Greedy AutoAugment

Alireza Naghizadeh, Mohammadsajad Abavisani, and Dimitris N. Metaxas, *Fellow, IEEE*

***Abstract*—A major problem in data augmentation is the number of possibilities in the search space of operations. The search space includes mixtures of all of the possible data augmentation techniques, the magnitude of these operations, and the probability of applying data augmentation for each image. In this paper, we propose Greedy AutoAugment as a highly efficient searching algorithm to find the best augmentation policies. We combine the searching process with a simple procedure to increase the size of training data. Our experiments show that the proposed method can be used as a reliable addition to the ANN infrastructures for increasing the accuracy of classification results.**

***Index Terms*—Gridcell, Neural Network, Navigation, Exploration, Robotic, ANN, Computational Model,Path-Planning.**

I. Introduction

Data augmentation is an important technique which can help to improve various computer vision tasks. For instance, a common practice in medical applications is class-specific data augmentation. In this case, gathering sufficient labeled data to train a high capacity deep model is impractical [1], [2]. This is the problem of long-tail distribution, which is prevalent in natural images as well[3], [4]. Such problems can be addressed by data augmentation [5]. Another example is the usage of data augmentation in unsupervised learning [6], [7], [8], [9]. A third example is to help generators and discriminators for training generative adversarial networks [10], [11]. In this paper, we focus on the problem of image classification with data augmentation. The main goal is to increase the accuracy of image classification by applying the right augmentation techniques on training data.

The data augmentation in image classification is directly related to image transformations. In the classification process, it is desirable to take into account a variety of target conditions from a primary condition. In other words, we want the perception of an object to be invariant to the properties that can vary in different environments such as scale, brightness, rotation, and viewing angle. These varying properties are called image transformations. Considering important image transformations and applying them in the learning process is a critical problem in Artificial Neural Networks (ANNs). For instance, it is desirable that a network, after learning an object from its original form, recognizes the same object with a change of location or added rotation. Currently, there are two ways to deal with this problem. First, by designing network architectures that can inherently be invariant to important image transformations. Second, with data augmentations.

The most basic network which considers the transformations of the input data is the Convolutional Neural Network (CNN). The CNN architecture, with the concept of convolutional layers, tries to be translation invariant [12], [13]. This network was very successful in its approach and has been used as a base for the development of more advanced architectures [14], [15], [16]. Another example of this approach is CapsuleNet, which tries to find the relevant pose information automatically [17], [18], [19]. While the design of CapsuleNet improved the results of basic datasets [20], unfortunately, it could not improve the accuracy for more complicated datasets such as ImageNet [21].

The second method for considering different transformations of the input data is to use data augmentation. In this method, the objective is to achieve invariance by applying different image transformations such as geometry transformation, kernel filtering, color transformation, image mixing, random erasing[22], etc. The main advantage of this method is simplicity and supporting all forms of ANN architectures. Additionally, there is a possibility to use transformation techniques in which current ANN architectures do not support.

One of the most important factors for data augmentation techniques is the constraint on increasing the size of the training data. In this regard, only a subset of the possible techniques can be used for data augmentation. Therefore, a search mechanism is needed to find the best possible techniques. The most common method to find the best data augmentation techniques is to find them manually[13], [23], [24] which needs prior knowledge and expertise. Recently, the AutoAugment [25] is proposed to automate the process of finding the best augmentations. In this method, finding the augmentation policy is reduced to a discrete search problem over various augmentation techniques, each having hyperparameters of the probability of applying the operation and the magnitude to which the operation is applied. AutoAugment emphasizes on applying the augmentation techniques without increasing the size of the training data. Because of the computational requirements for searching, it also relies on transferability of the augmentation techniques.

