Synbols: Probing Learning Algorithms with Synthetic Datasets

Alexandre Lacoste[1], **Pau Rodríguez**[1], Frédéric Branchaud-Charron[1], Parmida Atighehchian[1], Massimo Caccia[1,2], Issam Laradji[1], Alexandre Drouin[1], Matt Craddock[1], Laurent Charlin[2], and David Vázquez[1]

[1]Element AI
{allac, pau.rodriguez, frederic.branchaud-charron, parmida, massimo.caccia, issam.laradji, adrouin, matt.craddock, dvazquez}@elementai.com
[2]Mila, Université de Montréal
{massimo.p.caccia, lcharlin}@gmail.com

Abstract

Progress in the field of machine learning has been fueled by the introduction of benchmark datasets pushing the limits of existing algorithms. Enabling the design of datasets to test specific properties and failure modes of learning algorithms is thus a problem of high interest, as it has a direct impact on innovation in the field. In this sense, we introduce Synbols — Synthetic Symbols — a tool for rapidly generating new datasets with a rich composition of latent features rendered in low resolution images. Synbols leverages the large amount of symbols available in the Unicode standard and the wide range of artistic font provided by the open font community. Our tool's high-level interface provides a language for rapidly generating new distributions on the latent features, including various types of textures and occlusions. To showcase the versatility of Synbols, we use it to dissect the limitations and flaws in standard learning algorithms in various learning setups including supervised learning, active learning, out of distribution generalization, unsupervised representation learning, and object counting.

1 Introduction

Open access to new datasets has been a hallmark of machine learning progress. Perhaps the most iconic example is ImageNet [11], which spurred important improvements in a variety of convolutional architectures and training methods. However, obtaining state-of-the-art performances on ImageNet can take up to 2 weeks of training with a single GPU [58]. While it is beneficial to evaluate our methods on real-world large-scale datasets, relying on and requiring massive computation cycles is limiting and even contributes to biasing the problems and methods we develop:

- **Slow iteration cycles**: Waiting weeks for experimental results reduces our ability to explore and gather insights about our methods and data.
- **Low accessibility**: It creates disparities, especially for researchers and organizations with limited computation and hardware budgets.
- **Poor exploration**: Our research is biased toward fast methods.

Preprint. Under review.

- **Climate change impact**: Recent analyses [55, 34] conclude that the greenhouse gases emitted from training very large-scale models, such as transformers, can be equivalent to 10 years' worth of individual emissions.[1]

A common alternative is to use smaller-scale datasets, but their value to develop and debug powerful methods is limited. For example, image classification datasets such as MNIST [37], SVHN [44] or CIFAR [30] each contain less than 100,000 low-resolution (32×32 pixels) images which enables short learning epochs. However, these datasets provide a single task and can prevent insightful model comparison since, e.g., modern learning models obtain above 99% accuracy on MNIST.

In addition to computational hurdles, *fixed datasets* limit our ability to explore non-i.i.d. learning paradigms including out-of-distribution generalization, continual learning and, causal inference. I.e., the algorithms can latch onto spurious correlations, leading to highly detrimental consequences when the evaluation set comes from a different distribution [3]. Similarly, learning disentangled representations requires non i.i.d. data for both training and properly evaluating [40, 26, 53]. This raises the need for good synthetic datasets with a wide range of latent features.

We introduce Synbols[2], an easy to use dataset generator with a rich composition of latent features for lower-resolution images. Synbols uses Pycairo, a 2D vector graphics library, to render UTF-8 symbols with a variety of fonts and patterns. Fig. 1 showcases generated examples from several attributes (§ 2 provides a complete discussion). To expose the versatility of Synbols, we probe the behavior of popular algorithms in various sub-fields of our community. Our contributions are:

- Synbols: a dataset generator with a rich latent feature space that is easy to extend and provides low resolution images for quick iteration times (§ 2).
- Experiments probing the behavior of popular learning algorithms in various machine-learning settings including: the robustness of supervised learning and unsupervised representation-learning approaches w.r.t. changes in latent-data attributes (§ 3.1 and 3.4) and to particular out-of-distribution patterns (§ 3.2), the efficacy of different strategies and uncertainty calibration in active learning (§ 3.3), and the effect of training losses for object counting (§ 3.5).

Related Work Creating datasets for exploring particular aspects of methods is a common practice. Variants of MNIST are numerous [43, 48], and some such as colored MNIST [3, 2] enable proofs of concept, but they are fixed. dSprites [42] and 3D Shapes [27] offer simple variants of 2D and 3D objects in 64×64 resolution, but the latent space is still limited. To satisfy the need for a richer latent space, many works leverage 3D rendering engines to create interesting datasets (e.g, CLEVR [25], Synthia [51], CARLA [12]). This is closer to what we propose with Synbols, but their minimal viable resolution usually significantly exceeds 64×64, which requires large-scale computation.

2 Generator

The objective of the generator is to provide means of playing with a wide variety of concepts in the latent space while keeping the resolution of the image low. At the same time, we provide a high level interface that makes it easy to create new datasets with diverse properties.

2.1 Attributes

Font Diversity (Fig. 1a) To obtain a variety of writing styles, we leverage the large quantity of artistic work invested in creating the pool of open source fonts available online. When creating a new font, an artist aims to achieve a style that is diversified while maintaining the readability of the underlying symbols in a fairly low resolution. This is perfectly suited for our objective. While many fonts are available under commercial license, fonts.google.com provides a repository of open source fonts.[3] This leaves us with more than 1,000 fonts available across a variety of languages. To

[1] This analysis includes hyperparameter search and is based on the average United States energy mix, largerly composed of coal. The transition to renewable-energy can reduce these numbers by a factor of 50 [34]. Also, new developments in machine learning can help mitigate climate change [50] at a scale larger than the emissions coming from training learning algorithms.

[2] https://github.com/ElementAI/synbols

[3] Each font family is available under the Open Font License or Apache 2.

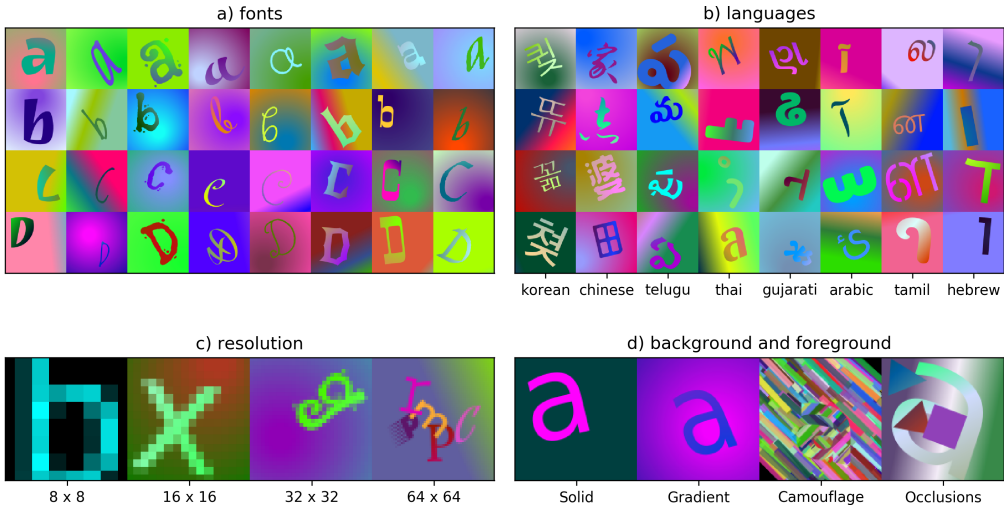

Figure 1: Generated symbols for different fonts, alphabets, resolutions, and appearances.

filter out the fonts that are almost identical to each other, we used the confusion matrix from a Wide ResNet [59] trained to distinguish all the fonts on a large version of the dataset.

Different Languages (Fig. 1b) The collection of fonts available at fonts.google.com spans 28 languages. After filtering out languages with less than 10 fonts or rendering issues, we are left with 14 languages. Interestingly, the Korean alphabet (Hangul) contains 11,172 syllables, each of which is a specific symbol composed of 2 to 5 letters from an alphabet of 35 letters, arranged in a variable 2 dimensional structure. This can be used to test the ability of an algorithm to discriminate across a large amount of classes and to evaluate how it leverages the compositional structure of the symbol.

