VA PLANT IMAGE CLASSIFICATION

A PREPRINT

Huilin Chang
School of Data Science
University of Virginia
Charlottesville, VA 22904
hc5hq@virginia.edu

Yihnew Eshetu
School of Data Science
University of Virginia
Charlottesville, VA 22904
yte9pc@virginia.edu

Celeste Lemrow
School of Data Science
University of Virginia
Charlottesville, VA 22904
ctl7t@virginia.edu

May 11, 2021

Abstract

We develop and compare performance between several transfer learning image classification models focused on native and invasive species of aquatic plants in Virginia. We aim to accurately identify hydrilla plants, an entrenched invasive species in Virginia waterways, compared with other types of submergent aquatic plants that are not invasive species but have similar appearance, to help identify which plants should be removed from a body of water as part of a conservation effort. We focus on transfer learning models, given their greater accuracy for image classification, but also experiment with original architecture for exploration and comparison purposes.

Motivation

Virginia has more than 3,000 square miles of water [16], including lakes, rivers, ponds, streams, and the Chesapeake Bay. The quality, health, and environmental ecosystems of those bodies of water are under threat from invasive aquatic plant species, which can choke out native species and prevent them from thriving, spur harmful changes in the water's chemistry, negatively impact the development of fish and aquatic organism populations, and make the water less hospitable for recreational use [4]. One of the most common invasive aquatic species in Virginia is hydrilla (*hydrilla verticillata*) [14], a vine-like plant with small, whorled groups of leaves. Hydrilla was first discovered in the area in 1982, in the Potomac River [15], and it is frequently found today in the Chesapeake Bay and a number of rivers, lakes, and ponds throughout the state. As it forms large mats over surface water, it can alter water chemistry and disrupt the food chain, reducing fish diversity. Economically, the prolific growth and propagation of Hydrilla can block pipes and underwater structures, hindering the function of irrigation systems, hydroelectric dams, and even boats. Management of Hydrilla costs millions each year. [21]

The objective of our project is to create a deep learning image classification model that accurately identifies hydrilla plants, compared with other types of submergent aquatic plants that are not invasive species but have similar appearance, to help identify which plants should be removed from a body of water as part of a conservation effort. For this project, we included four classes of native plants, arrowhead (*sagittaria latifolia*), duckweed (*lemnoideae*), grassy mud plantain (*heteranthera dubia*), and watercress (*nasturtium officinale*). Tracking and monitoring these species is labor intensive and can involve considerable expense; a deep learning model can reduce the time and labor needed to accurately identify where hydrilla is overtaking a body of water, to better support more targeted treatment of the problem and more efficient allocation of limited water conservation resources. Our background research on deep learning for plant classification indicates that there is some precedent for using this approach, such as a study that uses CNNs to identify an invasive species, smooth cordgrass, in the Yellow River Delta in China [5]. The authors state that their work would help protect and manage the coastal wetland ecosystem by early detection of Spartina Alterniflora. This particular study used high-resolution satellite images of the Yellow River's wetland areas from 2012 to 2019 and conducted their analysis with two image super-resolution models: super-Resolution Convolutional Neural Networks (SRCNN) and Fast Super-Resolution Convolutional Neural Networks (FSRCNN) [5]. Our goal was to produce a model that conducts assessment and identification of plants that would normally require a labor force of considerable size. While the present focus of the study is Virginia, the model could be used to benefit other areas of North America

where hydrilla is also a problem. We will focus on transfer learning models, given their greater accuracy for image classification, but will also experiment with original architecture for exploration and comparison purposes.

Dataset

We built the dataset using images largely from two free repositories. First, we used forestryimages.org and invasive.org, which are part of the Bugwood Images archive, a project of the University of Georgia, Center for Invasive Species and Ecosystem Health. The archive contains more than 300,000 images of trees, plants, invasive species, insects, wildlife and other types of natural resources, available for free for educational purposes [2]. The Bugwood Images archive contains just over 200 images of hydrilla alone [3], and more than 2,000 images of aquatic plants, including watercress, another Virginia native species that we included as a comparison class, given their similar appearance. We also used the image repository of the Global Biodiversity Information Facility, gbif.org, to further expand our dataset. Both repositories allow images to be used for educational purposes. We rounded out the image set to ensure a sufficient number of photos within each of the five classes with other publicly-available images, such as from Shutterstock and Google Images. The available sources allowed us to create a dataset of appropriate size.

