Inspector Gadget: A Data Programming-based Labeling System for Industrial Images [Scalable Data Science]

Geon Heo, Yuji Roh, Seonghyeon Hwang, Dayun Lee, Steven Euijong Whang
KAIST
{geon.heo, yuji.roh, sh.hwang, dayun.lee, swhang}@kaist.ac.kr

ABSTRACT

As machine learning for images becomes democratized in the Software 2.0 era, one of the serious bottlenecks is securing enough labeled data for training. This problem is especially critical in a manufacturing setting where smart factories rely on machine learning for product quality control by analyzing industrial images. Such images are typically large and may only need to be partially analyzed where only a small portion is problematic (e.g., identifying defects on a surface). Since manual labeling these images is expensive, weak supervision is an attractive alternative where the idea is to generate weak labels that are not perfect, but can be produced at scale. Data programming is a recent paradigm in this category where it uses human knowledge in the form of labeling functions and combines them into a generative model. Data programming has been successful in applications based on text or structured data and can also be applied to images usually if one can find a way to convert them into structured data. In this work, we expand the horizon of data programming by directly applying it to images without this conversion, which is a common scenario for industrial applications. We propose Inspector Gadget, an image labeling system that combines crowdsourcing, data augmentation, and data programming to produce weak labels at scale for image classification. We perform experiments on real industrial image datasets and show that Inspector Gadget obtains better accuracy than state-of-the-art techniques: Snuba, GOGGLES, and self-learning baselines using convolutional neural networks (CNNs) without pre-training.

1. INTRODUCTION

In the era of Software 2.0, machine learning techniques for images are becoming democratized where the applications range from manufacturing to medical. For example, smart factories regularly use computer vision techniques to classify defective and non-defective product images [22]. In medical applications, MRI scans are analyzed to identify diseases like cancer [20]. However, many companies are still reluctant to adapt machine learning due to the lack of labeled data where manual labeling is simply too expensive [31].

We focus on the problem of scalable image labeling for classification where *large* images are *partially analyzed*, and there are *few or no labels* to start with. Although many companies face this problem, it has not been studied enough. Based on a collaboration with a large manufacturing company, we provide the following running example. Suppose there is a smart factory application where product images are analyzed for quality control (Figure 1). These images

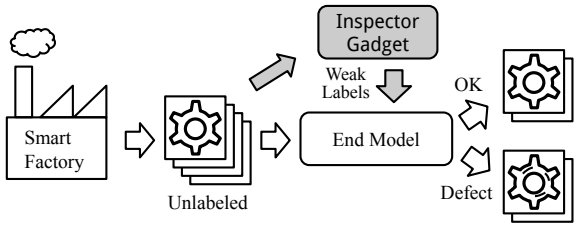


Figure 1: Labeling industrial images with Inspector Gadget in a smart factory application.

are usually taken from industrial cameras and have high-resolution. The goal is to look at each image and tell if there are certain defects (e.g., identify scratches, bubbles, and stampings). For convenience, we hereafter use the term *defect* to mean a part of an image of interest.

A conventional solution is to collect enough labels manually and train say a convolutional neural network on the training data. However, fully relying on crowdsourcing for image labeling can be too expensive. In our application, we have heard of domain experts demanding six-figure salaries, which makes it infeasible to simply ask them to label images. In addition, relying on general crowdsourcing platforms like Amazon Mechanical Turk may not guarantee high-enough labeling quality.

Among the possible methods for data labeling (see an extensive survey [26]), weak supervision is an important branch of research where the idea is to semi-automatically collect labels that are not perfect like manual ones (thus called "weak"), but have reasonable quality where the quantity compensates for the quality. Data programming [25] is a representative technique of employing humans to develop labeling functions that individually perform labeling, perhaps not accurately. However, the combination of labeling functions into a generative model results in a reasonable labeling program. These weak labels can then be used to train an end discriminative model.

So far, data programming has been shown to be effective in finding various relationships in text and structured data [24]. Data programming has also been successfully applied to images where they are usually converted to structured data beforehand [34, 36]. However, this conversion limits the applicability of data programming. As an alternative to data programming, the recently proposed GOGGLES [9] demonstrates that, on images, automatic approaches using pre-trained models may be more effective. Here the idea is to extract semantic prototypes of images using the pre-trained model and then cluster and label the images

using the prototypes. However, GOGGLES also has limitations (see Section 6.2), and it is not clear if it is the only solution for generating training data for image classification.

We thus propose Inspector Gadget, which opens up a new class of problems for data programming by enabling direct image labeling at scale without the need to convert to structured data using a combination of crowdsourcing, data augmentation, and data programming techniques. Inspector Gadget provides a crowdsourcing workflow where workers identify *patterns* that indicate defects. Here we make the tasks easy enough for non-experts to contribute. These patterns are augmented using general adversarial networks (GANs) [12] and policies [7]. Each pattern effectively becomes a labeling function by being matched with other images. The similarities are then used as features to train a multi-layer perceptron (MLP), which generates weak labels.

In our experiments, Inspector Gadget performs better overall than state-of-the-art methods: Snuba, GOGGLES, and self-learning baselines that use CNNs (VGG-19 [29] and MobileNetV2 [27]) without pre-training. We release our code as a community resource [1].

In the rest of the paper, we present the following:

- The architecture of Inspector Gadget (Section 2).
- The component details of Inspector Gadget:
  - Crowdsourcing workflow for helping workers identify patterns (Section 3).
  - Pattern augmenter for expanding the patterns using GANs and policies (Section 4).
  - Feature generator and labeler for generating similarity features and producing weak labels (Section 5).
- Experimental results where Inspector Gadget outperforms state-of-the-art image labeling techniques – Snuba, GOGGLES, and self-learning baselines using CNNs – where there are few or no labels to start with (Section 6).

