Learning Data Augmentation Strategies for Object Detection

Barret Zoph*, Ekin D. Cubuk*, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, Quoc V. Le
Google Research, Brain Team
{barretzoph, cubuk, golnazg, tsungyi, shlens, qvl}@google.com

Abstract

Data augmentation is a critical component of training deep learning models. Although data augmentation has been shown to significantly improve image classification, its potential has not been thoroughly investigated for object detection. Given the additional cost for annotating images for object detection, data augmentation may be of even greater importance for this computer vision task. In this work, we study the impact of data augmentation on object detection. We first demonstrate that data augmentation operations borrowed from image classification may be helpful for training detection models, but the improvement is limited. Thus, we investigate how learned, specialized data augmentation policies improve generalization performance for detection models. Importantly, these augmentation policies only affect training and leave a trained model unchanged during evaluation. Experiments on the COCO dataset indicate that an optimized data augmentation policy improves detection accuracy by more than +2.3 mAP, and allow a single inference model to achieve a state-of-the-art accuracy of 50.7 mAP. Importantly, the best policy found on COCO may be transferred unchanged to other detection datasets and models to improve predictive accuracy. For example, the best augmentation policy identified with COCO improves a strong baseline on PASCAL-VOC by +2.7 mAP. Our results also reveal that a learned augmentation policy is superior to state-of-the-art architecture regularization methods for object detection, even when considering strong baselines. Code for training with the learned policy is available online. [1]

1. Introduction

Deep neural networks are powerful machine learning systems that work best when trained on vast amounts of data. To increase the amount of training data for neural networks, much work was devoted to creating better data augmentation strategies [3, 42, 21]. In the image domain, common augmentations include translating the image by a few pixels, or flipping the image horizontally. Most modern image classifiers are paired with hand-crafted data augmentation strategies [21, 44, 16, 18, 56].

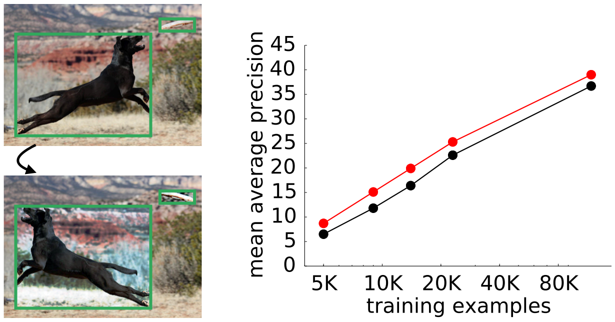


Figure 1: **Learned augmentation policy systematically improves object detection performance**. Left: Learned augmentation policy applied to example from COCO dataset [25]. Right: Mean average precision for RetinaNet [24] with a ResNet-50 backbone on COCO [25] with and without learned augmentation policy (red and black, respectively).

Recent work has shown that instead of manually designing data augmentation strategies, learning an optimal policy from data can lead to significant improvements in generalization performance of image classification models [22, 45, 8, 33, 31, 54, 2, 43, 37, 5]. For image classification models, data can be augmented either by learning a generator that can create data from scratch [33, 31, 54, 2, 43], or by learning a set of transformations as applied to already existing training set samples [5, 37]. For object detection models, the need for data augmentation is more crucial as collecting labeled data for detection is more costly and common detection datasets have many fewer examples than image classification datasets. It is, however, unclear how to augment the data: Should we directly reuse data augmentation strategies from image classification? What should we do

*Equal contribution.

[1] github.com/tensorflow/tpu/tree/master/models/official/detection

with the bounding boxes and the contents of the bounding boxes?

