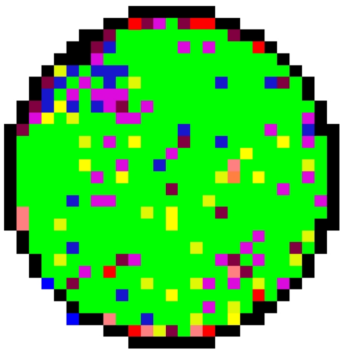
Self-Supervised Representation Learning of Semiconductor Wafer Maps

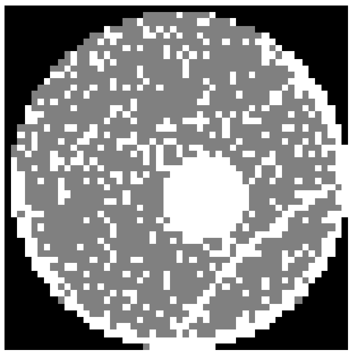
Faris Khan and Hibban Butt

1. Problem Description

1.1 Background Information

Semiconductor device fabrication involves a variety of processing steps to create integrated circuits (chips). After these fabrication processes are complete, semiconductor wafers go through a series of “probe” tests to determine the functionality of the chips on each wafer. The results of these tests are summarized in wafer maps, like the one below.





Typically, there are two types of wafer maps used to visualize probe data. Multi-bin wafer maps, like the one on the left, show where and how a chip fails. In wafer testing, a single probe program involves several electrical tests that check different aspects of device functionality, including device logic, memory, current leakage, short circuits, etc. If a chip fails a certain test, it is assigned a unique code or “bin” corresponding to that test. Wafer map images display these bins as distinct colors. For the image on the left, all the pink pixels would have failed the same test. The second type of wafer map is a “binary” or “pass/fail” wafer map, which simply shows where chips are failing, not *how* they are failing. Use cases of multi-bin vs binary wafer maps depend on device/process maturity and whether they’re being used in product/process development or high-volume manufacturing.

Issues in fabrication processes often lead to distinct “failure shadings” on wafer maps. For example, the binary wafer map shown earlier has a scratch in the bottom-right quadrant of the wafer. It’s quite possible that a surface particle caused the scratch during a chemical mechanical planarization step. Semiconductor engineers often have several wafer maps pulled up on a screen at once to compare wafer shadings in a wafer “lot” (group of 25 wafers processed together in manufacturing). In a wafer fab, you’ll often hear engineers say something like “A dozen of these wafers in Lot A are showing heavy Bin 10 fallout in the center and near the edge, just like the Lot B wafers from yesterday."

1.2 Summary of the Issue

Engineers use wafer map images to visually compare failure shadings across wafers, but thousands of wafers may be probed in a day. Even though you may have a different root cause of fabrication issues on two similar-looking wafers, **visual similarity** between wafers is often the first step in a fab engineer's data mining process.

Failure shadings are unlabeled; there often aren't any systems in place to automatically assign wafer shadings to every wafer that is probed. There is no way to query a database of wafer maps by particular failure shadings. There is no way to compare visual similarity between wafer maps without using your eyes, since **there is no visual similarity metric** to begin with.

2. Why Use Deep Learning Here?

In the earliest papers about machine learning applied to wafer map datasets, the creation of similarity metrics relied on hand-crafted features fed into classical machine learning algorithms (often SVM in conjunction with PCA). Researchers would compute geometric and statistical features from wafer maps themselves, or at best use simple "classical" computer vision techniques like thresholding to create visual representations of wafer maps. Why hand-craft these features at all, when deep learning can learn such strong visual representations from images?

3. Our Proposed Approach

This project is not the first application of deep learning to wafer map datasets, nor is it the first in learning visual representations of wafer maps in a self-supervised manner. However, we'd like to explore a recent paper on self-supervised learning and apply it to wafer map data. We believe that unsupervised/self-supervised learning techniques are the way to go here. In real-life, failure shading data is large and unlabeled. There are a variety of downstream image tasks that a fab would want to do using wafer maps, such as image retrieval and object detection. Therefore, we need a method of deep learning that learns strong, transferable visual representations of images to be used for a variety of downstream image tasks. Contrastive self-supervised learning has proven to be very powerful in this regard, but it is known to be very computationally intensive compared to other techniques.

