MINiature Interactive Offset Networks (MINIONs) for Wafer Map Classification

Yueling (Jenny) Zeng, Li-C. Wang, Chuanhe (Jay) Shan
University of California, Santa Barbara
Santa Barbara, California 93106

***Abstract*—We present a novel approach called MINiature Interactive Offset Networks (or MINIONs). We use wafer map classification as an application example. A Minion is trained with a specially-designed one-shot learning scheme. A collection of Minions can be used to patch a master model. Experiment results are provided to explain the potential areas Minions can help and their unique benefits.**

1. Introduction

For multi-class image classification, there are two common approaches as depicted in Figure 1. With a traditional Machine Learning (ML) approach, one first decides on a set of *features*. With the features defined, input samples are converted into *feature vectors* "$v_1, v_2, \cdots$" as shown in Figure 1. Then, a model building method (e.g. SVM [1]) is applied to learn a model based on the feature vectors.

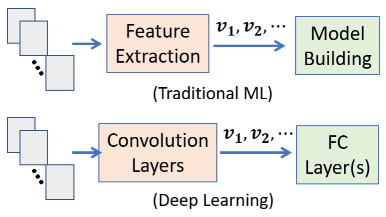


Figure 1. Two Basic ML Approaches

With a Deep Learning approach, the entire feature extraction step is automated, for example by multiple Convolution Neural Network (CNN) layers. The output of the last CNN layer can also be seen as a feature vector. Then, more network layers (e.g. Fully-Connected layers) can follow, to implement the model building step.

Take wafer map classification as an application example. One can observe these two common approaches through the works published over recent years. For example, the authors in [2] developed a comprehensive wafer map dataset called WM-811K (811,457 wafer maps where 172,950 with labels and the rest without label), and followed a traditional ML approach to build a wafer map pattern recognition and similarity ranking system. The work employed several types of features, including those based on Radon transform and those based on analyzing geometric properties (e.g. failing die counts, region labeling, line detection, etc.). Support Vector Machine (SVM) [1] was used for the model building step. The work reported an overall 94.63% accuracy on the dataset, comparing to the deep learning approach at the time which achieved 89.64% accuracy.

A number of works later also used the WM-811K dataset. For example, the work in [3] followed the same approach with a slightly different feature set and aimed to improve multi-pattern detection accuracy. The work in [4] advocated using more discriminant features based on Linear Discriminant Analysis (LDA) to simplify the model building step. With Radon transform based features, the work in [5] proposed a special Decision Tree based ensemble learning method. Decision tree models are generally more interpretable than SVM models.

To help deep learning on WM-811K, the author in [6] proposed using an Autoencoder (AE) as a pre-training step to learn features before learning a classifier. However, the author found that pre-selected features were still needed such that feature vectors, instead of the wafer map images themselves, were fed into the AE.

To apply deep learning, the author in [7] proposed a 2-stage classification process: first to classify between having a pattern and having no pattern (the later is called the "None" class in WM-811K), and if there is a pattern, classify which class it is. Note that both works [6] and [7] acknowledged that the wafer map images were noisy and a de-noising step was required to make the learning work.

WM-811K is a highly imbalanced dataset where some classes have many more samples than others [2]. The work in [8] proposed a special data augmentation method based on GAN (Generative Adversarial Network) [9]. Instead of using GAN, the authors in [10] used AE [11] for data augmentation. Moreover, the authors found that augmenting the samples with rotation could help. In contrast, the authors in [12] used pre-determined methods to augment the dataset and a deeper CNN for training the classifier.

The authors in [13] approached WM-811K dataset from a different angle. They tried to address the concern that a wafer map to be predicted might contain a new pattern not seen in the dataset, or contain a multi-pattern. As a result, the classifier's prediction might not be reliable. The work proposed using Selective Learning to estimate confidence of a prediction. Then, the deep learning model could include the choice to abstain from making a prediction.

In view of Figure 1, the earlier works [2][3][4][5] follow a traditional ML approach. The work [6] makes a transition to a deep learning approach. The later works [7][8][10][12][13] follow a deep learning approach where data augmentation and sample de-noising are two helpful steps.

It is interesting to observe that most of the previous works on wafer map classification were published in the semiconductor manufacturing field. Indeed, the problem has been studied in that field for decades. The works discussed above are only those based on the WM-811K dataset.

The problem attracts the interest of a fabless company because classifying failure patterns can help facilitate communication with the foundry, i.e. asking more tangible questions. This is particularly needed when unusual defectivity is observed with an advanced technology node.

1.1. Premises of this work

Following the research line reviewed above, a natural research path is to improve on a deep learning approach, e.g. dealing with the dataset imbalance problem or the difficulty among learning different classes. This is not the direction followed by this work though. Instead, we follow a path complementary to the deep learning direction.

With a deep learning approach, training a multi-class classifier essentially comprises three steps:

1) Split a dataset into set $\mathcal{T}$ and set $\mathcal{V}$.
2) Train a classifier with $\mathcal{T}$.
3) Evaluate (and validate) the classifier with $\mathcal{V}$.

We can call this an "end-to-end" training approach: Once the dataset is prepared, network architecture is determined, and training parameters are selected, a model is trained and validated in terms of its classification accuracy.

Our work is based on two premises. First, it is difficult, if not impossible, to obtain an end-to-end model where the "source" of its prediction is traceable (and interpretable). For example, suppose we desire to know why the label is predicted for a particular sample. It is difficult to make such a query and get an easily understandable answer. Our second premise is that there can be ambiguity in the labels provided in the dataset. For example, two very similar wafer maps are labeled as two different classes. Such ambiguity limits the accuracy achievable by an end-to-end model.

1.2. The master model and the Minions

In view of Figure 1, regardless of which approach is taken, the end result is a model for performing the classification task. We call such a model a *master*. The works reviewed above all try to optimize a master model, to improve its prediction accuracy and/or to add a capability it previously not having. With the two premises, our view is that a master model is not going to be ideal in every aspect. It will need some help. Hence in this work, we propose a novel approach called MINiature Interactive Offset Networks (MINIONS) that can be used to help a master.

For example, suppose when a master model is applied, it reports a prediction deemed by a person as a mistake. Instead of re-training the model, we envision that we can use a Minion to memorize the mistake to avoid a similar mistake in the future. With many Minions, many mistakes can be avoided. In essence, Minions can serve as "patches" to a master, or in Minions' own term, to *serve* a master.

For the rest of the paper, Section 2 reviews the problem of wafer map classification and our previous works. Section 3 discusses *one-shot learning* in view of Minion training and the various methods proposed for one-shot learning. Section 4 presents result from our selected master models. Section 5 reviews *manifestation learning* [14] as our baseline for training Minions, presents a novel two-part training approach, and discusses potential methods for improving the training. Section 6 demonstrates the two areas Minions can help a master: filtering out wafer maps containing no pattern and fixing a classification mistake. Section 7 discusses using Minions by themselves to perform unsupervised analysis on wafer batches. Section 8 concludes.