In this work, we create a set of simple transformations that may be applied to object detection datasets and then transfer these transformations to other detection datasets and architectures. These transformations are only used during training and not test time. Our transformations include those that can be applied to the whole image without affecting the bounding box locations (e.g. color transformations borrowed from image classification models), transformations that affect the whole image while changing the bounding box locations (e.g., translating or shearing of the whole image), and transformations that are only applied to objects within the bounding boxes. As the number of transformations becomes large, it becomes non-trivial to manually combine them effectively. We therefore search for policies specifically designed for object detection datasets. Experiments show that this method achieves very good performance across different datasets, dataset sizes, backbone architectures and detection algorithms. Additionally, we investigate how the performance of a data augmentation policy depends on the number of operations included in the search space and how the effective of the augmentation technique varies as dataset size changes.

In summary, our main contributions are as follows:

- Design and implement a search method to combine and optimize data augmentation policies for object detection problems by combining novel operations specific to bounding box annotations.

- Demonstrate consistent gains in cross-validated accuracy across a range of detection architectures and datasets. In particular, we exceed state-of-the-art results on COCO for a single model and achieve competitive results on the PASCAL VOC object detection.

- Highlight how the learned data augmentation strategies are particularly advantageous for small datasets by providing a strong regularization to avoid over-fitting on small objects.

2. Related Work

Data augmentation strategies for vision models are often specific dataset or even machine learning architectures. For example, state-of-the-art models trained on MNIST use elastic distortions which effect scale, translation, and rotation [42, 4, 47, 40]. Random cropping and image mirroring are commonly used in classification models trained on natural images [51, 21]. Among the data augmentation strategies for object detection, image mirror and multi-scale training are the most widely used [15]. Object-centric cropping is a popular augmentation approach [27]. Instead of cropping to focus on parts of the image, some methods randomly erase or add noise to patches of images for improved accuracy [9, 53, 13], robustness [50, 12], or both [29]. In the same vein, [48] learns an occlusion pattern for each object to create adversarial examples. In addition to cropping and erasing, [10] adds new objects on training images by cut-and-paste.