Our final data set consisted of 450 images across the five types of plants: Duckweed (98), Watercress (100), Arrowhead (76), Grassy Mud Plantain (75), and Hydrilla (101). As discussed previously, these are aquatic plants native to Virginia; Hydrilla is an invasive plant, and the others are indigenous and do not cause harm to the water ecosystem. The available images in the archives were of variable dimensions, so we resized them as needed as part of data preparation. The images also had different backgrounds, settings, angles, and other distinct qualities that provided diversity and "noise" among the images to help with training the model. We conducted data augmentation as part of building the dataset, discussed below.

Once we created our data set, we uploaded the dataset to Google Drive and mounted our drive to Google Colab. Next, we created a function that takes in image height, width, and batch size as parameters and returns a training, validation, and test set using the image_dataset_from_directory function. We split our initial dataset into 80 % training, 10 % validation, and 10 % test. Our input dimension was 224x224, based on the default input sizes of the models used for our experiments.

We visualized our images to ensure that our images were loaded correctly. Due to the size of our dataset, we implemented an image augmentation step in our process. Using Keras experimental preprocessing, we added steps to flip, rotate, contrast, and zoom images randomly. Then we tested our image augmentation layer to ensure that original images are getting augmented. The data augmentation code was included directly in the models.

Experiments

Initially, we built five different models, which served as our initial experiments to identify invasive species in Virginia wetlands using deep learning image classification techniques. We created four transfer learning models and one custom convolutional neural network (CNN) model for evaluation and comparison. We focused mainly on transfer learning, in order to maximize classification accuracy. The four transfer learning experiments used the following models – Xception, EfficientNet, DenseNet, and VGG – with demonstrated varying levels of performance. In all cases, we used the Softmax function as the activation function in the output layer, given that we are focusing on a classification problem.

Before we downloaded each of the model architectures, we created an input layer with set image width and height. Then we passed the input layer into a Keras data augmentation sequential layer. Using the Keras Application, we instantiated each architecture using the ImageNet weights. We ensured the design did not include the top layers and passed the data augmentation sequential result into the architecture. Afterward, we turned on all layers of the architecture for training. We added a GlobalAveragePooling2D layer to output the feature maps spatial average. Within some of the models, we experimented with a Batch Normalization layer following the Global Average Pooling layer to standardize the inputs and aim to reduce the number of training epochs. We also experimented with dropout layers, early stopping, and ReduceLROnPlateau; details on this are outlined below in each of the model descriptions. For each model, we compiled using a sparse categorical cross-entropy loss function, accuracy as the metric, and tried different optimizers and learning rates.

Below are details on our initial set of models and results. Final results after model tuning are discussed in the Results and Conclusion sections.

Xception

Xception merges the ideas of GoogLeNet and ResNet, claiming the usage of fewer parameters, less memory and fewer computations than a regular convolution layer, yet with better performance. We experimented with two

configurations to solve our image classification question. The first approach adopted an Adam optimizer with a learning rate of 0.01. BatchNormalization was adopted and serves as an inference mode. The initial model accuracy from the training data was 99%, compared to 85% for the validation dataset. The second configuration included a stochastic gradient descent (SGD) optimizer, a momentum of 0.9, a learning rate of 0.01, and a decay rate of 0.001. BatchNormalization was also adopted and served as inference mode. The model accuracy from the training data for this second configuration was 99%, while the validation dataset accuracy was 94%. The initial model accuracy improved with the SGD optimizer compared to Adam.

