Investigating Manifold Mixup

Computer Vision WS19/20

Julian Kanzler Michael Nening Manuel Widmoser

Abstract

In this work, we investigate manifold mixup, a simple regularizer that encourages deep neural networks to predict less confidently on adversarial examples. We evaluate manifold mixup on a subset of CIFAR-10, where we only use 500 rather than 50,000 training samples. In our experiments, we first attempt to get the baseline high by considering different preprocessing steps, learning rate schedulers and hyperparameters. Then, manifold mixup with different α-values, which controls the strength of interpolation between feature-target pairs, is applied on combinations of eligible layers. We find that for our task, this regularization technique increases labelling performance in all evaluated combinations, with a resulting peak accuracy of 61.73%.

1 Introduction

Deep neural networks often give incorrect but still highly confident predictions when evaluated on data which only comes from a slightly different distribution than examples from the training data. This can already be achieved by marginal perturbations that are imperceptible to the human eye. Such inputs, also known as *adversarial examples*, may be a serious hazard when machine learning systems are used in security-sensitive applications.

To address this issue, the training data can be extended by similar but different examples. This method is called *data augmentation* and leads to better generalization [Simard et al., 1996]. However, data augmentation depends on the dataset, and thus human knowledge is needed.

Mixup is a simple data-agnostic data augmentation procedure that increases the robustness of neural networks when facing adversarial examples by extending the training distribution [Zhang et al., 2017]. Mixup provides convex combinations of example/label-pairs where the network additionally will be trained on. This regularizes the neural network to favor simple linear behavior in-between training examples.

Basically, mixup constructs virtual training examples $(\tilde{x}, \tilde{y})$ such that

$$\tilde{x} = \lambda x_i + (1 - \lambda)x_j \quad (1)$$

$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j \quad (2)$$

where x_i, x_j are raw input vectors, y_i, y_j are one-hot label encodings and $\lambda \sim \text{Beta}(\alpha, \alpha)$ is the mixing coefficient sampled from the Beta distribution dependent on α. $\alpha = 1.0$ is equivalent to sampling $\lambda \sim U(0, 1)$.

2 Manifold Mixup

Manifold mixup trains neural networks on linear combinations of hidden representations of training examples [Verma et al., 2019]. Hence, it performs mixup in an intermediate layer in which feature spaces are more aligned. Furthermore, a neural network trained with manifold mixup learns class-representations with fewer directions of variance and yields a smoother decision boundary.

Training a deep neural network using manifold mixup is done as follows: We select a layer k at random from the set of eligible layers $\mathcal{S}$ in the neural network. $\mathcal{S}$ may contain the first layers of the residual blocks, for instance. Then perform input mixup [Zhang et al., 2017] on two random minibatches (x_i, y_i) and (x_j, y_j) which are processed through the forward pass until reaching layer k. This gives us a mixed minibatch $(\tilde{x}, \tilde{y})$. With that we continue the forward pass from layer k until we ultimately get the output. Finally, we update the parameters of the neural network by computing the loss and gradients using the output.

2.1 Implementation

Manifold mixup can be implemented in a few lines of code without heavy computations that would drop the training performance of the deep neural network a lot.

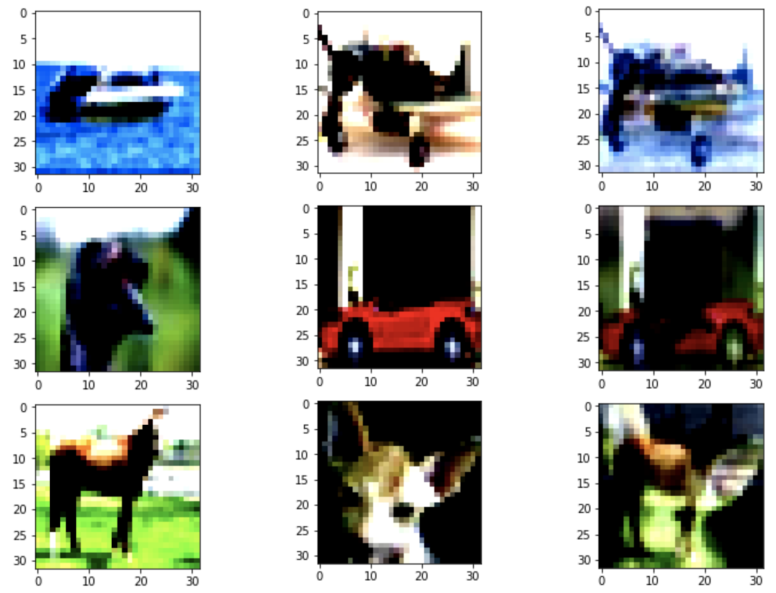
```
1   def mixup(x, y, alpha):
2       batch_size = x.size()[0]
3       lam = np.random.beta(alpha, alpha) if mixup_alpha > 0 else 1
4
5       if torch.cuda.is_available():
6           index = torch.randperm(batch_size).cuda()
7       else:
8           index = torch.randperm(batch_size)
9
10      # x[index, :] swaps the training samples
11      mixed_x = lam * x + (1 - lam) * x[index, :]
12      y_a, y_b = y, y[index]
13
14      return mixed_x, y_a, y_b
```

Listing 1: Mixup implementation

Listing 1 shows the actual mixup procedure that (1) draws λ from the Beta distribution (line 3), (2) picks batch-size many random indices (line 6 and 8, respectively), (3) mixes each sample in the batch with another sample determined by the random indices (line 11).