2. Wafer Map Pattern Recognition

A wafer map is an image of wafer where each die location is marked with a value, such as a binary value indicating pass/fail, or a numerical value indicating a measurement result. Analysis of wafer maps is a common practice for investigating a potential yield issue.

The WM-811K dataset categorizes wafer map patterns into 8 class. Figure 2 illustrates the 8 pattern classes (where yellow dots indicate fails). In addition, a "None" class is used to denote those wafer maps containing no pattern.

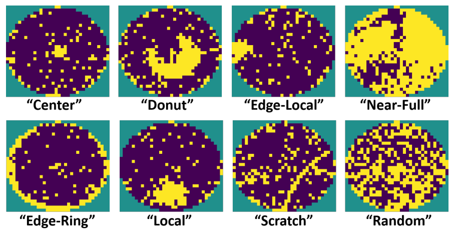


Figure 2. Eight pattern classes in WM-811K dataset

Division of such classes was done manually, and could be subjective. For example, Figure 3 shows different examples for the 8 classes. It is possible that a different person might perceive those examples with a different label. Also observe that the "Edge-Local" and the "Edge-Ring" examples look very similar, indicating potential label ambiguity. Figure 3 illustrates that labels provided in a dataset for training might not always be clear and consistent.

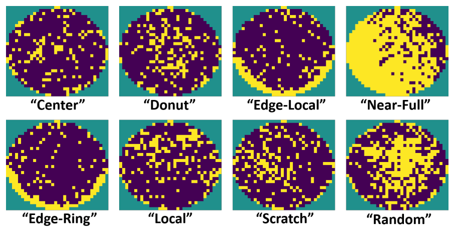


Figure 3. Additional examples assigned with the given label

Instead of using one model to classify all classes, our earlier solution [15] proposes training one *recognizer* for every class independently. This can avoid re-training an entire model when an adjustment on one class is needed. Then, following the work in [15], the work in [16] focuses on issues related to the deployment. Tensor computation based techniques [17][18][19] are introduced for two purposes: (1) to automatically decide a new class to learn and extract the corresponding training samples, and (2) to provide a verification method to validate the recognition result of a neural network recognizer.

The first capability is useful when a large batch of unlabeled wafer maps are processed for learning a set of initial recognizers. The second capability adds a safe guard and improves robustness of the classification. The reason for needing such a safe guard is because training a recognizer in [15][16] is based on a GAN setting [9][20][21] and the training could be tricky [20][21], i.e. the robustness of a GAN-based recognizer could be of concern [16].

3. The Minions

Instead of training one model for every class, in this work we train one Minion model for one given wafer map. For a Minion, we call the wafer map its *anchor*.

Figure 4 uses a scenario to illustrate one motivation to consider the Minion approach. From Figure 3, we see that there is an ambiguity between "Edge-Local" and "Edge-Ring". Suppose in application the master model classifies an unlabeled wafer map as "Edge-Local" as shown in Figure 4.

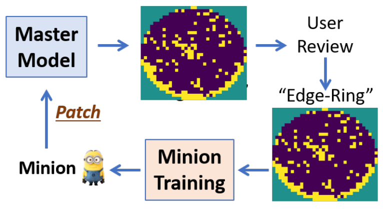


Figure 4. Traing a Minion to help a master

Suppose a user reviews the result and decides that the label should be "Edge-Ring". To adjust the prediction for future similar wafer maps, we train a Minion based on the wafer map to be adjusted on. Then, the Minion is used as a *patch* to the master model. The patch can work as the following: For a future wafer map w classified by the master as "Edge-Local, if w is also recognized by the Minion, then it is moved from "Edge-Local" to "Edge-Ring". In other words, we use the Minion to memorize the move.

Relative to the master they serve, Minions can be a much simpler model (**MIN**iature). A Minion might be learned through interacting with a user (**I**nteractive). A Minion performs a tweak to its master's prediction when needed (**O**ffset). Moreover, a Minion is a neural network (**N**etwork).

3.1. One-shot learning

Minion is trained with one sample. Training with one sample is generally referred as *one-shot learning* [22][23]. There can be two different setups: adjusting an existing model with one sample and training a new model with one sample. Our Minion training follows the second setup.

One-shot learning has been considered for many problem contexts, e.g. [24][25][26], just to name a few. The work in [24] considered one-shot learning in the drug discovery context. The work in [25] considered a recommendation system (e.g. Netflix) when a new user or a new class was presented. The term "cold-start" was used to describe the context. The work in [26] considered multiple application contexts such as hand-written character recognition and language translation. The focus was on implementing a memory module to remember rare training samples (one-shot learning) and as a result, to achieve *life-long learning*.

Comparing to [24], our problem formulation is simpler. While our one-shot learning focuses on recognizing a new sub-class, in [24] the challenge is on predicting the behavior of a molecule in a new experimental system. Similar to [25], Minion learning is considered as a cold-start, i.e. we prefer not to use other labeled samples. However, instead of taking a *meta-learning* approach like [25] to overcome the cold-start difficulty, we follow an approach called Manifestation Learning proposed in [14]. Unlike [26] where a unified memory module is used to remember rare samples, our approach remembers through each sample with its own dedicated Minion. Training a memory module is end-to-end. The advantage of our approach is traceability, i.e. it allows tracking the effect from every Minion.

3.2. Methods for One-Shot Learning

There are three sets of methods proposed to help one-shot learning. Below we provide a quick review.

3.2.1. Data augmentation. For one-shot (or few-shot) learning, the fundamental issue is labeled sample scarcity. Hence, an intuitive idea is to generate new labeled samples to augment the dataset. One of the earliest papers [27] proposed learning a set of geometric transformations to augment the training set. More recently, the work in [28] used Autoencoder [11] to learn intra-class variations and then applied the variations to synthesize new samples. The idea of learning sample variations was also employed in [29]. A single function was learned to answer an analogy question: "$z_1 : z_2 :: x :?$" where z_1, z_2 were feature vectors of two existing samples, and x was the feature vector of a sample from the new class to be augmented.

In *feature augmentation* (e.g. [29]), generated samples are in the feature space rather than in the input space. In addition to [29], the work in [30] proposed to model so-called feature trajectories induced by variations of object poses. The work in [31] employed a novel GAN design for feature augmentation. Instead of using GAN, the work in [32] employed Variational Autoencoder (VAE) [33].

The works reviewed above are mostly for processing "natural" images (same-class images have a high degree of visual similarity, e.g. images from the popular ImageNet dataset, as opposed to machine-generated or artificially-created images like wafer maps), except for [27] which focused on hand-writing characters. For processing images of characters, the work in [34] considered one-shot generation, i.e. generating different writings of a given character. The work was motivated by the challenge proposed in [35]. Worth noting, the approach in [34] employed a network component called *spatial transformer* which learned invariant with respect to four types of affine transformations [36].

