IT UNIVERSITY OF COPENHAGEN

Semi-supervised classification of engine parts

Christoffer Ebbe Sommerlund (csom@itu.dk)
Thomas Fosdam Claudinger (thcl@itu.dk)

Research Project
MSc Data Science
KIREPRO1PE

January 22, 2024
Supervisor: Stella Grasshof

Abstract

This paper tries to re-implement the semi-supervised method provided by FixMatch [24], which is a method for training on unlabeled data using the models own predictions on augmentations of the unlabeled data. We combine this with fine-tuning on a new domain, i.e. engine component classification. A proprietary dataset was provided by MAN-ES Copenhagen, which includes a bunch of noisy unlabeled samples totaling ≈100,000 images, of which many of them should be filtered out. We denote these outliers/anomalies as "no-class" labels. About 10% of the dataset has been labeled by a previously project [5]. Our results are summarised in three experiments. Firstly, *One-class classification* to have a method to sort out the "no-class" images in the dataset. Secondly, *Engine parts classification* to see if the FixMatch method were an improvement compared to a normal supervised approach including a ablation study of the data amount needed and the parameters of FixMatch. Thirdly, *FixMatch dataset comparison* to see if our re-implementation could achieve the same trends as presented in the FixMatch paper. We find promising results with respect to both rejecting "no-class" samples and classifying engine components, however, the FixMatch method does not yield consistent improvements compared to traditional supervised fine-tuning. The source code is available at `https://github.com/circle-queue/component-classifier`.

Contents

1 Introduction

In machine learning projects today there is a tendency to increase the parameters of models and the training data to reach higher accuracies. However, in industry it is often expensive and difficult to get large amounts of annotated data for their domain. A common solution is transfer learning, specifically fine-tuning, where a large model has been pre-trained on a general dataset, and only part of the model is trained in order to fit the new domain. Another approach is semi-supervised learning, in which a fraction of data is annotated, and is used in combination with the remaining unlabeled data in order to train the model.

This paper aims to combine the two. We propose to adapt a pre-trained deep learning model to a new domain using the semi-supervised approach proposed by FixMatch [24]. FixMatch claims its method does not demand thousands of annotated samples in different categories in order to train a model. Instead they show examples of using 4, 25, 100 and 400 labels per category and still getting an accuracy of up to 95 percent. *How effective is FixMatch when combined with fine-tuning on a realistic dataset from industry?* Specifically, if we re-implement FixMatch in PyTorch and evaluate it on our industry dataset, which includes unlabeled data which may not fit into any of the provided categories ("no-class" samples) and class labels which are unevenly distributed. We also evaluate our implementation on the original datasets from the FixMatch paper and compare the two.

The work can be summarised in three experiment sections. In section 3.1, we examine methods for rejecting these "no-class" samples. This can be used during deployment to reject unintended input data and during training to ensure high quality unlabeled training data. This paper, however, does not use it as a filtering mechanism for downstream training, but instead evaluates the method in isolation. In section 3.2 we evaluate the robustness of the FixMatch method on our dataset using ablation studies on the FixMatch parameters and the size of the training data. Finally, in section 3.3 we contrast our implementation and usage of fine-tuning to the original results of the FixMatch paper.

We collaborate with the company MAN-ES in Copenhagen who provide us the dataset. The aim is for this to become a filtering step in a larger pipeline for evaluating the state of an engine and whether service is needed. In the future, this pipeline should be deployed in a low-bandwidth low-resource environment at sea as e.g. a smartphone application.

1.1 Related Work

Fine-tuned models provide many benefits over training a model from scratch. First, it alleviates the issue of defining an effective network architecture, which can often be difficult in itself [12]. Second, it allows transferring some knowledge from one domain to another, reducing training data required to achieve good performance [14]. This assumes that there is common knowledge to be transferred, which means that the data which has been fine-tuned on is critical for importance. This is demonstrated in [4] which compares fine-tuning from a model trained on natural images versus digital documents, and finds the latter to be more effective for the task of table detection.

In our case we have unlabeled data spanning across multiple domains. Some images include digital documents or pictures of digital graphs, and others are of natural, out-of-domain images such as people or cranes. There's generally two approaches to filtering these images, either novelty or outlier detection, which differ in their definition of training. In case of novelty detection, we have a pure training set and want to reject new samples that do not belong. For outlier detection, our training set is contaminated, and we need to filter out some portion of it [15]. This is a very difficult topic where even human annotators can disagree due to having different definitions of anomalies. The review of [21] lists many of the challenges and how deep learning can be utilized to combat these. Note that due to time constraints, we do not follow most of these recommendations, such as end-to-end training and tailored feature representations.

It is often expensive to label this unlabeled data, which is the case for us. Semi-supervised learning is trying to leverage the unlabeled data we have and get a better predictor compared to only using the labeled

data [20]. The unlabeled data might improve the model with information about the distribution of the data or estimate a decision boundary between classes in the dataset [20].

The paper in [11] gives an example of the semi-supervised algorithm used in Google Photos. This algorithm clusters faces that look a like from the photos the user uploads. The user has to label a few of them. Hereafter, the algorithm is able to label the labeled faces in all of your photos.

In [3], they demonstrate how augmentations can prevent overfitting of the model to an imbalanced dataset of less than 100 samples. They used convolutional neural networks to detect cracks in images of engine parts. Their results showed that data augmentations improved the accuracy of their model by up to 7 %.

A literature review on transfer learning in medical imaging papers [14] proposes to start with a ResNet or Inception architecture depending on the task and fine-tuning only the last fully connected layers. If that does not give expected results or the model should be refined, they recommend unfreezing and fine-tuning layers in the direction output to input.

FixMatch is a continuation of MixMatch [2] and ReMixMatch [1] which aims to simplify the two. MixMatch used a approach with K number of augmented samples per unlabeled sample. The label prediction distribution of each K augmented sample is added together and averaged with the highest probable as the predicted label for that sample [2].

ReMixMatch was an improvement of MixMatch, they added two new techniques to the model; *distribution alignment* and *augmentation anchoring*. *Distribution alignment* in this case meant for the model to predict each class with equal frequency. So they normalized the class probability distribution for that sample by the distribution of previously predicted labels. *Augmentation anchoring* being generating a weakly augmented version of the unlabeled sample that is used as an anchor for "K" number of strongly augmented versions used in ReMixMatch [1].

2 Methods

2.1 FixMatch

FixMatch uses a semi-supervised approach combining learning from both supervised and unsupervised samples. They minimize a combination of the supervised loss l_s and unsupervised loss l_u, more specifically they minimize the loss function $l_{total} = l_s + \lambda_u l_u$, where λ_u is the weight you want to give the unsupervised loss.

The supervised loss is simply calculated as the cross-entropy $H(p, q)$ between our model probability prediction $f(y|x_i)$ on a weakly augmented image $\alpha(x_i)$ (flipped or shifted) and the true label y_i, as

$$l_s = \frac{1}{B} \sum_{i=1}^{B} \mathrm{H}(y_i, f(y \mid \alpha(x_i))) \tag{2.1}$$

where the cross-entropy H for C classes is defined as

$$H(y_i, f(y|x_i)) = -\sum_{c \in C} y_i * \log p(y_c|x_i). \tag{2.2}$$

The flow of calculating the unsupervised loss is shown in Figure 2.1 and consists of two augmentations of the same sample. In the top of the figure they compute the most probable (pseudo) label of a weakly augmented image. However the prediction confidence must surpass a threshold τ (shown as the dotted line). In the bottom flow, model prediction on the strongly augmented image (described in section 2.1.2) is compared with the pseudo label, and the loss is computed using cross-entropy H between the two. Finally, the FixMatch method defined μ as the number of unsupervised batches per supervised batch during training, where the supervised batch size is denoted using B.