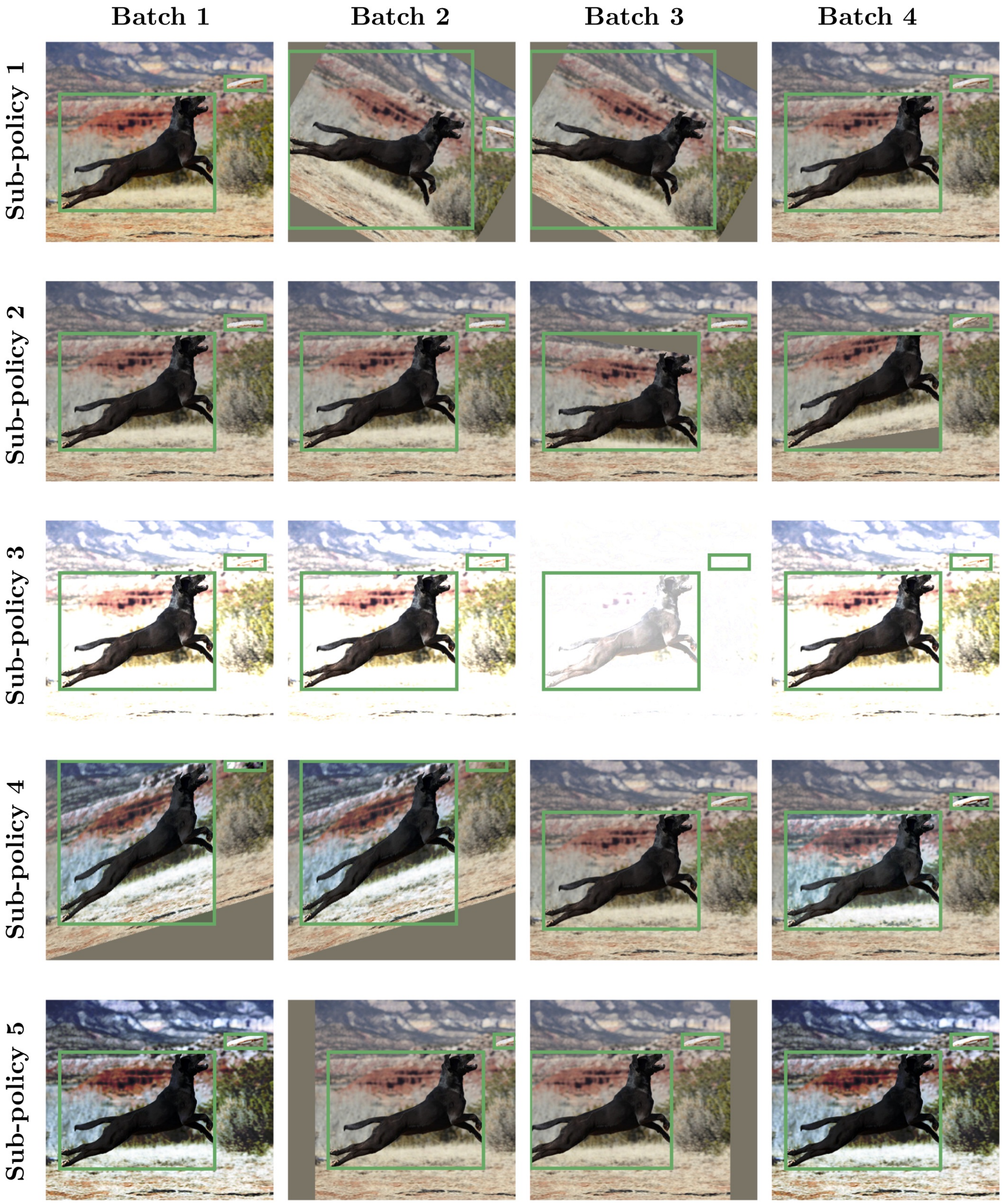
To avoid the data-specific nature of data augmentation, recent work has focused on learning data augmentation strategies directly from data itself. For example, Smart Augmentation uses a network that generates new data by merging two or more samples from the same class [22]. Tran et al. generate augmented data, using a Bayesian approach, based on the distribution learned from the training set [45]. DeVries and Taylor used simple transformations like noise, interpolations and extrapolations in the learned feature space to augment data [8]. Ratner et al., used generative adversarial networks to generate sequences of data augmentation operations [37]. More recently, several papers used the AutoAugment [5] search space with improved the optimization algorithms to find AutoAugment policies more efficiently [17, 23].

While all of the above approaches have worked on classification problems, we take an automated approach to finding optimal data augmentation policies for object detection. 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 ways and complexities introduced by distorting the image, bounding box locations, and the sizes of the objects in detection datasets. Our goal is to use the validation set accuracy to help search for novel detection augmentation procedures using custom operations that generalize across datasets, dataset sizes, backbone architectures and detection algorithms.

3. Methods

We treat data augmentation search as a discrete optimization problem and optimize for generalization performance. This work expands on previous work [5] to focus on augmentation policies for object detection. Object detection introduces an additional complication of maintaining consistency between a bounding box location and a distorted image. Bounding box annotations open up the possibility of introducing augmentation operations that uniquely act upon the contents within each bounding box. Additionally, we explored how to change the bounding box locations when geometric transformations are applied to the image.

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



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)`

Figure 2: **Examples of learned 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).

which are applied sequentially. We turn this problem of searching for a learned augmentation policy into a discrete optimization problem by creating a search space [5]. The 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 2 (bottom text) demonstrates 5 of the learned sub-policies. The probability parameter introduces a notion of stochasticity into the augmentation policy whereby the selected augmentation operation will be applied to the image with the specified probability.

In several preliminary experiments, we identified 22 operations for the search space that appear beneficial for object detection. These operations were implemented in TensorFlow [1]. We briefly summarize these operations, but reserve the details for the Appendix:

- **Color operations**. Distort color channels, without impacting the locations of the bounding boxes (e.g., Equalize, Contrast, Brightness). [2]
- **Geometric operations**. Geometrically distort the image, which correspondingly alters the location and size of the bounding box annotations (e.g., Rotate, ShearX, TranslationY, etc.).
- **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).