3.2.2. Transfer learning. Another popular idea to overcome sample scarcity is *transfer learning* [37]. Transfer learning utilizes "knowledge" learned from a source domain, to facilitate learning on a target domain (where training data is scarce). *Feature transfer* [38] is a common approach where portion of a learned neural network from the source is re-used in training for the target. A special context of transfer learning is called *domain adaptation* [39] where knowledge is transferred between two different domains for performing the same task. Domain-Adversarial Training (DAT) [40] is a way for domain adaptation. In DAT, adversarial training is used to remove domain-specific effect in the feature space so that same-class samples from two domains (e.g. black/white images vs. color images) are mapped to similar feature vectors, i.e. the feature space trained with one domain can be re-used for the other domain.

3.2.3. Embedding learning. Embedding learning such as Siamese Network [41], Matching Network [42], Prototypical Network [43], and Relation Network [44], conceptually uses two functions f, g to map a test sample x_{test} and a training sample x_{train} to their respective *embedding vectors*. A similarity (e.g. cosine similarity) is checked between two embedding vectors to determine if they are in the same class or not. Task-invariant embedding learning can be considered as a form of *meta-learning* [45].

3.2.4. Methods considered. Among the three sets of methods, we considered data augmentation and feature transfer in our study. We did not consider meta-learning. This decision was largely influenced by the recent work [46] on cross-domain few-shot learning. Their findings include: (1) advanced meta-learning methods are out-performed by earlier meta-learning approaches, (2) all meta-learning methods are out-performed by feature transfer, and (3) performance of a method correlates to dataset similarity between the source and the target domains. Instead of meta-learning, we did consider one variant of Siamese Network called Triplet Loss Siamese Network (TLSN) [47], which inspired our two-part Minion training scheme presented in Section 5.

4. The Selected Master Models

In this work, we select the popular VGG-16 architecture [48] to train a master model.

4.1. The dataset and the training

The WM-811K dataset [2] comprises 9 classes with various wafer sizes. We select two sizes which have the largest number of wafer maps. Table 1 summarizes the selected set of labeled samples for our experiments. In addition, there are 19086 unlabeled wafer maps to be tried on.

TABLE 1. LABELED WAFER MAPS FROM WM-811K DATASET (2 SIZES)

Center	Donut	Edge-L	Edge-R	Loc	N-Full	Random	Scratch	None
81	10	402	9	345	16	28	76	22115

First, let us focus on the 8 classes and ignore the "None" class. The training is challenging because some classes have many fewer samples than others. We can use image rotation (as that commonly used for this dataset before, e.g. [6][7]) to balance the dataset. For example, we can add rotated samples to make every class comparable to the number of the "Edge-Local" class (i.e. 402). Then, we can follow a simple 2/3-1/3 split on the augmented dataset to obtain a training set and a validation set, respectively. With this setup, we can learn a model with training accuracy 99.24% and validation accuracy 92.5%. Applying this model onto the original set of 967 samples from the 8 classes (without rotation), the resulting *confusion matrix* is shown in Table 2.

TABLE 2. CONFUSION MATRIX (ON ALL 967 WAFER MAPS): $\Downarrow$: GIVEN LABEL, $\Rightarrow$: PREDICTED LABEL

	Center	Donut	Edge-L	Edge-R	Loc.	N-Full	Random	Scratch
Center	71	0	0	0	10	0	0	0
Donut	0	10	0	0	0	0	0	0
Edge-L	0	0	387	5	7	1	1	1
Edge-R	0	0	0	9	0	0	0	0
Loc.	3	1	16	0	319	0	1	5
N-Full	0	0	0	0	0	16	0	0
Random	0	1	1	0	0	0	26	0
Scratch	0	2	1	1	7	0	0	65

4.2. Manual review

In total, there are 64 mistakes shown in Table 2. With a manual review, they are categorized into three types:

- **Label Ambiguity**: Two very similar patterns are assigned with two different labels.
- **Underspecification**: The pattern is unique when comparing to other wafer maps in the same class.
- **Model Deficiency**: If none of above is applicable, it is put into this category

Among the 64 mistakes, we found 30 as label ambiguity and 3 as underspecification. The remaining 31 were left as model deficiency. Figure 5 shows several examples from the label ambiguity category (another example is the "Edge-Local" map shown in Figure 3). When training the model, if the dataset contains such cases of label ambiguity, they can be the causes for the accuracy loss.

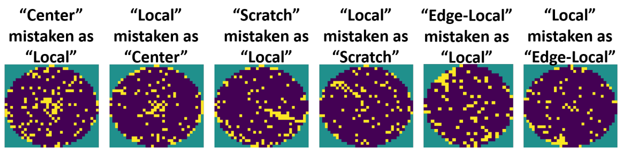


Figure 5. Examples of mistakes potentially caused by label ambiguity

Figure 6 shows three underspecification examples. Each of these patterns appears only on one wafer. If the wafer map were not in the training set, the model would never see the pattern. Hence, it was expected that the model could make a mistake on the sample.

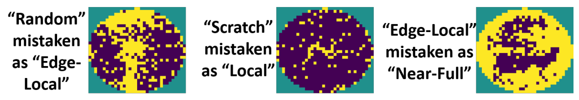


Figure 6. Examples of underspecification

4.3. Applied to unlabeled wafer maps

The 8-class VGG model cannot be applied to unlabeled wafer maps directly. This is because the model does not handle the "None" class. It assumes its input wafer map always has a pattern and then, its job is to classify the wafer map into one of the 8 classes. Therefore, before applying the model, we need another model serving as a frontend "filter". This filter's job is to screen out those wafer maps having no pattern, i.e. the "None" samples.

We trained a filter model based on the same VGG architecture and with a different training and validation set combination. The training set comprised the 967 samples from the 8 classes and all their rotated samples used above for training the VGG model. For the "None" class, 90K samples were put in the training set. The rest were put in the validation set. The validation set contained no sample from the 8 classes. In training, we looked for a model that achieved high accuracy on the training set and also screened out most of the "None" samples from the validation set. With this strategy, we obtained a filter model with training accuracy at 95.25%. This model screened out 99.33% of the "None" samples in the validation set.

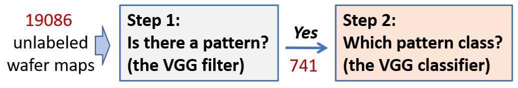


Figure 7. Two-step classification done by 2 master VGG models

To classify unlabeled wafer maps, a 2-step classification process is followed and depicted in Figure 7. After the first step, there are 741 wafer maps left (out of 19086). Then, Table 3 shows the classification result by the VGG classifier. Again, we manually reviewed the classification of each sample and identified those "questionable", i.e. those we thought they should not be in the class. The number of questionable samples are also shown in the table.

TABLE 3. CLASSIFICATION ON THE UNLABELED DATASET

Class	Center	Donut	Edge-L	Edge-R	Loc	N-Full	Random	Scratch
VGG classifier	114	37	211	44	233	9	34	59
Questionable	2	24	4	10	37	0	10	49

We can observe that although the VGG classifier has a high validation accuracy at 92.25%, its performance on the unlabeled set is not comparable. Figure 8 shows examples of questionable classification. In particular, its performance on "Donut" and "Scratch" is much worse. Its performance on other classes (e.g. "Loc") can also be doubtful.