2. OVERVIEW

The main technical contribution of Inspector Gadget is its effective combination of crowdsourcing, data augmentation, and data programming for scalable image labeling for classification. Figure 2 shows the overall process of Inspector Gadget. First, a crowdsourcing workflow helps workers identify patterns of interest from images that may indicate defects. While the patterns are informative, they may not be enough and are thus augmented using generative adversarial networks (GANs) [12] and policies [8]. Each pattern effectively becomes a labeling function where it is compared with other images to produces similarities that indicate whether the images contain defects. A separate development set is used to train a small model that uses the similarity outputs as features. This model is then used to generate weak labels of images in the test set. Figure 3 shows the architecture of the Inspector Gadget system. After training the Labeler, Inspector Gadget only utilizes the components highlighted in gray for generating weak labels. In the following sections, we describe each component in more detail.

3. CROWDSOURCING WORKFLOW

Since collecting enough labels to train a CNN on the entire images can be expensive, we would like to utilize human knowledge as much as possible and reduce the additional

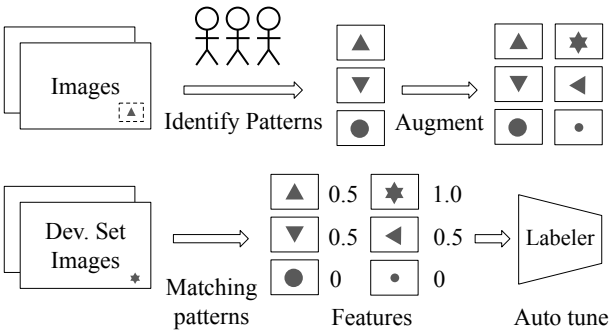


Figure 2: An overview of how Inspector Gadget constructs a model (labeler) that generates weak labels.

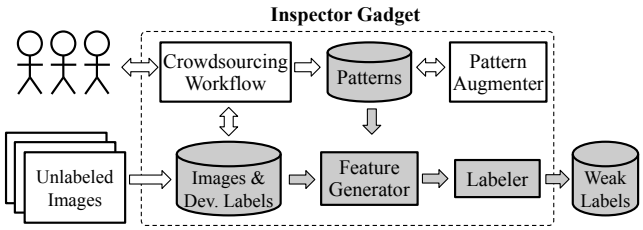


Figure 3: The architecture of Inspector Gadget.

amount of training data needed. We propose a crowdsourcing workflow shown in Figure 4. First, the workers mark defects using bounding boxes through a UI. Since the workers are not necessarily experts, the UI educates them how to identify defects beforehand. The bounding boxes in turn become the patterns we use to find other defects. Figure 5 shows sample images of a real-world smart factory dataset (called Product; see Section 6.1 for a description) where defects are highlighted with bounding boxes. Notice that the defects are not easy to find as they tend to be small and mixed with other larger parts of the product. Inspector Gadget can be extended to ask for more fine-grained segmentation of the defects instead of bounding boxes.

As with any crowdsourcing application, we may run into quality control issues where the bounding boxes of the workers vary even for the same defect. Inspector Gadget addresses this problem by first combining overlapping bounding boxes together. While there are several ways to combine boxes, we find that taking the average of their coordinates works reasonably well. For the remaining outlier bounding boxes, Inspector Gadget goes through a peer review phase where workers discuss which ones really contain defects. In Section 6.3, we perform ablation tests to show how each of these steps helps improve the quality of patterns.

Another challenge is determining how many images must be annotated to generate enough patterns. In general, we may not have statistics on the portion of images that have defects. Hence, our solution is to randomly select images and annotate them until the number of defective images exceeds a given threshold. In our experiments, identifying tens of defective images is sufficient (see the N_V^D values in Table 1). All the annotated images form a *development set*, which we use in later steps for model parameter tuning.

4. PATTERN AUGMENTER

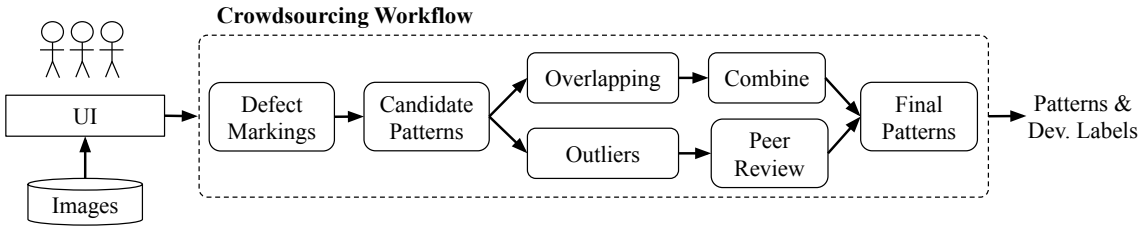


Figure 4: The crowdsourcing workflow of Inspector Gadget. Crowd workers can interact with a UI and provide bounding boxes that identify defects. The boxes are used as patterns, which are combined or go through a peer review phase.

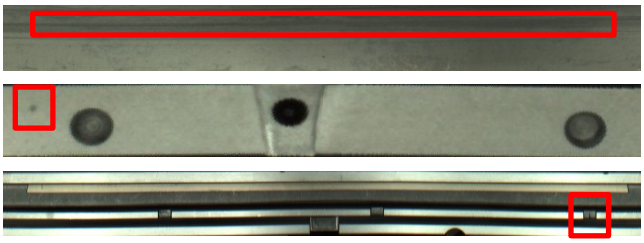


Figure 5: Sample images in the Product dataset (see Section 6.1) containing scratch, bubble, and stamping defects where we highlight each defect with a bounding box.

Pattern augmentation is a way to compensate for the possible lack of patterns even after using crowdsourcing. The patterns can be lacking if not enough human work is done to identify all possible patterns and especially if there are not enough images containing defects so one has to go through many images just to encounter a negative label. We would thus like to automatically generate more patterns without resorting to more expensive crowdsourcing.

We consider two types of augmentation – GAN-based [12] and policy-based [7]. The two methods complement each other and have been used together in medical applications for identifying lesions [11]. GAN-based augmentation is good at generating random variations of existing defects that do not deviate significantly. On the other hand, policy-based augmentation is better for generating specific variations of defects that can be quite different, exploiting domain knowledge. In Section 6.4, we show that neither augmentation subsumes the other, and the two can be used together to produce the best results.