EfficientNetB6

Most common CNN models are developed on fixed resources and can be scaled up for better accuracy if there are more resources. However, there are cases where there are limitations in resources, and that is where EfficientNet comes into play. Using EfficientNet, we can uniformly scale a network's depth, width, and input resolution using a compound coefficient. We picked EfficientNet because it has been shown to be more accurate and efficient than past CNNs under resource constraints. For our initial model experimentation, we decided to try EfficientNetB6 because it had a higher accuracy on the Imagenet dataset than the other EfficientNet models and fewer parameters than EfficientNetB7. We added a BatchNormalization layer to standardize the inputs and reduce the number of training epochs, following the GlobalAveragePooling2D layer. We included a dropout layer and a dense softmax layer with five classes. Once the model was designed, we compiled our model using a sparse categorical cross-entropy loss function, accuracy as the metric, and looked at three different optimizers, all with a learning rate of 0.01: SGD, NAdam, and RMSProp. We fit all models using an epochs value of ten. The initial model with SGD as the optimizer had a validation accuracy of 78.8%, while the training accuracy was 95%. That initial result showed that the model was overfitting, and we addressed the issue during the model tuning phase, discussed below. The model with NAdam as the optimizer had a validation accuracy of 16.6%, while the training accuracy was 33.3%. The model with RMSProp as the optimizer had a validation accuracy of 20%, while the training accuracy was 33.8%. One significant difference between SGD and the other optimizers was the training time. The SGD took 25 seconds per epoch, while RMSProp and NAdam took 120 seconds per epoch. We also noticed that the RMSProp and NAdam models' validation accuracy fluctuated significantly between epochs, and the accuracy of the model was extremely low. We plan on examining these results further to assess what may be causing a dramatic difference between SGD and the other optimizers. We leveraged the comparative performance information from these initial experiment iterations further in the model tuning phase.

DenseNet121

Developed to address the impact of the vanishing gradient problem on accuracy, DenseNet121 has 121 layers, most of which are packaged within four "dense blocks" of layers in the architecture. In the initial model, we used an SGD optimizer with a learning rate of 0.2 for the first pass with the base layer frozen, and a momentum parameter of 0.9 and a decay of 0.01. In the second pass with all layers made trainable, we used Adam as the optimizer with a learning rate of 0.001. The model demonstrated 95.5% accuracy on the test set, 84.4% accuracy on the validation set, and 84.4% accuracy on the test set. We used early stopping, which stopped the training process after 36 epochs. The difference between the training and validation accuracy rates indicated overfitting, which we addressed further in the model tuning phase.

VGG19

VGG19 has 19 layers and focused on the use of 3 x 3 fixed-size kernels within increasingly deep layers in the architecture. We used an SGD optimizer with a learning rate of 0.2 in the initial model for the first pass with the base layer frozen, and a momentum parameter of 0.9 and a decay of 0.01. In the second pass with all layers made trainable, we used Adam as the optimizer with a learning rate of 0.001. The model demonstrated 21.97% accuracy on the training set, and 26.7% accuracy on the validation set, with a test set accuracy of 26.6%. The low performance on the test set was not a surprise, given the indications in the training epochs that did run. We did some hyperparameter tuning with different combinations of optimizers and learning rates, which did not improve performance. As we moved to the more detailed phase of model tuning, we did not develop this model further, since several others had more promising initial performance.

Custom Neural Network

We experimented with a custom neural network in our image classification approach, in part to compare the performance of a custom archiecture design with established transfer learning architectures. In addition, creating a convolutional neural network could be treated as a baseline at the first step. A CNN convolves learned features with input data and uses 2D convolutional layers. The architecture is composed of two main blocks. The first block acts as a feature extractor, with which we filter the feature maps obtained with new kernels, which gives us new feature maps to normalize and resize, and we can filter again, repeat the process, and so on. Finally, the values of the last feature

maps are concatenated into a vector. The second block at the end of all the neural networks is used for classifications. The softmax function was adopted as an activation function which is suited for our multi-class classification question. We noticed that a dropout layer is necessary to prevent overfitting issues. One dropout layer was adopted. When the dropout layer was not adopted, the model accuracy for the training dataset was 99%, but the accuracy for the validation dataset was 85%, with a gap of 14 percentage points. The gap between the accuracy of training and validation datasets narrowed when a dropout layer was included, with the former being 95% and the latter being 82%, with a difference of 13 percentage points. As we moved to the more detailed phase of model tuning, we did not develop this model further, since several others had more promising initial performance, but the experience of assessing performance of a custom model against transfer learning model was a valuable element of our experiments.