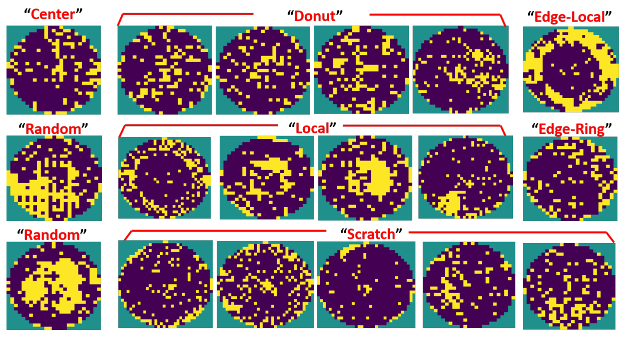


Figure 8. Examples of questionable classification

4.4. Issues with the two master VGG models

Result shown in Table 3 is by no means optimal. First, we did not have a mechanism to deal with the label ambiguity and underspecification issues discussed earlier. Furthermore, there are specific issues with the two VGG models as explained below.

For the filter model, its model selection is biased toward screening out the "None" samples. Although it can screen out 99.33% of the "None" samples, it also removes 249 wafer maps from the first 8 classes. Consequently, on the unlabeled set there should be more samples having a pattern than the 741 wafer maps being classified by the VGG classifier. We need a way to bring back those wafer maps mistakenly screened out by the VGG filter model.

For the VGG classifier, as shown before, it makes 64 mistakes on the 967 labeled wafer maps having a pattern. These mistakes are known to us according to their labels. These mistakes could be the causes for some of the mistakes seen on the unlabeled set. Hence, we need a way to fix these mistakes before applying to the unlabeled set.

5. Minions and Their Training

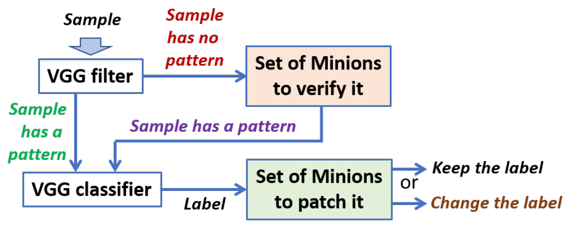


Figure 9. Minions for patching the two master models

We need two sets of Minions for patching the two master VGG models, as illustrated in Figure 9. The first set is to verify if a screened-out sample really has no pattern. As mentioned above, we know that the filter model mistakenly screen out 249 labeled samples. One Minion is trained for each of the 249 mistakes. Then in future screening, for every sample screened out by the filter model, if any of the 249 Minions considers the screening as a mistake, the screened sample will be restored and given back to the VGG classifier.

The second set of Minions is to check the label predicted by the VGG classifier. From its classification on the 967 labeled wafer maps, we know it makes 64 mistakes (see Table 2). Again, one Minion is trained for each mistake. In application, for example, to fix a mistake such that it should be in class X but the VGG classifier puts it in class Y, the Minion checks all samples predicted as class Y by the VGG classifier. If the Minion recognizes any sample, the sample is moved from class Y to class X.

Note that the two sets of mistakes mentioned above are not disjoint, because the 64 mistakes in Table 2 are based on all 967 wafer maps without the filtering. Further, with the 249 Minions patching the filter model, the Minions might classify some "None" samples as having a pattern. As a result, the VGG classifier should receive additional samples from the Minions, which might lead to more mistakes.

5.1. Manifestation Learning

For Minion training, *manifestation learning* [14] is our baseline. To start, first we pick a multi-class dataset, e.g. the MNIST dataset [49], and follow three steps:

1) Learn an encoder A (a CNN model) that maps each sample in the multi-class dataset onto a *code* (e.g. a 16-dimension vector) in a *code space*.

2) In the code space, we define a *concept region* (or *codebook*) for a selected class, e.g. digit “1”.
3) For the given in-class wafer map, learn an encoder B (a CNN model) that maps the sample to a code (e.g. the center) of the concept region.

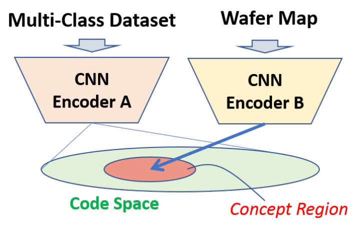


Figure 10. A recap of Manifestation Learning proposed in [14]

Moreover, a counter example can be generated based on the given wafer map. For a counter example, the training objective is to map it to a code outside the concept region.

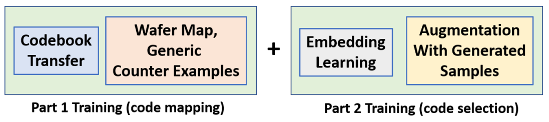
After training, encoder B is used as the recognizer for the wafer map. Samples mapped to a code inside the concept region are considered in-class. Otherwise, samples are considered unrecognized.

In the original setup [14], training encoder B was done by generating two generic counter examples. The encoder was trained with a VAE [50] setting. The concept region was defined based on the digit “1” class such that most of the “1” samples were included in the region while none of samples from other digits were included. Modeling of the concept region was based on one-class SVM [1]. Further implementation detail could be found in [14].

5.1.1. Concept region. For this work we built a concept region (using one-class SVM) that includes at least 99% of the “1” samples. The SVM model has 246 support vectors (out of 5000 samples). In a sense, the concept region contains (infinite) in-class codes and the outside contains (infinite) out-class codes, both defined by the SVM model. These codes can be seen as a *codebook*. In that sense, manifestation learning transfers the codebook learned from one domain (MNIST) to be reused for another domain (wafer map).

5.2. The Two-Part Training Approach

As pointed out in [14], for manifestation learning with one wafer map, counter examples can affect its learning [14]. To provide flexibility to supply more generated counter examples, we consider the *contrastive learning* idea presented in TLSN [47]. By adapting their idea with our codebook transfer idea, we came out with a novel two-part training approach as illustrated in Figure 11.



Codebook Transfer
Wafer Map, Generic Counter Examples
+
Embedding Learning
Augmentation With Generated Samples
Part 1 Training (code mapping)
Part 2 Training (code selection)

Figure 11. Illustration of the two-part training scheme

The first part is similar to the original manifestation learning. In the second part, we consider generating more in-class samples and more counter examples, using the given wafer map. For in-class samples, they are mapped to codes inside the concept region. For out-class samples, they are mapped to codes outside the concept region. We also constrain the codes to be away from the boundary of the concept region by a margin. The CNN is trained to simultaneously satisfy constraints from both parts.

5.3. Potential areas for improvement

Figure 12 illustrates four areas for potential improvement of Minion training: (1) image preprocessing, (2) strategy to generate in-class samples, (3) strategy to generate counter examples, and (4) feature transfer for the CNN layers.

