Capstone Project

Robert Young

May 19, 2018

1 Definition

1.1 Project Overview

The Kaggle Competition, "Invasive Species Monitoring", recognises the widespread impact of invasive species and emphasises the cost and time involved to track both the location and spread of these invasive species on a large scale.

The monitoring of an ecosystem and plant distribution requires expert knowledge in the form of trained scientists. These scientists must visit effected areas to examine and record the invasive species present. This is time consuming, expensive, and not practical on a large scale. The objective of this competition is to identify the invasive species, *hydrangea*, in images taken in the Brazilian national forest, using machine learning techniques. From the training images and labels provided, the designed model should predict the presence of this invasive species in the testing images.

1.2 Problem Statement

In this capstone, the problem can be solved by creating a machine learning algorithm that can identify, with reasonable accuracy, if an invasive plant is present in images of forests and foliage. This will be treated as an image classification problem, where the algorithm output will be the probability of an image containing the invasive species *Hydrangea*. The work of Razavian et al [12] found that features obtained from deep learning with a convolutional neural network should be considered as the primary

method for image recognition. I intend to solve this problem using a pre-trained convolutional neural network.

The model created will be trained, tested and validated against the "Invasive Species Monitoring" Kaggle Competition datasets [8]. The model accuracy will be validated on the area under the ROC curve [15] between predicted probability and the observed target. The model will be tested on 1531 test images.

1.3 Metrics

The Kaggle Competition evaluation rules state that submissions are evaluated on the area under the ROC curve (AUC) between the predicted probability and the observed target. It is created by plotting the true positive rate against the false positive rate. For my model, this is a two-class prediction problem, also known as binary classification. The ROC AUC varies between a value of 0 and 1. with an uninformative class yielding 0.5.

The evaluation metric will take the ROC curve and evaluate against the area under the predicted probability and observed target. Using normalised units, the area under the curve is equal to the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance. Mathematically, this can be expressed as follows [6]:

$$A = \int_{\infty}^{-\infty} TPR(T)(-FPR'(T))\,dT =$$
$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} I(T' > T) f_1(T') f_0(T)\,dT'\,dT = P(X_1 > X_0)[15]$$

2 Analysis

2.1 Data Exploration

This Capstone Project will work with the datasets as provided - Kaggle Invasive Species Monitoring Competition Datasets. There are two datasets, coupled with two files used for the correct classification of the images [11]. These are:

- train.7z - This is the training set containing 2295 images .

- train_labels.csv - These are the correct labels for the training set
- test.7z - This is the testing set containing 1531 images.
- sample_submission.csv - A sample submission file in the correct format.

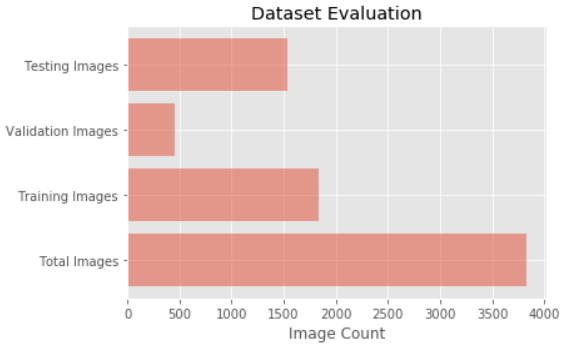
All images are colour, in JPEG format, no larger than 1.6MB in size, with dimensions of 1154 x 866 pixels each. The size of the training set is too small to properly train the CNN, especially as this set will be split to provide a validation set during training of the neural network. To counteract this, during the data pre-processing stage image augmentation will be implemented, as carried out by Francois Chollet [4]. This will be discussed in greater detail in Section 3.1.

One point of note is based on the training labels in the file, train_labels.csv, a class imbalance is present. The True learning examples make up approximately thirty-seven percent of the labels, whereas the False examples make up roughly sixty-three percent. However, the metric chosen for this classification task, which can be see in Section 1.3, is insensitive to class imbalance, thereby mitigating this concern [3].

2.2 Exploratory Visualisation

Figure 1, seen below, displays the breakdown of of total images for each relevant set - training, validation and testing. Twenty percent of the training set will be used for validation. The most striking thing that can be observed from this figure is that number of training images is close in size to the number of testing images. This raises concerns regarding model accuracy. This is as the training set could be too small to allow enough variance for the model to learn to generalise well to unseen data. I will discuss a proposal to counteract this concern in Section 2.2.