Results

This section covers the model tuning we conducted and the subsequent results obtained with further optimized models. We discuss the different approaches and parameters we experimented with for each model to maximize the accuracy, and the interpretability for each. Following on our initial model construction and basic hyperparameter adjustment, we used the initial performance comparison to identify three models for more detailed hyperparameter tuning to optimize the accuracy – EfficientNetB6, Xception, and DenseNet121.

EfficientNetB6

Our preliminary experiment examined EfficentNetB6 models using three different optimizers: SGD Nesterov, NAdam, and RMSProp. During our initial exploratory process, we noticed that the EfficentNetB6 model with SGD Nesterov as the optimizer had the highest validation accuracy of 78.8% and took less time to train than the NAdam and RMSProp models. Thus, we decided to tune the hyperparameter of the SGD Nesterov EfficentNetB6 model. Since we decided to use transfer learning on a state-of-the-art model, we added additional layers to the existing architecture of the model. We had added a GlobalAveragePooling2D, BatchNormalization, and Dropout layer in our previous work, but we chose to extend this architecture by adding the following layers: Dropout, Flatten, Dense, Dropout, and Dense layers. By including more Dropout layers, we aimed to address the overfitting issue we noticed in our prior work. Previously we had run our model using ten epochs, but we decided to increase our epochs value to 50 so that the model would have enough cycles. However, we noted that the model validation accuracy could stop improving at an earlier epoch value. Thus we incorporated an EarlyStopping callback. This callback monitors the 'val_accuracy' and checks whether the 'val_accuracy' is no longer increasing at the end of every epoch. Once it notices that the 'val_accuracy' has not increased in 10 epochs, it will stop training the model and reduce the computational cost of the training procedure. For this model, we initialized the learning rate to 0.01, but we understood that there could be cases where the model stops improving because it reaches a plateau. Therefore, we decided to use the ReduceLROnPlateau callback. This callback monitors the 'val_accuracy,' and if no improvement is seen for 5 epochs, the learning rate is reduced by a factor of 0.02. Even though we set the epochs to 50, the EfficentNetB6 training stopped at the 27th epoch because the last ten epochs had a validation accuracy of 1.0. Thus, the EarlyStopping callback triggered the stopping of the training of the model. We also noticed that as the training accuracy increases, so does the validation accuracy and that there is no significant gap between the training and validation accuracy. This clearly shows that we were able to address the overfitting issue we had encountered earlier. Furthermore, we noticed spikes in validation accuracy after five epochs. This could be due to the reduction of the learning rate caused by the ReduceLROnPlateau callback. The final training accuracy was 98.44%, the validation accuracy was 100%, and the testing accuracy was 97.78%.

Xception

We experimented with two configurations as part of optimization of this model. The first approach adopted an Adam optimizer with a learning rate of 0.01. BatchNormalization was adopted and served as an inference mode. The model accuracy from the training data was 99% compared to 85% for the testing dataset. The second configuration included a stochastic gradient descent (SGD) optimizer, a momentum of 0.9, a learning rate of 0.01, and a decay rate of 0.001. BatchNormalization was also adopted and served as inference mode. The model accuracy from the training data was 99%, the validation accuracy was 99%, and the testing accuracy was 95.56%. The model accuracy improved with the SGD optimizer when compared to Adam. We can explore the feature maps based on different activation layers. The key idea is to explore which feature maps are getting activated in the model and visualize them. We use blocks 2, 7, 14 (Fig. 1) layers to understand what parts of the image the model focuses on. We are also able to evaluate occlusion sensitivity to visualize which parts of the image affects our neural network model's confidence by hiding parts iteratively (Fig. 2). This is done by systematically occluding different parts of the input image and monitoring the output of the classifier. We can visualize different layers of what the neural network models are really seeing.

