InAugment: Improving Classifiers via Internal Augmentation

Moab Arar[1], Ariel Shamir[2], and Amit Bermano[1]

[1]Tel-Aviv University
[2]The Interdisciplinary Center Herzliya

Abstract

Image augmentation techniques apply transformation functions such as rotation, shearing, or color distortion on an input image. These augmentations were proven useful in improving neural networks' generalization ability. In this paper, we present a novel augmentation operation, InAugment, that exploits image internal statistics. The key idea is to copy patches from the image itself, apply augmentation operations on them, and paste them back at random positions on the same image. This method is simple and easy to implement and can be incorporated with existing augmentation techniques. We test InAugment on two popular datasets – CIFAR and ImageNet. We show improvement over state-of-the-art augmentation techniques. Incorporating InAugment with Auto Augment yields a significant improvement over other augmentation techniques (e.g., +1% improvement over multiple architectures trained on the CIFAR dataset). We also demonstrate an increase for ResNet50 and EfficientNet-B3 top-1's accuracy on the ImageNet dataset compared to prior augmentation methods. Finally, our experiments suggest that training convolutional neural network using InAugment not only improves the model's accuracy and confidence but its performance on out-of-distribution images.

1. Introduction

Data augmentation is a popular technique that generates new instances based on some processing of available training data to increase its amount and variance. For image classification tasks, data augmentation is useful in improving the network's generalization, performance, and robustness. This improvement, for example, promotes invariance to fundamental transformations, such as scale, rotation, or color changes.

Some augmentation techniques use handcrafted operations (e.g., CutOut [6]), while others learn the desired operation needed to achieve the most accurate results (e.g., AutoAugment [4])). Still, most augmentation techniques create new images using some global operation (e.g., spatial transformation) or combine images to increase the input space variance [39, 41, 35]. Despite the improvements achieved by these techniques, the final augmented image maintains a single view of the input, limiting each image's information variance. This limitation can hurt the model's generalization ability and robustness, especially for out-of-distribution images.

Our work draws inspiration from the line of works showing that natural images have unique internal statistics: small patches of the image recur abundantly within itself [10, 44]. Exploiting this property, we show that repeating patches at different scales help neural networks model the image's internal distribution, creating an image-specific prior. This property was utilized to solve many ill-posed vision tasks in an unsupervised manner [2, 10, 28]. Nonetheless, to the best of our knowledge, previous data augmentation methods do not exploit this property.

In this paper, we present a novel data augmentation technique we call *InAugment*. In the spirit of the aforementioned works, we seek to increase the information variance within each image rather than the entire dataset. This variance should be increased across different scales, as this is the behavior that natural images present. To adhere to these principles, we introduce a simple to implement two-step augmentation scheme, which can be incorporated in addition to other augmentation methods. In the first stage, we copy random patches from the input image. The patches are then resized, promoting multi-resolution views of the information in the image. In the second stage, the patches are randomly pasted back into the **same image**. This two-stage operation ensures that the image internal-information is repeated at different scales, helping neural networks capture the input image's inner-distribution prior. This patch repetition is beneficial when the patches that contain specific information about the target class (e.g., a dog's head) are pasted on non-salient regions (e.g., the background). In

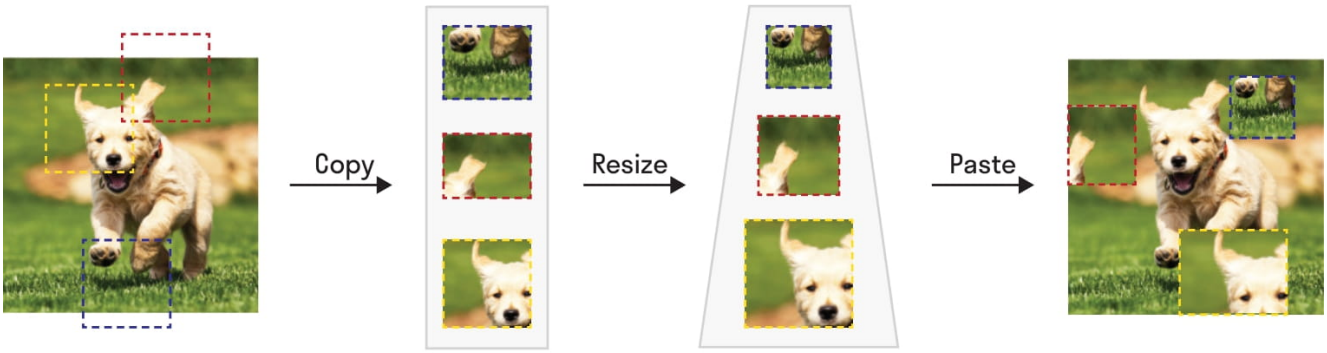


Figure 1: **InAugment overview.** We randomly extract patches from the input image (left). Each patch is then rescaled, and together they form a multi-scale resolution of the same input-image (middle). Finally, the multi-scale patches are pasted on the input image (right) to create the final augmented image. Note that since larger patches are more likely to occlude smaller ones, we sort the paste order according to the patch size.

this case, the target class's main features are repeated in different scales, encouraging the network to detect the input image's essential features.

