX} Symposium on Operating Systems Design and Implementation ({{OSDI}}'16) 265–283.
8. Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective Straggler Mitigation: Attack of the Clones. In Proceedings of the 10th {USENIX} Conference on Networked Systems Design and Implementation (nsdi'13). {USENIX} Association, Berkeley, CA, USA, 185–198. http://dl.acm.org/citation.cfm?id=2482626.2482645
9. Ganesh Ananthanarayanan, Srikanth Kandula, Albert G Greenberg, Ion Stoica, Yi Lu, Bikas Saha, and Edward Harris. 2010. Reining in the Cost of Hadoop and has the same drawbacks. Parameter server B may now become a bottleneck to training due to increased communication to and from it.

**Coded Computation:** Coded computation methods have been proposed to primarily provide resiliency to stragglers in distributed machine learning \[19, 32, 38, 46, 53\]. Most of the works target linear machine learning algorithms. A recent work which is close to this paper and was previously discussed in section 3 is \[29\]. Our work differs from \[29\] in encoding and decoding methods, training methodology, and demonstrates significantly higher accuracy in the presence of stragglers. We also provide evaluations of our method on the challenging Imagenet dataset.

**Straggler in Distributed training:** In Straggler mitigation techniques have been studied for speeding up Distributed training. Parameters servers are used in distributed SGD training. To mitigate failures, distributed parameter server approaches use chain replication of the parameter servers \[35\]\[36\]. In this replication when there is failure of a parameter server A its load is shifted onto another parameter server B. This approach is similar to the reactive approach of Hadoop and has the same drawbacks. Parameter server B may now become a bottleneck to training due to increased communication to and from it.
Outliers in Map-Reduce Clusters using Mantri. In OSDI, Vol. 10. 24.

[10] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. 1991. The Information Visualizer, an Information Workspace. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI ’91). ACM, New York, NY, USA, 181–186. https://doi.org/10.1145/108844.108874

[11] Mannmohan Chauhey and Erik Saule. 2015. Replicated Data Placement for Uncertain Scheduling. In Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW ’15). IEEE Computer Society, Washington, DC, USA, 464–472. https://doi.org/10.1109/IPDPSW.2015.50

[12] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. 2016. Revisiting distributed synchronous SGD. arXiv preprint arXiv:1609.00981 (2016).

[13] Daniel Crankshaw, Xin Wang, Guiluo Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. 2017. Clipper: A Low-Latency Online Prediction Serving System. In 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17). USENIX Association, Boston, MA, 613–627. https://www.usenix.org/conference/nsdi17-technical-sessions/presentation/crankshaw

[14] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. In Advances in Neural Information Processing Systems. 379–387.

[15] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. Commun. ACM 56 (2013), 74–80. https://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

[16] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. Commun. ACM 51, 1 (2008), 107–113.

[17] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In ACM SIGPLAN Notices, Vol. 48. ACM, 77–88.

[18] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware cluster Management. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’14). ACM, New York, NY, USA, 127–144. https://doi.org/10.1145/2541940.2541941

[19] Sanghamitra Dutta, Viveck Cadambe, and Pulkit Grover. 2016. Short-Dot: Computing Large Linear Transforms Distributedly Using Coded Short Dot Products. In Advances In Neural Information Processing Systems. 2092–2100.

[20] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. 2017. DSSD: Deconvolutional single shot detector. arXiv preprint arXiv:1701.06659 (2017).

[21] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyyttia. 2015. Reducing Latency via Redundant Requests: Exact Analysis. In Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS ’15). ACM, New York, NY, USA, 347–360. https://doi.org/10.1145/2745844.2745873

[22] Ross Girshick. 2015. Fast r-cnn. In Proceedings of the IEEE international conference on computer vision, 1440–1448.

[23] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition. 580–587.

[24] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakte, and Thu D. Nguyen. 2015. ApproxHadoop: Bringing Approximations to MapReduce Frameworks. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’15). ACM, New York, NY, USA, 383–397. https://doi.org/10.1145/2694334.2694351

[25] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch SGD: training imagenet in 1 hour. arXiv preprint arXiv:1706.02677 (2017).

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.

[27] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017).

[28] Chang-Hong Hsu, Yunqi Zhang, Michael A. Laurenczo, David Meisner, Thomas Wenisch, Ronald G. Dreelinke, Jason Mars, and Lingjia Tang. 2017. Reining in Long Tails in Warehouse-Scale Computers with Quick Voltage Boosting Using Adrenaline. ACM Trans. Comput. Syst. 35, 1, Article 2 (March 2017), 33 pages. https://doi.org/10.1145/3054742

[29] J. Kosaian, K. V. Rashmi, and S. Venkataraman. 2018. Learning a Code: Machine Learning for Approximate Non-Linear Coded Computation. arXiv e-prints (June 2018). arXiv:cs.LG/1806.01259

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems. 1097–1105.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. Proc. IEEE 86, 11 (1998), 2278–2324.