Figure 1: Dataset Image Exploration



2.3 Algorithms and Techniques

Deep Learning in the form of a Convolutional Neural Network will be deployed for image classification. A CNN provides benefits over a Multilayer Perceptron (MLP), such as the ability to understand that image pixels in close proximity are more related than those that are further apart [14].

Transfer learning is used in the algorithm, allowing the model to benefit from the learned understanding of CNN architectures. An assessment was carried out before transfer learning was carried out, which would define how it would be applied. The size of the dataset is small, relatively speaking, and there are similarities between the dataset to be used, and images are present on the ImageNet database. As a result, the approach applied to the model was to slice off the end of the neural network and add a new fully connected layer with a resulting number that matches the number of classes. As this is classification task, there will be a binary output in the final layer.

Five different transfer learning models were used. These models and their details can be seen below in Table 1. Each model was and made predictions on the testing set once, using comparable set parameters. This delivered a relative performance rating based on the competition submission score, indicating which models would be best suited for classification and refinement.

Table 1: Documentation for Individual Models
[9]

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
Inception V3	92 MB	0.788	0.944	23,851,784	159

To optimise the classifier, the following parameters will be tuned and altered:

- Network architecture:
  - Pre-trained model architecture.
  - Number of layers.
  - Type of layers.
  - Parameters for the layers.
- Training parameters:
  - Epochs - how many epochs directly effects the training time.
  - Batch size - how many examples to look at before making a weight update.
  - Optimizer - which optimizer to use to minimise the objective function (Adam, SGD, etc).
  - Learning rate - how quickly should the model learn and should it have a decay rate.
  - Momentum - prevents the model becoming stuck in a local minima. It takes the previous learning step into account when making the next one.
  - Nesterov - smarter application of momentum.
- Image Augmentation and preprocessing

Using a Convolutional Neural Network requires a large amount of training data. As discussed in Section 2.2, the training dataset contains 1836 image files, too small to adequately train the neural network on alone. Image augmentation is an important

tool in avoiding the overfitting of data through invariant representation of the images. The parameters to be tuned and used in various combinations will be discussed in Section 3.1.

2.4 Benchmark

The data set and task is based on the Kaggle Competition, Invasive Species Monitoring - Identify Images of Invasive Hydrangea. The public Leaderboard, ranked based on the evaluation criteria, will be used as the foundation for benchmarking my model. Submissions are evaluated on the area under the ROC curve between the predicted probability and the observed target, as discussed in Section 1.3. The Kaggle Public Leaderboard, which has 1794 entries, was downloaded, and this data was used to calculate benchmark values evaluated on the area under the receiver operator curve:

- Average performance Kaggle Leaderboard = 0.902378093
- 80th percentile performance Kaggle Leaderboard = 0.98887

Based on these values, my model should aim to achieve greater than the average performance present on the Kaggle Leaderboard of 0.9023. I will set a stretch target of trying to achieve performance that meets or exceeds the 80th percentile of the Kaggle Leaderboard.

3 Methodology

3.1 Data Preprocessing

The pre-processing carried out on the image datasets before model fitting consisted of the following steps:

1. Training images are shuffled, and then split into a training set consisting of 80% of the original images, and a validation set consisting of 20% of the original images.
2. The input images are converted into an array and then resized into a 4D tensor for use in a Keras CNN. The image size (rows x columns) can be adjusted

based on the pre-trained model being used. The channels are equal to three, as the images are in colour. The tensor has the form:

$$(nb\_samples, rows, columns, channels)$$

3. Each image is rescaled and converted to a 32 bit float.
4. The training images undergo image augmentation. Eight different transformation features were used.

One of the most important aspects of the pre-processing steps relates to Step 4 - image augementation. Eight different transformation methods were employed, as may be seen below. Image augmentation prevents the model from ever seeing the same image twice, and reduces the risk of overfitting while providing better model generalisation [4].

The following image augmentation techniques were used on the training images:

- rotation_range - degree range for random rotations.
- width_shift_range - float fraction range of image to shift.
- height_shift_range - float fraction range of image to shift.
- shear_range - float shear intensity.
- zoom_range - float range for random zoom.
- horizontal_flip - boolean to randomly flip the image horizontally.
- vertical_flip - boolean to randomly flip the image vertically.
- fill_mode - points outside the boundaries of the input are filled to given mode.