2.2 Visualization

Figure 1 visualizes three examples of the mixup process using $\lambda = 0.5$.



(a) (b) (c)

Figure 1: Mixup visualized: (a) and (b) depict two randomly drawn samples, respectively, (c) the mixup outcome of (a) and (b) by setting λ to 0.5. Each row represents a different example.

3 Baseline

All experiments were conducted on a subset of the CIFAR-10 dataset [Krizhevsky, 2009]. This dataset consists of 60.000 images labeled to ten different classes, of which we used only 50 of each class to conduct training. Our intention was to measure the impact of manifold mixup applied to training with such a small set of training examples.

3.1 Architecture & Training

The model used to conduct our experiments consists of three blocks, each implementing three convolution layers (convolution, BatchNorm, LeakyReLU) followed by a pooling and dropout operation. The final layer is a linear layer for classification. Figure 2 visualizes the full architecture.

The initial configuration was set to train using stochastic gradient descent (SGD) with the `StepLR`-scheduler at learning rate of 0.01, weight decay of $1e^{-3}$ and momentum of 0.9. Training was executed for 100 epochs and on batches of 32 images. Random cropping and random horizontal flipping was applied to the training data, as suggested in [Goyal et al., 2017]. Using these parameters, an average accuracy of 49.73% was reached.

3.2 Improving the Baseline

Before applying mixup, we wanted to increase the baseline accuracy, so the following changes to the initial configuration were evaluated:

(1) Learning Rate Schedulers:

- `StepLR`
- `CosineAnnealingLR` [Loshchilov and Hutter, 2016]
- `CosineAnnealingWarmRestarts` [Loshchilov and Hutter, 2016]

Also, (2) weight decay of $1e^{-3}$ and $1e^{-4}$, and (3) momentum at 0.9 and 0.95.

As can be seen in the results shown in table 1, the configuration using cosine annealing with warm restarts, a momentum of 0.9 and weight decay $1e^{-3}$ performed best, so all further evaluations were executed using these parameters. To further increase the baseline accuracy, (4) we increased training from 100 epochs to 400 epochs, resulting in 53.52% of the testing examples being correctly labeled. Moreover, (5) we chose different values for the learning rate, from 0.01 up to 0.1, and we ultimately achieved a baseline accuracy of $\mathbf{56.07}\%$ by setting the learning rate to 0.1. The impact of applying manifold mixup while training is described in the following section.

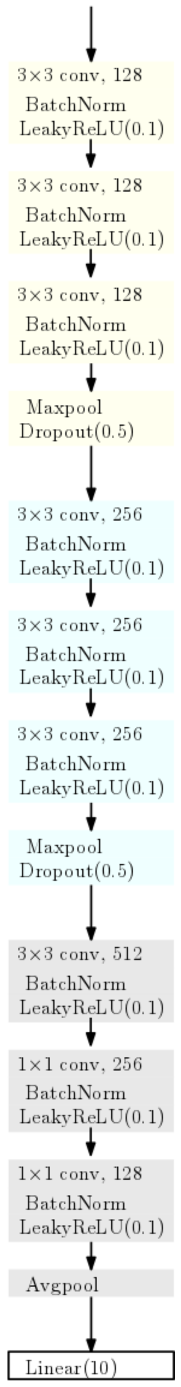


Figure 2: The network architecture: three blocks, each consisting of three convolution layers, and a linear layer for classification.

	StepLR (step size=30)			
momentum	0.9		0.95	
weight decay	$1e^{-3}$	$1e^{-4}$	$1e^{-3}$	$1e^{-4}$
accuracy	49.73%	49.80%	49.01%	48.85%

	CosineAnnealingLR (t_{max}=100)			
momentum	0.9		0.95	
weight decay	$1e^{-3}$	$1e^{-4}$	$1e^{-3}$	$1e^{-4}$
accuracy	51.16%	50.01%	51.62%	50.48%

	CosineAnnealingWarmRestarts (t_0=10)			
momentum	0.9		0.95	
weight decay	$1e^{-3}$	$1e^{-4}$	$1e^{-3}$	$1e^{-4}$
accuracy	52.59%	51.20%	50.90%	50.41%

Table 1: Evaluation results on our changes to increase the baseline accuracy of our network while training for 100 epochs at learning rate of 0.01. The first table shows accuracy when using the `StepLR` learning rate scheduler with varying momentum and weight decay, the second one the `CosineAnnealingLR`, and the third one the `CosineAnnealingWarmRestarts`. Entries show the average accuracy over five runs.