We validate our approach on the CIFAR [17] and ImageNet [25] datasets through training various standard bench-marking architectures using InAugment and several baselines. Throughout the experiments, we witness consistent improvements in the networks' accuracy, which are even more pronounced when we paste multiple patches at different scales. In particular, we gain between 1% to 1.5% improvement over state-of-the-art augmentation techniques. We further show that models trained with InAugment, are more robust and handle out-of-distribution images really well. Our experiments demonstrate that InAugment should be added as an additional augmentation step when designing new augmentation approaches.

2. Related work

The ability of augmentation methods to improve deep models' generalization led to extensive research in this field. Basic operations such as random horizontal flip and crop became standard in classification tasks [18, 30, 13, 40], while other techniques were proven to be useful for specific datasets. For example, approaches that learn spatial distortion [26, 29, 3, 36], are helpful on the MNIST [19] dataset. For the CIFAR [17] and SVHN [17] datasets, region dropout in the input forces the network to look at other features in the image. Unlike these methods, we show that InAugment achieves improvement on both small and large scale datasets.

Increasing the data can be achieved by mixing input instances. MixUp [41] achieves impressive results via linear interpolation of the two input images while expecting the class labels to follow the same convex combination. Following MixUP, other mixing strategies showed improvements in the robustness [35] and localization [39] of neural networks. These methods operate in batch-resolution, which means data points are combined after the preprocessing stage. Specifically, since we perform InAugment in the preprocessing step, it can be added to existing mixing augmentations [41, 39].

Generative methods can also be used to increase the data size. Lemley et al. [20] train a neural network that incorporates images from the same class to reduce the classification loss. In [34], Tran et al. simultaneously train a classification network and a Bayesian network that generates augmented data, which help improve the classifier performance. Lastly, generative adversarial methods [11] demonstrate excellent ability to generate high-quality images [16], and are leveraged for augmenting data by generating synthetic data [43, 7, 1, 31] or domain-specific image transformations [24]. These methods are usually used in the case where acquiring data is difficult (e.g, medical imaging), or for specific domains (e.g., human faces [16]), but they don't perform well on data with high variance of natural images.

The complexity of designing hand-crafted augmentation methods gave rise to the automated search for augmentation policies. Recently, a novel method named AutoAugment [4] uses reinforcement learning to search for a given dataset's best augmentation policy. The augmentation policy found consists of several sub-policies, and each sub-policy is composed of two basic image transformation operations (e.g., Rotation, Shearing, or Color Jitter). For each image, a sub-policy is chosen uniformly at random and is applied to produce the augmented image. One drawback of AutoAugment is the vast search space, which requires extensive computation power. Several methods, such as Fast-AutoAugment [21], Population-Based-Augmentation [15], RandAugment [5] and Adversarial AutoAugment [42] try to reduce the search complexity while maintaining compet-

itive results. Additionally, in a recent paper [37], Wei et al. argue that automated augmentation policies could lead to over distortion in the data. To overcome this, in addition to ground-truth labels, they use soft-labels provided by a teacher model to compensate for any semantic loss.

In our work, we employ AutoAugment [4] as the baseline augmentation, and show that InAugment boosts its performance; this suggests that policies found by other methods [21, 5, 15] could also benefit from our method.

3. Method

Our method consists of two stages, a *copy-stage*,and a *paste-stage*. In the copy-stage, we copy patches randomly from the underlying image, resize them and apply a base augmentation on the patches. The same base-augmentation is also performed on the input image. Later, during the paste-stage, we paste the augmented patches one after the other, according to their sizes, and produce the final augmented output (see figure 1). In the following subsections, we give an in-depth description about different parts of InAugment. We begin by giving a brief overview of the base-augmentation we use, which is based on AutoAugment [4]. Then we describe in detail the copy-and-paste stages. Throughout this section we let $I \in \mathbb{R}^{H\times W\times 3}$ be an input image of height H and width W.

3.1. Base augmentation

We define an augmentation sub-policy to be an ordered set of transformations $T = \{o_1, o_2, \dots, o_k\}$, such that when T is applied on an image I the resulting image is:

$$T(I) = o_1 \circ o_2 \dots \circ o_k(I) \tag{1}$$

Each operation o_i represents a basic image transformation function (e.g, rotation), and is performed on the image with some predefined probability. The operation $\circ$ denotes the composition operation.

In our implementation, we consider sub-policies used in the AutoAugment [4] method. These sub-policies consists of two transformations (i.e., $k = 2$). The operations used in AutoAugment are: *Shear-x/y, Translate-x/y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, and Sharpness*. Note, AutoAugment also uses CutOut [6], but we omit it from our implementation as our method operates as a region-drop as well (see subsection 4.1). For further information about these sub-policies, please refer to [4].

