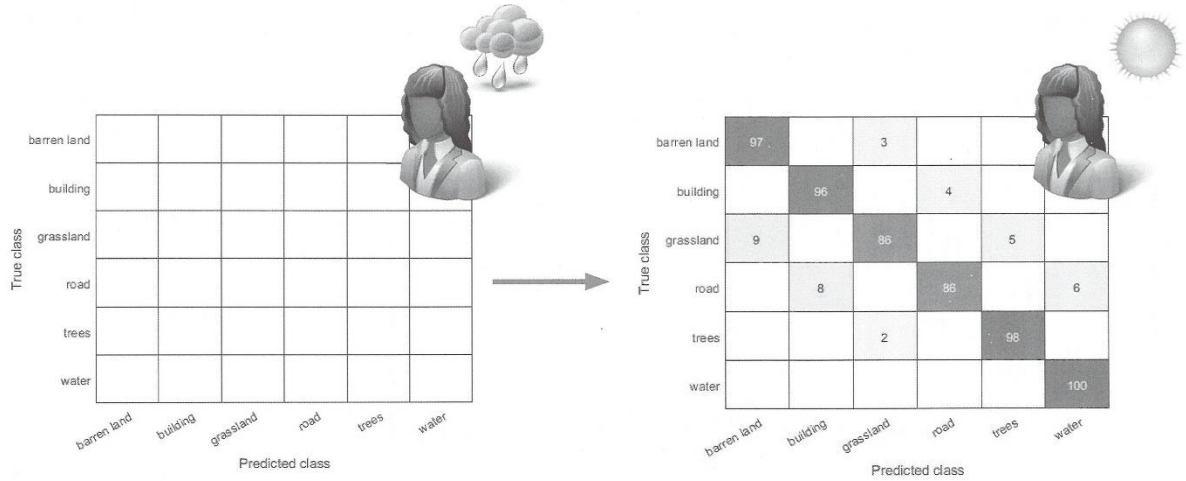
# Improving Performance of Land Cover Classifier

In the last chapter, you trained the land cover network using previously determined training options. How were these options selected?

The two main considerations when evaluating network performance are accuracy and training time. There is typically a trade-off between accuracy and training time.

A network trained using the default training options does not always provide good results. There is no "right" way to set the training options, but there are a few recommended steps to improve the performance of your network. In this chapter, you will improve the performance of the land cover classifier trained by changing various aspects of the training process.



# Improving Performance Overview

Training a deep neural network requires three components:

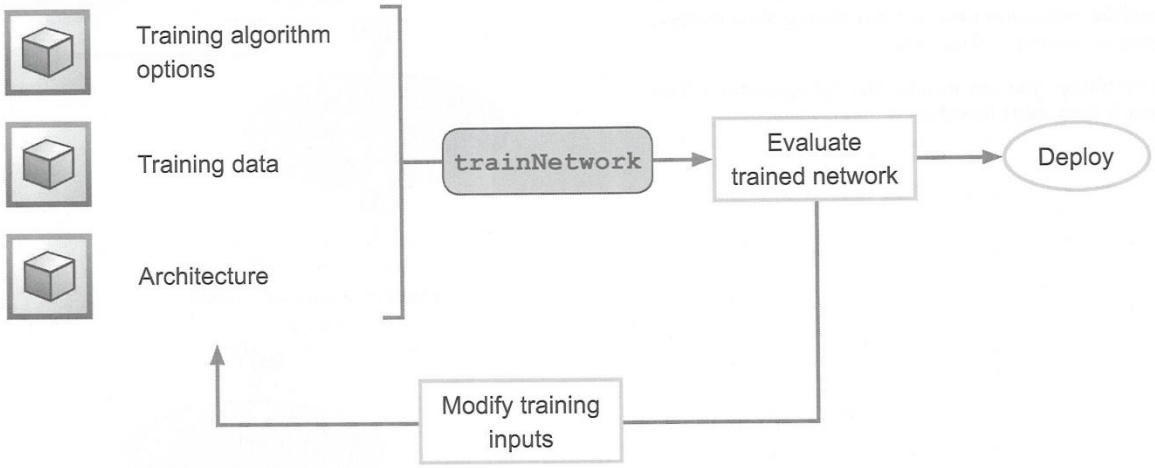
• Data with known labels

• An array of layers representing the network architecture

•Options that control the behavior of the training algorithm

After the network has been trained, accuracy you can evaluate the new network. If you are happy with the performance, you can begin using this network in production. However, the accuracy of the network is often not adequate.

