Invasive Species Detection

Christy Dennison Charles Hale
{cdenn,cphalepk}@stanford.edu
Stanford University

Abstract

Invasive species can cause devastation for native species. It is crucial that invasive species can be quickly and accurately identified so they can be removed. A Kaggle competition aims to find the best software-based solution for detecting invasive hydrangea in a forest environment. We experimented with several variations of transfer learning, input preprocessing, and input resizing to explore how best to apply CNNs to solve this problem. We found an ensemble of modified ResNet and VGG models achieves up to 99.36% *accuracy on the test set.*

1. Introduction

With unprecedented levels of global human activity, many plant species have found new homes in foreign lands. These plants, unchecked by natural competitors, can be detrimental to their foreign environments, and are therefore labeled "invasive species"[9]. In order to curb the damage caused by invasive plants, it is imperative that non-experts be able to identify and report them so that they may be removed swiftly. Ecologists are hosting a Kaggle competition to see if computers can accurately identify invasive species in photos. We found that this problem is very tractable with deep convolutional neural networks (CNNs), as introduced in [17].

2. Background

The Kaggle competition "Invasive Species Monitoring" to detect invasive hydrangea seeks to advance software that can quickly identify invasive species in a forest scene image [3]. The dataset has images that either contain hydrangea in them or do not, which makes this a binary classification problem. Submissions to Kaggle must give a probability that the invasive species is in the image for each image in the test set. We entered this competition using an application of Convolutional Neural Networks. Our goal was to get as far up the leaderboard for this competition as possible.

2.1. Competition Evaluation

The competition's loss function is the area under the ROC curve, so we wanted our submission scores to directly correspond to actual probabilities or overall confidence. Softmax loss was the most appropriate loss function for this problem, since we wanted to push our predictions as close to 0 or 1 as possible.

2.2. Dataset

The Kaggle dataset has 2295 training images (which we split 80/20 for training and validation) and 1531 test images. All images are 866x1154 pixels in size. A typical training image can be seen in figure 1. About two-thirds of the training set are positives, and most of the positive images have full-frame hydrangea bushes, like in the typical image. There are also a significant number of images that have only small portions of the image occupied by hydrangeas, sometimes without their flowers. The negative images are mostly forest scenes without hydrangea, as can be seen in figure 2. Occasionally they contain other features such as lakes, horses, sheds, or people.

Figure 1: Typical positive training image

2.3. Previous Work

The literature yields ubiquitous examples of using CNNs for binary classification problems similar to this one. Many papers experiment with several popular architec-

Figure 2: Typical negative training image

tures and compare their performance on their particular dataset. For instance, [16] used ResNet[12] and a standard deep convolutional network for classification of brain MRI data and saw almost no difference between the two. [10] used simple 3-layer CNN architectures, Conv/ReLU[21]/Norm[14]/Pool, for determining the age of a face (if age is greater or less than k), and found using CNNs for binary classification to be very effective. [20] used a single layer for binary classification, and achieved reasonable results given the depth of the network with averaging ensembles and transfer learning.

However, we recognized that binary model performance may heavily depend on the particular idiosyncrasies of our problem and dataset. We sought to investigate previous work specifically involving plants. We found most other plant classification problems, such as in [22], [7], and [18], involve full-frame photos of the plant leaf with the background removed, rather than in a natural environment. One paper that investigated classifying plants in their natural environment used ResNets of various depths[27], but the plants were usually always in full-frame. Our dataset contains many images where hydrangeas only appear at the margins or are occluded. Few papers feature a need for classification without regional detection, so the literature on this subject is limited.

Additionally, we reviewed the discussion section of the competition's page on Kaggle and found that not much had been said about this particular dataset or methods used for good performance, as the competition is still ongoing. However, some had claimed they were able to get up to 98% accuracy using simple transfer learning with VGGNet.

Furthermore, we reviewed other binary classification challenges on Kaggle. We found the interview with the winner of the "Cats vs Dogs" competition[26] to be particularly insightful. Most top performers on Kaggle use large model ensembles that simply average the results of each model's prediction. We replicated this approach for our best results.