Note that for any operations that effected the geometry of an image, we likewise modified the bounding box size and location to maintain consistency.

We associate with each operation a custom range of parameter values and map this range on to a standardized range from 0 to 10. 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 $M = 6$ provide a good balance between computational tractability and learning performance with an RL algorithm. Thus, finding a good sub-policy becomes a search in a discrete space containing a cardinality of $(22LM)^2$. In particular, to search over 5 sub-policies, the search space contains roughly $(22 \times 6 \times 6)^{2 \times 5} \approx 9.6 \times 10^{28}$ possibilities and requires an efficient search technique to navigate this space.

Many methods exist for addressing the discrete optimization problem including reinforcement learning [55], evolutionary methods [38] and sequential model-based optimization [26]. 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 [55]. The training setup for the RNN is similar to [55, 56, 6, 5]. We employ the proximal policy optimization (PPO) [41] 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 [16] and RetinaNet detector [24] using a cosine learning rate decay [30]. 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 [20] over 48 hours with identical hyper-parameters for the controller as [56]. The search can be sped up using the recently developed, more efficient search methods based on population based training [17] or density matching [23]. The learned policy can be seen in Table 7 in the Appendix.

4. Results

We applied our automated augmentation method on the COCO dataset with a ResNet-50 [16] backbone with RetinaNet [24] in order to find good augmentation policies to generalize to other detection datasets. We use the top policy found on COCO and apply it to different datasets, dataset sizes and architecture configurations to examine generalizability and how the policy fares in a limited data regime.

4.1. Learning a data augmentation policy

Searching for the learned augmentation strategy on 5K COCO training images resulted in the final augmentation

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

[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

policy that will be used in all of our results. 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 object. 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.

4.2. Learned augmentation policy systematically improves object detection

We assess the quality of the top augmentation policy on the competitive COCO dataset [25] on different backbone architectures and detection algorithms. We start with the competitive RetinaNet object detector [4] employing the same training protocol as [13]. Briefly, we train from scratch with a global batch size of 64, images were 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, trained for 150 epochs, used stepwise decay where the learning rate was reduced by a factor of 10 at epochs 120 and 140. All models were trained on TPUs [20].

The baseline RetinaNet architecture used in this and subsequent sections employs standard data augmentation techniques largely tailored to image classification training [24]. 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 640x640.

Our results using our augmentation policy on the above procedures are shown in Tables 1 and 2. In Table 1 the learned augmentation policy achieves systematic gains across a several backbone architectures with improvements ranging from +1.6 mAP to +2.3 mAP. In comparison, a previous state-of-the-art regularization technique applied to ResNet-50 [13] achieves a gain of +1.7% mAP (Table 2).

To better understand where the gains come from, we break the data augmentation strategies 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. Note that the policy found was only searched using 5K COCO training examples and still generalizes extremely well when trained on the full COCO dataset.

Backbone	Baseline	Our result	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

Table 1: **Improvements with learned augmentation policy across different ResNet backbones.** All results employ RetinaNet detector [24] on the COCO dataset [25].

Method	mAP
baseline	36.7
baseline + DropBlock [13]	38.4
Augmentation policy with color operations	37.5
+ geometric operations	38.6
+ bbox-only operations	**39.0**

最有效的三种变换依次是：
1、几何变换；
2、颜色操作；
3、只对bb的操作

Table 2: **Improvements in object detection with learned augmentation policy.** All results employ RetinaNet detector with ResNet-50 backbone [24] on COCO dataset [25]. DropBlock shows gain in performance employing a state-of-the-art regularization method [13].

