Plant Identification Using Tensorflow
Senior Project Final Report
Kody G. Dangtongdee
Dr. Franz Kurfess
CPE 462, Winter 2018
Computer Engineering Department
College of Engineering
California Polytechnic State University, San Luis Obispo
Abstract

Exploring the diverse nature that is available in San Luis Obispo is one of my favorite activities, however, I often am frustrated that I am unable to identify or learn more about the plants that I appreciate. This project is an attempt at using the concepts of neural networks to create an image classifier which can identify plants. Machine learning is one of the biggest topics in computer science at the moment, and its many uses make it a topic that will be continue to be researched for a very long time. Convolutional neural networks are a popular realm of machine learning, and are often used for image classification, as in this project.
Introduction

Problem Overview

It takes very little time living in San Luis Obispo for one to realize why so many active people, and fans of the outdoors, are drawn to the area. Countless beautiful trailheads and hikes lie within a 30 minute drive of the Cal Poly campus, and each features a diverse variety of plant-life throughout every season of the year. However, for people like myself who have very little experience with classifying types of plants, it can be frustrating to be unable to learn more about specific plants without sifting through plant classification field manuals. The purpose of this project is to use Tensorflow, an open-source dataflow and machine learning library, to build an image classifying Convolutional Neural Network (CNN) for classifying the plants native to San Luis Obispo (and California in general). Tensorflow, in addition to providing developers a simple way to build neural network layers, can also be run on mobile platforms such as Android. The ultimate goal of this project is to design and optimize a convolutional neural network for use with plant classification, and eventually build a simple classification app for mobile devices around the trained network. The mobile app will allow users to try and classify plants while outdoors or offline.

Background

There are many applications of CNNs, one of the most popular of which is image classification, and CNNs have even been used specifically for plant classification. These plant classification neural networks, however, generally use datasets with very specific details, such as the Swedish leaf dataset, Flavia (Liu 2018). Highly advanced methods of plant classification have been designed with focus on details such as the veins of a leaf using the snakes technique (Li 2005). Thus, this project is explorative and aimed at learning how to design a neural network using Tensorflow, but ultimately has practical applications for developers, botanists, or nature enthusiasts. The dataset used for training was collected through Google Images, and the images of the plants themselves are very general, rather than of specific characteristics of the plants, and typically include most of the identifying characteristics of a plant.
Project Overview

Potential Users

Developers

In its current state, the project is not yet ready for practical use, someone who wishes to learn the basics of CNNs and Tensorflow, substituting their own classes. The project is also ideal for anyone that wants to improve the project. Improvements can be made to both the dataset and the CNN architecture, however, this will require more research and knowledge of CNN design. Additionally, someone who is interested in Tensorflow mobile development or front end development could be interested in designing an mobile application with a user-friendly interface.

Scientific Use

The CNN and classifier have potential to be used in a lab setting by botanists or plant biologists, although any scientist would likely wish to use their own dataset. The project would most likely be useful to scientists for classifying large amounts
of data and generating statistics for the dataset. Another use of this project in its current state, could be for plant scientists who want help identifying diseases in plants if the CNN is trained with a dataset customized for this. Once a mobile application has been built around the CNN, Botanists or other scientists dealing with plants in the field could also benefit by being able to quickly identify if a plant is unhealthy.

Hikers or Nature Enthusiasts

The project requires more work before it will be practical for typical users while spending time outdoors, specifically, there needs to be a mobile application with a user friendly interface. However, once a user interface has been created, the application has the potential to be very useful for people who want to just know more about the plants they see.

Project Goals and Objectives

Project Goals

1. Research plants native to San Luis Obispo and databases of plant images.
2. Research Tensorflow and CNNs, and learn basic network architecture design.

Project Objectives

1. Collect a dataset of over 80,000 images of plants using their genus-species classification as the Google Image search term.
2. Produce a convolutional neural network which is capable correctly classifying images of plants with an average confidence level of 80% or more.

Project Results and Current State

The results of this project, as of Winter 2018, are semi-successful and has the potential for future improvements. Due to a lack of existing dataset for the specific classes desired for this project, a data collection script had to be used. The data collection script currently takes a list of scientific names for plants of the San Luis Obispo area, which was compiled from the Calflora plant observation tool. The dataset is lacking in quality because it is difficult to sift through the data and eliminate undesired images, which tend to appear in the Google Image searches. The CNN and the classifier are inconsistent, with some tests resulting in nearly 100% confidence during a correct classification, and
other tests which entirely fail to produce a correct classification. Improvements which can be made are explained in detail later on.