3. Method

3.1. Baseline

To have a baseline for which we could compare how well we should expect our later models to reasonably perform, we trained a 3-layer convolutional neural net, using BatchNorm[14], ReLU[21], Max Pooling, and Dropout[25] with the following architecture:

```
[CONV-BatchNorm-ReLU-Pool]*3 ->
Affine-BatchNorm-ReLU-Dropout-Affine
```

3.2. Transfer Learning

Since the dataset is relatively small (2295 training images is not enough to train a good model), we also used transfer learning against a set of pre-trained models, focusing on ResNets[12] and VGGNets[24] of various sizes that were pre-trained on ImageNet[8]. There is a range of methods that can be used to apply these pre-trained models to a new dataset.

The first step is always replacing the last fully-connected layer; this is necessary to output the classes of the new inputs. In our case, we replaced the last fully-connected layer with a final dimension of 2, with 0 meaning "invasive not present" and 1 meaning "invasive present".

Additionally, we explored a range of methods for retraining the adjusted models to our dataset, including:

- Train the last layer only
- Train the first and last layers only
- Train the entire network

We tried all of these methods in our experiments.

3.3. Image Size

The primary challenge we had to overcome when using pre-trained models was adapting the input images' size. The standard pre-trained ResNet and VGG available in PyTorch require images of size 224x224x3 pixels. With the training and validation images in our dataset having 866x1154x3 pixels, we had to explore different approaches for resizing them to fit into the pre-trained models.

3.3.1 Resizing Approaches

We experimented with all of the following methods for resizing dataset images to fit into our pre-trained models:

- Insert a convolutional layer at the beginning such that we scale the input down to size 224x224.
- Crop the input randomly to a square size (866x866), then rescale using Lanczos interpolation[23] to 224x224.

- Insert a Spatial Pyramid Max Pool[11] layer at beginning as a form of preprocessing to scale the image to 224x224.

- Resize the input image to 224x224 using Lanczos interpolation.

3.4. Preprocessing methods

We experimented with the following ways of preprocessing the image pixels:

- Normalize the input images with the means and standard deviations from ImageNet

- Normalize the input images with the means and standard deviations from the dataset itself

Source	R	G	B
Dataset Mean	0.4249	0.4764	0.3832
Dataset Std Dev	0.0819	0.0810	0.1059
ImageNet Mean	0.485	0.456	0.406
ImageNet Std Dev	0.229	0.224	0.225

Table 1: Means and Standard Deviations

As can be seen in table 1, the dataset means are relatively close to the ImageNet means compared to the standard deviations. The difference between the standard deviations can be attributed to the fact that our dataset images are mostly green and brown (trunks, ground), unlike ImageNet, which has a wide range of images from 1000 different classes.

3.5. Evaluation

To evaluate the results of each of our experiments, we compared accuracy against the validation and test sets and examined ROC curves, since ROC curves are used for the competition's evaluation. We also inspected images that were misclassified to evaluate the qualitative results. We conducted experiments comparing each class of methods independently. For example, to test how we should train our networks, we evaluated models' performance after training with each method, while keeping all other variables constant (like how we resize or preprocess the inputs). Similarly, when comparing the performance of our various proposed preprocessing methods, we trained each of the networks the same way, etc.

Of all the methods we tried, once a method was shown to be best, we moved forward with the other experiments using that best method and did not try every combination of methods due to limited time and computational resources.

4. Experiments

All code used for these results is available publicly at [2]. Code for saliency maps, fooling images, training code, and other code can be accredited in part to [1].

We used an assortment of ResNet and VGG pre-trained models with results reported in table 2 for ResNet18, table 3 for other models, and table 4 for ensembles. We discuss these results below. Note that there are few test accuracies listed because Kaggle limits the number of submissions per day to 3, so we did not submit until we believed the submission could do better than our previous submission.

All ResNet pre-trained models were trained for 2-5 epochs depending on what produced the most accurate results; overfitting must be occurring after the 2nd-5th epoch to explain the lower validation accuracies. For VGG, 5 epochs produced the most accurate results on the validation set. All training used the Adam[15] optimizer. We tuned the hyperparameters of the networks on a case by case basis based on validation results after 1 training epoch. Most learning rates were around 10^{-4}. The regularization constant (weight decay) for most models was around $10^{-2.7}$.

4.1. Simple Model

The simple model produced a 60% accuracy on the validation set within 2 epochs training on the training data, after which its validation accuracy did not climb. Without a larger dataset, it was difficult to see how we could achieve better performance with this model. This was clearly not good enough to enter into the competition, where the top ten entries are 0.99 AUROC or higher. We concluded we had to use transfer learning to make any reasonable entry into this competition.

