Identifying Characteristics of False Positives from a Flower Detection Network

Isa Lykke Hansen

May 28th 2020

Abstract

This will soon be a paper about image classification ... I hope.

Contents

1 Introduction

The method of using neural networks (NNs) to classify images have grown steadily in popularity over the last couple of years. These days, image classification algorithms help us solve diverse tasks such as detecting oil spills, improving early cancer detection, predicting the weather and even exploring the distant corners of our universe (Abhishek et al., 2012; Ekici and Jawzal, 2020; Krastev, 2020; Vestfalen, 2019).

Since 2018 researches at Universitetet i Tromsø (UiT) and Aarhus University (AU) have collaborated on a project to track the plant-pollinator interaction in selected species of Arctic flowers. This is done to investigate whether climate change is shifting the active seasons of the flowers and pollinating insects. Such a shift could potentially have very serious consequences, as it is ultimately the overlap between the pollinators and flowers that determine the outcome of food crops (Forskningsfond, 2019).

The project implements large scale video-surveillance of the plant-pollinator interaction by way of time-lapse images. For two years, time-lapse data was gathered from several cameras set up at various locations in the Arctic, collecting thousands of images for analysis. The idea then is to train a NN to automatically detect flowers and insects in the images. This creates the possibility for investigating changes in large areas and over long time periods with minimal amounts of equipment and manual labour.

1.1 Image Recognition

Broadly speaking, image recognition tasks can be divided into three sub-categories ("C4W3L01 Object Localization", 2017):

1. Classification
2. Classification with localization

3. Detection

When searching an image for an object, X, 1 answers the simple question "is X present in the image?". 2 in addition provides information on where in the image X is located. The exact location of X is specified by a so-called *bounding box*, which is parameterized by a vector containing the x,y coordinates for the midpoint of the box, as well as the height and width of the box. Finally, 3 allows for the localization of multiple objects from several classes within a single image. The output will be an array of bounding boxes each associated with a class and a probability for that class.

1.2 A Flower Detection Network

The images used in this paper were detections from a MASK R-CNN trained on data from four observation sites/periods (Narsarsuaq 2018, Narsarsuaq 2019, Thule 2018 and Ny Aalesund, Svalbard 2019). The network was trained on rectangular bounding boxes and obtained a precision of .84 and a recall of .92.

Thus, to put it more precisely, the images used here were crops created from the bounding boxes predicted by the network. The purpose of this paper is to investigate whether structural differences exist between true positive (TP) and false positive (FP) predictions from the network.

2 Methods

All scripts used for analysis can be found on the Github repository https://github.com/isalykke/Data-Science-Exam-2020

2.1 Sorting Images

The first task consisted of manually sorting the images into true- and false positives. From the original set of 7.420 images, 6.685 were man-

Figure 1: Example image from the group true positives

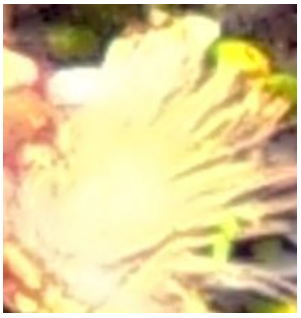


Figure 2: Example image from the group false positives

ually sorted into two groups of TP(n = 4.345) and FP(n = 2.340). The 735 images that could not clearly be defined as either TP or FP were excluded from the analysis. Example images from the two groups can be seen in in figures 1 and 2 respectively.

2.2 Extracting Meta Data

Secondly, meta data were extracted from all images using the script "extract_metadata.py". The script runs through the image directories and extracts image features for each image, before saving them to a .csv file for further statistical analysis (see section 2.3). Features

extracted for each image included:

- Filename, location and label (TP/FP)
- Simple features such as height, width, size and ratio
- Complex features such as blurriness and complexity measures

Blurriness was calculated as variance of the Laplacian, a widely accepted standard in the field of image recognition (Bansal et al., 2016). The Laplacian filter performs edge detection, and the idea is that the sharper an image is, the more edges it will contain. In other words, in sharp images the variance of the Laplacian will be high, as there will be large differences in the gray-scale values of neighbouring pixels. Conversely, when the variance is smaller, we can interpret the image as being "blurry".

