Image-based Flower Type Recognition

Shailaja Mallick
smallic@ncsu.edu

Abhilasha Lokannavar
alokann@ncsu.edu

Abhishek Bhardwaj
abhardw3@ncsu.edu

Abstract

When it comes to image recognition, human brain processes it in a much precise and fast manner; thanks to the human vision system. But when the same job comes to the computer, it gets difficult to accurately identify which type of image it is. In computer vision, researchers have made steady progress such as using the ImageNet data set to classify images to different classes. However, the models and the results achieved by the researchers are difficult to reproduce. In this project, we work towards applying theoretical concepts such as deep learning and ensemble methods into image recognition by using Neural Network model, AdaBoost methodology in order to verify if the model can predict/label the name of a flower accurately. The results of our model show that Convolutional Neural network outperforms AdaBoost and Deep Neural Network model in terms of accuracy.

1 Introduction

How to make a computer learn different kinds of flowers? When it comes to image recognition, human brain processes it in a much precise and fast manner; thanks to the human vision system. But when the same job comes to the computer, it gets difficult to accurately identify which type of image it is. Human eye sees a flower but what helps us in recognizing the species of the flower? We can see the shape and color of the flower and recognize them but there exists many other flower types with similar specifications. Besides shape and color, there are other features also such as size, patterns etc. that help us in recognizing and these features can be used in a learning algorithm to train a classifier in a flower recognition system.

Till date, it has been found that there are hundreds of thousands of species of flowers. To be able to capture flower images and classify them, is a daunting task and one may get confused about different types of flower. Hence, having a fast and accurate automated flower recognition system will bring a lot of eagerness in peoples' lives. However, there are some challenges in flower recognition like the similarity between the different types of flowers, the complex background of flower image, etc. We cannot just rely on a single feature. Therefore, appropriate model selection is an important aspect that can make use of multiple features in a recognition algorithm. In this paper we investigate different flower recognition models using multiple kernel learning approach where the flower images are mostly downloaded from web and vary considerably in size, resolution, scale, lighting etc.

One of the most well-known machine-learning technology for image recognition or classification, object recognition, speech recognition, etc. is the deep learning [7]; which are representation learning methods composed of multiple processing layers to learn the patterns of input data. The methodology behind the learning mechanism can be supervised, semi-supervised or unsupervised. We, in this project, incorporate few of the deep learning architectures such as Deep Neural Network and Convolutional Neural Network (CNN). In our data set, as the images are labelled based on its category, we use supervised learning methodology to train our model. Supervised learning refers to given an image, the model outputs in the form of a vector of scores; one for each category, aiming towards having the highest score for the desired category. We also use such characteristics of supervised learning in training our model by providing adjustable parameters and thousands of different images to properly train our model. In order to validate our model, we have used 5-fold cross validation and to test our model we have divided the data set randomly to training and testing.

We are not the first one to train a model for flower recognition application, there has been a lot a research done on very large data sets such as ImageNet [3], where the task was to classify 1000 categories and nearly every year since the ImageNet challenge, there has been many breakthroughs in developing different deep learning models for the task of image classification/recognition. One of the recent one is the paper by Gavai et al. [4] where they have a developed a new model called MobileNets which takes less space and time for classification compared to Google Inception V-3 model [9], however, compromising in accuracy. Through this project, we plan to learn and understand how the theoretical concepts of deep learning, data mining techniques translate to practical applications of machine learning algorithms by using different frameworks. This paper has the following objectives. First, we seek to pre-process the data to fit it to our model(s), train it and obtain the scores. Second, upon training and validation, we also want to investigate the accuracy of recognition, error rate and compare and analyze the results of different models for flower recognition. The rest of the paper is organized as follows. In Section 2, we provide the methodology used behind the model of our flower recognition system. In Section 3, we describe the data sets employed for our study, set up of our frameworks and experiments. And finally in Section 4, we present the results, discuss and analyze them and in Section 5, we conclude by providing implications of this project and discuss the prospects of future work.

2 Methodology

The methodology of our work is to fit different models and compare the accuracy obtained. We implemented our model in Python and recommend using Anaconda for installing required packages as it works on all the systems, Windows, Linux and Mac OS. Also, it has Spyder editor, a powerful Python editor that has all the functionalities needed to write and run the codes, and also view the results in the IPython console. It also has the option to create an environment to install the packages and let the system be untouched. It is essential if we have another version of Python installed and we don't want to have incompatibilities with it.

