Visual Classifier for Invasive Plant Species

Elliott Jobson
Stanford University
emjobson@stanford.edu

Andres Hernandez
Stanford University
andresh@stanford.edu

Abstract

Invasive species monitoring by trained scientists can be a slow, costly process in terms of hired, skilled labor. In this project, we aim to examine effective ways to take on the provided Kaggle challenge of identifying hydrangea plants in images of forest. Herein, we evaluate and compare the performance of a VGG-16 CNN architecture, pre-trained on on the ImageNet dataset and subsequently trained on various augmented versions of the limited dataset provided; we describe our path to finding an ideal set of parameters and architectures for these models.

1. Introduction

Invasive plant species have several negative ecological impacts; they often compete for the same nutrients as the native species and can also be toxic to the local fauna. Invasive species cost an estimated 33 billion dollars per year and are "the second-greatest threat" to endangered wildlife after habitat destruction [1]. We chose to examine this problem because of its ecological, cross-industry impact as well as its reliance on a limited training dataset.

The ability to use smaller datasets to train a neural network particularly impacts situations where examples of the intended input are scarce or the collection of data is costly. By augmenting a training dataset in a targeted way, computer scientists can reduce the required time and cost to create tools for environmental scientists. Furthermore, data augmentation by adding noise can often have the impact of creating a model that is more robust to variations in orientation, lighting, and occlusion [2].

The input of our predictive model is a 3-channel color image of plants on forest floors. Our model uses transfer learning based on a VGG-16 Convolutional Neural Network then yields a probability that the foliage in the provided image contains hydrangea within.

Figure 1: A training image with hydrangea flowers.

2. Background and Related Work

Due to the scientific and social value of automated plant classification, the topic has already been well-explored. In the past, plant classification often relied upon traditional computer vision techniques used to make predictions on extracted features such as shape, color, or texture. However, with the recent success of CNNs in image classification, new plant classification methods have shifted toward a deep learning based approach [3].

The shift toward a deep learning based approach has marked an increase in the generalizability of classification methods – while feature extraction and engineering has often required domain-specific knowledge, such as markers for specific plant diseases, deep learning has shifted the burden of deciding which features to extract to the model [18].

Here, we will briefly review a few traditional methods used in plant classification, then cover a variety of deep learning approaches.

A few techniques have achieved high levels of success using traditional computer vision methods. For example, by using a histogram of oriented gradients to extract shape features from the image, Patil and Bodhe have classified plant diseases with 98.6% test accuracy [3]. However, this method requires clean images of the plants. Additionally, other teams have used features based on color and texture to recognize diseases on leaves [4, 5].

Once again, these methods often require clean images of the leaves, limiting the usefulness of the classifiers.

With enough data, convolutional neural networks offer hope in improving the generalizability of these classifiers. As we will discuss later, other CNN-based plant classification methods have faced similar problems that we faced in our project, including limitations on the volume and variety of training data.

A group led by Mohanty, Hughes, and Salathé sought to develop an accurate plant and disease classifier based on a large amount of training data and a transfer learning based approach [6]. While they were able to reach 99.35% test accuracy (without feature engineering) by repurposing AlexNet to their task, they report that their model's accuracy is reduced to 31.4% when applied to images taken under conditions different than that of the training data. Thus, in our own efforts, we hope to apply data augmentation such that our model will generalize well [2, 8]. Finally, the group reports that their model is limited to classifying single leaves facing upwards. Once again, they hope to make their model more generalizable by varying the training data to reflect more realistic image conditions. Ultimately, both these limitations point out that despite CNNs' relative success, challenges remain in training models that generalize well, especially based on data limitations.

Next, we explore the importance of applying transfer learning to our problem. According to Francois Chollet, transfer learning allows for the creation of powerful classifiers based on limited training data [2, 9, 13]. Since deep CNNs such as AlexNet or VGG16 have already been trained to extract increasingly-high level or abstract features as input data traverses the model, one can adapt such models for new tasks by training a few fully-connected layers at the end of the transferred model [10]. Moreover, VGG16 has been trained on similar data with the ImageNet dataset, as ImageNet contains many natural images.

