Learning Data Augmentation Strategies for Object Detection

Barret Zoph(✉), Ekin D. Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V. Le

Brain Team, Google Research, Mountain View, USA
barretzoph@google.com

Abstract. Much research on object detection focuses on building better model architectures and detection algorithms. Changing the model architecture, however, comes at the cost of adding more complexity to inference, making models slower. Data augmentation, on the other hand, doesn't add any inference complexity, but is insufficiently studied in object detection for two reasons. First it is more difficult to design plausible augmentation strategies for object detection than for classification, because one must handle the complexity of bounding boxes if geometric transformations are applied. Secondly, data augmentation attracts less research attention perhaps because it is believed to add less value and to transfer poorly compared to advances in network architectures.

This paper serves two main purposes. First, we propose to use AutoAugment [3] to design better data augmentation strategies for object detection because it can address the difficulty of designing them. Second, we use the method to assess the value of data augmentation in object detection and compare it against the value of architectures. Our investigation into data augmentation for object detection identifies two surprising results. First, by changing the data augmentation strategy to our method, AutoAugment for detection, we can improve RetinaNet with a ResNet-50 backbone from 36.7 to 39.0 mAP on COCO, a difference of +2.3 mAP. This gain exceeds the gain achieved by switching the backbone from ResNet-50 to ResNet-101 (+2.1 mAP), which incurs additional training and inference costs. The second surprising finding is that our strategies found on the COCO dataset transfer well to the PASCAL dataset to improve accuracy by +2.7 mAP. These results together with our systematic studies of data augmentation call into question previous assumptions about the role and transferability of architectures versus data augmentation. In particular, changing the augmentation may

B. Zoph and E. D. Cubuk—Equal contribution.
Code and models are available at https://github.com/tensorflow/tpu/tree/master/models/official/detection.

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-58583-9_34) contains supplementary material, which is available to authorized users.

A. Vedaldi et al. (Eds.): ECCV 2020, LNCS 12372, pp. 566–583, 2020.
https://doi.org/10.1007/978-3-030-58583-9_34

lead to performance gains that are equally transferable as changing the underlying architecture.

1 Introduction

Much work in object detection was devoted to building better model architectures or detection algorithms [10,11,13,22,32–35,39]. Although these changes often lead to more accurate models, they also add complexity that potentially slows down the detection system at both training and inference.

Data augmentation, on the other hand, is relatively understudied in object detection. So far the most common distortions are horizontal flips and scale jittering [23]. We suspect that the lack of research in this area is due to two reasons. First, object detection often comes with bounding boxes, so transferring strategies from image classification to object detection will be sub-optimal. These added degrees of freedom from the bounding boxes suggest some automation is needed to find a very good augmentation policy. Secondly, it is also a common belief that custom data augmentation adds smaller performance improvements than architectures and transfers poorly from one detection dataset to another.

Here we investigate the value of data augmentation in object detection. As aforementioned, since it can be difficult to design good data augmentation strategies for object detection, and inspired by the recent success of AutoAugment for classification [3], we use AutoAugment [3] to find good combinations of transformations for detection. To tailor AutoAugment for detection, we add novel operations that handle bounding boxes differently from the image which improves results.

Our experiments with AutoAugment for detection identify two key surprising findings. First, data augmentation is more valuable than commonly believed. In particular, by changing the data augmentation, we can improve RetinaNet with a ResNet-50 backbone from 36.7 to 39.0 mAP on COCO, a difference of 2.3 mAP. This gain is even slightly better than changing the backbone from ResNet-50 to ResNet-101 which only gives +2.1 mAP improvement but with a higher cost of training and inference. Secondly, it is also common wisdom that intricate data augmentation can "overfit" to the dataset of interest and does not transfer well to other datasets. Our experiments show that data augmentation is transferable between detection datasets just like architectures. For example, the best data augmentation found on COCO transfers well to PASCAL to improve the accuracy by +2.7 mAP. This means that the augmentation strategies we found on COCO can be used directly on future object detection datasets without changing any parameters.

In summary, our main contributions are as follows:

- We propose a novel set of data augmentation operations that uniquely act upon the content of bounding boxes and even sometimes change their location.
- We implement a search method based on AutoAugment [3] to combine and optimize data augmentation policies for object detection problems by utilizing novel operations specific to bounding box annotations.

– We show surprising results that improving data augmentation can be as effective as improving architectures while adding no cost to the inference and minimal cost to training.
– We show surprising results that data augmentation strategies are transferable across different detection datasets, architectures and algorithms.

2 AutoAugment for Object Detection

2.1 Search Space Definition

As mentioned above, commonly used data augmentation methods for object detection are quite simple: horizontal flipping and scale jittering. Here we would like to add more augmentation operations to object detection to further boost the value of data augmentation. Hence we turn our attention to a search method to compose basic image transformations into sophisticated distortions to improve generalization performance. The search method expands on our previous work, AutoAugment [3], where a reinforcement learning method is used to learn to compose image processing operations mainly from the Python Image Library (PIL).