4.1.1 Learning Method Results

Training only the first and last layers of a modified pretrained ResNet18 performed better than our simple baseline model, with 92.94% accuracy on the validation set. Training only the last layer proved to be ineffective, scoring at only 87.11% accuracy on the validation set. Given that the worst entire-network-trained model scored 95.44%, we abandoned all other techniques in favor of retraining the entire network for the rest of our experiments.

4.1.2 Resizing

4.1.2.1 Convolutional Layer

We used a number of different methods for resizing the dataset images. We first tried replacing the first layer with a convolutional layer whose output dimensions were 224x224x3: Our original input size was 866x1154x3 so we first cropped the images to size 866x1120x3, and then sent

Figure 3: Randomly cropped difficult image with original. Note the hydrangea bush is almost completely excluded by the random crop.

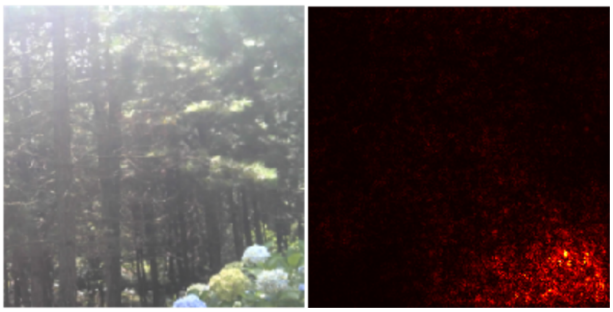

Figure 4: Random cropping false negative with saliency map

them through a convolutional layer with 3 filters with size 5x7 at a stride of 4x5 and padding of 13x3.

Replacing this first layer with this downsizing convolutional layer had unimpressive results, at best scoring 90.21% accuracy on the validation set. We concluded that the first layer of a pre-trained network has vital information for the rest of the downstream filters. We simply do not have enough data to train this first layer effectively, and so replacing this first layer can only yield poor results. We replaced only the last layer for the rest of our experiments.

4.1.2.2 Random Cropping

A standard approach to resizing is simply to crop the image to a square and then scale down to the desired size. This technique preserves the aspect ratio. To keep the cropping as equal as possible, we used random cropping. However, the downside of cropping is that important information at the boundaries of the image can be lost. We found that this was especially the case for some of the more difficult images in this dataset, where 25% of the image was being eliminated by cropping. Additionally, several of the trickier training and validation images had the bulk of the image's hydrangeas in that last quarter of the picture, for instance, in figure 3, which almost completely excludes the hydrangeas.

Furthermore, the positive images generally do not feature hydrangeas in a consistent region. Many images feature Hydrangeas on the edges, or shifted to one side, etc. Additionally, some plants may be occluded with a random horse blocking half the image, etc., so the network did not get a perspective of the hydrangeas that was consistent.

With an accuracy of 96.81%, this cropping technique performed better than changing the first layer into a downsizing convolutional layer, but results were wildly inconsistent between runs, with a standard deviation of 1.4, compared to 0.97 with the scaling technique.

The saliency maps of the incorrectly classified results in figure 4 were as expected. They showed the networks were correctly focusing on the Hydrangea flowers, but simply were not confident enough to give the correct score. The cropped image just barely missed where the bulk of the hydrangeas were in the image (see original in figure 5), leading to a false negative.

Figure 5: Original image from random cropping from fig. 4

4.1.2.3 Spatial Pyramid Max Pool

Another resizing approach we experimented with was using a Spatial Pyramid Max Pool[11] layer at the beginning. Pooling layers typically appear in the middle of network architectures as a way to reduce memory cost and focus only on the largest signals. Using a pooling layer at the beginning of a network is usually ill-advised since the raw signals from an image are the pixel values corresponding to the color in the image, and color can be a bad signal for classification, especially in our test set where the majority of the pixels from the dataset are green. However, the Spatial Pyramid Max Pool layer has the advantage of performing pooling and resizing, which is great for CNNs where datasets have dynamically sized inputs; this pooling layer is needed before any fixed-size input layer.