2.1 Keras Sequential Model

In order to implement the model, following is the procedure that needs to be followed (both for Keras Sequential and Ensemble),

Keras Python Library: One of our model i.e. the Deep Neural Network model is implemented using Keras with backend Tensorflow; which is a Python library built by *Françqis Chollet*. It is an open source neural network library and is capable of running on top of Tensorflow, Microsoft Cognitive Toolkit, or Theano. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. Keras contains numerous implementations of commonly used neural network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools that makes working with image data easier. In addition to standard neural networks, Keras also supports Convolutional and Recurrent neural networks.

Scikit-Learn Python Library Scikit-Learn or Sklearn is a machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to inter-operate with the Python numerical and scientific libraries NumPy and SciPy. For instance, in our Deep Neural Network Model, we have imported *train_test_split* method from Sklearn's sub-library *model_selection* to split the data set into training and test set depending upon the *test_size* and *train_size* parameter values.

TensorFlow Created by the Google Brain team, TensorFlow is an open source library for numerical computation and large-scale machine learning. TensorFlow bundles together a slew of machine learning and deep learning models and algorithms. It is capable of training and running deep neural networks for handwritten digit classification, image recognition, word embedding, recurrent neural networks, sequence-to-sequence models for machine translation, natural language processing, and Partial Differential Equation (PDE) based simulations.

A TensorFlow computation is described by a directed graph, which is composed of a set of nodes. The graph represents a data flow computation, with extensions for allowing some kinds of nodes to maintain and update persistent state and for branching and looping control structures within the graph. Computational graphs are typically constructed using one of the supported front-end languages like Python. In a TensorFlow graph, each node has zero or more inputs and zero or more outputs, and represents the instantiation of an operation. Values that flow along normal edges in the graph (from outputs to inputs) are tensors, arbitrary dimensional arrays where the underlying element type is specified or inferred at graph-construction time. Special edges, called control dependencies, can also exist in the graph: no data flows along such edges, but they indicate that the source node for the control dependence must finish executing before the destination node for the control dependence starts executing. The biggest advantage of using TensorFlow for machine learning development is abstraction. Instead of dealing with the underlying details of implementing algorithms, the developer can simply focus on the overall logic of the application.

2.2 Ensemble Method

In machine learning, ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions. It constructs a set of base classifiers from training data and performs classification by taking a vote on the predictions made by each base classifier.

Ensemble methods are meta-algorithms that combine several machine learning techniques into one predictive model in order to decrease variance (bagging), bias (boosting), or improve predictions (stacking). **Bagging** is an ensemble method where a random sample of the training data set is created and a classifier is built for each sample. Finally, results of these multiple classifiers are combined using average or majority voting. Bagging helps to reduce the variance error. **Boosting** provides sequential learning of the predictors. The first predictor is learned on the whole data set, while the following are learned on the training set based on the performance of the previous one. It starts by classifying original data set and giving equal weights to each observation. If classes are predicted incorrectly using the first learner, then it gives higher weight to the missed classified observation. Being an iterative process, it continues to add classifier learner until a limit is reached in the number of models or the accuracy. **Stacking** works in two phases. First, multiple base classifiers are used to predict the class. Second, a new learner is used to combine their predictions with the aim of reducing the generalization error.

Image-based Flower Type Recognition implements boosting ensemble method using AdaBoost machine learning algorithm. **AdaBoost**, short for Adaptive Boosting, is a machine learning meta-algorithm formulated by *Yoav Freund* and *Robert Schapire*. It begins by fitting a classifier on the original data set and then fits additional copies of the classifier on the same data set but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. The algorithm below describes AdaBoost method:

```
Algorithm    AdaBoost
1: Init data weights $\{w_n\}$ to $1/N$
2: for $m = 1$ to $M$ do
3:     fit a classifier $y_m(x)$ by minimizing weighted error function $J_m$:
4:     $J_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n]$
5:     compute $\epsilon_m = \sum_{n=1}^{N} w_n^{(m)} 1[y_m(x_n) \neq t_n] / \sum_{n=1}^{N} w_n^{(m)}$
6:     evaluate $\alpha_m = \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
7:     update the data weights: $w_n^{(m+1)} = w_n^{(m)} \exp\{\alpha_m 1[y_m(x_n) \neq t_n]\}$
8: end for
9: Make predictions using the final model: $Y_M(x) = \mathrm{sign}\Big(\sum_{m=1}^{M} \alpha_m y_m(x)\Big)$
```

The first base classifier $y_1(x)$ is trained using weighting coefficients that are all equal. In subsequent boosting rounds, the weighting coefficients are increased for data points that are misclassified and decreased for data points that are correctly classified. The quantity epsilon represents a weighted error rate of each of the base classifiers. Therefore, the weighting coefficients alpha give greater weight to the more accurate classifiers. The process continues until a pre-set number of weak learners have been created or no further improvement can be made on the training data set.

sklearn.ensemble.AdaBoostClassifier is used to implement multi-class adaboost classifier on the flower-recognition data set. This class implements the algorithm known as AdaBoost-SAMME.

