A Comparative Study of Learning Models for Invasive Hydrangea Presence Identification

Anna Dong
Biomedical Engineering
University of Waterloo
Waterloo, Canada
rh2liu@edu.uwaterloo.ca

Elaine Huang
Biomedical Engineering
University of Waterloo
Waterloo, Canada
e23huang@edu.uwaterloo.ca

Emily Lam
Biomedical Engineering
University of Waterloo
Waterloo, Canada
em2lam@edu.uwaterloo.ca

***Abstract*—Employing machine learning techniques to invasive species monitoring helps to solve ecological problems and promote environmental health. Various machine learning algorithms are explored in this paper to evaluate their effectiveness on classifying the presence of invasive hydrangea in images; namely SVM, VGG16, ResNet, and CapsNet. SVM, VGG16, ResNet, with the latter two being types of CNN, are established methodologies that are popularly used in image classification applications. On the other hand, CapsNet is a novel deep learning network with lots of potential to be explored and its feasibility has yet to be determined. Experimentation on the four approaches helped to identify the best performing model for the identification of invasive hydrangea in images of the Brazilian forest. The dataset consisted of training images labelled with the presence or absence of invasive hydrangea and unlabelled testing images. The CNN models demonstrated the highest prediction accuracies of all the different models, yielding accuracies of 98.579% and 98.893% for the VGG16 and ResNet50 architectures respectively.**

***Index Terms*—Support Vector Machine, Convolutional Neural Network, Capsule Network, Invasive Species, Plant Image Classification**

I. Introduction

An invasive species is any living organism that is not native to a specific geographic location and causes harm to the local environment, the economy, or to public health. Often, invasive species do not have natural predators, causing their population to increase rapidly, and thus threatening the survival of native species. In some parts of the world, up to 80% of endangered flora species are at risk as a result of pressures from the presence of non-native species [1]. The economic repercussions of invasive species can be enormous. For example, there are approximately 50,000 invasive species in the United States. These species are responsible for approximately $120 billion USD in damages and control costs on an annual basis [2]. Using crowd-sourcing techniques and mobile-phone cameras, machine learning algorithms can be used to monitor the proliferation and growth of invasive plants [3]. This is one of many potential applications for automatic plant identification, alongside ecological surveillance, studying biodiversity, determining the impact of climate change and ecological science popularization [4], [5].

In this paper, a variety of machine learning algorithms were evaluated to observe which was the best suited to identifying the presence of invasive hydrangea in images taken of the Brazilian national forest. The public dataset that was analyzed was taken from The Invasive Species Monitoring Kaggle challenge, and contains 2295 labelled training images and 1531 test images [1]. The accuracy of the classifications that were employed were verified by submitting the predicted results to the Kaggle competition. The best algorithm that was submitted to the competition resulted in an overall accuracy of 99.77% using a pretrained neural network called Inception-v3 on Imagenet with 5-fold cross-validation [1].

The machine learning methods that were investigated include a support vector machine (SVM), convolutional neural networks (CNNs) such as the Visual Geometry Group (VGG) and the residual neural network (ResNet), and the new Capsule Network (CapsNet). Comparing the SVM, CNNs, and CapsNet algorithms allows for an analysis of the past, present and future standard machine learning algorithms. CapsNet is a novel neural network introduced by Sabour and Hinton in 2017 [6]. It was claimed that CapsNet could theoretically outperform CNN in prediction accuracy. Thus far, CapsNet has not been applied to plant species identification as it is a fairly new technique and researchers are still exploring its potential. This paper investigates the performance of CapsNet for classifying images that may contain invasive hydrangeas in comparison to established image classification methodologies.

II. Related Works

In machine learning, plant classification is a supervised learning problem [5] that has been approached in 3 different ways: model-based, model-free, and deep learning.

Model-based approaches focus on feature detection, extraction, and encoding methods to create models for a specific application, such as looking at a plant flower. For example, Wu et al. [7], classified leaves by first binarizing the image to separate the background from the leaf, de-noising the image, detecting contours, and then extracting geometrical derivations of 12 shape features. This was evaluated on 32% species with an accuracy of 90%. However, the leaves differed significantly in their shapes. Other studies have looked at even more specific features such as leaf teeth shape [8]. Model-based approaches rely heavily on application-specific expert

[1] https://www.kaggle.com/c/invasive-species-monitoring/overview

knowledge, requiring a lot of time to move from an image to a feature vector [9].

Model-free approaches solve this problem by focusing on features that are not application specific, and instead detect characteristic interest points and describe them using generic algorithms. This lead to the development of deep learning approaches such as CNNs, which do not require explicit feature detection and extraction steps. So far, deep learning approaches have proven to be the most successful in plant classification. For comparison, Table I shows the results of different machine learning approaches on the Flavia dataset. This dataset consists of 1907 leaf images of 32 different species. Some of the images are scans, while others are photographs, but all have a plain background [5]. Note that the deep-learning approach has a much higher accuracy than the model-based and the model-free approaches.

TABLE I: Classification accuracy of different machine learning approaches on Flavia dataset.