Dataset

There are a number of existing datasets which have images of specific plant aspects such as leaves, fruits, and flower petals. These datasets were generally collected for very specific uses with neural networks that were designed to classify plants based on certain characteristics. The dataset for this project was produced by searching Google Images using a Python script adapted from a web-crawler created by hardikvasa, available on Github. As a result, the images of the plants are a diverse collection of plants in their natural setting. This adds the benefit of training the network for use outdoors. The script had to be modified so that it would receive the list of search keywords from a text file. The search keywords for the dataset are a long list of plant observations in San Luis Obispo County, and was created by querying the observation database on the Calflora website. The search keywords use the full genus-species classification in order to increase the quality of the dataset. For example, rather than searching for the word “Iris”, which would produce undesired images, the preferred keyword would be “Iris douglasiana”. The labels for classification, however, are grouped by genus so that each class has 400 to over 1000 images.
Convolutional Neural Network

Background

Convolutional neural networks are a class of machine learning networks which are commonly applied to image visualization problems such as classification. CNNs were inspired by the connections of the neurons and synapses in the brain. The design of these networks is made up of series of convolutional, pooling, and fully connected layers. The convolutional layer does what its name describes, it applies a number of convolutional filters to the input images in order to acquire the learning parameters for the network. Pooling layers are placed in between convolutional layers, and are used to reduce the number of parameters used for learning, and thus reduce the computation required (Karpathy 2017). Finally, fully connected layers are full connections to the previous layer, rather than the small window the convolutional layers are connected to in the input.

| Collecting images in 'Abronia pogonantha' |
| Collecting images in 'Acourtia microcephala' |
| Collecting images in 'Bloomeria humilis' |
| Collecting images in 'Deinandra corymbosa' |
| Collecting images in 'Cladonia firma' |
| Collecting images in 'Agastache urticifolia' |
| Collecting images in 'Cordylanthus maritimus' |
| Collecting images in 'Echium candicans' |
| Collecting images in 'Gnaphalium bicolor' |
| Collecting images in 'Eschscholzia californica' |
| Collecting images in 'Euphorbia characias' |
| Collecting images in 'Brodiaea jolonensis' |
| Collecting images in 'Abronia latifolia' |
Convolutional neural networks are commonly used for image classification, however, there are limitations to this application. A human can identify the contents of certain images much more quickly than a computer, but CNNs have proven to have a 97.6% success rate when applied to facial recognition.

Tensorflow
Tensorflow is an open sourced library for dataflow programming, and is commonly used for machine learning purposes. Tensorflow was developed by the Google Brain team, and it provides developers with access to a library of functions which make developing machine learning applications such as image classification convolutional neural networks.

OpenCV
OpenCV is a library of functions which allows developers to work with and transform image data for applications in computer vision. In this project, OpenCV is used to open the images, and then resize them so that the input for the CNN is uniform. The current input image size is 128x128, but this is due to hardware limitations. On the main testing platform, the training program would require far too much for the machine that was normally running it.
NumPy

NumPy is a Python library which provides functions for working with multi-dimensional data such as the matrices of pixels that represent images. The NumPy library was used to test the idea of adding distortions to the image input in order to increase the performance of the CNN and classifier. NumPy is also used to divide the pixel values of the input images while transforming them to a size of 128x128.

```python
def open_image(file_path, image_size):
    try:
        image = cv2.imread(file_path)
        image = cv2.resize(image, (image_size, image_size),
                          0, 0, cv2.INTER_LINEAR)
        image = image.astype(np.float32)
        image = np.multiply(image, 1.0/255.0)
        return image
    except cv2.error as e:
        print("Error opening image." + file_path + ":" + e)
        return None
```

Image Classifier

The Python script which uses the trained Tensorflow model is very simple, because most of the time spent on this project was for collecting the data and learning how to design a CNN. The classifier is designed to take a directory of images, a text file of the labels used in the network, and the trained model itself as inputs. The classifier tests the images with the specified model and displays the results comparing the correct label with the top four classes based on the confidence level of the predictions.
Engineering Specifications

Network Design

Convolutional Layer 1

Input:
The image data is reduced to a size of 128x128 pixels in order to not overwhelm the hardware the program was normally tested on. Batches of 32 images are fed into the convolutional layer and 16 filters of 8x8 pixels are applied to the images.

```python
conv_layer1 = 
    lb.build_convolutional_layer(input=image_placeholder, 
                                num_channels=NUM_CHANNELS, 
                                filter_size=FILTER_SIZE,
```

Figure 4. Output from image classifier.
num_filters=NUM_FILTERS)

Pooling layer 1
Input:
Each pooling layer uses a pool size of 2x2 and a stride size of 2.

pool_layer1 = tf.layers.max_pooling2d(inputs=conv_layer1,
        pool_size=[2, 2], strides=2)

Convolutional Layer 2
The second convolutional layer has the same parameters as the first.

