# Learned Image Augmentation for Distribution Shift Robustness Using GFlowNets

Pratham Soni
Department of Computer Science
Stanford University
prathams@stanford.edu

Nathan Kim
Department of Computer Science
Stanford University
nathangk@stanford.edu

## Abstract

*Deployed machine learning systems can suffer performance issues when the test-time distribution of samples differs from that seen at train-time [11]. This issue is a critical barrier to the generalization and use of ML models at scale in-the-wild. Traditionally, augmentation methods have been used to help bridge gaps between sampling domains and stem the issue of out of distribution test samples. However, choosing what augmentation policy to use is a large hyperparameter search problem [6]. In this project, we propose GFlowAug, using a GFlowNet as a method of learning distribution robust augmentations for downstream computer vision tasks. We train downstream models using these learned augmentations improving test-time out-of-sample performance.*

## 1. Introduction

Holistically, many machine learning systems can be viewed in a probabilistic sense: learning which output is most probabilistically likely given a particular input. As such, most ML systems are dependent on the distribution of training samples to closely align with those that will be seen during inference. Deployed machine learning systems can suffer performance issues when the test-time distribution of samples differs from that seen at train-time [11]. This issue is a critical barrier to the generalization and use of ML models at scale in-the-wild in many domains from agriculture to healthcare.

For most tasks, this issue of generalizability is combated by seeking out enough training data to generalize model performance. However, in the case where such data is not available, augmentation methods have been used to help bridge gaps between sampling domains and stem the issue of out of distribution test samples by generating additional training samples. It is not clear, however, what the best augmentation policy is for any given task.

Further, attempting to learn what augmentation policy to use is difficult due to a large hyperparameter search problem [6]. In some sense, methods like VAEs and GANs are able to learn a generative process for constructing realistic inputs through the use of self-labels and generator/discriminator networks, respectively. However, they are hindered by brittle hyperparameter choices and computational expensiveness. In this project, we propose the use GFlowNets [4] as a method of learning distribution robust augmentations for downstream computer vision tasks.

As opposed to other learned generative policies (from MCMC to GANs), GFlowNets strive to learn a sampling form that is proportional to a given reward function rather than simply maximizing it. We apply this methodology to images by considering a self-supervised sampling task to generate image samples. We then use these augmented images to train SOTA architectures, aiming to improve performance.[1]

## 2. Literature Review

**GFlowNets** Generative Flow Networks (from here, GFlowNets) are a recent generative network architecture first proposed in [4]. Much like Markov Chain Monte Carlo (MCMC) methods, GFlowNets sample composite objects $s$ from a probability distribution $P_T(s)$ specified over a set or graph whose shape approximates the normalization of a reward/energy function $R(s)$. However, they offer several potential benefits over MCMC methods which motivate a broader study of their applicability as a sampling method: first, while Metropolis-Hastings sampling methods draw each sample from a proposal distribution which can often be slow to converge in practice, GFlowNet samples can be constructed in a single generative pass, eliminating the need for mixing periods. Second, GFlowNets can be trained with labeled pairs $(x, R(x))$ to generalize across multiple distinct regions of the state space, mitigating the mode mixing intractability that is one of the key limitations

[1] Code can be found at https://github.com/ThrunGroup/GFlowAug

of MCMC methods [9]. Finally, GFlowNets exhibit modelling capabilities that are typically considered intractable using MCMC methods, such as the computation of partition functions, marginal probabilities, entropy and mutual information [4].