Feature Extraction

AdaBoost learning algorithm is implemented on the features extracted from the images in the flower recognition data set. Features are the information or list of numbers that are extracted from an image. There is a wide range of feature extraction algorithms in Computer Vision. There are various features that can quantify a flower image but Color, Texture and Shape are the primary ones. To produce better results, Global Feature Descriptors are used that quantify an image globally. Three global feature decriptors used in this project are **Color Histogram** that quantifies color of the flower, **Hu Moments** that quantifies shape of the flower and **Haralick Texture** that quantifies texture of the flower. Figure 1 describes the feature extraction method.

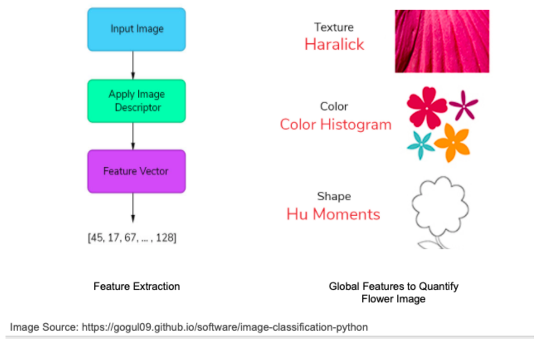

Figure 1: Feature Extraction

To extract Color Histogram features from the image, we used *cv2.calcHist()* function provided by *OpenCV*. The arguments it expects are the image, channels, mask, histSize (bins) and ranges for each channel [typically 0-256).

To extract Hu Moments features from the image, we used *cv2.HuMoments()* function provided by *OpenCV*. The argument to this function is the moments of the image *cv2.moments()* flatenned.

To extract Haralick Texture features from the image, we used *mahotas* library. The function used is *mahotas.features.haralick()*. Before doing that, we converted our color images into a gray scale image as haralick feature descriptor expect images to be gray scale.

2.3 Transfer Learning

The other model, which uses CNN, is based on transfer learning. In this model, we take the advantage of already existing trained neural network model for a particular task and then retrain it again for a similar task rather than working from scratch. It is not very efficient compared to a model built from scratch but it is very effective as it allows model training with significantly reduced training data and time by modifying already existing rich neural network model.
So for our transfer learning model we used an already optimal neural network *(*Inception from Google) which has been trained on 200,000 images for 1000 categories, and we retrained the neural network for our project using our flower- image data set. In this experiment, rather than training the whole neural network again, we just trained the penultimate layer of the neural network, also called the bottleneck layer, not because it is a bottleneck but because this layer has sufficient data compression information so that the final layer can easily identify the class of a given image.

3 Experiment

As the Methodology section gives a brief background about the different technical details of our model such as its architecture, in this section, we provide the details of our data set like its source, how the experiments were set up and frame work that was used to run the experiments. All the codes can be found in Github via `https://github.ncsu.edu/smallic/P15_ImageRecognition`

3.1 Data

Considering the availability of large scale image data sets, we narrowed our source data set of flower images to approximately 4000 images. This data set is obtained from Kaggle: `https://www.kaggle.com/alxmamaev/flowers-recognition` which consists of 4242 images of flowers divided into five classes: Chamomile, Tulip, Rose, Sunflower, Dandelion, where each class has about 800 images. The data collection is based on scraped data from Flicr, Google images and Yandex image. The image specifications are:

- Images are not high resolution
- Resolution is about 320x240 pixels
- Images are not reduced to a single size and have have different proportions.

The data is organized in the form of folder structure where each folder represents the class label and contains within it images belonging to that particular flower type.

3.2 Experiment Setup

In order to verify if the method dedicated for flower recognition can be successfully applied to recognize the type of flower, we perform experiment on the data set with the aim being to verify which of the model results in the best accuracy.

Keras Sequential Model

The first model we developed is a Sequential model using Tensorflow Keras. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. The core data structure of Keras is a model which is a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers.

