Tree species classification by leaves pictures

Hudson Martins Silva Bruno*
Anderson Rocha†

Abstract

The use of Convolutional Neural Networks (CNNs) have become popular especially in computer vision tasks, they achieved state-of-the-art performance on image classification. On this work a CNN is used to classify trees species from pictures of their leaves. In addition, pre-trained CNNs are also useful for other computer vision tasks as generic feature extractors. Thus, random forests are applied to classify the pre-trained CNN feature maps with 83% of accuracy.

1. Introduction

A plant specie can be recognized by botanists by their leaves. Although, there are estimated to be nearly half a million species of plant in the world. Algorithms for recognition of a plant by its leaves can be very useful for species population tracking and preservation, plant-based medicinal research and crop or food supply management.

A wide variety of image processing algorithms have been proposed to leaf classification over the history [1] [2] [3]. Despite those algorithms could achieve good classification results, they still have several shortcoming, specially when they are submitted to images with noises or varying lighting conditions.

Deep Learning techniques have been widely explored after the recent development of computational power and data availability. One of the most popular deep networks are the Convolutional Neural Networks (CNNs) that are used for processing data that has a known grid-like topology, mainly images. If the database is varied enough, CNNs are good for generalize the data, and the system can be robust to noise and light conditions.

In this work, a CNN model is employed to build a system that is capable of classify the plant by images of its leaves. Furthermore, the features learned by the CNN are classified by an ensemble method called random forest.

*Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: hudsonbr95@gmail.com

†Is with the Institute of Computing, University of Campinas (Unicamp). **Contact**: anderson.rocha@ic.unicamp.br

2. Proposed Solutions

2.1. Dataset

The dataset chosen for this project is called leafsnap [4]. It was jointly created by computer scientists from Columbia University and the University of Maryland, and botanists from the Smithsonian Institution in Washington, DC. The dataset consists of images of leaves from 185 tree species from the Northeastern United States, taken from two different sources, called lab images and field images.

There are 23147 lab images that consists of high-quality images taken of pressed leaves, from the Smithsonian collection. These images appear in controlled backlit and frontlit versions, with several samples per species. Furthermore, there are 7719 field images, consisting of ”typical” images taken by mobile devices in outdoor environments. These images contain varying amounts of blur, noise, illumination patterns, shadows, etc., as seen on figure 1.

For this project the dataset was split in training images and test images. The training images consists of 80% of the dataset (merging field and lab images), and the test images are composed of the other 20% of the dataset. Furthermore, 20% of the training images were used only for validation.

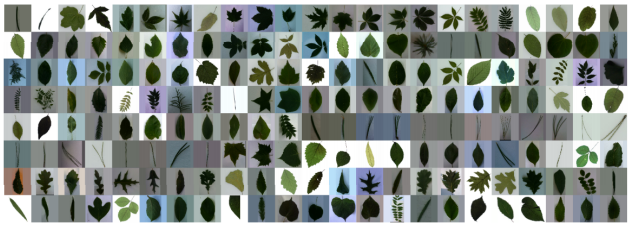

Figure 1: Field thumbnails of 184 of the 185 species present in the Leafsnap Dataset

2.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a specialized kind of neural network for processing data that has a known grid-like topology, e.g. images. The most important operation in a CNN is the convolution, that is basically

a weighted sum between two signals. We calculate convolution at a particular location of the image by extracting a chunk from the image and then multiplying the values in this chunk element-by-element with a convolution filter (or kernel) and then add them all to obtain a single output. The convolution filter is an array of weights that will be learned during the training.

Other operations can be performed on a CNN after the convolution, such as a non-linear activation function, e. g. rectified linear activation function, and a pooling function, that is applied to replace the output of the net with a summary statistic of the nearby outputs [5]. The pooling helps to avoid overfitting, since it reduces the number of parameters to be computed. The most common form of pooling is Max-pooling where we take a filter of size p and apply the maximum operation over the sized part of the image [6].

After several convolutional layers, a fully connected (FC) layer is attached to the end of the network, it takes as input the output of the convolutional network and returns an N dimensional vector, where N is the number of classes. The output vector is composed of the probabilities of each class given the input. Therefore, the FC layer looks at what high level features most strongly correlate to a particular class and has particular weights so that when you compute the products between the weights and the previous layer, you get the correct probabilities for the different classes [7].