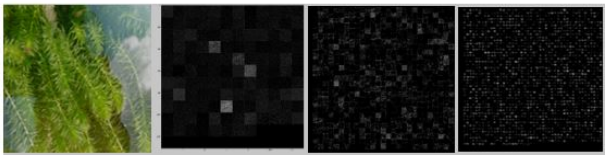

Figure 1: (far left) original image (left) block layer = 2 (right) block layer = 7 (far right) block layer = 14

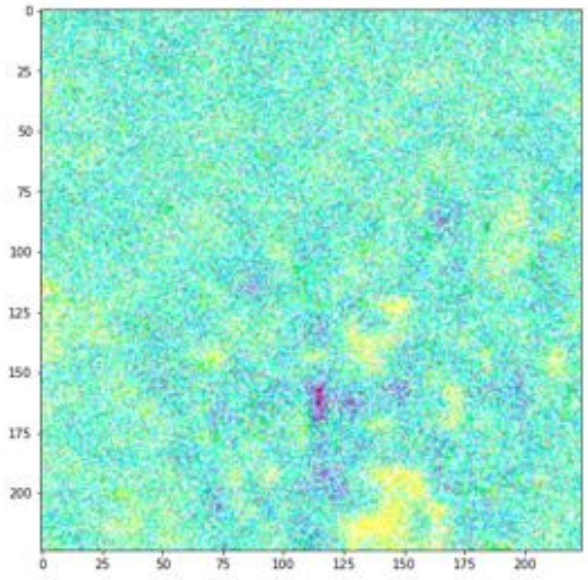

Figure 2: Occlusion sensitivity

DenseNet121

For the DenseNet121 model, we experimented with multiple learning rate and other parameter configurations with both the Adam and SGD optimizers. In the end, the Adam optimizer performed better, and we obtained a 93.3% accuracy on the test set with a learning rate of 0.001, a beta_1 parameter of 0.89, and a beta_2 parameter of 0.999. The training accuracy of 96%, and the validation accuracy of 91%, indicated that overfitting was not a significant issue. We did some further tuning with the Adam optimizer with early stopping and drop-out, including three different drop-out rates (0.5, 0.2, and 0.1), and those did not improve performance. The configuration with a drop-out rate of 0.1 (all other parameters equal to the previous model referenced above) yielded a training accuracy of 94%, a validation accuracy of 93%, and a test accuracy of 89%, so very comparable to the model with no drop-out parameter. The top two performing versions of DenseNet121 indicated that overfitting was not a concern, given the reasonable proximity of the training and validation accuracy rates, as well as the strong performance on the test set. We used 85 epochs, and noticed that around epoch 50, the training and validation accuracies began to converge more consistently.

Conclusion

As demonstrated in the results outlined above, the top three models, Xception, EfficientNetB6, and DenseNet121, all returned respectable accuracy rates after hyperparameter tuning, along with reasonable differences between training, validation, and testing dataset accuracies, indicating that the development and tuning processes sufficiently addressed overfitting. The results also validate the use of transfer learning models for this particular research question, demonstrating the appropriateness of leverage transfer learning architectures to provide greater accuracy. Overall, the results indicate that the models are able to recognize and distinguish all five classes of plant with an acceptable level of reliability. This has several implications in relation to the initial motivation for the project and the relevance to a Virginia policy issue. First, the level of accuracy and reliability is important from the practical application perspective, to ensure that if the model were deployed more widely, it would not carry high risk of incorrectly identifying a plant, thereby resulting in the potential that a healthy plant would be eradicated, rather than an invasive species. In addition,

the model's reliability also supports the potential for wider use, with a larger sample of photos, to conduct more efficient inspections of waterways for invasive species, rather than having to rely on more limited and costly in-person inspections. The efficiencies gained from this analytical solution can allow for limited resources to be allocated more directly to intervention efforts to curb the spread of invasive species in Virginia waterways. To do some conceptual deployment work, we deployed the Xception model to the Google Cloud Platform. The procedure includes uploading the model to Google Storage, hosting the model on AI Platform, and creating a service account to access it. Since we ultimately want to enhance end-to-end user collaboration, we built an initial version of an app in Streamlite linked to our Google AI Platform so users can upload the images and generate and verify model predictions based on the images. This provided an initial proof of concept that could be development further. Potential future work also could include the expansion of the training set to improve accuracy and generalizability, particularly in terms of more diverse images, as well as the addition of more types of plan species, both healthy and invasive, to allow for additional classification capability. The five species we included in this project are fairly widespread in Virginia, but there are many more types of aquatic plants that could be included, given their prevalence in the state's waterways. We could also collaborate with the public and non-governmental and government organizations, such as the Virginia Department of Conservation and Recreation, to control the spread of Hydrilla and other invasive plant species in Virginia. Although the focus is on Virginia, our models could also be applied to other states where hydrilla is also a problem. Also as part of potential expansion, we could add one or more non-plant classes (such as animals) so when users upload non-plant images, our model will be able to label them as such. Given the role Virginia's waterways play in both recreation and the state's economy, an image classification model to assess healthy and invasive species populations quickly and accurately can contribute to the state and local environmental conservation efforts.