The final augmentation is determined by a policy, which is a finite set of sub-policies, i.e., $\mathcal{T} = \{T_1, T_2, \dots\}$. To generate the augmented image, we randomly draw a sub-policy $T_i \in \mathcal{T}$, and apply it on I.

3.2. Copy stage

In the copy-stage, we randomly copy n-patches of size $H_p \times W_p$. The patch size can be randomly chosen (for each patch), or it can be set to some fixed value. In our implementation, we choose fixed-size patches for low-resolution images, and random-size patches when the training set has varying image sizes. In the case where the patches deviate from the image boundary, then we trim the patch to be only inside the image (this means that we don't use padding, which is commonly used for random copy). We denote $P = \{P_1, P_2, \dots, P_n\}$ to be the set of copied patches. Once the patches are copied, we perform resizing and augmentation on them. We consider two implementations for the following stage, namely, *resize-first* and *augment-first*.

Resize-first: in the resize first implementation, we first resize the copied patches, and only then apply the base augmentation on them. Formally, given an ordered set of target patch sizes $S_1, S_2, \dots S_n$, we let $\texttt{Resize}\left(\texttt{P}_\texttt{i}, \texttt{S}_\texttt{i}\right)$ be the operation that resizes the patch P_i to the target size S_i. After we resize the patches, the augmentation $\mathcal{T}$ is applied, yielding the final copied patch for this implementation to be:

$$P_{copied} = \left\{\mathcal{T}\left(\texttt{Resize}\left(\texttt{P}_\texttt{i}, \texttt{S}_\texttt{i}\right)\right)\right\}_{i=1}^{n} \tag{2}$$

Augment-first: in this implementation, as the name suggests, we first apply the base augmentation and then resize the augmented patch. Therefore, the final copied patches in this implementation are:

$$P_{copied} = \left\{\texttt{Resize}\left(\mathcal{T}\left(\texttt{P}_\texttt{i}\right), \texttt{S}_\texttt{i}\right)\right\}_{\texttt{i=1}}^{\texttt{n}} \tag{3}$$

In our implementation, we found that for small images (e.g., images for the CIFAR [17] dataset), it is best to use **Augment-first** implementation. For high-resolution images (e.g., images from the ImageNet [25] dataset), we use the Resize-first implementation since it is more efficient to perform the base augmentation on smaller patches (after the resize). Furthermore, we found that it is better to apply the same transformations on the input image and the copied patches. We believe that sampling different augmentations for different patches will most likely yield a patch augmented with an easier transformation than the other patches. This, in turn, will bias the network to concentrate on easier patches. Note that we omit to transform the base-image first for efficiency reasons. It is better to copy smaller patches and perform the augmentation on smaller images than having to perform the augmentation on the entire image.

3.3. Paste stage

Let P_{copied} be the copied patches after resize and augmentation from the previous stage. Then, we paste the patches at random locations on top of the image $\mathcal{T}(I)$ (i.e, the image after the augmentation). We also drop patches with probability p_i, meaning, each patch in P_{copied} is pasted

onto the image with probability $1 - p_i$. We found this helpful when training large networks on larger datasets. Note, the paste order follows the patches' sizes in P_{copied}. This means that we first paste the largest patch, and finally, we paste the smallest patch. This ensures that larger patches will not occlude smaller ones. Furthermore, to make the implementation as simple as possible, in the case where the pasted patch deviates from the image boundary, we clip it to fit the final augmented image.

4. Experiments

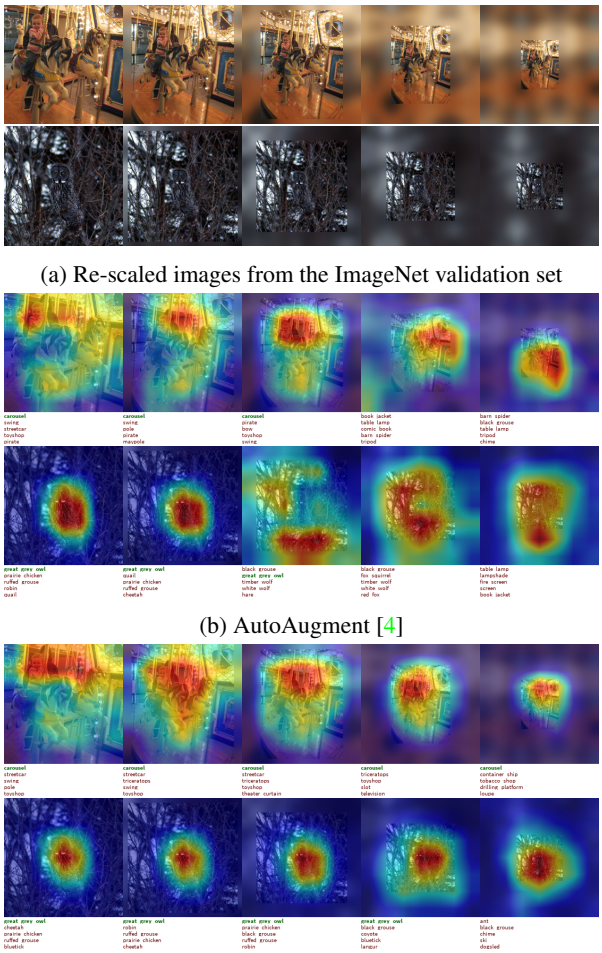
In this section, we conduct several experiments to support our claim. First, we show a direct comparison between InAugment and Cutout [6] and argue that InAugment achieves better results even for larger patch sizes. Second, we show that resizing the patches is beneficial and using multiple patches can boost the network's accuracy. We also show that networks trained with InAugment are robust to scale, especially for out-of-distribution object sizes. Finally, we compare InAugment with previous state-of-the-art augmentation techniques and evaluate the test-accuracy on two popular datasets, ImageNet ILSVRC12 [25] and the CIFAR [17] dataset. Alternative approaches to InAugment are reported in the supplemental materials.