As a preprocessing layer, Spatial Pyramid Max Pooling layer amounts to simply resizing the input using maxing rather than interpolation. In our experiments, it performed better than random cropping at 97.27%, but there was likely room for improvement when we knew that the pooling layer

Figure 6: Spatial Pyramid Max Pooling false negative with saliency map

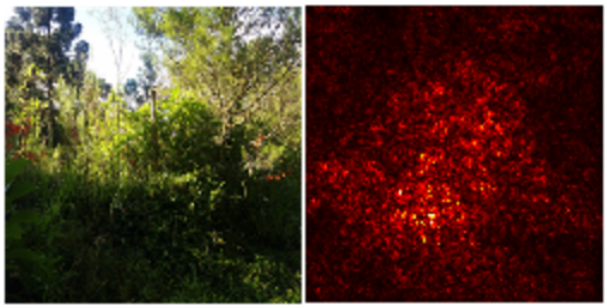

Figure 7: Lanczos false negative with saliency map

was not used as intended: our inputs do not have dynamic sizes. Interestingly, this method produced no false positives in the best case, which no other model achieved except for the best ensemble.

The saliency maps for using Spatial Pyramid Max Pooling (in figure 6) were interesting.

Compared to those for random cropping, these maps look nearly blank. The signals that come across are much more subdued. We suppose that the pooling layer kills a lot of important information. Therefore, by the time the reduced image is processed at the end of the network, there are very weak signals to work with. The focus is still correctly placed on the hydrangea bush, but it is harder to see.

4.1.2.4 Lanczos Scaling

Lanczos[23] scaling is a technique for downsizing images that is not as fast as other techniques, but preserves quality very well[5]. We scaled the images directly down to size 224x224, essentially "squishing" the images. The mean run using this technique produced 98.18% accuracy on the validation set, which was the best performance we encountered among techniques for resizing our inputs. We used Lanczos scaling for the rest of the experiments.

Notably, the same image in figure 6 that the Spatial Pyramid Max Pooling model misclassified was also misclassified by our model using this scaling method. However, comparing the saliency maps, we see this model is focusing on the wrong part of the image in figure 7. The difference could be explained by the pooling layer pooling a specific bright color that corresponds to the leaves of the hydrangea (the leaves of the hydrangea bush appear to be brighter than other leaves in the image), but we are not exactly sure.

4.1.3 Preprocessing Methods

As can be seen in rows 3-12 of table 2, all types of normalization preprocessing performed worse in all cases except for random scaling than simply having no normalization. Using the dataset's means and standard deviations tended to perform better than using ImageNet's. This may be because the ImageNet normalization introduces an unhelpful bias. We found these results especially surprising since the pre-trained models were all originally trained with this preprocessing.

4.1.4 Other Methods

4.1.4.1 Change Validation Set Size

We were using an 80/20 split for our training and validation sets from the training set that was provided by Kaggle, but we were able to get better results when our models had more information. We decided to try changing the ratio to 90/10. This improved the validation score slightly from 98.18% to 98.23%, and improved the test score from 98.54% to 98.79%. However, when all of the models were re-trained on this split, we got worse results for validation and test, as can be seen in table 3 and 4. Note that the validation accuracies are not completely comparable since the validation set is now cut in half, but notably all ResNets except ResNet18 got 4 incorrect in the 90/10 validation set, but ResNet101 and ResNet152 only got 3 incorrect in the 80/20 split; VGG13 got 2 fewer incorrect and ResNet18 got none incorrect in the 90/10 split. The final ensembled submission to Kaggle gave us AUROC of 99.2%, which was greater than our validation accuracy of 98%. Since our test and validation accuracies were not wildly lower than training accuracies, we know our models were not overfitting. Instead, we believe that this did not end up improving our score because using the 90/10 split gave us a validation set that was too small to give us an accurate estimate for generalization error. As a result, our hyperparameters were chosen sub-optimally and did not give us the benefit of having the extra training data.

4.1.4.2 Train on Errors

Given the better results from changing the validation set, we explored how we could get better performance by adding

some supervision to how we split the training and validation set. We tried a somewhat unconventional approach where we trained the model with the incorrectly classified training or validation images again, as can be seen in the last two rows of table 2. Training again with the incorrect training images decreased the accuracy against the validation set, most likely because we started to overfit. Training with the validation images improved both validation and test accuracy by more than a whole percentage point, from 98.18% to 99.23%, which makes sense given that we are obtaining more information about the set we were validating against. However, other nets, such as ResNet34 and ResNet50, did not gain in validation accuracy from this extra training, while ResNet18, ResNet101, and ResNet152 did.