4.3. Exploiting learned augmentation policies achieves state-of-the-art object detection

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 learned augmentation policy on a different backbone architecture and detection model. To test how the learned policy transfers to a state-of-the-art detection model, we replace the ResNet-50 backbone with the AmoebaNet-D architecture [38]. The detection algorithm was changed from RetinaNet [24] to NAS-FPN [14]. 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 learned augmentation policy improves +1.5% mAP on top of a competitive, detection architecture and setup. These experiments additionally show that the augmentation policy transfers well across a different backbone architecture, detection algorithm, image sizes (i.e. 640 → 1280 pixels), and training procedure (training from scratch → using ImageNet pre-training) . We can extend these results even further by increasing the image resolution from 1280 to 1536 pixels and likewise increasing

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

the number of detection anchors[5] following [49]. Since this model is significantly larger than the previous models, we increase the number of sub-policies in the learned policy by combining the top 4 policies from the search, which leads to a 20 sub-policy learned augmentation.

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 [32]. 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 [49, 19]. In contrast, previous state-of-the-art results relied on roughly multiple, custom modifications of the model architecture and regularization methods in order to achieve these results [32]. Our method largely relies on a more modern network architecture paired with a learned data augmentation policy.

4.4. Learned augmentation policies transfer to other detection datasets.

To evaluate the transferability of the learned policies to an entirely different dataset and another different detection algorithm, we train a Faster R-CNN [39] model with a ResNet-101 backbone on PASCAL VOC dataset [11]. 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 [19] 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).

4.5. Learned augmentation policies mimic the performance of larger annotated datasets

In this section we conducted experiments to determine how the learned 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.

As we expected, the improvements due to the learned augmentation policy is larger when the model is trained on smaller datasets, which can be seen in Fig. 3 and in Table 5. We show that for models trained on 5,000 training samples, the learned augmentation policy can improve mAP by more than 70% relative to the baseline. As the training set size is increased, the effect of the learned augmentation policy is decreased, although the improvements are still significant. It is interesting to note that models trained with learned 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 learned augmentation policy seems to be better than increasing the dataset size by 50%, as seen in Table. 5. For small objects, training with the learned 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.

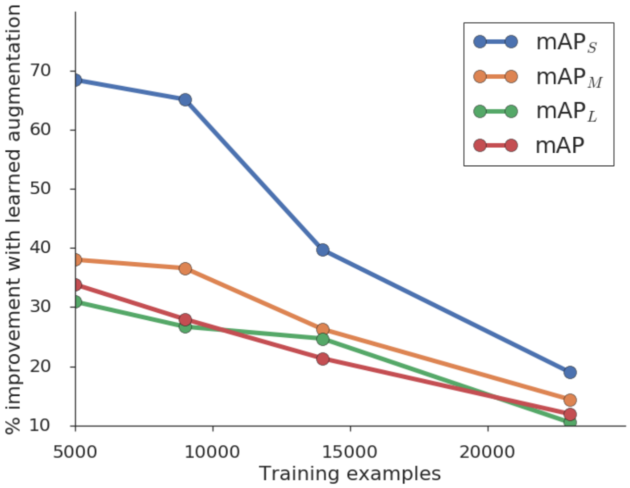


Figure 3: Percentage improvement in mAP for objects of different sizes due to the learned augmentation policy.

Another interesting behavior of models trained with the learned augmentation policy is that they do relatively better on the harder task of AP75 (average precision IoU=0.75). In Fig. 4, we plot the percentage improvement in mAP, AP50, and AP75 for models trained with the learned augmentation policy (relative to baseline augmentation). The relative improvement of AP75 is larger than that of AP50 for all training set sizes. The learned data augmentation is particularly beneficial at AP75 indicating that the augmentation policy helps with more precisely aligning the bounding box

[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 [49]. The anchor scale was also increased from 4 to 5 to compensate for the larger image size.