Formally, the unsupervised loss on an unlabeled image u_i is described as the following:

$$l_u = \frac{1}{\mu B} \sum_{i=1}^{\mu B} \mathbb{1}(\max(Z_i) \geq \tau) \mathrm{H}(\arg\max(Z_i), f(y \mid \mathrm{A}(u_i))) \tag{2.3}$$

where Z is defined as the models assigned probability distribution to the weakly augmented image, i.e. $Z_i = f(y|\alpha(u_i))$. $\mathbb{1}$ is the identity function, in this case evaluating to 1 if the pseudo label is confident and 0 otherwise, while $A(u_i)$ the strongly augmented image.

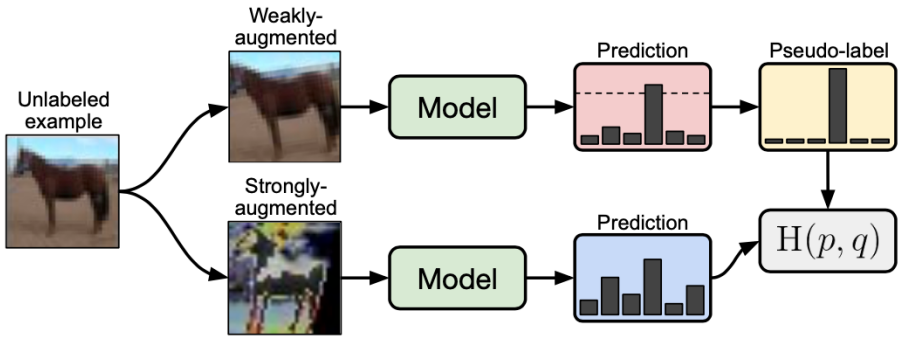


Figure 2.1: Diagram of the FixMatch method from their paper [24].

2.1.1 Implementation details

Recall that B is the supervised batch size and μ the ratio of unlabeled to labeled data. FixMatch defines an unsupervised batch as *"a batch of μB unlabeled examples"* [24]. It is unclear if the unlabeled dataset is sampled before each run, i.e. they are training on the same unlabeled samples each epoch, or if they are sampled from the entire unsupervised training set each batch (e.g. with replacements). We found the latter to be unrealistic, having many orders of magnitude more unlabeled data compared to labeled. While that is true in some scenarios, we find it more transparent to sample the unsupervised dataset before training, as that defines the sizes of both the supervised- and unsupervised dataset. The latter would also require excessive memory and time. We therefore implement the first definition of sampling, meaning we sample the unsupervised dataset *prior to each model training*, such that the same unlabeled samples are trained on in each epoch. Formally, let n_U denote the number of unlabeled samples in the dataset and n_S the number of supervised samples available, then we can denote μ as

$$\mu = \frac{n_U}{n_S} \implies n_U = \mu * n_S \tag{2.4}$$

e.g. if $n_s = 1000$ samples and $\mu = 7$ then the total number of unsupervised training labels is

$$n_U = 7 * 1000 = 7000. \tag{2.5}$$

FixMatch also does not comment on their test sets, which could either be the entire pre-defined test sets of each dataset, or a sample of the test set. They state, *"We compute the mean and variance of accuracy when training on 5 different "folds" of labeled data"* [24]. The definition of a fold is unclear from the FixMatch paper, which can have a huge impact, especially on the subset of just 40 labels. It is unlikely that folds are 5-fold cross validation, as that would imply a test set of 20%, which results in 1 sample per class for the CIFAR-10 40 label subset. From the results, the variability seems too low for this to be the case. Instead, we interpret a fold as a random sampling, and assume the test set is the same size as the training set. As mentioned previously, once could also be evaluating on the entire test set, however, we believe this would be misleading, as one would in practice not have a huge test set available if training data is limited.

To summarize, prior to each training we sample the supervised and unsupervised splits, with a train/test/unsupervised ratio of 1:1:7 respectively. Samplings are stratified for our dataset, meaning it preserves the class label distribution.

2.1.2 RandAugment

FixMatch applies a strong augmentation method as shown in Figure 2.1. The original paper examines two such methods, whereas we only apply RandAugment [8] since it is available in PyTorch. In the original RandAugment implementation, there's a predefined set of augmentation which can distort an image with some magnitude. RandAugment takes two parameters: The number of augmentations to apply and the magnitude of each augmentation. The transformations are applied sequentially, resulting in an augmented image. FixMatch proposes a slight change which eliminates the magnitude parameter. They instead uniformly sample from a range. The paper specifies ranges for each function, however, the PyTorch implementation has a single magnitude, which it internally converts to ranges for each function. Therefore, we sample a magnitude in the span of 5% to 95%, which is approximately the same as the paper. We demonstrate the differences in the appendix Table A.1.

Notably, the paper does not discuss the parameter denoting the number of augmentations to apply. We imagine this can have significant effects on FixMatch's performance, and we therefore apply an ablation study on the number of augmentations to perform as seen in appendix C. We find an optimum at 2, which is also the default value on their GitHub[1].

[1] line 182 @ `github.com/google-research/fixmatch/blob/d4985a158065947dba803e626ee9a6721709c570/libml/augment.py`

2.2 One-class classification (OCC)

Once models are deployed, they are likely exposed to data which they were not intended to handle. In our case, users may try to apply the model on a component which is not represented in the training data (a "new" class label). Another use case may be annotating large and messy collections of data. In these cases, it is desirable for the model to reject unintended input, which we assign the label of "no-class" and the training data as "has-class". Therefore in regards to one-class classification we have the following two classes:

- "no-class": Negative cases, Anomalies, outliers, out-of-distribution labels. Samples that "do not belong".
- "has-class": Positive cases, Samples which fit an annotated label (i.e. one of the 8 classes in Appendix D.

One method for this could be setting a confidence threshold, which rejects all images which result in a predicted confidence less than e.g. 95%. However, this doesn't work great for our dataset, since classes may have overlapping data distributions. Consider the case of "single piston ring" and "piston ring overview". Even if the model is uncertain, it does not mean the image does not fit into any class. Another approach could be adding a "no-class" label to the annotation process, and perform the same multi-class classification process with this extra label. However, due to a lack of annotated "no-class" data and a lack of time for annotating said data, this was not pursued. Instead, we apply a method known as one-class classification (OCC), which aims to create a binary in/out-of-domain classifier based purely on the training data.

We examine 5 approaches for OCC: 3 are distance based, 1 is O-SVM [13] and the final approach is ECOD [17] using an implementation by PyOD [29]. The methods are in increasing complexity. We will introduce the 3 distance methods in more detail compared to the remaining two, since the latter have implementations in the citations.

The 3 distance methods all aim to define a "center-of-mass" of data. The first and simplest case is the centroid, which averages N training embeddings $\mathbf{e}$ into a single embedding vector (the centroid).

$$centroid = \frac{1}{N}\sum_{i}^{N}\mathbf{e}_i \tag{2.6}$$

We can now compute the distance from every training sample to the centroid, and using a hyperparameter q, select a quantile for our cutoff threshold. We use $q = 0.05$ (5% quantile) in the experiments. This results in 5% of the training data being labeled "no-class", which can also be interpreted as a false negative rate. Without additional data, we can apply this data to a development dataset and an unlabeled dataset. For the development set, we expect a similar (false) negative rate as the training data, and the difference between the negative rate for the development dataset and the unlabeled dataset is the expected amount of "no-class" data in an unlabeled dataset. More formally, let D be either the train, development or unlabeled dataset, where the train dataset is what the model is trained on, the development set contains unseen data for the model, but from the same distribution of the train data, and the unlabeled data contains an unknown distribution of data. Let NR_D and TNR_D denote respectively the negative rate and true negative rate of the dataset D.

$$NR_{train} = \frac{\text{number of "no-class" predictions}}{\text{number of samples}} \tag{2.7}$$

Also assume $NR_{train} \approx NR_{development}$, since they are from the same distribution. Then the following is *loosely expected*:

$$TNR_{unlabeled} \approx NR_{unlabeled} - NR_{development} \tag{2.8}$$

It is worth emphasising the utility of this. One can estimate a dataset "pollution" ratio simply by using the labeled data with no additional annotation of unlabeled data. If the TNR is 50%, we expect half of the dataset is unsuitable as input for the model. This could also be used to evaluate different scraped data

source for their similarity to previous training material. Note that the inverse could also be useful in some cases, e.g. finding datasets with diverse data compared to the original.