Additionally, the Chollet suggests methods to avoid overfitting the model to a small dataset – in addition to regularization and data augmentation, one can limit the entropic capacity of a model by limiting the size of the model [9]. This reduces the number of features that the model can learn, forcing it to learn features more relevant to the task.

3. Methods

In this section, we will cover the algorithms that our various models utilize. First, we will briefly cover basic activation and loss functions, as well as the layers we used. Afterwards, we will explain the intuition behind transfer learning, as well as explain the importance of data augmentation as it relates to our task.

For all of our models, we use binary cross-entropy loss. This is simply a specific case of cross-entropy loss, in which we are classifying between only two categories. Given by the following equation, one can intuitively think of cross-entropy loss as a measure of distance between the truth, y, and a model's prediction, $\hat{y}$:

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i) \quad [11]$$

It is important to note that the prediction value, $\hat{y}$, must come from a normalized distribution of probability "scores."

Next, since our problem is binary classification, we used the sigmoid activation function to transform raw probability "scores" into our normalized probability distribution. As the output of our final layer was simply one number for each sample, the probability represents the likelihood that the species is invasive. Since the sigmoid activation function is an element-wise map of inputs to (0,1), it is well-suited for creating our probability distribution [11].

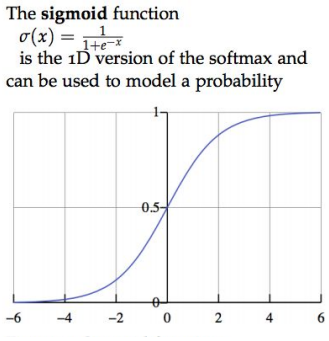

Figure 2: Our loss function of choice.

Before delving into our use of transfer learning and data augmentation, we will briefly cover a few basic tools used in all of our models. Due to its effectiveness in computer vision tasks and ability to speed the convergence

of Stochastic Gradient Descent (SGD), we used the ReLU activation function throughout our models:

$$f(x) = \max(0, x) \qquad [12]$$

In addition to its other benefits, the ReLU operation is cheap to compute, which helps with training large models on a lot of data [14].

Next, we used convolutional layers as the backbone of all of our models. At a high level, the convolutional layers involve sliding a filter across an image, computing the dot product at each location. The filters learn to activate (i.e. output a higher dot product) for certain image features – filters closer to the input of the network learn low-level features, while filters toward the end of the network learn more abstract, high-level features [15]. Since the convolutional filters may reduce the output size, we often use zero-padding to control the size of the output volumes.

Between our convolutional layers, we insert a number of max-pooling layers. This reduces the size of the layers' outputs, helping to prevent overfitting [15].

After our initial output layers, we flatten the filter outputs and apply a number of fully-connected layers to transform the convolutional output into our raw probability scores (which are then passed to the sigmoid activation and cross-entropy loss function). The fully-connected layers consist of alternating affine transformations ($h = Ax + b$) and nonlinear functions (which allow for nonlinear data separation).

Now that we have covered the building blocks of our models, we will cover their general structure:

Model	Structure	Training Epochs
baseline	6 conv layers, with zero-padding and max-pooling. Followed by 2 FC layers.	20
no_aug	2 FC layers trained on top of VGG16, without data augmentation.	40
no_aug_exp1	2 FC layers (with more trainable parameters than in *no_aug*), trained on top of VGG16, without data augmentation.	45
no_aug_exp2	2 FC layers (with more trainable parameters than in *no_aug_exp1*), trained on top of VGG16, without data augmentation.	50
no_aug_exp3	3 FC layers trained on top of VGG16, without data augmentation.	60
min_aug	2 FC layers trained on top of VGG16, with data augmentation (random horizontal and vertical translations, horizontal reflections, and rotations between 0° and 30°).	40
contrast_aug	Same as min_aug, but added random contrast adjustments to a random 20% of the inputs.	40
contrast_aug_exp	Same as *contrast_aug*, but with more trainable parameters in the final 2 FC layers.	45

Finally, it is clear that most of our models relied on transfer learning to achieve high performance on a small dataset. As we will explain in the evaluation section, most of our models achieved an AUC score of around .95, indicating that our model was able to effectively and accurately separate its distribution of outputs. However,

the baseline model did not come anywhere close to reaching this level of performance, largely because it trained its weights from scratch.