Here we adapt the method to perform well on object detection. Our key observation is that object detection introduces many additional complications such as maintaining consistency between a bounding box location and a distorted image. To handle this complication we explore how to change the bounding box locations when geometric transformations are applied to the image. To further adapt our method to object detection, we notice that bounding box annotations open up the possibility of introducing augmentation operations that uniquely act upon the contents within each bounding box. Using this observation, we add many new augmentation operations that work on each bounding box independently in addition to existing image augmentation operations.

For our data augmentation policy[1] we use the following parameterization. We define an augmentation policy as a unordered set of K sub-policies. During training one of the K sub-policies will be selected at random and then applied to the current image. Each sub-policy has N image transformations which are applied sequentially. We turn this problem of searching for a data augmentation policy into a discrete optimization problem by creating a search space. This space gives us the flexibility to have a diversity of operations in a single policy, while having a constraint on how large the space can be. We also want our augmentation policy to benefit from augmentation diversity, which has been found to be useful in the classification domain [3].

In our implementation, our search space consists $K = 5$ sub-policies with each sub-policy consisting of $N = 2$ operations applied in sequence to a single image. Additionally, each operation is also associated with two hyperparameters specifying the probability of applying the operation, and the magnitude of the operation. Figure 1 (bottom text) demonstrates 5 of the learned sub-policies. The

[1] In this paper, we use "policy" and "strategy" interchangeably.

Sub-policy 1. `(Color, 0.2, 8), (Rotate, 0.8, 10)`
Sub-policy 2. `(BBox_Only_ShearY, 0.8, 5)`
Sub-policy 3. `(SolarizeAdd, 0.6, 8), (Brightness, 0.8, 10)`
Sub-policy 4. `(ShearY, 0.6, 10), (BBox_Only_Equalize,0.6, 8)`
Sub-policy 5. `(Equalize, 0.6, 10), (TranslateX, 0.2, 2)`

Fig. 1. Examples of data augmentation sub-policies. 5 examples of learned sub-policies applied to one example image. Each column corresponds to a different random sample of the corresponding sub-policy. Each step of an augmentation sub-policy consists of a triplet corresponding to the operation, the probability of application and a magnitude measure. The bounding box is adjusted to maintain consistency with the applied augmentation. Note the probability and magnitude are discretized values (see text for details).

probability parameter introduces a notion of stochasticity into the augmentation policy where the selected augmentation operation is applied to the image with the specified probability.

Our goal was to include as many augmentation transformations as possible to get the best understanding of what operations would be useful for object detection. To limit the complexity of including every operation, we identified 26 unique operations for the search space, 13 of which are novel to object detection, that appear to cover the widest range of available transformations. These operations were implemented in TensorFlow [1]. We briefly summarize these operations, but reserve the details for the Appendix A.1:

– **Color operations**. Distort color channels, without impacting the locations of the bounding boxes (e.g., `Equalize`, `Contrast`, `Brightness`).[2]

[2] The color transformations largely derive from transformation in the Python Image Library (PIL). https://pillow.readthedocs.io/en/5.1.x/.

- **Geometric operations**. Geometrically distort the image, which correspondingly alters the location and size of the bounding box annotations (e.g., `Rotate`, `ShearX`, `TranslationY`, etc.). Note that for any operations that effect the geometry of an image, we likewise modify the bounding box size and location to maintain consistency.
- **Bounding box operations**. Only distort the pixel content contained within the bounding box annotations (e.g., `BBox_Only_Equalize`, `BBox_Only_Rotate`, `BBox_Only_FlipLR`).

Since many augmentation operations have a "strength" parameter, such as how many degrees to rotate an image, we associate with each operation a custom range of parameter values. We map this range onto a standardized range from 0 to 10 for all operations. We discretize the range of magnitude into L uniformly-spaced values so that these parameters are amenable to discrete optimization. Similarly, we discretize the probability of applying an operation into M uniformly-spaced values. In preliminary experiments we found that setting $L = 6$ and $M = 6$ provides a good balance between computational tractability and learning performance with an RL algorithm. Thus, finding a good subpolicy becomes a search in a discrete space containing a cardinality of $(26LM)^2$. In particular, to search over 5 sub-policies, the search space contains roughly $(26 \times 6 \times 6)^{2\times 5} \approx 5.2 \times 10^{29}$ possibilities and requires an efficient search technique to navigate this space. This number comes from the fact that each operation in a subpolicy has 26 transformation options and there are also 6 options for the probability and 6 options for the magnitude. This number gets raised to the (2×5) as there are 2 operations per subpolicy and 5 different subpolicies.

2.2 Controller Settings

Now that we have our search space setup, we want to optimize it to find the augmentation policy that allows the model to achieve the best validation set performance. As done in other work we will have a controller that will predict an augmentation policy, which will be used to train a neural network (child model) on a detection dataset. After training the network we evaluate its validation accuracy to judge how well the augmentation policy performed. Using this signal we update the controller to generate better and better augmentation policies over time according to the validation set.