While the above formula is simple to apply since it does not require annotations, a sample estimate of the TNR is still preferable, since a models perception of a negative/"no-label" case can be vastly different from that of an annotator. One should always consider this model perspective and especially the method and data of which it was trained on. Embeddings from a model trained on the ImageNet dataset may transfer well to other object classification tasks, but it may struggle representing fine nuances within radiology scans. The hyperparameter NR_{train} may also significantly effect the results. Finally, there's the fact that forcing the model to have any level of NR_{train} is conceptually wrong, since none of the training labels belong to "no-class". Once could easily pollute a fraction of the training dataset, and have the models identify that same fraction, however, then you're introducing another type of bias, that being which data to introduce. Do you pick random images from the internet? One could select data from the unlabeled dataset, but that would require annotating the unlabeled dataset to know what pollution rate to provide the model. In conclusion, the above method should only be used as a data exploration tool, and not as a concise metric.

2.2.1 Class means & K-means

This section describes the remaining 2 distance methods. The first was described above in formula 2.6 using the centroid of our training samples. This has the potential drawbacks of being very biased, since majority classes contribute more to the center of mass. As two alternative approaches, we can compute multiple center of masses, and find the smallest distance to any of these masses when evaluating a new sample. By computing a mean for each class, the class imbalance should have no impact. Alternatively, we can use a clustering algorithm to find n clusters, which still may unfairly prioritize majority classes, but the combination of the spherical clusters can cover a non-spherical shape of data compared to a single mean embedding vector, and therefore cover classes with a larger variance. The more clusters, the more complex this shape becomes.

For clustering, we use Scikit-learn's [22] implementation of K-Means with 20 clusters and otherwise default parameters as of v1.3.2. K-means is an iterative clustering algorithm which randomly initialises a number of cluster centers, and assigns each point to the closest center. From this point assignment, each cluster can compute a new center, and the process repeats until convergence. The number of clusters is a hyperparameter, which we choose not to optimize due to time constraints. Instead, we prioritize comparing multiple OCC methods. Ideally one would also perform multiple runs, since the algorithm is non-deterministic due to the random initialization.

2.2.2 One-class support vector machines (OSVM)

Breifly explained, support vector machines (SVM) aim to split a space using a hyperplane where each side of the hyperplane corresponds to either a positive or a negative classification. Often there's many hyperplanes which separates the two classes, so SVM adds the additional constraint of selecting the hyperplane which maximises the distance (or margin) to the closest set of samples (also called support vectors), which turns it into an optimisation problem with an analytical solution using Lagrangian multipliers. One-class SVM (OSVM) behaves similarly, except instead of a hyperplane, it defines a hyper-sphere which encapsulates some percentage of the training data (the TPR, since the training data only contains positive cases). Anything which lies outside of the hyper-sphere is then classified as "no-class". It may not be possible to completely separate the two classes, which is why there exists a "soft" variant of SVM. This variant allows some errors when it comes to maximising the margins, potentially allowing it to misclassify some samples, but at the same time also generalize better. Another method for separating classes is using the kernel trick, which warps the space allowing for a non-spherical decision boundary (but spherical from the perspective of this warped space).

We use Scikit-learn's [22] implementation of OSVM with default parameters as of v1.3.2.

2.2.3 Empirical-Cumulative-distribution-based Outlier Detection (ECOD)

ECOD [17] assumes that outliers are often at the tails of data distributions, and that the likelihood of a sample can be estimated using the product of these tail probabilities (independence assumption). When

viewed in the context of semantic embeddings (e.g. a 768 length CLIP embedding), it would flag samples which often deviate significantly from the modes of each semantic embedding, and therefore has a significantly different semantic meaning (in the perspective of the model). The ECOD independence assumption means it is unable to capture higher order multivariate events, which is fine in our case, since our embeddings are the logits, i.e. layer activations right before the final linear classification layer, so no further nonlinear events take place from a model training perspective. For implementation, we use the python library PyOD [29].

2.2.4 Dimensionality reduction

By applying dimensionality reduction, we summarize our models perception of the high dimensional images in a two dimensional plot. The purpose of this is to asses the difficulty of separating "no-class" from "has-class". If we label a portion of the data, and visually see the two groups clearly separated, we expect the model to perform well. We also apply it to find cases of difficult or boundary cases where the image contains a class, but it is too blurry or dark for a human to deem it valuable/"has-class".

We achieve this by first converting the pixels into semantic embeddings with a pretrained model, such that distances no longer describe differences in color intensity per pixel, but instead describe semantic differences as perceived by the model [19]. Then, we can apply a dimensionality reduction method like TSNE [28]. Note that we do not standard scale prior to embedding, since if an embedding dimension has a narrow span, that reflects a narrow semantic difference, and standard scaling would normalize that semantic difference to be equal for all embedding dimension. Another important detail is to only train TSNE on the training data, and project both train and test set using this same instance of TSNE. Otherwise the two projections will not be comparable, as they are learning two different projections.

3 Experiments

In this chapter we evaluate the effectiveness of applying FixMatch to our dataset. The chapter is split into three main parts. The first aims to make the model reject unintended input as described in section 2.2. The second demonstrates the capabilities of FixMatch on our imbalanced and domain-specific dataset. The third and final part applies our FixMatch implementation to datasets benchmarked in the original FixMatch paper, and compares the results. Since the original FixMatch paper uses error rate defined as the inverse of accuracy, we apply the same metric with the exception of using macro accuracy, which is important for fairly evaluating on unevenly distributed datasets. Macro accuracy calculates the average accuracy within each class, and then averages those averages. Micro accuracy, on the other hand, would simply calculate the accuracy without considering class labels.

All experiments run on a desktop PC with two Intel Xeon Gold 6234 @3.30GHz CPU, 128 GB RAM @2933MHz, and one NVIDIA Quadro P4000 GPU with 8 GB VRAM.

3.1 One-class classification (OCC)

3.1.1 Data and Models

We are provided with an proprietary dataset containing 100k images of ship engine components (a *scavenge port inspection*). There's 3 different subsets of this data. The first is a training set of 6k images, whose samples are entirely marked "has-class", and are further divided into 8 classes depicting different parts of an engine inspection. The remaining images are unlabeled, of which we have asked an expert annotator to perform binary labeling of 1k samples at random using LabelStudio [25]. We therefore have a 6k "has-class" training dataset, a 100k mixed unlabeled dataset and a 1k mixed "has-class"/"no-class" test set. The practical goal is to accurately remove the "no-class" images from the large mixed dataset. However, this section only uses the train- and test set for evaluation purposes.

In the other experiments described later in this section a ResNet-50 architecture were used for the models. For this experiment we found it necessary to use a different and much larger model compared to the following experiments. ResNet50 failed to generalize for OCC, achieving around 50% accuracy on a binary classification task on the test set. We achieve non-trivial results upon using the model from openai/clip-vit-large-patch14 [23]. We're able to use a much larger model, since we can compute the embeddings once and store them on disk, since no augmentation is performed (in contrast to FixMatch).