**Learning Augmentations** Data augmentation can be used to improve domain adaptation and generalization in trained models by increasing the variance of the training data and exposing the model to more regions of the input space [14]. Traditional data augmentation methods, which generally involve generating new examples by applying one or many transformations to existing training examples, have been effectively extended by generative models: Cubuk et al. [6] train an RNN to generate augmentation policies for the transformation of existing data, while Antoniou et al. [1] train GANs to sample new images from a down-projection of an existing image. This particular use of neural generative models is notable because the augmentations can be explicitly optimized to improve the generalization capability of the target model. [6]

# 3. Methods

## 3.1. Image Generation with GFlowNets

In this setting, we use the GFlowNet architecture to procedurally generate image examples as a way to augment training data. For GFloWNets, we wish to have a generative policy $\pi$ to generate some example $x \in \chi$, where $\chi$ represents the distribution we are sampling from with ill-defined or unknown distribution. We wish for $\pi(x) \propto R(x)$, where $R$ represents a reward function. Thus, we sample corresponding to the downstream reward. We can consider this generation process as a list of actions $(a_1 \cdots a_n)$, where $a_i$ is a transition $S \rightarrow S$, for state space $S$. Particularly, $S$ represents partial image constructions, with action $a_i$ representing sampling the next pixel value. All partial images are assigned reward value 0. Thus, $\chi \subset S$ is all the final fully generated images.

Under this framework we introduce a new method to generate high-resolution color images using GFlowNets, based on the Wavelet Flow method proposed in [16]. Our states are still partial images, and our initial state $s_0$ is the uniform zero (black) image. Actions are additions of Haar wavelets.

As with other GFlowNets, the model is trained not with a dataset of observed examples but with a set of trajectories $\tau = (s_0, \ldots, s_n)$ branching from the initial state, generated *stochastically* using a *training policy*; in our initial experiments we use a uniform policy, though the literature on reinforcement learning offers many viable extensions, e.g. policies which encourage exploration of the state space.

Once a trajectory is chosen, the reward $R(s_n) = R(x)$ of the final state is computed and the following loss is computed:

$$\mathcal{L}_\theta(\tau) = \left( \log \frac{Z_\theta \prod_{t=0}^{n-1} P_F(s_{t+1} \mid s_t; \theta)}{R(s_n) \prod_{t=0}^{n-1} P_B(s_t \mid s_{t+1}; \theta)} \right)^2 .$$

This loss is referred to as the *trajectory balance* loss, as it optimises the trajectory balance constraint, an extension of the detailed balance condition for GFN trajectories. A more complete explanation of the trajectory balance loss can be found in the foundations paper [5].

Once we've trained the GFlowNet to produce samples for some subset of the domains of the dataset, we then use the generate samples as train examples for downstream CV models on the WILDS datasets selected [11]. Therefore, we measure the ability of GFlowNets to serve as a robust learned dataset augmentation, testing against distribution shifted data.

We now describe the GFlowAug Pipeline. The full training pipeline is described by Algorithm 1. Sampling images as described in Algorithm 2. During training, we utilize the original image as a ground truth prior. Accordingly, we sample a mask of indices to fill in, with the remaining copied in from the ground truth. In each update step, we pick $dim_{output}$ positions to fill in, derive the sampling statistics using the Feature Extractor, and get the score of the ground truth sample at these positions given the learned statistics. We then fill in the selected positions with the ground truth values and iterate. Thus, in each step, we learn statistics conditional on the state of the filled in image. During inference time, we instead sample from the learned distribution.

**Feature Extractor** We rely on an embedded feature extraction network (a CNN) to learn a mapping between a given partial image and the optimal distribution of the pixels. We then sample from this distribution during inference time. The Feature Extractor takes in an image input of (C+2)xHxW, where C is the number of channels in the input, and is constructed as follows: two successive blocks of 5x5 convolution (32 and 64 channel output, respectively), Leaky ReLU activation, and 2x2 Max Pool followed by a linear layer. From the linear layer, we choose the first $channels_{output} * k * dim_{output}$ features represent $\mu$, the next $channels_{output} * k * dim^2_{output}$ features to represent $\sigma$, and the final $channels_{output} * k$ features to represent $\pi$. Here, $dim_{output}$ is the number of pixels we are sampling, $k$ is the number of Gaussian distributions, and $channels_{output}$ is the number of channels in the image (we fill in all channels at once). $\pi$ represents the score of this sample; $\mu$ is

the mean of the Multivariate Normal Distribution; $\sigma$ is the covariance matrix.