Many methods exist for addressing the discrete optimization problem of training the controller including reinforcement learning [47], evolutionary methods [31] and sequential model-based optimization [25]. In this work, we choose to build on previous work by structuring the discrete optimization problem as the output space of an RNN and employ reinforcement learning to update the weights of the model [47]. The training setup for the RNN is similar to [3,4,47,48]. We employ the proximal policy optimization (PPO) [37] for the search algorithm.

The RNN is unrolled 30 steps to predict a single augmentation policy. The number of unrolled steps, 30, corresponds to the number of discrete predictions

that must be made in order to enumerate 5 sub-policies. Each sub-policy consists of 2 operations and each operation consists of 3 predictions corresponding to the selected image transformation, probability of application and magnitude of the transformation.

In order to train each child model, we selected 5K images from the COCO training set as we found that searching directly on the full COCO dataset to be prohibitively expensive. We found that policies identified with this subset of data generalize to the full dataset while providing significant computational savings. Briefly, we trained each child model[3] from scratch on the 5K COCO images with the ResNet-50 backbone [14] and RetinaNet detector [23] using a cosine learning rate decay [27]. The reward signal for the controller is the mAP on a custom held-out validation set of 7392 images created from a subset of the COCO training set.

The RNN controller is trained over 20K augmentation policies. The search employed 400 TPU's [18] over 48 hours with identical hyper-parameters for the controller as [48]. The search can be sped up using the recently developed, more efficient search methods based on population based training [15] or density matching [21]. Since our learned augmentation method is being used as a method to study and evaluate the performance of data augmentation on COCO, we leave the algorithmic speedup to future work. The learned augmentation policy can be seen in Table 7 in the Appendix.

3 Experiments

We applied our search method to the COCO dataset with a ResNet-50 [14] backbone with RetinaNet [23]. We are mainly interested in answering the following two questions:

- How important is data augmentation for object detection?
- How generalizable are the found data augmentation policies?

To answer the first question, we compare the improvement of our AutoAugment data augmentation policy to changing the model architecture across various sizes. We additionally show that the augmentation policy can push the state-of-the-art using a much simpler system than previous results on COCO. To answer the second question, we use the top policy found on COCO and apply it to different datasets, dataset sizes and architecture configurations to examine generalizability. Finally, we study properties of what kinds of operations are needed for a good augmentation policy on an object detection dataset.

[3] We employed a base learning rate of 0.08 over 150 epochs; image size was 640×640; $\alpha = 0.25$ and $\gamma = 1.5$ for the focal loss parameters; weight decay of $1e{-}4$; batch size was 64.

3.1 Understanding the Policies Found by AutoAugment

Searching for the data augmentation strategy on 5K COCO training images resulted in the final augmentation policy that will be used in all of our results. Before diving into the results, we would like to inspect the best policy found during the search to gain a better understanding of what operations are used. Upon inspection, the most commonly used operation in good policies is `Rotate`, which rotates the whole image and the bounding boxes. The bounding boxes end up larger after the rotation, to include all of the rotated objects. Despite this effect of the `Rotate` operation, it seems to be very beneficial: it is the most frequently used operation in good policies. Two other operations that are commonly used are `Equalize` and `BBox_Only_TranslateY`. `Equalize` flattens the histogram of the pixel values, and does not modify the location or size of each bounding box. `BBox_Only_TranslateY` translates only the objects in bounding boxes vertically, up or down with equal probability. This is quite encouraging as some operations, such as `BBox_Only_TranslateY`, uniquely act on the bounding boxes. Utilizing a combination of color, geometric and bounding box specific operations appears to be crucial to creating an optimal data augmentation policy for object detection.

3.2 Data Augmentation Policy Found by AutoAugment Systematically Improves Object Detection

We assess the quality of the data augmentation policy found by AutoAugment on the competitive COCO dataset [24] on different backbone architectures and detection algorithms. We start with the competitive RetinaNet object detector[4] employing the same training protocol as [9]. Briefly, we train from scratch with a global batch size of 64, images are resized to 640×640, learning rate of 0.08, weight decay of $1e-4$, $\alpha = 0.25$ and $\gamma = 1.5$ for the focal loss parameters, train the models for 150 epochs, use stepwise decay with the learning rate being reduced by a factor of 10 at epochs 120 and 140. All models were trained on TPUs [18].

The baseline RetinaNet architecture used in this and subsequent sections employs standard data augmentation techniques typically used for object detection training [23]. This consists of doing horizontal flipping with 50% probability and multi-scale jittering where images are randomly resized between 512 and 786 during training and then cropped to 640×640.