The augmentation can be done efficiently because we are augmenting small patterns instead of the entire images. For high-resolution images, it is sometimes infeasible to train a GAN at all. In addition, if most of the image is not of interest for analysis, it is difficult to generate fake parts of interest while leaving the rest of the image as is. By only focusing on augmenting small patterns, it becomes practical to apply sophisticated augmentation techniques.

4.1 GAN-based Augmentation

The first method is to use generative adversarial networks (GAN) to generate variations of patterns that are similar to the existing ones. Intuitively, these augmented patterns can fill in the gaps of the existing patterns. The original GAN [12] trains a generator to produce realistic fake data where the goal is to deceive a discriminator that tries to distinguish the real and fake data. More recently, many variations of the original GAN have been proposed (see a recent survey [37]).

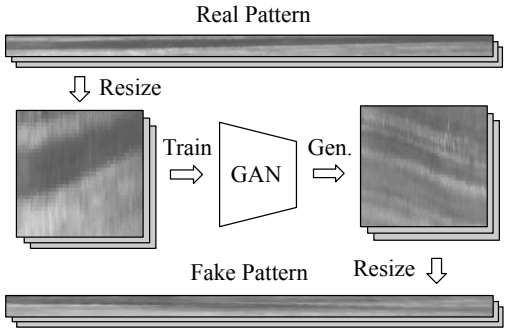


Figure 6: GAN-based augmentation on a pattern containing a scratch defect from the Product dataset.

Since we need the generated patterns to be as realistic as possible, we use a Relativistic GAN (RGAN) [17], which is designed for this purpose. The formulation of RGAN is:

$$\max_D \mathbb{E}_{(x_r, G(z))\sim(\mathbb{P},\mathbb{Q})}[\log(\sigma(D(x_r) - D(G(z))))]$$
$$\max_G \mathbb{E}_{(x_r, G(z))\sim(\mathbb{P},\mathbb{Q})}[\log(\sigma(D(G(z)) - D(x_r)))]$$

where G is the generator, x_r is real data, $D(x)$ is the probability that x is real, z is a random noise vector that is used by G to generate fake data, and σ is the sigmoid function. While training, the discriminator of RGAN not only distinguishes data, but also tries to maximize the difference between the probability that a real image is real and a fake image is fake. This setup enforces fake images to be as realistic as possible, in addition to simply being distinguishable from real images as in original GANs. We also use Spectral Normalization [21], which is a commonly-used technique applied to a neural network structure where the discriminator restricts the gradient to adjust the training speed for better training stability. Finally, since we are dealing with images, we use CNN models for the generator and discriminator.

Another preprocessing issue is that most GANs assume that the input images are of the same size and have a square shape. While this assumption is reasonable for a homogeneous set of images, the patterns identified may have different shapes. We thus fit patterns to a fixed-sized square shape by stretching or shrinking them and then augmenting the patterns. Then we re-adjust the new patterns into one of the sizes of the original patterns selected randomly. Figure 6 shows the entire process applied to an image.

4.2 Policy-based Augmentation

Policies [7] have been proposed as another way to augment images and complement the GAN approach. The idea is to use manual policies to decide how exactly an image is varied.

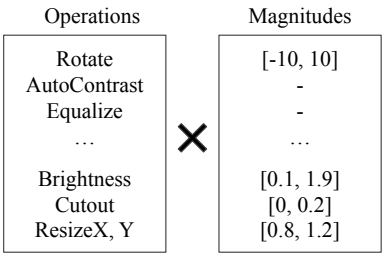


Figure 7: Augmentation using policies. We can evaluate and choose different combinations of operations and magnitudes.

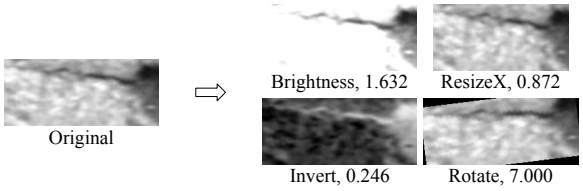


Figure 8: Policy-based augmentation on a pattern with a crack defect from the KSDD dataset (see Section 6.1). For each augmentation, we show the operation and magnitude.

We consider 17 types of policies where some are shown in Figure 7. Figure 8 shows the results of applying four of these policies on a surface defect from the KSDD dataset (see description in Section 6.1).

Policy-based augmentation is effective for patterns where applying operations based on human knowledge may result in quite different, but valid patterns. For example, if a defect is line-shaped, then it makes sense to stretch or rotate it. There are two parameters to configure: the operation to use and the magnitude of applying that operation. Recently, policy-based techniques have become more automated, e.g., AutoAugment [7] uses reinforcement learning to decide to what extent can policies be applied together.

We use an approach that is simpler than AutoAugment. Among certain combinations of policies, we choose the ones that work best on the development set. We first split the development set into train and test sets. For each policy, we specify a range for the magnitudes and choose 10 random values within that range. We then iterate all combinations of three policies. For each combination, we augment the patterns in the train set using the 10 magnitudes and train a model (see details in Section 5) on the train set images until convergence. Then we evaluate the model on the separate test set. Finally, we use the policy combination that results in the best accuracy and apply it to the entire set of patterns.

5. WEAK LABEL GENERATION

Once Inspector Gadget gathers patterns and augments them, the next step is to generate features of images (also called primitives according to Snuba [35]) and train a model that can produce weak labels. Note that the way we generate features of images is more direct than existing data programming systems that first convert images to structured data based on object detection (e.g., identify a vehicle) before applying labeling functions.