**Algorithm 1** GFlowAug Training

Train Data, Val Data$\leftarrow$ Split Full Dataset
$epoch \leftarrow 0$
$dim_{output}$

**procedure** LOGLIKELIHOOD($x$)
  $x \leftarrow$ downsample($x$) ▷ new shape = (C,H,W)
  $p \leftarrow$ rand$(0.05, 1)$
  $mask_{vis} =$ rand$(0, 1) > p$ ▷ shape = (H, W)
  $\hat{x} \leftarrow x$
  $\hat{x}[!\, mask_{vis}] = -1$
  $fill \leftarrow \sum mask_{vis}$
  $loss \leftarrow 0$
  **while** $fill > 0$ **do**
    $idx \leftarrow$ randSelect$(!\, mask_{vis}, dim_{output})$
    $mask_{take} \leftarrow \mathbf{0}$ ▷ shape = (H, W)
    $mask_{take}[idx] = 1$
    $inp \leftarrow \hat{x} \oplus mask_{vis} \oplus mask_{take}$
    $\pi, \mu, \sigma \leftarrow$ FeatureExtractor$(inp)$
    $dist \leftarrow$ MultivariateNormal$(\mu, \sigma)$
    $x_{gt} \leftarrow x[idx]$
    $prob \leftarrow pi * dist.pdf(x_{gt})$
    $score \leftarrow \log(\sum prob)$ ▷ average across batch
    $loss \leftarrow loss - score$
    $\hat{x}[idx] = x_{gt}$
    $mask_{vis}[idx] = 1$
    $fill \leftarrow fill - dim_{output}$
  **end while**
**end procedure**

**while** $epoch < epochs_{max}$ **do**
  $loss_{train} = 0$
  **for** $x \in$ Train Data **do**
    $loss \leftarrow$ LogLikelihood$(x)$
    $loss_{train} \leftarrow loss_{train} + loss$
    backpropogate $loss$
  **end for**
  $loss_{val} = 0$
  **for** $x \in$ Val Data **do**
    $loss \leftarrow$ LogLikelihood$(x)$
    $loss_{val} \leftarrow loss_{val} + loss$
  **end for**
  $epoch \leftarrow epoch + 1$
**end while**

**Algorithm 2** GFlowAug Sampling

Train Data, Val Data, Test Data$\leftarrow$ Split Full Dataset
$epoch \leftarrow 0$
$dim_{output}$
Model

**procedure** SAMPLE($x$)
  $x \leftarrow$ downsample($x$) ▷ new shape = (C,H,W)
  $p \leftarrow$ rand$(0.05, 1)$
  $mask_{vis} =$ rand$(0, 1) > p$ ▷ shape = (H, W)
  $\hat{x} \leftarrow x$
  $\hat{x}[!\, mask_{vis}] = -1$
  $fill \leftarrow \sum mask_{vis}$
  $loss \leftarrow 0$
  **while** $fill > 0$ **do**
    $idx \leftarrow$ randSelect$(!\, mask_{vis}, dim_{output})$
    $mask_{take} \leftarrow \mathbf{0}$ ▷ shape = (H, W)
    $mask_{take}[idx] = 1$
    $inp \leftarrow \hat{x} \oplus mask_{vis} \oplus mask_{take}$
    $\pi, \mu, \sigma \leftarrow$ FeatureExtractor$(inp)$
    $dist \leftarrow$ MultivariateNormal$(\mu, \sigma)$
    $x_{gt} \leftarrow x[idx]$
    $sample \leftarrow dist.sample(dim_{output})$
    $\hat{x}[idx] = x_{sample}$
    $mask_{vis}[idx] = 1$
    $fill \leftarrow fill - dim_{output}$
  **end while**
**end procedure**