Our results using our augmentation policy found by AutoAugment using the above procedures are shown in Tables 1 and 2. In Table 1, the data augmentation policy achieves systematic gains across several backbone architectures with surprising improvements ranging from +1.6 mAP to +2.3 mAP. In comparison, a previous state-of-the-art regularization technique (DropBlock) applied to ResNet-50 [9] only achieves a gain of +1.7 mAP. Additionally, going from a ResNet-50 model to ResNet-101 achieves a 2.1 mAP gain and going from

[4] https://github.com/tensorflow/tpu.

a ResNet-101 to ResNet-200 achieves a 1.1 mAP gain. Our data augmentation policy achieves a 2.3 gain on ResNet-50, which is a larger improvement than substantially increasing the architecture size, while incurring no additional inference cost. Clearly we see that changing augmentation can be as, if not more, powerful than changing around the underlying architectural components.

Table 1. Improvements with AutoAugment data augmentation policy across different ResNet backbones. All results employ RetinaNet detector [23] on the COCO dataset [24]

Backbone	Baseline	AutoAugment	Difference
ResNet-50	36.7	39.0	+2.3
ResNet-101	38.8	40.4	+1.6
ResNet-200	39.9	42.1	+2.2

To better understand where augmentation benefits, we break the data augmentation policies applied to ResNet-50 into three parts: color operations, geometric operations, and bbox-only-operations (Table 2). Employing color operations only boosts performance by +0.8 mAP. Combining the search with geometric operations increases the boost in performance by +1.9 mAP. Finally, adding bounding box-specific operations yields the best results when used in conjunction with the previous operations and provides +2.3 mAP improvement over the baseline.

Table 2. Improvements in object detection with the data augmentation policy. All results employ RetinaNet detector with ResNet-50 backbone [23] on COCO dataset [24].

Method	mAP
baseline	36.7
baseline + DropBlock [9]	38.4
AutoAugment with color operations	37.5
+ geometric operations	38.6
+ bbox-only operations	**39.0**

Interestingly, we observe that the custom operations designed for object detection (geometric operations and bbox-only operations) contributes 1.5 mAP of the 2.3 mAP gain from this data augmentation policy. Using object detection specific operations is clearly beneficial when trying to find good augmentation policies. This further confirms our result that a diversity of augmentation operations spanning color, geometric and unique bounding box only operations are

needed to make a high performing augmentation policy. Also note that the policy found was only searched using 5K COCO training examples and still generalizes well when trained on the full COCO dataset.

3.3 Data Augmentation Policy Found by AutoAugment Push the State-of-the-Art on Object Detection Models

A good data augmentation policy is one that can transfer between models, between datasets and work well for models trained on different image sizes. Here we experiment with the AutoAugment data augmentation policy on a different backbone architecture and detection model. To test how the data augmentation policy transfers to a state-of-the-art detection model, we replace the ResNet-50 backbone with the AmoebaNet-D architecture [31]. The feature-pyramid network [22] was changed to NAS-FPN [10]. Additionally, we use ImageNet pre-training for the AmoebaNet-D backbone as we found we are not able to achieve competitive results when training from scratch. The model was trained for 150 epochs using a cosine learning rate decay with a learning rate of 0.08. The rest of the setup was identical to the ResNet-50 backbone model except the image size was increased from 640×640 to 1280×1280.

Table 3 indicates that the data augmentation policy improves +1.5 mAP on top of a competitive detection architecture and setup. These experiments show that the augmentation policy transfers well across a different backbone architecture, feature pyramid network, image sizes (i.e. $640 \rightarrow 1280$ pixels), and training procedure (training from scratch $\rightarrow$ using ImageNet pre-training). This is a surprising result that shows our data augmentation policy is quite general. We can extend these results even further by increasing the image resolution from 1280 to 1536 pixels and likewise increasing the number of detection anchors[5] following [44].

This result of these simple modifications is the first single-stage detection system to achieve state-of-the-art, single-model results of 50.7 mAP on COCO. We note that this result only requires a single pass of the image, where as the previous results required multiple evaluations of the same image at different spatial scales at test time [29]. Additionally, these results were arrived at by increasing the image resolution and increasing the number of anchors - both simple and well known techniques for improving object detection performance [16,44]. In contrast, previous state-of-the-art results relied on multiple, custom modifications of the model architecture and regularization methods in order to achieve these results [29]. Our method largely relies on a more modern network architecture paired with a learned data augmentation policy.

[5] Specifically, we increase the number of anchors from 3×3 to 9×9 by changing the aspect ratios from {1/2, 1, 2} to {1/5, 1/4, 1/3, 1/2, 1, 2, 3, 4, 5}. When making this change we increased the strictness in the IoU thresholding from 0.5/0.5 to 0.6/0.5 due to the increased number of anchors following [44]. The anchor scale was also increased from 4 to 5 to compensate for the larger image size.

Table 3. Exceeding state-of-the-art detection with the AutoAugment data augmentation policy. Reporting mAP for COCO validation set. Previous state-of-the-art results for COCO detection evaluated a single image at multiple spatial scales to perform detection at test time [29]. Our current results only require a single inference computation at a single spatial scale. The backbone model is AmoebaNet-D [31] with NAS-FPN as the feature pyramid network [10]. For the **50.7** result, in addition to using the data augmentation policy, we increase the image size from 1280 to 1536 and the number of detection anchors from 3×3 to 9×9.