Resolution (Fig. 1c) The number of pixels per image is an important trade-off when building a dataset. High resolution provides the opportunity to encode more concepts and provide a wider range of challenges for learning algorithms. However, it comes with a higher computational cost and slower iteration rate. In Fig. 1c-left, we show that we can use a resolution as low as 8×8 and still obtain readable characters.[4] Next, the 16×16 resolution is big enough to render most features of Synbols without affecting readability, provided that the range of scaling factors is not too large. Most experiments are conducted in 32×32 pixels, a resolution comparable to most small resolution datasets (e.g., MNIST, SVHN, CIFAR). Finally, a resolution of 64×64 is enough to generate useful datasets for segmentation, detection and counting.

Background and Foreground (Fig. 1d) The texture of the background and foreground is flexible and can be defined as needed. The default behavior is to use a variety of shades (linear or radial with multiple colors). To explore the robustness of unsupervised representation learning, we have defined a *Camouflage* mode, where the color distribution of independent pixels within an image is the same for background and foreground. This leaves the change of texture as the only hint for detecting the symbol. Finally, we provide a mechanism for adding a variety of occlusions or distractions using any UTF-8 symbol. By default, we use *circle*, *square*, and *triangle* with various scales and translations.

Attributes Each symbol has a variety of attributes that affect the rendering. Most of them are exposed in a Python dict format for later usage such as a variety of supervised prediction tasks, as some hidden ground truth for evaluating the algorithm, or simply a way to split the dataset in a non i.i.d. fashion. The list of attributes includes *character*, *font*, *language*, *translation*, *scale*, *rotation*, *bold*, and *italic*. We also provide a segmentation mask for each symbol in the image.

2.2 Interface

The main purpose of the generator is to make it easy to define new datasets. Hence, we provide a high-level interface with default distributions for each attribute. Then, specific attributes can be redefined in Python as follows:

[4] For readability, scale and rotation are fixed and solid colors are used for the background and foreground

```
sampler = attribute_sampler(scale=0.5, translation=lambda: np.random.uniform(-2, 2, 2))
dataset = dataset_generator(sampler, n_samples=10000)
```

The first line fixes the scale of every symbol to 50% and redefines the distribution of the x and y translation to be uniform between -2 and 2, (instead of -1, 1). The second line constructs a Python generator for sampling 10,000 images. Because of the modified translation, the resulting dataset will have some symbols partially cropped by the image border. We will see in Sec. 3.3 how this can be use to study the brittleness of some active learning algorithms.

Using this approach, one can easily redefine any attributes independently with a new distribution. When distributions need to be defined jointly, for studying e.g. latent causal structures, we provide a slightly more flexible interface.

Datasets are stored in HDF5 or NumPy format and contain images, symbols masks, and all attributes. The default (train, valid, test) partition is also stored to help reproducibility. We also use a combination of Docker, Git versioning, and random seed fixing to make sure that the same dataset can be recreated even if it is not stored.

3 Experiments

To expose the versatility of the dataset generator, we explore the behavior of learning algorithms across a variety of machine-learning paradigms. For most experiments, we aim to find a setup under which some algorithms fail while others are more resilient. For a concise presentation of results, we break each subsection into *goal*, *methodology* and *discussion* with a short introduction to provide context to the experiment. All results are obtained using a (train, valid, test) partition of size ratio (60%, 20%, 20%). Adam [28] is used to train all models, and the learning rate is selected using a validation set. Average and standard deviation are reported over 3 runs with different random seeds. More details for reproducibility are included in App. B and the benchmarking code will be open sourced.

3.1 Supervised Learning

While MNIST has been the hallmark of small scale image classification, it offers very minor challenges and it cannot showcase the strength of new learning algorithms. Other datasets such as SVHN offer more diversity but still lack discriminative power. Synbols provides a varying degree of challenges exposing the strengths and weaknesses of learning algorithms.

Goal: Showcase the various levels of challenges for Synbols datasets. Establish reference points of common baselines for future comparison.

Methodology: We generate the **Synbols Default** dataset by sampling a lower case English character with a font uniformly selected from a collection of 888 fonts. The translation, scale, rotation, bold, and italic are selected to have high variance without affecting the readability of the symbol. We increase the difficulty level by applying the **Camouflage** feature shown in Fig. 1d. The **Korean** dataset is obtained by sampling uniformly from the first 1000 symbols. Finally we explore font classification using the **Less Variations** dataset, which removes the bold and italic features, and reduces the variations of scale and rotation. See App. B.1 for previews of the datasets.

Backbones: We compare 7 models of increasing complexity. Unless specified, all models end with global average pooling (GAP) and a linear layer. **MLP**: A three-layer MLP with hidden size 256 and leaky ReLU non-linearities (72k parameters). **Conv4 Flat**: A four-layer CNN with 64 channels per layer with flattening instead of GAP, as described by [54] (138k parameters). **Conv4 GAP**: A variant of Conv4 with GAP at the output (112k parameters). **Resnet-12**: A residual network with 12 layers [20, 45] (8M parameters). **WRN**: A wide residual network with 28 layers and a channel multiplier of 4 [59] (5.8M parameters). **Resnet12+** and **WRN+** were trained with data augmentation consisting of random rotations, translation, shear, scaling, and color jitter. More details in App. B.1.

Discussion: Table 1 shows the experiment results. The MNIST column exposes the lack of discriminative power of this dataset, where an MLP obtains 98.5% accuracy. SVHN offers a greater challenge, but the default version of Synbols is even harder. To estimate the aleatoric noise of the data, we experiment on a larger dataset using 1 million samples. In App. B.1, we present an error analysis

Dataset	MNIST	SVHN	Synbols Default		Camouflage	Korean	Less Variations
Label Set	10 Digits	10 Digits	26 Symbols	26 Symbols	26 Symbols	1000 Symbols	888 Fonts
Dataset Size	60k	100k	100k	1M	100k	100k	100k
MLP	98.51 ±0.02	85.04 ±0.21	34.23 ±0.36	74.41 ±0.47	14.70 ±0.20	0.09 ±0.01	0.12 ±0.02
Conv-4 Flat	99.32 ±0.06	90.74 ±0.27	79.13 ±0.34	91.75 ±0.02	72.69 ±0.24	5.86 ±2.97	0.45 ±0.11
Conv-4 GAP	99.06 ±0.07	88.32 ±0.21	77.84 ±0.48	93.14 ±0.10	65.04 ±0.03	23.94 ±3.71	6.29 ±0.69
ResNet-12	**99.70** ±0.05	96.38 ±0.03	96.11 ±0.05	98.91 ±0.00	94.70 ±0.00	97.35 ±0.40	45.93 ±0.46
ResNet-12+	**99.73** ±0.05	97.19 ±0.04	97.49 ±0.09	99.30 ±0.01	96.90 ±0.05	98.78 ±0.05	67.20 ±0.63
WRN-28-4	99.64 ±0.06	96.07 ±0.07	95.07 ±0.16	99.07 ±0.01	92.43 ±0.12	97.85 ±0.11	36.29 ±0.50
WRN-28-4+	**99.74** ±0.03	**97.30** ±0.05	**97.97** ±0.04	**99.37** ±0.01	**97.39** ±0.08	**99.28** ±0.05	**80.35** ±0.08

Table 1: **Supervised learning:** Accuracy of various models on supervised classification tasks. Results within 2 standard deviations of the highest accuracy are in **bold**.

to further understand the dataset. On the more challenging versions of the dataset (i.e., Camouflage, Korean, Less Variations) the weaker baselines are often unable to perform. Accuracy on the Korean dataset are surprisingly high, this is in part due to the lower diversity of font for this language. For font classification, where there is less variation, we see that data augmentation is highly effective. The total training time on datasets of size 100k is about 3 minutes for most models (including ResNet-12) on a Tesla V100 GPU. For WRN+ the training time goes up to 16 minutes, and about 10× longer for datasets of size 1M.

3.2 Out of Distribution Generalization

The supervised learning paradigm usually assumes that the training and testing sets are sampled independently and from the same distribution (i.i.d.). In practice, an algorithm may be deployed in an environment very different from the training set and have no guarantee to behave well. To improve this, our community developed various types of inductive biases [60, 15], including architectures with built-in invariances [36], data augmentation [36, 31, 10], and more recently, robustness to spurious correlations [3, 1, 32]. However, the evaluation of such properties on out-of-distribution datasets is very sporadic. Here we propose to leverage Synbols to peek at some of those properties.