Author	***Accuracy (%)***	***Approach***
[7]	90.3%	*Model-based:* 12 features extracted via principle component analysis (PCA) inputted into a probabilistic neural network (PNN)
[10]	95.9%	*Model-free:* Images described using Bag-of-Words (BOW) and inputted into a Support Vector Machine (SVM)
[4]	99.7%	*Deep-learning:* Residual learning model (ResNet26) pre-trained on ImageNet

The real challenge comes with imaging plants in their natural environments, as there may be degraded leaves or plants that are overlapping, partial, and compounded [4], [5]. Traditional classification models rely heavily on pre-processing to eliminate complex backgrounds and enhance desired features. This may not be feasible with datasets acquired in natural settings. However, many of the larger datasets are comprised of mobile-based plant images that differ in the area of the plant imaged, the area of focus, lighting, magnification, and were captured using different cameras at different times of the year [4], [11]–[13]. Good examples of datasets acquired in natural settings, and consequently having a high degree of variation are the Oxford Flower 17 and 102 datasets. These datasets contain images gathered from websites, along with supplementary images showing species in their natural habitat [14]. As shown in Table II, with these datasets, deep-learning has been observed to be the most successful.

TABLE II: Classification accuracy of different machine learning approaches on Oxford Flower 17 and 102.

Author	***Accuracy (%)***	***Approach***
[15]	91.8%	*Model-free:* SIFT, SURF, and HOG descriptors inputted to a SVM
[16]	90.2%	*Model-free:* BoW with higher-order Occurrence Pooling
[17]	96.6%	*Deep-learning:* VGG19 with SDE pooling
[18]	96.6%	*Deep-learning:* LG-CNN with VGG16

While many approaches opt for networks based on the VGG16 or VGG19 architectures, other architectures have also been used. Sun et al. [4] trained and tested various ResNets of varying layers (18, ,26, 34, and 50) on the BJFU100 dataset. This dataset contains images captured by mobile phones of 100 species of ornamental plants in the Beijing Forestry University campus. All images were re-sized to 224x224 pixels. In this study, 26 layers were found to be the optimal trade-off between model capacity and optimization difficulty. Mohanty et al. [19] re-purposed AlexNet to develop an accurate plant and disease classifier, reaching a 99.5% test accuracy. However, this was reduced to 31.4% when images taken under differing conditions from the training data were used. Many new and state-of-the-art architectures, such as ultra-deep (FractalNet [20]), densely connected (DenseNet [21]), capsule networks (CapsNet [6]) and smaller networks (SqueezeNet [22]), have not been extensively evaluated for plant species identification. Too et al. [23] compared different deep learning models for plant disease identification. CNNs with InceptionV4, VGG16, ResNet50, ResNet101, ResNet152, DenseNet121 architectures were pre-trained on the ImageNet dataset, then fine tuned and evaluated on the PlantVillage dataset; a dataset of 54,306 images containing 26 classes of diseased and healthy images of leaves from 14 plants. In this study, it was found that DenseNets were least prone to overfitting, required less parameters, had sensible computing times, and had the highest accuracy (99.75%) in comparison to the other architectures.

III. Description of Dataset

The dataset used was taken from a publically accessible Kaggle challenge and contained images of the Brazilian national forest. The dataset contains 2295 images labelled according to whether or not an invasive hydrangea is present in the photo, and these are used for training. In addition, the dataset contains 1531 unlabelled images for testing. Examples of images that do and do not contain an invasive hydrangea are shown in Figure 1. 1448 images within the training dataset contain invasive hydrangea which makes up 63.1% of all of the training images, whereas there were only 847 images without an invasive hydrangea present. Each raw image was 1154x866 pixels and was subsequently resized due to GPU processing restrictions. An algorithm's accuracy was assessed by submitting the probability for the target variable of the test images to the Kaggle challenge which reports the area under the ROC curve. This ensures that a consistent method of assessing the accuracy of the algorithms was employed. The correctly labelled test images were obtained from the organizers of the Kaggle challenge and was composed of 695 images that contained invasive hydrangea and 836 images that did not.

IV. Machine Learning Algorithms Investigated

A. Model-Based Approach: Support Vector Machine

To select a model-based approach for investigation, features had to first be extracted from the images. Many different feature extraction methods were explored such as Hu moments,

(a) (b)

Fig. 1: Images of the Brazilian Forest (a) with and (b) without invasive hydrangea present

Haralick textures, color histogram, scale invariant feature transform (SIFT), histograms of oriented gradients (HOG), etc. However, it was found that local binary pattern (LBP) features gave the best results. In fact, LBP features are often used in plant species identification methods [24]. This is because they are texture descriptors that are known to be unaffected by local grayscale variations and by variations in lighting, perspective distortions, image blur, and image zoom [24]–[26]. Hence, it is extremely useful for this dataset where consistent lighting could not be controlled.

Default configurations of common machine learning models were fit to the extracted LBP features. These methods included logistic regression (LR), Linear Discrimination Analysis (LDA), K Nearest Neighbours (KNN), Classification and Regression Trees (CART), Random Forest (RF), Gaussian Naive Bayes (NB), and SVM. The resulting validation accuracies reported in Table IX and Figure 7 located in Appendix A revealed that SVM model resulted in the most accurate predictions.

SVMs were first introduced by Boser et al in 1992 [11] and are now established classifiers. In this application, a SVM is trained using the extracted LBP features to find hyperplanes to separate the classes with a maximal margin in higher dimensional space [27]. This produces a model which is then used to predict the class of a test image given only the LBP features extracted from that test image [27].

