RL-Mixup: Learning mix-up policies with Reinforcement Learning

Long M. Luu [1] Zeyi Huang [2] Haohan Wang [2]

Abstract

Mix-up has been proven efficient in improving the neural network's generalization ability, and multiple extensions of the original mix-up have been introduced in recent years. However, these techniques mainly focus on the data instead of the neural network's performance. In this paper, we propose a new method to automatically learn the mix-up strategy with the gradient information and the reinforcement learning module, called *RL-Mixup*. The mix-up strategy is controlled by a neural network trained with reinforcement learning to maximize the accuracy of the classifier on the validation set. Initial results show an on-par performance with other methods' baselines while running in a significantly shorter amount of time.

1. Introduction

Convolutional neural networks (CNN) are the backbone of many computer vision tasks with remarkable performance. To improve the neural network's generalization capability *Mix-up*, a simple data augmentation technique has been proposed. The original Mix-up, called *Input Mix-up* (Zhang et al., 2018) is a task-independent technique that linearly interpolates two samples. *Manifold Mix-up* (Verma et al., 2019) extends the mix-up to the perturbations of embeddings. *CutMix* (Yun et al., 2019) extends to use a spatial copy and paste-based strategy from other samples to augment the current sample. Modern mix-up methods, such as *PuzzleMix* (Kim et al., 2020) and *Co-Mixup* (Kim et al., 2021) use saliency information and local statistics to ensure the mixed data to have rich supervisory signals.

However, these approaches mainly focus on enriching the data itself instead of directly improving the classifier's performance. What if these mix-up style augmentations are not what the classifier needs? Other works, such as (Cubuk et al., 2019), (Zhang et al., 2019), (Zhou et al., 2020), (Pham & Le, 2021) are the work experimenting with the approach that performs augmentations directly based on the classifier's performance. In this paper, we combine the gradient information (saliency) and the Input Mix-up strategy on patch level with the Reinforcement Learning (RL) module to automatically learn the best mix-up policy. The policy is controlled by a neural network trained with RL that maximizes the long-term accuracy of the model.

There are two major differences of our method compared to (Cubuk et al., 2019) and (Pham & Le, 2021): (1) (Cubuk et al., 2019) finds the optimal combination of augmentation operations, (Pham & Le, 2021) finds the optimal dropout strategy, while we experiment with mix-up strategy. (2) Both (Cubuk et al., 2019) and (Pham & Le, 2021) have a major drawback: they train a classifier from scratch when the new policy is given (the policy is unchanged during training), and the process repeats until the optimal policy is found, which is extremely computationally expensive. Our work learns the mix-up policy within a run only (in 300 epochs), and the policy continuously changes to adapt to the new weights of the classifier and the input images.

We verify the capability of the method by training classifiers on the CIFAR-100 dataset (Krizhevsky & Hinton, 2009) for 300 epochs (when the classifier reaches convergence). Initial results show an on-par performance with other mix-up baselines while running in a significantly shorter amount of time.

2. Preliminary

Reinforcement Learning: In RL, a decision-making agent interacts with the environment in a sequence of actions, in order to maximize its expected long-term reward (Sutton & Barto, 2018). In our work, the interaction can be formalized using the framework of Partially Observable Markov Decision Process (POMDPs). At each time step t, the agent is in a state $s_t \in \mathcal{S}$, observes the observation $o_t \in \mathcal{O}$, takes action $a_t \in \mathcal{A}$ accordingly, and receives the reward r_t that is $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, continues to the next state s_{t+1}, then the process repeats. We use the discount factor γ and the future rewards are defined as $\sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$, so that the far-future rewards are less important. The goal of

[1]School of Computer Science and Engineering, International University - VNUHCM, Ho Chi Minh city, Vietnam [2]School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA. Correspondence to: Long M. Luu <minhlong9413@gmail.com>.