5.1 Feature Generator

Inspector Gadget provides feature generation functions that matches the patterns with new images to identify similar defects on any location and returns a real number for the similarity. Each output value of the feature generation function is used as an input of the labeler. Depending on the type of defect, the pattern matching may differ. A naïve approach is to do an exact pixel-by-pixel comparison, but this is unrealistic when the defects have variations. Instead, a better way is to compare the distributions of pixel values. This comparison is more robust to slight variations of a pattern. On the other hand, there may be false positives where an obviously different defect matches just because its pixels have similar distributions. In our experiments, we found that comparing distributions on the x and y axes using normalized cross correlation [2] is effective in reducing such false positives. Given an image I with pixel dimensions $W \times H$ and pattern P_i with dimensions $w \times h$, the i^{th} feature generation function FG_i is defined as:

$$FG_i(I) = \max_{x,y} \left| \frac{\sum_{x',y'} P_i(x',y') \cdot I(x+x', y+y')}{\sqrt{\sum_{x',y'} P_i(x',y')^2 \cdot I(x+x', y+y')^2}} \right|$$

where $0 \leq x < W - w$, $0 \leq y < H - h$, $0 \leq x' < w$, and $0 \leq y' < h$. When matching a pattern against an image, a straightforward approach is to make a comparison with every possible region of the same size in the image. However, scanning the entire image may be too time consuming. Instead, we use a pyramid method [3] where we first search for candidate parts of an image by reducing the resolutions of the image and pattern and performing a comparison quickly. Then just for the candidates, we perform a comparison using the full resolution.

5.2 Labeler

After the features are generated, Inspector Gadget trains a model on the output similarities of the feature generation functions, where the goal is to produce weak labels. The model can have any architecture and be small because there are not as many features as say the number of pixels in an image. We use a multilayer perceptron (MLP) because it is simple, but also has good performance compared to other models. An interesting observation is that, depending on the model architecture (e.g., the number of layers in an MLP), the model accuracy can vary significantly as we demonstrate in Section 6.5. Inspector Gadget thus performs model tuning where it chooses the model architecture that has the best accuracy results on the development set. This feature is powerful compared to existing CNN approaches where the architecture is complicated, and it is too expensive to consider other variations.

The final image labeling process consists of two steps. First, the patterns are matched to images for generating the features. Second, the trained MLP is applied on the features to make a prediction. We note that latency is not the most critical issue because we are generating weak labels, which are used to construct the training data for training the end discriminative model. Training data construction is usually done in batch mode instead of say real time.

6. EXPERIMENTS

We evaluate Inspector Gadget on real datasets and answer the following questions.

Dataset	Image size	N (N^D)	N_V (N_V^D)	Defect Type	Task Type
KSDD [32]	500 x 1257	399 (52)	78 (10)	Crack	Binary
Product (scratch)	162 x 2702	1673 (727)	170 (76)	Scratch	Binary
Product (bubble)	77 x 1389	1048 (102)	104 (10)	Bubble	Binary
Product (stamping)	161 x 5278	1094 (148)	109 (15)	Stamping	Binary
NEU [16]	200 x 200	300 per defect	100 per defect	Rolled-in scale, Patches, Crazing, Pitted surface, Inclusion, Scratch	Multi-class

Table 1: For each of the five datasets, we show the image size, the dataset size(N) and number of defective images(N^D), the development set size(N_V) and number of defective images within it(N_V^D), the types of defects detected, and the task type.

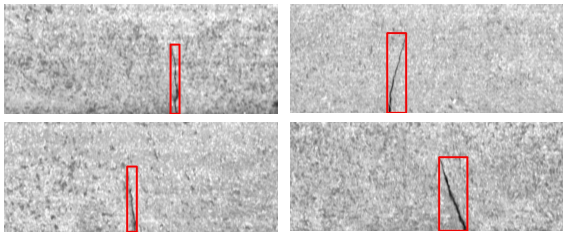
- How accurate are the weak labels of Inspector Gadget compared to other state-of-the-art methods and are they useful when training the end discriminative model?
- How useful is each component of Inspector Gadget?
- What are the errors made by Inspector Gadget?

We implement Inspector Gadget in Python and three machine learning libraries: Pytorch, TensorFlow, and Scikit-learn. We use an Intel Xeon CPU to train our MLP models and an NVidia Titan RTX GPU to train larger CNN models. Other details can be found in our released code [1].

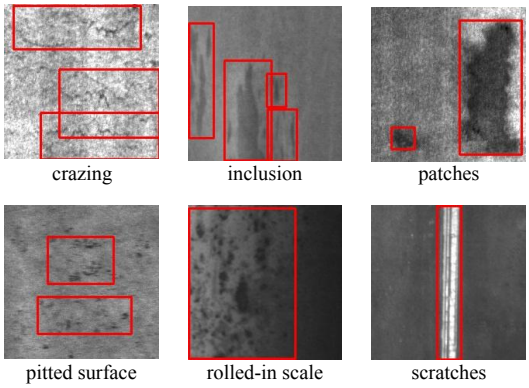
6.1 Settings

Datasets. We use real datasets for classification tasks. For each dataset, we construct a development set as described in Section 3. Table 1 summarizes the datasets with other experimental details, and Figure 9 shows samples of them. We note that the results in Section 6.2 are obtained by varying the size of the development set, and the rest of the experiments utilize the same development set size as described in Table 1. The NEU dataset has the smallest images, and the Product dataset has the largest. For each dataset, we have a gold standard of labels. Hence, we are able to compute the accuracy of the labeling on separate test data.

- The Kolektor Surface-Defect Dataset (KSDD [32]) is constructed from images of defected electrical commutators that were provided and annotated by the Kolektor Group. There is only one type of defect – cracks – but each one varies significantly in shape.
- The Product dataset (Figure 5) is proprietary and obtained through a collaboration with a manufacturing company that has a smart factory application. Each product has a circular shape where different strips are spread into rectangular shapes. There are three types of defects: scratches, bubbles, and stampings, which occur in different strips. The scratches vary in length and direction. The bubbles are more uniform, but have small sizes. The stampings are small and appear in fixed positions. We divide the dataset into three, as if there is a separate dataset for each defect type.
- The Northeastern University Surface Defect Database (NEU [16]) contains images that are divided into 6 defect types of typical surface defects of hot-rolled steel strips: rolled-in scale, patches, crazing, pitted surface, inclusion, and scratches. Compared to the other datasets, these defects take larger portions of the images. Since there are no images without defects, we solve the different task of multi-class classification where the goal is to determine which defect is present.