Architecture	Change	# Scales	mAP	mAP_S	mAP_M	mAP_L
MegDet [29]		multiple	50.5	–	–	–
AmoebaNet + NAS-FPN	baseline [10]	1	47.0	30.6	50.9	61.3
	+ AutoAugment policy	1	48.6	32.0	53.4	62.7
	+ ↑ anchors, ↑ image size	1	**50.7**	**34.2**	**55.5**	**64.5**

3.4 Data Augmentation Policy Found by AutoAugment Transfers to Other Detection Datasets

To evaluate the transferability of the data augmentation policy to an entirely different dataset and a different detection algorithm, we train a Faster R-CNN [35] model with a ResNet-101 backbone on PASCAL VOC dataset [8]. We combine the training sets of PASCAL VOC 2007 and PASCAL VOC 2012, and test our model on the PASCAL VOC 2007 test set (4952 images). Our evaluation metric is the mean average precision at an IoU threshold of 0.5 (mAP50). For the baseline model, we use the Tensorflow Object Detection API [16] with the default hyperparameters: 9 GPU workers are utilized for asynchronous training where each worker processes a batch size of 1. Initial learning rate is set to be 3×10^{-4}, which is decayed by 0.1 after 500K steps. Training is started from a COCO detection model checkpoint. When training with our data augmentation policy, we do not change any of the training details, and just add our policy found on COCO to the pre-processing. This leads to a 2.7% improvement on mAP50 (Table 4).

Table 4. Data augmentation policy transfers to other object detection tasks. Mean average precision (%) at IoU threshold 0.5 on a Faster R-CNN detector [35] with a ResNet-101 backbone trained and evaluated on PASCAL VOC 2007 [8]. Note that the augmentation policy was learned from the policy search on the COCO dataset

	plane	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
baseline	86.6	82.2	75.9	63.4	62.3	84.7	86.8	92.0	55.5	83.3	63.1	89.2	89.4	85.0	85.6	50.7	76.2	73.0	86.6	76.3	76.0
ours	88.0	83.3	78.0	65.9	63.5	85.5	87.4	93.1	58.5	83.9	65.2	90.1	90.2	85.9	86.6	55.2	78.6	76.6	88.6	80.3	78.7

This result is surprising because the best policy found on COCO may appear to be too intricate to generalize to other datasets. But the result confirms that just like architectures, data augmentation policies transfer well across datasets. This means that the augmentations learned on the COCO dataset are very generic and can be used for many other object detection datasets in the future.

4 Analysis

In this section, we analyze the impact of training with the augmentation policy in more detail. We find that:

- Relative improvement of AP due to the augmentation policy is larger for smaller datasets. This is good news since data augmentation policies are needed mostly for models that have small amount of data available.
- Relative improvement is larger for more difficult detection tasks. Average precision for small objects as well as average precision at more strict thresholds benefit more from the learned augmentation.
- Data augmentation regularizes the detection model, which can be seen either by the increased training loss or decreased magnitude of the trainable weights.
- Having data augmentation operations that modify the locations and the sizes of the objects in the search space is important for achieving good results with the augmentation policy.

Below we describe these points in detail.

4.1 Data Augmentation Policy Found by AutoAugment Mimics the Performance of Larger Annotated Datasets

In this section we conducted experiments to determine how the data augmentation policy will perform if there is more or less training data. To conduct these experiments we took subsets of the COCO dataset to make datasets with the following number of images: 5000, 9000, 14000, 23000 (see Table 5). All models trained in this experiment are using a ResNet-50 backbone with RetinaNet and are trained for 150 epochs without using ImageNet pretraining.

Table 5. Data augmentation policy is especially beneficial for small datasets and small objects. Mean average precision (mAP) for RetinaNet model trained on COCO with varying subsets of the original training set. mAP_S, mAP_M and mAP_L denote the mean average precision for small, medium and large examples. Note the complete COCO training set consists of 118K examples. The same policy found on the 5K COCO images was used in all of the experiments. The models in the first row were trained on the same 5K images that the policies were searched on.

Training	Baseline				Our results			
Set size	mAP_S	mAP_M	mAP_L	mAP	mAP_S	mAP_M	mAP_L	mAP
5000	1.9	7.1	9.7	6.5	3.2	9.8	12.7	8.7
9000	4.3	12.3	17.6	11.8	7.1	16.8	22.3	15.1
14000	6.8	17.5	23.9	16.4	9.5	22.1	29.8	19.9
23000	10.0	24.3	33.3	22.6	11.9	27.8	36.8	25.3

As we expected, the improvements due to the data augmentation policy is larger when the model is trained on smaller datasets, which can be seen in Fig. 2

and in Table 5. We show that for models trained on 5000 training samples, the data augmentation policy can improve mAP by more than 70% relative to the baseline. As the training set size is increased, the effect of the data augmentation policy is decreased, although the improvements are still significant. It is interesting to note that models trained with the data augmentation policy seem to do especially well on detecting smaller objects, especially when fewer images are present in the training dataset. For example, for small objects, applying the data augmentation policy seems to be better than increasing the dataset size by 50%, as seen in Table 5. This is quite a striking finding as in many detection applications detecting small objects is of great importance. For small objects, training with the data augmentation policy with 9000 examples results in better performance than the baseline when using 15000 images. In this scenario using our augmentation policy is almost as effective as doubling your dataset size.