Initially, all the required packages are imported including *Tensorflow*, *Keras*, *Matplotlib*, *cv2*, *random* and *sklearn.model_selection*. The images are then imported in the array data structure and re-scaled to 100X100 size to ensure uniformity and also to speed up the process. Simultaneously, while importing the images, labels are created for each images and stored in a separate array. Images and labels are shuffled together using *random.shuffle* method and the data along with its label is obtained using *zip*. To be able to fit a model on the training set, we convert it into Numpy array type. In order to perform cross validation, we use *StratifiedKFold* method which performs validation for 10 folds by splitting images and its labels using stratified sampling. So, in each fold the whole data set is split into training and validation data set and then the model is applied to it which is described as below.

Once the pre-processing is done, a Sequential model is developed which allows us to build a model layer by layer. Each layer has weights that correspond to the layer that follows it. *add* method is used to add layers to the model. In our case, we have added 3 layers: 1 *flatten* and 3 *dense*. The *Flatten* layer flattens the input and doesn't affect the batch size.*dense* is a regular densely connected Neural Network-layer. In a *Dense* layer, all nodes in the previous layer connect to the nodes in the current layer, which is shown in Figure 2. Dense implements the operation: $output = activation(dot(input, kernel))$ where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer. The activation function we have used are, *tanh*(Hyperbolic Tangent Activation Function), *relu* and *softmax* activation function. Once our model is ready, we need to compile it. Compiling the model takes three parameters: optimizer,loss and metrics. The optimizer controls the learning rate. We use *Adam* as our optimizer. The Adam optimizer adjusts the learning rate throughout the training. For our loss function, we use *sparse_categorical_crossentropy*. A *metric* is a function that is used to judge the performance of our model. Here, we use accuracy to compare our models. The final task is to train the model. We use *fit* function on our model with the following parameters: training images, training labels and number of epochs. The number of epochs is the number of times the model will cycle through the data. The more epochs we run, the more the model will improve, up to a certain point. After the saturation

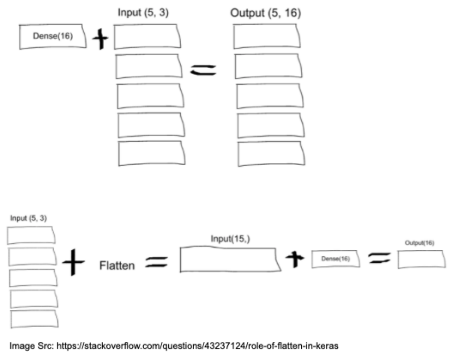

Figure 2: Dense Layers

point, the model will stop improving during each epoch. Finally, model accuracy can be obtained using *model.evaluate* function.

Ensemble Method: AdaBoost

Just like the Keras sequential model, at first all the required packages are imported including *sklearn.ensemble.AdaBoostClassifier*, *MinMaxScaler*, *random*, *pandas*, *cv2*, *os*, *h5py* and *train_test_split*. The images are then imported in the array data structure and re-scaled to 100X100 size to ensure uniformity and also to speed up the process. Simultaneously while importing the images, labels are created for each images and stored in a separate array. Images and labels are shuffled together using *zip* and *random.shuffle* methods. The output is then divided into train and test set using *train_test_split* method of *sklearn.model_selection* in 70-30 ratio. To be able to fit a model on the training set, we convert it into numpy array type.

Once the pre-processing of the data is done, feature extraction is performed to extract the features of the images. Color Histograms, Hu Moments and Haralick Texture are used to quantify the color, shape and texture of the images respectively. After extracting the features and concatenating it, we need save this data locally. As we have used different global features, one feature might dominate the other with respect to it's value. Thus, we normalized the features using scikit-learn's *MinMaxScaler()* function. After doing these two steps, we use *h5py* to save our features and labels locally in *.h5* file format.

After extracting, concatenating and saving global features from our training dataset, we trained our system. For creating our AdaBoost machine learning model, we took the help of scikit-learn and used *sklearn.ensemble.AdaBoostClassifier* to train our model with the locally stored features.Once AdaBoost model is trained with the training data, we test it on unseen flower images in the test data.*model.predict()* function is then used to predict the labels of the flower images in the test data set which are then compared with the original labels of the flower images in test data set. Finally, the model accuracy can be calculated using *model.accuracy_score* function comparing predicted labels with the actual labels.