Goal: Evaluate the inductive bias of common learning algorithms by changing the latent factor distributions between the training, validation, and tests sets.

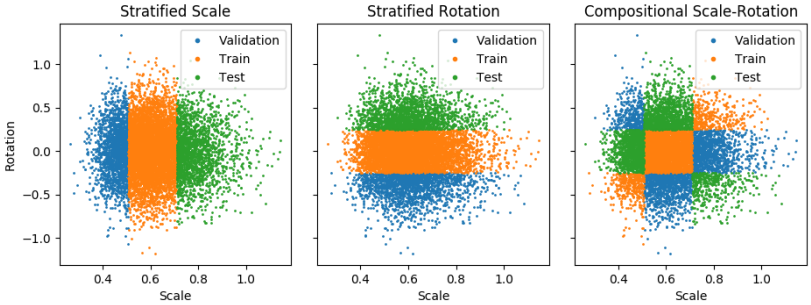

Figure 2: Example of the different types of split using scale and rotation

Dataset		Synbols Default					Less Variations
Partition	i.i.d.	Stratified Font	Stratified Scale	Stratified Rotation	Compositional Rot-Scale	Stratified x-Translation	Stratified Char
MLP	34.23 ±.36	33.33 ±.15	**35.83** ±.21	22.98 ±.60	28.23 ±.66	21.19 ±.30	0.15 ±.01
Conv-4 Flat	79.13 ±.34	**78.69** ±.13	**81.05** ±.26	62.03 ±.30	77.88 ±.22	58.83 ±.89	0.46 ±.08
Conv-4 GAP	77.84 ±.48	**77.54** ±.17	71.73 ±.46	57.59 ±.37	73.31 ±.35	73.95 ±.52	3.36 ±.44
Resnet-12	96.11 ±.05	**95.71** ±.26	93.62 ±.22	84.47 ±.41	94.23 ±.11	95.13 ±.17	1.11 ±.29
Resnet-12+	97.49 ±.09	96.44 ±.10	96.23 ±.08	92.86 ±.07	96.67 ±.03	**97.38** ±.07	11.72 ±.24
WRN-28-4	95.07 ±.16	**94.96** ±.12	92.69 ±.13	83.02 ±.51	93.27 ±.20	94.31 ±.03	9.62 ±.17
WRN-28-4+	97.97 ±.04	96.87 ±.08	96.95 ±.03	93.64 ±.42	97.26 ±.12	**97.89** ±.01	12.86 ±.71

Table 2: **Out of Distribution:** Results reporting accuracy on various train, valid, test partitions. Results in blue are the reference point for the Synbols Default dataset, results in red have a drop of more than 5% absolute accuracy compared to reference point, and **bold** results have a drop of less than 0.5% absolute accuracy compared to reference point. Refer to Table 1 for the reference point of font classification on the Less Variations dataset.

Methodology: We use the Synbols Default dataset (Sec. 3.1) and we partition the train, validation, and test sets using different latent factors. **Stratified** Partitions uses the first and last 20 percentiles of a continuous latent factor as the validation and test set respectively, leaving the remaining samples for training (Fig. 2). For discrete attributes, we randomly partition the different values. Solving such a challenge is only possible if there is a built-in invariance in the architecture. We also propose **Compositional** Partitions to evaluate the ability of an algorithm to compose different concepts. This is done by combining two stratified partitions such that there are large holes in the joint distribution while making sure that the marginals keep a comparable distribution (Fig. 2-right).

Discussion: Results from Table 2 show several drops in performance when changing the distribution of some latent factors compared to the i.i.d. reference point highlighted in blue. Most algorithms are barely affected in the **Stratified Font** column. This means that character recognition is robust to fonts that are out of distribution.[5] On the other hand, all architectures are very sensitive to the **Stratified Rotation** partition, suffering from a systematic drop of at least 5% absolute accuracy compared to the i.i.d. evaluation. While data augmentation helps it still suffers from a drop of 4% accuracy. Scale is also a factor that affects the performance, but interestingly, some architectures are much more resilient than others.[6] Next, the **Compositional Rot-Scale** partition shows less performance drop than its stratified counterparts. This hints that neural networks are capable of some form of compositional learning. Moving on to the **Stratified x-Translation** partition, we see that many algorithms not affected by this intervention and retains its original performance. This is expected since convolution with global average pooling is invariant to translation. However, the MLP and Conv-4 Flat do not share this property and they are significantly affected. Finally, we observe an important drop of performance on font prediction when it is evaluated on characters that were not seen during training. This hints that many font classification results in Table 1 are memorizing pairs of symbols and font.

3.3 Active Learning

Instead of using a large supervised dataset, active learning algorithms aim to request labels for a small subset of unlabeled data. This is done using a query function seeking the most informative samples. A common query function is the predictive entropy (P-Entropy) [52]. However, a more principled method uses the mutual information between the model uncertainty and the prediction uncertainty of a given sample. This can be efficiently approximated using BALD [23]. However, in practice, it is common to observe comparable predictive performances between BALD and P-Entropy [6, 17]. We hypothesize that this may come from the lack of aleatoric uncertainty in common datasets. We thus explore a variety of noise sources, ranging from pixel noise to ablative noise in the latent space. We also explore the importance of uncertainty calibration using Kull et al. [33]. This is done by learning a Dirichlet distribution as an extra layer after the logits, while keeping the rest of the network fixed. To look at the optimistic scenario, we learn it using the full validation set with all the labels (after the network has converged using its labeled set).

Goal: Showcase the brittleness of P-Entropy on some types of noise. Search for cases where BALD may fail and see where calibrated uncertainty can help.

Methodology: We compare BALD and P-Entropy to random sampling. These methods are evaluated on different variations of Synbols Default. The **Label Noise** dataset introduces uniform noise in labels with probability 0.1. The **Pixel Noise** dataset adds $\epsilon \sim \mathcal{N}(0, 0.7)$ to each pixel in 50% of the images and clips back the pixel value to the range $(0, 1)$. The **10% missing** dataset omits the symbol with probability 0.1. In the **Cropped** dataset, translations are sampled uniformly between -2 and 2, yielding many symbols that are partly cropped. Finally, the **20% Occluded** dataset draws a large shape, occluding part of the symbol with probability 0.2. For concise results, we report the Negative Log Likelihood after labeling 10% of the unlabeled set. Performance at 5% and 20% labeling size can be found in Sec. B.3, along with implementation details and previews of all datasets.

Discussion: In Table 3, we recover the result where P-Entropy is comparable to BALD using CIFAR 10 and the No Noise dataset. Interestingly, the results highlighted in red show that P-Entropy will perform systematically worse than random when some images have unreadable symbols while BALD keeps its competitive advantage against Random. When training on 10% Missing, we found that P-Entropy selected 68% of it's queries from the set of images with omitted symbols vs 4% for

[5] This is not too surprising since there is a large amount of fonts and many share similarities.

[6] The test partition uses larger scale which tend to be easier to classify, leading to higher accuracy.

	CIFAR10	No Noise	Label Noise	Pixel Noise	10% Missing	Out of the Box	20% Occluded
BALD	**.59** ±.03	**.57** ±.02	**1.68** ±.06	**.56** ±.01	**1.43** ±.06	**1.98** ±.05	**1.50** ±.05
P-Entropy	**.59** ±.01	**.56** ±.01	1.75 ±.06	.62 ±.02	2.10 ±.08	2.26 ±.08	1.79 ±.07
Random	.66 ±.04	.72 ±.03	1.81 ±.04	.73 ±.02	1.54 ±.02	2.10 ±.10	1.64 ±.05
BALD Calibrated	**.58** ±.01	**.55** ±.03	**1.67** ±.05	**.57** ±.05	**1.38** ±.05	**1.95** ±.08	**1.51** ±.04
Entropy Calibrated	**.57** ±.02	.60 ±.02	**1.67** ±.04	.60 ±.03	2.11 ±.13	2.22 ±.05	1.74 ±.04

Table 3: **Active Learning** results reporting Negative Log Likelihood on test set after labeling 10% of the unlabeled training set. Results worse than random sampling are shown in red and results within 1 standard deviation of the best result are shown in **bold**.