To improve the performance, you can look at changing the three training inputs.



## Train Algorithm Option

Modifying the training options is generally the first place to begin improving a network. Up until now, training options that were found to be effective were provided to you. In this chapter you will learn the process behind the selection of those options.

## Training Data

If you do not have enough training data, your network may not generalize to unseen data. If you cannot get more training data, augmentation is a good alternative.

## Architecture

If you are performing transfer learning, you often do not need to modify the network architecture. However, if you are training a network from scratch, is more difficult to know whether your architecture is effective or not. One option is to use an existing architecture from a problem similar to yours.

# Guidelines for Training Options

The chart on the next page shows some general guidelines with setting training options to train a convolutional neural network.

When possible, you should use a validation data set and use the training progress plot to observe network training.

The ideal learning rate is the largest value such that the network trains effectively.

•If the learning rate is too small, it will take a very long time to train your network

•If the learning rate is too large, you may converge to a suboptimal set of parameters.

To find this learning rate, you should increase the learning rate until you see the loss “explode”. Then, decrease the loss until training proceeds appropriately.

Next, you should ensure that your network finishes learning. Increase the number of epochs until the training loss plateaus. Once training has converged, you can set the learning rate to drop at this point.

If your network is overfitting, you can increase the *L2 regularization*. This parameter controls much the weights should decay during training.

# Modifying Training Data

## Balance or Weight Classes

If you have very unbalanced classes, try splitting the data so that the training images have an equal number of samples from each class.

## Normalize

If you are performing regression, normalizing can help stabilize and speed up network training.

## Image Augmentation

If you notice that the validation loss is consistently higher than the training loss, you can to augment your training data by transforming them with a combination of translation rotation, scaling, etc. Augmentation prevents overfitting because the network is unable to memorize the exact details of the training data.

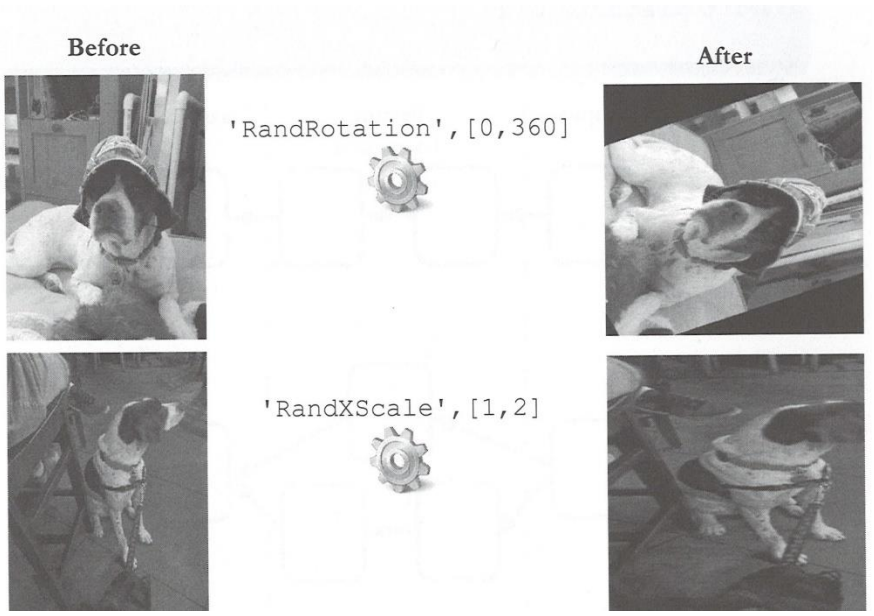
# Augmenting Images

You can use an augmentedImageDatastore to perform augmentation. When you train the network, the datastore will randomly transform the images using settings you provide. The transformed images are not stored in memory, which is useful when training on large data sets.

To provide those settings, use the imageDataAugmenter function. For example:

•RandXScale: Range of horizontal scaling specified as a 2-element vector; similarly RandYScale affects the range of vertical scaling.

• RandXReflection: Horizontally flip image with 50% probability.




Thus, the augmentation of your images is done in the following two steps:

```
aug = imageDataAugmenter( …);
augds = augmentedImageDatastore(outputSize, imds, ‘DataAugmentation’, aug);
```