Transfer learning Sequential Model

For this model, we used a library for Tensorflow called Tensorflow for Poets. This library contains Inception neural network from Google which was made for classifying images and it is installed on top of Tensorflow. Then, the images are randomly split into test and training data in eighty twenty ratio, eighty being training. Then, the model was trained using the training set and then the model was tested using testing data. As of now only individual pictures were tested against the model but we are in the process of making a script that would automate this process and give us the average accuracy all the testing data rather than posing individual images of flowers.The figure 3 was taken through tensorboard which is an addon software that helps in visualizing the neural network in tensorflow. In the image the node final_training_ops was the bottleneck that was retrained.

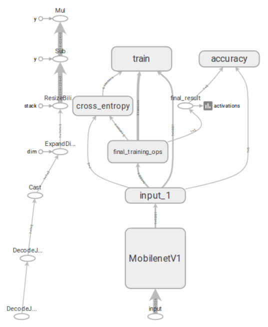

Figure 3: TensorFlow for Poets

4 Result

4.1 Keras Sequential Model

After running the algorithm using 10-fold cross validation each for 10 epochs using Sequential model, we found that the accuracy of the model was approximately around 23-24% consistently. The reason for achieving such low accuracy can be attributed to the fact that some neurons are dropped during the training phase while in evaluation phase it does not get dropped, thereby, differing in accuracy and also a lower one. Figure 4 shows both loss and accuracy of the model against each epoch.

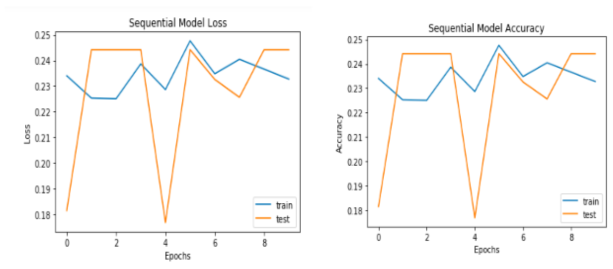

Figure 4: Accuracy

4.2 Ensemble Method: AdaBoost

Implementing the trained AdaBoost model on the features extracted from the images in the test dataset, we could calculate the model accuracy which was approximately 54%, much better than the Keras Sequential model. The accuracy can be improved further incorporating various techniques like considering more images for each class, using data augmentation, using Global features along with local features such as SIFT, SURF or DENSE, testing local features with BOVW technique, etc.

4.3 Transfer learning model

The testing for this was done with using 20% of the images. The accuracy for the model hovered around 80%. This proved to be an effective and fast way to create models with high accuracy.

The accuracy was found better for the transfer learning model and the graph in Figure 5 shows the comparison of all the three models used in this project.

5 Conclusion

There has been many breakthroughs in the field of Deep learning in image processing, text, speech and video recognition. Through this paper, it is our naive attempt in achieving the best accuracy, perform validation and compare the results between three different models, draw conclusions from

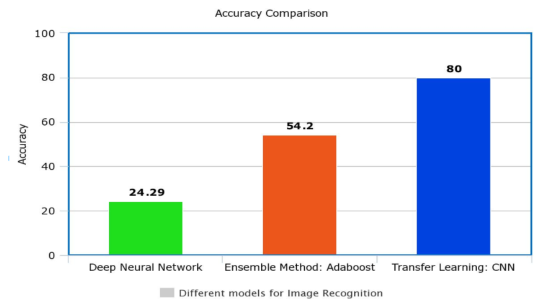

Figure 5: Accuracy Comparison

the results and analyze them for further improvements. We have introduced the application of Deep Neural Network model, Convolutional Neural Network model and Ensemble methods for recognizing different classes of flowers, which can be extended for any classes of images. Rather than just using different models Transfer Learning methodology was also used in this project. Although Transfer Learning is not that good compared to a model specifically created for this task but paired with a already trained good model in this case a convolutional neural network trained on two million images it gave the best results. Github: `https://github.ncsu.edu/smallic/P15_ImageRecognition`

References

[1] François Chollet et al. Keras. `https://keras.io`, 2015.

[2] Krzysztof J Cios and Inho Shin. Image recognition neural network: Irnn. *Neurocomputing*, 7(2):159–185, 1995.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. Ieee, 2009.

[4] Nitin R Gavai, Yashashree A Jakhade, Seema A Tribhuvan, and Rashmi Bhattad. Mobilenets for flower classification using tensorflow. In *Big Data, IoT and Data Science, 2017 International Conference on*, pages 154–158. IEEE, 2017.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[8] Takeshi Saitoh and Toyohisa Kaneko. Automatic recognition of wild flowers. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 507–510. IEEE, 2000.

[9] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[10] Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.