As implied by the name, transfer learning allows us to apply information learned by one model to another task. More specifically, one may use pre-trained deep convolutional models, such as VGG16 or AlexNet, to extract a range of feature-rich representations from the input data [16]. From there, one may train a few layers (often fully-connected layers) on top of the pre-trained model to determine which features are relevant to the task. Using this method, it is possible to build a classifier using limited training data.

In addition, we initially found it important to apply data augmentation to our dataset. Data augmentation consists of applying transformations to our training set, in order to increase the dataset size and variation. Although we reached a high-performance model without data augmentation, data augmentation has proven successful for image classification (especially when using a small dataset). By increasing the size of our dataset, data augmentation helps prevent overfitting. However, it is not fool-proof, since the augmented images will be highly-correlated. Our method of data augmentation consists of warping images in the data-space, rather than the feature-space [17]. In other words, our augmentations maintain label-data, rather than using other methods that artificially over-sample certain features. We hoped that adding random vertical and horizontal translations would remove the tendency of flowers to be located in the middle of the image. Moreover, based on a qualitative examination of our early models' errors, we realized that areas of high contrast may have led to false-positive classifications. Therefore, we later experiment with random contrast adjustments during training.

For our parameter updates, we settled upon using SGD with momentum and a small learning rate for gradient descent:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

[19].

When using transfer learning, it is important to avoid large updates on the pre-trained weights. Therefore, it is often necessary to "freeze" the updates of the pre-trained layers while training the final network layers, then fine-tune the entire model with small updates. Stochastic gradient descent with a small learning rate works well for fine-tuning, since it avoids large weight updates (compared to adaptive learning algorithms). For this reason, we train our models using SGD and a low learning rate.

4. Datasets and Features

We obtained our dataset from a Kaggle challenge [20]. It consists of 2294 labeled images, along with 1351 unlabeled images to be used for testing. We split the dataset into 2000 images to use for training and 294 for validation. Because we are using such a small validation set, we noticed a lot of noise in the graphs for validation accuracy and loss. In the training dataset, 63% of the images contained the invasive species.

Since the images were 1154 by 866 pixels, and because VGG16 requires 224 by 224 pixel 3-channel images, we resized all images to those specifications during preprocessing. An interesting feature of the dataset, which presented problems during training, was that the majority of images with hydrangeas in them had the flowers centered in the middle of the image. Similar to the problem faced by Mohanty et. al. [5], the general homogeneity of our dataset reduces the classification accuracy on images taken under different conditions. For example, since most of our images are taken with the flowers centered and with no patches of sunlight in the background, the model often fails to classify images that deviate from this pattern.

Figure 3: Typical "Invasive" Image

As we will discuss later, we applied data augmentation to increase the number of images that our models see, along with address some of the issues mentioned above.