BALD. We also found that label noise and pixel noise are not appropriate to highlight this failure mode. Finally, calibrating the uncertainty on a validation set did not significantly improve our results.

3.4 Unsupervised Representation Learning

Unsupervised representation learning leverages unlabeled images to find a semantically meaningful representation that can be used on a downstream task. For example, a Variational Auto-Encoder (**VAE**) [29] tries to find the most compressed representation sufficient to reconstruct the original image. In Kingma and Welling [29], the decoder is a deterministic function of the latent factors with independent additive noise on each pixel. This inductive bias is convenient for MNIST where there is a very small amount of latent factors controlling the image. However, we will see that simply adding texture can be highly detrimental on the usefulness of the latent representation. The Hierarchical VAE (**HVAE**) [62] is more flexible, encoding local patterns in the lower level representation and global patterns in the highest levels. Instead of reconstructing the original image, **Deep InfoMax** [22] optimizes the mutual information between the representation and the original image. As an additional inductive bias, the global representation is optimized to be predictive of each location of the image.

Goal: Evaluate the robustness of representation learning algorithms to change of background.

Methodology: We generate 3 variants of the dataset where only the foreground and background change. To provide a larger foreground surface, we keep the Bold property always active and vary the scale slightly around 70%. The other properties follow the Synbols Default dataset. The **Solid Pattern** dataset uses a black and white contrast. The **Shades** dataset keeps the smooth gradient from Default Synbols. The **Camouflage** dataset contains many lines of random color with the orientation depending on whether it is in the foreground or the background. Samples from the dataset can be seen in Sec. B.4. All algorithms are then trained on 100k samples from each dataset using a 64-dimensional representation. Finally, to evaluate the quality of the representation on downstream tasks, we fix the encoder and use the training set for learning an MLP to perform classification of the 26 characters or the 888 fonts. For comparison purposes, we use the same backbone as Deep InfoMax for our VAE implementation and we also explore using ResNet-12 for a higher capacity VAE.

	Character Accuracy			Font Accuracy		
	Solid Pattern	Shades	Camouflage	Solid Pattern	Shades	Camouflage
Deep InfoMax	**89.32** ±0.30	12.53 ±0.80	**9.37** ± 4.11	**18.61** ±0.30	0.39 ±0.07	**0.36** ±0.19
VAE	74.61 ±0.41	39.98 ±4.14	7.74 ±0.79	4.16 ±0.02	0.46 ±0.09	0.20 ±0.01
HVAE (2 level)	80.05 ±1.03	47.76 ±2.38	8.05 ±0.21	3.81 ±0.18	0.53 ±0.17	0.20 ±0.01
VAE ResNet	81.84 ±0.25	52.63 ±0.22	6.41 ±0.24	6.23 ±0.26	0.66 ±0.05	0.19 ±0.02
HVAE (2 level) ResNet	79.92 ±0.02	**70.42** ±1.06	7.12 ±0.24	4.37 ±0.03	**1.05** ±0.06	0.21 ±0.05

Table 4: **Unsupervised Representation Learning:** Results reporting classification accuracy on 2 downstream tasks. Highest performance is in **bold** and red exposes unexpected results.

Discussion: While most algorithms perform relatively well on the Solid Pattern dataset, we can observe a significant drop in performance by simply applying shades to the foreground and background. When using the Camouflage dataset, all algorithms are only marginally better than random predictions. Given the difficulty of the task, this result could be expected, but recall that in Table 1, adding camouflage hardly affected the results of supervised learning algorithms. Deep InfoMax is often the highest performer, even against HVAE with a higher capacity backbone. However the performances on Camouflage are still very low with high variance indicating instabilities. Surprisingly, Deep InfoMax largely underperforms on the Shades dataset. This is likely due to the global structure of the gradient pattern, which competes with the symbol information in the limited size

	Fixed scale		Variable scale		Crowded
	Non-overlapping	Overlapping	Non-overlapping	Overlapping	
FCN8-L (MAE)	**0.53** ±0.04	**1.42** ±0.07	**0.41** ±0.05	**1.15** ±0.04	9.87 ±0.21
FCN8-D (MAE)	1.23 ±0.06	**1.41** ±0.10	1.27 ±0.04	1.37 ±0.08	**1.87** ±0.06
FCN8-L (GAME)	**2.13** ±0.10	**4.44** ±0.12	**0.66** ±0.08	**1.21** ±0.09	28.75 ±0.31
FCN8-D (GAME)	4.41 ±0.15	8.23 ±0.13	1.61 ±0.11	1.83 ±0.12	**18.76** ±0.25

Table 5: **Object counting** results reporting MAE and GAME for exploring the sensitivity to scale variations, object overlapping and crowdedness. **Bold** indicates best results within 2 standard deviation and red indicates failures.

representation. This unexpected result offers the opportunity to further investigate the behavior of Deep InfoMax and potentially discover a higher performing variant.

3.5 Object Counting

Object counting is the task of counting the number of objects of interest in an image. The task might also include localization where the goal is to identify the locations of the objects in the image. This task has important real-life applications such as ecological surveys [4] and cell counting [9]. The datasets used for this task [18, 4] are often labeled with point-level annotations where a single pixel is labeled for each object. There exists two main approaches for object counting: detection-based and density-based methods. Density-based methods [38, 39] transform the point-level annotations into a density map using a Gaussian kernel. Then they are trained using a least-squares objective to predict the density map. Detection-based methods first detect the object locations in the images and then count the number of detected instances. LCFCN [35] uses a fully convolutional neural network and a detection-based loss function that encourages the model to output a single blob per object.

Goal: We compare between an FCN8 [41] that uses the LCFCN loss and one that uses the density loss function. The comparison is based on their sensitivity to scale variations, object overlapping, and the density of the objects in the images.

Methodology: We generate 5 datasets consisting of 128×128 images with a varying number of floating English letters on a shaded background. With probability 0.7, a letter is generated as 'a', otherwise, a letter is uniformly sampled from the remaining English letters. We use mean absolute error (MAE) and grid average mean absolute error (GAME) [19] to measure how well the methods can count and localize symbol 'a'. We first generate a dataset with symbols of **Fixed Scale** and of **Variable Scale** where the number of symbols in each image is uniformly sampled between 3 and 10. These two datasets are further separated into **Non-Overlapping** and **Overlapping** subsets where images are said to overlap when the masks of any two symbols have at least one pixel that coincides. The **Variable Scale** dataset is used to evaluate the sensitivity of the methods to scaling where $\text{scale} \sim 0.1e^{0.5\epsilon}$, and $\epsilon \sim \mathcal{N}(0, 1)$, whereas the **Overlapping** dataset is used to evaluate how well the counting method performs under occlusions. The fifth dataset is **Crowded**, which evaluates the model's ability to count with a large density of objects in an image. It consists of images where the number of symbols, all with the same scale, is uniformly sampled between 30 and 50. Sample images from all datasets are shown in Sec. B.5.

Discussion: Table 5 reports the results of FCN8-D and FCN8-L, which are FCN8 trained with the density-based loss function [38] and the LCFCN loss function, respectively. For **Fixed Scale** and **Variable Scale**, the MAE and GAME results show that FCN8-L consistently outperforms FCN8-D. However, the gap between the results is smaller for **Fixed Scale**. This can be explained by the fact that FCN8-D uses a fixed size Gaussian kernel which assumes a fixed size and shape of the objects, allowing it to perform better for **Fixed Scale** than **Variable Scale**. FCN8-L has significantly better localization since it explicitly localizes the objects before counting them. Further, the performance of both methods degrades significantly when overlapping between symbols exists, which suggests room for improvement under occlusion. For the **Crowded** dataset, the heavy occlusions make FCN8-L not train well at all. In this setup, density-based methods such as FCN8-D are clear winners as they do not require localizing individual objects before counting them.

3.6 Few Shot Learning

The large amount of classes available in Synbols makes it particularly suited for episodic meta-learning, where small tasks are generated using e.g., 5 classes and 5 samples per class. In this setting, a learning algorithm has to meta-learn to solve tasks with very few samples [47]. Interestingly, the different attributes of a symbol also provide the opportunity to generate different kinds of tasks and to evaluate the capability of an algorithm to adapt to new types of tasks.