5.3.1. Preprocessing. The WM-811K wafer images come with different sizes. Image resizing is a common step in the previous works. De-noising the images can also impact CNN training significantly (e.g. [7]). For preprocessing we therefore adopt both image resizing and de-noising. For denoising, we consider a *median filter* as that used in [7]. In addition, we also consider using region-labeling method to highlight the most salient region as that used in [2] for creating an in-class sample.

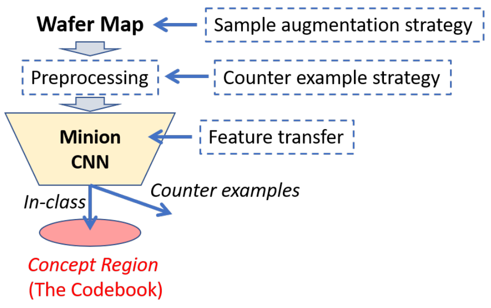


Figure 12. Possible areas to improve learning of encoder B

5.3.2. Augmenting the input sample. Rotation-invariant is a common consideration in pattern recognition based on WM-811K dataset [2]. Image rotation is therefore a straightforward method to help CNN training for capturing this invariant [6][7]. Originally, we considered generating more in-class samples using rotation. However, in our Minion approach, to take care of rotation invariant, we could also consider rotating the input image to be recognized. We found this method more robust than using rotated in-class samples in the training.

Another consideration is scale-invariant [2]. In this work, we do not involve scale-invariant rules for in-class sample generation. Instead, we study the possibility of using a pretrained model to capture variations and then apply the model to generate samples from the given wafer map. This learning can be achieved by the *spatial transformer* proposed in [36].

5.3.3. Using spatial transformer. In one study, we learned a spatial transformer to capture the variations from the given “1” anchor image to other “1” images in the concept region. We then applied the spatial transformer to transform the given anchor wafer map to new wafer maps. We found that, while some generated wafer maps could make sense, many did not. They looked more like a new pattern than a variation of the original pattern. This means that invariant learned on the digit “1” is not the same as the invariant we would like to capture for a given wafer map. Because not all transformed wafer maps make sense, a manual review is required to hand pick those to be included as proper in-class samples. This manual step makes the approach difficult to use in practice.

The same concern applies to using AE [11] (or VAE [33]) and GAN [9] for sample generation. Manual review is required to exclude generated samples that do not make sense, making them difficult to use in practice.

5.3.4. Generating counter examples. We consider generating random counter examples. Generating random wafer maps is straightforward. However, we take the yield loss on the anchor wafer map into account. Based on the yield loss, we randomly select locations to put failing dies. For the results reported in this work, the number of counter examples generated is fixed at 500.

5.3.5. Feature transfer. There can be three types of settings to adopt feature transfer to start the learning of encoder B in Figure 10: (1) Transfer the weights of CNN layer(s) from encoder A; (2) Train a VAE [33] with labeled and/or unlabeled samples and then transfer its CNN weights; (3) Transfer the weights of CNN layers from the master.

In our study, we found that the first type of setting did not help, indicating difficulty to adopt cross-domain feature transfer. The second type also did not improve Minion learning. Often, it significantly reduced the recognized set of a Minion, which was not a desirable effect. We observed a similar trend with the third type of transfer.

Overall, we did not find a setting where feature transfer could provide a clear benefit for Minion training. As a result, we did not involve feature transfer in Minion training.

6. Experiment Results

Continuing from the discussion before with Figure 9, Figure 13 shows the numbers of Minions trained for fixing the two sets of mistakes. As mentioned before, the first set of 249 Minions are for patching the VGG filter.

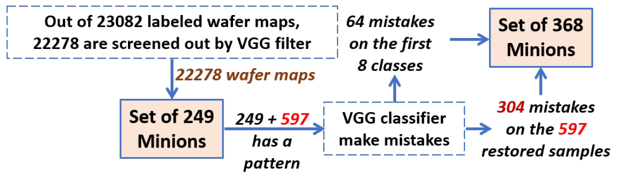

Out of 23082 labeled wafer maps, 22278 are screened out by VGG filter
22278 wafer maps
Set of 249 Minions
249 + 597 has a pattern
VGG classifier make mistakes
64 mistakes on the first 8 classes
Set of 368 Minions
304 mistakes on the 597 restored samples

Figure 13. Workflow view of the result on labeled wafer maps

6.1. Minions for patching the VGG filter

For the 23082 labeled wafer maps, the VGG filter screens out 22278 wafer maps which need to be verified by the 249 Minions. The 249 Minions guarantee to restore their anchor wafer map back. In addition, the 249 Minions capture 597 “None” samples as having a pattern.

We manually reviewed those 597 wafer maps to verify if any obviously had no pattern. Then, we applied the region-labeling method to measure a pattern size. We found that all of them had a pattern size greater than the smallest size from the wafer maps labeled as having a pattern. As a result, they were included in the input to the VGG classifier.

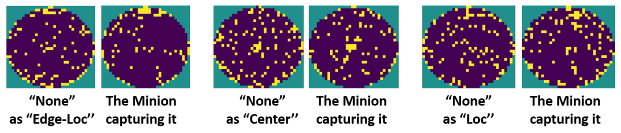


Figure 14. Examples of “None” considered having a pattern by Minions

For example, Figure 14 shows three smallest-size “None” samples captured by a Minion as having a pattern. Each sample receives a label from the corresponding Minion whose anchor is shown. Each Minion accurately captures a similar pattern even though the pattern is small.

Based on the Minions capturing them, each of the 597 “None” samples receive a new label. For a sample captured by multiple Minions whose labels are different, a dual or multi-class label is assigned. For example, if a sample is captured by a “Loc” Minion and an “Edge-Loc” Minion, then its label is “[Loc, Edge-Loc]”. In total, 72 of the 597 wafer maps receive a dual/multi-class label. For wafer maps with such a label, we do not consider them as a mistake if the VGG classifier correctly puts them into one of the classes.

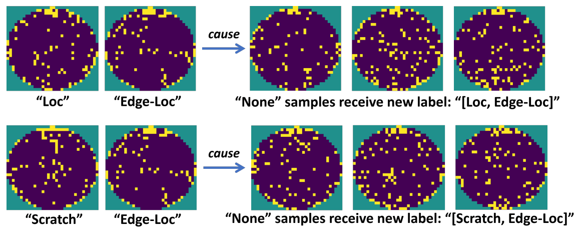

cause
“Loc”
“Edge-Loc”
“None” samples receive new label: “[Loc, Edge-Loc]”
cause
“Scratch”
“Edge-Loc”
“None” samples receive new label: “[Scratch, Edge-Loc]”

Figure 15. Pairs of Minions causing a dual label

Among the 72 wafer maps, many of them are caused by the two pairs of Minions shown in Figure 15. With the first pair, the label is “[Loc, Edge-Loc]”. The figure shows three “None” examples receiving this label. Similarly, with the second pair, the label is “[Scratch, Edge-Loc]”.