In this paper, we answer the question of how to effectively search for data augmentation techniques and apply them on training data. For this purpose, we propose Greedy AutoAugment. In this process, we develop a greedy-based search algorithm, which is computationally much more efficient than the methodology that is used in AutoAugment. Since our search method is very efficient, it is possible to perform the search for each dataset and network separately. Using the results from the greedy search algorithm, we propose a simple augmentation selection process which gives more priority to the augmentation policies that have higher accuracies on child networks. For training, our priority queue is designed in a way that it uses the original training data at least once. In this way, the original data remains intact, and the augmentations are used only to improve the results without affecting the original information. Our experimental results show that the proposed method provides higher accuracies when it is combined with

Corresponding author is A. Naghizadeh (ar.naghizadeh@cs.rutgers.edu).

current infrastructures.

II. Data Augmentation

Data augmentation techniques are standard transformations, which can be applied to image data. These augmentation techniques are standard image transformation operations that are defined in [26], [22], [27]. To use data augmentation, we randomly select these transformations and apply them on the image data as a pre-processing step. Applying data augmentation techniques in practice does not necessarily mean an increase in the size of the data. In fact, most of the standard networks use data augmentations as a pre-processing step without increasing the size of the training data.

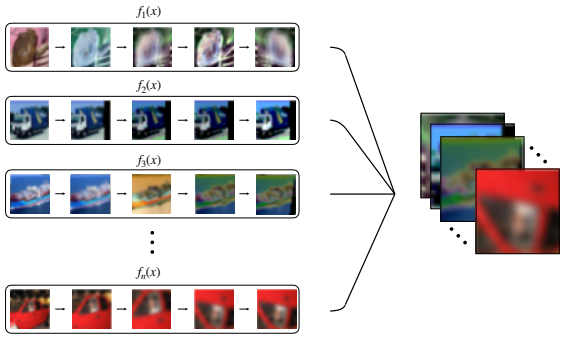


Fig. 1: Random data augmentation applied to samples from CIFAR-10.

In Figure 1, the effects of applying random data augmentation techniques to the images from CIFAR-10 dataset is shown. Each of the images at the left undergoes a series of random data augmentation techniques and the final image is fed to the convolutional neural network. As we can see, in some instances the change is not noticeable. However, in most cases, as we combine more techniques, the change becomes more drastic. The main concern here is to find the best combinations that can help the network to achieve better accuracy. The first approach is to take a trial-error approach and find the best combinations manually. The second approach is AutoAugment, which automates the process of finding the best techniques. In this method, a series of combinations of augmentation policies are found which are applied to the datasets without increasing the size of the datasets.

To search for the best combinations, the AutoAugment method utilizes NasNet [29] as a controller to direct the search forward. For this purpose, a one-layer LSTM is used, which contains 100 hidden units and 30 softmax predictions. This architecture employs a policy gradient method called Proximal Policy Optimization algorithm (PPO) which uses the accuracy obtained from a child network to train the LSTM network. The child network could be a small subset of a particular dataset. The accuracy updates from child network are used to update the LSTM network. In the end, the trained LSTM network helps for the selection of the best policies. Each policy has three elements 1- a data augmentation technique, 2- the probability of applying the operation, and 3- the magnitude of the operation. In the next section, we talk about these three elements in more details and provide an alternative approach to search for the best policies and applying them on the training data.

III. Greedy AutoAugment

To perform data augmentation on image data, we use policies. Each policy has three essential elements, 1- the augmentation technique, 2- the magnitude of the operation, and 3- the probability. Therefore, a search mechanism for finding the best augmentation techniques is a search space that should consider all of the possible combinations of these three elements. The number of augmentation techniques that we use in this paper is 20 (see Table I). The magnitude is the degree in which an operation is applied. For instance, in the rotate augmentation, the magnitude specifies how much we should rotate an image. The third element specifies the probability of applying the augmentation on the image.