conv_layer2 =
    lb.build_convolutional_layer(input=pool_layer1,
        num_channels=NUM_FILTERS,
        filter_size=FILTER_SIZE,
        num_filters=NUM_FILTERS)

Pooling layer 2

Fully connected layer 1
Each fully connected layer performs an activation on each of its inputs. The first, however, performs a RELU activation function on the data.

connected_layer1 =
    lb.create_connected_layer(input=flat_layer,
        num_inputs=flat_layer.get_shape()[1:4].num_elements(),
        num_outputs=32,
        use_relu=True)

Fully connected layer 2
The second FC layer does not perform the RELU activation.

connected_layer2 = \
    lb.create_connected_layer(input=connected_layer1,
        num_inputs=32,
num_outputs=num_classes,
use_relu=False)

Training Step

The number of training steps can be specified as a command line parameter. Each training step is validated and tested, and the results of each step are printed to standard out.

```python
for i in range(total_iter, total_iter+FLAGS.training_steps):
    print("Running training step: " + str(i))
    image_batch, label_batch = ds.get_next_batch(images,
        'training', BATCH_SIZE, i, IMAGE_SIZE)
    feed_dict_train = {label_placeholder: label_batch,
        image_placeholder:image_batch}
    start = int(round(time.time() * 1000))
    session.run(optimizer, feed_dict=feed_dict_train)
    end = int(round(time.time() * 1000))
    print("Runtime: "+str(end - start)+"ms")
    saver.save(session, FLAGS.output_dir+"model")

    image_batch, label_batch = ds.get_next_batch(images,
        'validation', BATCH_SIZE, i, IMAGE_SIZE)
    feed_dict_validate = {label_placeholder: label_batch,
        image_placeholder:image_batch}
    loss = session.run(cost, feed_dict=feed_dict_validate)
    print("Loss: " + str(loss))

    image_batch, label_batch = ds.get_next_batch(images,
        'testing', BATCH_SIZE, i, IMAGE_SIZE)
    feed_dict_test = {image_placeholder: image_batch,
        label_placeholder: np.zeros(num_classes)}
    training_accuracy = session.run(accuracy,
        feed_dict=feed_dict_train)
    validation_accuracy = session.run(accuracy,
        feed_dict=feed_dict_validate)
    print("Training accuracy: {0:.1%} Validation accuracy:
        {1:.1%}".format(training_accuracy,
        validation_accuracy))
```
The first phase of this project was to research the available machine learning libraries, convolutional neural network design, and existing datasets. There are a number of libraries available, but Tensorflow was chosen because there are many tutorials and documentation for the library. The initial testing with tensorflow as simply following their own tutorial with their sample dataset. Once I began to have a basic understanding of the Tensorflow library, I retrained the Inception model with my own dataset, which proved to be very successful in testing, however, the goal of this project was to learn how to develop and optimize a neural network.
The initial designs of the CNN for this project were based on several different tutorials about how to use Tensorflow to design an image classifier. As simple as these networks were, they still proved successful with small specific sets of data from the dataset. The next step was to begin modifying the initial network to try and find a design that worked for the application of this project. For the sake of focusing on building a better classifier, the goal of building an Android application around the model was ditched. However, I intend to continue working on this project with better hardware, so that I can take advantage of Tensorflow’s CUDA support.

Final Design

The design of CNN which was tested the most, and is being submit as a deliverable for this project, is not highly advanced. I did not expect to become a master of designing neural networks overnight, so I would still declare this project a success, despite some of its flaws. This model also was trained using the dataset at the genus-species level, which only allows for approximately 100 images per class, which is not nearly enough. The current model in testing, which has not produced enough data to add to this report, generalizes the dataset by using labels at the genus level, which allows most classes to have between 400 and 1000 classes. This model is currently being trained and tested, but without a doubt should prove more successful due to the dataset changes.

The parameters of the network such as number of training steps, output directory, and image input directory can all be specified, however, their defaults will place all the output directory in the current working directory. The input image directory must be specified, and the contents of the directory must be folders of images. The CNN trainer will also output two text files: one containing the labels for the classifier, and the other lists which images were selected for training, testing, and validation. The classifier uses these to provide an easy to read result for each image classification, and allows for quick testing of the trained model.

Future Work

This project has plenty of room for future work, by myself or a future interested student.

1. The first changes that need to be made, as stated above, are further testing with the improved dataset.
2. Improved CNN design. The network at the moment is still basic, and much more research and practice is needed to optimize the design.
3. The dataset itself can be further improved by adding images, removing undesired ones, adding or removing classes, or by making the image content more specific, as in the Flavia dataset.

4. A mobile application can be created for use by average users or scientists working in the field.
References