Figure 15 illustrates that with Minions, we are able to detect label ambiguity across samples. For example, as shown in the figure, although samples in each pair have different labels, they turn out to recognize each other. Hence, we can identify them as a case of label ambiguity.

6.2. Minions for patching VGG classification

For the VGG classifier, 64 Minions are trained for the 64 mistakes discussed before. Then, among the 597 “None” samples that receive new labels, the VGG classifier makes 304 mistakes. As a result, 304 additional Minions are trained. In total, the second set has 368 Minions.

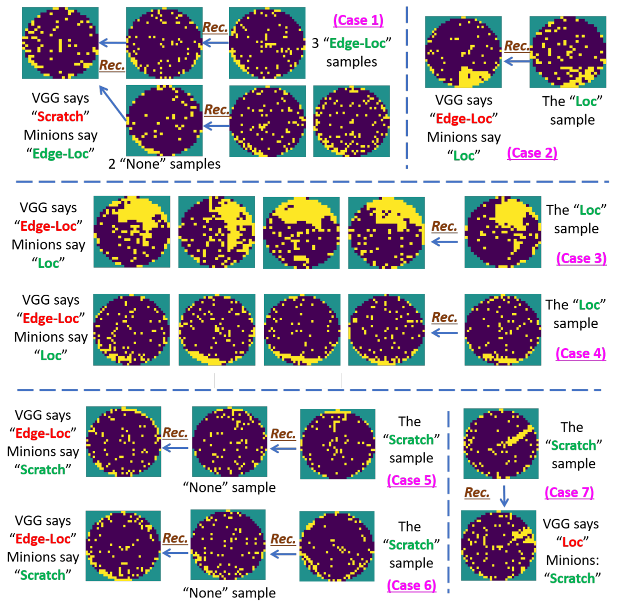
When applying these 368 Minions to the VGG classification result, every Minion guarantees moving its anchor to its own class. Collectively, they also move additional 146 wafer maps. Table 4 shows the number of wafers with its labels given by the VGG classifier and the Minions.

TABLE 4. Additional 146 moves done by Minions

VGG says	Edge-R	Scratch	Edge-L	Edge-L	Edge-L	Loc
Minions say	Edge-L		[Loc,Scratch]	Loc	Scratch	
#	5	9	16	102	13	1

We manually reviewed all 146 cases. We found that most of the disagreements involve label ambiguity. Figure 16 shows 7 interesting cases where their detailed recognition traces are included. Overall, we did not find an obvious case where the Minion made a wrong move.

The VGG labels are those originally given in the dataset. For each label given by the Minions, Minions allow us to trace back to find out who provides the label. For cases 1, 5, and 6, the trace goes beyond one level of recognition.



(Case 1)
3 "Edge-Loc" samples
Rec.
VGG says "Scratch" Minions say "Edge-Loc"
2 "None" samples
VGG says "Edge-Loc" Minions say "Loc"
The "Loc" sample
(Case 2)
VGG says "Edge-Loc" Minions say "Loc"
The "Loc" sample
(Case 3)
(Case 4)
VGG says "Edge-Loc" Minions say "Scratch"
"None" sample
The "Scratch" sample
(Case 5)
(Case 6)
(Case 7)
VGG says "Loc" Minions: "Scratch"

Figure 16. Seven disagreements and their reasons provided by Minions

Take case 1 as an example. The disagreement is between "Scratch" by VGG and "Edge-Loc" by Minions. The "Edge-Loc" label is due to two Minions whose anchors are originally labeled as "None". They receive the "Edge-Loc" label because they are captured by three Minions whose labels are all "Edge-Loc". Hence, five Minions directly and indirectly classify the wafer map as an "Edge-Loc". The example shows that all wafer maps supporting this classification are traceable. This traceability is one major benefit with the Minions. In all 7 cases, one cannot say for certain that a disagreement is due to a wrong move by the Minion(s).

6.3. Additional results on unlabeled wafer maps

Table 3 shows that originally, out of 19086 unlabeled samples, VGG classifier receives 741 samples from the VGG filter. Applying the same workflow in Figure 13, the remaining 18345 samples need to be verified by the 249 Minions. Figure 17 shows that the verification restores 649 samples and considers them as having a pattern. As a result, these 649 samples, together with the original 741 samples, are both given to the VGG classifier. After VGG classification, Figure 17 further shows that the 368 Minions change the label on 316 wafers in total.

Table 5 summarizes the moves by Minions from each class claimed by VGG classifier. Note that not all 8 classes are shown because for some, Minions are not applicable.

We manually reviewed all 316 moves by Minions, and did not find any unreasonable. Figure 18 shows 9 cases and their traces, similar to those shown in Figure 16 before. As seen, all moves made by Minions make sense.

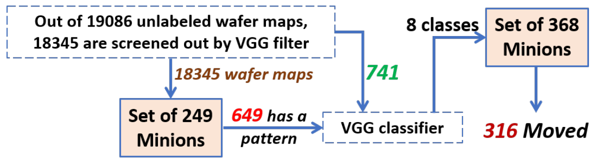
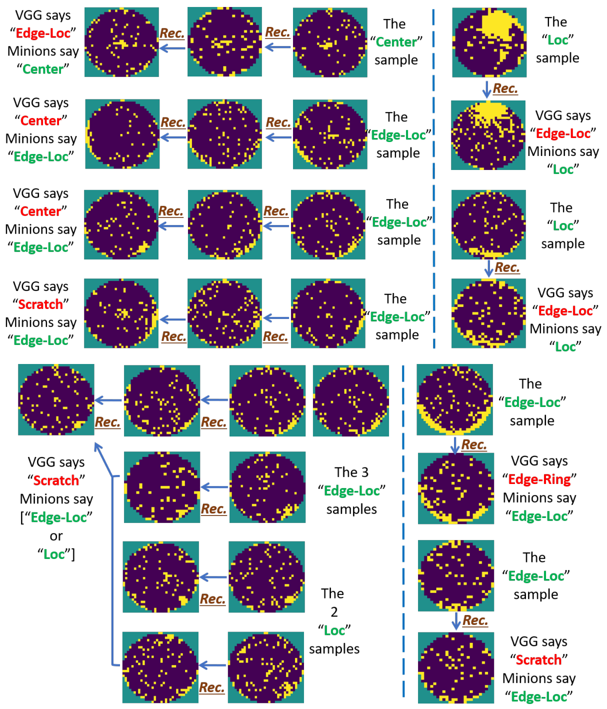


Figure 17. Summary result on the 19086 unlabeled samples

TABLE 5. IN TOTAL 316 WAFERS MOVED BY 368 MINIONS

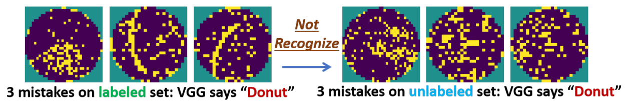
VGG says	Center	Edge-Loc	Edge-Ring	Loc	Scratch
Total # claimed by VGG:	154	499	58	275	324
# moved by Minions:	4	120	8	13	171



VGG says "Edge-Loc" Minions say "Center"
The "Center" sample
VGG says "Center" Minions say "Edge-Loc"
The "Edge-Loc" sample
VGG says "Scratch" Minions say "Edge-Loc"
The "Loc" sample
VGG says "Edge-Loc" Minions say "Loc"
Rec.
VGG says "Scratch" Minions say ["Edge-Loc" or "Loc"]
The 3 "Edge-Loc" samples
The 2 "Loc" samples
The "Edge-Loc" sample
VGG says "Edge-Ring" Minions say "Edge-Loc"
VGG says "Scratch" Minions say "Edge-Loc"

Figure 18. Example moves by Minions and their traces

Note that not all questionable cases on the unlabeled samples discussed with Table 3 (and shown in Figure 8) are fixable by Minions. Figure 19 shows three such examples for the "Donut" class. On the labeled samples, VGG's mistakes include the 3 samples shown, resulting in 3 Minions trained.