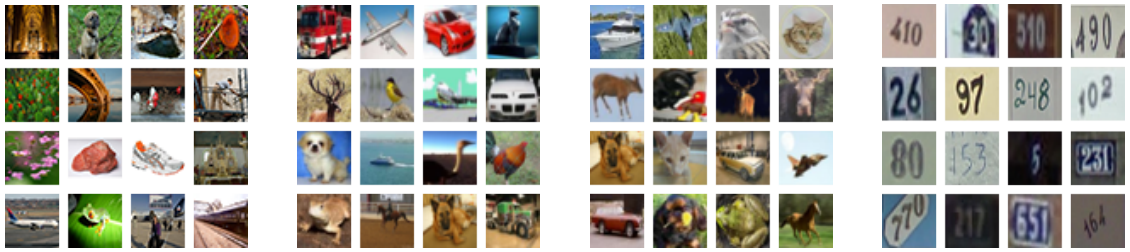
Before searching for the best policy, we have to first, discretize the spaces of probabilities and the magnitudes. The discretization of probabilities is with 11 values with uniform space, and the discretization of magnitudes is with 10 values with uniform space. The discretization schemes are in compliance with [25]. With this setting, the search space is simply $(20 \times 10 \times 11)$. This search space is defined to consider all of the possible combinations for elements of one policy (first search layer). If we expand the search space to find all of the possible combinations for two policies, the search space increases in size to $(20 \times 10 \times 11)^2$ (two search layers). Continuously, we can expand the search space for more layers infinitely. In general, we define $(20 \times 10 \times 11)^l$, where l is the number of layers for search space. In [25], the search is done for $l = 2$.

Our first fundamental assumption which distinguishes the searching process from [25] is that we do not search for the probability of applying an augmentation technique. Instead, we propose a static process which gives higher probabilities to the augmentation techniques that had higher accuracy with their child networks. Therefore, the search space reduces to $(20 \times 10)^l$. Based on our observations, data augmentations in child networks can increase the accuracy only to a certain degree. For instance, if we search for all policies with different degrees of magnitudes, it does not mean that it would affect the achieved top accuracies considerably. We use this observation and for the main part of our search process, we only search for policies with one magnitude. Different ranges of magnitudes come into play only after the best policies are found with a static magnitude. In this way, the search space almost reduces to $(20)^l$ where we use a greedy search algorithm and expand the search layers only when it is required.

To perform the greedy search, we use Algorithm 1. At first, observe that in line 4, we use fixed values of 1 and 6 for the values of probabilities and magnitudes. The value 1 is chosen for the probability because we want all of the images

	Technique	Description		Technique	Description
1.	FlipLR	Filliping the image along the vertical axis.	11.	Contrast	Changing the contrast of the image.
2.	FlipUD	Filliping the image along the horizontal axis.	12.	Brightness	Adjusting the brightness of the image.
3.	AutoContrast	Increasing the contrast of the image.	13.	Sharpness	Adjusting the sharpness of the image.
4.	Equalize	Equalizing the histogram of the image.	14.	ShearX	Sheering the image in horizontal axis.
5.	Invert	Inverting the color of the pixels in the image.	15.	ShearY	Sheering the image in vertical axis.
6.	Rotate	Rotating the image by certain degrees.	16.	TranslateX	Translating the image in horizontal axis.
7.	Posterize	Redicing the number of Bits for each pixel.	17.	TranslateY	Translating the image in vertical axis.
8.	CropBilinear	Croping with bilinear interpolation strategy.	18.	Cutout[28], [22]	Changing a random square patch of the image to gray pixels.
9.	Solarize	Inverting the color of all the pixels above a certain threshold.	19.	Blur	Blurring the image.
10.	Color	Changing the color balance of the image.	20.	Smooth	Smoothing the image(Low-pass filtering).

TABLE I: Augmentation techniques with their descriptions which are used in GAutoAugment.



(a) (b) (c) (d)

Fig. 2: Samples from real datasets used in our experiments: (a) Tiny ImageNet (b) CIFAR-10 (c) CIFAR-100 (d) SVHN.

to receive the policy. This helps us understand the overall effect that each policy can have on the final results. The value 6 is chosen as a random number for the magnitude. In lines 5,10, we specify that, if the current level provides us with higher accuracy compared to the previous layer, continue the search in the next layer. Otherwise, the search should terminate. In lines 6-9, one by one, we apply all of the augmentation techniques, and with the help of child networks, we get their accuracies. In line 3, the aforementioned searching process reiterates with a specific number of times (in this case 5 times). For iterations where $i > 1$, the search starts with a combination that has next-best accuracy. Similar to the first iteration ($i = 1$), searching continues through the layers until better combinations of policies cannot be found.