3.1.2 Results

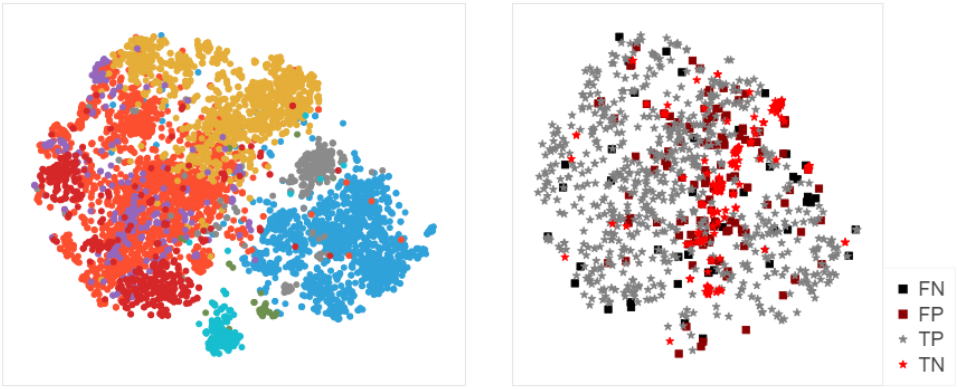
As discussed in section 2.2, we can estimate the proportion of polluted data (no-class data) in an unlabeled collection of images by comparing the negative rate (NR) of the development set[1] with the negative rate of the unlabeled dataset. Across all methods of OCC, we find the negative rate of the development set to be approximately that of the train set ($5.13\%_{\pm 0.76\%} \approx 5\%$).

Table 3.1: The negative rate for each method of one-class classification, along with the 1k sample estimate

	OSVMClassifier	ECODClassifier	MeanClassifier	KMeanClassifier	ClassMeanClassifier	true
NR	14%	12%	12%	9%	8%	24%

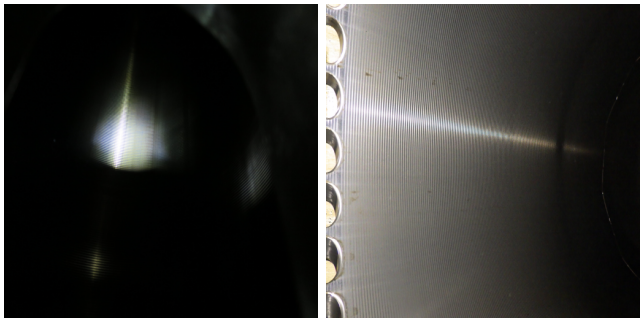
Once the theoretical TNR is computed in Table 3.1, we also had an expert annotate images in order to establish an empirical sample TNR. We see a large gap between the true sample estimate in Table 3.1. We discuss some of the potential explanations of this in section 2.2.

[1] Incidentally, the NR of the development set is also the false NR by definition of not including no-class data



(a) train TSNE

(b) test TSNE



(c) FN : Falsely predicted in-class

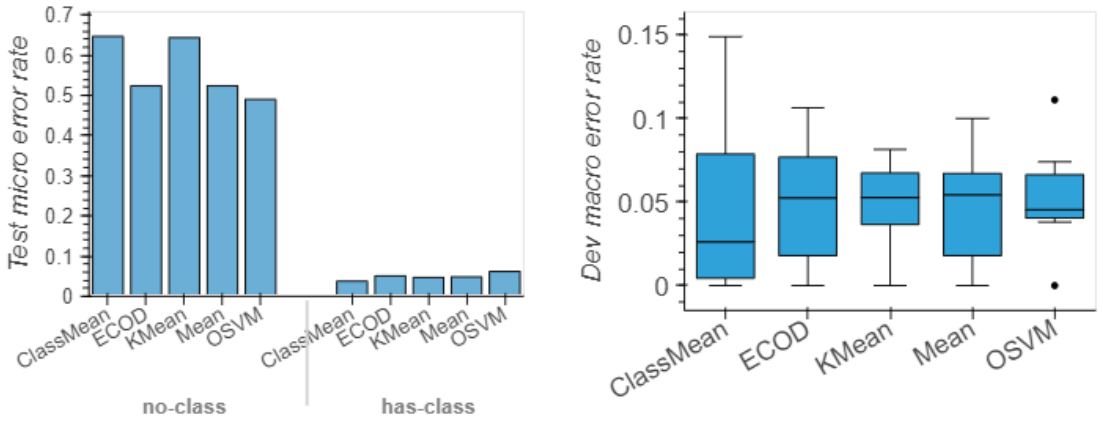
(d) TP : Liner image

Figure 3.1: TSNE applied to *clip-vit-large-patch14* embeddings of the 3.1a train set and 3.1b test set. For 3.1a each color represents one of the 8 classes. For 3.1b the "no-class" annotations are red, and incorrect predictions are squares □ (using an OSVM model). The second row demonstrates the two rightmost samples from 3.1b, one FN ■ and one TP ⋆: The model detects similar features/surfaces, but a human deems 3.1c too dark to be in-class while 3.1d is ok

To bring evidence to some of these claims, we cherrypicked an example in Figure 3.2. Both photographs in Figure 3.1c and Figure 3.1d are in the rightmost domain of the blue group (*Liner walls*) in Figure 3.1a, which indicates that the model views both images as closely related. Despite a human annotator deeming this image too dark to be a useful liner wall image, it still has glimpses of the same textured grey surface, which is likely what the model picks up on and why it deems the images so similar.

20 more random example images are shown in the appendix B, of which some show the same tendencies of similar visual patterns in unintended settings as the example described above. These examples also show that the model easily identifies clearly out of class images, such as screenshots or human fingers, but struggles with other components which resemble the classes. To differentiate against these similar type of components, a supervised method with more annotated training data may be more successful. This is also explained from the plot in Figure 3.1b, of which the large portion of TN ⋆ are located in the empty "no-mans-land" of Figure 3.1a, and often contain images such as screenshots or non-engine components (e.g. people). The FP ■ are scattered in denser areas, indicating a closer similarity to the training set. Finally, sometimes the model is just plain wrong, failing to classify images as in-class, despite correctly classifying an adjacent image in the embedded space.

When examining the predictive capabilities of each model, we examine two dimensions. The first in Figure 3.2a is the overall test *micro* accuracy for both no-class and has-class annotations. We use micro accuracy since our annotations does not include *which* class an image belongs to, only that it does belong. In contrast, Figure 3.2b contains the development set with all 8 labels represented, but it only contains has-class images



(a) (b)

Figure 3.2: The error rate of each OCC classifier model. Figure 3.2a depicts the test set micro error rate for no-class and has-class labels, whereas 3.2b only evaluates on the development set (has-class) to present the span of inaccuracies across the 8 different classes, along with the macro class accuracy (boxplot median).

(given that they're taken from the orignal labeled dataset). The Figure 3.2b is a boxplot of the error rates $(1 - TPR)$ for each of the 8 labels, meaning the median is across the 8 labels, and the min and max show the highest and lowest inaccuracies within a label. With this in mind, it is curious to find the ClassMean classifier to have the largest error rate span, given that it is trying to represent each class equally. An explanation for this might be that some classes have a larger variance, and therefore assigning a single class mean to each class is unfairly under-representing those classes. This could explain why the KMean classifier achieves the lowest worst-case error rate at around 8%, since it tries to cover as much of the space irrespective of class. No model is clearly superior to any other, and even the naive Mean classifier performs comparably to the much more complex ECOD classifier.

When examining results of the annotated test set, the results are similar. The methods which have a lower no-class error rate generally have a higher has-class error rate. Across all models, the has-class group is clearly the easiest to detect. Surprisingly, the no-class error rate is much larger than 50%, while the has-class is below 10% error rate (close to the 5% parameter during training). This may be explained by the class imbalance (24% NR in Table 3.1) along with previous findings, that no-class labels are much more nuanced and can seem closely related to in-class labeled images. Some images are even of the label, but gets discarded due to the image condition and setting (brightness, perspective or other concerns of an expert). Again, see appendix B for visual examples.