Architecture	Change	# Scales	mAP	mAP$_\mathrm{S}$	mAP$_\mathrm{M}$	mAP$_\mathrm{L}$
MegDet [32]		multiple	50.5	-	-	-
AmoebaNet + NAS-FPN	baseline [14]	1	47.0	30.6	50.9	61.3
	+ learned augmentation	1	48.6	32.0	53.4	62.7
	+ ↑ anchors, ↑ image size	1	**50.7**	**34.2**	**55.5**	**64.5**

Table 3: **Exceeding state-of-the-art detection with learned 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 [32]. Our current results only require a single inference computation at single spatial scale. Backbone model is AmoebaNet-D [38] and the NAS-FPN detection system [14]. For the **50.7** result, in addition to using the learned data augmentation policy, we increase the image size from 1280 to 1536 and the number of detection anchors from 3x3 to 9x9.

	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

Table 4: **Learned augmentation policy transfer to other object detection tasks.** Mean average precision (%) at IoU threshold 0.5 on a Faster R-CNN detector [39] with a ResNet-101 backbone trained and evaluated on PASCAL VOC 2007 [11]. Note that the augmentation policy was learned from the policy search on the COCO dataset.

training	Baseline				Our results			
set size	mAP$_\mathrm{S}$	mAP$_\mathrm{M}$	mAP$_\mathrm{L}$	mAP	mAP$_\mathrm{S}$	mAP$_\mathrm{M}$	mAP$_\mathrm{L}$	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

Table 5: **Learned 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$_\mathrm{S}$, mAP$_\mathrm{M}$ and mAP$_\mathrm{L}$ 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 5000 COCO images was used in all of the experiments. The models in the first row were trained on the same 5000 images that the policies were searched on.

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.

4.6. Learned data augmentation improves model regularization

In this section, we study the regularization effect of the learned data augmentation. We first notice that the final training loss of a detection models is lower when trained on a larger training set (see black curve in Fig. 5). When we apply the learned data augmentation, the training loss is increased significantly 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 learned augmentation policy have a smaller L_2 norm than models trained with baseline augmentation (see Fig. 6).

5. Discussion

In this work, we investigate the application of a learned data augmentation policy on object detection performance. We find that a learned data augmentation policy is effective across all data sizes considered, with a larger improvement when the training set is small. We also observe that the improvement due to a learned data augmentation policy is larger on harder tasks of detecting smaller objects and detecting with more precision.

We also find that other successful regularization techniques are not beneficial when applied in tandem with a learned data augmentation policy. We carried out several experiments with Input Mixup [52], Manifold Mixup [46] and Dropblock [13]. For all methods we found that they either did not help nor hurt model performance. This is an interesting result as the proposed method independently outperforms these regularization methods, yet apparently these regularization methods are not needed when applying a learned data augmentation policy.

Future work will include the application of this method

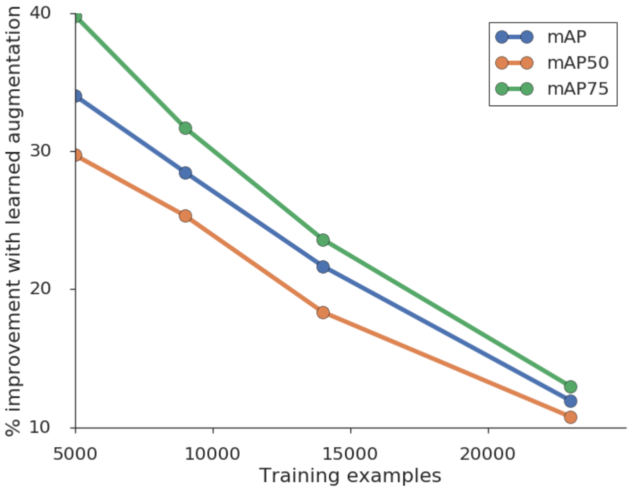


Figure 4: Percentage improvement due to the learned augmentation policy on mAP, AP50, and AP75, relative to models trained with baseline augmentation.