B. Deep-Learning Approach: Convolutional Neural Networks (CNNs)

CNNs represent a class of deep-learning networks that have proven to be very effective in image recognition and classification. Notably, in 2012, Krizhevsky, Sutskever, and Hinton used a CNN to set the record in the ImageNet Large Scale Visual Recognition Challenge, winning with a 10.9% margin over second place [28], [29]. CNNs use filters or kernels, to detect features within an image. A filter moves over each part of an image to detect whether or not a specific feature is present. To quantify this, the filter carries out a convolution operation. This would comprise a layer of the network, and the output of such a layer could serve as an input to further layers. Other layers in the network can perform different functions such as ReLU (activation function) and pooling (to reduce spatial dimensions).

1) VGG16:: VGG16 is a CNN architecture devised by Simonyan and Zisserman [30]. With only 3x3 convolutional layers, it is modest but boasts great results, securing second place in the ILSVRC-2014 challenge and performing well in plant-specific classifications [17], [18], [31]. Since the Kaggle training dataset only has 2295 images, a transfer-learning approach with a VGG16 model trained on ImageNet was used. This is a technique employed by Dyrrman et al, [31] and makes sense, as lower layers of the network look for lower-level abstract features applicable to many datasets.

2) ResNet50:: As deep networks increase in layers, models are typically at risk of overfitting. ResNets [32] also allow for skipping of layers; a feature that speeds up learning and prevents overfitting. Also consisting of 3x3 filters, ResNet has a similar architecture to VGG, but is much deeper. Residual networks rely on residual units composed of convolution, pooling, and fully-connected layers. ResNet has been used successfully for plant recognition with fewer layers [4] and with a larger number of layers [23]. To provide greater contrast with the VGG16 architecture chosen, a ResNet50 model was trained (also using transfer learning). In Too et al. [23], 50 layers was shown to still be quite accurate, and required much less computational power and training time than a deeper network.

C. Novel Deep-Learning Approach: CapsNet

Capsule networks consist of capsules instead of neurons [6]. Capsules encapsulate important information about the state of the features they are detecting and outputs a vector, whereas neurons output a scalar [33]. Vectors can encode more information, but more specifically they can encode relational and relative information [6], [33]. Hence, when detected feature poses are changed, the output vector orientation changes accordingly [6], [33]. In other words, hierarchical pose relationships between image features can be preserved using vectors. Additionally, max pooling in CNN is replaced by a process called "routing-by-agreement" between capsules [6]. During this process, lower level capsules cooperate with higher level capsules [6]. This more effectively adds invariance and retains valuable spatial information [6].

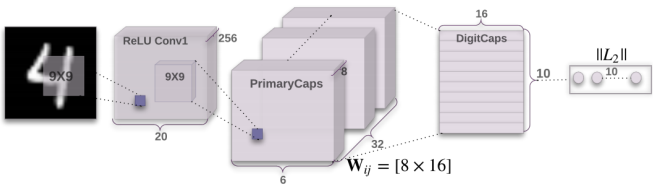


Fig. 2: CapsNet Architecture for MNIST Dataset [6]

Hinton's MNIST model architecture with 3 color channels, shown in Figure 2, was used as a baseline model, as implemented by Xifeng Guo [2] [6]. Since the hydrangea dataset contained significantly higher dimensions than the

[2]Source Code: https://github.com/XifengGuo/CapsNet-Keras

MNIST dataset, more capsule layers should be stacked to accommodate this. However, CapsNet requires heavy computational resources; therefore, the computational constraints limit the complexity of the model and the capsule layers were kept the same as the MNIST model. Furthermore, the image sizes had to be scaled down significantly to 75x75 pixels. Alternatively, additional convolution layers and max pooling layers could be employed for larger image dimensions. Note here that using max pooling should be avoided in CapsNet as it theoretically contradicts Hinton's ideal version of Capsule Networks; however, the computational limitations imposed by GPU resources constrained the dimensionality of the data that could be worked with.

V. Experiments and Results

The performance of each of the machine learning algorithms' ability to properly classify the images will be analyzed and compared in this section. All models were also evaluated using k-fold cross-validation. Through trial and error, a k-fold value of 10 yielded the best results when the SVM experiments were performed. Thus, a k-fold value of 10 was maintained for all algorithms so that a more consistent comparison between the performance of each of the algorithms could be conducted. All algorithms were implemented using Python in Google's Colaboratory which contains a Tesla K80 GPU and is limited to 12GB of RAM.

A. Support Vector Machine

The training image dataset was first resized to either 110x110, 224x224, 400x400, or 600x600 pixels. Larger image sizes could not be processed because there was not enough RAM. Normalized LBP features were then extracted from all images and the code used to extract the 26 LBP features from the images is shown in Appendix B. A SVM model was trained using stratified k-fold (k=10) on the extracted features and the resulting model was used to predict the labels for the test dataset for each set of image sizes. The optimal SVM model was found through trial and error and operates using the radial basis function (RBF) kernel with a γ value of 0.08 and a penalty parameter value of 3. Each set of predicted values was then submitted to the Kaggle challenge which revealed the accuracy of each set of image sizes as shown in Table III. The best accuracy of 84.51% came from images that were resized to 400x400 pixels which took 5.03 minutes to extract the LBP features and to train and predict the test values.