For complexity, since there is no widely accepted standard measure in the field, we calculated two different versions, in the hope that they would be able to explain some of the variance in our data. In sections 2.2.1 and 2.2.2 below we go more in depth with each of these.

2.2.1 The Q Complexity Measure

For the calculation of the Q complexity we implemented the method described in (Zanette, 2018). In short, the method creates several resized versions of a B/W image with dimensions NxM. These are then further divided into boxes of size LxL[1] and the sample variance of the gray-level for each box, b, is calculated as:

$$V_b = \frac{1}{L^2 - 1} \sum_{i\,in\,b} (g_i - \bar{g}_b)^2 \tag{1}$$

where g_i denotes the gray level of each pixel and

[1]In theory L could be any number as long as its smaller than the dimensions of the resized image. In practice though, the best results are obtained for smaller values of L. This makes intuitive sense as less information is aggregated as L grows smaller. In this paper, as well as in the original, L was kept at 2.

$$\bar{g}_b = \frac{1}{L^2} \sum_{i\,in\,b} (g_i) \tag{2}$$

The sample variance for each box is then combined to find the gray-level mean variance of the resized image:

$$V = \frac{L^2}{N'M'} \sum_{b} (V_b) \tag{3}$$

Where N'M' are the dimensions of the resized image. This process is repeated for all resized versions of the original image, and the associated scaling factor, S, is calculated as:

$$S = \frac{LN}{N'} = \frac{LM}{M'} \tag{4}$$

Finally, the complexity measure, Q[0:1], for the B/W image is calculated as:

$$Q = \frac{1}{s_{max} - s_{min}} \int_{s_{min}}^{s_{max}} \left[1 - \frac{1}{4} \left(\frac{dv}{ds} \right)^2 \right]_+ ds \tag{5}$$

Notes on Implementation Because the code used for calculating Q in the Zanette paper was unavailable, we implemented the method in python(v3.8.2), by writing the functions **gray_level_mean_variance** and **Q_complexity** which can be found in the script 'extract_metadata.py'. Using this method we were able to very closely replicate findings from the original paper (see Appendix A). In the sections below we elaborate on some of the choices that were made during implementation of the Q complexity measure.

. In the original paper a series of values for N' were chosen to match the dimensions of the original image while at the same time maintaining the aspect ratio, so that $N'/M' = N/M$. As all of our images differed in dimensions, we instead implemented a method that downsized the images according to a percentage-wise scaling factor,

P, such that $N' = N \cdot P/100$. The values of P were given by the set {3,4,5,7,10,12,15,17,20,25,30,40,50,60,70,80,90}. This we believe, makes the method more broadly applicable, while at the same time still satisfying the requirement of maintaining the aspect ratio.

. In addition, as 1 requires the division of images into 2x2 boxes, and all our images were of varying sizes, we decided to crop the images prior to the calculation of V_b. In practice, this was done by downsizing the dimensions of the resized image by one on the instances where the height or width did not satisfy $x \, mod \, 2 = 0$.

. Finally, as described in the article, in order to calculate 5 we need to integrate over the derivative dv/ds. Since we have only calculated discrete values of V(S) an interpolation function is required for the calculation of the derivative. Here we used zero-smoothed Bspline interpolation of the fourth degree utilizing the scipy funcitons **BSpline** and **splrep** (Virtanen et al., 2020).

2.2.2 The IC_{LS} Complexity Measure

The IC_{LS} is another proposed measure of complexity described in (Yu and Winkler, 2013). The IC_{LS} measures the compression ratio, CR, as the file sizes of uncompressed images relative to the file size of the compressed images, such that:

$$CR = \frac{s(I)}{s(C(I))} \tag{6}$$

Here, $s(I)$ denotes the file size (measured in bytes) of the uncompressed image and similarly, $s(C(I))$ is the file size of the image after compression. The lossless compression ratio of the image, IC_{LS}, can then be calculated as:

$$CR = \frac{1}{CR} \tag{7}$$

Notes on Implementation Similar to the Q-measure, calculation of the IC_{LS} was implemented in python(v3.8.2) via the function **ICLS_complexity** defined in 'extract_metadata.py'. Below are some notes on the implementation of the IC_{LS}.