(a) KSDD data



(b) NEU data

Figure 9: Sample images in KSDD [32] and NEU [16] datasets where we highlight the defects with bounding boxes. See Figure 5 for the Product dataset images.

Pattern Matching. We use an OpenCV library function [2] that compares pixel distributions on the x and y axes using normalized cross correlation (explained in Section 5.1).

GAN-based Augmentation. We provide more details for Section 4.1. For all datasets, the input random noise vector has a size of 100, the learning rates of the generator and discriminator are both 1e−4, and the number of epochs is about 1K. We fit patterns to a square shape where the width and height are set to 100 or the averaged value of all widths and heights of patterns, whichever is smaller.

MLP Model Tuning. We use an L-BFGS optimizer [18], which provides stable training on small data, with a 1e−5 learning rate. We use k-fold cross validation where each fold has at least 20 examples per class and early stopping in order

to compare the accuracies of MLPs before they overfit.

Accuracy Measure. We use the F_1 score, which is the harmonic mean between precision and recall. Suppose that the set of true defects is D while the set of predictions is P. Then the precision $Pr = \frac{|D \cap P|}{|P|}$, recall $Re = \frac{|D \cap P|}{|D|}$, and $F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$. While there are other possible measures like AUC, F_1 is known to be more suitable for data where the labels are imbalanced [13] as in most of our settings.

Systems Compared. We compare Inspector Gadget with state-of-the-art image labeling systems and self-learning baselines that train a CNN model on available labeled data.

Snuba [35] automates the process of labeling function construction by starting from a set of primitives that are analogous to our feature generation functions and iteratively selecting subsets of them to train decision tree models, which becomes the labeling functions. Each iteration involves comparing models trained on all possible subsets of the primitives up to a certain size. Finally, the labeling functions are combined into a generative model. We faithfully implement Snuba and use our crowdsourced and augmented patterns for generating primitives, in order to be favorable to Snuba. However, adding more patterns quickly slows down Snuba as its runtime is exponential to the number of patterns.

We also compare with GOGGLES [9], which takes the opposite approach of not using crowdsourcing. However, it relies on the fact that there is a pre-trained model and extracts semantic prototypes of images where each prototype represents the part of an image where the pre-trained model is activated the most. Each image is assumed to have one object, and GOGGLES clusters similar images for unsupervised learning. In our experiments, we use the opensourced code of GOGGLES and a pre-trained VGG-16 model [29].

Finally, we compare Inspector Gadget with self-learning [33] baselines that train CNN models on the development set using cross validation and use them to label the rest of the images. To make a fair comparison, we experiment with both heavy and light-weight CNN models. For the heavy model, we use VGG-19 [29], which is widely used in the literature. For the light-weight model, we use MobileNetV2 [27], which is designed to train efficiently in a mobile setting, but nearly has the performance of heavy CNNs. We also make a comparison with VGG-19 whose weights are pre-trained on ImageNet [10]. (A pre-trained MobileNetV2 does not perform as well and is thus not compared.) Transfer learning obviously gives a significant boost in performance, and we do not claim that Inspector Gadget, which is not pre-trained, is better than this version of VGG-19. In addition, we use preprocessing techniques on images that are favorable for the baselines. For example, the images from the PRODUCT dataset are long rectangles, so we split each image in half and stack them on top of each other to make them more square-like, which is advantageous for CNNs.

6.2 Weak Label Accuracy

We compare the weak label accuracy of Inspector Gadget with the other methods. Figure 10 compares Inspector Gadget with GOGGLES and the self-learning baselines by increasing the amount of training data and observing the F_1 scores. To clearly show how Inspector Gadget compares with other methods, we use a solid line to draw its plot while using dotted lines for the rest. Among the models that are *not pre-trained* (i.e., ignore "SL (VGG19 + Pre-training)" for now), we observe that Inspector Gadget performs best overall because it is either the best or second-best method in all figures. This result is important because industrial images have various defect types that must all be identified correctly. For KSDD (Figure 10d), Inspector Gadget performs the best because the pattern augmentation helps Inspector Gadget find more variations of cracks (see Section 6.4). For Product (Figures 10a–10c), Inspector Gadget consistently performs the first or second best despite the different characteristics of the defects. For NEU (Figure 10e), Inspector Gadget ranks first for the multi-class classification.