4.1. Ablation experiments

In this section, all networks are trained on the CIFAR-100 dataset. Unless otherwise stated, then the training setup follows the one in Section 4.2.

InAugment vs CutOut [6]: CutOut is an augmentation method in which random regions of the input image are removed. The idea in CutOut, is that eliminating certain parts of the image could encourage the network to look at other features and essentially serve as a regularization technique. This technique is very effective on the CIFAR dataset, and it is usually adopted when training neural networks on this dataset.

In our method, random patches are copied and pasted on the image, which could lead to the case where non-important regions (e.g., background patches) are pasted on top of essential areas. Therefore, InAugment may be performing CutOut, where important parts are removed continuously by pasting non-important regions on top of them. To see that this is not the case, we trained our network in three different settings. In one experiment, we apply CutOut augmentation with varying cut sizes. For InAugment, we only copy one random patch and randomly paste it back on the input image after augmentation (we do not perform any resize). We conduct two experiments for InAugment, one where the patch is entirely within the image boundary. In the second experiment, we allow patches to deviate from the image boundary. In this case, the patch is trimmed to only copy pixels from the image itself (we do not perform any padding). Finally, we also trained the network using the standard augmentation (i.e., random crop and random horizontal flip) and reported baseline accuracy.



(a) Re-scaled images from the ImageNet validation set

(b) AutoAugment [4]

(c) AutoAugment [4] + InAug (ours)

Figure 2: Grad-CAM [27] visualization for ResNet50 (**best viewed when zoomed**). The input images appear in figure 2a, where the leftmost column contains images that underwent the standard validation pre-processing. In figure 2b we show the Grad-CAM visualization for a network trained using AutoAugment. We also print the top-5 predictions of the network below each image. Labels are ordered by the network's confidence, and correct labels are highlighted in green. Similarly, we report the same visualization for ResNet50 trained with AutoAugment + InAugmet (ours) in figure 2c.

In Figure 3, we report the average top-1 accuracy as observed over five different runs. As can be seen, both CutOut and InAugment improve over the baseline training. One of the main issues with the Cutout is that removing larger ar-

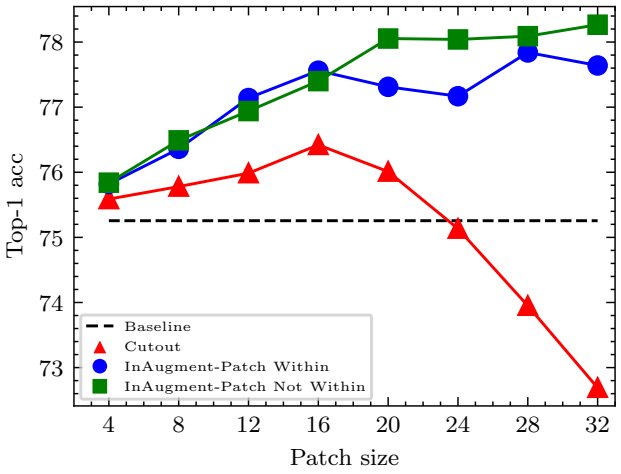


Figure 3: Comparison between CutOut [6] and InAugment (ours). We plot PreAct-ResNet18 [14] top-one accuracy as a function of the patch size used for CutOut and InAugment. In all experiments, only the standard augmentations were applied on the base image (i.e., random crop and random horizontal flip). As can be seen from the figure, CutOut has a certain patch-size threshold, after which applying CutOut result in information loss. On the other hand, InAugment benefits from larger patches. For InAugment, we considered two setups. In the first setup, patches are restricted to be within the image boundary (blue), while in the other setup, patches can deviate from the image boundary (green). Note that the second setup yields better results, which indicates that varying patch-sizes could be beneficial.

eas usually leads to loss of information, and the effective cut size for CIFAR100 is 16. Interestingly, with InAugment, larger patch sizes only improve the network's performance and lead to a 2% performance boost in network accuracy. One reason for this phenomenon is that it is less likely that these large patches will contain only useless information when copied from the image. Hence, the pasted patch will repeat essential features in the input images while still occluding other parts. Lastly, note that when we allow patch sizes to vary, we achieve a 0.4% performance gain relative to constant size patches. Therefore, applying varying patch sizes could lead to better results. It is worth mentioning that this was strongly observed in the ImageNet [25] experiments, where the input images have varying patch sizes. In our experiments, we found that copying patches with random-sizes are essential in the ImageNet experiment.