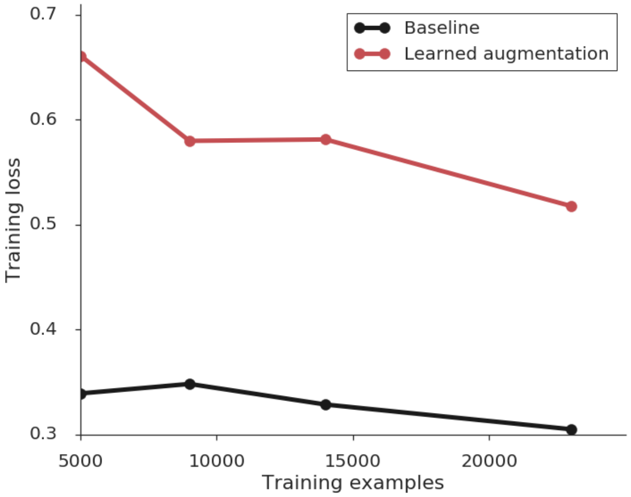


Figure 5: Training loss vs. number of training examples for baseline model (black) and with the learned augmentation policy (red).

to other perceptual domains. For example, a natural extension of a learned augmentation policy would be to semantic [28] and instance segmentation [34, 7]. Likewise, point cloud featurizations [35, 36] are another domain that has a rich set of possibilities for geometric data augmentation operations, and can benefit from an approach similar to the one taken here. Human annotations required for acquiring training set examples for such tasks are costly. Based on our findings, learned augmentation policies are transferable and are more effective for models trained on limited training data. Thus, investing in libraries for learning data augmentation policies may be an efficient alternative to acquiring additional human annotated data.

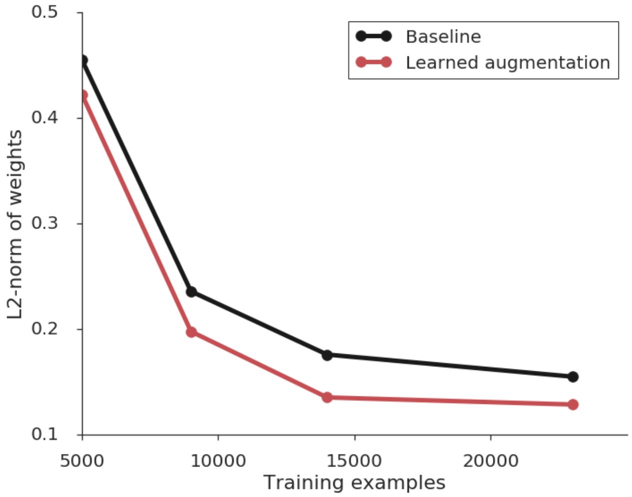


Figure 6: L_2 norm of the weights of the baseline (black) and our (red) models at the end of training. Note that the L_2 norm of the weights decrease with increasing training set size. The learned augmentation policy further decreases the norm of the weights.

Acknowledgments

We thank Ruoming Pang and the rest of the Brain team for their help.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA, 2016. USENIX Association. 4

[2] A. Antoniou, A. Storkey, and H. Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017. 1

[3] H. S. Baird. Document image defect models. In *Structured Document Image Analysis*, pages 546–556. Springer, 1992. 1

[4] D. Ciregan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 3642–3649. IEEE, 2012. 2

[5] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 1, 2, 4

[6] E. D. Cubuk, B. Zoph, S. S. Schoenholz, and Q. V. Le. Intriguing properties of adversarial examples. *arXiv preprint arXiv:1711.02846*, 2017. 4

[7] J. Dai, K. He, and J. Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3150–3158, 2016. 8

[8] T. DeVries and G. W. Taylor. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538*, 2017. 1, 2

[9] T. DeVries and G. W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 2, 13

[10] D. Dwibedi, I. Misra, and M. Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1301–1310, 2017. 2

[11] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 6, 7

[12] N. Ford, J. Gilmer, N. Carlini, and D. Cubuk. Adversarial examples are a natural consequence of test error in noise. *arXiv preprint arXiv:1901.10513*, 2019. 2

[13] G. Ghiasi, T.-Y. Lin, and Q. V. Le. DropBlock: A regularization method for convolutional networks. In *Advances in Neural Information Processing Systems*, pages 10750–10760, 2018. 2, 5, 7