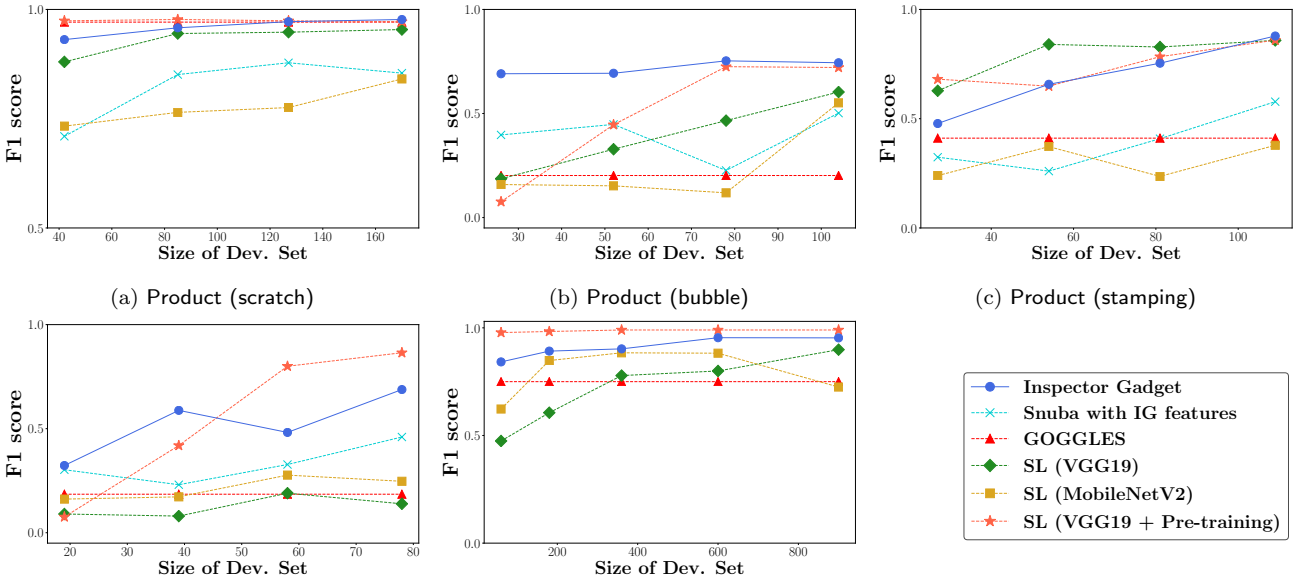
We explain the performances of other methods. Snuba consistently has a lower F_1 than Inspector Gadget possibly because the number of patterns is too large to handle. Instead of considering all combinations of patterns and training decision trees, Inspector Gadget's approach of training and auto-tuning an MLP works better for our experiments. GOGGLES does not need training data and thus has a constant accuracy. In Figure 10a, GOGGLES has a high F_1 because the defect sizes are large, and the pre-trained VGG-16 is effective in identifying them as objects. For the other figures, however, GOGGLES does not perform as well because the defect sizes are small and difficult to identify as objects. VGG-19 without pre-training ("SL (VGG19)") only performs the best in Figure 10c where CNN models are very good at detecting stamping defects because they appear in a fixed location on the images. For other figures, VGG-19 performs poorly because there is not enough labeled data. MobileNetV2 does not perform well in any of the figures. Finally, the pre-trained VGG-19 ("SL (VGG19 + Pre-training)") does outperform Inspector Gadget in Figures 10a, 10d, and 10e. While we do not claim that Inspector Gadget outperforms pre-trained models, it is the best option when pre-training is not possible.

6.3 Crowdsourcing Workflow

We evaluate how effectively we can use the crowd to label and identify patterns using the Product datasets. Table 2 compares the full crowdsourcing workflow in Inspector Gadget with two variants: (1) a workflow that does not average the patterns at all and (2) a workflow that does average the patterns, but still does not perform peer reviews. For each scenario, we compare the F_1 score of the MLP trained on the similarity features generated by matching the patterns on the development set, without using pattern augmentation. As a result, the full workflow clearly performs the best for the Product (scratch) and Product (stamping) datasets. For the Product (bubble) dataset, the workflow that does not combine patterns has a better average F_1, but the accuracies vary among different workers. Instead, it is better to use the stable full workflow without the variance.

6.4 Pattern Augmentation

We evaluate how augmented patterns help improve the weak label F_1 score of Inspector Gadget. Table 3 shows the impact of the GAN-based and policy-based augmentation on the five datasets. For each augmentation, we add 100–500 patterns, which empirically results in the best F_1 improvements (see below). When using both augmentations, we simply combine the patterns from each augmentation. As



(a) Product (scratch) (b) Product (bubble) (c) Product (stamping)

(d) KSDD [32] (e) NEU [16]

Figure 10: Weak label accuracy comparison between Inspector Gadget, Snuba [35], GOGGLES [9], and the self-learning baselines (SL) using VGG-19 [29] (with or without pre-trained weights) and MobileNetV2 [27] on different sizes of training data. Among the models that are *not pre-trained*, Inspector Gadget performs either the best or second-best in all figures.

	F_1 scores		
Dataset	**No avg. ($\pm std/2$)**	**No peer review**	**Full workflow**
Product (scratch)	0.940 ($\pm$0.005)	0.952	0.960
Product (bubble)	0.616 ($\pm$0.045)	0.525	0.605
Product (stamping)	0.299 ($\pm$0.142)	0.543	0.595

Table 2: Crowdsourcing workflow ablation results on the Product datasets. Each F_1 score is the performance of the MLP trained on each workflow.

Dataset	**No Aug.**	**Policy Based**	**GAN Based**	**Using Both**
KSDD [32]	0.415	0.578	0.509	**0.688**
Product (scratch)	0.958	0.965	0.962	**0.979**
Product (bubble)	0.617	0.702	**0.715**	0.701
Product (stamping)	0.700	0.700	0.765	**0.859**
NEU [16]	0.936	**0.954**	0.930	**0.954**

Table 3: Pattern augmentation impact on Inspector Gadget. For each dataset, we highlight the highest F_1 score.

a result, while each augmentation helps improve F_1, using both of them usually gives the best results.

Figure 11 shows how adding patterns impacts the F_1 score for the Product (bubble) dataset. While adding more patterns helps to a certain extent, it has diminishing returns afterwards. The results for the other datasets are similar, although sometimes noisier. The best number of augmented patterns differs per dataset, but falls in the range of 100–500.

6.5 Model Tuning

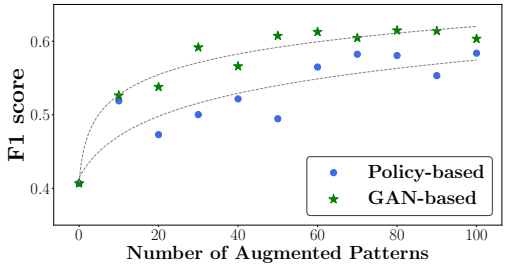


Figure 11: Policy-based and GAN-based augmentation results on the Product (stamping) dataset.

