Identifying Characteristics of False Positives from a Flower Detection Network

Isa Lykke Hansen

May 28th 2020

Abstract

In this paper we investigate systematic differences between true- and false positives from a MASK R-CNN detecting flowers of the species *Dryas integrifolia* from time-lapse images. We implement novel measures of image complexity in python and combine them with other image features to see which of these are most useful for detecting false positives.

Running large scale cross validations we are able to find a model which predicts 77% of the the total variance in the data but which does not generalize well across locations.

We discuss considerations that should be taken into account when further investigating the data, including how to improve the model and suggestions for alternative methods of analysis. In addition, we comment on the usefulness of the complexity measures in combination with other image parameters and express optimism that these could be useful in classifying false positives from the MASK R-CNN in the future.

Finally, we conclude that an intricate knowledge of the data is crucial in any modelling task and that future research should take such knowledge into account when designing solutions to distinguish between true- and false positives.

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 ({Danmarks Frie 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.

Image Recognition Broadly speaking, image recognition tasks can be divided into three sub-categories (Deeplearning.ai, 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.

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 namely,Narsarsuaq 2018, Narsarsuaq 2019, Thule 2018 and Ny Aalesund, Svalbard 2019. The network was trained to predict rectangular bounding boxes from images of the plant species *Dryas integrifolia* (see Figure 1) and obtained a precision of .84 and a recall of .92.

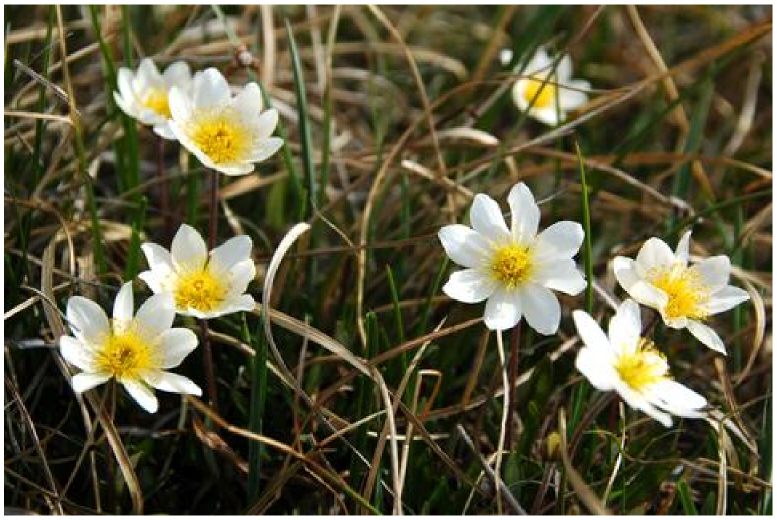


Figure 1: *Dryas integrifolia.* Image from ("Dryas Integrifolia", 2020)

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- and false positive predictions from the network. Incorporating such knowledge into the network could potentially lead to an increase in the recall and precision.

2 Methods

All scripts used for analysis can be found on the Github repository https://github.com/isalykke/Data-Science-Exam-2020. Code can also be viewed in Appendixes D and E. Data can be accessed via the link https://1drv.ms/u/s!Ar7cNsdeTQo_ibRhifZl0sGDizAnmA?e=OWvxF2 using the password 'YourMothersMaidenName'.[1]

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 manually sorted into two groups of true(n = 4.345) and false(n = 2.340)positives. The 735 images that could not clearly be defined as either true or false positives were excluded from the analysis. Example images from the two groups can be seen in Figure 2.

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 (true/false positive)