. The name "lossless compression ratio" is rather ill-fitting in our case, as the images we are investigating have already been compressed to .jpgs prior to analysis. Since we did not know the exact compression rate of the images, but were told they had been saved with standard jpg formatting, we accepted this as being roughly equivalent to lossless compression. In any case, since IC_{LS} is a ratio the true size matters less than the differences between sizes. We calculated three measures of the IC_{LS}, namely the IC_{LS10}, IC_{LS30} and IC_{LS50} where C(I) was set to 10, 30 and 50, respectively.

. The original paper uses gray-scale versions of the images for the calculation of IC_{LS}. Here we chose instead to use the full RGB versions of the images, as we wished to maintain as much information from the original images as possible.

2.3 Statistical Analysis in R

Meta data were extracted for a total of 6.683 images[2]. For 73 of these (FP = 60, TP = 13), the calculation of Q resulted in NAs. In order to include as much data as possible in the model selection, in these instances we chose to assign the group average as Q.

2.3.1 Model Selection

In order to investigate which combination of the predictors was most effective in predicting false positives from the network, we ran a model selection process to asse

[2]Two of the FPs were excluded due to errors in the computation of Q

Multicollinearity Checks In order to decide which parameters to include in the model selection we did a multicollinearity check of all predictors, the results of which can be seen in Figure 5, Appendix B.

Since the three IC_{LS} measures were generally highly correlated with each other (all $\rho > .6$) we chose to include only IC_{LS50} in the analysis, since this parameter also had the lowest correlation with the width, height and size parameters. Similarly, and perhaps unsurprisingly, the width and height parameters were also both highly correlated ($\rho > .8$) with the size parameter. For simplicity, and to lower the computational cost, both of these were therefore left out of the final model selection. The 6 parameters included in the final model selection process were:

- Size
- Ratio
- Blurriness
- IC_{LS50}
- Q-complexity
- Location

Cross Validation In order to find out which model best predicted the data, we ran a series of cross validations using the R packages groupdata2 (Olsen, 2020b) and cvms (Olsen, 2020a). Model formulas were determined with the function **combine_predictors** which allows for unique combinations of all predictors. Due to computational restraints[3] we only allowed for up to two-way interactions. We ran 10-folds cross validation on each of the 6.214 model combinations and extracted the best 1.000 models as ranked by a balanced accuracy measure. These were then run though a 10-folds 5 times repeated

[3] . . . and because no-one understands what a 2+ way interaction is, anyway. . .

Table 1: hej

Model	Balanced Accuracy	AUC
$blur * location + blur * ratio + blur * size + location * ratio + Q * size + ratio * size$	0.823	0.893
$blur * location + blur * size + location * ratio + location * size + Q * size + ratio * size$	0.823	0.893
$blur * location + blur * size + location * ratio + Q * size + ratio * size$	0.822	0.893
$blur * location + blur * Q + blur * size + location * ratio + Q * size + ratio * size$	0.822	0.892
$blur * location + blur * size + location * ratio + Q * ratio + Q * size + ratio * size$	0.821	0.893
$blur * location + blur * ratio + blur * size + location * ratio + Q * size$	0.821	0.889
$blur * location + blur * ratio + blur * size + location * ratio + location * size + Q * size$	0.821	0.889
$blur * location + blur * size + location * Q + location * ratio + Q * size + ratio * size$	0.821	0.895
$blur * location + blur * Q + blur * ratio + blur * size + location * ratio + Q * size$	0.821	0.888
$blur * location + blur * ratio + blur * size + location * Q + location * ratio + Q * size$	0.820	0.890

cross-validation. The top five models from this process along with their Balanced Accuracy score can be seen in Table 1.