3.2 Implementation

Through implementation, there were two clear stages that the body of work was broken into:

- Benchmarking all five pre-trained applications against each other.
- Selecting the two most suitable features for further refinement.

Benchmarking was achieved by training all five models, seen in Table 1, on the same parameters and then comparing the score achieved on the Kaggle submission page.

The baseline parameters used can be seen in Table 2 below:

Table 2: Baseline Model Parameters

Baseline Parameter	**Value**
Image Shape	224, 224, 3
Dropout Layer	0.5
Epochs	20
Batch Size	32
Optimizer	SGD
Learning Rate	1.00E-0.4
Momentum	0.9
Loss Function	Binary Cross Entropy

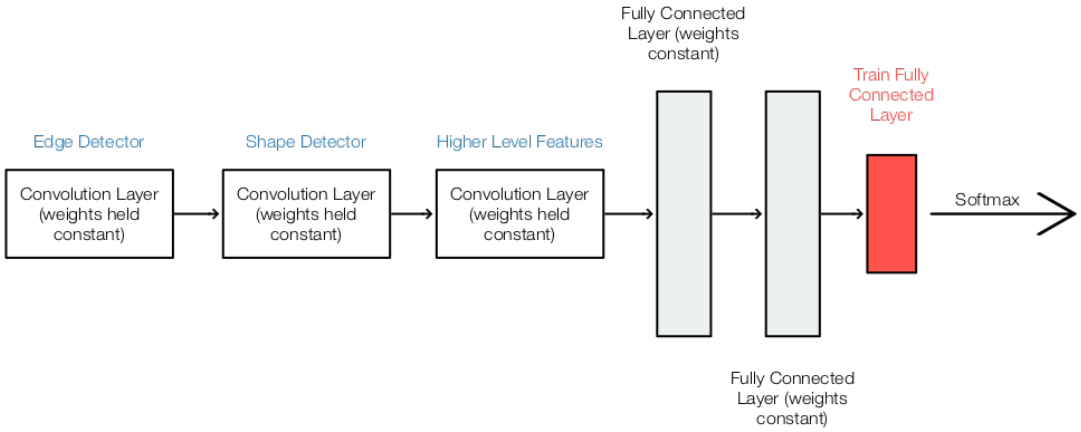
Initially, I omitted momentum from the training of my models. This resulted in the subsequent models becoming stuck in a local minima, seen as the validation accuracy (0.6623) and validation loss (5.3836) improved no further. I adjusted the models to include momentum. Momentum adds inertia to the descent, acting as both a smoother and an accelerator. It dampens oscillations and allows continuous travel through narrow valleys, and out of small bumps and, in this case of this model, local minima [7]. One of the most commonly used values for momentum is a beta of 0.9 and above when paired with SGD, so this is the value I chose to begin with for my model [2]. Using this value corrected my issue, and allowed validation loss and validation accuracy to continue to improve with iterations.

One of the limiting factors when using a neural network is the time to train. Using a combination of three separate workstations, and Amazon Web Services, an epoch with the baseline parameters seen in 2 , and image augmentation discussed in Section 3.1 could range from 1400 to 3000 seconds per epoch. Over a 50 epoch run, this could take a considerable amount of time and resource.

The data set is small and similar to the ImageNet Database, therefore the end of the neural network was removed, a new fully connected layer was added that matched the class output (binary), the weights of the new layer were randomised and the pre-trained layer weights were frozen. The network was trained to allow the weights

of the new fully connected layer to be updated. The illustration shown below in Figure 2 displays the network architecture, and adaption of the neural network to the dataset.

Figure 2: Pre-Trained Architecture



The steps that were followed in the implementation of the classifier training stage can be summarised below:

1. Training, validation and testing image datasets are loaded into memory.
2. The images are pre-processed into tensors, and re-scaled.
3. Correct input shapes for each of the datasets is checked.
4. The required bottleneck features for each of the pre-trained models is acquired.
5. The network architecture is defined.
6. The optimiser and optimiser parameters are set.
7. A function is implemented to save the best performing weights after each epoch.
8. Image augmentation is carried out.
9. The training parameters are set.

10. The network is trained with the best validation loss stored.

11. The best weight for the chosen architecture is loaded, predictions are carried out and a submission file for the competition is created.