The training step is the most exhaustive one on Deep Learning algorithms. This occurs due to the presence of the backpropagation function, which is responsible to calculate the fittest values for the parameters in the network. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights. The "backwards" part of the name stems from the fact that calculation of the gradient proceeds backwards through the network, with the gradient of the final layer of weights being calculated first and the gradient of the first layer of weights being calculated last. It is considered an efficient algorithm, and modern implementations take advantage of specialized GPUs to further improve performance.

Mathematically, a backpropagation function can be defined as a generalization of the delta rule used on multilayer perceptrons. Training a neural network with gradient descent requires the calculation of the gradient of the error function $(E(X, \theta))$ with respect to the weights w_{ij}^k and biases b_i^k . Then, according to the learning rate α, each iteration of gradient descent updates the weights and biases (collectively denoted θ) according to the equation 1.

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial(E(X, \theta))}{\partial \theta} \tag{1}$$

where θ^t denotes the parameters of the neural network at iteration *t* in gradient descent.

The CNN architecture used on this work is based on the architecture proposed by [8]. The CNN structure is presented on the figure 2, and it is composed of 4 convolutional layers and 4 Max-pooling layers. The input size of the network is 300x300x3, then all of the images were resized to this dimensions. As the number of classes on this problem is 185, the last FC layer is composed of 185 neurons.

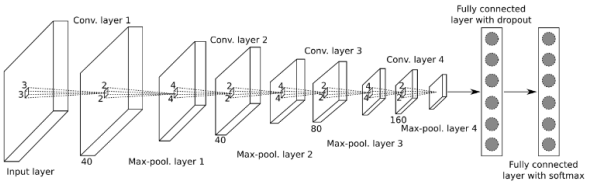

Figure 2: The chosen architecture, proposed by [8]. The two numbers at the small block indicate the kernel sizes in convolutional or max-pooling layers. The digit below a convolutional layer specifies the number of kernels. The stride of convolutional and max-pooling layers it is 1 and 2, respectively. The first fully connected layer consists of 500 output nodes.

2.3. Data Augmentation

Data augmentation is one of the most used techniques when training CNNs, especially when the dataset is not large enough, which leads to overfitting. Data augmentation is the process of artificially creating more images from the images that we already have, this can be done in a lot of ways, e.g. rotating the image, changing its size, shifting the image, etc. On this way, with a few operations on each image we can increase the dataset significantly. To avoid memory waste, we perform data augmentation on the fly on each batch of training.

The data augmentation operations made on this work were:

- Horizontal and Vertical flips: Randomly flipping half of the images horizontally or vertically;
- Width shift and Height shift: Randomly translate pictures vertically or horizontally in the range 0 - 20% of the size of the images;
- Rotation: Randomly rotate the images in the range 0 - 30 degrees
- Zoom: Randomly zoom inside the image in the range 0 - 20% of the size of the images.

2.4. Random Forest

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. It uses the bagging algorithm idea, thus each tree is built from a sample drawn with replacement from the training set.

The random forest creation pseudocode is the following:

1. Randomly select k features from total m features, Where $k << m$
2. Among the k features, calculate the node d using the best split point
3. Split the node into daughter nodes using the best split
4. Repeat 1 to 3 steps until l number of nodes has been reached.
5. Build forest by repeating steps 1 to 4 for n number times to create n number of trees.

For this project the number of features selected for each classifier (k) is defined as $\sqrt{m}$, where m is the total number of features.

To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a class probability, voting for a class. The forest chooses the classification by averaging the probabilistic prediction of each classifier (scikit learn implementation [9]).

2.5. Classifying CNN features

CNN models which are trained for classification have been used as feature extractors by removing the output layer. Features extracted from a pre-trained CNN have been successfully used in computer vision tasks such as scene recognition, object attribute detection and achieves better results compared to handcrafted features [10]. The results from [11] shows that Random Forest and Support Vector Machines (SVMs) can be used with features from a CNN to yield better a prediction accuracy compared to the original CNN.

Therefore, random forests were employed on this project to classify the features from the last convolutional layer of the CNN. Thus, the fully connected layers were removed and a random forest is connected at the end of the CNN, getting the flattened output of the last convolutional layer, as seen on figure 3.