**while** $epoch < epochs_{max}$ **do**
  $loss_{train} = 0$
  **for** $x \in$ Train Data **do**
    $x \leftarrow$ Transform$(x)$
    $x \leftarrow$ Sample$(x)$
    $x \leftarrow$ upsample$(x)$
    $loss \leftarrow$ Model$(x)$
    $loss_{train} \leftarrow loss_{train} + loss$
    backpropogate $loss$
  **end for**
  $loss_{val} = 0$
  **for** $x \in$ Val Data **do**
    $x \leftarrow$ Sample$(x)$
    $loss_{val} \leftarrow loss_{val} + loss$
  **end for**
  $epoch \leftarrow epoch + 1$
**end while**
$loss_{test} = 0$
**for** $x \in$ Test Data **do**
  $x \leftarrow$ Sample$(x)$
  $loss_{test} \leftarrow loss_{test} + loss$
**end for**

### 3.2. Comparison Methods

We turn to the the WILDS leaderboard [11] for downstream algorithms to test the augmentation strategies on. For the iWildCam task, we use the deepCORAL adaptation algorithm [13]. For the other two tasks, we use the ERM adapatation algorithm [10], adding in Random Augmentations for the PovertyMap task. These learning methods are used to train downstream models. For iWildCam, we use the ResNet-50 architecture [2]; for PovertyMap, we use ResNet-18 [2]; for GlobalWheat, we use Faster-RCNN [12]. For all of these models, we leverage pre-trained weights. We turn to the implementation provided by the WILDS team [11] [2] for dataloading and downstream algorithms. We implemented our own GFN-based augmentor and added it to this code-base.

**deepCORAL** [13]: deepCORAL refers to any model architecture that is adaptively trained using CORAL loss. The CORAL loss is posed as the differences between the covariances of the model features output against two differing train distributions. By minimizing the difference in covariances, we aim to learn a model that is robust against the differing training distributions. The loss is incorporated alongside the main objective task (say classification).

**ERM** [10]: Empirical Risk Minimization (ERM) is a adaptation algorithm that aims to use the training losses as a heuristic to measure the risk of a given model output (hypothesis) during testing.

**ResNet** [2]: ResNet, or Residual Net, architecutres refer to a class of CNNs that leverage a distinctive residual block architecture. The residual connection allows for the propogation of identity transforms through the network, allowing for more efficient backpropogation with diminished risk of vanishing gradients.

**FasterRCNN** [12]: FasterRCNN is an architecture for bounding box prediction that leverages a single-pipeline for object detection in image samples. We utilize a CNN backbone to extract a feature map from which we propose Regions of Interest (RoIs). These ROIs are then pooled, and the selected image features are passed to a classifier.

## 4. Data

We take a subset of the Stanford WILDS datasets [11], a collection of in-the-wild datasets that allow for the measurement of data distribution shifts across samples. For sake of scope, we select three of the datasets at differing scales to test robustness of the augmentation learning methodology. All three datasets consist of image or image-like samples. The datasets are as follows:

**iWildCam** [3]: $>$ 203000 images of wildlife across $>$ 300 domains of camera type. We wish to generalize across camera trap types with the task of multi-class species classification across 182 species. This serves as the large dataset. We measure performance using a macro-averaged F1 score across the test domains. Examples are 448x448x3. An example is shown in Figure 1.


Figure 1. iWildCam sample image.

**PovertyMap** [15]: $>$ 19000 multispectral LandSat satellite images across 46 domains consisting of pairs of country and location type (rural versus urban). We wish to generalize the task of asset wealth prediction as measured by a real-valued index computed from Demographic and Health Surveys (DHS) data. This serves as the medium dataset. We measure performance using worst Pearson correlation factor ($r$), comparing between urban and rural domain splits. Examples are 224x224x8. An example is shown in Figure 2.

[2] https://github.com/p-lambda/wilds/tree/main/wilds

Figure 2. PovertyMap sample image (RGB channels only).

**GlobalWheat** [7, 8]: $>$ 6000 images of wildlife across 47 domains consisting of time and place combinations. We wish to generalize the task of predicting weat-head bounding boxes. This serves as the small dataset. We measure performance using the average accuracy across domains. Examples are 1024x1024x3. An example is shown in Figure 3.