Accepted at ICML2021 ML4Data Workshop, Copyright 2021 by the author(s).

the agent is to find the optimal policy π^* that maximizes the future rewards $\mathbb{E}_\pi[R_0|s_0]$ where s_0 is sampled from a distribution of initial states $p(s_0)$ and the expectation is taken over the dynamics specified above. The *Q-function* or *action-value* function of a given policy π is defined as $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_t|s_t, a_t]$, while the *V-function* or *state-value* function is defined as $V^\pi(s_t) = \mathbb{E}_\pi[R_t|s_t]$. The value $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$ is called the *advantage* and tells whether the action a_t is better or worse than an average action the policy π takes in the state s_t.

In practice, the policy and value function are represented as neural networks. Specifically, we maintain one neural network for the policy π, and one neural network to approximate the value function of the current policy $V \approx V^\pi$.

Pipeline: Let $\langle \mathbf{x}, \mathbf{y} \rangle$ denotes a data sample where $\mathbf{x} \in \mathbb{R}^{C \times W \times H}$ (C, W, H are the image's channel, width and height respectively) and the label $\mathbf{y} \in \{0, 1\}^N$ with N classes. Let $f(\cdot; \theta_c)$ denotes the convolutional neural network classifier whose parameters are denoted as θ_c. Let $g(\mathbf{x}) \in \mathbb{R}^{p \times p}$ denotes the saliency of the input $\mathbf{x}$ which is computed by taking ℓ_2 norm of the gradient values across input channels (Simonyan et al., 2014), then is down-sampled to a specific size ($p \times p$).

The reinforcement learning agent consists of the policy and value networks, but for simplicity we denote the agent is controlled by the parameter ψ. Let $o_t^k \in \mathcal{O}$, $a_t^k \in \mathcal{A}$ and r_t^k denote the observation, the action and the reward of ψ at time step t of episode k respectively. Let τ_k denotes the trajectory of observations, actions and rewards of the k-th episode: $\tau_k = \{o_1^k, a_1^k, r_1^k, o_2^k, a_2^k, r_2^k, ...\}$.

Aliases: In this paper, we use "time step" as one training batch, an "episode" as one training epoch, so that it fits to the RL module.

3. Method

3.1. Gradient-based patch level mix-up

We describe the method to use *Input Mix-up* (Zhang et al., 2018) on patch level using saliency map. For two pairs of data samples $\langle \mathbf{x}_1, \mathbf{y}_1 \rangle$ and $\langle \mathbf{x}_2, \mathbf{y}_2 \rangle$, we compute the saliency of two images as $g(\mathbf{x}_1)$ and $g(\mathbf{x}_2)$ respectively ($g(\mathbf{x}) \in \mathbb{R}^{p \times p}$). The saliency map is down-sampled to a specific size (16×16) using Average Pooling and normalized to sum up to 1. Then we compute the q^{th} percentile value of the saliency (where q corresponds to the action a chosen from $\mathcal{A}$ by the agent). We construct two masks $\mathbf{m}_1$ and $\mathbf{m}_2$ as follows. For the i-th element in $g(\mathbf{x})$:

$$\mathbf{m}(i) = \begin{cases} 1, & \text{if } g(\mathbf{x}) \geq q \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

We then up-sample $\mathbf{m}$ to match the size of the inputs. Let $\mathbf{m}_{\text{inter}}$ is a mask constructed as follows. For the i-th element in $\mathbf{m}_1$ and $\mathbf{m}_2$:

$$\mathbf{m}_{\text{inter}}(i) = \begin{cases} 1, & \text{if } m_1(i) = m_2(i) \\ 0, & \text{otherwise,} \end{cases} \tag{2}$$

i.e. we mix the patches where both of them belong to and neither belongs to the top percentile values.