Member Contribution

Our group worked well together, and everyone contributed substantially to each of the project milestones. When developing our proposal and finding a dataset, each of us put forward ideas for consideration, identified research papers for the literature review, and identified potential data sources. As we honed in on our specific proposal to focus on Virginia aquatic plants, we discussed the idea extensively together to conceptualize the proposal. When putting together the dataset, we divided up the number of photos needed by species, and each of us collected a portion of images for the total dataset. As shown in our last deliverable, on initial models and implementation, we each put together at least two initial models, with initial tuning and optimization, as shown by the three code files we submitted, to demonstrate that everyone participated in the coding. We then each tuned and further developed one of the "final three" models that we optimized. For all of the write-ups, we each drafted specific portions, and then we discussed and worked together on the editing. For example, in the literature review and initial proposal, we each drafted certain sections, and then we met as a group to bring it all together. For the write-up on the initial models and implementation, we each drafted text to cover the models we worked on, and then we worked together to combine everything, and we took the same approach for the deliverable on model tuning.

Huilin Chang

- Downloaded 98 images of Duckweed and 100 images of Watercress
- Built and tuned her Inception, Xception, and Custom CNN models and wrote a detailed explanation about her models
- Conducted tuning and optimization for the Xception model, as part of the development of the final three most promising models
- Experimented with different data augmentation approaches
- Contributed to writing for all deliverables in Overleaf document
- Contributed to development of several sections of the final presentation
- Developed the Streamlit app used for the model deployment segment of the project

Yihnew Eshetu

- Downloaded 101 images of Hydrilla
- Merged all 5 types of plants image into one folder and ensured all images were readable and were PNG format
- Created a template on how to mount the image folder to Google Colab and data pre-processing and augmentation steps

- Built and tuned his 3 EfficientNetB6 models and wrote a detailed explanation about his models
- Conducted tuning and optimization for the EfficientNet model, as part of the development of the final three most promising models
- Contributed to writing for all deliverables in Overleaf document
- Contributed to development of several sections of the final presentation

Celeste Lemrow

- Proposed the idea to identify invasive species using image data
- Found web sources to download hydrilla images and references to support the importance of this project
- Downloaded 76 images of Arrowhead and 75 images of Grassy Mud Plantain
- Built and tuned her DenseNet121 and VGG19 models and wrote a detailed explanation about her models
- Conducted tuning and optimization for the DenseNet121 model, as part of the development of the final three most promising models
- Contributed to writing for all deliverables in Overleaf document
- Contributed to development of several sections of the final presentation

References

[1] Abdullahi, Halimatu Sadiyah, and Ray E. Sheriff. "Convolution Neural Network in Precision Agriculture for Plant Image Recognition and Classification." *International Conference on Innovative Computing Technology 2021. IEEE Xplore.* Web. 26 Feb. 2021.

[2] Center for Invasive Species and Ecosystem Health. "About Forestry Images" *https://www.forestryimages.org/about/*

[3] Center for Invasive Species and Ecosystem Health. "Hydrilla" *https://www.invasive.org/browse/subthumb.cfm?sub=3028*

[4] Center for Invasive Species and Ecosystem Health "I am a fisherman/boater. Why should I care about invasive species?" *https://www.invasive.org/101/FishermanBoater.cfm*

[5] Chen, Mengmeng, Yinghai Ke, Junhong Bai, Peng Li, Mingyuan Lyu, Zhaoning Gong, and Demin Zhou. "Monitoring Early Stage Invasion of Exotic Spartina Alterniflora Using Deep-learning Super-resolution Techniques Based On Multisource High-resolution Satellite Imagery: A Case Study in the Yellow River Delta, China." *International Journal of Applied Earth Observation and Geoinformation.* 20 June 2020. Web. 13 Mar. 2021.