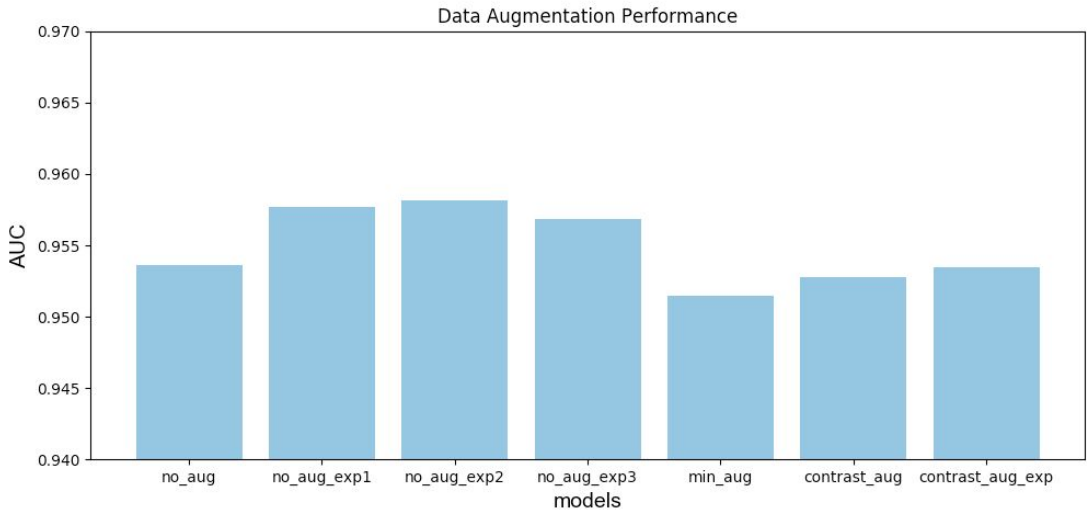

Figure 5: The above chart describes the AUC values received for different augmentation schemes.

5. Experiment

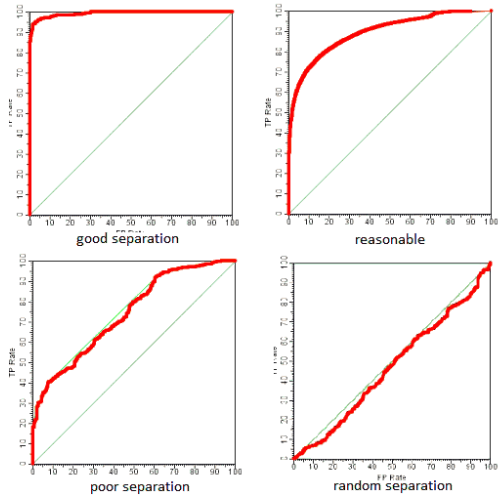

Figure 4: Four plots describing the appearance of an AUC curve and its interpretation in each case. The x-axis denotes the False Positive Rate. The y-axis denotes the True Positive Rate

The primary metric for determining the success of our model is the Area Under the Receiver Operating Characteristic curve, abbreviated "AUROC," and often shortened to "AUC" curve. In the context of our binary classification problem, the most clear interpretation of the AUC curve is a graph that describes the threshold for correct classification of an input image. The AUC considers all possible thresholds, and we use 0.5, meaning that the prediction values are true probabilities. As the threshold decreases, results will likely correctly classify all of the images that contain hydrangeas, however there will also be an increasing amount of false positives, or images without hydrangeas classified as having them. The opposite, more false negatives, will occur as the the threshold increases. For example, in Figure 5, we can see that an AUC curve with a value of 0.95 (i.e. 95% of the graph's area is under the ROC curve) would look approximately like the graph labeled "good separation."

Because of the structure of the Kaggle challenge competition and to prevent cheating, we are not provided the test label data, and so we are unable to compute a confusion matrix or graph the AUC curve from our results.

Figure 6: min_aug: 50.1% | aug_contrast: 49.7%

By examining the images that were incorrectly classified we can evaluate the different properties that our model "seeks" and emphasizes in its search for hydrangea flowers. For example, Figure 6 was classified by the

min_aug model as containing hydrangea flowers with probability 50.1%. It is likely that the model correlates the juxtaposition of green and white pixels with the presence of the hydrangea flower, in this case, either the sign or even possibly the patches of sky visible behind the trees. It is possible Figure 7 is also an example of the same behavior; we see several leaves reflecting sunlight in the foreground, in stark contrast to the surrounding green foliage.