For the next step (lines 14-21), we still assume fixed probability value, but consider all possible magnitude values on the 5 combinations that we found in the previous step. Note that in line 12, all of the combinations of policies are stored in variable $\mathbf{P}_i$. For each member of $\mathbf{P}_i$, we go through all of the individual policies and make child networks for all of the possible magnitudes. As we go through each layer, a magnitude with the highest accuracy is chosen (see lines 14-19). Similar to the previous part of the algorithm, the newly found policies from this process are also stored in separate $\mathbf{P}_i$s. In the end, we store the 5 best $\mathbf{P}_i$s which show highest accuracy results in set $\mathbf{P}$ and return the results as suitable policies (lines 22-24).

In Algorithm 1, the probability is always set to one, which helps us reduce the search space significantly. Instead of searching for different combinations of the probability, we introduce a manual process which gives a higher probability selection to the policies with better results. We know that $\mathbf{P}$, is the set of all policies returned from Algorithm 1. The $\mathbf{P}$ is a descending ordered set where the leftmost member is a policy with the best result, and the rightmost member is a policy with the worst result. If $\mathbf{P}$ has k elements, we define vector $\vec{v} = [v_1, \dots v_k]$ which represents the probability of choosing each element of $\mathbf{P}$. To fill the values of $\vec{v}$, we use the probabilities by Pareto Distribution [30] which assigns the highest probability to v_1 and lowest probability to v_k, as follows,

$$\rightarrow v_i = \begin{cases} (\frac{1}{i})^{\alpha} & i > 1 \\ 1 & i \leq 1 \end{cases} \quad (1)$$

in which α is a positive parameter. When $i = 1$ the probability is one, and for $i > 1$ the probability is less than one but not zero.

To choose the best policy, we start from the rightmost element of $\vec{v}$ and go to the leftmost element of $\vec{v}$. Each of these elements has a chance to be selected based on their respective v_i value. In this way, based on parameter α, the rightmost element has the least probability to be taken, and the leftmost element is selected with the probability of 1. Based on this scheme, the training data can be expanded with the new augmented data as much as required.

IV. Results

In this section, we compare our method with current solutions. For this purpose, first, we show the accuracy results of our method compared to the other methods. Next, we provide a computational analysis of the overall augmentation process. In these experiments, we use five different prominent ANN architectures. The GoogLeNet[14], ResNet[15] and ResNeXt[31] are used as state-of-the-art networks which need more resources for training. The MobileNet[32], and

Algorithm 1 Greedy AutoAugment Algorithm.

Input: A set of 20 operations each with a magnitude and a probability.
Output: Suitable policies $\mathbf{P}$.
for i = 1, 2, ..., 5: **do**
    set all magnitudes to 6 and all probabilities to 1;
    while a better augmentation exists: **do**
        for all operations k=1,2,... ,20: **do**
            combine k with existing augmentations;
            train child network to get accuracy of applying the augmentation;
        end for
        check if the best combination improves the accuracy;
    end while
    store policies in $\mathbf{P}_i$;
end for
for i = 1, 2, ..., 5: **do**
    for combinations of policies in $\mathbf{P}_i$: **do**
        for all magnitudes j=1,2,... ,11: **do**
            train child network to get accuracy of applying the augmentation with magnitude j;
        end for
    end for
    store policies in $\mathbf{P}_i$
end for
sort policies in descending order with accuracy values
store $\mathbf{P}_1$, $\mathbf{P}_2$, ...,$\mathbf{P}_5$ in $\mathbf{P}$.
return $\mathbf{P}$.

ShuffleNetG2[33] are used as prominent lightweight networks. To implement these networks, we forked the implementations from [34]. The default settings of the network implementations are not changed. The networks accept 32×32 images and provide output based on the number of classes.