A recent paper proposes "FastSiam," a modification of the SimSiam framework from 2020. The hallmarks of FastSiam are improved training stability and faster convergence (to a baseline performance of ImageNet weights) compared to SimSiam and many other contrastive learning techniques such as SimCLR. It's a relatively recent paper (late September 2022), so we'll attach the PDF to this proposal. We aren't necessarily drawn to FastSiam because of its power in learning strong visual representations. It doesn't claim to be anything like DINO. Rather, we're interested in it because of its speed:

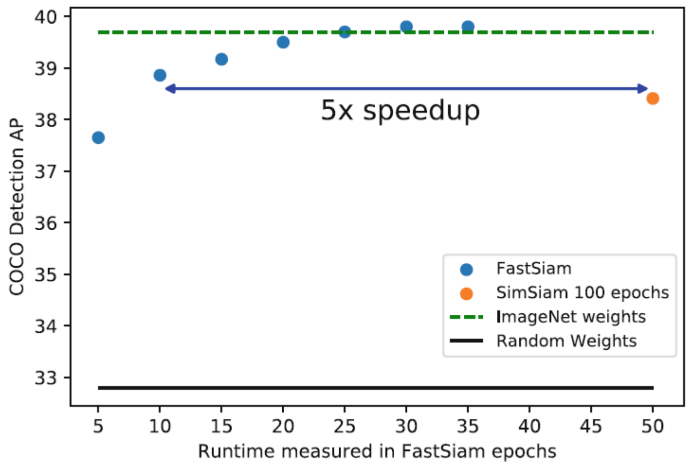


Fig. 1. Finetuning pretrained weights for object detection on MS COCO [22] following the 1x evaluation protocol with a Mask R-CNN and FPN-backbone as first described in MoCo [18]. FastSiam matches ImageNet weights if trained with a batch size of 32 on a single GPU for 25 epochs. FastSiam uses 4 views per image whereas SimSiam uses 2. Because of that FastSiam requires twice as long per epoch than SimSiam does but overall runtime to match SimSiam is still cut by a factor of more than 5.