Metric learning approaches [57, 54, 45, 49] usually dominate the leader board for few-shot image classification. They find a representation such that a similarity function can perform well at predicting classes. Perhaps the most popular is **ProtoNet** [54], where the representations of all images in a class are averaged to obtain the class prototype. Next, the closest prototype to a test image's representation is used for predicting its label. **RelationNet** [56] learns a shared similarity metric to compare supports and queries in an episode. Class probabilities are obtained with softmax normalization. For more flexible adaptation we also consider initialization-based meta-learning algorithms. **MAML** [13] does so by meta-learning the initialization of a network such that it can perform well on a task after a small number of gradient steps e.g., 10.

Goal: Evaluate the capability of meta-learning algorithms to adapt to new types of tasks.

Methodology: We generate a dataset spanning the 14 available languages. For a more balanced dataset, we limit it to 200 symbols and 200 fonts per language. This leads to a total of 1099 symbols and 678 fonts. Next, we evaluate our ability to solve font prediction tasks when meta-training on char tasks vs meta-training on font tasks. We also perform the converse to evaluate our ability to solve symbol prediction tasks at meta-test time. All tasks consist of 5 classes with 5 examples per classes, and the classes used to generate tasks during meta-test phase were never seen during the meta-train phase or meta-validation phase.

Meta-Test	**Characters**		**Fonts**	
Meta-Train	**Characters**	**Fonts**	**Fonts**	**Characters**
ProtoNet	98.00 ±0.08	77.69 ±0.50	82.26 ±0.91	46.21 ±0.63
RelationNet	93.19 ±1.59	59.61 ±1.72	52.24 ±21.47	31.16 ±9.86
MAML	93.59 ±2.52	72.99 ±0.98	76.42 ±2.90	43.69 ±1.31

Table 6: **Few-Shot Learning:** Accuracy over 5 classes and 5 samples per class for different configurations of types of tasks.

Discussion: The first column of Table 6 exhibits the expected behavior of few-shot learning algorithms i.e., the accuracy is high and ProtoNet is leading the board. When looking at the other columns, we observe an important drop in accuracy when the Meta-Train set and the Meta-Test set don't share the same distribution of tasks.[7] This is to be expected, but the magnitude of the drop casts doubts over the re-usability of the current meta-learning algorithms. By highlighting this use-case, we hope to provide a way to measure progress in this direction. Our results are inline with the ones presented in recent work [8] which leverages the Synbols dataset to show that MAML trained on character classification tasks also incurs an important drop when tested on font classification tasks.

4 Conclusion

This work presented Synbols, a new tool for quickly generating datasets with a rich composition of latent features. We showed its versatility by reporting numerous findings across 5 different machine learning paradigms. Among these, we found that using prediction entropy for active learning can be highly detrimental if some images have insufficient information for predicting the label.

As future work, we intend to support video generation, visual question answering, and the possibility of using natural images for foreground and background. Other attributes such as symbol border, shadow, and texture can also be added to create even more precise and insightful diagnostics.

[7] Note that for a given task, the train and the test sets comes from the same distribution

Broader Impact

The introduction of benchmark new datasets has historically fueled progress in machine learning. However, recent large-scale datasets are immense, which makes ML research over-reliant on massive computation cycles. This biases research advances towards fast and computationally-intensive methods, leading to economic and environmental impacts. Economically, reliance on massive computation cycles creates disparities between researchers and organizations with limited computation and hardware budgets, versus those with more resources. With regard to the environment and climate change, recent analyses [55, 34] conclude that the greenhouse gases emitted from training very large-scale models, such as transformers, can be equivalent to 10 years' worth of individual emissions. While these impacts are not unique to ML research, they are reflective of systemic challenges that ML research could address by developing widely available, high-quality, and diverse datasets that mimic real-world concepts and are conscious of computational hurdles.

The Synbols synthetic dataset generator is designed to explore the behavior of learning algorithms and discover brittleness to certain configurations. It is expected to stimulate the improvement of core properties in vision algorithms:

- Identifiability of latent properties
- Reusability of machine learning models in new environments
- Robustness to changes in the data distribution
- Better performance on small datasets

We designed Synbols with diverse and flexible features (i.e., font diversity, different languages, lower resolution for POC, flexible texture of the background and foreground) and we demonstrated its versatility by reporting numerous findings across 5 different machine learning paradigms. Its characteristics address the economic and environmental challenges and we expect this tool to have a transversal impact on the field of machine vision with potential impact on the field of machine learning. Its broader impacts, both positive and negative, will be guided by the progress that it stimulates in the machine vision community, where potential applications range from autonomous weapons to climate change mitigation. Nevertheless, we hope that our work will help develop more robust and reliable machine learning algorithms while reducing the amount of greenhouse gas emissions from training by way of its smaller scale.

References

[1] Kartik Ahuja, Karthikeyan Shanmugam, Kush Varshney, and Amit Dhurandhar. Invariant risk minimization games. *arXiv preprint arXiv:2002.04692*, 2020.

[2] Jacob Andreas. Measuring compositionality in representation learning. *arXiv preprint arXiv:1902.07181*, 2019.

[3] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. Invariant risk minimization. *arXiv preprint arXiv:1907.02893*, 2019.

[4] Carlos Arteta, Victor Lempitsky, and Andrew Zisserman. Counting in the wild. *ECCV*, 2016.

[5] Parmida Atighehchian, Frederic Branchaud-Charron, Jan Freyberg, Rafael Pardinas, and Lorne Schell. Baal, a bayesian active learning library. https://github.com/ElementAI/baal/, 2019.

[6] William H. Beluch, Tim Genewein, Andreas Nürnberger, and Jan M. Köhler. The power of ensembles for active learning in image classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2018.

[8] Massimo Caccia, Pau Rodriguez, Oleksiy Ostapenko, Fabrice Normandin, Min Lin, Lucas Caccia, Issam Laradji, Irina Rish, Alexandre Lacoste, David Vazquez, et al. Online fast adaptation and knowledge accumulation: a new approach to continual learning. *CVPR2020 - Workshop on Continual Learning*, 2020. URL https://arxiv.org/abs/2003.05856.

[9] J. P. Cohen, G. Boucher, C. A. Glastonbury, H. Z. Lo, and Y. Bengio. Count-ception: Counting by Fully Convolutional Redundant Counting. *ICCV Workshops*, 2017.

[10] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 113–123, 2019.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE conference on computer vision and pattern recognition*, 2009.

[12] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, 2017.

[13] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[14] Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz, and Lawrence Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. *arXiv preprint arXiv:1903.10145*, 2019.

[15] Kunihiko Fukushima and Nobuaki Wake. Handwritten alphanumeric character recognition by the neocognitron. *IEEE transactions on Neural Networks*, 2(3):355–365, 1991.

[16] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[17] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

[18] Ricardo Guerrero, Beatriz Torre, Roberto Lopez, Saturnino Maldonado, and Daniel Onoro. Extremely overlapping vehicle counting. *IbPRIA*, 2015.

[19] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto Javier López-Sastre, Saturnino Maldonado-Bascón, and Daniel Oñoro-Rubio. Extremely overlapping vehicle counting. In *IbPRIA*, 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[21] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. *Iclr*, 2(5):6, 2017.

[22] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

[23] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. Bayesian active learning for classification and preference learning. *arXiv preprint arXiv:1112.5745*, 2011.

[24] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

[25] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[26] Ilyes Khemakhem, Diederik P Kingma, and Aapo Hyvärinen. Variational autoencoders and nonlinear ica: A unifying framework. *arXiv preprint arXiv:1907.04809*, 2019.

[27] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *arXiv preprint arXiv:1802.05983*, 2018.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[30] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, Canada, 2009.

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

[32] David Krueger, Ethan Caballero, Joern-Henrik Jacobsen, Amy Zhang, Jonathan Binas, Remi Le Priol, and Aaron Courville. Out-of-distribution generalization via risk extrapolation (rex). *arXiv preprint arXiv:2003.00688*, 2020.

[33] Meelis Kull, Miquel Perello Nieto, Markus Kängsepp, Telmo Silva Filho, Hao Song, and Peter Flach. Beyond temperature scaling: Obtaining well-calibrated multi-class probabilities with dirichlet calibration. In *Advances in Neural Information Processing Systems*, pages 12295–12305, 2019.