To summarize, the overall purpose of this project is to categorise images from a *scavenge port inspection*. A technician uploads a diverse bunch of images, however, we don't want any no-class images to get into the unsupervised training data. Currently our model achieves this goal very well with a high has-class accuracy (precision). However for model deployment, it might be better to lower the threshold so the model accepts images that would've ended up in the no-label class, as false negatives has a huge cost, requiring a new inspection to be performed. This could be alleviated by deploying the model as a phone app, providing the user immediate feedback.

With respect to models, we find that good semantic embeddings are far more important than the distance metric/heuristic implementation of the models, since ResNet50/ImageNet embeddings failed to produce nontrivial results, meaning results that surpassed the strategy of guessing at random. One may benefit further from fine-tuning the embeddings on the classification task, or perform supervised training on a larger dataset instead of OCC, but we leave this for future works.

3.2 Engine parts classification

We want to examine how the FixMatch approach would work on the task of classifying engine parts on a dataset with a large amount of unlabeled data. In this section we describe our setup compared to the FixMatch paper. We investigate differences in performance compared to amount of data used for training and test how the parameters for FixMatch is associated in an ablation study (section 3.2.4).

3.2.1 Training setup

This section describes the setup for this experiment as well as for section 3.3. We summarize points of ambiguity in the FixMatch paper, and state our interpretation, along with other differences in setup. Details can also be found in

We train all models using the hyper-parameter configuration described in the FixMatch paper. For detailed definitions of the first three, see section 2.1.

- $\lambda = 1$, unlabeled loss weight
- $\tau = 0.95$, pseudo label confidence threshold
- $\mu = 7$, Ratio of unlabeled:labeled training data
- $\eta = 0.03$, learning rate
- $\beta = 0.9$, nesterov gradient descent momentum
- $B = 64$, supervised batch size
- $K = 2^{20}$, maximum training steps (Where each batch counts as 64 steps)
- $L_2 = 0.0005$, L_2-regularisation/weight decay

Below we list the differences between our implementation and training when compared to the paper

- We fine-tune the last layer of a 25B parameter ResNet50 model using ImageNet weights for the remaining layers. The paper uses a 1.5B parameter wide ResNet-28-2 model trained from scratch. This is a significant difference.
- We sample the unlabeled and test dataset prior to training, which is undefined in the FixMatch paper.
- We implement early stopping where training stops if the loss does not decrease within five epochs or 1000 samples, whichever is higher
- We do not apply a learning rate scheduler, partly because we use early stopping and partly because of uncertainty around implementation details.
- We do not apply EMA, since we found it computationally infeasible to collect the EMA averages every batch
- We did not double the L_2 decay for CIFAR-100 (0.001).

3.2.2 Data

We have the same dataset as the previous section 3.1.1: a dataset with 100k images of ship engine components. A subset of the images (6k) are labelled as one or more categories, with 8 categories in total. We are assuming an image to have only one category, and dropping any which do not. The labels are highly unbalanced, since the largest category consists of 2000+ images and the smallest category of less that 100 images. Prior to any analyses, we create a 80/20 train/test split, and store the IDs of the test split to avoid accidentally leaking test data. We choose a relatively high test split ratio due to our small amount of labeled data. The train set is further divided into 80/20 train/development splits, which is used during development. All plots use the test set if not otherwise specified. For a precise breakdown of the data distribution, see appendix D.

3.2.3 Model and preprocessing

We have chosen the ResNet50 architecture [12] as the model architecture for this project. ResNet is a popular network that is often used for computer vision tasks. Its input layer takes an image of size 224x224 pixels, followed by 49 convolutional layers and one fully connected layer at the end. We replace and train the final classification layer of the weights "IMAGENET1K_V2"[2] which have been pretrained on the ImageNet dataset [9]. Due to hardware and time constraints, we chose not to use a larger transformer model.

Since the weights expect a certain data distribution and shape, we use the PyTorch IMAGENET1K_V2 preprocessor, which resizes to 232x232, followed by a crop to the model input size of 224x224. It also standard-scales all channels to have ≈ 0.45 mean and ≈ 0.225 standard deviation[3].

The training follows the approach by FixMatch [24] using RandAugment as a strong augmentation method. Since our dataset is imbalanced, we weight our cross-entropy-loss function by the inverse frequency of each class, and use (1 - macro accuracy) as our evaluation metric (lower is better).

3.2.4 Ablation study

Here we will go through the two ablations studies and their results.

Data amount

In our examination of the effectiveness of the FixMatch method, we compare it to a traditional supervised fine-tuning when varying the training data size. We run a total of 6 experiments running each of these 5 times to take randomness of the dataset samples and training into account. We tested 3 different sample sizes of supervised data; ≈150 samples in total (48 in largest class, 1 in the smallest), ≈500 samples in total (161 in largest class, 4 in smallest) and ≈1,000 samples in total (323 in largest class, 8 in smallest). For each sample size, we run the experiment purely supervised ($\mu = 0$ = unsupervised/supervised ratio) and with contribution from unlabeled data ($\mu = 7$).

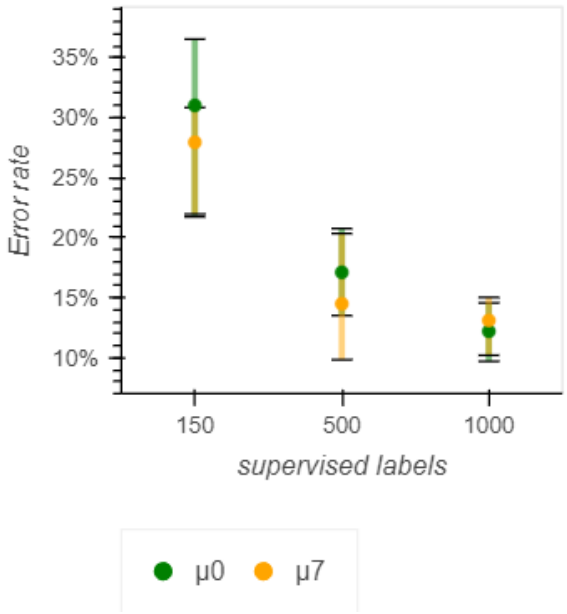


Figure 3.3: Error rate as data volumes increase, split by supervised ($\mu 0$) vs FixMatch ($\mu 7$). Error-bars indicate the min/max value across 5 runs.

[2] https://pytorch.org/vision/0.16/models/generated/torchvision.models.resnet50.html
[3] https://pytorch.org/vision/0.16/_modules/torchvision/models/resnet.html#ResNet50_Weights

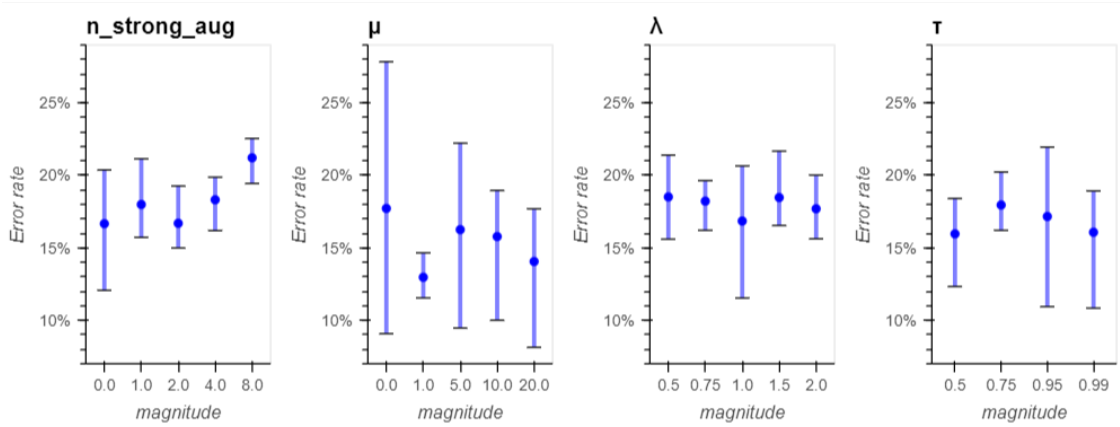


Figure 3.4: Different hyperparameters effect on the model's average error rate. Error-bars indicate the min/max value across 5 runs of 500 labels.