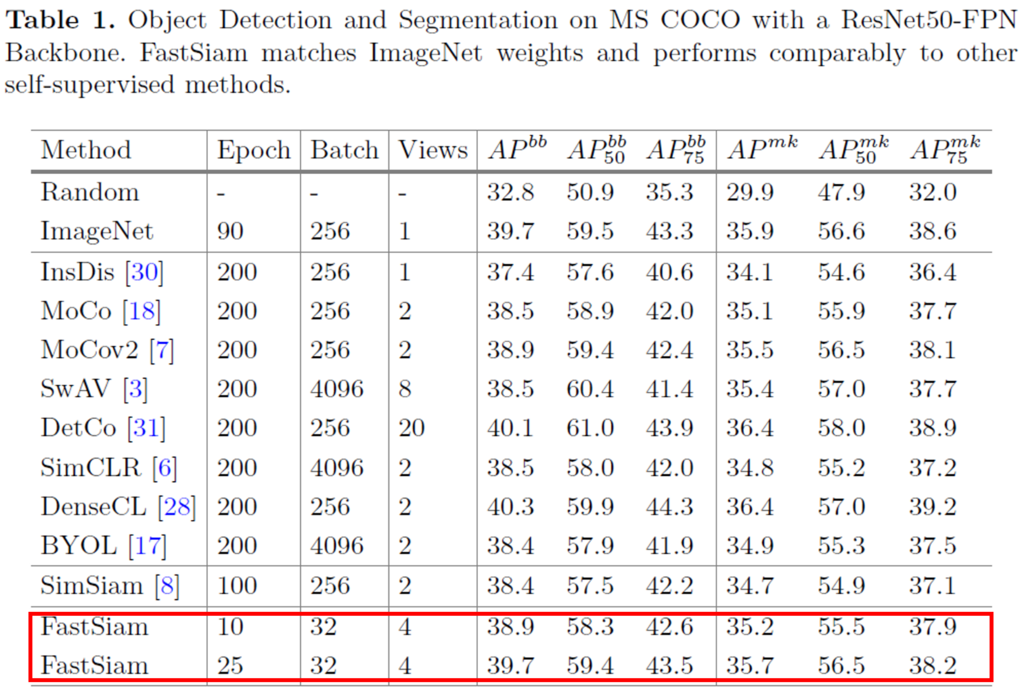


Table 1. Object Detection and Segmentation on MS COCO with a ResNet50-FPN Backbone. FastSiam matches ImageNet weights and performs comparably to other self-supervised methods.

Method	Epoch	Batch	Views	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{mk}	AP^{mk}_{50}	AP^{mk}_{75}
Random	-	-	-	32.8	50.9	35.3	29.9	47.9	32.0
ImageNet	90	256	1	39.7	59.5	43.3	35.9	56.6	38.6
InsDis [30]	200	256	1	37.4	57.6	40.6	34.1	54.6	36.4
MoCo [18]	200	256	2	38.5	58.9	42.0	35.1	55.9	37.7
MoCov2 [7]	200	256	2	38.9	59.4	42.4	35.5	56.5	38.1
SwAV [3]	200	4096	8	38.5	60.4	41.4	35.4	57.0	37.7
DetCo [31]	200	256	20	40.1	61.0	43.9	36.4	58.0	38.9
SimCLR [6]	200	4096	2	38.5	58.0	42.0	34.8	55.2	37.2
DenseCL [28]	200	256	2	40.3	59.9	44.3	36.4	57.0	39.2
BYOL [17]	200	4096	2	38.4	57.9	41.9	34.9	55.3	37.5
SimSiam [8]	100	256	2	38.4	57.5	42.2	34.7	54.9	37.1
FastSiam	10	32	4	38.9	58.3	42.6	35.2	55.5	37.9
FastSiam	25	32	4	39.7	59.4	43.5	35.7	56.5	38.2

The authors note that a large portion of the computer vision community is effectively excluded from the recent advances in self-supervised learning of visual representations, since not every lab has access to 8 TPUs. Ironically, the authors have not yet shared their code implementation of FastSiam, effectively excluding all of us from using it. **We'd like to implement FastSiam and make our implementation publicly available, working off of existing implementations of SimSiam in order to do so**. Below is the loss and overall setup of FastSiam from the paper. It doesn't seem impossibly difficult to implement, so we'd like to put it to the test.

SimSiam loss: $L(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2}$

FastSiam loss:

of views. The resulting loss function in which multiple views are combined for target generation is shown in Eq. 4.

$$L(z_1, z_2, z_3, ..., z_K, p_{K+1}) = -\frac{p_{K+1}}{\|p_{K+1}\|_2} \cdot \frac{\frac{1}{K}\sum_{i=1}^{K} z_i}{\|\frac{1}{K}\sum_{i=1}^{K} z_i\|_2} = -\frac{p_{K+1}}{\|p_{K+1}\|_2} \cdot \frac{\bar{T}_K}{\|\bar{T}_K\|_2} \tag{4}$$

The total loss that is optimized is formulated in Eq. 5. Each view is passed through the prediction branch once and the remaining views are used for target computation. We find $K = 3$ to be best, meaning that the combination of three views is used for target computation.

$$L_{total} = \sum_{i=1}^{K+1} \frac{1}{K+1} L(\{z_j | j \neq i \wedge 1 \leq j \leq K+1\}, p_i) \tag{5}$$