Figure 7: min_aug: 55% | aug_contrast: 51.4%

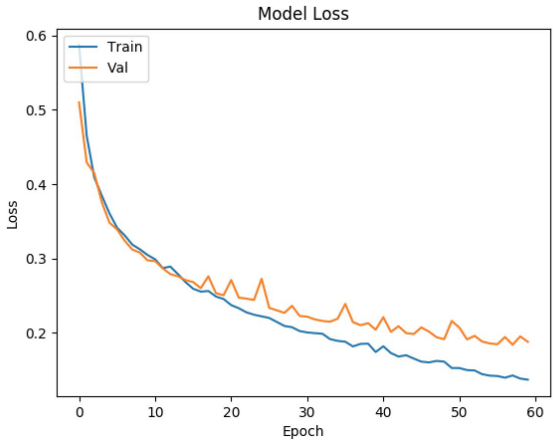

Figure 8: Training and Validation Loss for no_aug_exp3

These incorrect classifications are examples of why we felt that augmenting our training dataset with contrast-reduced samples would aid our prediction accuracy. We hypothesized that if the new model learned to place less emphasis on contrast, the false positives described above would be correctly classified. While the AUC only increased by a few thousandths, we saw that in edge cases like these, our contrast augmentation scheme was enough to push samples like in Figure 6 over the fence to being correctly classified. We also saw a reduction in probability of the incorrectly classified image in Figure 7, showing that the model is 4% less certain that the image in question contains hydrangeas.

In most of our models we found that the training and validation loss and accuracy tracked each other relatively well. This means that we should expand the entropic capacity of the model by increasing the number of trainable parameters. In other words, since the model performed nearly as well on the unseen

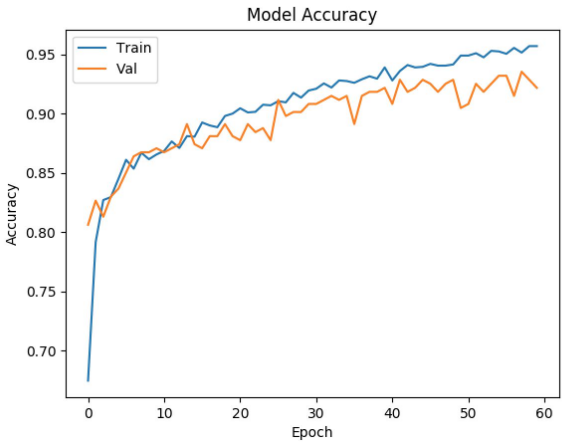

Figure 9: Training and Validation Accuracy for no_aug_exp3

validation data as it did on the training data, we observed that we could increase the number of trainable parameters in the model, allowing it to learn more specific features. To an extent, this was supported by our AUC values and accuracy statistics for the no_aug_exp1 and no_aug_exp2 models. Furthermore, we felt that by expanding the model we would also need to increase the number of training epochs. We stopped increasing the number of parameters in the model when we saw separation between the training and validation statistics but a reduced AUC value-- likely an overfit [7]. The batch size of 32 was relatively appropriate given the amount of noise visible in the loss and accuracy graphs for models. We also noticed that more noise was present in the graphs of the augmented datasets, likely because with each training epoch the model was seeing new images.

In the Methods section, we justify our use of SGD with a small learning rate (10^{-4}). Additionally, we can see in our loss graphs that the learning rate was appropriate for the model.

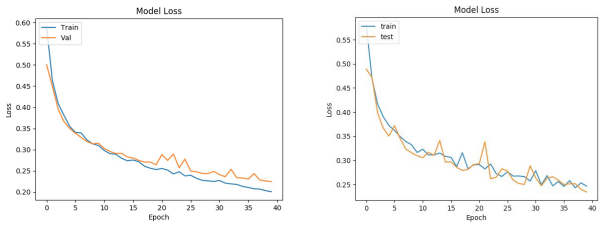

Figure 10: Training Loss for no_aug (left). Training loss for min_aug (right)

6. Conclusion

Overall, our highest-performing models tended to be the ones with more trainable parameters, since we started our experiments with a relatively modest number of trainable parameters in the final fully-connected layers. Though we observed that adding contrast adjustments improved the models that used data augmentation, our models without data augmentation actually outperformed our models with data augmentation. We hypothesize that the extra noise that the data-augmented models faced during training caused the final parameters of the data-augmented models to be sub-par. To address this going forward, we would increase the batch size for these models, as well as save the weights at every epoch.

Looking ahead, it is possible that increasing the probability of contrast augmentation from 1 in 5 would further decrease classification error based on localized, adjacent brightness disparity. However, future experiments with contrast-related data augmentation would have to take care to avoid adding so many contrast reduced images that the model no longer uses any contrast information to make a prediction.