In the experiments, we also use four real datasets, 1- Tiny ImageNet[35] includes 120000 natural images in 200 classes with each class having a training set of 500 images a test set of 50 images along with 50 validation images. 2,3- CIFAR-10 and CIFAR-100 datasets[36], both containing 60000 images of size 32×32 in 10 and 100 classes respectively. 4- SVHN[37] which contains over 600000 images of real-world images of digits $0-9$. These selected datasets are used for three main reasons. First, while they are complex datasets, they have a reasonable number of images and features which makes working with them with our available computational resources feasible. Second, they are well-known datasets with known and predictable results on a variety of ANN architectures. Third, they are compatible with the official experiments of AutoAugment paper [25], [38].

For training GAutoAugment networks, we used learning rates of 0.5. Because the size of the data is smaller in other scenarios, we used the learning rate of 0.1. When we wanted to find the best policies for GAutoAugment, because in this level the size of the data did not change, we used learning rate 0.1. The number of the epochs for all of the training scenarios was 200. To find the best policies, we need to create child networks. The child networks and training networks share the same infrastructures. The only difference is using the learning rate 0.1 for child networks and learning rate 0.5 for training networks. To obtain the accuracies from child networks, we divided training data into two parts. The training part and the test part. For Tiny Imagenet and SVHN, 5000 images are used for testing. For CIFAR-10 and CIFAR-100, 2500 images are used for testing. All of the images are selected randomly with i.i.d. distribution. The α value for Pareto Distribution was always 2.

A. Accuracy

In this section, we test the accuracy of our proposed method using four different datasets, 1- Tiny ImageNet, 2- CIFAR-10, 3- CIFAR-100, and 4- SVHN. The results are shown in Table II. In this table, the "Original" column stands for the images without any augmentation methods. The "Aug" column stands for the manual augmentation. For manual augmentation, the exact augmentation techniques which are released with AutoAugment code are used. These techniques include zero padding, cropping, random-flip, and cutout. To prevent probable bugs, the same source code is used for manual augmentation [38]. This helps to provide a fair environment for all methods. The only extra pre-processing step that we used is for resizing Tiny Imagenet from 64×64 to 32×32. This helped us to use the same infrastructure for all datasets without having adverse effects on the experiments. The values in the table are the average results from five trials.

The column "AutoAug" stands for the AutoAugment method. For CIFAR-10, CIFAR-100, and SVHN, we use the same policies that are found from AutoAugment method. For Tiny ImageNet, we use the policies that are found for ImageNet dataset. Because Tiny ImageNet is a subset of ImageNet, it can test the generalization of the AutoAugment method. The column "GAutoAug" stands for Greedy AutoAugment, which is the proposed method. In our method, we suggest a specific searching process for each scenario to find the best possible policies. This is possible because (as we will see in the next section) our search method is computationally much more efficient than AutoAugment method. We also need an increase in the size of the training data. In all of our experiments, we doubled the size of the original training data with the proposed method.

For Tiny ImageNet, as we can see, the policies from AutoAugment are not effective when they are applied to the original images. Four of the five networks had worse results where the worst result was for GoogLeNet with 7.07% less accuracy. The only increase of accuracy is for ResNeXt with 0.32% better accuracy. Comparatively, our method increased accuracy for all five networks with at most 10.97% better accuracy for ResNeXt. When AutoAugment was added on top of manually added augmentations, it made the results for four networks worse. The worst result was for MobileNet with 6.47% less accuracy, and the best result was for GoogLeNet with 1.1% better accuracy. Comparatively, when GAutoAugment was added on top of manually added augmentations, the results were better for all five networks with at most 13.69% better accuracy for MobileNet and at least