4 Experiments

We apply manifold mixup regularization on our best baseline model that is summarized in table 2. We experimented with different α values and all possible combinations of eligible layers $\mathcal{S}$. The first layers of the three blocks of our network architecture are the set of all eligible layers. Furthermore, $\mathcal{S} = \{0\}$ means input mixup as in [Zhang et al., 2017]. If $|\mathcal{S}| > 1$ then a layer $k \in \mathcal{S}$ is chosen at random before a minibatch is processed during training as done in [Verma et al., 2019].

epochs	optimizer	lr	momentum	weight decay	t_0	accuracy
400	SGD	0.1	0.9	$1e^{-3}$	10	56.07%

Table 2: Our final baseline model

Table 3 lists our evaluations with a multitude of different α-values and eligible layers $\mathcal{S}$. As described in section 1, α controls the strength of the mixup applied to one randomly selected layer from the set of eligible layers $\mathcal{S}$.

Our results show that applying manifold mixup regularization has a positive effect regardless of where and how strongly applied compared to our baseline. Allowing mixup at the last layer only yields the least improvements. On the other hand, in most experiments allowing mixup at both, the input layer and the last layer, outperformed all other combinations of eligible layers with an increase in accuracy of 2.74% to 5.66% compared to our baseline model.

When looking at the influence of the hyperparameter α, the overall best performance can be observed at $\alpha = 4.0$, with an average accuracy of $60, 17\%$, and $\alpha = 2.0$ with 60.12%. Considering that α controls the mixup strength based on the beta-distribution, our findings suggest that a strong mixup, i.e., a mixup factor λ with a high probability of being close to 0.5, has indeed a positive influence on the results.

The overall best test set accuracy was achieved with the configuration of $\mathcal{S} = \{0, 2\}$ and $\alpha = 4.0$, with an increase in accuracy of 5.66% compared to the baseline, and an improvement of 0.97% compared to our best accuracy using only input mixup ($\mathcal{S} = \{0\}$).

$\mathcal{S}$	α						
	0.2	0.5	1.0	1.5	2.0	4.0	5.0
$\{0\}$	58.66	59.06	60.36	60.76	60.41	59.91	60.33
$\{1\}$	58.07	58.87	59.09	58.86	59.50	59.01	58.43
$\{2\}$	57.51	57.38	57.46	57.80	57.86	58.36	57.79
$\{0, 1\}$	58.45	59.64	60.08	**61.09**	61.20	60.96	61.21
$\{0, 2\}$	**58.80**	**59.76**	60.67	60.16	**61.33**	**61.73**	**61.61**
$\{1, 2\}$	57.73	58.56	59.37	59.32	59.69	59.79	59.37
$\{0, 1, 2\}$	58.29	59.25	**61.13**	60.76	60.87	61.44	61.05

Table 3: Test set accuracies using manifold mixup regularization with different mixing coefficients α.

5 Conclusion

In this report, we have explored the ability of the neural network described in section 3 to correctly assign labels to images from the CIFAR-10 dataset [Krizhevsky, 2009] when trained only on a very small number of training examples. First, different learning rate schedulers were evaluated as well as the impact of changes in momentum, weight decay and the learning rate. Ultimately, a test set accuracy of 56.07% was reached.

Based on this, the regularization technique manifold mixup [Verma et al., 2019] was applied during training to observe further increases in accuracy. The influence of hyperparameters of mixup was evaluated, and in our experiments (1) applying mixup always led to better labeling performance and (2) manifold mixup yielded even better results than input mixup only. In general, mixup at the input and at the last layer gave highest accuracy. Additionally, our results suggest that a high probability for strong mixup (i.e., taking half of the first image and half of the second) is advantageous.

Overall, applying manifold mixup when training with only 50 images per class led to an increase of 5.66% in test set accuracy compared to our best baseline performance, and a final accuracy of 61.73%. This easy-to-implement regularization technique showed to be an effective tool for such classification tasks.

6 References

[Goyal et al., 2017] Goyal, P., Dollár, P., Girshick, R. B., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., and He, K. (2017). Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[Loshchilov and Hutter, 2016] Loshchilov, I. and Hutter, F. (2016). SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983.

[Simard et al., 1996] Simard, P. Y., LeCun, Y., Denker, J. S., and Victorri, B. (1996). Transformation invariance in pattern recognition-tangent distance and tangent propagation. In *Neural Networks: Tricks of the Trade*, pages 239–27.

[Verma et al., 2019] Verma, V., Lamb, A., Beckham, C., Najafi, A., Mitliagkas, I., Lopez-Paz, D., and Bengio, Y. (2019). Manifold mixup: Better representations by interpolating hidden states. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 6438–6447.

[Zhang et al., 2017] Zhang, H., Cissé, M., Dauphin, Y. N., and Lopez-Paz, D. (2017). mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412.