[34] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

[35] Issam H Laradji, Negar Rostamzadeh, Pedro O Pinheiro, David Vazquez, and Mark Schmidt. Where are the blobs: Counting by localization with point supervision. *ECCV*, 2018.

[36] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[37] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[38] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. *NIPS*, 2010.

[39] Yuhong Li, Xiaofan Zhang, and Deming Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. *CVPR*, 2018.

[40] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Rätsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

[41] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CVPR*, 2015.

[42] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[43] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[44] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[45] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018.

[46] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[47] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[48] David Rawlinson, Abdelrahman Ahmed, and Gideon Kowadlo. Sparse unsupervised capsules generalize better. *arXiv preprint arXiv:1804.06094*, 2018.

[49] Pau Rodríguez, Issam Laradji, Alexandre Drouin, and Alexandre Lacoste. Embedding propagation: Smoother manifold for few-shot classification. *arXiv preprint arXiv:2003.04151*, 2020.

[50] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

[51] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[52] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[53] Rui Shu, Yining Chen, Abhishek Kumar, Stefano Ermon, and Ben Poole. Weakly supervised disentanglement with guarantees. *arXiv preprint arXiv:1910.09772*, 2019.

[54] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[55] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[56] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[57] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[58] Yang You, Zhao Zhang, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Imagenet training in 24 minutes. *arXiv preprint arXiv:1709.05011*, 2017.

[59] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[60] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[61] Xiangyu Zhang, Jianhua Zou, Kaiming He, and Jian Sun. Accelerating very deep convolutional networks for classification and detection. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):1943–1955, 2015.

[62] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Learning hierarchical features from deep generative models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4091–4099. JMLR. org, 2017.

A Emissions and Energy Usage

Experiments were conducted using private infrastructure, with a carbon efficiency of 0.026 $kgCO_2eq/kWh$. A cumulative total of 23916 hours of computation was performed on hardware of type Tesla V100 (TDP of 300W). This includes all phases of the project including debugging, failed experiments and hyperparameter search.

Total emissions are estimated to be 186.54 $kgCO_2eq$ of which 1000 $kgCO_2eq$ were compensated using Gold Standard credits.

Estimations were conducted using the MachineLearning Impact calculator presented in Lacoste et al. [34].

B Extended Details

For each experiment of Section 3, we provide more details. Implementation details are described for reproducibility purposes, and samples for each datasets are presented. Also, when available, we provide extended results.

B.1 Supervised Learning

We provide implementation details for algorithms of Section 3.1 and present samples for each dataset in Figure 3. In addition, we present error analysis on the Default 1M dataset.

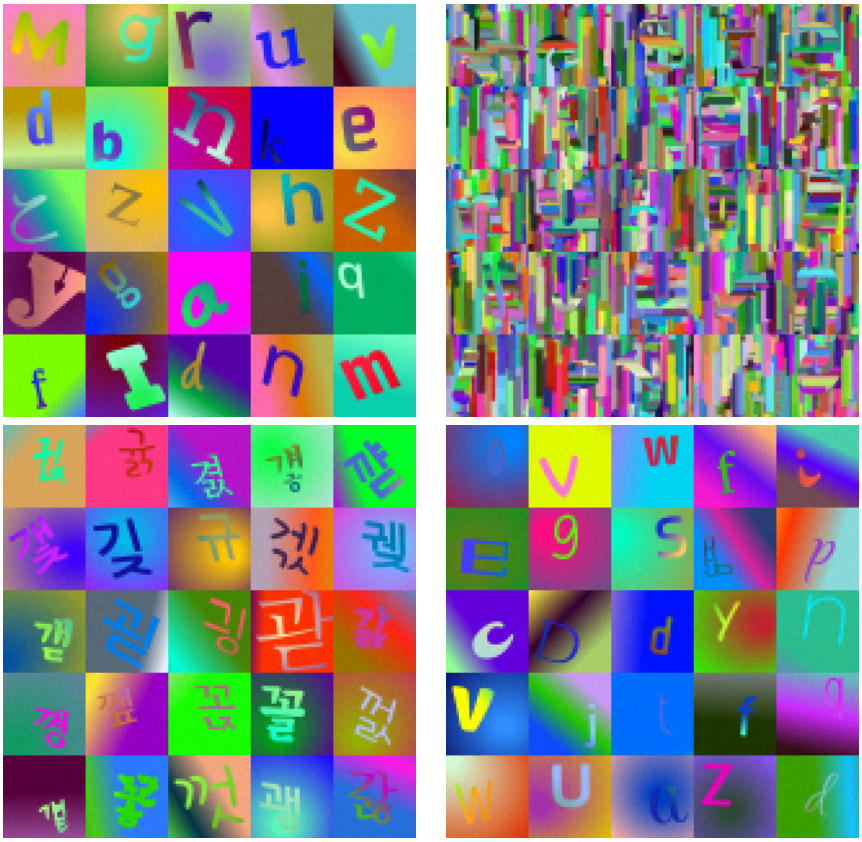

Figure 3: **Supervised Learning:**. Samples from datasets of Section 3.1. **Upper Left:** Default Symbol dataset. **Upper Right:** Camouflage dataset. **Lower left:** 1000 Korean Syllables dataset. **Lower Right:** Less Variations dataset.

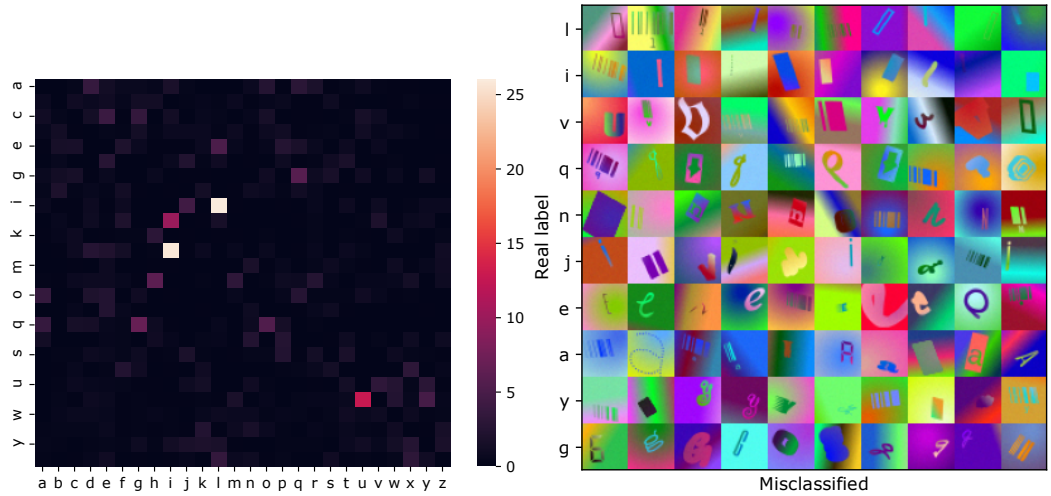

Figure 4: Error analysis on Synbols Default 1M with WRN-28-4+ architecture. **Left:** Confusion matrix with ground truth on the vertical axis and predictions on the horizontal axis. **Right:** Example of errors for the top 10 most confused letters.

B.1.1 Implementation Details

For all experiments we trained a three-layer MLP, a 4-layer CNN with 64 channels per layer and max pooling [54], a Resnet-12 [20, 45], and a WRN-28-4 [59].

As specified in [45], Resnet-12 is composed of four residual blocks with three 3×3 convolutional layers per block and output size 64, 128, 256, and 512, respectively. All the layers are followed by batch normalization [24] and ReLU non-linearities. Dropout of 0.1 is placed after the first and second convolution of each block. Max pooling is performed at the output of each block.

WRN is a 28-layer residual architecture [20]. In this architecture, the input is first fed to a 16 channel 3×3 convolution, followed by three groups of four residual blocks with two convolutions per block. Blocks in the same group share the same amount of channels: 32, 64, and 128, respectively. WRN-28-4 uses $\times4$ that amount of channels. The last two groups start with a stride of 2 to reduce the dimensionality of the input. A dropout of 0.1 is placed after the first convolution of each block.