We then sample the weights $\lambda_1, \lambda_2 \sim Dirichlet(\alpha, \alpha)$ where $\alpha = 2$ [1], then the mixed image can be described as follows:

$$\begin{aligned} \mathbf{x}' = {} & \mathbf{m}_{\text{inter}} \odot (\lambda_1 \mathbf{x}_1 + \lambda_2 \mathbf{x}_2) \\ & + \neg \mathbf{m}_{\text{inter}} \odot (\mathbf{m}_1 \odot \mathbf{x}_1 + \mathbf{m}_2 \odot \mathbf{x}_2) \end{aligned} \tag{3}$$

where $\odot$ denotes the element-wise product, and $\neg$ denotes the logical NOT operator.

We use $c(\mathbf{m})$ to denote the number of patches of $\mathbf{m}$ that are active, *i.e.*, $c(\mathbf{m}) = |i|\mathbf{m}(i) = 1|$, where $|\cdot|$ denotes the cardinality of a set. We define $\mathbf{y}'$ as

$$\begin{aligned} \mathbf{y}' = {} & \frac{c(\mathbf{m}_{\text{inter}})}{W \times H}(\lambda \mathbf{y}_1 + \lambda \mathbf{y}_2) \\ & + \frac{c(\neg \mathbf{m}_{\text{inter}} \odot \mathbf{m}_1)}{W \times H}\mathbf{y}_1 + \frac{c(\neg \mathbf{m}_{\text{inter}} \odot \mathbf{m}_2)}{W \times H}\mathbf{y}_2 \end{aligned} \tag{4}$$

The mixed sample $\langle \mathbf{x}', \mathbf{y}' \rangle$ will be used to train $f(\cdot; \theta_c)$. In practice, $\langle \mathbf{x}, \mathbf{y} \rangle$ can be a batch of training samples, and the other batch can be obtained by randomly permuting the current batch. $f(\cdot; \theta)$ is trained to estimate the soft target labels by minimizing the cross-entropy loss. Sample visualizations can be found in Appendix B.

For each batch of training data, the value q is decided by the reinforcement learning agent, which we describe in the next section.

3.2. Training the reinforcement learning agent

To train ψ, we define one episode of the environment as one epoch of $f(\cdot; \theta_c)$, and one time step is one training batch. Let T denotes the total time steps in one episode. The episode terminates after $f(\cdot; \theta_c)$ finishes training the last batch.

The agent receives an observation o_t at each time step. We use a simple convolutional neural network (of which we will call the Features Extractor from now on) to encode it to an embedding vector. This embedding vector is used as the inputs of both the policy and value network of ψ, *i.e.* the Features Extractor is shared between both networks. The agent returns $a_t \in \mathbb{R}^B$ where where $B = 100$ is the batch size of $f(\cdot; \theta_c)$. a_t indicates the q value for each $\mathbf{x}$.

[1]This parameter is not tuned throughout the experiments

We give the agent r every $t_{\text{reward}} \leq T$ steps, and linearly scale r. Specifically, at the reward step:

$$r_t^k = \frac{k}{K}(v - V) \tag{5}$$

Where v is the current validation accuracy, V is the previous validation accuracy, k is the current epoch and K is the total number of epochs. A subset from the validation set is uniformly sampled to calculate v. Empirical results show no difference between a subset and the full set.

τ_k of each episode will be used to train ψ. To reduce variance, we use τ collected in two episodes to train ψ. Let $o_t = g(\mathbf{x}_t)$, r as specified in equation 5, and the action space $\mathcal{A}$ is discrete with 10 values ranging from 0 to 99, uniformly spaced. ψ's objective is to maximize v when $k \to K$. Since π is changed continuously, we name our method *RL-Mixup*: the mix-up strategy is adapted to the classifier's performance and the input images.

We use the Proximal Policy Optimization algorithm with the PPO-Clip variant (Schulman et al., 2017) from the Stable-Baselines3 code base (Raffin et al., 2019). Unless specified otherwise, we keep the same hyperparameters. Implementation details can be found in Appendix A. The algorithm to train $f(\cdot;\theta_c)$ and ψ is described in Algorithm 1.

