EE 260 HW#3 - Data Augmentation (due May 31 2020)

Motivation: This exercise will explore the benefit of data augmentation on the learning performance. You will work with the CIFAR10 dataset. CIFAR10 is a standard image classification dataset frequently used as a benchmark for machine learning algorithms. The dataset is available at https://www.cs.toronto.edu/~kriz/cifar.html.
Forming your training data: CIFAR10 images are of size 32×32 and contains pictures corresponding to 10 classes (e.g. airplane, cat, dog, ...) and our goal is deducing which class the image corresponds to. The dataset has 50,000 training and 10,000 test examples. During this assignment, in order to simplify the computation, **we will work with 10,000 training examples**. You should obtain your training set by sampling 1,000 examples uniformly at random out of 5,000 for each class. Use the exact same dataset for each task of the assignment.
Model: In this exercise you are expected to use ResNet20. Example ResNet models can be found at

- PyTorch https://pytorch.org/hub/pytorch_vision_resnet/
- TensorFlow/Keras https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet

You should use the same hyperparameter configuration in all your experiments for fair comparison. We recommend training for 100 epochs with Adam optimizer and learning rate 0.001. You are encouraged to use more epochs or better learning rate schedules to get better performance. For augmentations to work well, it is critical that you **normalize each image to have mean zero**[1].
Remarks: This exercise will ask you to design different data augmentation types and quantify the benefit of data augmentation. Thus you should **disable all types of data augmentation** in your initial model so that we can assess the benefit of different augmentation types. Also make sure your ResNet model is **not pretrained**. You are supposed to implement all data augmentation types **by your own code**.

Assignment

Your tasks are as follows. Besides the initial ResNet model, all algorithms should be your own code. For each step you are expected to report the results by plotting test accuracy, training accuracy and training loss over epochs and recording the final test accuracy.

1. (3 pts) Train your Resnet model without augmentation and report the results.
2. (4 pts) **Mixup** augmentation is based on the paper https://arxiv.org/pdf/1710.09412.pdf. As the name suggests, it mixes a pair of training examples (both inputs and labels). Given a pair of training example $(x_1, y_1), (x_2, y_2)$, we obtain the augmented training example (x, y) via

$$x = \lambda x_1 + (1 - \lambda)x_2 \quad y = \lambda y_1 + (1 - \lambda)y_2$$

   where mixing parameter λ has β distribution[2] with parameter α.

   TODO: Implement mixup and report the results for $\alpha = 0.2$ and $\alpha = 0.4$. Note that, in each minibatch, all training examples should have mixup transformation before gradient calculation (e.g. from original minibatch obtain a new minibatch by mixing random pairs of training examples).

[1] Optionally, you can also normalize each image to have the same standard deviation.
[2] https://en.wikipedia.org/wiki/Beta_distribution, In Python: numpy.random.beta

3. (4 pts) **Cutout** augmentation is based on the paper https://arxiv.org/pdf/1708.04552.pdf. For each training image with 50% probability you keep the image intact. With 50% probability, select a random pixel which serves as the center of your cutout mask. Then, set the square mask of size $K \times K$ pixels around this center pixel to be zero. Note that part of the mask is allowed to be outside of the image. For visualization, see Figure 1 of the paper.

   TODO: Implement and use cutout augmentation with $K = 16$ and report the results.

4. (4 pts) **Standard** augmentation applies horizontal flip and random shifts. See the website https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-ne for illustrations. Given an input image, first you shift it left-right and up-down as follows. Pick two independent integers k_1, k_2 uniformly between $[-K, K]$ range. Move image upwards by k_1 and rightwards by k_2 pixels (negative value means downwards and leftwards). Zero pad the missing pixels. After this random shift, with 50% probability, apply a horizontal flip on the image.

   TODO: Implement standard augmentation with $K = 4$ and report the results[3].

5. (3 pts) Combine all augmentations together. First apply standard and cutout augmentations on the training images and then apply mixup to blend them. For mixup, use the parameter α that has higher test accuracy. Report the results. Does combining improve things further?

6. (2 pts) Comment on the role of data augmentation. How does it affect test accuracy, train accuracy and the convergence of optimization? Is test accuracy higher? Does training loss converge faster?

[3]Note that you can accomplish this random shift by zero-padding the images 4 pixels on each side to obtain a 40x40 image and then choosing a 32x32 random crop.