Not Recognize
3 mistakes on labeled set: VGG says "Donut"
3 mistakes on unlabeled set: VGG says "Donut"

Figure 19. VGG mistakes on labeled/unlabeled samples for "Donut" class

On the unlabeled samples, 3 VGG's mistakes are also shown, each with a pattern looking quite differently from the three labeled samples. As a result, the 3 Minions do not recognize any of the unlabeled samples and hence, none of the mistakes can be fixed by the 3 Minions. Cases like these can happen because Minions are only for fixing mistaken patterns that have been seen before. If a mistake involved a new pattern, it would not be fixed by Minions.

7. Unsupervised Batch Analysis

Minions were meant to serve a master. However, in some scenarios our Minions can operate effectively by themselves. Consider a scenario where batches of wafers are coming in. On each batch, one desires to answer two questions: (1) if there is a *systematic pattern* on this batch, and (2) if yes, whether or not the pattern occurred previously.

For the scenario, starting with a multi-class classification is unnecessary. With Minions, we can apply an unsupervised analysis on each batch to find systematic patterns. Once a pattern is found, Minions can also help search for occurrences of the pattern in a set of past wafer maps.

Because a Minion is specific to an anchor wafer map, if we want to check if two wafer maps have the same pattern, we can simply apply one Minion to recognize the other Minion and vice versa. Each pair of Minions can establish their relationship in this way. With n Minions, we can use their relationships to extract a cluster. For example, Figure 20 shows such a cluster extracted from 4 lots of wafers. The cluster initially contains 10 unlabeled wafer maps. Each red edge indicates a relationship via mutual recognition. The cluster represents a systematic pattern discovered by the unsupervised analysis.

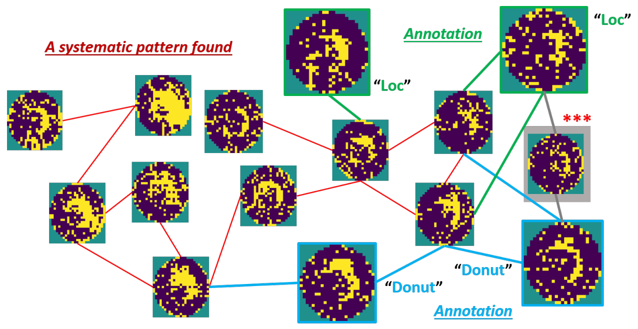


Figure 20. An example result from Minion-based unsupervised analysis

For answering the 2nd question, let us assume that the set of the labeled wafer maps is the source to search for previous occurrences. Using the Minions trained on the unlabeled wafer maps and the Minions trained on the labeled wafer maps, we can establish their relationships as well. The figure shows that 3 of the 10 unlabeled wafer maps are related to two "Loc" wafer maps and their relationships are marked by three green edges. In addition, two "Donut" wafer maps are related to 3 unlabeled wafer maps, and their relationships are marked by four blue edges. By checking the relationships between those 4 labeled wafer maps and other unlabeled wafer maps left in the 4 lots, we found one additional unlabeled wafer map, shown in the figure with a "***" mark. This 11th wafer is connected to the cluster due to indirect relationships through the labeled wafer maps.

With the result shown, answers to the two questions may go like this: (1) There are 11 wafer maps exhibiting a systematic pattern; (2) There is one previous occurrence which was named "Loc" and is represented by two wafers; (3) There is another previous occurrence which was named "Donut" and is represented by two wafers.

Notice that in the analysis, labels are used only to *annotate* the cluster. As a result, labels have no impact on the analysis. For the cluster, what label we choose to name it also has nothing to do with future analysis. The name will be used for annotation and issue tracking only. Consequently, with our Minion-based unsupervised analysis, label ambiguity is no longer an issue.

Furthermore, underspecification is less an issue as well. Suppose a pattern has no previous occurrence and only appear on a given batch. As long as the pattern is on two or more wafers, Minions may find it. Then, the search will simply report no previous occurrence found. Interestingly, for the two fundamental issues discussed lengthily before, both are not of great concern if only Minions are involved.

8. Conclusion

Minions are models each trained with one sample. We investigate various options for improving Minion training. Originally, Minions were meant for patching a master model. A master model can make mistakes due to its own deficiency, label ambiguity, and training set underspecification. Through experiments, we demonstrate that our Minions can effectively help a master and also provide traceability to detect cases of label ambiguity. Without a master, Minions can also operate effectively in an unsupervised way, when analyzing batches of wafers looking for systematic patterns. When Minions operate without their master, label ambiguity and underspecification are no longer of great concern. This benefit along with the traceability Minions can provide, make the Minion-based unsupervised analysis an attractive approach to be applied in practice.

Acknowledgment This work is supported in part by National Science Foundation Grant No. 2006739. The authors are thankful to Sergio Mier and Leon Wang of Qualcomm for their valuable inputs to our research.

References

[1] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* The MIT Press, 2001.

[2] M.-J. Wu, J.-S. R. Jang, and J.-L. Chen, "Wafer map failure pattern recognition and similarity ranking for large-scale data sets," *IEEE Transactions on Semiconductor Manufacturing*, vol. 28, no. 1, pp. 1–12, 2015.

[3] M. Fan, Q. Wang, and B. van der Waal, "Wafer defect patterns recognition based on optics and multi-label classification," *IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016.

[4] J. Yu and X. Lu, "Wafer map defect detection and recognition using joint local and nonlocal linear discriminant analysis," *IEEE Transactions on Semiconductor Manufacturing*, vol. 29, no. 1, pp. 33–43, 2016.

[5] M. Piao, C. H. Jin, J. Y. Lee, and J.-Y. Byun, "Decision tree ensemble-based wafer map failure pattern recognition based on radon transform-based features," *IEEE Transactions on Semiconductor Manufacturing*, vol. 31, no. 2, pp. 250–257, 2018.

[6] J. Yu, "Enhanced stacked denoising autoencoder-based feature learning for recognition of wafer map defects," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 613–624, 2019.

[7] N. Yu, Q. Xu, and H. Wang, "Wafer defect pattern recognition and analysis based on convolutional neural network," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 4, pp. 566–573, 2019.