Effect of resizing patches: the repetition of patches in varying scales is an integral part of InAugment. To see the effectiveness of resizing the patches, we experiment with copying and pasting a single patch. In particular, we copy a random patch of size $s_p \times s_p$ and resize it to size $s_r \times s_r$, where $s_p, s_r \in \{12, 16, 20, 24\}$ (note, in the case where $s_p = s_r$, then we don't perform any resizing). In this experiment, both the input image and the patch undergo the standard transformation (e.g. random horizontal flip).

Patch size (s_p) \ Patch resize (s_r)	12	16	20	24
12	76.78	77.45	77.63	**77.73**
16	77.65	77.45	**78.01**	77.66
20	77.33	77.87	77.67	**78.02**
24	**77.93**	77.78	77.58	**77.9**

Table 1: Resize affect on top-1 accuracy. Rows represent the size of the copied patch. Columns represent the new size of the patch before we paste it (after resizing). We also report the accuracy in the case we don't perform any resize (diagonal).

As can be seen from Table 1, the best accuracy is obtained when we resize the patches before pasting them back into the image. In particular, the smaller the patch, the more crucial resizing becomes. For example, the best accuracy reported for patches of size (12×12), is when we resize the patch to (24×24), which is 1% improvement over not performing any resize. For larger patches, we see that the resizing helps for most s_r. We believe this is because in our implementation, if the random patch we try to copy deviates from the image boundary, then it is trimmed to fit the image boundary. Therefore, larger patches are frequently clipped, which means that their effective size is small on average. Therefore, we choose to copy multiple patches in our final implementation and resize them to form exponentially increasing sizes.

Effect of multiple patches: To see the effect of pasting multiple patches onto the image, we trained three models, each with three different settings, added on top of the AutoAugment baseline. For settings-k, we copy k random patches of size (32×32). The patches are then resized such that the i-th patch is re-scaled by factor $\sigma_i = (0.5)^{i-1}$.

The average test accuracy and the average test loss are reported in Table 2 for all three settings. Also, we report the results obtained by training the models using AutoAugment as a baseline augmentation technique. As can be seen from the Table, incorporating InAugment not only improves the overall accuracy but significantly improves the Cross-Entropy (CE) loss on the test-set. Interestingly, training Pre-ActResNet18 [14] with two patches improves the average CE loss by 0.03, even though the average accuracy is

	AA	InAug x1	InAug x2	InAug x3
PreActResNet	79.11 / 0.95	**80.27** / 0.82	80.19 / **0.79**	79.70 / 0.80
WideResNet-28-10	83.84 / 0.59	**84.80** / **0.54**	84.46 / **0.54**	84.27 / 0.55
ShakeShake-96	85.90 / 0.55	86.65 / 0.52	**86.89** / **0.50**	86.65 / **0.50**

Table 2: Number of patches effect. We report the average top-1 accuracy and the test Cross Entropy Loss as observed over 5 different runs. In this experiment we copy and paste up-to three patches. We let InAugment xn represent the experiment in which n-patches are used (where $n = 1, 2, 3$). We also report the results obtained for Auto Augment using our implementation.

somewhat negatively affected. This suggests that the network's confidence improves when increasing the number of patches. We believe the network accuracy has not improved because the network has a small capacity, and using multiple patches makes the optimization process harder. On the other hand, training ShakeShake26-96 with InAugmentx2 is the best setting for that network. We conclude that using multiple-patches becomes more significant when training larger models for a longer period of time.

Alternative approaches: we copy patches from the original input image in our method and perform the same augmentation on the copied patches. We also considered applying different AutoAugment policies on the copied patches, where we hoped to create a more significant visual variability in the augmented image. However, we found this not to work well, especially for large-capacity networks. We hypothesize that this happens because drawing multiple augmentations will more likely apply easier augmentations on some patches, leading networks to ignore patches that underwent harder augmentations. For further details, please refer to the supplemental material.

4.2. CIFAR Experiments

We compare InAugment with previous augmentation methods: CutOut [6], AutoAugment (AA) [4], Population Based Augmentation (PBA) [15], FastAutoAugment (FAA) [21], and RandAugment (RA) [5]. In our experiment, we train three different networks, PreAct-ResNet18 [14, 13], WideResNet-28-10 [40], and Shake-Shake26-2x96 [8] on the CIFAR10 and CIFAR100 datasets [17].