We see from the results in Figure 3.3 that 150 labels were way too few for us to achieve any reasonable performance. As expected as we increase the amount of labels ending a 1,000 we get a better performance. For 150 labels, using the FixMatch method produced a lower variability compared to purely supervised, however, this trend reversed for 500 labels. In all but the last case, we find the average error to be just barely lower for FixMatch, whereas for the last case of 1000 labels the results are completely overlapping. This backs up the findings from the FixMatch paper, that effects are the most significant in the extremely sparse data domain of only a few labels per class. The effects are, however, not as dramatic as in the FixMatch paper. This could be attributed to two things. The first is the fact that we only train a small part of the network, so less data is required to train the model to begin with. Secondly, we apply a pre-trained model for the remaining part of the network, which is already robust since it has been trained on large volumes of data. We therefore expect the point of diminishing returns to arrive sooner when applying FixMatch to fine-tuned models compared to when training from scratch or when performing a full fine-tuning of all layers. Due to time constraints and limited hardware (GPU VRAM), we were unable to compare to a training from scratch.

Hyperparameters

We wanted to see how the different parameters of FixMatch affected the error rate of the model. Therefore we did an ablation study of the four parameters; μ the ratio of unlabeled and labeled samples used for training, λ the weight the model puts on the loss from unsupervised samples, τ the threshold whether to say a pseudo label is confident or not hence we include or discard the unsupervised loss for that sample, and as the last parameter *number of augmentations* for the strong augmented samples. All ablation results are run on subsamplings of 500 supervised training labels (7 times that for unsupervised), and each configuration is run 5 times on the same 5 random samplings for different magnitudes of λ, τ & $\#aug$ (μ requires more data and hence cannot be run on the same subset). Results are presented in Figure 3.4. Variability should therefore be comparable across magnitudes and parameters if model variability is assumed to be insignificant.

Results for μ in the second column of Figure 3.4 indicates that a minor ratio of unlabaled data $magnitude = 1$ reliably produce better performance, while the remaining ratios jumps a lot inbetween runs. The purely supervised method varies the most, where increasing proportions of unlabeled data slightly lower the loss on average, albeit with large variance. The fact that exactly $\mu = 1$ produces the lowest variance could be attributed to the non-linear relationship of increasing μ, as increasing values leads to increased stability of the unlabeled data distribution, but it also leads to an indirect increased weighting of the supervised to unsupervised loss due an increased sample imbalance between the two. $\mu = 1$ may be a balance point of increased overall representation of the classes while not dominating the loss.

Another interesting find is the first column, which shows the impact of increasing number of strong aug-

mentations for the unlabeled data. We observe that no augmentations produce comparable results to two augmentations, albeit again with large variance. Furthermore, for the remaining plots we observe high variability which obscures any potential small differences in error rate. This indicates that FixMatch neither improves nor hinders the training for our dataset, which is also reflected in the ablation study on supervised labels in Figure 3.3. We did experience complete model collapse around $\lambda = 8$ (not shown) and a sharp increase in loss at $\lambda = 4$ (not shown) on the development set and at $n\_strong\_aug = 8$ (Figure 3.4), so, as expected, parameters significantly deviating from those proposed in FixMatch will degrade performance.

3.3 FixMatch datasets comparison

We wanted to test how close our implementation of FixMatch was to the one proposed in the paper [24]. Hence we tested our models performance on the same datasets as the authors evaluate FixMatch on. These datasets are described in the following Data section. As mentioned previously, the experiment setup from the previous experiment applies to this experiment as well, see section 3.2.1 for details on setup.

In order to evaluate our FixMatch implementation, we evaluate it using the same metrics of the original authors and compare our results. Note that for this section we do not use a development set as we are not performing any analysis or running any hyperparameter search for the model.

3.3.1 Data

CIFAR-10 and CIFAR-100

The two datasets were made as a part of a research paper by the Canadian Institute For Advanced Research (CIFAR), *"the CIFAR-10 set has 6000 examples of each of 10 classes and the CIFAR-100 set has 600 examples of each of 100 non-overlapping classes."* [16]. That gives a total of 60.000 images for each datasets. Since the dataset doesn't contain an unlabeled set of images, we use the training set for the unlabeled split. All images are in size 32x32 pixels.

Note that this makes it impossible for us to recreate the 10000 label subset of CIFAR-100 with the specified parameters of the paper, since we require 70000 unsupervised samples, which is why we have "-" in Table 3.2.

SVHN

The Street View House Numbers (SVHN) dataset has 10 classes, one for each digit in a house number [18]. Like our dataset, it contains imbalanced classes with the largest class being 13861 samples (the number "1") and the smallest 4659 samples (the number "9"). However, from the writing of FixMatch, it seems they enforce a balanced sample, so we do the same. In total the dataset consists of 73,257 digits for training 99,289 digits (73257 train, 26032 test) and 73,257 extra digits which we use for unsupervised training. These are less difficult and meant to be used as extra training data [18]. The format we used are MNIST-like with digits in 32x32 pixels images.

STL-10

This datasets has 10 classes with 1,300 images per class (500 train, 800 test) and besides that a 100,000 images without labels for unsupervised learning [7] which follow a slightly broader data distribution compared to the other splits. All images are labeled examples of ImageNet. All images are 96x96 pixels.

3.3.2 Model

The original FixMatch paper used a Wide ResNet architecture ranging from 1.5M to 5.9M parameters. PyTorch did not have an implementation of this, so we again used the 25.5M parameter ResNet50 model as detailed in section 3.2.3, with the same hyperparameter configuration. It is worth re-iterating that we use pre-trained weights and fine-tune the last classification layer, whereas FixMatch trains from scratch

3.3.3 Results

As expected, more data reduces the error rate across all datasets, and generally FixMatch seems to improve performance with the exception of the SVHN dataset from Figure 3.5. One would expect the opposite, that it may perform worse on the STL-10 dataset whose training-set is more diverse, whereas on the SVHN dataset the samples are supposedly easier. It may be that the model is over-fitting to the easier distribution, since up to 7/8ths of the loss is attributed to the unsupervised training with the current setup, however this theory could be countered by the fact that the unsupervised data is strongly augmented, significantly complicating the distribution.

When looking at the results alongside the FixMatch implementations in Table 3.2 we see that our implementation has losses up to 30x higher than that of the original authors, specifically when looking at SVHN dataset. This is likely due to the fact that classifying street view house numbers is far from the task ResNet originally was trained, that being object classification. For this task, color and textures are irrelevant, and only the shapes and silhouettes are significant. The high-level ImageNet features are therefore likely little use, and it may even be that using earlier and lower-level convolutional layers may be more benefitial to model performance. The claim of poor representation is confirmed when looking at the mean standard deviation across the 2048 embedding dimensions when looking at a sample of 1000 images. SVHN has the smallest standard deviation by a significant margin of 0.12, compared to 0.16, 0.18 and 0.20 for STL10, CIFAR10 and CIFAR100 respectively. SVHN therefore occupies a much smaller span of the possible embedding space compared to the remaining dataset. This is likely due to the difference in domains, as the remaining datasets all rely heavily or entirely on the nouns of WordNet (these and the following claims are described further in appendix E). Our ResNet50 model is even pretrained on Imagenet, a superset of the STL10 dataset, which likely explains the extremely low error rate; We may be indirectly evaluating on samples which the pretrained model had once been trained on, and our results are therefore not a fair depiction of performance of unseen data. The CIFAR datasets are collected using the same methodology as ImageNet, but do not directly have any overlap, which may indicate a more fair representation. Interestingly, CIFAR10 performs worse than our dataset, despite being seemingly closer in domain. At 4000 samples it achieves 20% error rate, whereas the error rate is closer to 15% at 1000 samples for our dataset. Even accounting for CIFAR10 having 2 more classes, thats still a large gap considering the differences in training data. This is likely due to our dataset having a much higher resolution (over 224x224) whereas CIFAR10 is 32x32 [16]. Since we're using the ResNet50 transformer, we're upscaling these to 224x224, which may also introduce upscaling artifacts causing lower signal-to-noise ratio.