Figure 3. GlobalWheat sample image.

# 5. Experimental Setup

## 5.1. GFlowAug Pre-training

While GFlowNets present computational improvements over pure MCMC methods, we are still faced with computational bottlenecks in training the GFlowAug architecture. Accordingly, we downsample all inputs to a resolution of 32x32 prior to input into Feature Extractor. During downstream training we upsample back to the original resolution. While, we were not explicitly ablate against this factor due to computational limitations, we suspect that these downsampling and upsampling steps introduce further robustness due to a data loss effect.

Because of time constraints, we train a unique augmentor for each of the datasets over a subsample of 1000 images from the train set. These represent data fractions as shown in Table 1. We observe improved performance using non-batched stochastic learning, with little improvement in loss after 1 epoch. We utilize PyTorch Lighning's learning rate optimization feature to automatically choose optimal learning rate. Aligning with prior work, we choose to use 8 Gaussian distributions to compose the Multivariate Normal latent space.

| Dataset | Pre-train Data Fraction |
|---|---|
| iWildCam | 0.00679 |
| PovertyMap | 0.0727 |
| GlobalWheat | 0.208 |

Table 1. Data fractions used for pre-training the GFlowAug augmentors.

## 5.2. Downstream Training

We utilize provided hyperparemeters from the WILDS team [11] and simply inject the GFlowAug sampling as part of a transformation pre-processing pipeline. We enumerate the configurations in Table 2.

# 6. Results

## 6.1. Quantitative Results

We present numerical results for the algorithm in Table 3. We compare our results (denoted by the GFlowAug column) against both the published leaderboard values and baselines re-run under the same environment as our results. As can be seen in the table, we improve on the leaderboard score for the iWildCam dataset, while improving against re-run baselines for the other two datasets. Our methodology is least successful for the PovertyMap dataset, and we suspect this is a combination of three factors. First, the images are relatively smaller, diminishing the augmentation effect of the re-sampling. Further, the smaller resolution implies diminished sampling variety during GFlowAug pre-training. Lastly, we posit that the multi-channel (8) input of these images poses a harder learning task for the Feature Extractor.

| Hyperparemeter | iWildCam | PovertyMap | GlobalWheat |
|---|---|---|---|
| Resolution | 448 | 224 | 1024 |
| Channels | 3 | 8 | 3 |
| Downsample Resolution | 32 | 32 | 32 |
| Base Model | ResNet-50 | ResNet-18 | FasterRCNN |
| Algorithm | deepCORAL | ERM | ERM |
| Loss Function | Cross Entropy | MSE | FasterRCNN Criterion |
| Metric | Macro-F1 | Worst-R | Average Accuracy |
| Learning Rate | $3 * 10^{-5}$ | $1 * 10^{-3}$ | $1 * 10^{-5}$ |
| Learning Rate Scheduler | – | Step | – |
| Scheduler Gamma | – | 0.96 | – |
| Weight Decay | 0 | 0 | $1 * 10^{-3}$ |
| Batch Size | 16 | 64 | 4 |
| Epochs | 12 | 200 | 12 |
| Optimizer | ADAM | ADAM | ADAM |
| CORAL penalty | 10 | – | – |

Table 2. Hyperparameters for downstream training.

| Dataset | Leaderboard (std.) | Baseline | GFlowAug |
|---|---|---|---|
| iWildCam | 32.8 (0.1) | 32.79 | **33.27** |
| PovertyMap | 0.49 (0.06) | 0.3561 | 0.3751 |
| GlobalWheat | 51.2 (1.8) | 48.87 | 50.86 |

Table 3. Table of results comparing GFlowAug results against both published leaderboard values and re-run baselines.

