Object recognition and computer vision : Assignment 3

Thibault de Surrel
thibault.de-surrel-de-saint-julien@ensta-paris.fr

This document is the report for the third assignment for the Object recognition and computer vision class. It details the method I used in order to train a CNN to do image classification of the Caltech-UCSD Birds-200-2011 bird dataset.

1. The model

For my CNN I used the ResNet152 model as described in [2]. In fact, this model is one on the top tier CNN models for image classification and it is already implemented in PyTorch. This model has over 60 millions parameters and I used the weights already trained using ImageNet so I did not have to train the whole model from scratch. I just added a final fully connected layer that has 20 outputs, corresponding to the 20 classes of the data set.

In order to train this model, I used Stochastic Gradient Descent with a learning rate of 10^{-3} and a momentum of 0.9.

The images on which ResNet152 has been trained are of size 224×224, so I decided to resize the images of the dataset to this size. In fact, most of the training images have a larger size than 224×224, so we are downscaling the images.

2. Improvements

2.1. Data augmentation

The training dataset had not a lot a images for each classes : between 50 and 60 for each classes. Therefore, I tried to use two data augmentation techniques in order to increase the size of the training set.

Autoaugment I used the AutoAugment technique as described in [1]. This technique is already implemented in PyTorch and it transform automatically the training data based on a policy learned on ImageNet. This way, we do not have to choose manually the transformations that will be applied to the data, but the AutoAugment automatically chooses the best transformations to apply.

Personal data augmentation For this, I chose to impose a random affine transformation of the image keeping center invariant, then a random crop to size 224×224, a random horizontal flip 50% of the time. Finally, I randomly select a rectangle region the image and erase its pixels. All of these transformations are implemented in Pytorch.

2.2. Batch Size

By default, the batch size for the training was 64. I tried to reduce this batch size to 32. In fact, a smaller batch size gives a learning process that converges quicker than a larger one but at the cost of noise in the training process.

2.3. Learning rate scheduler

I also implemented a learning rate scheduler to have the learning rate at 10^{-2} during the first 10 epochs and then have it decrease to 10^{-3}. This way I hope to do some finer tuning at the end of the training.

3. Results

This are the results I get for my network, more precisely the accuracy on the validation set :

Data augmentation	Accuracy on validation set
Autoaugment	88 %
Personal data augmentation	89 %

These are the accuracy on the validation set. Strangely, once I upload my results to Kaggle, the accuracy was a lot less (around 72 %). I did not manage to understand why there was such a difference between the validation and test set. I thought of overfitting but I would have seen it on the validation set.

References

[1] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1