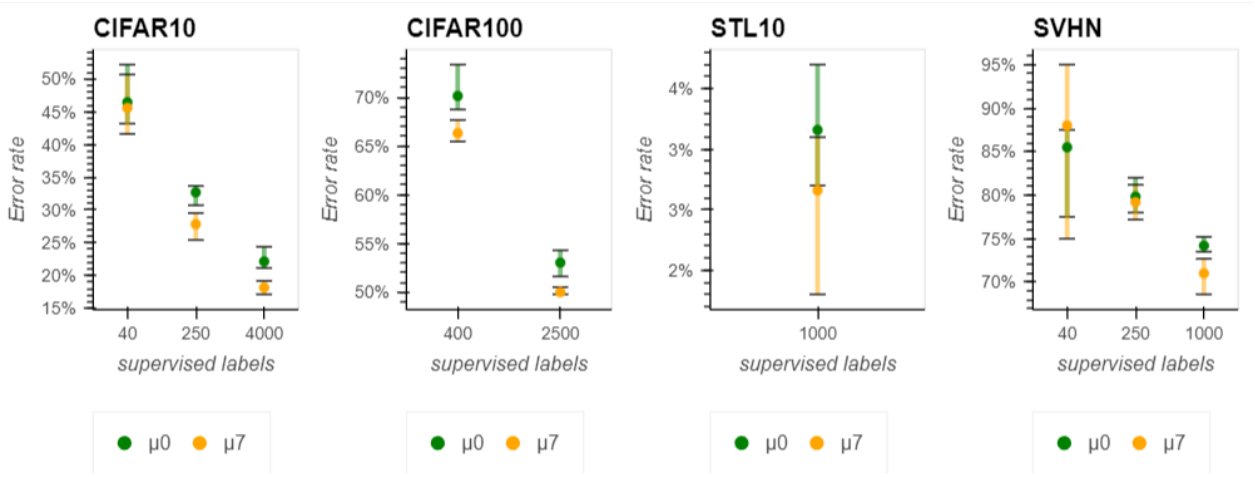


Figure 3.5: Error rate for increasing number of training samples, split by 1) dataset and 2) supervised ($\mu 0$) vs semi-supervised ($\mu 7$)

Table 3.2: Recreation of FixMatch dataset subsampling. Header denotes the number of *supervised* training samples. The first two rows are from Table 2 in FixMatch [24]. CIFAR-100 has "-", since the dataset has 60000 images, whereas $\mu = 7$ and n=10000 requires 80000 images

	CIFAR-10			CIFAR-100			SVHN			STL-10
Method \ n labels	40	250	4000	400	2500	10000	40	250	1000	1000
Supervised baseline (Ours)	$46.43_{\pm 5.73}$	$32.68_{\pm 1.97}$	$22.14_{\pm 2.23}$	$70.13_{\pm 3.22}$	$53.03_{\pm 1.41}$	-	$85.50_{\pm 8.00}$	$79.84_{\pm 2.16}$	$74.20_{\pm 1.00}$	$3.16_{\pm 0.54}$
FixMatch (Ours, RA)	$45.57_{\pm 5.10}$	$27.83_{\pm 2.40}$	$18.16_{\pm 1.04}$	$66.32_{\pm 1.35}$	$49.99_{\pm 0.51}$	-	$88.00_{\pm 13.00}$	$79.20_{\pm 2.00}$	$71.02_{\pm 2.42}$	$2.66_{\pm 0.86}$
FixMatch (Paper, RA)	$13.81_{\pm 3.37}$	$5.07_{\pm 0.65}$	$4.26_{\pm 0.05}$	$48.85_{\pm 1.75}$	$28.29_{\pm 0.11}$	$22.60_{\pm 0.12}$	$3.96_{\pm 2.17}$	$2.48_{\pm 0.38}$	$2.28_{\pm 0.11}$	$7.98_{\pm 1.50}$
FixMatch (Paper, CTA)	$11.39_{\pm 3.35}$	$5.07_{\pm 0.33}$	$4.31_{\pm 0.15}$	$49.95_{\pm 3.01}$	$28.64_{\pm 0.24}$	$23.18_{\pm 0.11}$	$7.65_{\pm 7.65}$	$2.64_{\pm 0.64}$	$2.36_{\pm 0.19}$	$5.17_{\pm 0.63}$

4 Conclusion

For the task of rejecting unintended input, we find promising results once we apply more complex embeddings. We achieve a high true positives rate of $\approx 95\%$ ("has-class"), though having a much lower true negative rate ("no-class") around $\approx 55\%$, although this trade-off can be adjusted using hyper-parameters depending on user requirements. To achieve truly great results in both dimensions, a deep, end-to-end method should be examined. We also achieve good performance for classifying engine components, however, the FixMatch method does not yield consistent improvements compared to traditional supervised fine-tuning. This may be due to differences in data distributions between the labeled and unlabeled datasets. It is more likely that the choice of fine-tuned model embeddings has a larger effect given the findings of the "no-class" experiment, meaning one may find benefits from exchanging the compute cost of on-the-fly image augmentation for static but more expensive and higher quality transformer image embeddings. Future work may examine the combination of the two above stated approaches: applying the "no-class" classifier as a filtering step prior to using the unlabeled data for the FixMatch method.

Bibliography

[1] D. Berthelot, N. Carlini, E. D. Cubuk, A. Kurakin, K. Sohn, H. Zhang, and C. Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring, 2020.

[2] D. Berthelot, N. Carlini, I. Goodfellow, N. Papernot, A. Oliver, and C. Raffel. Mixmatch: A holistic approach to semi-supervised learning, 2019.

[3] M. A. Berwo, Y. Fang, J. Mahmood, N. Yang, Z. Liu, and Y. Li. Faeccd-cnet: Fast automotive engine components crack detection and classification using convnet on images. *Applied Sciences*, 12(19), 2022.

[4] Á. Casado-García, C. Domínguez, J. Heras, E. Mata, and V. Pascual. The benefits of close-domain fine-tuning for table detection in document images. In X. Bai, D. Karatzas, and D. Lopresti, editors, *Document Analysis Systems*, pages 199–215, Cham, 2020. Springer International Publishing.

[5] J. Y. Chun and M. I. Naylor. Analysis of images of cylinders, 2021. Bachelor thesis at IT University of Copenhagen.

[6] A. Coates. Stl-10 dataset — cs.stanford.edu. `https://cs.stanford.edu/~acoates/stl10/`. [Accessed 25-11-2023].

[7] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[8] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. Randaugment: Practical data augmentation with no separate search. *CoRR*, abs/1909.13719, 2019.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[11] A. Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Incorporated, 2019.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[13] S. S. Khan and M. G. Madden. One-class classification: taxonomy of study and review of techniques. *The Knowledge Engineering Review*, 29(3):345–374, 2014.