Table 3 displays the relative scores for each of the pre-trained models using the baseline parameters from Table 2, and the image augmentation techniques discussed in Section 3.1.

Table 3: Baseline Model Scores

Model	**Kaggle Submission Score**
VGG16	0.98875
VGG19	0.9877
Xception	0.95969
Inception V3	0.96715
ResNet50	0.98157

3.3 Refinement

VGG16 was the best performing model, as seen previously in Table 3, and was selected for further refinement. Inception V3 was the best performing model with respect to low complexity (depth and number of parameters), and was also selected.

To improve on these scores, a number of changes were made to each model iteratively. VGG16 went through seven stages to achieve its score. Inception V3 had three iterations. The baseline and final tuned scored can be seen below in Table 4.

Table 4: Baseline Vs. Tuned Model Scores

Pre-Trained Model	**Kaggle Score - Baseline**	**Kaggle Score - Tuned**
VGG16	0.98875	0.99202
Inception V3	0.96715	0.98952

A number of parameters were altered and used in various combinations to provide the best submission score. The following parameters and techniques were used to achieve this:

- Epochs - the number of epochs was increased from 20 to 50. This was seen as the optimum level based on validation accuracy and validation loss, as may be seen in Figure 3.
- Batch size - batch size was varied between 16, 32 and 64. 32 was selected as the ideal value, based on results. It has been observed that larger batch sizes result in a notable drop in the quality of the model, measured by its ability to generalise [10].
- Data augmentation and tuning - an important step to avoid overfitting. Guidance for parameters to use and values to set were taken from the article written by Francois Chollet [4] and the Kaggle competition winner James Requa [13].
- Input image size - the image size was increased from 224 x 224 pixels to 400 x 400 pixels. While this significantly increased training time, it resulted in an increase in accuracy and a decrease in loss.
- Optimizer choice - optimizers trialled were SGD and Adam.
- Learning rate - set to 1.00E-04.
- Momentum - maintained at 0.9, as discussed previously.
- Nesterov - set to True for the final tuned model.
- Dropout layer - was set to 0.5, as this is found to result in maximum regularisation [1].

Figure 3: VGG16 Training Loss and Accuracy - Run 7

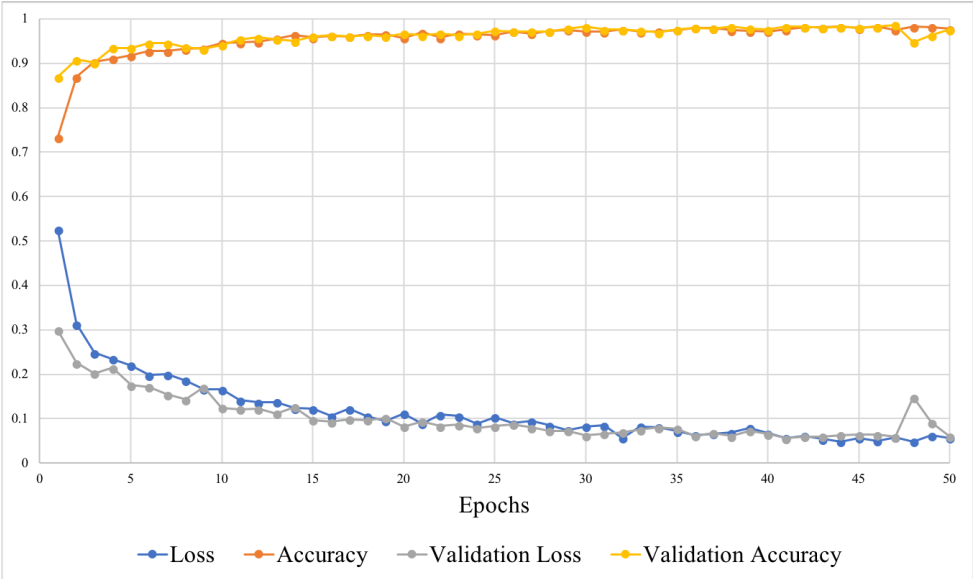


Figure 3 is the best performing VGG16 model. This final model was achieved by using the aforementioned iterative training steps and adjusting the parameters based on the performance of the model, which was observed in plots such as the one above. The final submission score of the model was 0.99202. This exceeds both benchmarks targeted in Section 2.4.