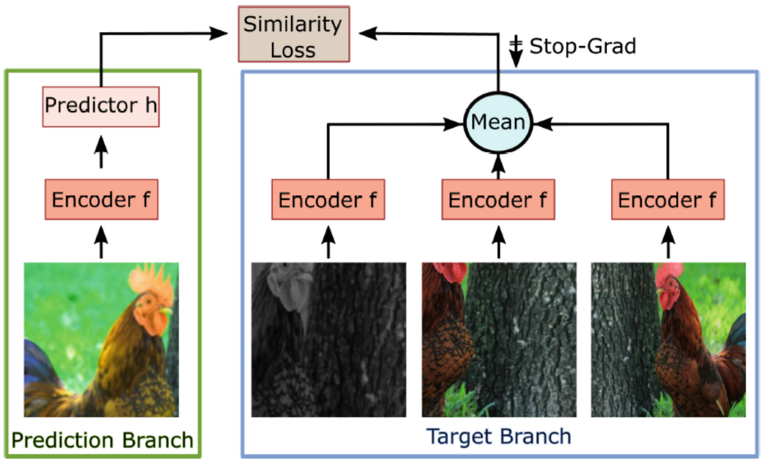
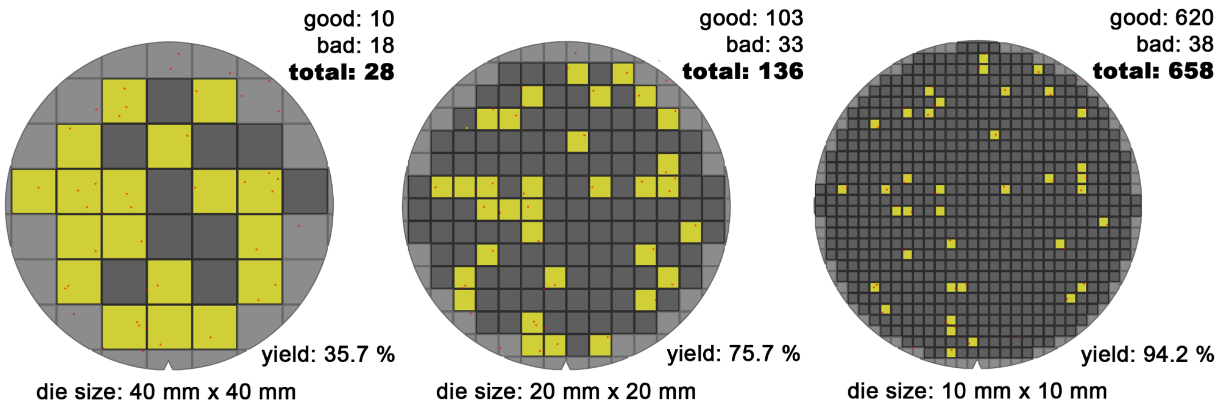


Fig. 3. Overall setup of FastSiam. On the target branch, FastSiam combines the information from multiple views to increase target stability. Gradients are not backpropagated, meaning that the target vector can be seen as a constant. On the prediction branch, only one view is used. The resulting prediction is trained to be similar to the output of the target branch. By increasing target stability, convergence can be reached much faster and with smaller batch sizes both in terms of unique images and the total number of views.