### 6.2. Qualitative Results

Here we present some of the sampled images for the iWildCam dataset. As shown in Figure 4, there is high variability in the quality of the filled in images. This can be attributed to a multitude of factors. First, it is apparent that the downsampling leads to strong data loss, this impedes learning strong sampling patterns. Accordingly, when the model is tasked with filling in large amounts of the images, it is easy for it to select into a trajectory far away from the ground trouth. This is because the model outputs are conditioned based on the state of what is already filled in. Finally, we observe that the model is generally good at texture reconstruction, while it obfuscates particular features such as the light in the second row of samples.

## 7. Conclusion

In this paper, we propose a novel method for combatting distribution shift data by building robustness through learned image augmentation. We utilize a GFlowNet based architecture to learn a sampling distribution across an image dataset. We then sample from this learned distribution to construct artificial images representative of the dataset. We then leverage this augmentation procedure against existing SOTA algorithms. With this methodology, we are able to improve model performance by training against more distributionally-robust datasets. However, we are limited by computational bottlenecks and training time. In future work, further improvement in performance is possible with more through training of the augmentors as well as achieving better balance in memory constraints. Qualitative analysis also shows that there is a strong information bottleneck imposed by the downsample/upsample steps. In this study, we also do not perform split/fold replication as per the WILDS guideline, which represents a necessary addition for leaderboard submission. Despite these drawbacks, we posit that GFlowAug presents a promising avenue for probabilistic image augmentation.

## 8. Acknowledgments

We thank Mo Tiwari and the Thrun Lab for seeding our project idea and for the gracious support.