4.1.5 Ensembling

We used Resnet18 for all of our experiments. Since we then knew what approximately produced the best results, i.e., retraining the whole network, using Lanczos interpolation to resize inputs with no input normalization on a 80/20 percent split between training and validation sets, we were able to quickly replicate good results using other models. As such, we trained ResNet34, ResNet50, ResNet101, ResNet152, and VGG13 models, with accuracy results in table 3. (We attempted training SqueezeNet[13] as well, but even with extensive hyperparameter tuning, we were not able to get over 97% accuracy on the validation set, so SqueezeNet was abandoned.) The main goal with creating these other models was for use in an ensemble. Results for the ensemble are available in table 4. We tried three methods for combining the results of the outputs from the different ensembles.

4.1.5.1 Max Probability

The first method for combining the results from the models in the ensemble was to take the maximum probability. Whichever of the models was most confident, we would trust that model with the result. However, this technique saw a near percentage point difference between the validation results and test results. This is most likely because while a model may be highly confident, that reason for confidence may not be correct, and the different models may be less confident but the consensus between these others can be more meaningful and correct. To measure the consensus between these models, we tried averaging.

4.1.5.2 Average Probability

Instead of taking the maximum result, we averaged the output probabilities from each model for each image. This gave us by far the best test accuracy, despite sub-optimal validation accuracy. The deviation between the validation accuracies and test accuracies for maxing and averaging may be because picking the most certain model can incur much more loss when all or one of the models is confidently wrong. These penalties end up outweighing the small performance increase seen where the models were right.

4.1.5.3 Average Probability with Exponential Weighting

One problem with averaging the probability scores is that models that had lower validation accuracy scores were still weighted as much as those that had higher scores. What we actually have with a set of well-performing models is a class of experts. By creating an ensemble, we are trying to find the best way to combine each model's opinion in a way that gives us the best performance. The theory of competition between a class of experts is a well developed field of statistics[4], particularly in non-Bayesian statistical Decision Theory. This theory suggests that when making sequential decisions, we should weight each "expert" (model) according to how correct it was for a particular timestep. For our specific problem, this translates to weighing the probability scores for each of our models in proportion to the model's validation set error, as seen in [4]. These methods are usually used for online learning and weighting of experts, where we see how each of our experts perform over time, e.g., following the stock market. Since we only have two epochs that we care about (accuracy on the validation set and accuracy on the test set), we only have 1 timestep of exponential weighting. In simple words, we weight the scores given by each model according to the negative exponentiation of its error rate:

$$w_m = e^{-error\_rate_m} \tag{1}$$

$$p_{image} = \frac{\sum_{m=1}^{M} w_m score_m}{\sum_{m=1}^{M} w_m} \tag{2}$$

where M is the number of models in the ensemble.

While this approach increased our validation accuracy for the ResNet/VGG ensemble in table 4, our test accuracy decreased, suggesting that our greater "experts" were not as strong on the test set as the validation set, relative to their peers. This result is not terribly surprising considering the performance of exponential weighting only enjoys optimal theoretical guarantees as $t \to \infty$. Additionally, validation set error can only be thought of as an estimate of the true expert's performance, since it comes from a small sampling of the overall class of images we care about, so theory might also require that n be large.

4.2. Best Approach

By far, the VGG and ResNet ensemble using averaged probabilities performed the best, and brought us up to 5th

place on the leaderboard, from 12th with the ResNet ensemble, and 21st from the best ResNet model. Our ROC curve is impressive, almost a straight horizontal line in figure 8.

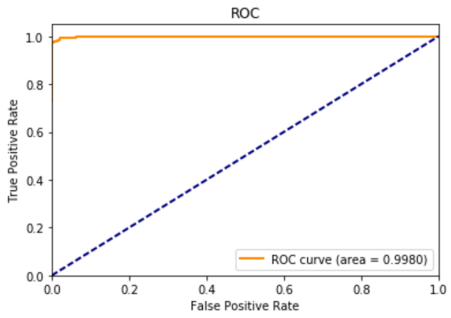

Figure 8: Ensemble ROC Curve

When looking at each of these models, it is interesting to note that each mostly got a different set of images incorrect, so averaging these results means that these individual losses don't equate to a loss for the whole.