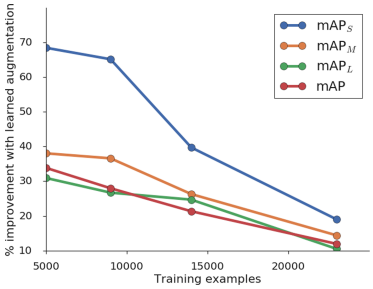

Fig. 2. Percentage improvement in mAP for objects of different sizes due to the data augmentation policy.

Another interesting behavior of models trained with the data augmentation policy is that they do relatively better on the harder task of AP75 (average precision IoU = 0.75). In Fig. 3, we plot the percentage improvement in mAP, AP50, and AP75 for models trained with the data augmentation policy (relative to baseline augmentation). The relative improvement of AP75 is larger than that of AP50 for all training set sizes. The data augmentation policy is particularly beneficial at AP75 indicating that the augmentation policy helps with more precisely aligning the bounding box prediction. This suggests that the augmentation policy particularly helps with learned fine spatial details in bounding box position – which is consistent with the gains observed with small objects.

4.2 Data Augmentation Improves Model Regularization

In this section, we study the regularization effect of the data augmentation policy. We first notice that the final training loss of a detection model is lower when trained on a larger training set (see black curve in the left plot in Fig. 4). When we apply the data augmentation policy, the training loss is increased significantly

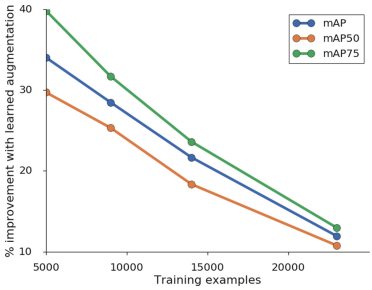

Fig. 3. Percentage improvement due to the data augmentation policy on mAP, AP50, and AP75, relative to models trained with baseline augmentation.

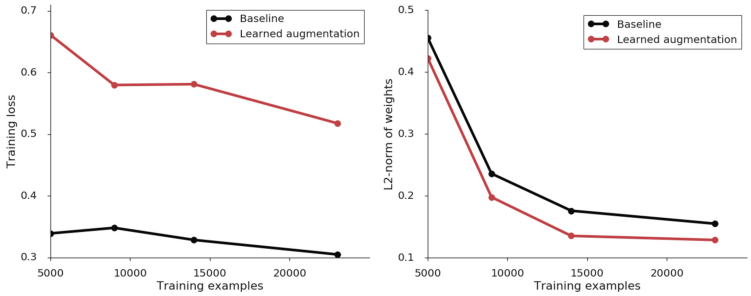

Fig. 4. Two plots showing data augmentation regularizes detection models. On the left is training loss vs. number of training examples for baseline model (black) and with the data augmentation policy (red). One the right is L_2 norm of the weights of the baseline (black) and our (red) models at the end of training. Note that the L_2 norm of the weights decrease with increasing training set size. The data augmentation policy further decreases the norm of the weights. (Color figure online)

for all dataset sizes (red curve). The regularization effect can also be seen by looking at the L_2 norm of the weights of the trained models. The L_2 norm of the weights is smaller for models trained on larger datasets, and models trained with the data augmentation policy have a smaller L_2 norm than models trained with baseline augmentation (see right plot in Fig. 4).

5 Related Work

Data augmentation strategies for vision models are often focused on the image classification domain [3,6,15,17,21,28,41]. For example, state-of-the-art classification models trained on MNIST use elastic distortions which effect scale, translation, and rotation [2,36,38,42]. Random cropping and image mirroring are commonly used in classification models trained on natural images [19,45]. Among the limited data augmentation strategies for object detection, image mirror and multi-scale training are the most widely used [12]. Object-centric cropping is also a popular augmentation approach [26]. Instead of cropping to focus

on parts of the image, some methods randomly erase image contents for augmentation [9,46]. In the same vein, [43] learns an occlusion pattern for each object to create adversarial examples. In addition to cropping and erasing, [7] adds new objects on training images by cut-and-paste. While these object-detection approaches work decently well, there is a real lack of studying how truly transferable they are, doing compositions of many different augmentation methods at once and not typically performing as well as modeling changes [10].

To avoid the overwhelming amount of options when designing a data augmentation policy, recent work has focused on learning data augmentation strategies directly from data itself. For example, Smart Augmentation uses a network that generates new data by merging two or more samples from the same class [20]. Tran et al. generate augmented data, using a Bayesian approach, based on the distribution learned from the training set [40]. DeVries and Taylor used simple transformations like noise, interpolations and extrapolations in the learned feature space to augment data [5]. Ratner et al., used generative adversarial networks to generate sequences of data augmentation operations [30]. More recently, several papers use the AutoAugment [3] search space with improved optimization algorithms to find AutoAugment policies more efficiently [15,21].