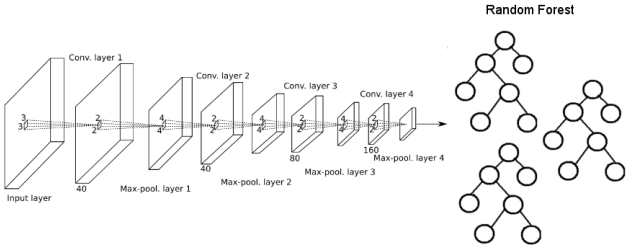

Figure 3: Random forest classifying CNN features.

3. Experiments and Discussion

At first, the CNN was trained only with the lab images on the training and test sets. After 50 epochs of training, the network reached an accuracy of 81.03% on the test set with data augmentation. However, in order to increase the network robustness the field images were merged in the dataset, respecting the proportion of 80% of the images for training and 20% for the test set. After this, the network accuracy on the test set has fallen to 60.88%.

Then, the CNN was trained for 50 more epochs with field and lab images merged on the training and test sets, using as initial weights the ones learned on the previous network (trained with lab images only). The accuracy with this new training set is 72.08% on the field and lab images test set.

After this, the fully connected layers were removed and the random forest was trained with the features learned by the CNN. The total number of features outputed by the CNN is 640, thus, each tree will select 25 features ($\sqrt{640}$). The number of trees that gives the best trade-off between accuracy and performance is 200, found after a grid search. After training the random forest with the lab and field images dataset, the classification accuracy of the system (CNN + Random Forest) went to 83.34%. The figure 4 presents a confusion matrix on the test set for this classifier, it is possible to notice that the diagonal of the matrix is much higher than the other positions, as expected for a good classifier.

4. Conclusions and Future Work

The system was able to classify with high accuracy the 185 classes of leaves from the Leafsnap dataset. Furthermore, random forest were successfully applied to classify the features learned by the CNN. It is also important to notice that with a small CNN (about 500.000 parameters) a good classifier was built, using few training time and computer memory when compared to others architectures.

For future works, other classifiers such as SVM, should be employed to learn the CNN features. Besides that, as

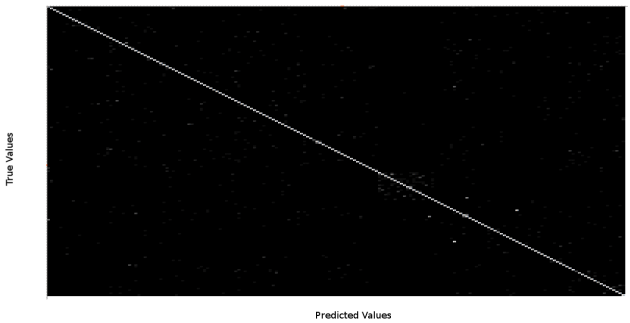

Figure 4: Confusion matrix for the CNN+Random forest classifier, that got 83.34% of accuracy. On this matrix the lower values are colored in black and the higher values are colored in white.

shown in [11], classifying the features from other convolutional layers can yield good results as well.

References

[1] S. ZHANG and K. CHAU. Dimension reduction using semi-supervised locally linear embedding for plant leaf classification. pages 948–955, 2009. 1

[2] AHIAOUI I. MOUINE, S. and A. VERROUST. A shape-based approach for leaf classification using multi scale triangular representation. 2010. 1

[3] A. EHSANIRAD and S. Y. H. KUMAR. Leaf recognition for plant classification using glcm and pca methods. pages 31–36, 2010. 1

[4] et. al. KUMAR, N. Leafsnap: A computer vision system for automatic plant species identification. http://leafsnap.com/, 2012. 1

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 2

[6] Vikas Gupta. Image classification using convolutional neural networks in keras. https://www.learnopencv.com/image-classification-using\-convolutional-neural-networks-in-keras/, 2017. 2

[7] Adit Deshpande. A beginner's guide to understanding convolutional neural networks. https://adeshpande3.github.io/, 2016. 2

[8] C. Wick and F. Puppe. Leaf identification using a deep convolutional neural network. University of Wurzburg, Germany, 2017. 2

[9] Scikit Learn. Ensemble methods. http://scikit-learn.org/stable/modules/ensemble.html, 2018. 3

[10] et.al. Razavian, A. S. Cnn features off-the-shelf: an astounding baseline for recognition. (Royal Institute of Technology, Sweden, 2014. 3

[11] B. Athiwaratkun and K. Kang. Feature representation in convolutional neural networks. Department of Statistical Science, Cornell University, 2015. 3, 4