	Network	Original	Org + AutoAug	Org + GAutoAug	Org + Aug	Org + Aug + AutoAug	Org + Aug + GAutoAug	Org + Aug+ AutoAug + GAutoAug
Tiny ImageNet	ResNet[15]	28.64	27.94	30.34	31.66	25.19	35.44	35.71
	ResNeXt[31]	32.36	32.68	43.33	33.87	29.45	45.36	40.11
	GoogLeNet[14]	34.76	32.03	41.81	33.38	34.48	43.62	44.07
	MobileNet[32]	20.77	13.70	30.24	15.15	13.34	28.84	22.68
	ShuffleNetG2[33]	28.02	21.65	29.76	24.34	21.29	30.26	30.75
CIFAR-10	ResNet	75.89	80.58	80.92	77.83	76.48	78.38	82.85
	ResNeXt	81.08	81.10	82.26	76.11	78.53	83.26	81.45
	GoogLeNet	78.54	76.36	87.59	74.61	74.12	83.97	81.26
	MobileNet	74.84	73.52	81.16	65.04	67.54	77.38	74.75
	ShuffleNetG2	73.14	74.37	81.04	71.04	72.02	80.03	81.93
CIFAR-100	ResNet	47.34	42.92	55.21	39.67	42.90	54.82	45.37
	ResNeXt	49.54	46.75	58.07	42.67	51. 26	57.15	51.01
	GoogLeNet	40.40	47.53	55.24	38.53	43.04	59.40	50.13
	MobileNet	42.26	43.67	50.26	40.28	42.70	46.65	44.61
	ShuffleNetG2	51.33	50.30	56.18	43.93	49.01	55.95	52.05
SVHN	ResNet	92.95	92.58	92.51	90.69	88.67	92.42	87.99
	ResNeXt	92.29	92.04	93.80	91.11	90.03	90.53	92.77
	GoogLeNet	91.92	92.43	94.39	85.91	88.38	93.66	92.84
	MobileNet	87.93	87.80	91.53	81.64	84.32	86.63	87.15
	ShuffleNetG2	87.20	91.93	93.64	87.16	87.47	88.79	90.95

TABLE II: The result table for accuracy analysis. Abbreviations include: Org = Original, Aug = Manual Augmentation, AutoAug = AutoAugment, GAutoAug = Greedy AutoAugment.

3.78% better accuracy for ResNet. When we added our method on top of the AutoAugment, we increase the accuracy on five networks with at most 10.66% better accuracy for ResNeXt. Overall, the generalization from ImageNet to Tiny ImageNet was not very smooth, and results from AutoAugment were mostly worse.

For CIFAR-10, the transition is better for AutoAugment policies. From five networks, three networks had better results with at most 4.69% better accuracy and at least 2.18% worse accuracy than the original images. Our method increased the accuracy for all five networks, with at most 9.05% better accuracy for GoogLeNet and at least 1.18% better accuracy for ResNeXt. When AutoAugment was added on top of manually added augmentations, the results were not much different. There are three networks which had better results with at most 2.5% better accuracy and at least 1.35% worse accuracy than the original images with manual augmentations. When GAutoAugment was added on top of manually added augmentations, our method increased the accuracy for all five networks with at most 12.34% better accuracy in MobileNet. When we added our method on top of the AutoAugment policies, we increased the accuracy on all five networks with at most 9.91% better accuracy and at least 2.92% better accuracy than AutoAugment.

The results for CIFAR-100 show that when policies from AutoAugment were applied to original images, the accuracies could only improve for two networks. Comparatively, when the policies from the proposed method are applied to the original network, we could improve the results for five networks. The accuracy could be up to 14.84% and down to 4.85% better than the original images. When policies from AutoAugment are applied to the original images combined with manual augmentation policies, this time, the accuracy was improved for all five networks with up to 8.59% better accuracy. The proposed method could also improve the results for all five networks with up to 20.87% better accuracy. When we added our method to the AutoAugment itself, the accuracy increased for four networks with up to 7.09% better accuracy and down to 0.25% worse accuracy, which could be a random fluctuation on the results.

Overall, in our experiments for CIFAR-10 and CIFAR-100, while we could replicate the results from AutoAugment source code, the transition of the augmentation policies did not have impressive results on our own infrastructure. The main reason could be that different network architectures, or even different implementations of the same network architecture can reduce the generality of augmentation policies.

