Data Augmentation with Policy Optimization

Jason Lin
Georgia Institute of Technology
jlin401@gatech.edu

Yousef Emam
Georgia Institute of Technology
emamy@gatech.edu

Abstract

Recent work by Google Brain has shown that Reinforcement Learning could be used to automate the process of data augmentation for image classification. Specifically, following successes in applying Reinforcement Learning (RL) to design optimal neural network architectures with Neural Architecture Search (NAS) [1], the authors have continued their work to meta-learn autonomous data augmentation policies using RL using a RNN as controller [2]. The novel task is formulated as a discrete search problem, where the goal is to find groups of transformations that yield the highest improvement in the performance of an image classifier from an extensive pre-defined list of possible transformations (crop, flip, rotate, shear, etc.). The reward is obtained as the loss function yielded when the image classifier, trained using the augmented data, is tested on a validation set with a child model. In this project, we initially aimed at extending their work to the problem of image segmentation. However, due to the extensive scope of the project's preassumed knowledge and computational power, we instead focused on implementing and improving their closed-source algorithm in the context of image classification.

1 Introduction

In the context of data-driven computer vision, data augmentation involves increasing the size of available data by applying various content-invariant geometric transformations to generate artificial samples for training, thereby improving the learner's generalization capacity. For example, when dealing with deep learning-based image classifiers, rotating each image does not fundamentally alter the desired content within nor its final label, and could therefore be applied to increase the size and diversity of the dataset. The notion of invariance in the transformation depends on the task at hand, where the goal is to transform input images such that it preserves semantic meaning so the model will be i.e. scale or rotation invariant. The concept of data augmentation is particularly helpful when data sample for the task at hand is small, i.e. in medical imaging domains. In addition to bootstrapping few-shot learning, data augmentation is known to be effective in reducing overfitting in training. Indeed, certain objects could be solely identifiable through subset of features, i.e. color, geometric shapes, edge orientation, or a mutually exclusive combination of them. Therefore, data augmentation has become an indispensable step in training state-of-the-art image classifiers where human operators use their intuition to decide which transformations are deemed invariant.

2 Related Work

Hyperparameter search: Neural architecture design has been an active area of research [] that saw recent successes with a class of Bayesian Optimization (BO) methods, i.e. Gaussian Processes [3]. The task of automating neural network training by replacing manual grid search with meta-learning methods is an ambitious yet highly rewarding goal in Machine Learning. Historically, BO was mostly used to find optimal hyperparameters and model configurations that are non-differentiable, discontinuous, and time-consuming to evaluate. It uses an underlying Gaussian Process to model a given black box objective function and is well suited to

optimizing the performance of classification and regression models. Recent work such as AutoKeras [4] have shown that in the absence of RL under Google's closed source work on AutoML, BO can in fact be a more efficient approach to NAS at the cost of less effective improvement.

Policy Gradients: In the recent era of deep reinforcement learning, policy gradients has been instrumental to breakthroughs in control and robotics applications. Vanilla PG methods used to suffer from poor sample complexity and high variance, where their sensitivity to step size makes it susceptible to catastrophic effects of noise. TRPO [5] introduces a trust region for continuous control tasks, while PPO [6] clips the gradient update so that the policy deviation is safe within a typical stochastic gradient descent. ACER [7] improves sample efficiency with robust off-policy corrections and experience replay. Both PPO and TRPO belong to the class of Minorize-Maximization algorithms that exploit the convexity of a function in order to reach optima. Because PPO strikes a balance between simplicity and performance that is close to ACER, we use it for updating policies generated by our controller.