Algorithm 1 RL-Mixup

```
Input: train dataset $\langle \mathbf{X}, \mathbf{Y} \rangle$, validation dataset $\langle \mathbf{X_{test}}, \mathbf{Y_{test}} \rangle$, number of episodes $K$, number of steps $T$ in each episode, number of steps to give the reward $t_{\text{reward}}$, the CNN classifier $f(\cdot;\theta_c)$, the agent's parameter $\psi$, initial validation accuracy $V = 0$.
for every 2 epochs do
  Initialize an empty trajectory $\tau_k$
  Get the policy $\pi_k = \pi(\psi_{\text{policy}})$
  for each timestep $t$ ($t \leq T$) do
    Get the $t$-th batch from the training set: $\langle \mathbf{x_t}, \mathbf{y_t} \rangle$
    Compute $o_t^k = g(\mathbf{x_t})$
    Get the percentile value $q = a_t^k = \pi_k(o_t^k)$
    Mix-up the batch using equations 1, 2, 3 and 4
    Train $f(\cdot;\theta_c)$ with $\langle \mathbf{x_t}', \mathbf{y_t}' \rangle$
    if reaches $t_{\text{reward}}$ step then
      Get $v$ on a subset of $\langle \mathbf{X_{test}}, \mathbf{Y_{test}} \rangle$
      Return $r_t^k$ according to equation 5
      Set $V = v$
    else
      $r_t^k = 0$
    end if
    Append $o_t^k, a_t^k, r_t^k$ to $\tau_k$
  end for
  Train $\psi$ using PPO-Clip algorithm to optimize $\pi$
end for
```

4. Classification Experiments

4.1. Classification on CIFAR-100

We train PreActResNet18 (He et al., 2016) on CIFAR-100 for 300 epochs. We use stochastic gradient descent with Nesterov momentum and weight decay using the OneCycleLR learning rate scheduler (Smith & Topin, 2017). Intuitively, we keep the learning rate small in the beginning, so that $f(\cdot;\theta_c)$ does not learn too much from random π. The Top-1 accuracy result is reported in Table 1. Our method currently does not outperform *Co-Mixup* (Kim et al., 2021), but is on par with *PuzzleMix* (Kim et al., 2020).

Table 1. Top-1 accuracy on CIFAR-100 on PreActResNet18 for 300 epochs. Higher is better. We run our method for five different seeds, while other mix-up methods' results are from (Kim et al., 2021).

METHOD	TOP-1 ACCURACY
BASELINE (NO MIX-UP)	76.41
INPUT MIX-UP	77.57
MANIFOLD MIX-UP	78.36
CUTMIX	78.71
PUZZLEMIX	79.38
RL-MIXUP (OURS)	79.80±0.18
CO-MIXUP	**80.13**

4.2. Runtime

We compare the runtime of *PuzzleMix* and *Co-Mixup* with our method using PreActResNet18 on CIFAR-100, where all methods require saliency computation. *PuzzleMix* and *Co-Mixup*'s runtime are directly reported from the official implementation checkpoints [2] [3]. Table 2 shows that our method runs in a significantly shorter amount of time (2x compared to *Co-Mixup*, 3x compared to *PuzzleMix*), on a less powerful GPU.

Table 2. Runtime of our method compared to PuzzleMix and Co-Mixup on CIFAR-100 using PreActResNet18 in 300 epochs, when the classifier reaches convergence (except PuzzleMix reaches convergence in 1200 epochs). Lower is better. We calculate the mean of the relative time (in second) of our method in five runs on NVIDIA Tesla P100, while PuzzleMix used NVIDIA TITAN Xp and Co-Mixup used NVIDIA RTX 2080 Ti.

METHOD	RUNTIME(HOURS)
PUZZLEMIX (300 EPOCHS)	7
PUZZLEMIX (1200 EPOCHS)	28
CO-MIXUP	15.3
RL-MIXUP(OURS)	**7.5**

[2] Co-Mixup released checkpoint
[3] Puzzle-Mix released checkpoint

5. Limitations and Discussions

To find π^* within 300 epochs proves to be extremely challenging. In this section, we investigate some challenges that the agent must address, limitations in the current state of the experiments, and some findings from empirical results.