TABLE III: Comparison of Image Sizes and Resulting SVM Performance

Image Sizes	***Accuracy (%)***
110x110	78.16%
224x224	82.94%
400x400	84.51%
600x600	81.24%

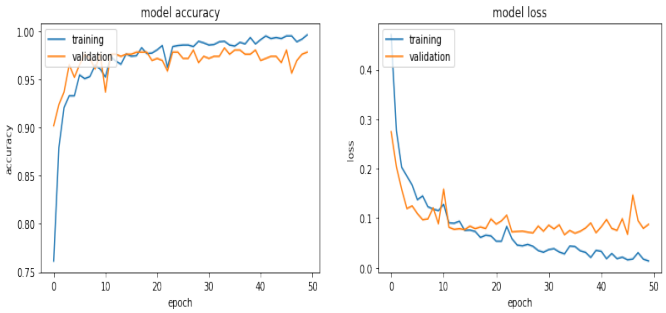
B. CNN

Given that the dataset was small, image transformations were utilized to increase the amount of training data. Permissible transformations included rotation, vertical and horizontal shifts, and the flipping the image across the vertical axis [31]. Given that the images were of plants, these were found to be feasible. The original size of the images was quite large, and the neural networks prefer square images, so the images were re-sized to 224x224 pixels and were padded at both the top and the bottom with black. 224x224 pixels was also chosen because it is a default size that is widely used in literature [4], [17], [18], [23], [31], and because using larger images with the ResNet50 was found to cause the GPU to run out of RAM. A smaller image size was not used, at risk of reducing the image quality too much, making it difficult to see the hydrangeas in the images, as some were already quite small. For equal comparison, both networks were trained with images at 224x224 pixels. All images were normalized to values between 0 and 1 by dividing by 255.

1) VGG16: VGG16 was implemented using a transfer learning model built into Keras. Network and training parameters can be found in Table IV. To determine the optimal number of epochs for convergence, the training data was split 70/30 for training and validation. The model was trained for 50 epochs. Figure 3 shows the losses and accuracy of the model up to 50 epochs. Note that the optimal number of epochs seems to be between 15 and 25. Hence, the model was re-run at 15, 20, and 25 epochs, yielding accuracies of 98.255%, 98.541%, and 98.481% respectively. Consequently, 20 epochs were chosen to run with the k-fold cross validation. Taking 100 minutes to train, this resulted in an average accuracy of 98.579% or ROC of 0.98579.

TABLE IV: VGG16 Parameter Values

Parameters	**Values**
Batch Size	32
Learning Rate	0.0001
Optimizer	Stochastic Gradient Descent
Momentum	0.9
Learning Rate Decay	0
Loss Function	Binary cross-entropy



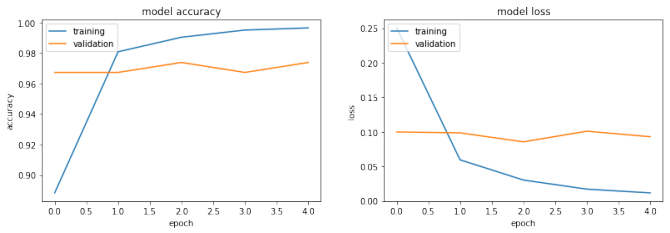
(a) (b)

Fig. 3: VGG16 up to 50 epochs (a) accuracy and (b) losses

2) ResNet50: ResNet50 was implemented via transfer learning using the built-in ResNet50 Keras model. Network and training parameters can be found in Table V. The network was first trained using 5 epochs, and a batch size of 32. Using just a 70/30 split on the test data, this took about 1 hour to run and achieved an accuracy of 0.98741. Looking at model accuracy and losses over time, shown in Figure 4, it seemed that 3 epochs could be used during the stratified k-fold cross validation to reduce training time. This was trained using the same 70/30 data split, and achieved a slightly lower accuracy of 0.98710. Lastly, the ResNet50 was trained using stratified k-fold (k=10) cross-validation. The resulting accuracy was 98.892% or ROC of 0.98892. Note that with the k-fold cross-validation, the training took approximately 10.5 hours. This is much longer than the other techniques investigated.

TABLE V: ResNet50 Parameter Values

Parameters	Values
Batch Size	32
Learning Rate	0.0001
Optimizer	Stochastic Gradient Descent
Momentum	0.9
Learning Rate Decay	0



(a) (b)

Fig. 4: ResNet50 up to 5 epochs (a) accuracy and (b) losses

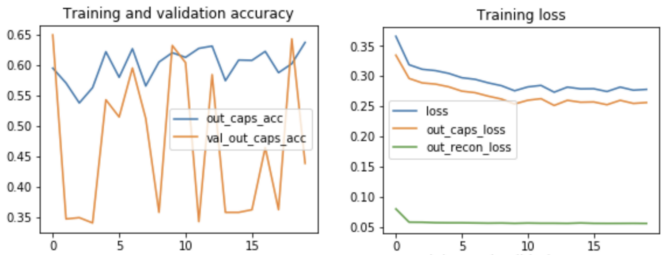
C. CapsNet

Initially, images were resized to 75x75 pixels and CapsNet was trained over 20 epochs with parameters shown in Table VI.