We have considered a number of other possible extensions and paths of experimentation that we would have liked to explore, given more time. For example, a fair amount of information is lost simply by reducing the image resolution during preprocessing. This may have been particularly harmful in cases where the flowers occupied a small part of the image – images for which the model often failed. Though it would make it impossible to use transfer learning with VGG16 due to the new input size, it would be interesting to explore the use of a model that accepts higher resolution images.

As another way of addressing the misclassification of images with small, off-center flowers, we would like to explore the use of a sliding window classifier that considers the image at various locations. Alternatively, since the training images are of high resolution before preprocessing, we could have divided the images into large quadrants, running the test images on those. Though we would have to do extensive hyperparameter tuning to relate the outputs of the classifier at different locations to the overall classification, we believe that this method would effectively address our current models' weakness in classifying images with small flowers. Finally, given more time, we would like to explore test-time data augmentation – given a test image, we would collect a weighted vote on classifications of various transformations of the image. Once again, this may help solve the misclassification issues mentioned previously.

References

[1] *The Impact of Invasive Plants*. The California Invasive Plant Council. http://www.cal-ipc.org/ip/definitions/impact.php

[2] Lee, Fei Fei. Johnson, Justin. Yeung, Serena. *CS231N Lecture 7: Training Neural Networks.* Stanford University. 2017.

[3] Wäldchen, Jana. Mäder, Patrick. Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review. Arch Computat Methods Eng DOI 10.1007/s11831-016-9206-z. 24 November 2016.

[4] S. B. Patil and S. K. Bodhe. Leaf disease severity measurement using image processing. International Journal of Engineering and Technology, 2011.

[5] P. Chaudhary, A. K. Chaudhary, A. N. Cheeran, and S. Godara. Color transform based approach for disease spot detection on plant leaf. International Journal of Computer Science and Telecommunications, 2012.

[6] J. K. Patil and R. Kumar. Feature extraction of diseased leaf images. Journal of Signal and Image Processing, 2012.

[7] Sharada Prasanna Mohanty, David Hughes, and Marcel Salathé. Using Deep Learning for Image-Based Plant Disease Detection. Digital Epidemiology Lab, EPFL, Switzerland, 2016.

[8] Wong, Sebastien C. Gatt, Adam. Stamatescu, Victor. McDonnel, Mark D. *Understanding data augmentation for classification: when to warp?* Arxiv:1609.08764. 26 November 2016.

[9] Francois Chollet. Building powerful image classification models using very little data. The Keras Blog, 2016.

[10] Lee, Fei Fei. Johnson, Justin. Yeung, Serena. Course Notes 3. CS231N, 2017.

[11] Christopher Manning and Richard Socher. Course Notes 1. CS224N, 2017.

[12] Krizhevsky, Sutskever, and Hinton. ImageNet Classification with Deep Convolutional Neural Networks. University of Toronto, 2012.
[13] Martin, Glen. *Stanford researchers use dark of night and machine learning to shed light on global poverty.* Stanford News. 24 February 2016.
[14] Lee, Fei Fei. Johnson, Justin. Yeung, Serena. Neural Networks 1. CS231N, 2017.
[15] Lee, Fei Fei. Johnson, Justin. Yeung, Serena. Convolutional Neural Networks. CS231N, 2017.
[16] Ariadna Quattoni. Transfer Learning Algorithms for Image Classification. MIT Department of Electrical Engineering and Computer Science, 2009.
[17] Sebastien Wong, Adam Gatt, Victor Stamatescu, and Mark McDonnell. *Understanding data augmentation for classification: when to warp?*. DICTA 2016.
[18] Seeland M, Rzanny M, Alaqraa N, Wäldchen J, Mäder P (2017) Plant species classification using flower images—A comparative study of local feature representations. PLoS ONE 12(2): e0170629. https://doi.org/10.1371/journal.pone.0170629
[19] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.
[20] https://www.kaggle.com/c/invasive-species-monitoring
[21] Fujisan. *use Keras pre-trained VGG16 acc 98%. https://www.kaggle.com/fujisan/use-keras-pre-trained-vgg16-acc-98*