[1]Expires on June 30th 2020, contact author if access is needed past this date

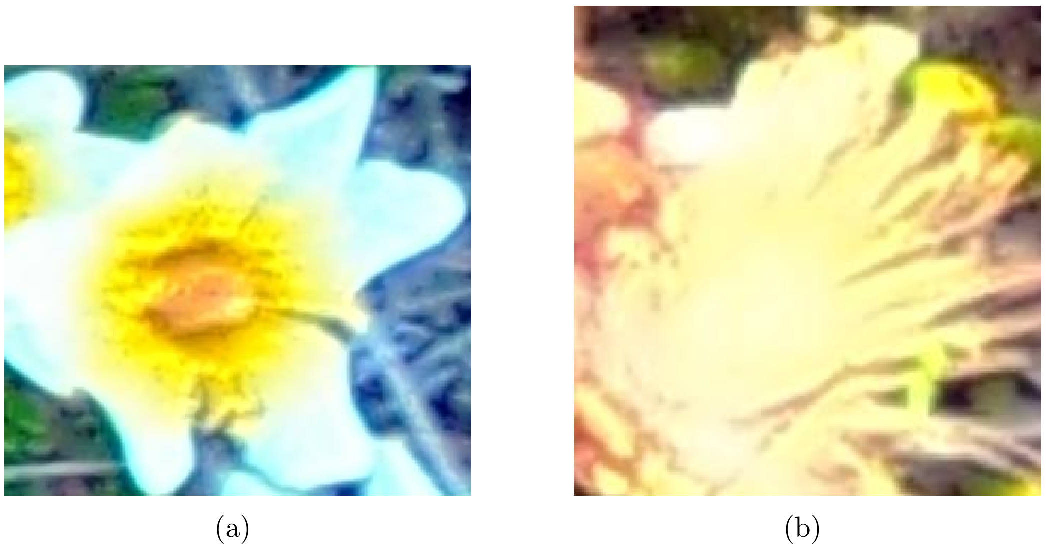


Figure 2: Example images from a) the true positive group and from b) the false positive group

- 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[2] 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

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

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:

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

[2]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.

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)({Python Software Foundation}, 2020), 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

All statistical analysis was carried out in R(v.3.6.1) ({R Core Team}, 2020). Meta data were extracted for a total of 6.683 images[3]. 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.

2.4 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 12, Appendix C.

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 cross-validation process. The 5 parameters included in the final model selection process were:

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

[3]Two of the false positives were excluded due to errors in the computation of Q

The VIF statistics for the final model were also calculated and can be viewed in Table 4, Appendix C.

2.5 Cross-Validation

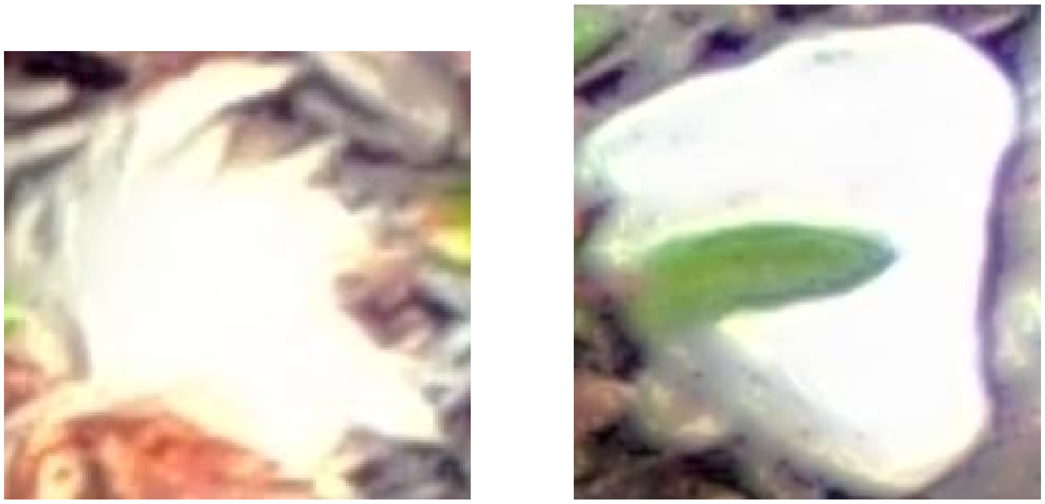
To evaluate which model best predicted the data, we ran a series of cross-validations using the R packages groupdata2 (Olsen, 2020b) and cvms (Olsen, 2020a). Model formulas were determined with the function **combine_predictors** which allows for unique combinations of all predictors. Due to computational restraints we allowed for a maximum of two-way interactions. We ran a 10-folds cross-validation on each of the 1.268 model combinations and extracted the best 100 models as ranked by a Balanced Accuracy measure. These were then run though a 10-folds 5-times-repeated cross-validation. The top ten models from this process along with their Balanced Accuracy scores can be seen in Table 1. The model with the highest Balanced Accuracy score was expressed as:

$$\begin{aligned}false\ positives \sim blur * Q + blur * size + ICLS50 * Q \\ + ICLS50 * size + Q * size + ratio * size\end{aligned}$$

The parameter estimates for the model can be seen in Table 2 and additional model statistics can be seen in the first row of Table 3.

2.6 Evaluating Predictions

Because our goal ultimately is to be able to classify images from various research stations, it is important that the selected model generalises to data from more than one location. As we observed systematic structural differences (see Figure 3) between the false positives from location one (Narsarsuaq, n = 5681) and two (Thule, n = 1002) we suspected that the model might not generalize well across locations. In order to test this we fitted the model to a subset of the data collected at location one and tested it against data collected at location two. The results of this comparison can be seen in the second row of Table 3.