TABLE VI: CapsNet Parameter Values

Parameters	Values
Batch Size	10
Learning Rate	0.0001
Learning Rate Decay	0.9
Decoder Loss Coefficient	0.392
Shift Fraction	0.2

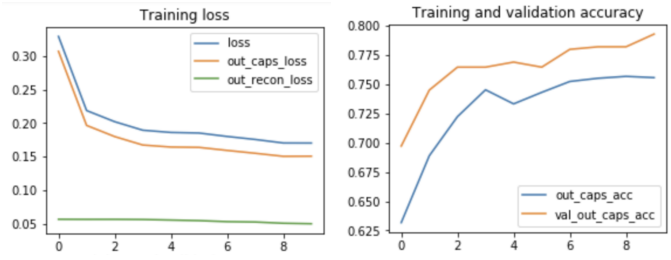
The resulting training and validation losses and accuracies are plotted in Figure 5. This yielded a prediction with an accuracy of 57.80% on the testing data. The training and validation accuracies shown in Figure 5 suggest that 10 epochs would likely result in a higher prediction accuracy, so the same resized dataset was trained over 10 epochs which demonstrated an accuracy of 72.34%. These results are noticeably inferior to the results of the other methods, which could be attributed to the substantial scaling down of the images. Furthermore, there is no obvious trend observed on the accuracy plot, suggesting poor effectiveness of the current implementation.



(a) (b)

Fig. 5: CapsNet up to 20 epochs (a) accuracy and (b) losses with 75x75 image size

Alternatively, bigger image sizes can be used with max pooling to fit the computational constraints. Thus, two more convolution layers and a max pooling layer were added prior to the primary capsule layer to reduce the number of parameters. The images were resized to 110x110 pixels and trained over 10 epochs on the modified CapsNet with the same parameters used previously. This resulted in a prediction accuracy of 86.2%. Since max pooling of higher dimensions resulted in a more accurate prediction, this setup was selected to be trained with k-fold cross validation where k=10, yielding an accuracy of 93.75%. The training and validation losses and accuracies are plotted in Figure 6. A summary of the results from these experiments is shown in Table VII.



(a) (b)

Fig. 6: CapsNet up to 10 epochs (a) accuracy and (b) losses with 110x110 image size with max pooling

TABLE VII: CapsNet Result

Experiment	Image Sizes	Epochs	Test Acc
CapsNet	75x75	20	57.80%
CapsNet	75x75	10	72.34%
CapsNet + Pooling	110x110	10	86.22%
CapsNet + Pooling + k-fold	110x110	10	93.75%

D. Comparing the Methods

To compare the different machine learning algorithms, Table VIII was created to summarize the versions that resulted in the greatest accuracy for each type of algorithm.

TABLE VIII: Summary of Machine Learning Algorithm Results

Algorithm	Image Sizes	Time (min)	Test Acc
SVM	400x400	5.03	84.51%
VGG16	224x224	100	98.58%
ResNet	224x224	630	98.89%
CapsNet	110x110	180	93.75%

Table VIII shows that the best performing algorithm was VGG16 because it resulted in the second greatest accuracy of 98.58% given that it only took 100 minutes to train the model. The ResNet algorithm had the highest accuracy of 98.89% but took 6.3 times longer to execute in comparison to the VGG16 algorithm. This marginal accuracy increase is negligible in comparison to the massive computation resources required to power ResNet. With this consideration, VGG16 is concluded to be the best performing algorithm for this application.

For this dataset, it can be expected that CNN outperforms SVM since SVM performs better on smaller datasets. Conversely, although CNN demonstrated superior performance than CapsNet, CapsNet's full potential was not explored. The images had to be resized to smaller dimensions and capsule layers were limited due to computational constraints. Given these limiting conditions, CapsNet's 93.75% accuracy shows promising indication of its effectiveness on plant image classification.

VI. Summary and Conclusion

In summary, a novel neural network which was first introduced in 2017 called CapsNet was applied to the identification of invasive hydrangea in images of the Brazilian forest. Three other established methods, SVM, VGG16, and ResNet50 were also applied to the same dataset to ascertain which of the four methods would perform the best at discerning between images that did and did not contain invasive hydrangea.

Though CapsNet's accuracy of 93.75% is promising, due to computational constraints, the best performing algorithm for this application was the VGG16 architecture. The VGG16 architecture was the most balanced as it resulted in an accuracy of 98.58% and only took 100 minutes to train. Meanwhile, the SVM model took the least amount of time to execute, but resulted in the lowest accuracy of 84.51% and the ResNet model, which took 630 minutes, had the highest accuracy of 98.89%.

Unfortunately, the ResNet model was unable to outperform the Inception-v3 algorithm that was submitted to the Kaggle challenge and resulted in the highest accuracy of 99.77%. One of the main reasons for this is because the individual who ran the algorithm used a GPU that was able to process large images that were up to 800x800 pixels [3]. This submission also took a weighted average of all predictions that were made at various image sizes which further increased the prediction accuracy [3].

In the future, a more sophisticated GPU would allow for further investigation into the potential of CapsNet for invasive plant species identification. A more powerful GPU would also allow for the processing of larger images which can help to correctly identify images with invasive hydrangea in cases where the hydrangea are very small in relation to the rest of the raw image.

References

[1] S. Armstrong, "Rare plants protect Cape's water supplies," *New Scientist*, vol. 145, no. 1964, p. 8, 1995.