Experiment details: for all experiments we use the Stochastic Gradient Descent (SGD) optimizer, with momentum 0.9 and weight decay of $1e-3$, $1e-4$, and $5e-4$ for PreAct-ResNet18, WideResNet28-10 and ShakeShake26-2x96, respectively. A nesterov momentum we used in the experiments of WideResNet28-10 and Shake-Shake26-2x96. The training settings for PreAct-ResNet18 matches those in [41], and for WideResNet and Shake-Shake26-2x96, we follow the training settings used in [4, 5]. In particular, we train PreAct-ResNet18 and WideResNet-28-10 for 200 epochs with the initial learning rate set to 0.1. For Shake-Shake26-2x96, we use a learning rate 0.01 and train the network for 1800 epochs. The learning rate schedule used for PreAct-ResNet18 is a multi-step scheduler, in which the learning rate is scaled by 0.1 on epochs 100 and 150. A cosine-learning [23] rate was used for both WideResNet-28-10 and Shake-Shake26-2x96. For the WideResNet28-10 and PreAct-ResNet18 models, we copy one patch of size (32×32) and don't perform any resize on the patch. For Shake-Shake26-2x96 model, we copy two patches of size 32, and rescale the second patch by 0.5. Also, we always chose to paste the copied pasted, i.e., our drop rate is $p_i = 0$ (see section 3.3).

Results: the results are shown in Table 3. Specifically, incorporating InAugment with AutoAugment yielded the highest accuracy for all networks used. We obtain about 1.0% improvement for all networks trained on the CIFAR100 dataset compared to the best-reported result for all the other methods. For CIFAR10, we also achieve the highest accuracy compared to all the networks and augmentation methods on which we experimented, although by a smaller margin. We postulate the reduced improvement is due to the high initial accuracy, and the heavily tuned regularization schemes used by some of the methods. We note that our experiments were impelemented using the Pytorch framework, while the AA baseline was originally implemented using the Tensorflow one. This induces minor changes in the resulting accuracies (e.g., in our implementation, the WideResNet28-10 model trained with AA achieves only 97.15% as opposed to the 97.4% reported in the original paper). In any case, we report in Table 3 the accuracy which is better between our implementation and that of the original paper [4].

4.3. ImageNet Experiments

We also show that InAugment is effective on larger datasets with high-resolution images. In particular, we train ResNet-50 [13] and EfficientNet-B3 [17] from scratch on the ImageNet-ILSVRC2012 [25] dataset.

Experiment details: the training setup follows the default implementation in the official TensorFlow-1.15.4 code. In particular, the ResNet50 models were trained for 180 epochs and the EfficientNet-B3 models were trained for 350 epochs. We use the same training hyper-parameters used in [5, 4]. The training images follow the standard augmentation where they are randomly-cropped and resized to the size (224×224) for the ResNet50 model, and to size (300×300) for the EfficientNet-B3 model (a random horizontal flip is applied). During test-time, images are center cropped and resized to the same resolution as in the train-

	Baseline	CutOut	AA	PBA	FAA	RA	AA+InAug
CIFAR10							
PreAct-ResNet-18	94.32 ± .18	95.67 ± .15	96.0 ± .05	-	-	-	**96.35 ± .08**
WideResNet-28-10	96.1	96.9	97.4	97.4	97.3	97.3	**97.45 ± .04**
ShakeShake26 2x96	97.1	97.4	98.0	98.0	98.0	98.0	**98.30 ± .05**
CIFAR100							
PreAct-ResNet-18	75.32 ± 0.11	76.48 ± 28	79.11 ± .11	-	-	-	**80.27 ± .31**
WideResNet-28-10	81.2	81.6	83.80 ± .17	83.3	82.7	83.3	**84.80 ± .20**
ShakeShake26 2x96	82.9	84.0	85.90 ± .11	84.7	85.4	-	**86.89 ± .20**

Table 3: Top-1 accuracy report on CIFAR. The reported results with both mean and std are based on our implementation. Results for AutoAugment (AA), Population-Based Method (PBA) Fast AutoAugment (FAA) and RandAumgnet (RA) are reported in the original papers of these methods [5, 4, 15, 21]. In particular, for AA we report the better result between our implementation and those reported in the original paper [4].

ing. The ResNet50 network was trained with the SGD optimizer, with momentum 0.9 and weight decay $1e-4$. We trained the network on a single v3-8 TPU, with a global batch size of 1024. The learning rate was set to 0.1 and is linearly scaled by the batch-size divided by 256, following [12]. To train EfficientNet-B3, we use a single v2-28 TPU pod with global batch size of 4096. The network was trained with RMSProp with learning rate of 0.016, momentum 0.9, $\epsilon = 0.001$ and decay of 0.9. The learning rate is also linearly scaled as in the case of the ResNet50 model.