[14] G. Ghiasi, T.-Y. Lin, R. Pang, and Q. V. Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 5, 7

[15] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron, 2018. 2

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 4

[17] D. Ho, E. Liang, I. Stoica, P. Abbeel, and X. Chen. Population based augmentation: Efficient learning of augmentation policy schedules. *arXiv preprint arXiv:1905.05393*, 2019. 2, 4

[18] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 1

[19] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017. 6

[20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 1–12. IEEE, 2017. 4, 5

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 1, 2

[22] J. Lemley, S. Bazrafkan, and P. Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017. 1, 2

[23] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim. Fast autoaugment. *arXiv preprint arXiv:1905.00397*, 2019. 2, 4

[24] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 1, 4, 5

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 1, 5

[26] C. Liu, B. Zoph, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. *arXiv preprint arXiv:1712.00559*, 2017. 4

[27] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2

[28] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 8

[29] R. G. Lopes, D. Yin, B. Poole, J. Gilmer, and E. D. Cubuk. Improving robustness without sacrificing accuracy with patch gaussian augmentation. *arXiv preprint arXiv:1906.02611*, 2019. 2

[30] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 4

[31] S. Mun, S. Park, D. K. Han, and H. Ko. Generative adversarial network based acoustic scene training set augmentation and selection using svm hyper-plane. In *Detection and Classification of Acoustic Scenes and Events Workshop*, 2017. 1

[32] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 6, 7

[33] L. Perez and J. Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017. 1

[34] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *European Conference on Computer Vision*, pages 75–91. Springer, 2016. 8

[35] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 652–660, 2017. 8

[36] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 8

[37] A. J. Ratner, H. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré. Learning to compose domain-specific transformations for data augmentation. In *Advances in Neural Information Processing Systems*, pages 3239–3249, 2017. 1, 2

[38] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019. 4, 5, 7

[39] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 6, 7

[40] I. Sato, H. Nishimura, and K. Yokoi. Apac: Augmented pattern classification with neural networks. *arXiv preprint arXiv:1505.03229*, 2015. 2

[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 4

[42] P. Y. Simard, D. Steinkraus, J. C. Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *Proceedings of International Conference on Document Analysis and Recognition*, 2003. 1, 2

[43] L. Sixt, B. Wild, and T. Landgraf. Rendergan: Generating realistic labeled data. *arXiv preprint arXiv:1611.01331*, 2016. 1

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 1

[45] T. Tran, T. Pham, G. Carneiro, L. Palmer, and I. Reid. A bayesian data augmentation approach for learning deep models. In *Advances in Neural Information Processing Systems*, pages 2794–2803, 2017. 1, 2

[46] V. Verma, A. Lamb, C. Beckham, A. Courville, I. Mitliagkis, and Y. Bengio. Manifold mixup: Encouraging meaningful on-manifold interpolation as a regularizer. *arXiv preprint arXiv:1806.05236*, 2018. 7

[47] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013. 2

[48] X. Wang, A. Shrivastava, and A. Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2606–2615, 2017. 2

[49] T. Yang, X. Zhang, Z. Li, W. Zhang, and J. Sun. Metaanchor: Learning to detect objects with customized anchors. In *Advances in Neural Information Processing Systems*, pages 318–328, 2018. 6

[50] D. Yin, R. G. Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer. A fourier perspective on model robustness in computer vision. *arXiv preprint arXiv:1906.08988*, 2019. 2

[51] S. Zagoruyko and N. Komodakis. Wide residual networks. In *British Machine Vision Conference*, 2016. 2

[52] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017. 7

[53] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017. 2, 13

[54] X. Zhu, Y. Liu, Z. Qin, and J. Li. Data augmentation in emotion classification using generative adversarial networks. *arXiv preprint arXiv:1711.00648*, 2017. 1

[55] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. 4

[56] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image

recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1, 4

A. Appendix

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 [9, 53]	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 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 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		

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.