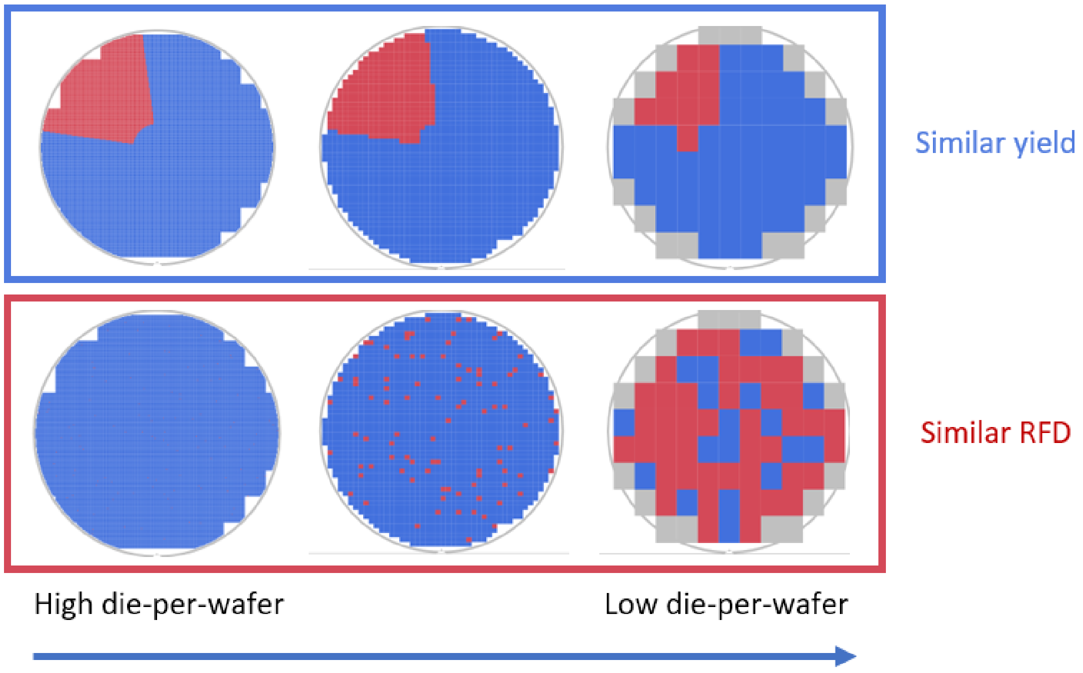
If we can't implement FastSiam, we can use existing frameworks such as SimSiam, SimCLR, or MoCo instead. For any type of self-supervised learning on images, you need to define the set of image augmentations to use. In other words, what transformations must my visual representation be invariant to? Many traditional augmentation techniques such as color jitter and Gaussian blur don't make sense for wafer maps. One of the paper's we've linked does an excellent job of detailing which augmentations are valid. In short, random horizontal flips, random resized crops, and random rotations are valid for wafer maps. The authors of that paper also define a few custom augmentation techniques, such as a simple function that flips a passing die to a failing die and vice-versa for a wafer (effectively adding random die noise to wafer maps). We plan to use these augmentation techniques during our self-supervised training.

An interesting area to explore would be an image augmentation that transforms the die-per-wafer (DPW) density of a wafer map. In a 300mm wafer fab, every wafer has the same size (300mm diameter), but different products can have different chip sizes. Small chips may have wafers of tens of thousands of dies (chips) per wafer, while large chips may only have a DPW density of one thousand.

The image below (Wikimedia link to full resolution for better viewing) shows the effect of a fixed root manufacturing issue manifesting as very different-looking wafer maps for high and low DPW wafer maps. Imagine sneezing on a wafer with 10,000 chips. "Only" a few hundred would fail due to your sneeze particles. Now imagine sneezing on a wafer with just 30 chips. Chances are, most of the chips will not yield.



Yield percent and random fail density (RFD) are two methods of calculating defectivity on wafers. The image on the next page shows that systemic yield issues ("correlated" or one-piece wafer shadings) have similar yield percents but different RFDs. On the other hand, purely random fails like "sneezes" lead to similar RFDs but different yield percents for wafers (the wikimedia image above indeed shows very different yield percents for varying DPW and fixed random issue). This is more of a nice-to-have feature, but it would be really cool to "pixelate" wafer maps to make them "lower DPW versions" of themselves, so you would effectively learn a visual representation of a wafer map that is invariant to DPW density. To our knowledge, something like this has not been done before.

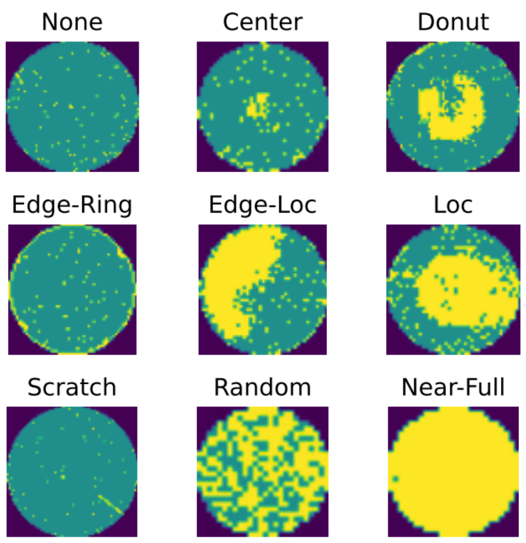


The actual task we'd like to perform after learning our visual representations is image retrieval. Like the example given in the introduction, very often a semiconductor engineer wants to know which wafer maps are visually the most similar to a "query" wafer map. This blog post about Tensorflow Similarity gave us this idea. We essentially want to learn an embedding/metric space of our wafers in a self-supervised manner using FastSiam, then do a few image retrieval examples using the *labels* from labeled WM811K wafers to compute our accuracy metrics.