The above learned augmentation approaches were found to be quite effective in the classification domain due to the complexity of designing a good augmentation procedure. When designing augmentation policies for object detection the complexity only increases. Unlike classification, labeled data for object detection is more scarce because it is more costly to annotate detection data. Compared to image classification, developing a data augmentation strategy for object detection is harder because there are more complexities introduced by distorting the image, bounding box locations, and the sizes of the objects in detection datasets. Furthermore, it is much less clear that augmentation policies are transferable due to images having a richer label structure and the models and detection algorithms being more complex. Our goal is to show that these added complexities are handle-able using learned augmentation procedures and that high performing data augmentation policies can be found. We surprisingly find that these policies are highly generalizable across difference datasets, models and detection algorithms.

6 Discussion

In this work, we challenge the common belief that focusing on changing the detection model is the most promising research direction. Our augmentation procedure gets larger improvements than increasing the model size, while incurring no additional inference cost and minimal training cost. And although data augmentation strategies can be intricate, they can be as transferable as architectures. Our augmentation policy learned on COCO transfers to PASCAL with great performance. Additionally, we are able to further improve the state-of-the-art on COCO using our learned augmentation policy found on a small 5K subset of the COCO dataset with much smaller model, a different image resolution and detection algorithm.

Acknowledgements. We thank the larger teams at Google Brain for their help and support. We also thank Dumitru Erhan for detailed comments on the manuscript.

A Appendix

A.1 AutoAugment Controller Training Details

Table 6. Table of all the possible transformations that can be applied to an image. These are the transformations that are available to the controller during the search process. The range of magnitudes that the controller can predict for each of the transforms is listed in the third column. Some transformations do not have a magnitude associated with them (e.g. Equalize).

Operation name	Description	Range of magnitudes
ShearX(Y)	Shear the image and the corners of the bounding boxes along the horizontal (vertical) axis with rate *magnitude*	[−0.3,0.3]
TranslateX(Y)	Translate the image and the bounding boxes in the horizontal (vertical) direction by *magnitude* number of pixels	[−150,150]
Rotate	Rotate the image and the bounding boxes *magnitude* degrees	[−30,30]
Equalize	Equalize the image histogram	
Solarize	Invert all pixels above a threshold value of *magnitude*	[0,256]
SolarizeAdd	For each pixel in the image that is less than 128, add an additional amount to it decided by the magnitude	[0,110]
Contrast	Control the contrast of the image. A *magnitude* = 0 gives a gray image, whereas *magnitude* = 1 gives the original image	[0.1,1.9]
Color	Adjust the color balance of the image, in a manner similar to the controls on a colour TV set. A *magnitude* = 0 gives a black & white image, whereas *magnitude* = 1 gives the original image	[0.1,1.9]
Brightness	Adjust the brightness of the image. A *magnitude* = 0 gives a black image, whereas *magnitude* = 1 gives the original image	[0.1,1.9]
Sharpness	Adjust the sharpness of the image. A *magnitude* = 0 gives a blurred image, whereas *magnitude*=1 gives the original image	[0.1,1.9]
Cutout [6,46]	Set a random square patch of side-length *magnitude* pixels to gray	[0,60]
BBox_Only_X	Apply X to each bounding box content with independent probability, and magnitude that was chosen for X above. Location and the size of the bounding box are not changed	

Table 7. The sub-policies used in our learned augmentation policy. P and M correspond to the probability and magnitude with which the operations were applied in the sub-policy. Note that for each image in each mini-batch, one of the sub-policies is picked uniformly at random. The *No operation* is listed when an operation has a learned probability or magnitude of 0

	Operation 1	P	M	Operation 2	P	M
Sub-policy 1	TranslateX	0.6	4	Equalize	0.8	10
Sub-policy 2	BBox_Only_TranslateY	0.2	2	Cutout	0.8	8
Sub-policy 3	ShearY	1.0	2	BBox_Only_TranslateY	0.6	6
Sub-policy 4	Rotate	0.6	10	Color	1.0	6
Sub-policy 5	No operation			No operation		