The data augmentation consists of uniformly sampled affine deformations. Concretely, rotations are sampled in $[-10, 10]$ degrees. Shear is sampled uniformly in $[-2, 2]$ degrees. Vertical and horizontal translations are sampled uniformly in $[-10\%, 10\%]$ pixels. Scaling is sampled in $[0.9, 1.1]\times$ original scale.

All the models are trained with Adam [28] and a learning rate of $4 * 10^{-4}$ for 200 epochs with a batch size of 512. For WRN, the batch size is 128 and the learning rate is 10^{-4}. Early stopping is performed after 10 epochs without decrease in validation loss. Models are trained with mixed precision.[8]

B.1.2 Error Analysis

In Figure 4, we report a confusion matrix and error analysis for WRN-28-4+ on Synbols Default 1M. Errors are as expected: 'i' and 'l' are the most confused characters, followed by 'v' which often looks like 'u' on some fonts. We can also observe that many errors comes from bogus fonts, where 2 of them render a rectangle instead of the symbol and another original font prints bar-codes with a very small version of the character. These 3 fonts are now removed from Synbols.

[8] https://github.com/NVIDIA/apex

B.2 Out of Distribution Generalization

We provide implementation details for algorithms of Section 3.1. The dataset are the same as in Section 3.1 and samples are presented in Figure 3.

B.2.1 Implementation Details

B.3 Active learning

We provide implementation details for algorithms of Section 3.3 and present samples for each dataset in Figure 5. In addition, we present extended results.

B.3.1 Implementation Details

For all experiments, we train a VGG-16 [61] trained on Imagenet [11] until convergence on a validation set or a maximum of 10 epochs. Our active learning loop is implemented using the BaaL [5] library. We estimate epistemic uncertainty using MC-Dropout [16] using 20 samples. We also perform 20 Monte-Carlo sampling when computing the test performance. At each step, we label 100 images before resetting the weights to their original values as in Gal et al. [17]. For all reported results, we report the mean and standard deviation across 3 trials.

For calibration, we train the extra calibration layer for 10 epochs on the validation set. Following Kull et al. [33], we use the Adam optimizer and train two times, the latter with a reduced learning rate.

Figure 5: **Active Learning:**. Samples from datasets of Section 3.3. **Upper Left:** 50% pixel noise dataset. **Upper Right:** 10% Missing dataset. **Lower left:** Out of the Box dataset. **Lower Right:** 20% Occluded dataset.

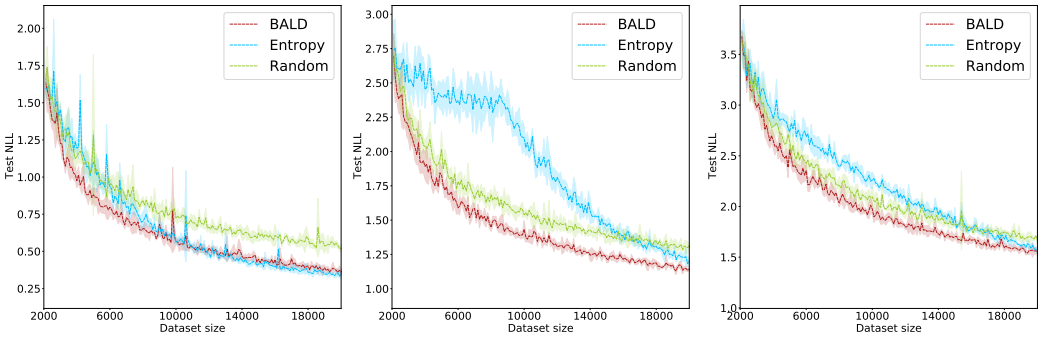

Figure 6: Active learning convergence speed for various datasets. **Left:** Original dataset., **Center:** 10% Missing dataset. **Right:** 20% Occluded dataset.

@5%	no-noise	label noise	pixel noise	10% missing	out of the box	20% occluded
BALD	**0.88** ±0.05	**2.04** ±0.11	**0.87** ±0.07	**1.83** ±0.06	2.56 ±0.05	**1.93** ±0.16
P-Entropy	1.10 ±0.22	2.19 ±0.09	1.04 ±0.11	2.40 ±0.10	2.74 ±0.06	2.37 ±0.05
Random	1.28 ±0.54	2.15 ±0.07	1.10 ±0.05	2.02 ±0.13	2.64 ±0.11	2.05 ±0.12
BALD calibrated	1.05 ±0.28	**2.07** ±0.07	**0.91** ±0.04	**1.75** ±0.08	**2.43** ±0.07	**1.92** ±0.09
Entropy calibrated	1.19 ±0.27	**2.03** ±0.06	1.19 ±0.29	2.55 ±0.36	2.82 ±0.09	2.41 ±0.08
@20%	**no-noise**	**label noise**	**pixel noise**	**10% missing**	**out of the box**	**20% occluded**
BALD	**0.36** ±0.01	**1.37** ±0.03	0.37 ±0.01	**1.14** ±0.01	**1.56** ±0.02	**1.21** ±0.01
P-Entropy	**0.34** ±0.03	1.48 ±0.03	**0.35** ±0.01	1.20 ±0.03	**1.57** ±0.03	**1.19** ±0.02
Random	0.52 ±0.03	1.51 ±0.01	0.51 ±0.02	1.31 ±0.01	1.67 ±0.04	1.37 ±0.05
BALD calibrated	**0.36** ±0.02	1.41 ±0.03	**0.35** ±0.02	**1.12** ±0.03	**1.54** ±0.03	**1.19** ±0.02
Entropy calibrated	**0.34** ±0.01	1.42 ±0.04	**0.35** ±0.01	1.22 ±0.05	1.61 ±0.04	**1.18** ±0.04

Table 7: **Active Learning** results reporting Negative Log Likelihood on test set after labeling at 5% and 20% of the unlabeled training set. Results worse than random sampling are shown in red and results within 1 standard deviation of the best result are shown in **bold**.

B.3.2 Extended Results

B.4 Unsupervised Representation Learning

We provide implementation details for algorithms of Section 3.4 and present samples for each dataset in Figure 7. In addition, we present qualitative results.

Figure 7: **Unsupervised Representation Learning:** Samples from datasets of Section 3.4 **Left:** Solid Pattern dataset. **Center:** Shades dataset. **Right:** Camouflage dataset.

B.4.1 Implementation Details

For Deep Infomax [22], we use the pytorch implementation in `https://github.com/DuaneNielsen/DeepInfomaxPytorch`.