4. Data

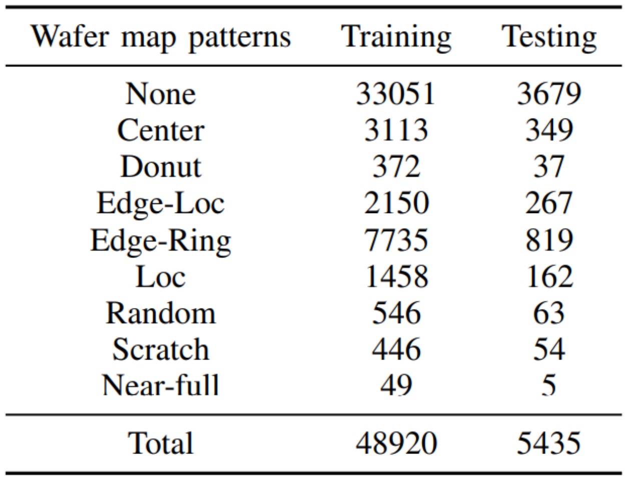
We plan to use the WM-811K wafer map dataset for this project. This is the largest publicly available dataset of real binary wafer maps that come from a few unnamed semiconductor fabs. The dataset includes wafers that have a wide range of die-per-wafer density, so each wafer map matrix doesn't have the same dimensions. However, each wafer map is typically resized to be a 128 by 128 single-channel image.

The whole dataset has around 811,000 wafer maps, but only about 173,000 of these are labeled. Of these labeled wafers, the original authors have arbitrarily assigned 54,355 as "training" and 118,595 as "test" data. The authors have assigned one of nine labels to each labeled wafer as shown below:



In reality, there are far more than nine possible wafer shadings that wafers can have in real manufacturing settings. Regardless of how many classes there should be, the WM-811K dataset is highly class-imbalanced since the data comes from real fabs. Below is a breakdown of classes in the 54,355 “training” wafer maps that one of our source papers further split into training/test:

TABLE I: WM-811K dataset statistics.



Wafer map patterns	Training	Testing
None	33051	3679
Center	3113	349
Donut	372	37
Edge-Loc	2150	267
Edge-Ring	7735	819
Loc	1458	162
Random	546	63
Scratch	446	54
Near-full	49	5
Total	48920	5435

The vast majority of wafers have a failure shading of “none.” This is true of both the 54,355 “training” and 118,595 “test” wafer maps. One idea is to use all of the “training” wafers, and all of the non- “none” shaded “test” wafers for this project. We want to learn strong visual representations of a variety of failure shadings regardless of “class.”

Unfortunately, there aren’t any large, high-quality datasets of *multi-bin* wafer maps that are publicly available. Faris works in the semiconductor industry and has access to multi-bin wafer maps, but these can’t be shared outside of work. A project of this kind applied to multi-bin wafer maps would be a first (at least in terms of what you can find in academic papers; it’s quite possible TSMC or Intel have done something like this already in-house). We hope that the approach we take for this project can extend from binary to multi-bin wafer maps, as it would make Faris very happy → 🙂.

5. Research Papers

Please see the upload and comment on Canvas for the list of papers we plan to read. Below is just a starting point.

Deep learning specific:

- **FastSiam**
- SimSiam
- Pages 2-4 from SPICE (nice background on what people have done to "cluster" visual representations of images in the past and why people use contrastive self-supervised methods now) https://arxiv.org/pdf/2103.09382.pdf

Domain-specific:

- **SimCLR on wafer maps**: Semi-supervised Wafer Map Pattern Recognition using Domain-Specific Data Augmentation and Contrastive Learning
- Absolutely ancient method from data source paper: (don't read this but we need to cite) Wafer Map Failure Pattern Recognition and Similarity Ranking for Large-Scale Data Set