4.2.1 Saliency Maps

As can be seen in figure 9, the image that the ensemble incorrectly classified as a negative is a tough image to classify, with the flowers far in the background, unlike many of the other images that have the hydrangea flowers in full-frame (see figure 1). The saliency map for one of the models that got the image incorrect (ResNet101 in this case, but there were others) shows that the model is again focusing on the correct spot, but it as a whole the ensemble is not confident enough to classify this image as a positive.

As can be seen in figure 10, the image that the ensemble incorrectly classified as a positive is also tough image to classify. The saliency map for one of the models that got the image incorrect (again ResNet101 in this case, but there were others) shows that the model is focusing on a spot that has bunched flowers, and the bright spot in the saliency map shows the confidence that this model (and others) have.

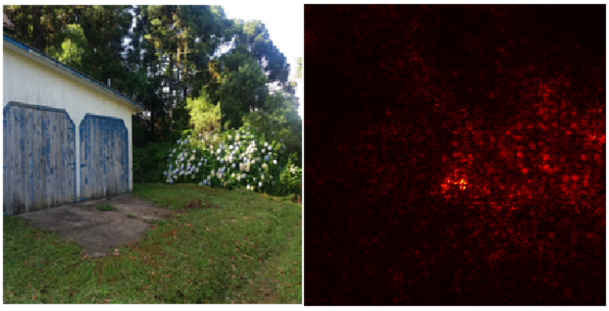

Figure 9: Ensemble false negative with saliency map

Figure 10: Ensemble false positive with saliency map

4.2.2 Fooling Images

We "fool" the network by using gradient ascent to modify the image to be classified as the correct class. In figure 11 we can see that the background flowers are emphasized, even visibly in the fooled photo. In figure 12, we can see that the white flowers are also changed, likely deemphasized, even visibly in the photo, and the surroundings are emphasized, likely to draw attention away from the flowers.

Figure 11: Ensemble fooling false negative to positive

Figure 12: Ensemble fooling false positive to negative

4.3. SSD Attempt

Based on the errors made by our best model ensemble, we had reason to believe an attention-based model may be best for this problem. Most errors were due to hydrangea flowers that were far in the background, as in figure 9, so we thought that a model with the ability to zoom in on important regions and classify them at a more reliable scale could be key to tackling these edge cases. SSD [19] was implemented based on [6]. We attempted to work this code into our problem again using transfer learning. However, procuring the training data required us to manually label

images by drawing bounding boxes around the flowers. We found that we had insufficient time to make a training set with meaningful size to train a reliable SSD model. In a future attempt, Mechanical Turk could be used to speed up this process.

5. Conclusion

By analyzing the false negatives and positives of our network, we conclude that hydrangea detection is very reliably and accurately solved using transfer learning with a CNN ensemble and Lanczos scaling for input resizing. However, this method is less reliable for photos where hydrangea are far in the background or are at a small scale. We believe that an attention-based model with a larger dataset would be able to overcome these limitations.

6. Thanks

A special thanks is warranted specifically to Timnit and Leo for their help on this project, but also the entire course faculty and staff for creating such a fun and interesting class. This class was exceptionally well-run and has made us excited for the future of ML. Thank you for helping us learn such a critical path for the future!

ResNet18 Model Setup	Val	Test
Insert conv layer for scaling	90.21	
Insert conv layer for scaling, only change first and last	92.94	
Random crop w/o norm.	96.81	
Random crop w/ ImageNet norm	95.44	
Random crop w/ dataset norm	97.72	
Spatial Pyramidal Pooling w/o norm.	98.18	
Spatial Pyramidal Pooling w/ ImageNet norm	97.27	
Spatial Pyramidal Pooling w/ dataset norm	97.72	
Lanczos Scaling w/o norm.	98.18	98.54
Lanczos Scaling w/ ImageNet norm	97.49	
Lanczos Scaling w/ dataset norm	97.49	
Lanczos Scaling w/o norm., 90/10	98.23	98.79
Lanczos Scaling w/o norm., change only last layer	87.11	
Lanczos Scaling w/o norm., plus train on training errors	97.95	
Lanczos Scaling w/o norm., plus train on validation errors	99.32	98.81