We evaluate the impact of model tuning on accuracy described in Section 5.2 as shown in Figure 12. We use an MLP with 1 to 3 hidden layers and varied the number of nodes per hidden layer to be one of $\{2^n | n = 1 \dots m$ and $2^{m-1} \leq I \leq 2^m\}$ where I is the number of input nodes. For each dataset, we first obtain the maximum and minimum possible F_1 scores by evaluating all the tuned models we considered directly on the test data. Then, we compare these results with the (test data) F_1 score of the actual model that Inspector Gadget selected after comparing the models using the development set. We observe that the model tuning in Inspector Gadget can indeed improve the model accuracy, close to the maximum possible value.

6.6 End Model Accuracy

We now address the issue of whether the weak labels are actually helpful for training the end discriminative model. We compare the F_1 score of this end model with the same model that is trained on the development set. For the discriminative model, we use VGG-19 [29] for the binary classification tasks on KSDD and Product, and ResNet50 [14] for

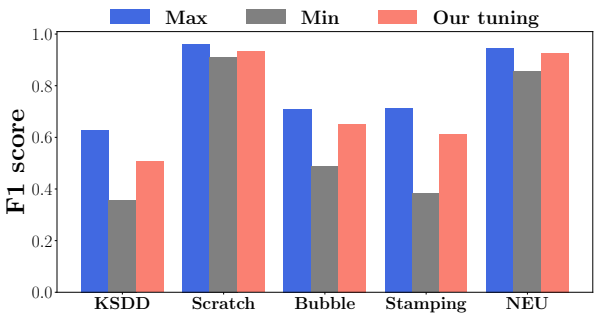


Figure 12: The variations in F_1 scores when tuning the MLP model hyper-parameters.

Dataset	End Model	Dev. Set	WL (IG)
KSDD [32]	VGG19	0.499	0.700
Product (scratch)	VGG19	0.925	0.978
Product (bubble)	VGG19	0.359	0.720
Product (stamping)	VGG19	0.782	0.876
NEU [16]	ResNet50	0.953	0.970

Table 4: F_1 scores of end models trained on the development set (Dev. Set) only or the development set combined with the weak labels produced by Inspector Gadget (WL (IG)).

the multi-class task on NEU. We can use other discriminative models that have higher absolute F_1, but the point is to show the *relative* F_1 improvements when using weak labels. Table 4 shows that the F_1 scores improve by 0.02–0.36.

6.7 Error Analysis

We perform an error analysis on which cases Inspector Gadget fails to make correct predicts for the five datasets based on manual investigation. We use the ground truth information for the analysis. Table 5 shows that most common error is when certain defects do not match with the patterns, which can be improved by using better pattern augmentation and matching techniques. The next common case is when the data is noisy, which can be improved by cleaning the data. The last case is the most challenging where even humans have difficulty identifying the defects because they are not obvious (e.g., a near-invisible scratch).

7. RELATED WORK

Crowdsourcing for machine learning. Using humans for advanced analytics is increasingly becoming mainstream [38]. While there is a push to make all steps of machine learning automatic, there are cases where humans are essential and the key issue is using them with the minimum effort. There is also a heavy literature in HCI on how to integrate crowdsourcing in machine learning [6]. These methods focus on how to provide the right interfaces to help users construct features. Inspector Gadget builds on top of these approaches by relying on crowdsourcing for identifying patterns.

Data Programming. Data programming [25] is a recent paradigm where workers program labeling functions, which are used to generate weak labels at scale. Snorkel [24, 4] is a seminal system that demonstrates the practicality of data programming, and Snuba [35] extends it by automatically constructing labeling functions using primitives. In comparison, Inspector Gadget does not assume any accuracy guarantees on the feature generation function and directly labels images without converting them to structured data.

Dataset	Cause		
	Matching failure	Noisy data	Difficult to humans
KSDD [32]	10 (52.6 %)	5 (26.3 %)	4 (21.1 %)
Product (scratch)	11 (36.7 %)	11 (36.7 %)	8 (26.6 %)
Product (bubble)	19 (45.2 %)	15 (35.7 %)	8 (19.1 %)
Product (stamping)	15 (45.5 %)	13 (39.4 %)	5 (15.1 %)
NEU [16]	35 (63.6 %)	4 (7.3 %)	16 (29.1 %)

Table 5: Error analysis of Inspector Gadget.

Several systems have studied the problem of automating labeling function construction. CrowdGame [19] proposes a method for constructing labeling functions for entity resolution on structured data. Adversarial data programming [23] proposes a GAN-based framework for labeling with labeling function results and claims to be better than Snorkel-based approaches. In comparison, Inspector Gadget solves the different problem of partially analyzing large images.

Automatic Image labeling. There is a variety of general automatic image labeling techniques. Data augmentation [28] is a general method to generate new labeled images. Generative adversarial networks (GANs) [12] have been proposed to generate fake, but realistic images based on existing images. Policies [7] were proposed to apply custom transformations on images as long as they remain realistic. Most of the existing work operate on the entire images. In comparison, Inspector Gadget is efficient because it only needs to augment patterns, which are much smaller than the images. Label propagation techniques [5] organize images into a graph based on their similarities and then propagates existing labels of images to their most similar ones. In comparison, Inspector Gadget is designed for images where only a small part of them are of interest while the main part may be nearly identical to other images, so we cannot utilize similarities. There are also application-specific defect detection methods [30, 16, 15], some of which are designed for the datasets we used. In comparison, Inspector Gadget provides a general framework for image labeling. Recently, GOGGLES [9] is an image labeling system that relies on a pre-trained model to extract semantic prototypes of images and construct an affinity matrix that can be used to identify similar images. In comparison, Inspector Gadget does not rely on pre-trained models and is more suitable for partially analyzing large images using human knowledge.

8. CONCLUSION

We proposed Inspector Gadget, a scalable image labeling system for classification problems that effectively combines crowdsourcing, data augmentation, and data programming techniques. Inspector Gadget targets applications in manufacturing where large industrial images are partially analyzed, and there are few or no labels to start with. Unlike existing data programming approaches that convert images

to structured data beforehand, Inspector Gadget directly labels images by providing a crowdsourcing workflow to leverage human knowledge for identifying patterns of interest. The patterns are then augmented and matched with other images to generate similarity features for MLP model training. Our experiments show that Inspector Gadget outperforms the state-of-the-art methods Snuba, GOGGLES, and self-learning baselines using CNNs without pre-training. We thus believe that Inspector Gadget opens up a new class of problems to apply data programming.

