Appendices: Automated Detection of Geological Landforms on Mars using Convolutional Neural Networks

Leon F. Palafox[a], Christopher W. Hamilton[a], Stephen P. Scheidt[a], Alexander M. Alvarez[b]

[a] *Lunar and Planetary Laboratory, University of Arizona, Tucson, Arizona, USA*
[b] *College of Medicine, University of Arizona, Tucson, Arizona, USA*

Appendix A. ConvNet Architecture

Like a traditional ANN, a ConvNet has layers, units and activation functions. The input of a traditional ConvNet is an image I of size $m \times m \times r$, where m is the height and width, and r is the number of channels. In traditional Red, Green, Blue (RGB) channel images; the number of channels is three, while in multispectral remote sensing images the number of channels can be larger. For our paper, we use grayscale images, in which the number of channels is one.

Appendix A.1. Data Processing in a ConvNet

The input data of a ConvNet is an image I and a target label L. The image starts in the input layer, and is processed sequentially in a series of intermediate layers. These layers transform the data and extract important features of the image via convolutions and pooling. At the end of the network, the output will be a predicted label L^*.

To train the ConvNet, it needs a set of images D, which contains N images and N labels. The objective of the training step is that the predicted labels L^* are the same as the real N labels. To achieve this result, the training is done via a process called backpropagation.

Appendix A.2. ConvNet Architecture

The first layer consists of k filters (or convolutions), each of which is of size $n \times n \times q$, n is a number lower than m, and q can be the same size or lower than the number of channels r. Each of the k filters is convolved with the input image I to generate k feature maps. Each of the feature maps is of size $mn + 1$. For instance, an image of size 100×100, and a first layer of 100 units each with 8×8 filter size, would generate 100 feature maps, each of size 95×95. This

*Corresponding author
Email address: leonp@lpl.arizona.edu (Leon F. Palafox)

configuration results in $952 \times 100 = 902,500$ features per example, only for the first layer.

To reduce the number of features in subsequent layers, and prevent overfitting, the next layer is designed to reduce the number of features. The layer is called a pooling layer. The pooling layer subsamples each of the k feature maps using either mean or max pooling. Max pooling calculates the max value in a region of size $p \times p$ in the feature map, while mean pooling calculates the mean of the same region. The $p \times p$ regions are contiguous, non-overlapping in each of the feature maps. For the previous example of 96×96 sized feature maps we can select 24 pooling regions of 4×4, to generate 100 maps of 24×24. Greatly reducing the number of features of the classifier, from ~900,000 features to ~50,000 features. The pooling features add the property of translational invariance, since it focuses on contiguous regions in the images rather than the whole image to calculate the features. A slightly translated feature will look the same if the translation happens inside the same pooling region.

After the pooling layer, we apply the transformation function f. The function, like a standard ANN, bounds the output within the region $[0, 1]$ or $[1, 1]$. We can use any activation function that maps the domain of the real numbers to a bounded output. However, the most used functions are the sigmoid, the hyperbolic tangent (tanh) and the Rectified Linear Unit (ReLu; Nair and Hinton 2010).Wan et al. (2013) presented a survey of the effect of different activation functions in the results of the classification.

The next layer after the transformation function can be another convolutional layer, or the final layer, which usually is a fully connected ANN. The fully connected ANN will have its own transformation function f, which will be the final output for the classifier. In the case of binary classification (only two classes), f is a sigmoid function, while in the case of multiclass classification, f is a softmax function. In the final layer, the output is a matrix of $1 \times C$, where C is the number of target classes. Each value of the matrix will be a bounded real number. The predicted class is the maximum value within the output matrix.

Appendix A.3. Training of a ConvNet

To calculate the bests set of filters, we used the backpropagation algorithm to find the optimum parameters of the network. The optimization algorithm consists of the following steps:

1. Forward propagation: A subset of the training data (also called batch) is passed through the layers of the networks. The result of this step is a subset of predicted labels;
2. Computation of error: An error for the batch is calculated based on the real labels and the predicted labels;
3. Backpropagation of error: The error for the batch is backpropagated thorough the network, by calculating the "share" of error of each individual unit; and
4. Update of weights: New kernels are calculated using the errors of each unit and a Newtonian optimization method called stochastic gradient descent.

When all the batches have been passed, we start again from the first batch, until the final error converges to a minimum value.

Appendix A.4. Hyperparameters of a ConvNet

A convolutional neural network has a relatively low number of hyperparameters. To prevent overfitting, we usually use a regularization parameter (Bishop, 2006). Regularization parameters are commonly used to prevent the network's weights from growing. The effect of the regularization is to prevent overfitting the architecture to the dataset used for training.

The dataset is not usually trained sequentially, instead it is trained in batches of data, to enhance convergence speed and thus performance speed. The patch size is a requirement of the stochastic gradient descent method, which requires to calculate the gradient at each step of the optimization.

References

Bishop, C.M., 2006. Pattern Recognition and Machine Learning. 1st ed., Springer.

Nair, V., Hinton, G.E., 2010. Rectified Linear Units Improve Restricted Boltzmann Machines, in: Proceedings of the 27th International Conference on Machine Learning, pp. 807–814.

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., Fergus, R., 2013. Regularization of neural networks using dropconnect, in: Proceedings of the International Conference on Machine Learning, pp. 109–111.