[32] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. 2016. Speeding up distributed machine learning using codes. In 2016 IEEE International Symposium on Information Theory (ISIT). 1143–1147. https://doi.org/10.1109/ISIT.2016.7541478

[33] Kangwook Lee, Ramtin Pedarsani, and Kannan Ramchandran. 2017. On Scheduling Redundant Requests With Cancellation Overheads. IEEE/ACM Trans. Netw. 25, 2 (April 2017), 1279–1290. https://doi.org/10.1109/TNET.2016.2622248

[34] Jacob Leverich and Christos Kozyrakis. 2014. Reconciling High Server Utilization and Sub-millisecond Quality-of-service. In Proceedings of the Ninth European Conference on Computer Systems (EuroSys ’14). ACM, New York, NY, USA, Article 4, 14 pages. https://doi.org/10.1145/2592798.2592821

[35] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In OSDI.

[36] Mu Li, David G. Andersen, Alexander J. Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. In NIPS.

[37] Songze Li, Mohammad Ali Maddah-Ali, and Amir Salman Avestimehr. 2015. Coded MapReduce. 53rd Allerton Conference (Sept. 2015).

[38] Songze Li, Mohammad Ali Maddah-Ali, and A Salman Avestimehr. 2016. A Unified Coding Framework for Distributed Computing with Straggling Servers. e-print arXiv:1609.01690 (Sept. 2016). A shorter version to appear in IEEE NetCod 2016.

[39] Songze Li, Mohammad Ali Maddah-Ali, Qian Yu, and A Salman Avestimehr. 2016. A Fundamental Tradeoff between Computation and Communication in Distributed Computing, to appear in IEEE Transactions on Information Theory (2016).

[40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. Ssd: Single shot detector. In Conference on computer vision and pattern recognition. Springer, 21–37.

[41] David Lo, Liqun Cheng, Rama Govindaraju, Luiz André Barroso, and Christos Kozyrakis. 2014. Towards Energy Proportionality for Large-scale Latency-critical Workloads. In Proceeding of the 41st Annual International Symposium on Computer Architecture (ISCA ’14). IEEE Press, Piscataway, NJ, USA, 301–312. https://dl.acm.org/citation.cfm?id=2665671.2665718
Collage Inference: Tolerating Stragglers in Distributed Neural Network Inference using Coding

[42] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. PyTorch.

[43] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In Advances in neural information processing systems. 693–701.

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition. 779–788.

[45] Joseph Redmon and Ali Farhadi. 2018. Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018).

[46] Amirhossein Reisizadeh, Saurav Prakash, Ramtin Pedarsani, and Amir Salman Avestimehr. 2017. Coded Computation over Heterogeneous Clusters. In 2017 IEEE International Symposium on Information Theory (ISIT).

[47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems. 91–99.

[48] Nihar B. Shah, Kangwook Lee, and Kaman Ramchandran. 2013. When do redundant requests reduce latency ?. In 2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton). 731–738. https://doi.org/10.1109/Allerton.2013.6736597

[49] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. 2017. Dsod: Learning deeply supervised object detectors from scratch. In The IEEE International Conference on Computer Vision (ICCV), Vol. 3. 7.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[51] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition. 1–9.

[52] Da Wang, Gauri Joshi, and Gregory Wornell. 2014. Efficient Task Replication for Fast Response Times in Parallel Computation. SIGMETRICS Perform. Eval. Rev. 42, 1 (June 2014), 599–600. https://doi.org/10.1145/2637364.2592042

[53] Qian Yu, Mohammad Maddah-Ali, and A. Salman Avestimehr. 2017. Polynomial Codes: an Optimal Design for High-Dimensional Coded Matrix Multiplication. In to appear Advances In Neural Information Processing Systems (NIPS).

[54] Sergey Zagoruyko and Nikos Komodakis. 2016. Wide residual networks. arXiv preprint arXiv:1605.07146 (2016).

[55] Matei Zaharia, Andy Konwinski, Anthony D. Joseph, Randy Katz, and Ion Stoica. 2008. Improving MapReduce Performance in Heterogeneous Environments. In Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation (OSDI’08). USENIX Association, Berkeley, CA, USA, 29–42. http://dl.acm.org/citation.cfm?id=1855741.1855744

[56] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2019. Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems (2019).

[57] Timothy Zhu, Michael A. Kozuch, and Mor Harchol-Balter. 2017. WorkloadCompactor: Reducing Datacenter Cost While Providing Tail Latency SLO Guarantees. In Proceedings of the 2017 Symposium on Cloud Computing (SoCC ’17). ACM, New York, NY, USA, 598–610. https://doi.org/10.1145/3127479.3132245