Chapter 6

Data augmentation, Loss functions

Reading

1. Bishop 5.5.3

6.1 Data augmentation

In the previous chapter, we looked at convolutions as a way to reduce the parameters in a deep network, but more importantly as a way of building equivariance/invariance to translations. There are a lot of nuisances other than translation that do not have a group structure which precludes operations such as convolutions that we can perform to generate equivariance/invariance.

In this section, we will discuss techniques to build invariance to nuisances that are more complex than just translations, these techniques will seem brute-force but they also allow us to handle more complex nuisances. The main trick is to *augment* the data, i.e., create variants of each input datum in some simple way such we *know* that its label is unchanged. If our original dataset is $D = \left\{(x^i, y^i)\right\}_{i=1,\ldots,n}$ we create an augmented dataset

$$D^T = \left\{(T(x^i), y^i)\right\}_{i=1,\ldots,n} \cup D. \tag{6.1}$$

where T is some operation of our choice. We have therefore expanded the number of samples in the training dataset to $2n$ instead of the original n. Effectively, data augmentation is a technique to create a dataset that is sampled from some other data distribution P than the original one.

6.1.1 Some basic data augmentation techniques

The most popular data augmentation techniques are setting T to be changes in brightness, contrast, cropping the image to simulate occlusions, flipping the image horizontally or vertically, jittering the pixels of the input image to simulate noise in the CCD of the camera/weather, padding the image which changes the borders of the input image, warping the image using a projection that simulates the same picture taken from a different viewpoint, thresholding the RGB color channels, zooming into an image to simulate changes in the scale etc.

You can see these operations at https://fastai1.fast.ai/vision.transform.html#List-of-transforms.

⚠ FastAI is a wrapper on top of PyTorch and is an excellent library to learn for doing your course projects.

6.1.2 How does augmentation help?

A number of such augmentations are applied to the input data while training a deep network. This increases the number of samples n we have for training but note that different samples share a lot of information, so the effective novel samples has not increased by much. Let us get an idea of when augmentation is useful and when it is not. Consider a regression and classification problem as shown below.

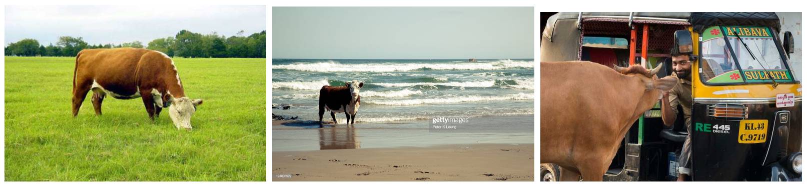

Figure 6.1: Cows live in many different parts of the world. A classifier that also uses background information to predict the category is likely to make mistakes when it is run in a different part of the world. Augmenting the input dataset on the left by replacing the background to include a mountain or a city is therefore a good idea if we want to run the classifier in a different part of the world. This will also force the classifier to *ignore* the background pixels when it classifies the cow, in other words the classifier is forced to become invariant to backgrounds by brute-force showing it different backgrounds.

In essence, data augmentation forces the model to tackle a larger dataset than our original dataset. The model is forced to learn what nuisances the designer would like it to be invariant to. Compare this to the previous chapter: by replacing fully-connected layers with convolutions and pooling we made the model became invariant to translations. In principle, we could have trained a fully-connected deep network on a very large augmented dataset with translated objects. In principle, this would make the fully-connected network invariant to translations as well.

6.1.3 What kind of augmentation to use when?

In the example with regression, we saw that the regressor on the augmented data was essentially linear and had much less discriminative power than a polynomial regressor. This was of course by design, we chose to augment the data. If the test data for the problem came from the polynomial instead of our augmented distribution, the new classifier will perform poorly.

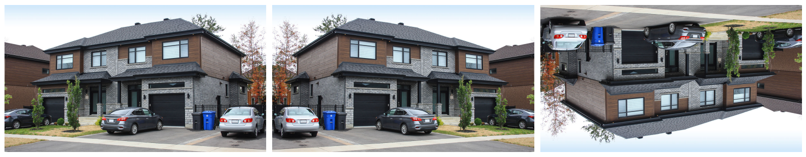
Figure 6.2: The second panel shows the original scene with a mirror flip (i.e., across the horizontal axis) while the third panel shows the original scene after a water reflection (i.e., flip across the vertical axis). The latter is an image that is very unlikely to occur in the real world, so it is not a good idea to use it for training the model.

By being invariant to a larger set of nuisances than necessary, we are wasting the parameters of the model and risk getting a large error if the test data was not from the augmented distribution. By being invariant to a smaller set of nuisances than necessary, we are risking the situation that the test data will have some new nuisances which the classifier will perform poorly on. It is important to bear in mind that we do not always know what nuisances the model should be invariant to, the set of transformations in data augmentations necessarily depends—often critically—upon the application.

❓ If you are building a classifier for detecting cars, motorbikes, people etc. for autonomous driving application, do you want to be the invariant to rotations?

Data augmentation requires a lot of domain expertise and often plays a huge role in the performance of a deep network. You should think about what kind of augmentations you will apply to data for speech processing, or for data from written text.

6.2 Loss functions

Bibliography