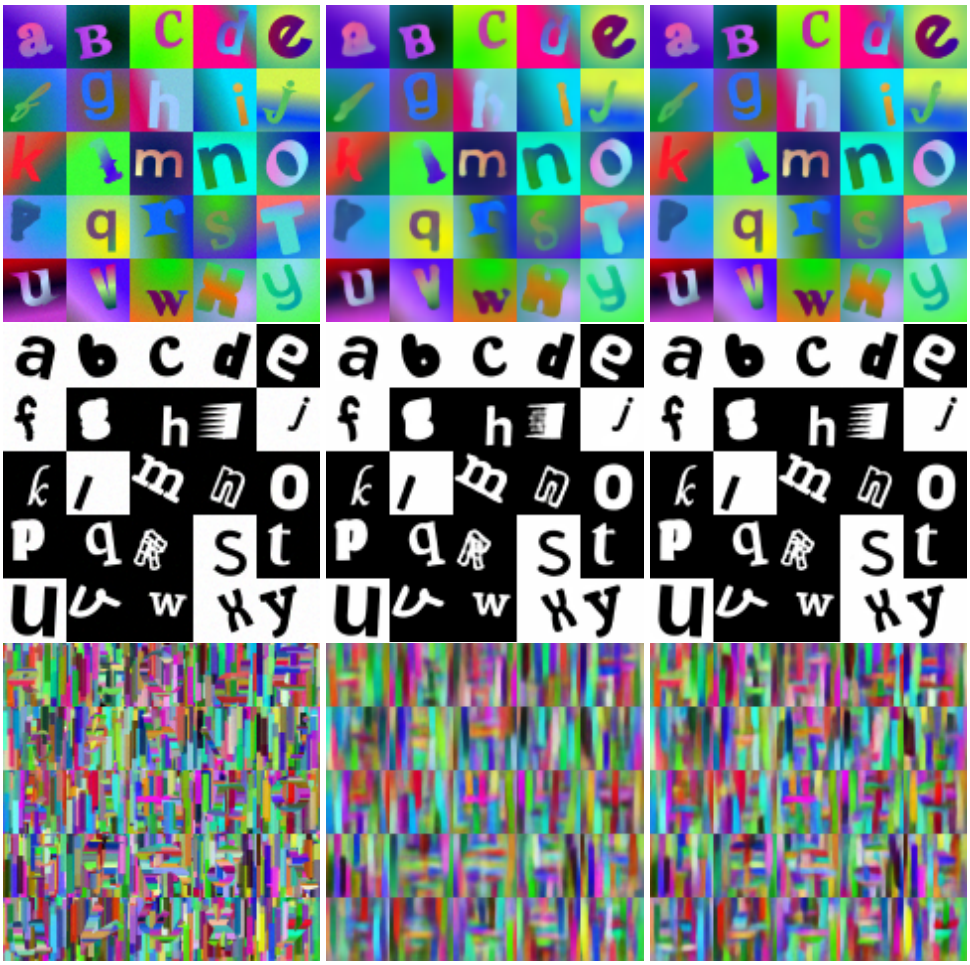

Figure 8: Generation examples. **Left:** original, **center:** VAE reconstruction, **right:** HVAE reconstruction. Best viewed in color.

For the VAE and HVAE, we use a residual architecture based on BigGAN [7]. The encoder consists of three residual blocks of two convolutions followed by average pooling. Each block has 128, 128, and 256 channels respectively. The decoder follows an inverse structure, with nearest neighbor upsampling at the input of each block. For the HVAE, we follow the structure of VLAE [62]. That is, a latent code is sampled from the output of the first block of the encoder and a linear projection of it is concatenated to the input of the last block of the decoder. For fair comparison we also provide the VAE and HVAE with the same encoder described by Hjelm et al. [22]. The latent code size is 64. Models are optimized by minimizing the beta-VAE loss function [21]:

$$\mathcal{L} = |\mathbf{x} - \hat{\mathbf{x}}| + \beta D_{KL}(q(\mathbf{z}|\mathbf{x})||p(\mathbf{z})), \tag{1}$$

where $\mathbf{x}$ are input images, $\hat{\mathbf{x}}$ are the VAE reconstructions, and $\mathbf{z}$ are latent codes. We set $\beta = 0.01$ for all experiments, and we use cyclical annealing [14].

For attribute classification on the latent codes, we train a 3-layer MLP with hidden-size 256 on the training latent codes. We report classification and regression scores on the validation dataset.

B.4.2 Qualitative results

In Figure 8, we provide reconstructions of the VAE and HVAE with the residual architecture. As can be seen there, the HVAE recovers sharper reconstructions. For the original dataset, the difference is more visible when the background and foreground share the same color (for instance, letter "K"). For

the black and white dataset, the difference can be found in details, for instance, see the "I", or the bottom of "M". For camouflage, the VAE was unable to generate some of the characters, while they can still be recognized in the HVAE reconstructions, see letter "R".

B.5 Object Counting

We provide implementation details in Section B.5.1 and present samples for the object counting datasets in Figure 9.

B.5.1 Implementation Details

FCN-L and FCN-D use an Imagenet-pretrained VGG16 FCN8 network [41]. The models are trained with a batch size of 1 for 100 epochs with ADAM [28] and a learning rate of 10^{-4}. The reported scores are on the test set which were obtained with early stopping on the validation set. The Gaussian sigma for FCN-D was chosen to be 1 and the weights of the coefficients for the 4 terms in the FCN-L loss were also chosen as 1.

B.6 Few-shot Learning

For this few-shot learning benchmark, we perform 5-way classification. We fix the support and query size to 5 and 15, respectively.

B.6.1 Implementation Details

For all experiments, we use a 4-layer convolutional neural network with 64 hidden units as commonly used in the few-shot literature [57, 54, 56]. All the methods are implemented using the PyTorch library [46], run on a single 12GB GPU and 8 CPUs .

For all methods, we use ADAM [28]. However, for MAML, in the inner-loop we use SGD. We use a batch size of one episode.

Hyper-parameter search For ProtoNet and RelationNet, we run a random search on the learning rates $\{0.005, 0.002, 0.001, 0.0005, 0.002, 0.0001\}$. We also search the patience hyper-parameter for early-stopping and for the learning rate schedule chosen in $\{10, 25, 50\}$ where the unit of measure is 500 training episodes. For MAML, we run random trials using the same learning rates as above, inner learning rates chosen from $\{0.5, 0.1, 0.01, 0.001, 0.0001\}$, and a number of inner updates chosen from $\{1, 2, 4, 8, 16\}$. Across the 4 settings presented in Table 6, we evaluate 14, 14, and 38 hyperparameter configurations for ProtoNet, RelationNet and MAML, respectively. For each configuration we repeat with 3 different random seeds. This results in a total of 34 days of compute to reproduce the current results.

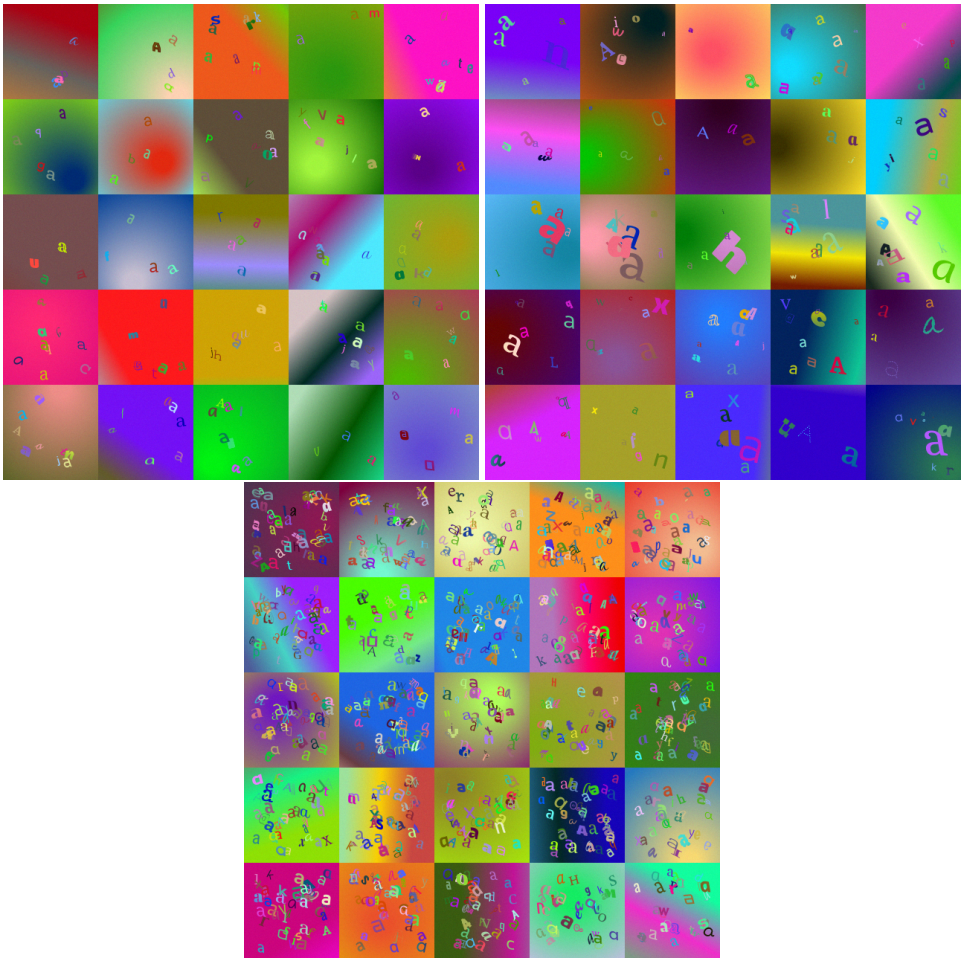

Figure 9: **Object Counting:** Samples from datasets of Section 3.5 **Upper Left:** Fixed Scale dataset. **Upper Right:** Varying Scale dataset. **Lower Center:** Crowded dataset.