[8] J. Wang, Z. Yang, J. Zhang, Q. Zhang, and W.-T. K. Chien, “Adabalgan: An improved generative adversarial network with imbalanced learning for wafer defective pattern recognition,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 310–319, 2019.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and J. Bengio, “Generative adversarial networks,” *arXiv:1406.2661*, 2014.

[10] T.-H. Tsai and Y.-C. Lee, “A light-weight neural network for wafer map classification based on data augmentation,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 4, pp. 663–672, 2020.

[11] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986. [Online]. Available: https://doi.org/10.21105

[12] M. Saqlain, Q. Abbas, and J. Y. Lee, “A deep convolutional neural network for wafer defect identification on an imbalanced dataset in semiconductor manufacturing processes,” *IEEE Transactions on Semiconductor Manufacturing*, vol. 33, no. 3, pp. 436–444, 2020.

[13] M. B. Alawieh, D. Boning, and D. Z. Pan, “Wafer map defect patterns classification using deep selective learning,” *ACM/IEEE Design Automation Conference*, 2020.

[14] Y. J. Zeng, L.-C. Wang, C. J. Shan, and N. Sumikawa, “Learning a wafer feature with one training sample,” in *IEEE International Test Conferencel*. IEEE, 2020, pp. 1–10.

[15] M. Nero, J. Shan, L.-C. Wang, and N. Sumikawa, “Concept recognition in production yield data analytics,” *IEEE International Test Conference*, 2018.

[16] C. Shan, A. Wahba, L.-C. Wang, and N. Sumikawa, “Deploying a machine learning solution as a surrogate,” in *IEEE International Test Conferencel*. IEEE, 2019, pp. 1–10.

[17] A. Wahba, L.-C. Wang, Z. Zhang, and N. Sumikawa, “Wafer pattern recognition using tucker decomposition,” in *VLSI Test Symposium (VTS), 2019 IEEE 37th*. IEEE, 2019, pp. 1–6.

[18] A. Wahba, J. Shan, L.-C. Wang, and N. Sumikawa, “Wafer plot classification using neural networks and tensor methods,” in *ITC-Asia*. IEEE, 2019, pp. 79–84.

[19] A. Wahba, C. Shan, L.-C. Wang, and N. Sumikawa, “Measuring the complexity of learning in concept recognition,” in *Int. Symposium on VLSI Design, Automation and Test*. IEEE, 2019, pp. 1–4.

[20] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training GANs,” *arXiv:1606.03498v1*, 2016.

[21] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv:1511.06434v2*, 2016.

[22] F.-F. Li, R. Fergus, and P. Perona, “One-shot learning of object categories,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[23] M. Fink, “Object classification from a single example utilizing class relevance metrics,” *Advances in Neural Information Processing Systems*, pp. 449–456, 2005.

[24] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande, “Low data drug discovery with one-shot learning,” *Central Science*, vol. 3, no. 4, pp. 283–293, 2017.

[25] M. Vartak, A. Thiagarajan, C. Miranda, J. Bratman, and H. Larochelle, “A meta-learning perspective on cold-start recommendations for items,” *Advances in Neural Information Processing Systems*, pp. 6904–6914, 2017.

[26] L. Kaiser, O. Nachum, A. Roy, and S. Bengio, “Learning to remember rare events,” *International Conference on Learning Representations*, pp. 1–10, 2017.

[27] E. G. Miller, N. E. Matsakis, and P. A. Viola, “Learning from one example through shared densities on transforms,” *Conference on Computer Vision and Pattern Recognition*, pp. 464–471, 2000.

[28] E. Schwartz, L. Karlinsky, J. Shtok, and e. a. Harary, “Deltaencoder: An effective sample synthesis method for few-shot object recognition,” *Advances in NIPS*, pp. 2850–2860, 2018.

[29] B. Hariharan and R. Girshick, “Low-shot visual recognition by shrinking and hallucinating features,” *International Conference on Computer Vision*, 2017.

[30] B. Liu, X. Wang, M. Dixit, R. Kwitt, and N. Vasconcelos, “Feature space transfer for data augmentation,” *Conference on Computer Vision and Pattern Recognition*, p. 9090–9098, 2018.

[31] H. Gao, Z. Shou, A. Zareian, H. Zhang, and S. Chang, “Low-shot learning via covariance-preserving adversarial augmentation networks,” *Advances in Neural Information Processing Systems*, pp. 983–993, 2018.

[32] Z. Cheny, Y. Fuy, Y. Zhang, and e. a. Jiang, “Multi-level semantic feature augmentation for one-shot learning,” *IEEE Transactions on Image Processing*, vol. 28, no. 9, pp. 4594–4605, 2019.

[33] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[34] D. Rezende, I. Danihelka, K. Gregor, and D. Wierstra, “One-shot generalization in deep generative models,” *International Conference on Machine Learning*, pp. 1521–1529, 2016.

[35] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[36] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, “Spatial transformer networks,” *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[37] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 22, pp. 1345–1359, 2010.

[38] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” *Advances in Neural Information Processing Systems*, vol. 2, pp. 3320–3328, 2014.

[39] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, “Analysis of representations for domain adaptation,” *Advances in Neural Information Processing Systems*, no. 22, pp. 137–144, 2007.

[40] Y. Ganin, E. Ustinova, H. Ajakan, and e. a. Germain, “Domain-adversarial training of neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1–35, 2016.

[41] G. Koch, R. Zemel, and R. Salakhutdinov, “Siamese neural networks for one-shot image recognition,” *ICML deep learning workshop*, 2015.

[42] O. Vinyals, C. Blundell, D. T. Lillicrap, and e. a. Wierstra, “Matching networks for one shot learning,” *Advances in Neural Information Processing Systems*, pp. 363–368, 2016.

[43] J. Snell, K. Swersky, and R. S. Zemel, “Matching networks for one shot learning,” *Advances in Neural Information Processing Systems*, pp. 4077–4087, 2017.

[44] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, “Matching networks for one shot learning,” *Conference on Computer Vision and Pattern Recognition*, pp. 1199–1208, 2018.

[45] S. Hochreiter, A. S. Younger, and P. R. Conwell., “Learning to learn using gradient descent,” *International Conference on Artificial Neural Networks*, pp. 87–94, 2001.

[46] Y. Guo, N. C. Codella, L. Karlinsky, and e. a. Codella, “A broader study of cross-domain few-shot learning,” *A. Vedaldi et al. (Eds.): ECCV 2020, LNCS, Springer Nature Switzerland AG 2020*, vol. 12372, pp. 124–141, 2020.

[47] X. Dong and J. Shen, “Triplet loss in siamese network for object tracking,” in *European Conference on Computer Vision (ECCV)*, September 2018.

[48] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015.

[49] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[50] S. Zhao, J. Song, and S. Ermon, “Infovae: Information maximizing variational autoencoders,” 2017. [Online]. Available: https://arxiv.org/abs/1706.02262