5.1. High-dimensional observation space

In our setup, $o_t = g(\mathbf{x}_t) \in \mathbb{R}^{B \times p \times p}$ and $p = 16$. This implies that ψ observes 25600 total values each time step. For comparison, OpenAI Five (Berner et al., 2019) observes only ≈ 16000 total values, and in Deep Q-Learning (Mnih et al., 2013), the agent observes $(4 \times 84 \times 84) = 28224$ total values.

We also set ψ's batch size to 100. Therefore, during training, ψ is trained with inputs of shape $(B \times B \times p \times p)$. For a small and simple network, the observation space's dimension might be too high.

5.2. Analyzing ψ's runtime

Training ψ accounts for a very small proportion of the total training time (approximately 5% overall). We downsample o to a specific size $(p \times p)$ so that ψ's training and computing time is the same regardless of $\mathbf{x}$'s size. The main bottleneck is $f(\cdot; \theta_c)$ runtime.

5.3. Limited agent's control over mix-up strategy

We use $q = a \in \mathcal{A}$ to calculate the percentile value of the $g(\mathbf{x})$, of which ψ still has very limited control of the mix-up strategy. ψ currently answers "how many patches to mix" on the image level, but not yet "how to mix", unlike other mix-up methods.

5.4. Small and limited experiences

We define $\tau_k = \{o_1^k, a_1^k, r_1^k, o_2^k, a_2^k, r_2^k, ...\}$, but we collect τ from two episodes (τ_k and τ_{k+1}) to train ψ, which consists of 1000 tuples of three of such, and train ψ using batch size of 100.

Since the tuple at each time step depends on θ_c, it is not possible to obtain more τ without using and training another $f(\cdot; \theta_c)$. Moreover, PPO is an on-policy algorithm, *i.e.* old experiences are discarded when the policy changes. This implies that the number of τ available is limited, and might not be enough for ψ to learn something actually useful, which we discuss in the next section.

5.5. Comparing with randomness

We compare the RL method with the random method using PreActResNet18, *i.e.* the action is sampled uniformly at each step. Empirical results show that the random method is on par with the RL method's performance. We think that the challenges and limitations described in 5.1, 5.2, 5.4 and 5.3 are the contributing factors (but not limited to).

However, when we decrease ψ's learning rate to a very small number $(1e-8)$, we notice the drop in performance (achieved 78.82% Top-1 Accuracy), indicating that the RL method does help $f(\cdot; \theta_c)$ to learn better, but currently is not better than randomness.

5.6. Reward function design

Reward function design plays a major role in the process. In contrast to most RL problems, we actually pay attention to the long-term reward rather than the near-future one. From empirical experiments, we observe the following:

- If ψ only receives r at the end of each episode, the policy collapses to a single action.
- If ψ receives r at every step, the computational resources become costly.
- We did not have good results with $r = v$. We have two hypothesis: (1) using current validation accuracy means that $R = \{r_0, r_1, ..., r_T\}$ are all positive, and (2) the increase of r does not translate to a better π: it might because $f(\cdot; \theta_c)$ is trained one more epoch.
- Using the discount factor $\gamma = 1$ gives the best result compared to 0.99, which suits our intention.

Therefore, we decide to reward ψ r every 50 time steps and use the scaled factor (equation 5). Although it is better than giving r at the end of each episode, it is still sparse. Intuitively, the equation 5 helps ψ to know:

- The scaled factor indicates that initial r values are not significant (when k is small). When k is big, small changes in $(v - V)$ are magnified.
- If v decreases, ψ knows that it selects a worse π, and how much worse.

We sample 10% of the validation set uniformly to calculate r. Empirical results show no difference between the full set and subset.