3 AutoAugment

Automatic data augmentation has very recently become an active area of research. Indeed, most efforts towards the improvement of Neural Network (NN) training have been focused on optimizing the neural nets' architecture rather than the data augmentation process itself X. Moreover, the first published work addressing auto-augmentation [2] for image classification has achieved state-of-the-art results, indicating that this is a fruitful area of research. The authors' best augmentation policies were also found to be transferable when applied to different datasets than the one used for training. The authors' model, which is inspired from [8], is composed of a reinforcement learning agent (the controller) as well as a child Convolutional Neural Network (CNN) utilized for training. The high-level procedure is for the controller to output a set of transformations to augment the data, which is then used to train the child CNN. Once trained, the child CNN is then tested against a validation set and its validation loss is used to train the controller. This procedure is used to iteratively improve the controller so that it can output better transforms. The original aim of this project was to extend this methodology to image segmentation. However, due to many difficulties mainly dealing with lack of computational power, we instead focused on the re-creation and improvement of this methodology for image classification.

4 The Problem Setup

The problem of finding the best policy can be re-framed as one of searching a large discrete space. Currently, there are 20 widely used operations for data augmentation including Shear-X(or Y), Translate-X(or Y), Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout and Sample-Pairing. Each of those operations can vary in magnitude, for example, the magnitude of the Rotate operations specifies the degree of rotation. Moreover, we also associate a probability to each of those operations indicating the likelihood that they are applied to an image during the augmentation process. Therefore, each transformation can be thought of as consisting of three variables: operation (type), magnitude and probability. As in [2], we chose to define a sub-policy as two transformations applied in sequence, and a policy as a set of five sub-policies. During augmentation with a given policy, the augmenter uniformly chooses a sub-policy and applies to an image, and the process is repeated for each image. By discretizing the magnitude and probability of each possible transformation, we obtain the desired discrete search problem. Also because of the stochastic nature of applying the operations, the process is non-differentiable and we cannot use naive back-propagation to learn the policies without knowledge of the best combination of operations as a priori groundtruth.

5 The Controller

5.1 Background: Long Short Term Memory (LSTM)

An LSTM [9] is a RNN architecture that is used for time series or sequence data. Similar to any other RNN architecture, an LSTM has a feedback connection that allows it keep track of previous inputs

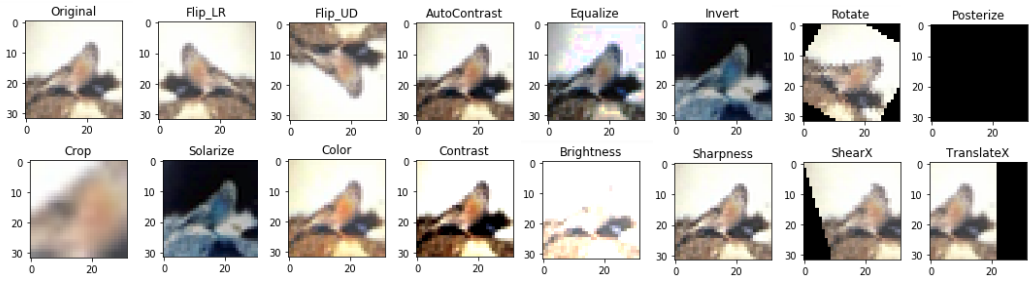

Figure 1: A table showing the set of image transformations considered as part of our subpolicies.