[2] D. Pimentel, R. Zuniga, and D. Morrison, "Update on the environmental and economic costs associated with alien-invasive species in the United States," *Ecological Economics*, vol. 52, no. 3, pp. 273–288, 2 2005. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921800904003027

[3] H. Goëau, P. Bonnet, and A. Joly, "Plant identification in an open-world (LifeCLEF 2016)," Tech. Rep. [Online]. Available: https://pdfs.semanticscholar.org/ff9d/27db44186f206b3996eed4d4fc86fc9882e3.pdf

[4] Y. Sun, Y. Liu, G. Wang, and H. Zhang, "Deep Learning for Plant Identification in Natural Environment," *Computational Intelligence and Neuroscience*, vol. 2017, pp. 1–6, 5 2017. [Online]. Available: https://www.hindawi.com/journals/cin/2017/7361042/

[5] J. Wäldchen, M. Rzanny, M. Seeland, and P. Mäder, "Automated plant species identificationTrends and future directions," *PLOS Computational Biology*, vol. 14, no. 4, p. e1005993, 4 2018. [Online]. Available: https://dx.plos.org/10.1371/journal.pcbi.1005993

[6] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic Routing Between Capsules," 10 2017. [Online]. Available: https://arxiv.org/abs/1710.09829

[7] S. G. Wu, F. S. Bao, E. Y. Xu, Y.-X. Wang, Y.-F. Chang, and Q.-L. Xiang, "A Leaf Recognition Algorithm for Plant Classification Using Probabilistic Neural Network," in *2007 IEEE International Symposium on Signal Processing and Information Technology*. IEEE, 12 2007, pp. 11–16. [Online]. Available: http://ieeexplore.ieee.org/document/4458016/

[8] T. Jin, X. Hou, P. Li, and F. Zhou, "A Novel Method of Automatic Plant Species Identification Using Sparse Representation of Leaf Tooth Features," *PLOS ONE*, vol. 10, no. 10, p. e0139482, 10 2015. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0139482

[9] P. Barré, B. C. Stöver, K. F. Müller, and V. Steinhage, "LeafNet: A computer vision system for automatic plant species identification," *Ecological Informatics*, vol. 40, pp. 50–56, 7 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574954116302515

[10] Q.-K. Nguyen, T.-L. Le, and N.-H. Pham, "Leaf based plant identification system for Android using SURF features in combination with Bag of Words model and supervised learning," in *2013 International Conference on Advanced Technologies for Communications (ATC 2013)*. IEEE, 10 2013, pp. 404–407. [Online]. Available: http://ieeexplore.ieee.org/document/6698145/

[11] S. J. Kho, S. Manickam, S. Malek, M. Mosleh, . Sarinder, and K. Dhillon, "Automated plant identification using artificial neural network and support vector machine," *Frontiers in Life Sciences*, vol. 10, no. 1, pp. 98–107, 2018. [Online]. Available: https://www.tandfonline.com/action/journalInformation?journalCode=tfls20

[12] J. Wäldchen and . P. Mäder, "Plant Species Identification Using Computer Vision Techniques: A Systematic Literature Review," *Archives of Computational Methods in Engineering*, vol. 25, no. 2, pp. 507–543, 2018. [Online]. Available: https://doi.org/10.1007/s11831-016-9206-z

[3] https://www.kaggle.com/c/invasive-species-monitoring/discussion/38165#latest-21527

[13] A. Joly, H. Goëau, P. Bonnet, V. Bakić, J. Barbe, S. Selmi, I. Yahiaoui, J. Carré, E. Mouysset, J.-F. Molino, N. Boujemaa, and D. Barthélémy, "Interactive plant identification based on social image data," *Ecological Informatics*, vol. 23, pp. 22–34, 9 2014. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S157495411300071X

[14] M.-E. Nilsback and A. Zisserman, "Automated Flower Classification over a Large Number of Classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 12 2008, pp. 722–729. [Online]. Available: http://ieeexplore.ieee.org/document/4756141/

[15] M. Seeland, M. Rzanny, N. Alaqraa, J. Wäldchen, and P. Mäder, "Plant species classification using flower imagesA comparative study of local feature representations," *PLOS ONE*, vol. 12, no. 2, p. e0170629, 2 2017. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/28234999http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC5325198http://dx.plos.org/10.1371/journal.pone.0170629

[16] P. Koniusz, F. Yan, P.-H. Gosselin, and K. Mikolajczyk, "Higher-Order Occurrence Pooling for Bags-of-Words: Visual Concept Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 2, pp. 313–326, 2 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7439823/

[17] G.-S. Xie, X.-Y. Zhang, S. Yan, and C.-L. Liu, "SDE: A Novel Selective, Discriminative and Equalizing Feature Representation for Visual Recognition," *International Journal of Computer Vision*, vol. 124, no. 2, pp. 145–168, 9 2017. [Online]. Available: http://link.springer.com/10.1007/s11263-017-1007-9

[18] G.-S. Xie, X.-Y. Zhang, W. Yang, M. Xu, S. Yan, and C.-L. Liu, "LG-CNN: From local parts to global discrimination for fine-grained recognition," *Pattern Recognition*, vol. 71, pp. 118–131, 11 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317302248

[19] S. P. Mohanty, D. P. Hughes, and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection," *Frontiers in Plant Science*, vol. 7, p. 1419, 9 2016. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fpls.2016.01419/full

[20] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-Deep Neural Networks without Residuals," 5 2016. [Online]. Available: https://arxiv.org/abs/1605.07648

[21] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," 8 2016. [Online]. Available: https://arxiv.org/abs/1608.06993