6. Conclusion

In this paper, we propose *RL-Mixup*, a method to combine mix-up-based data augmentation strategy with the reinforcement learning module. Our experiments show an on-par performance with other mix-up baselines while running in a shorter amount of time. We also discuss the challenges and limitations of the current experiments. We hope that the paper will be used as an inspiration to perform data augmentations that directly help the neural networks to learn and generalize better by using performance feedback.

References

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 113–123. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00020.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

Kim, J., Choo, W., and Song, H. O. Puzzle mix: Exploiting saliency and local statistics for optimal mixup. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5275–5285. PMLR, 2020.

Kim, J., Choo, W., Jeong, H., and Song, H. O. Co-mixup: Saliency guided joint mixup with supermodular diversity. *CoRR*, abs/2102.03065, 2021.

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Pham, H. and Le, Q. V. Autodropout: Learning dropout patterns to regularize deep networks. *CoRR*, abs/2101.01761, 2021.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Simonyan, K., Vedaldi, A., and Zisserman, A. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.

Smith, L. N. and Topin, N. Super-convergence: Very fast training of residual networks using large learning rates. *CoRR*, abs/1708.07120, 2017.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–1139–III–1147. JMLR.org, 2013.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., and Bengio, Y. Manifold mixup: Better representations by interpolating hidden states. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6438–6447. PMLR, 2019.

Yun, S., Han, D., Chun, S., Oh, S. J., Yoo, Y., and Choe, J. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 6022–6031. IEEE, 2019. doi: 10.1109/ICCV.2019.00612.

Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. mixup: Beyond empirical risk minimization. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Zhang, X., Wang, Q., Zhang, J., and Zhong, Z. Adversarial autoaugment. *CoRR*, abs/1912.11188, 2019.

Zhou, F., Li, J., Xie, C., Chen, F., Hong, L., Sun, R., and Li, Z. Metaaugment: Sample-aware data augmentation policy learning. *CoRR*, abs/2012.12076, 2020.

A. Implementation details

We divide the training and validation sets to batches of size 100. For each batch of training samples, we calculate the saliency by using the gradient values of training loss with respect to the input data and measure ℓ_2 norm of the gradient values across input channels (Simonyan et al., 2014). We down-sample this saliency using Average Pooling and use it as the observation of the agent. The observation is passed through a Features Extractor to create a 512 dimensions vector as the input of the value and policy networks, of which is described in Table 3. This Features Extractor is shared between the policy and value networks.

Table 3. A simple Features Extractor CNN architecture. All Conv2D layers use kernel size 3, stride 1 and padding 0. The last layer encodes the Gloval Averge Pooling features into a 512 dimensions vector. GAP indicates Global Average Pooling.

STAGE	LAYER	CHANNELS
1	CONV2D	200
2	RELU	
3	CONV2D	300
3	RELU	
4	CONV2D	400
5	RELU	
6	GAP	
7	LINEAR	
8	TANH	

The action space $\mathcal{A}$ has 10 values, ranging from 0 to 99 uniformly. After getting q as a from ψ, the batch is mixed with itself using equations 1, 2, 3, 4, and then is used to train the $f(\cdot;\theta_c)$. We use stochastic gradient descent with Nesterov momentum 0.9 (Sutskever et al., 2013) and weight decay 0.0001 using OneCycleLR learning rate scheduler (Smith & Topin, 2017). The initial learning rate is 0.004, the max learning rate is 0.2 and the final learning rate is 0.002 using cosine annealing. The learning rate increases for 100 epochs. We give the agent reward as described in equation 5 every 50 steps. The trajectory is of size 1000 and the agent's batch size is 100. The surrogate loss of PPO is optimized for 20 epochs. Discount factor is set to 1.001 indicates that the future rewards are more important. We sample 10% of the validation set to calculate v. The agent's value network is $256 - 256$ with Tanh activation, and the policy network is $64 - 64$ with Tanh activation. If not specified, we keep the same hyperparameters as in Stable-Baselines3 (Raffin et al., 2019).