Animal	Description	Price ($)
Gnat	per gram	13.65
	each	0.01
Gnu	stuffed	92.50
Emu	stuffed	33.33
Armadillo	frozen	8.99

2.3.2

However not there are more measures of a good model than accuracy ælkaflkdf something about being simple fewer predictors etc

2.4 CNN

As a proof of concept a simple CNN was implemented to distinguish between positives and FPs

3 Results

We found that positives could best be distinguished from FPs with the linear model

4 Discussion

Due to time constraints we were not able to... In future analysis of the data one might include more measures of ... Another thing that might be interesting to investigate is the possibility to add the NN to the original R-CNN to see if it leads to fewer FP. However, layering lsksdlk

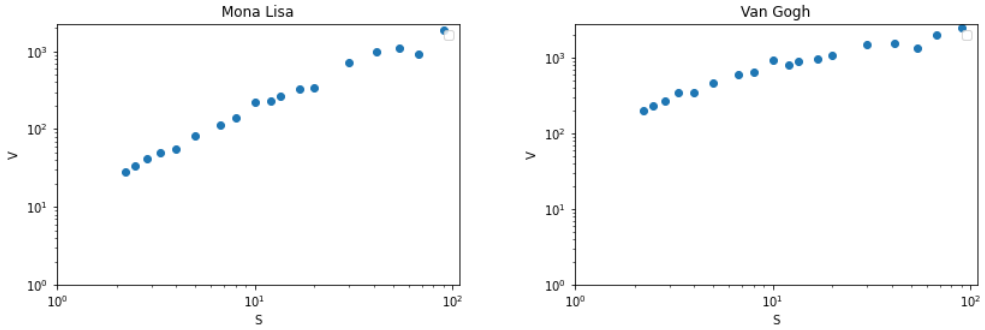
Serial Considerations During any test/train regime such as the cross-validation performed here it is important that the is no leakage of information between the training and testing sets. This is to make sure that the model is not trained on the same data it is evaluated against, as this would lead to over-fitting. The image data presented in this article could be argued to be serially correlated (so-called *autocorrelation*), as it consists of images of the same flower over time. Therefore, in principle, it is important to make sure that images containing the same flower are kept in separate folds during cross-validation. However, due to the way the original images were encoded we did not have this information available, which unfortunately causes our results to be less robust. Future studies should strive to include time-sensitive measures in the analysis so as to avoid over-fitting and increase generalisability of the results.

5 Conclusion

In this paper we investigated systematic differences between true- and false positives from a MASK R-CNN detecting flowers from time-lapse images. We implemented novel measures of complexity in python and were able to replicate results from an original paper. Using both simple and complex image features we ran large scale cross validations in order to find the model parameters that best described the variance in our data.

A The Q Complexity Measure

In Figure 3 below we present two example variance profiles of the images from (Zanette, 2018) recreated using the methods described in 2.2.1. Compare these to the original variance profiles seen in Figure 4.



(a) (b)

Figure 3: Variance profiles of the a) Mona Lisa and b) Van Gogh images recreated in python

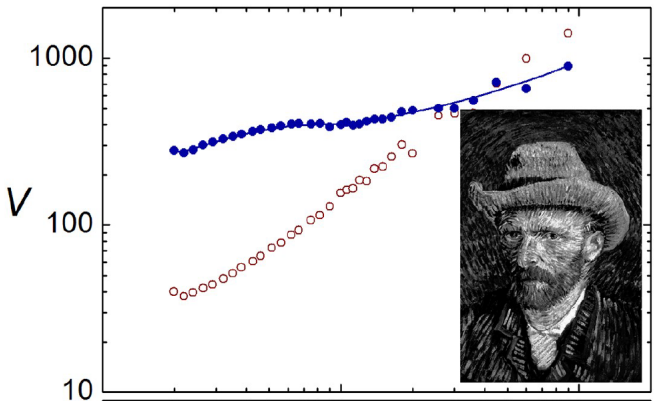


Figure 4: Original variance profiles of the Mona Lisa (red) and Van Gogh (blue) images. Image is an exempt from (Zanette, 2018)