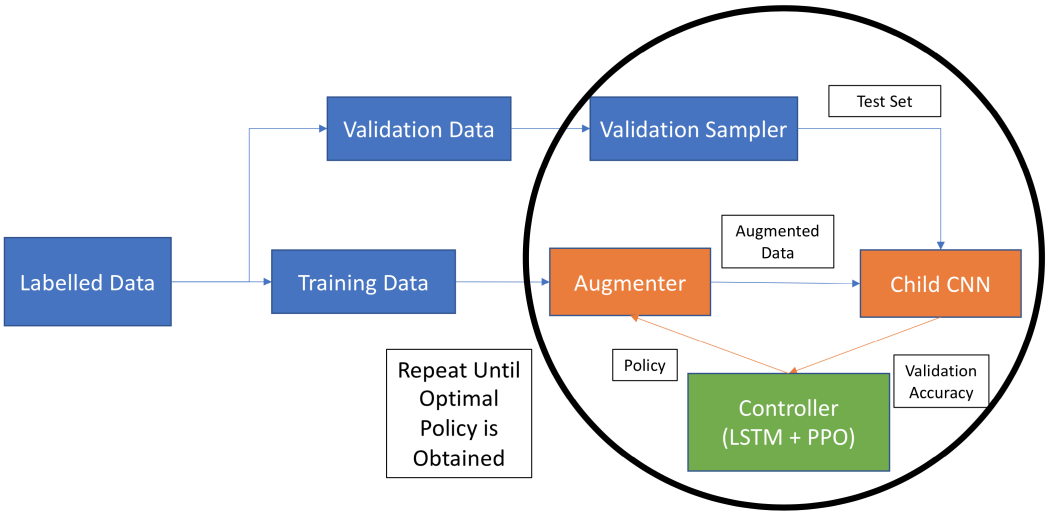

Figure 2: Learning Model's Architecture. Process denoted within the circle is repeated iteratively to improve the policy outputted by the controller.

allowing it to capture the necessary time dependency. The main motivation behind the development of LSTM is combating the vanishing gradient problem common to other architectures. This issue arises in deep NNs, as gradients get smaller and smaller as they travel back through the deep-NN's layers. To do so, a basic LSTM cell consists of four gates as shown in Figure 3, two of which allow it to "forget" parts of the stored state and ignore parts of the input through sigmoid functions.

5.2 Background: Proximal Policy Optimization (PPO)

PPO [6] is an algorithm used to compute the gradient in Policy Gradient Methods. For vanilla policy gradients such as REINFORCE, the objective is computed as follows:

$$\mathcal{L}^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t], \tag{1}$$

where $\hat{A}_t$ is the estimated advantage obtained using a critic (which has a similar architecture to the net used for policy prediction). On the other hand, PPO introduces additional constraints to stabilize the learning process by bounding the gradient's size at each step. The best of those objectives is the Clipped Surrogate Objective

$$\begin{aligned} &\mathcal{L}^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \\ &\text{where } r_t(\theta) = \frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}(a_t|s_t). \end{aligned} \tag{2}$$

The intuition behind the clipped objective is that it limits the step size if the advantage is positive and a large step was previously taken towards increasing the probability of that action ($r_t(\theta) > 1$) so as

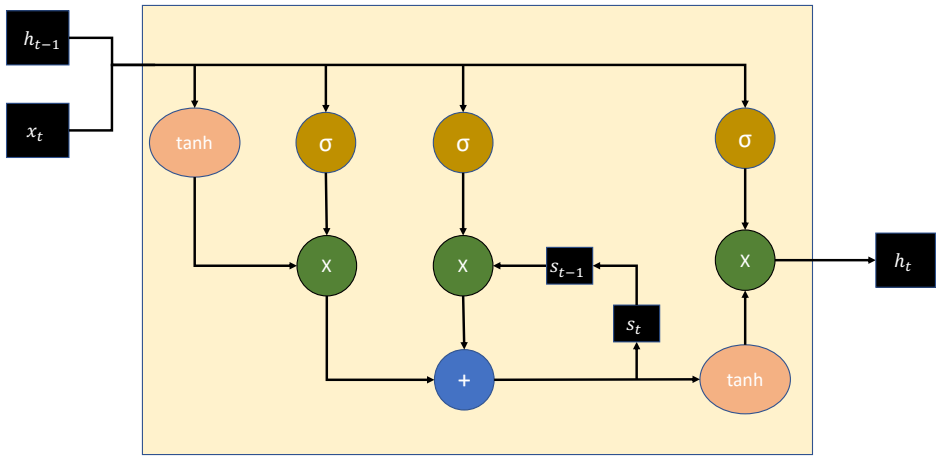

Figure 3: A basic LSTM cell. The blocks x and h indicate the input and cell's output respectively. The sigmoid and hyperbolic tangent functions are denoted through the σ and $tanh$ blocks respectively.

to stabilize learning. The clipping also occurs if the advantage is negative and the action previously underwent a large step towards decreasing the probability of that action. Other objective functions involving KL divergence are suggested in [6] but are deemed to perform at most as good as the clipped objective.