B. Sample Visualization

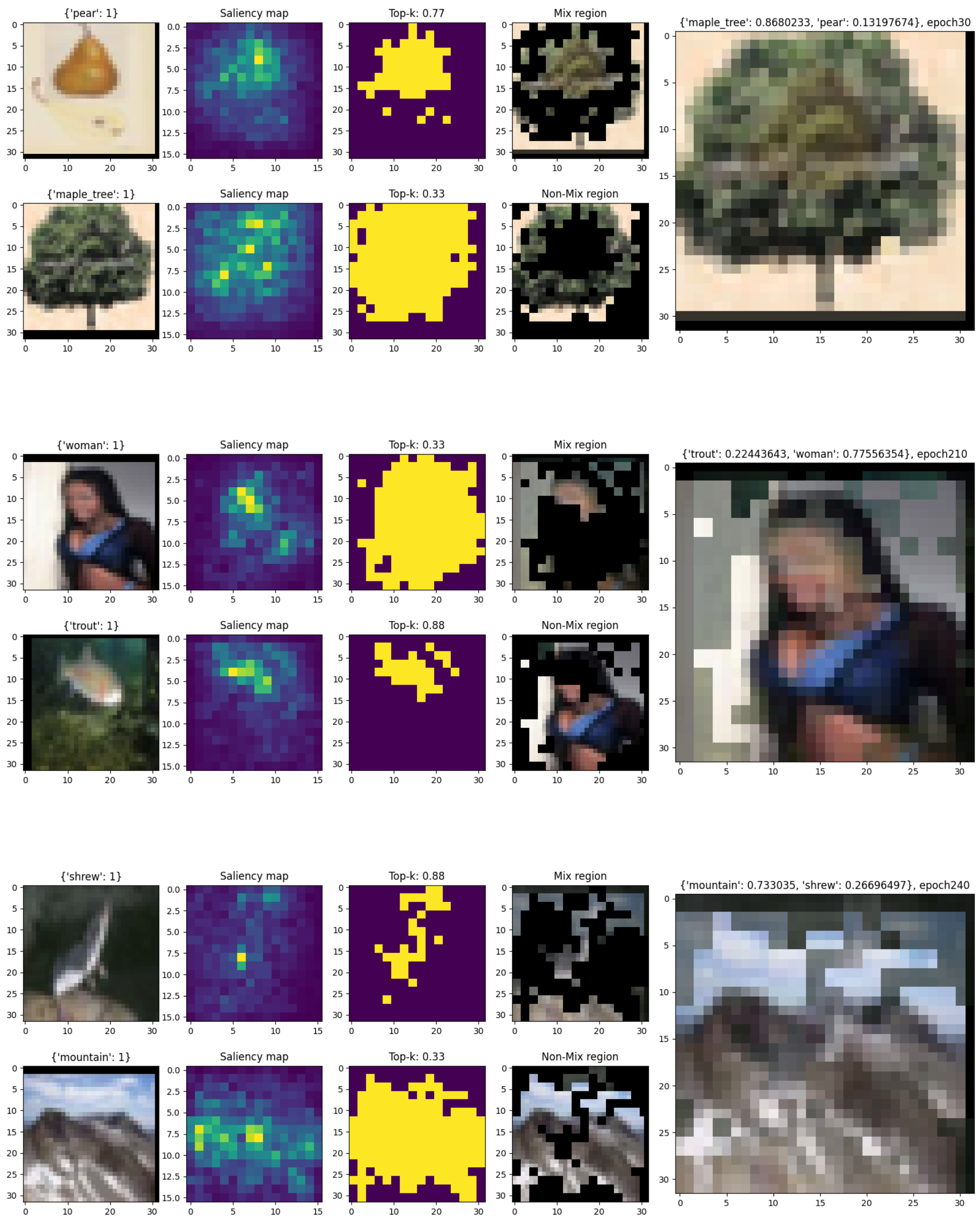

Figure 1. Sample visualization. First column: original image. Second column: saliency map. Third column: top-k most important patches picked by the agent. Fourth column: mix and non-mix regions. Fifth column: mixed image with its soft labels.