For SVHN, applying policies from AutoAugment only improves the accuracy of two networks. The proposed method could improve the accuracy of the four networks. When we added policies from AutoAugment to the manually augmented images, three networks improved accuracy with up to 2.68% better results. On the other hand, when we added the proposed method, the accuracy improved for four networks with up to 7.75% better accuracy. When we joined our method with policies from AutoAugment, again, we improved the accuracy for four networks with up to 4.46% improvement on the accuracy. As we can see, the results for SVHN show that the accuracy is less reliable, and random fluctuations could happen more than other datasets. However, these fluctuations are negligible, and at most, we have seen 0.68% worse result with the proposed method. Overall, in this dataset, the results show a similar trend compared to the previous datasets.

B. Computational Analysis

In this section, we analyze the computational requirement for our method compared to the original AutoAugment method. As described in [25], the AutoAugment needs $15,000$ samples

of child networks. Comparatively, in our method, the number of child networks were usually between 300 to 650. Also, AutoAugment uses 120 epochs to evaluate the accuracies of child networks. The number of epochs used for our child networks was only 5 epochs. Since the exact infrastructure for different datasets is not known for AutoAugment, and child networks are interchangeable, we consider the child networks to have the same efficiency.

	Network	Org + GAutoAug	Org + Aug + GAutoAug	Org + Aug AutoAug GAutoAug
Tiny ImageNet	ResNet[15]	731	733	730
	ResNeXt[31]	743	687	810
	GoogLeNet[14]	931	814	1087
	MobileNet[32]	905	794	1052
	ShuffleNetG2[33]	918	920	916
CIFAR-10	ResNet	527	794	835
	ResNeXt	568	512	607
	GoogLeNet	853	831	779
	MobileNet	677	779	779
	ShuffleNetG2	638	761	664
CIFAR-100	ResNet	628	920	596
	ResNeXt	578	543	559
	GoogLeNet	733	875	628
	MobileNet	607	512	746
	ShuffleNetG2	589	779	717
SVHN	ResNet	677	875	616
	ResNeXt	570	714	605
	GoogLeNet	965	994	831
	MobileNet	534	701	689
	ShuffleNetG2	618	814	568

TABLE III: The result table for performance analysis. Abbreviations include: Org = Original, Aug = Manual Augmentation, AutoAug = AutoAugment, GAutoAug = Greedy AutoAugment.

As a simple example for comparisons between the two methods, let us assume that our method needs 500 child networks for a specific scenario. To calculate the overall computation, we need to take all of the epochs into account, which is $500 \times 5 = 2500$. For the AutoAugment, we need to consider the computations of 15000 child networks with 120 epochs, which is $15000 \times 120 = 1800000$. If both networks use the same infrastructure for child networks, the comparison is $1800000 \div 2500 = 720$ times fewer computations for the proposed method. By considering this example, we provide Table III. In this table, we use the same networks and datasets that we used for accuracy analysis. We needed the policies for three types of environments, 1) finding policies for original images, 2) finding policies when combined with manual policies, 3) finding policies when combined with AutAugment method. Therefore, the table has three columns for these three environments.

The results show that for Tiny ImageNet at least the proposed method was 684 times better than AutoAugment. In the same dataset, the proposed method was at most 1048 times better than AutoAugment. For CIFAR-10, we were at least 512 times and at most 854 times computationally more efficient than AutoAugment. For CIFAR-100, we were at least 534 times and at most 994 times computationally more efficient than AutoAugment. At last, for SVHN, we were at least 512 times, and at most 920 times computationally more efficient than AutoAugment. On average for the four datasets, we were 851.4, 706.9, 718.0, and 667.3 times computationally more efficient than AutoAugment. The average value for all datasets is 735.93 with the least value of 512 and the highest value of 1087. In practice, the computational efficiency is in a way that it is practical to do a specific search for a scenario before doing the actual training. On the other hand, since we increase the training size, the training time increases according to the increase in the size of the training data.