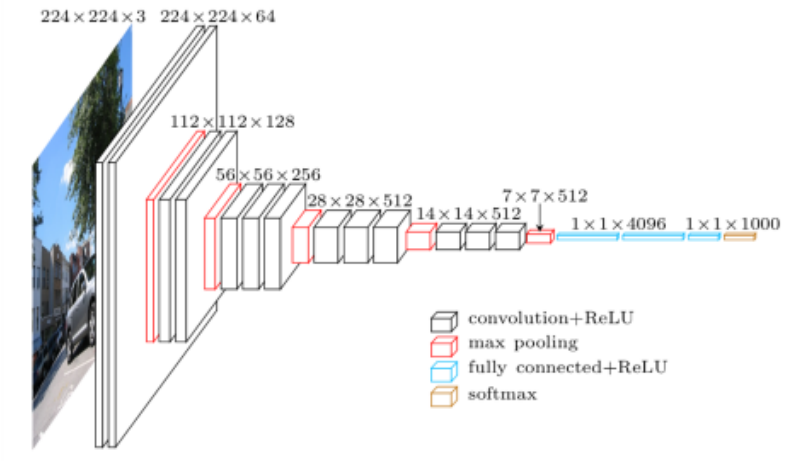
4 Results

4.1 Model Evaluation and Validation

During training, a validation set was used to evaluate model performance after each step in each epoch.

The final choice for both the architecture and the parameters used were chosen as they delivered the highest submission score. In total, fourteen different permutations of architectures and parameters were trialled. The final model employed the VGG16 pre-trained architecture, as may be see below in Figure 4.

Figure 4: VGG16 Architecture



[5]

This model combines the pre-trained VGG16 architecture seen in Figure 4, with a new fully connected layer to be trained at the end, as seen in Figure 2. The full list of parameters used for this model are as follows:

- The input image size was set at (400,400), a higher resolution than the default (224,224) used for VGG16.
- The total epochs run were set at 50.
- The batch size used was 32.
- The steps per epoch were set to the amount of training images divided by the batch size, resulting in 57 steps per epoch.
- A dropout layer of 0.5 was used.
- The final chosen optimiser used was SGD.
- Momentum was set to 0.9, with Nesterov set to True.
- The learning rate was set at 0.0001.

The code for the fully connected layer to be trained at the end of the VGG16 architecture can be seen below:

```
VGG16_model = Sequential()
VGG16_model.add(Flatten(input_shape = ...
VGG16_bottleneck_features.output_shape[1:]))
```

```
VGG16_model.add(Dense(256, activation = 'relu'))
VGG16_model.add(Dropout(0.5))
VGG16_model.add(Dense(1, activation = 'sigmoid'))
```

The robustness of the model was tested and verified by the testing set and the resulting submission score. The testing set had a number of images which would test the robustness of the model and would be reflected in the submissions score. Some examples of images which would affect the models score are:

- Pictures of buildings with no forest or foliage present.
- Blurry images with a person taking up over 50% of the image space.
- Images of lakes.
- Mixed light condition images. See Figure 5.
- Scrambled images with no clear discernible features. See Figure 6.

Figure 5: Mixed Light Condition

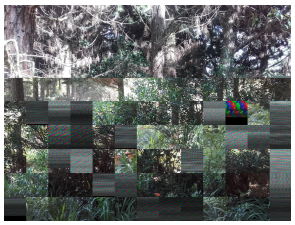


Figure 6: Scrambled Condition

Based on the submission score of over 99% on 1531 testing images, the following observations can be made:

- The classifier can reliably detect in the cases of both high, low and mixed light conditions.
- Blurry images, or partly obscured images, do no result in false classification.
- Scrambled images do no negatively affect classification.

4.2 Justification

Using VGG16, optimised and refined over seven runs, the following results for submission on The Kaggle Competition versus the original benchmark targets were

achieved. VGG16 achieved a submission score of 0.99202. This beats both the average Kaggle Leader Board performance target of 0.90238, and the stretch target of the 80th percentile, which was a score of 0.98887.

This score would not have been possible to achieve without the image augmentation data processing step. The application is ideal for this competition, returning over 99% success in classification. It accurately and quickly (when compared to the human eye) classifies invasive hydrangea is present.

However, should this application be applied to other classification tasks, re-tuning and re-structuring of the architecture, along with further training examples may be required. It perhaps performs as well as it does due to the fact that this is binary classification, and is only looking for one instance of an object in the images.