5.3 The Controller's Architecture

The controller (i.e. the RL agent selecting the policies) consists of two components: a prediction algorithm and a training algorithm. The first is a deep neural net consisting of a single LSTM layer containing 100 units. The output of the LSTM at each step is then fed into a full-connected layer outputting a vector of variable size depending on which decision is being taken (i.e. operation, magnitude or probability). Finally, the output of the fully-connected layer is then passed through a softmax to obtain a decision. This process is repeated 30 times since each policy requires 5 sub-policies, each requiring 2 transforms, each requiring 3 decisions (operation, magnitude and probability). The controller's second component is PPO which is used to perform the gradient updates using the child CNN's validation accuracy. The main incentive behind the use of PPO is that its stable and fast (when compared to TRPO for example). The validation accuracy from the child model is fed into PPO as the advantage function which is in turn used to compute the gradient for the prediction net.

5.4 The Child Model

The child model used is a simple 4-layer CNN for the CIFAR10 dataset shown in Figure 4. Each convolution layer is followed by RELU activation and a max-pooling layer to reduce the feature size. The max-pooling layer simply slides a max filter through the fed feature map to reduce its size and is known to work quite well [10]. The last layer in the architecture is a fully-connected layer that outputs a vector of softmax probabilities the same length as the number of labels.

5.5 Experiments and Results

We use CIFAR-10 to evaluate our data augmentation policies. CIFAR-10 has 10 output classes and 60,000 images in the full dataset. For the interest of time and cross-algorithm comparison, we randomly sample 300 images as training data and 50 as test data. We train a LSTM using PPO to obtain its loss value, as defined in 5.2. For the PPO loss objective we use $\epsilon = 0.2$ as

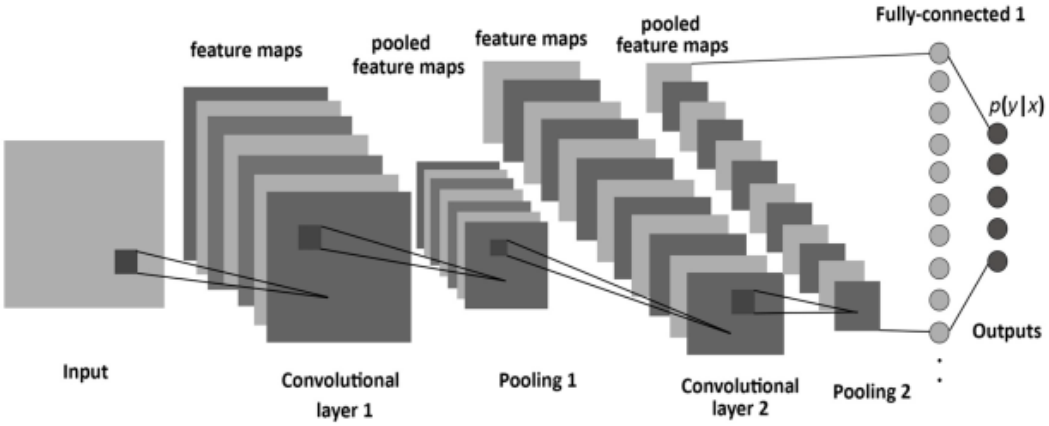

Figure 4: A simple 4-layer (conv-32, conv-64, FC-256, FC-10) CNN on the CIFAR10 dataset.

it was found to be best from the original authors [6]. Our LSTM has 100 hidden units which are fed into an array of 30 fully connected predictor heads that are each then passed through a softmax for final probabilities. The 30 predictors output alternating number of units in series of $\{|operations|, |probabilityIntervals|, |magnitudeIntervals|\}$. We then concatenated the softmax outputs into a $10 \times (20 + 11 + 10) = 410$ unit binary mask to denote the operation type, probability, and magnitude of the 5 sub-policies (2-operations each) within a final output policy. The output policy is weighted over the probability it is chosen by the old model as part of the surrogate objective, and finally multiplied with the reward from child network $\hat{A}_t$ as loss to the current step of training to be optimized by minibatch Adam [11]. The child reward is validation accuracy from evaluating a child network trained for 100 epochs using each generated policy.