The base augmentation we use is the policy that AutoAugment found for Efficientnet [33] (this policy is available in the official TensorFlow code). Furthermore, we adopt the resize-first implementation (see copy-stage in subsection 3.2) since it achieves the right balance between efficiency and accuracy. Specifically, the pre-processing time is lower when applied to a low-resolution image; therefore, augmenting the resized patches will take less time (which is crucial for large-scale datasets). For ResNet50, we copy three patches of random sizes and resize them to (134×134), (80×80), and (48×48), the patches are then pasted according to the order of their size. For Efficientnet-B3 [33], we copy two-patches and randomly resize them to size $(s_1 \times s_1)$ and $(s_2 \times s_2)$, where s_1 and s_2 are uniformly sampled from $[300, 150]$ and $[150, 8]$, respectively. In both experiments, patches are dropped and not pasted onto the images with a probability 0.5.

Results: the results are reported in Table 4. As shown from the table, we improve both the top-1 and top-5 accuracy of ResNet-50 compared with previous augmentation methods. Specifically, we achieve 78.2-% top-1 accuracy, which is about 0.6% improvement over previous augmentation methods [4, 21, 5]. We also exhibit an approximately 0.2% improvement in the top-5 accuracy. Similarly, we also witnessed an improvement in the Efficientnet-B3 experiments, where a top-1 accuracy of 81.8% was achieved,

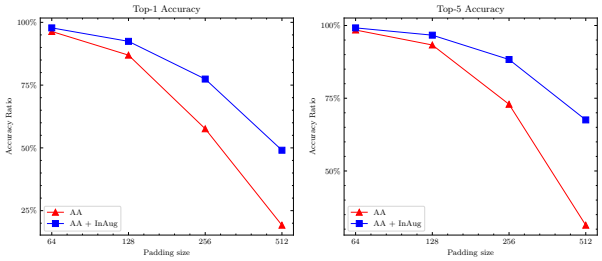


Figure 4: ResNet50 [13] robustness to scale. We plot the ratio between the network's accuracy after scaling the validation set and its accuracy on the original validation set. The ratio of both the top-1 (left) and the top-5 accuracy (right) are shown. As can be seen, training the network using AutoAugment (AA) + InAugment (InAug) achieves significantly better results than training it solely with AA augmentation.

which is 0.2% higher than using AutoAugment as the only augmentation.

4.4. Out-of-distribution samples

The standard ImageNet [25] training consists of randomly cropping images and resizing them to a fixed size (e.g., 224×224 for ResNet50 [13]). Therefore, the network will most likely observe the same object at different sizes during training. However, we show that this pre-processing is not enough to handle out-of-distribution scales, and using InAugment in training achieves a robust model that is more scale-invariant.

To test our hypothesis, we want to measure the network's ability to classify objects at different scales. To scale the entire ImageNet's validation set, we pad each image with fixed padding D, and the rest of the pre-processing remains the same. Furthermore, to avoid introducing artifacts, we use symmetric padding and strongly blur the padded pix-

	Baseline	Fast AA	RA	AA	AA + InAug (Ours)
ResNet-50	76.3 / 93.1	77.6 / 93.7	77.6 / 93.8	77.6 / 93.8	**78.2** / **94.0**
EfficientNet-B3	81.1 / -	-	-	81.6 / -	**81.8** / 95.6

Table 4: Top-1 and Top-5 accuracy results on ImageNet. The results for AutoAugment on ResNet50 are replicated in our experiments. The results for other methods and models are from their original papers [5, 21, 4]

els. Note that we cannot simply resize the images since ResNet50 was designed to process images of size at least 224×224. We create four different validation sets, using different pad sizes, namely $D \in \{64, 128, 256, 512\}$ (see figure 2a).

In figure 4 we plot the ratio of the network's accuracy on the padded validation sets and the original set. As can be seen, incorporating InAugment in pre-processing the training data yields models with better robustness to scale. In particular, you can see that for large padding (i.e., smaller scales), the network's performance trained solely using Auto Augment witnesses a catastrophic performance drop. Note that performance drop on both settings is inevitable since some images already appear on a small scale (e.g., in the owl image in figure 2a, you can see that it is difficult to distinguish the owl in the small scale example). We also visualize the regions that the network bases its decision on using GradCAM [27]. In figure 2b, it is evident that the baseline network no-longer looks at important regions when it makes its decision on small-scale images. On the other hand, as seen in figure 2c, models trained with our methods exhibit localized decisions for smaller-scale instances. Note in the owl example, even though the network trained with our method did not correctly classify the image with padding $D = 512$, it still based its decision on the owl location. However, due to the extreme scale, the owl features are no longer distinguishable.

Finally, we also considered two alternative approaches to scale the validation set. In one setup, we used zero-padding, and in the other, we tiled the image multiple times. In both cases, training with our-method improves upon solely training with Auto Augment. Especially in the zero-padding case, where the network trained with Auto Augment only, confuses the padded-images to be 'Television' (see supplemental material for more examples).