[22] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and," 2 2016. [Online]. Available: https://arxiv.org/abs/1602.07360

[23] E. C. Too, L. Yujian, S. Njuki, and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification," *Computers and Electronics in Agriculture*, 3 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169917313303

[24] C. H. Arun, W. R. Sam Emmanuel, and D. Christopher Durairaj, "Texture Feature Extraction for Identification of Medicinal Plants and Comparison of Different Classifiers," Tech. Rep. 12, 2013. [Online]. Available: https://pdfs.semanticscholar.org/e592/5c0fa391484778ba3213b18606fffc4dd7fb.pdf

[25] M. Eisa, A. Elgamal, R. Ghoneim, and A. Bahey, "Local Binary Patterns as Texture Descriptors for User Attitude Recognition," Tech. Rep. 6, 2010. [Online]. Available: https://pdfs.semanticscholar.org/1408/70347fda6990622a07236338bb2c3a0d53f8.pdf

[26] M. Šulc and J. Matas, "Fine-grained recognition of plants from images," vol. 13, p. 115, 2017. [Online]. Available: https://doi.org/10.1186/s13007-017-0265-4

[27] Q.-K. Man, C.-H. Zheng, X.-F. Wang, and F.-Y. Lin, "Recognition of Plant Leaves Using Support Vector Machine," in *Recognition of Plant Leaves Using Support Vector Machine. In: Huang DS., WunschD.C., Jo KH. (eds) Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 192–199. [Online]. Available: http://link.springer.com/10.1007/978-3-540-85930-7_26

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 5 2017. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3098997.3065386

[29] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 9 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[31] M. Dyrmann, H. Karstoft, and H. S. Midtiby, "Plant species classification using deep convolutional neural network," *Biosystems Engineering*, vol. 151, pp. 72–80, 11 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1537511016301465

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," 12 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[33] G. E. Hinton, A. Krizhevsky, and S. D. Wang, "Transforming Auto-encoders," Tech. Rep. [Online]. Available: https://www.cs.toronto.edu/~fritz/absps/transauto6.pdf

APPENDIX A

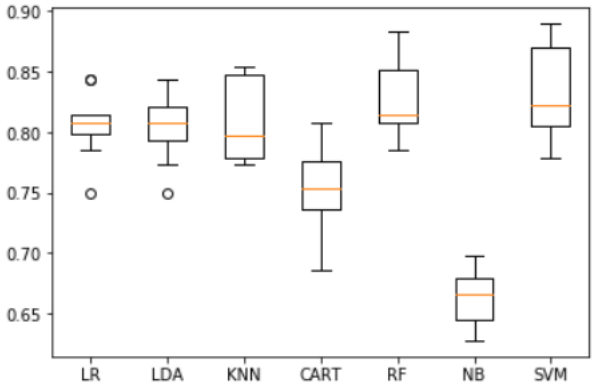


Fig. 7: Box Plot Comparison of Linear Regression (LR), Linear Discrimination Analysis (LDA), K Nearest Neighours (KNN), Classification and Regression Tree (CART), Random Forest (RF), Gaussian Naive Bayes (NB) and Support Vector Machine (SVM) Accuracies and Standard Deviations.

TABLE IX: Comparison of Common Machine Learning Algorithm Accuracies

Machine Learning Algorithm	*Accuracy (%)*	*Standard Deviation*
LR	80.65%	2.56
LDA	80.36%	2.62
KNN	80.94%	3.41
CART	75.54%	3.37
RF	82.69%	3.30
NB	66.24%	2.13
SVM	83.50%	3.67

APPENDIX B

SVM Example Code

```
from skimage.feature import local_binary_pattern
from scipy.stats import itemfreq

LBP_RADIUS = 3
LBP_NB_POINTS = 8 * LBP_RADIUS
LBP_METHOD = 'uniform'
```

```
def get_lbp(img_gray_scale, radius=3):
    return local_binary_pattern(img_gray_scale,
                                8 * radius,
                                radius,
                                LBP_METHOD)

def get_hist(lst):
    x = itemfreq(lst)
    hist = x[:, 1]/sum(x[:, 1])
    return hist

def get_lbp_hist(img, r=3):
    g = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    return get_hist(get_lbp(g, radius=r).reshape(-1))

def load_lbp_features(img, training=True):
    descriptor = list(get_lbp_hist(img, 3))
    return descriptor

##extracting features from training images
lbp_features = []
for q in range(len(x)):
  fv_lbp_feat  = load_lbp_features(x[q])
  lbp_features.append(fv_lbp_feat)

rescaled_features_train = np.zeros((2295, 26))
for i in range(len(lbp_features)):
  rescaled_features_train[i] = \
  np.array(lbp_features[i])
rescaled_features_train = (rescaled_features_train
       - rescaled_features_train.mean(axis=0))/\
rescaled_features_train.std(axis=0)

##extracting features from test images
lbp_test_features = []
for q in range(len(test_images)):
  fv_lbp_feat  = load_lbp_features(test_images[q])
  lbp_test_features.append(fv_lbp_feat)

rescaled_features_test = np.zeros((1531, 26))
for i in range(len(lbp_test_features)):
  rescaled_features_test[i] = \
  np.array(lbp_test_features[i])
rescaled_features_test = (rescaled_features_test
       - rescaled_features_test.mean(axis=0)) /\
rescaled_features_test.std(axis=0)
```