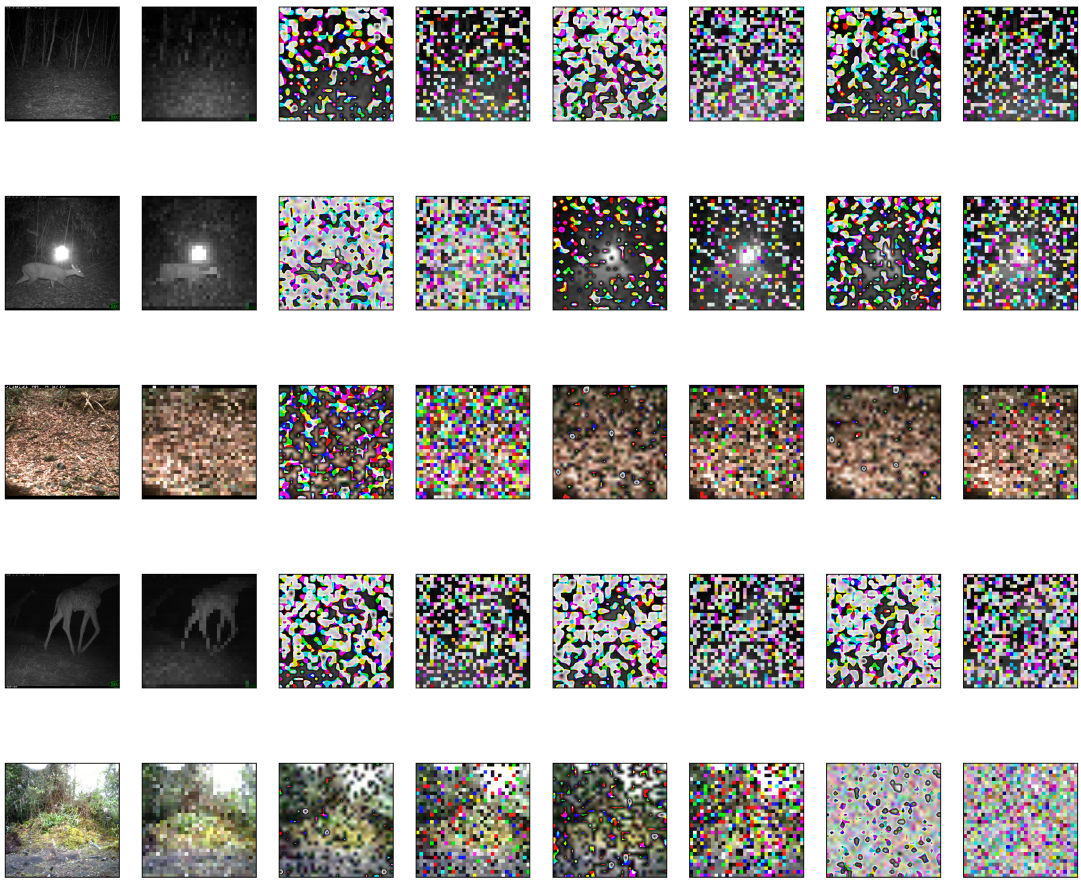

Figure 4. Examples of sampled images from the iWildCam dataset. The left-most column represents the original images, the next column is the downsampled original. We then present three sampled pairs, where the right image is at 32x32 and left image is upsampled back to 1024x1024.

## References


[1] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks, 2017. 2

[2] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks, 2017. 4

[3] Sara Beery, Elijah Cole, and Arvi Gjoka. The iwildcam 2020 competition dataset. *arXiv preprint arXiv:2004.10340*, 2020. 4

[4] Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation, 2021. 1, 2

[5] Yoshua Bengio, Tristan Deleu, Edward J. Hu, Salem Lahlou, Mo Tiwari, and Emmanuel Bengio. Gflownet foundations, 2021. 2

[6] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data, 2018. 1, 2

[7] Etienne David, Simon Madec, Pouria Sadeghi-Tehran, Helge Aasen, Bangyou Zheng, Shouyang Liu, Norbert Kirchgessner, Goro Ishikawa, Koichi Nagasawa, Minhajul A Badhon, Curtis Pozniak, Benoit de Solan, Andreas Hund, Scott C. Chapman, Frederic Baret, Ian Stavness, and Wei Guo. Global wheat head detection (gwhd) dataset: a large and diverse dataset of high-resolution rgb-labelled images to develop and benchmark wheat head detection methods. *Plant Phenomics*, 2020, 2020. 5

[8] Etienne David, Mario Serouart, Daniel Smith, Simon Madec, Kaaviya Velumani, Shouyang Liu, Xu Wang, Francisco Pinto Espinosa, Shahameh Shafiee, Izzat S. A. Tahir, Hisashi Tsujimoto, Shuhei Nasuda, Bangyou Zheng, Norbert Kichgessner, Helge Aasen, Andreas Hund, Pouria Sadhegi-Tehran, Koichi Nagasawa, Goro Ishikawa, Sebastien Dandrifosse, Alexis Carlier, Benoit Mercatoris, Ken Kuroki, Haozhou Wang, Masanori Ishii, Minhajul A. Badhon, Curtis Pozniak, David Shaner LeBauer, Morten Lilimo, Jesse Poland, Scott Chapman, Benoit de Solan, Frederic Baret, Ian Stavness, and Wei Guo. Global wheat head dataset 2021: an update to improve the benchmarking wheat head localization with more diversity, 2021. 5

[9] W.R. Gilks, S. Richardson, and D. Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall/CRC Interdisciplinary Statistics. CRC Press, 1995. 2

[10] Kazuki Irie, Imanol Schlag, Róbert Csordás, and Jürgen Schmidhuber. Improving baselines in the wild, 2021. 4

[11] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts. 1, 2, 4, 5

[12] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. 4

[13] Baochen Sun and Kate Saenko. Deep coral: Correlation alignment for deep domain adaptation, 2016. 4

[14] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John Duchi, Vittorio Murino, and Silvio Savarese. Generalizing to unseen domains via adversarial data augmentation, 2018. 2

[15] Christopher Yeh, Anthony Perez, Anne Driscoll, George Azzari, Zhongyi Tang, David Lobell, Stefano Ermon, and Marshall Burke. Using publicly available satellite imagery and deep learning to understand economic well-being in africa. *Nature Communications*, 2020. 4

[16] Jason J. Yu, Konstantinos G. Derpanis, and Marcus A. Brubaker. Wavelet Flow: Fast training of high resolution normalizing flows. In *NeurIPS*, 2020. 2