9. REFERENCES

[1] Inspector gadget github repository. `https://github.com/geonheo/InspectorGadget`. Accessed April 1st, 2020.

[2] Opencv. `https://docs.opencv.org/2.4/modules/imgproc/doc/object_detection.html`. Accessed April 1st, 2020.

[3] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden. 1984, Pyramid methods in image processing. *RCA Engineer*, 29(6):33–41, 1984.

[4] S. H. Bach, D. Rodriguez, Y. Liu, C. Luo, H. Shao, C. Xia, S. Sen, A. Ratner, B. Hancock, H. Alborzi, R. Kuchhal, C. Ré, and R. Malkin. Snorkel drybell: A case study in deploying weak supervision at industrial scale. In *SIGMOD*, pages 362–375, 2019.

[5] T. D. Bui, S. Ravi, and V. Ramavajjala. Neural graph learning: Training neural networks using graphs. In *WSDM*, pages 64–71, 2018.

[6] J. Cheng and M. S. Bernstein. Flock: Hybrid crowd-machine learning classifiers. In *CSCW*, pages 600–611, 2015.

[7] E. D. Cubuk, B. Zoph, D. Mané, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[8] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pages 113–123, 2019.

[9] N. Das, S. Chaba, S. Gandhi, D. H. Chau, and X. Chu. GOGGLES: automatic training data generation with affinity coding. In *SIGMOD*, 2020.

[10] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[11] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan. Gan-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[12] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, pages 2672–2680, 2014.

[13] Q. Gu and Z. Zhu, Liand Cai. Evaluation measures of the classification performance of imbalanced data sets. In Z. Cai, Z. Li, Z. Kang, and Y. Liu, editors, *CIIS*, pages 461–471, Berlin, Heidelberg, 2009.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[15] Y. He, K. Song, H. Dong, and Y. Yan. Semi-supervised defect classification of steel surface based on multi-training and generative adversarial network. *Optics and Lasers in Engineering*, 122:294–302, 2019.

[16] Y. He, K.-C. Song, Q. Meng, and Y. Yan. An end-to-end steel surface defect detection approach via fusing multiple hierarchical features. *IEEE Transactions on Instrumentation and Measurement*, 69:1493–1504, 04 2020.

[17] A. Jolicoeur-Martineau. The relativistic discriminator: a key element missing from standard GAN. In *ICLR*, 2019.

[18] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Math. Program.*, 45(1-3):503–528, 1989.

[19] T. Liu, J. Yang, J. Fan, Z. Wei, G. Li, and X. Du. Crowdgame: A game-based crowdsourcing system for cost-effective data labeling. In *SIGMOD*, pages 1957–1960, 2019.

[20] Y. Liu, T. Kohlberger, M. Norouzi, G. Dahl, J. Smith, A. Mohtashamian, N. Olson, L. Peng, J. Hipp, and M. Stumpe. Artificial intelligence based breast cancer nodal metastasis detection: Insights into the black box for pathologists. *Archives of Pathology Laboratory Medicine*, 2018.

[21] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *CoRR*, abs/1802.05957, 2018.

[22] E. Oztemel and S. Gursev. Literature review of industry 4.0 and related technologies. *Journal of Intelligent Manufacturing*, 2018.

[23] A. Pal and V. N. Balasubramanian. Adversarial data programming: Using gans to relax the bottleneck of curated labeled data. In *CVPR*, pages 1556–1565, 2018.

[24] A. J. Ratner, S. H. Bach, H. R. Ehrenberg, and C. Ré. Snorkel: Fast training set generation for information extraction. In *SIGMOD*, pages 1683–1686, 2017.

[25] A. J. Ratner, C. D. Sa, S. Wu, D. Selsam, and C. Ré. Data programming: Creating large training sets, quickly. In *NIPS*, pages 3567–3575, 2016.

[26] Y. Roh, G. Heo, and S. E. Whang. A survey on data collection for machine learning: a big data - AI integration perspective. *IEEE TKDE*, 2019.

[27] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, 2018.

[28] C. Shorten and T. M. Khoshgoftaar. A survey on image data augmentation for deep learning. *J. Big Data*, 6:60, 2019.

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[30] K.-C. Song and Y. Yan. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Applied Surface Science*, 285:858–864, 2013.

[31] M. Stonebraker and E. K. Rezig. Machine learning and big data: What is important? *IEEE Data Eng.*

Bull., 2019.

[32] D. Tabernik, S. Šela, J. Skvarč, and D. Skočaj. Segmentation-Based Deep-Learning Approach for Surface-Defect Detection. *Journal of Intelligent Manufacturing*, May 2019.

[33] I. Triguero, S. García, and F. Herrera. Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowl. Inf. Syst.*, 42(2):245–284, 2015.

[34] P. Varma, B. D. He, P. Bajaj, N. Khandwala, I. Banerjee, D. L. Rubin, and C. Ré. Inferring generative model structure with static analysis. In *NeurIPS*, pages 240–250, 2017.

[35] P. Varma and C. Ré. Snuba: Automating weak supervision to label training data. *PVLDB*, 12(3):223–236, 2018.

[36] P. Varma, F. Sala, A. He, A. Ratner, and C. Ré. Learning dependency structures for weak supervision models. In K. Chaudhuri and R. Salakhutdinov, editors, *ICML*, volume 97, pages 6418–6427, 2019.

[37] Z. Wang, Q. She, and T. E. Ward. Generative adversarial networks: A survey and taxonomy. *CoRR*, abs/1906.01529, 2019.

[38] D. Xin, L. Ma, J. Liu, S. Macke, S. Song, and A. G. Parameswaran. Accelerating human-in-the-loop machine learning: Challenges and opportunities. In *DEEM@SIGMOD*, pages 9:1–9:4, 2018.