B Multicollinearity Check

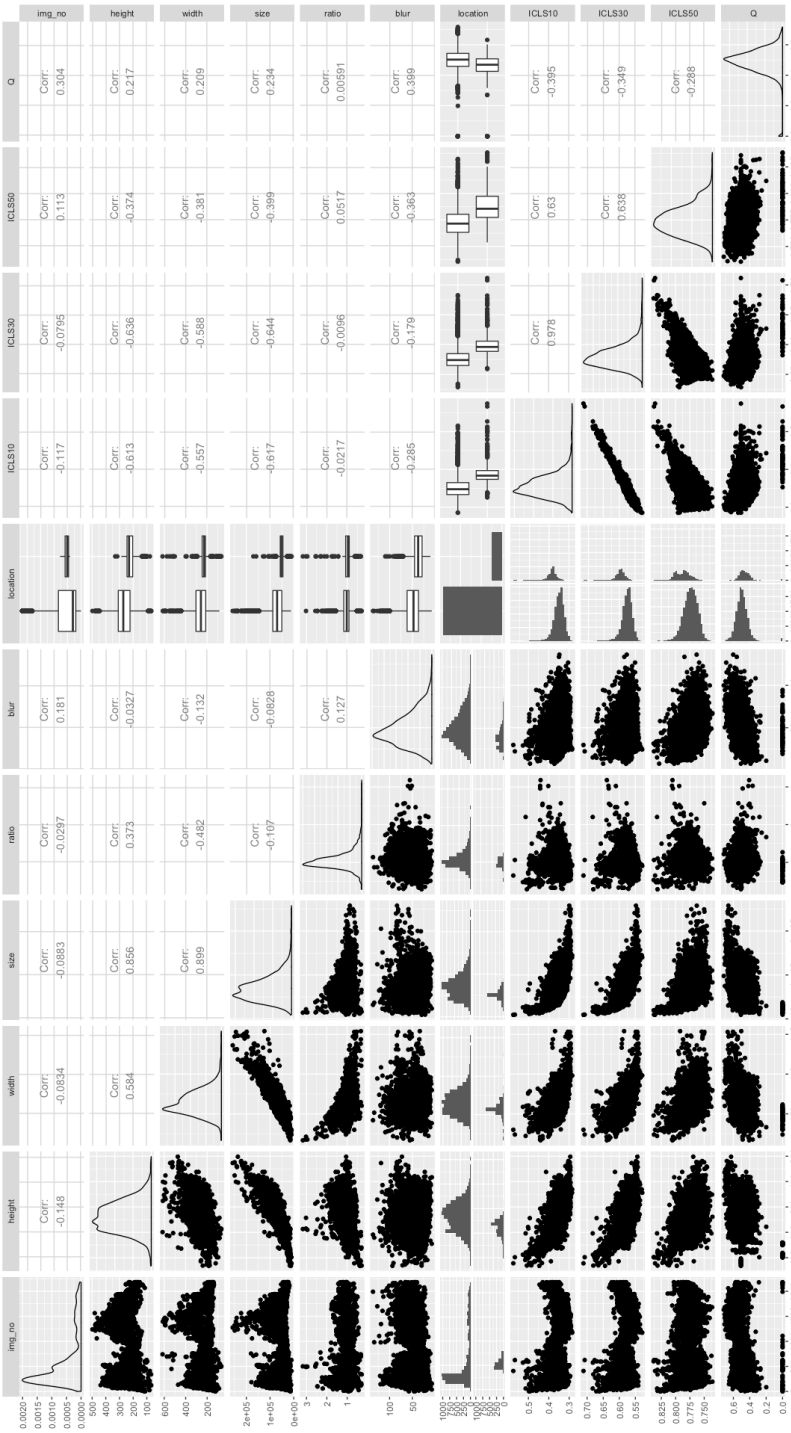


Figure 5: Multicollinearity matrix for the metadata parameters