[6] Chen, Shuchang, Bingchan Li, Jie Cao, and Bo Mao. "Research on Agricultural Environment Prediction Based on Deep Learning." *Procedia Computer Science.* Elsevier, 18 Oct. 2018. Web. 26 Feb. 2021.

[7] Dahikar, Snehal and Rode, Sandeep. "Agricultural Crop Yield Prediction Using Artificial Neural Network Approach." *International Journal of Innovative Research in Electrical, Electronics, Instrumentation and Control Engineering.* January 2014. Web. 26 Feb. 2021.

[8] Google Machine Learning. "Convolutional Neural Network definition, Google Machine Learning Glossary." *https://developers.google.com/machine-learning/glossary*

[9] Lu, Yuzhen, and Sierra Young. "A Survey of Public Datasets for Computer Vision Tasks in Precision Agriculture." *Computers and Electronics in Agriculture.* Elsevier, 08 Sept. 2020. Web. 26 Feb. 2021.

[10] Nasir, Inzamam Mashood, Bibi, Asima, Shah, Jamal Hussain, Khan, Muhammad Attique. "Deep Learning-Based Classification of Fruit Diseases: An Application for Precision Agriculture." *Computers, Materials & Continua.* Web. 26 Feb. 2021.

[11] Nguyen, Thanh Tam, Thanh Dat Hoang, Minh Tam Pham, Tuyet Trinh Vu, Thanh Hung Nguyen, Quyet-Thang Huynh, and Jun Jo. "Monitoring Agriculture Areas with Satellite Images and Deep Learning." *Applied Soft Computing.* Elsevier, 23 July 2020. Web. 26 Feb. 2021.

[12] Tahir, Muqadas Bin, Muhammad Attique Khan, Kashif Javed, Seifedine Kadry, Yu-Dong Zhang, Tallha Akram, and Muhammad Nazir. "Recognition of Apple Leaf Diseases Using Deep Learning and Variances-Controlled Features Reduction." *Microprocessors and Microsystems.* Elsevier, 15 Jan. 2021. Web. 26 Feb. 2021.

[13] Tombe, Ronald. "Computer Vision for Smart Farming and Sustainable Agriculture." *2020 IST-Africa Conference (IST-Africa) IEEE Xplore.* Web. 26 Feb. 2021.

[14] Virginia Department of Conservation and Recreation. "Virginia Invasive Plant Species List" *https://www.dcr.virginia.gov/natural-heritage/document/nh-invasive-plant-list-2014.pdf*

[15] Virginia Institute of Marine Science. "Hydrilla Overview" *https://www.vims.edu/research/units/programs/sav/species/hydrilla.php*

[16] Virginia Places. "How much of Virginia is water?" *http://www.virginiaplaces.org/watersheds/howmuchwater.html*

[17] Wallelign, Serawork, Mihai Polceanu, and Cédric Buche. "Soybean Plant Disease Identification Using Convolutional Neural Network." *Artificial Intelligence Research Society Conference.* 05 June 2018. Web. 26 Feb. 2021.

[18] Yuan, Yuan, Lei Chen, Huarui Wu, and Lin Li. "Advanced Agricultural Disease Image Recognition Technologies: A Review." *Information Processing in Agriculture.* Elsevier, 30 Jan. 2021. Web. 26 Feb. 2021.

[19] Zapotezny-Anderson, Paul, and Chris Lehnert. "Towards Active Robotic Vision in Agriculture: A Deep Learning Approach to Visual Servoing in Occluded and Unstructured Protected Cropping Environments." *IFAC-PapersOnLine. Elsevier, 31 Dec. 2019.* Web. 26 Feb. 2021.

[20] Zheng, Yang-Yang, Jian-Lei Kong, Xue-Bo Jin, Xiao-Yi Wang, Ting-Li Su, and Min Zuo. "CropDeep: The Crop Vision Dataset for Deep-Learning-Based Classification and Detection in Precision Agriculture." *MDPI. Multidisciplinary Digital Publishing Institute*, 01 Mar. 2019. Web.

[21] Center for Invasive Species and Ecosystem Health Biological Control of Invasive Plants in the Eastern United States *https://www.invasive.org/biocontrol/7Hydrilla.html*