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI 2016, pp. 265–283. USENIX Association, Berkeley (2016)
2. Ciregan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp. 3642–3649. IEEE (2012)
3. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: AutoAugment: learning augmentation policies from data. arXiv preprint arXiv:1805.09501 (2018)
4. Cubuk, E.D., Zoph, B., Schoenholz, S.S., Le, Q.V.: Intriguing properties of adversarial examples. arXiv preprint arXiv:1711.02846 (2017)
5. DeVries, T., Taylor, G.W.: Dataset augmentation in feature space. arXiv preprint arXiv:1702.05538 (2017)
6. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552 (2017)
7. Dwibedi, D., Misra, I., Hebert, M.: Cut, paste and learn: surprisingly easy synthesis for instance detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1301–1310 (2017)
8. Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A.: The pascal visual object classes (VOC) challenge. Int. J. Comput. Vision **88**(2), 303–338 (2010)
9. Ghiasi, G., Lin, T.Y., Le, Q.V.: DropBlock: a regularization method for convolutional networks. In: Advances in Neural Information Processing Systems, pp. 10750–10760 (2018)
10. Ghiasi, G., Lin, T.Y., Pang, R., Le, Q.V.: NAS-FPN: learning scalable feature pyramid architecture for object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2019
11. Girshick, R.: Fast R-CNN. In: The IEEE International Conference on Computer Vision (ICCV), December 2015
12. Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., He, K.: Detectron (2018)
13. He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask R-CNN. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2961–2969 (2017)
14. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)

15. Ho, D., Liang, E., Stoica, I., Abbeel, P., Chen, X.: Population based augmentation: efficient learning of augmentation policy schedules. arXiv preprint arXiv:1905.05393 (2019)
16. Huang, J., et al.: Speed/accuracy trade-offs for modern convolutional object detectors. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 7310–7311 (2017)
17. Inoue, H.: Data augmentation by pairing samples for images classification. arXiv preprint arXiv:1801.02929 (2018)
18. Jouppi, N.P., et al.: In-datacenter performance analysis of a tensor processing unit. In: 2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), pp. 1–12. IEEE (2017)
19. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems (2012)
20. Lemley, J., Bazrafkan, S., Corcoran, P.: Smart augmentation learning an optimal data augmentation strategy. IEEE Access **5**, 5858–5869 (2017)
21. Lim, S., Kim, I., Kim, T., Kim, C., Kim, S.: Fast autoaugment. arXiv preprint arXiv:1905.00397 (2019)
22. Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125 (2017)
23. Lin, T.Y., Goyal, P., Girshick, R., He, K., Dollár, P.: Focal loss for dense object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2980–2988 (2017)
24. Lin, T.-Y., et al.: Microsoft COCO: common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8693, pp. 740–755. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10602-1_48
25. Liu, C., et al.: Progressive neural architecture search. arXiv preprint arXiv:1712.00559 (2017)
26. Liu, W., et al.: SSD: single shot multibox detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46448-0_2
27. Loshchilov, I., Hutter, F.: SGDR: stochastic gradient descent with warm restarts. arXiv preprint arXiv:1608.03983 (2016)
28. Miyato, T., Maeda, S.i., Koyama, M., Ishii, S.: Virtual adversarial training: a regularization method for supervised and semi-supervised learning. In: International Conference on Learning Representations (2016)
29. Peng, C., et al.: MegDet: a large mini-batch object detector. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018
30. Ratner, A.J., Ehrenberg, H., Hussain, Z., Dunnmon, J., Ré, C.: Learning to compose domain-specific transformations for data augmentation. In: Advances in Neural Information Processing Systems, pp. 3239–3249 (2017)
31. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Thirty-Third AAAI Conference on Artificial Intelligence (2019)
32. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2016
33. Redmon, J., Farhadi, A.: YOLO9000: Better, faster, stronger. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017

34. Redmon, J., Farhadi, A.: YOLOV3: an incremental improvement. CoRR abs/1804.02767 (2018). http://arxiv.org/abs/1804.02767
35. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: Advances in Neural Information Processing Systems, pp. 91–99 (2015)
36. Sato, I., Nishimura, H., Yokoi, K.: APAC: augmented pattern classification with neural networks. arXiv preprint arXiv:1505.03229 (2015)
37. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
38. Simard, P.Y., Steinkraus, D., Platt, J.C., et al.: Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of International Conference on Document Analysis and Recognition (2003)
39. Tan, M., Pang, R., Le, Q.V.: EfficientDet: scalable and efficient object detection. arXiv preprint arXiv:1911.09070 (2019)
40. Tran, T., Pham, T., Carneiro, G., Palmer, L., Reid, I.: A Bayesian data augmentation approach for learning deep models. In: Advances in Neural Information Processing Systems, pp. 2794–2803 (2017)
41. Verma, V., Lamb, A., Beckham, C., Courville, A., Mitliagkis, I., Bengio, Y.: Manifold mixup: encouraging meaningful on-manifold interpolation as a regularizer. arXiv preprint arXiv:1806.05236 (2018)
42. Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., Fergus, R.: Regularization of neural networks using dropconnect. In: International Conference on Machine Learning, pp. 1058–1066 (2013)
43. Wang, X., Shrivastava, A., Gupta, A.: A-fast-RCNN: hard positive generation via adversary for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2606–2615 (2017)
44. Yang, T., Zhang, X., Li, Z., Zhang, W., Sun, J.: MetaAnchor: learning to detect objects with customized anchors. In: Advances in Neural Information Processing Systems, pp. 318–328 (2018)
45. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: British Machine Vision Conference (2016)
46. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. arXiv preprint arXiv:1708.04896 (2017)
47. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: International Conference on Learning Representations (2017)
48. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (2017)