VGG16 Example Code

```
# Function to create model
def build_VGG16(input_tensor):
  base_model = applications.VGG16(weights='imagenet'\
    , include_top=False, input_shape=input_tensor)
  add_model = Sequential()
  add_model.add(Flatten(input_shape= \
    base_model.output_shape[1:]))
  add_model.add(Dense(256, activation='relu'))
  add_model.add(Dense(1, activation='sigmoid'))

  model = Model(inputs=base_model.input, outputs= \
    add_model(base_model.output))
  model.compile(loss='binary_crossentropy', \
    optimizer=optimizers.SGD(lr=1e-4, momentum=0.9)\
    , metrics=['accuracy'])
  #model.summary()
  return model

img_rows, img_cols, img_channel = 224, 224, 3
batch_size = 32
epochs = 20
prediction_total = []
prediction_mean = []

#Run k-fold (k=10) cross-validation

input_tensor_shape=(img_rows, img_cols, img_channel)

# Loop through the indices the split() method returns
for index, (train_indices, val_indices) in \
  enumerate(skf.split(x, y)):

    print("Training on fold " + str(index+1) + \
      "/10...")

    # Generate batches from indices
    x_train, x_test = x[train_indices], \
      x[val_indices]
    y_train, y_test = y[train_indices], \
      y[val_indices]

    # Clear model, and create it
    model = None
    model = build_VGG16(input_tensor_shape)

    # Normalize training and validation images
    x_train = x_train.astype('float32')
    x_test = x_test.astype('float32')
    x_train /= 255
    x_test /= 255

    #Image transforms
    train_datagen = ImageDataGenerator(
        rotation_range=30,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True)
    train_datagen.fit(x_train)

    #Train model
    history = model.fit_generator(
      train_datagen.flow(x_train, y_train, \
        batch_size=batch_size),
      steps_per_epoch=x_train.shape[0] // batch_size,
      epochs=epochs,
      validation_data=(x_test, y_test),
      callbacks=[ModelCheckpoint( \
        'VGG16-transferlearning.model', monitor= \
        'val_acc', save_best_only=True)]
    )

    accuracy_history = history.history['acc']
    val_accuracy_history = history.history['val_acc']

    # History for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss - k = ' + str(index) )
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['training', 'validation'], \
      loc='upper left')
    plt.show()

    # History for accuracy
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('model accuracy   - k = ' + str(index))
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['training', 'validation'], \
      loc='upper left')
    plt.show()

    #Run predictions to average later
    predictions = model.predict(test_images)
```

```
    if index == 0:
      prediction_total = predictions
    else:
      prediction_total += predictions

    print("Last training accuracy: " + str( \
      accuracy_history[-1]) + \
      ", last validation accuracy: " + str( \
      val_accuracy_history[-1]))

prediction_mean = prediction_total / kfold_splits
```

CapsNet Example Code

```
for index, (train_indices, val_indices) in \
enumerate(skf.split(x,y)):

  /* data batches generation and normalization
  implementation not shown, refer to VGG Example
  Code for implementation details */

  model = None
  model = CapsNet(input_shape=x_train.shape[1:],
  n_class=len(np.unique(np.argmax(y_train, 1))),
  num_routing=3)

  #args contain all parameter values
  train(model=model,
  data=((x_train, y_train),(x_test, y_test)),
  args=args)

  /*prediction implementation not shown,
  refer to VGG code */


def CapsNet(input_shape, n_class, num_routing):
    from keras import backend as K
    K.clear_session()

    x = layers.Input(shape=input_shape)

    # Layer 1-3: Just a conventional Conv2D layer
    conv1 = layers.Conv2D(filters=256,
    kernel_size=3, strides=1, padding='valid',
    activation='relu', name='conv1')(x)
    conv2 = layers.Conv2D(filters=256,
    kernel_size=3, strides=1, padding='valid',
    activation='relu', name='conv2')(conv1)
    conv3 = layers.Conv2D(filters=256,
    kernel_size=3, strides=1, padding='valid',
    activation='relu', name='conv3')(conv2)
    # Layer 4: Max pooling reduce parameters
    pooled = layers.MaxPool2D((2,2))(conv3)

    # Layer 5: Conv2D layer with squash activation,
    # then reshape to [None,num_capsule,dim_vector]
    primarycaps = PrimaryCap(pooled, dim_vector=8,
    n_channels=32, kernel_size=3, strides=2,
    padding='valid')

    # Layer 6: Capsule layer. Routing algorithm
    digitcaps = CapsuleLayer(num_capsule=n_class,
    dim_vector=16, num_routing=num_routing,
    name='digitcaps')(primarycaps)

    # Layer 7: This is an auxiliary layer to
    # replace each capsule with its length.
    out_caps = Length(name='out_caps')(digitcaps)

    # Decoder network.
    y = layers.Input(shape=(n_class,))
    masked = Mask()([digitcaps, y])
    x_recon = layers.Dense(512,
    activation='relu')(masked)

    x_recon = layers.Dense(1024,
    activation='relu')(x_recon)
    x_recon = layers.Dense(np.prod(input_shape),
    activation='sigmoid')(x_recon)
    x_recon = layers.Reshape(target_shape=input_shape,
    name='out_recon')(x_recon)

    # two-input-two-output keras Model
    return models.Model([x, y], [out_caps, x_recon])
```