5 Conclusion

5.1 Free-Form Visualisation

As discussed in Section 4.1, the reliability and robustness of the classifier is high based on the relative submission score it achieves. One way to confirm this is to observe visually several photos from the testing set, and if it correctly identifies what it is seeing.

Below are three samples, with their relative probability displaying correct behaviour from the classifier:

- Figure 7 shows the invasive species hydrangea. The classifier has rated the probability of the plant being present as 1, which is 100% probability.
- Figure 8 is the lowest ranked probability of hydrangea being present in the testing set at 0.000170888. As can be seen, there is clearly no presence of the invasive plant in the image. It is also ironically image 404.
- Figure 6 is the previously mentioned scrambled image, which is used to test the robustness and reliability of the classifier. It has rated the probability of hydrangea being present as 0.001392973, which on close inspection, appears correct.

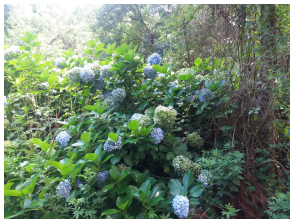


Figure 7: High Probability

Figure 8: Low Probability

5.2 Reflection

The steps taken in the process to successfully complete this project can be summarised as follows:

1. The problem and datasets were found via the Kaggle Competition - Invasive Species Monitoring.
2. The data was downloaded, split into training, validation and testing sets and the data was pre-processed.
3. A benchmark was defined using the competition evaluation criteria and the competition public leader board.
4. Five baseline classifiers were trained and from these five, two were selected for further refinement.
5. VGG16 and Inception V3 were trained multiple times, with different parameters, until a score greater than the benchmark target was exceeded.
6. VGG16 was selected, providing a submission score of 0.99202.

The most interesting aspect of the project was successfully training multiple models on the data, optimising the parameters, and observing incremental improvement. It enabled me to learn a great deal about CNNs through trial and error, and has expanded my knowledge and my curiosity.

The most difficult aspect of the project concerned data pre-processing to achieve the correct tensors, datasets and image augmentation methods. One difficulty mentally was the time taken to train the models. As this could be quite long using the resources available to me, it could be time consuming tweaking and waiting for the resulting best weights to be generated.

In summary, the final model chosen does fit the expectations and requirements of

the competition problem statement. CNNs in general work extremely well in most image classification problems and the use of a pre-trained model allows the years of model architecture design and ImageNet learning to transferred very easily for use in these type of problems.

5.3 Improvement

It would be difficult to improve this model further then the 0.99202 score achieved, however, the top public score on the leader board obtained a score of 0.99770, so improvement is possible. Suggested steps that could be taken to improve on the score my model obtained are:

- Introduce hyper-parameter tuning automation
- Examine in great detail the time and resource trade off versus score achieved
- Improve on the manual coding of the CNN architecture and image augmentation process, giving greater customisation and capability.

Based on the top score on the public leader board, a better solution does exist. An argument would have to be made, however, about the benefit of spending time and resource on improving the model further to obtain marginal gains (approximately 0.5%) using my final score as the new benchmark.

References

[1] Pierre Baldi and Peter J Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc., 2013.

[2] Vitaly Bushaev. Stochastic gradient descent with momentum. online, 2017.

[3] Chioka. Class imbalance problem. *Chioka.in*, 2013.

[4] Francois Chollet. Building powerful image classification models using very little data. online, 2016.

[5] Matthieu Cord. Deep cnn and weak supervision learning for visual recognition. online, 2016.

[6] Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27:861–874, 2006.

[7] Gabriel GOH. Why momentum really works. *GOH, 2017*, 2017.

[8] Kaggle. Invasive species monitoring. *Kaggle*, 2017.

[9] Keras. Documentation for individual models. *Keras*, 2018.

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

[11] Christian Requena Mesa, Thore Engel, Amrita Menon, and Emma Bradley., 2017. Data from Brazilian rainforest. Hydrangea invasive species.

[12] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: An astounding baseline for recognition. In *CVPR Workshops*, pages 512–519. IEEE Computer Society, 2014.

[13] James Requa. 1st place solution overview lb 0.99770. online, 2017.

[14] Udacity. Machine learning nanodegree course. online, 2018.

[15] Wikipedia. Receiver operating characteristic. *Wikipedia*, 2018.