V. Conclusion

In this paper, we proposed Greedy AutoAugment as a highly efficient method to find the best augmentation policies. Our experiments show that on average, we used 735.92 times fewer computations for finding best policies compared to the original AutoAugment. We combined the searching process with a simple procedure to increase the size of training data. For the experiments on the classification accuracy, we used four real datasets and five networks. Our results show that the proposed method could reliably improve the accuracy for classification results, which could provide up to 20.87% better accuracy compared to the base models.

References

[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*. Springer, 2015, pp. 234–241.

[2] C.-C. Kuo, C.-M. Chang, K.-T. Liu, W.-K. Lin, H.-Y. Chiang, C.-W. Chung, M.-R. Ho, P.-R. Sun, R.-L. Yang, and K.-T. Chen, "Automation of the kidney function prediction and classification through ultrasound-based kidney imaging using deep learning," *npj Digital Medicine*, vol. 2, no. 1, p. 29, 2019.

[3] X. Zhu, D. Anguelov, and D. Ramanan, "Capturing long-tail distributions of object subcategories," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 915–922.

[4] Z. Tang, Y. Zhang, Z. Li, and H. Lu, "Face clustering in videos with proportion prior," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[5] Y.-X. Wang, D. Ramanan, and M. Hebert, "Learning to model the tail," in *Advances in Neural Information Processing Systems*, 2017, pp. 7029–7039.

[6] A. Dosovitskiy, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with convolutional neural networks," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 766–774.

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

[8] Y. Bengio, G. Mesnil, Y. Dauphin, and S. Rifai, "Better mixing via deep representations," in *International conference on machine learning*, 2013, pp. 552–560.

[9] S. Ozair and Y. Bengio, "Deep directed generative autoencoders," *arXiv preprint arXiv:1410.0630*, 2014.

[10] X. Peng, Z. Tang, F. Yang, R. S. Feris, and D. Metaxas, "Jointly optimize data augmentation and network training: Adversarial data augmentation in human pose estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2226–2234.

[11] F. Luan, S. Paris, E. Shechtman, and K. Bala, "Deep photo style transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4990–4998.

[12] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*, 1990, pp. 396–404.

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[15] 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*, 2016, pp. 770–778.

[16] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[17] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming auto-encoders," in *International Conference on Artificial Neural Networks*. Springer, 2011, pp. 44–51.

[18] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing," 2018.

[19] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in neural information processing systems*, 2017, pp. 3856–3866.

[20] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

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

[23] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *arXiv preprint arXiv:1202.2745*, 2012.

[24] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European conference on computer vision*. Springer, 2016, pp. 646–661.

[25] 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.

[26] wiredfool, A. Clark, Hugo, A. Murray, A. Karpinsky, C. Gohlke, B. Crowell, D. Schmidt, A. Houghton, S. Johnson, S. Mani, J. Ware, D. Caro, S. Kossouho, E. W. Brown, A. Lee, M. Korobov, M. Górny, E. S. Santana, N. Pieuchot, O. Tonnhofer, M. Brown, B. Pierre, J. C. Abela, L. J. Solberg, F. Reyes, A. Buzanov, Y. Yu, eliempje, and F. Tolf, "Pillow: 3.1.0," Jan. 2016. [Online]. Available: https://doi.org/10.5281/zenodo.44297

[27] H. Inoue, "Data augmentation by pairing samples for images classification," *arXiv preprint arXiv:1801.02929*, 2018.

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

[29] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[30] B. C. Arnold, *Pareto Distribution*. Wiley Online Library, 2015.

[31] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500.

[32] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[34] "Deep model infrastructures used for training gautoaugment," https://github.com/kuangliu/pytorch-cifar, accessed: 2019-07-26.

[35] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," 2015.

[36] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," Citeseer, Tech. Rep., 2009.

[37] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[38] "Training deep models on cifar-10 and cifar-100 using autoaugment," https://github.com/tensorflow/models/tree/master/research/autoaugment, accessed: 2019-07-26.