5. Discussion

In this paper, we introduced a novel image augmentation method. We showed that copying-and-pasting random patches expose the network to higher quality features, especially when the patches are resized. In particular, InAugment improves the robustness of the network to scale and improves its confidence and accuracy. As mentioned throughout the text, we compared InAugment with previous augmentation methods, including AutoAugment [4], RanAugment [5], Fast AutoAugment [21], and CutOut [6], showing consistent improvement. We believe that incorporating InAugment with existing methods could further boost classifiers' performance. For example, InAugment can be considered an image transformation function, which means it can also be added to automated augmentations' search space (e.g., AutoAugment). Furthermore, we believe that performing different augmentations on the copied patches could increase the image's information variance, boosting the network's performance even further. Beyond improving validation accuracy on the test set, some augmentation techniques corrupt [22, 38, 32, 9] the input data and lead to robust models. The same distortions could also be applied to the copied patches, which could increase the distortion variance. Therefore, we believe InAugment can also be adapted for improving the model robustness for out-of-distribution images. Finally, since patches are copied randomly in InAugment, it might be fruitful to investigate attention-based InAugment, where meaningful patches are copied instead of random ones and are pasted onto non-important regions.

Acknowledgment

Research supported with Cloud TPUs from Google's TensorFlow Research Cloud (TFRC).

References

[1] Antreas Antoniou, Amos J. Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *CoRR*, abs/1711.04340, 2017. 2

[2] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA*, pages 60–65. IEEE Computer Society, 2005. 1

[3] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *2012 IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, USA, June 16-21, 2012*, pages 3642–3649. IEEE Computer Society, 2012. 2

[4] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer*

Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pages 113–123. Computer Vision Foundation / IEEE, 2019. 1, 2, 3, 4, 6, 7, 8

[5] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14-19, 2020*, pages 3008–3017. IEEE, 2020. 2, 3, 6, 7, 8

[6] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *CoRR*, abs/1708.04552, 2017. 1, 3, 4, 5, 6, 8

[7] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using GAN for improved liver lesion classification. *CoRR*, abs/1801.02385, 2018. 2

[8] Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017. 6

[9] Justin Gilmer, Nicolas Ford, Nicholas Carlini, and Ekin D. Cubuk. Adversarial examples are a natural consequence of test error in noise. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2280–2289. PMLR, 2019. 8

[10] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*, pages 349–356. IEEE Computer Society, 2009. 1

[11] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014. 2

[12] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 7

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2, 6, 7

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pages 630–645. Springer, 2016. 5, 6

[15] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2731–2741. PMLR, 2019. 2, 3, 6, 7

[16] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 4401–4410. Computer Vision Foundation / IEEE, 2019. 2

[17] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 2, 3, 4, 6

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. 2

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

[20] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017. 2

[21] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 6662–6672, 2019. 2, 3, 6, 7, 8

[22] Raphael Gontijo Lopes, Dong Yin, Ben Poole, Justin Gilmer, and Ekin D. Cubuk. Improving robustness without sacrificing accuracy with patch gaussian augmentation. *CoRR*, abs/1906.02611, 2019. 8

[23] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 6

[24] Alexander J. Ratner, Henry R. Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. Learning to compose domain-specific transformations for data augmentation. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 3236–3246, 2017. 2

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2, 3, 4, 5, 6, 7

[26] Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. APAC: augmented pattern classification with neural networks. *CoRR*, abs/1505.03229, 2015. 2

[27] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 618–626. IEEE Computer Society, 2017. 4, 8

[28] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), 18-23 June 2007, Minneapolis, Minnesota, USA*. IEEE Computer Society, 2007. 1

[29] Patrice Y. Simard, David Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pages 958–962. IEEE Computer Society, 2003. 2

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. 2

[31] Leon Sixt, Benjamin Wild, and Tim Landgraf. Rendergan: Generating realistic labeled data. *Frontiers Robotics AI*, 5:66, 2018. 2

[32] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014. 8

[33] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. 7

[34] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle J. Palmer, and Ian D. Reid. A bayesian data augmentation approach for learning deep models. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 2797–2806, 2017. 2

[35] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6438–6447. PMLR, 2019. 1, 2

[36] Li Wan, Matthew D. Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1058–1066. JMLR.org, 2013. 2

[37] Longhui Wei, An Xiao, Lingxi Xie, Xin Chen, Xiaopeng Zhang, and Qi Tian. Circumventing outliers of autoaugment with knowledge distillation. *CoRR*, abs/2003.11342, 2020. 3

[38] Dong Yin, Raphael Gontijo Lopes, Jon Shlens, Ekin Dogus Cubuk, and Justin Gilmer. A fourier perspective on model robustness in computer vision. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 13255–13265, 2019. 8

[39] Sangdoo Yun, Dongyoon Han, Sanghyuk Chun, Seong Joon Oh, Youngjoon Yoo, and Junsuk Choe. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pages 6022–6031. IEEE, 2019. 1, 2

[40] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press, 2016. 2, 6

[41] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. 1, 2, 6

[42] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. 2

[43] Xinyue Zhu, Yifan Liu, Zengchang Qin, and Jiahong Li. Data augmentation in emotion classification using generative adversarial networks. *CoRR*, abs/1711.00648, 2017. 2

[44] Maria Zontak and Michal Irani. Internal statistics of a single natural image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 977–984, 2011. 1