Interestingly, while conducting initial literature review [12, 13, 14, 3], we came across a few sources that use other search methods that also perform rather well in similar tasks. In fact, the DeepAugment repository on github includes a similar architecture to the AutoAugment one except that the authors use a Bayesian Optimizer instead of Reinforcement Learning to reduce the computational cost. Therefore, we additionally implemented a Bayesian Optimizer from `scikit-opt` and a Random-Walk optimizer for the sake of comparison. The cross-algorithm test results are tabulated in 2. Note that we also tried hand-tuning our own policy, which performed the worst out of all. This is an interesting result since this hand-tuned policy mostly applied geometric operations. The mediocre performance of the hand-tuned policy implies that for many researches who are unfamiliar with the process of data augmentation, Auto-Augment can be greatly beneficial. Bayesian optimization also yielded improvement over the baseline and was computationally faster than our policy-gradient based controller, although our RL controller had far superior performance in terms of accuracy.

Opt. Method	Rand Avg.	Rand Best	Hand-tuned	Bayesian	RL-controller
% Improvement	-4.286	6.000	-12.000	6.000	12.000

Table 1: Comparison of different Optimization Methods. Absolute percent improvement is measured with respect to the baseline performance of the child (i.e. without augmentation). The Bayesian Optimizer was trained for a 1000 iterations vs. RL controller trained for 20 iterations with 100 training epochs per child model.

6 Conclusion

In this project, we have successfully re-implemented the work done in [2] and demonstrated that the Reinforcement Learning controller, although computationally costly, does yield policies capable of greatly increasing the classification accuracy. This increase in performance is strongly reflected in our experiment since we only used 300 images. Additionally, we implemented various other optimization

Operation Type 1	Probability 1	Magnitude 1	Operation Type 2	Probability 2	Magnitude 2
Rotate	1.000	9.00	Invert	0.900	8.00
Flip_LR	0.700	2.00	ShearX	0.900	4.00
Solarize	1.000	9.00	Flip_UD	1.000	0.00
Posterize	0.800	7.00	ShearY	0.000	1.00
Blur	0.000	0.00	TranslateY	0.600	3.00

Table 2: The best policy returned by the RL agent.

methods to contrast our results. Although random walk and the Bayesian optimizers do improve performance and are computationally less costly, it is clear from our results that the RL approach is still more fruitful.

Moreover, the initial objective of this project was to apply these suggested auto-augmentation methods to the task of image segmentation rather than classification, which as mentioned earlier, we were unable to do due to the learning curve of prerequisite knowledge and demand for computational power. Therefore, we intend to keep working towards this goal even after the end of the semester. Intuitively, the same architecture should work for segmentation except for the child model which may need to undergo slight modifications. Because our code is implemented in TensorFlow, we can also readily scale compute by training larger CNN models with TPU on the cloud. The segmentation mean Average Precisions (mAP) can then be used to train the controller instead of the validation accuracy.

References

[1] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

[2] Ekin Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *CoRR*, abs/1805.09501, 2018.

[3] Jon Crall. https://github.com/fmfn/BayesianOptimization.

[4] Haifeng Jin, Qingquan Song, and Xia Hu. Efficient neural architecture search with network morphism, 2018. cite arxiv:1806.10282.

[5] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1889–1897. JMLR.org, 2015.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[7] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016.

[8] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[12] Baris Ozmen. DeepAugment:https://github.com/barisozmen/deepaugment.

[13] Kyle Matoba. https://github.com/kylematoba/deeplearning-project.

[14] Itai Caspi, Gal Leibovich, Gal Novik, and Shadi Endrawis. Reinforcement learning coach, December 2017.