Table 2: ResNet Variations

Other Models Setup (w/o norm)	Val	Test
ResNet18 Lanczos Scaling, 90/10	100.00	
ResNet34 Lanczos Scaling, 90/10	98.00	
ResNet50 Lanczos Scaling, 90/10	98.00	
ResNet101 Lanczos Scaling, 90/10	98.00	
ResNet152 Lanczos Scaling, 90/10	98.00	
VGG13 Lanczos Scaling, 90/10	97.50	
ResNet18 Lanczos Scaling (prev.)	99.32	
ResNet34 Lanczos Scaling	99.09	
ResNet50 Lanczos Scaling	99.09	
ResNet101 Lanczos Scaling	99.32	
ResNet152 Lanczos Scaling	99.32	
VGG13 Lanczos Scaling	98.17	

Table 3: Other Pre-Trained Models

Ensemble Model Setup	Val	Test
ResNet (18, 34, 50, 101, 152), max	99.54	98.33
ResNet (18, 34, 50, 101, 152), ave.	99.32	99.16
VGG13 + ResNet (18, 34, 50, 101, 152), max	99.77	98.39
VGG13 + ResNet (18, 101), ave.	99.32	
VGG13 + ResNet (18, 101, 152), ave.	99.54	99.24
VGG13 + ResNet (18), ave.	99.77	99.20
VGG13 + ResNet (18), ave., 90/10	99.50	98.93
VGG13 + ResNet (18, 152), ave.	99.77	99.20
VGG13 + ResNet (18, 34, 50, 101, 152), ave., error weighting, 90/10	100.00	99.20
VGG13 + ResNet (18, 34, 50, 101, 152), ave., 90/10	98.00	99.23
VGG13 + ResNet (18, 34, 50, 101, 152), ave., error weighting	99.77	99.32
VGG13 + ResNet (18, 34, 50, 101, 152), ave.	99.54	99.36

Table 4: Ensembles

References

[1] CS231N assignments. cs231n.github.io. Accessed: 2017-06-10.

[2] Github cs231n-final. https://github.com/cs231n-2017/cs231n-final. Accessed: 2017-06-10.

[3] Kaggle invasive species monitoring. https://www.kaggle.com/c/invasive-species-monitoring. Accessed: 2017-05-15.

[4] MIT 9.520 online learning. http://www.mit.edu/~9.520/spring08/Classes/online_learning_2008.pdf. Accessed: 2017-06-11.

[5] Pillow filters comparison table. http://pillow.readthedocs.io/en/4.1.x/handbook/concepts.html#concept-filters. Accessed: 2017-06-01.

[6] SSD pytorch. https://github.com/amdegroot/ssd.pytorch. Accessed: 2017-06-10.

[7] I. Çugu, E. Sener, Ç. Erciyes, B. Balci, E. Akin, I. Önal, and A. O. Akyüz. Treelogy: A novel tree classifier utilizing deep and hand-crafted representations. *CoRR*, abs/1701.08291, 2017.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[9] V. T. Eviner and C. V. Hawkes. The effects of plant-soil feedbacks on invasive plants: mechanisms and potential management options. In S. R. L. Monaco, T. A., editor, *Invasive plant ecology and management: linking processes to practice.*, chapter 7, page 122. 2012.

[10] J. F. J. Z. Hao Liu, Jiwen Lu. Group-aware deep feature learning for facial age estimation. 2016.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[13] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[15] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[16] S. Korolev, A. Safiullin, M. Belyaev, and Y. Dodonova. Residual and plain convolutional neural networks for 3d brain MRI classification. *CoRR*, abs/1701.06643, 2017.

[17] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comput.*, 1(4):541–551, Dec. 1989.

[18] S. H. Lee, C. S. Chan, P. Wilkin, and P. Remagnino. Deep-plant: Plant identification with convolutional neural networks. *CoRR*, abs/1506.08425, 2015.

[19] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[20] S. Mehri. On binary classification with single-layer convolutional neural networks. *CoRR*, abs/1509.03891, 2015.

[21] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In J. Frnkranz and T. Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress, 2010.

[22] A. K. Reyes, J. C. Caicedo, and J. E. Camargo. Fine-tuning deep convolutional networks for plant recognition. In *CLEF*, 2015.

[23] J. Rissanen. Solution of linear equations with hankel and toeplitz matrices. *Numerische Mathematik*, 22(5):361–366, 1974.

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[26] K. Team. Dogs vs. cats redux playground competition, 3rd place interview: Marco lugo. April 2017.

[27] G. W. Yu Sun, Yuan Liu and H. Zhang. Deep learning for plant identification in natural environment.