[14] H. E. Kim, A. Cosa-Linan, N. Santhanam, M. Jannesari, M. E. Maros, and T. Ganslandt. Transfer learning for medical image classification: a literature review. *BMC Medical Imaging*, 2022.

[15] S. kit learn. 2.7. novelty and outlier detection, 2023.

[16] A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

[17] Z. Li, Y. Zhao, X. Hu, N. Botta, C. Ionescu, and G. Chen. Ecod: Unsupervised outlier detection using empirical cumulative distribution functions. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2022.

[18] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 01 2011.

[19] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization.

[20] Y. Ouali, C. Hudelot, and M. Tami. An overview of deep semi-supervised learning. *CoRR*, abs/2006.05278, 2020.

[21] G. Pang, C. Shen, L. Cao, and A. V. D. Hengel. Deep learning for anomaly detection: A review. *ACM Comput. Surv.*, 54(2), mar 2021.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[23] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision, 2021.

[24] K. Sohn, D. Berthelot, C. Li, Z. Zhang, N. Carlini, E. D. Cubuk, A. Kurakin, H. Zhang, and C. Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *CoRR*, abs/2001.07685, 2020.

[25] M. Tkachenko, M. Malyuk, A. Holmanyuk, and N. Liubimov. Label Studio: Data labeling software, 2020-2022. Open source software available from https://github.com/heartexlabs/label-studio.

[26] A. Torralba, R. Fergus, and W. T. Freeman. Tiny images. 2007.

[27] P. University. About wordnet. `https://wordnet.princeton.edu/`, 2010. [Accessed 25-11-2023].

[28] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

[29] Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

A RandAugment augmentations

Table A.1: Parameters provided in each implementation of RandAugment, *Ours* implemented in PyTorch and *FixMatch* implemented from scratch. When empty, the method is parameter free. Note that PyTorch may also negate parameters for some methods, so the parameter ranges are not directly comparable. For details, see `https://pytorch.org/vision/0.16/_modules/torchvision/transforms/autoaugment.html#RandAugment`.

Method	**Ours**	**FixMatchs**
Identity		
ShearX	[0.015; 0.295]	[−0.3, 0.3]
ShearY	[0.015; 0.295]	[−0.3, 0.3]
TranslateX	[0.023; 0.43]	[−0.3, 0.3]
TranslateY	[0.023; 0.43]	[−0.3, 0.3]
Rotate	[0.015; 0.295]	[−30, 30]
Brightness	[0.055; 0.855]	[5%; 95%]
Color	[0.055; 0.855]	[5%; 95%]
Contrast	[0.055; 0.855]	[5%; 95%]
Sharpness	[0.055; 0.855]	[5%; 95%]
Posterize	[8 bits; 4 bits]	[8 bits; 4 bits]
Solarize	[242.3, 12.8]	[0, 1]
AutoContrast		
Equalize		

B OCC prediction examples

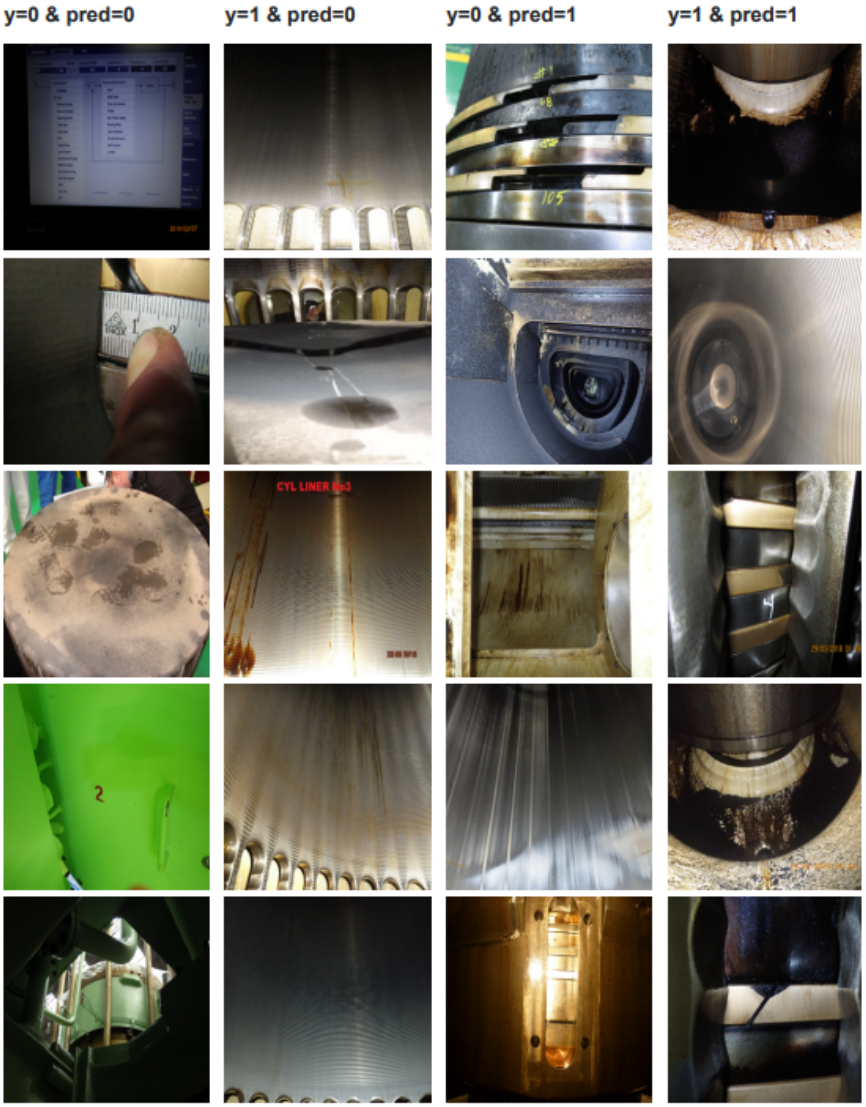


Figure B.1: Examples of correct and incorrect OCC predictions when compared to the annotated label (y). The two middle columns denote incorrect predictions. Notice the first and last image of the third column, which both depict cylinder rings (which the model captures) but from unconventional perspectives (which is why they were rejected by the annotator)

C Number of RandAugmentations

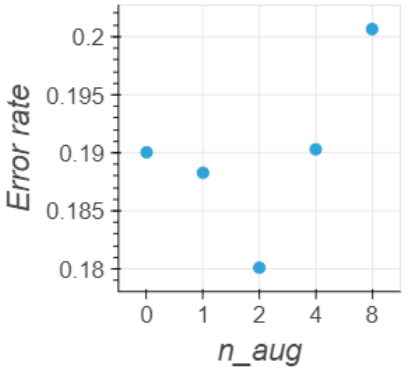


Figure C.1: Ablation study on the development set demonstrating the error rate (inverse accuracy) on the engine component dataset for increasing number of random augmentations. We note that 2 is a minimum, and use this for future experiments.

D Exact class label distribution

Table D.1: The exact number of samples for each split. All labels include both the train and test set.

Category	Total	Train	Test	150	500	1000
Piston Ring Overview	2359	1376	376	48	161	323
Liner	1792	1078	291	38	126	253
Single Piston Ring	1504	857	235	30	100	201
Topland	659	376	110	13	44	88
Skirt	332	201	55	7	23	47
Piston top	330	187	48	6	22	44
Scavange box	254	136	34	4	16	32
Piston rod	61	36	8	1	4	8
Total	7291	4247	1157	147	496	996

E WordNet source

This section explains how STL10, CIFAR and ImageNet all relate to WordNet. WordNet is a database containing nouns among other things [27]. These nouns are used as the basis for Google image search for ImageNet [10], which STL10 is a subset of [6]. The CIFAR datasets are a subset of the Tiny Images dataset [16], which is also built from searches of WordNet nouns [26].