(a) (b)

Figure 3: Representative examples of false positives from a) location one, Narsarsuaq and b) location two, Thule

3 Results

Table 1: The ten highest scoring models from cross-validation

Model formula ($false\ positives \sim$)	Balanced Accuracy
$blur * Q + blur * size + ICLS50 * Q + ICLS50 * size + Q * size + ratio * size$	0.766
$blur * Q + blur * size + ICLS50 * Q + Q * ratio + Q * size + ratio * size$	0.763
$blur * size + ICLS50 * Q + ICLS50 * size + Q * ratio + Q * size + ratio * size$	0.763
$blur * Q + blur * size + ICLS50 * Q + Q * size + ratio * size$	0.763
$blur * size + ICLS50 * Q + ICLS50 * size + Q * size + ratio * size$	0.763
$blur * ICLS50 + blur * Q + blur * size + ICLS50 * Q + Q * size + ratio * size$	0.762
$blur * Q + blur * size + ICLS50 * Q + ICLS50 * ratio + Q * size + ratio * size$	0.762
$blur * ICLS50 + blur * size + ICLS50 * Q + ICLS50 * size + Q * size + ratio * size$	0.762
$blur * size + ICLS50 * Q + ICLS50 * ratio + ICLS50 * size + Q * size + ratio * size$	0.762
$blur * ratio + blur * size + ICLS50 * Q + ICLS50 * size + Q * size + ratio * size$	0.761

Table 2: Model coefficients for the highest scoring model

Coefficient	Estimate	SE	p
Intercept	-69.62	19.57	<.001***
Blurriness	-0.04	0.02	0.02*
Q	99.61	38.95	0.01*
Size	-0.0006	0.00009	<.001***
$ICLS50$	100.50	24.19	<.001***
Ratio	5.03	0.45	<.001***
Blurriness:Q	0.12	0.03	<.001***
Blurriness:Size	-0.000006	0.00000009	<.001***
Q:$ICLS50$	-160.70	48.31	<.001***
Size:$ICLS50$	0.0004	0.0001	<.001***
Q:Size	0.0006	0.00003	<.001***
Size:Ratio	-0.00009	0.000007	<.001***

Note R^2 = .32 (Hosmer-Lemeshow), .34 (Cox-Snell), .46 (Nagelkerke)

Table 3: Model parameters for the highest scoring model trained on all data vs a subset of the data (see Section 2.6 for details)

Data	Balanced Accuracy	AUC	Lower CI	Upper CI
All	0.77	0.86	0.85	0.87
Subset	0.35	0.33	0.30	0.37

4 Discussion

4.1 Interpreting Results

Table 2 displays the model coefficients for the model that best predicts the differences between true- and false positives from the MASK R-CNN, as determined by the cross-validation process. We see that both Blurriness(-0.036), Q(99.61), Size(-0.0006), IC_{LS50}(100.50) and Ratio(5.03) were significant predictors of false positives.

Similarly, the interaction terms blur:Q(0.12), blur:size(-0.000006), IC_{LS50}:Q(-160.7), IC_{LS50}:size(0.0004), Q:size(0.0006) and ratio:size(-0.00009) were all significant predictors of false positives. However, according to the literature, given the presence of significant interaction effects the interpretation of main effects is meaningless (Field et al., 2012).

To better interpret their meaning, in Figures 6 though 11 in Appendix B the 6 different interaction effects of the model have been plotted. Recall that an interaction effect encodes a difference of differences for the effect of a predictor on an outcome. In other words, the significant interaction terms in our model suggest that these specific combinations of predictors differently affect the likelihood of an image being a false positive.

Taking as an example Figure 7, the interaction between the Blurriness and Size predictors, we see that for blurred images[4] false positives (blue line) are likely to be smaller than true positives (red line). For sharp images, this effect reverses, such that larger images are more likely to be false positives. Another way to say this is that false positives are likely to be more blurred than true positives *only if* the size of the image is lower than a certain threshold. As the image size surpasses that threshold, the effect reverses, and blurrier images are more likely to be true positives. We leave the interpretation of the remaining five interaction plots up to the reader.

[4]e.i. those that have a lower value for Blurriness

What the model coefficients tell us is that all five predictors (or combinations of these) included in the model were valuable in distinguishing between true- false positives.

Since we are working with machine learning, more than being able to explain the data we were also interested in the ability of our model to predict new data. Table 3 shows the predictive power of the model when it is fitted to all data and when it is fitted to a subset of the data. We see that all measures of model performance drop substantially when the model is fitted to data from a new location. In fact, the model that was fitted to the subset from Narsarsuaq was worse than chance when it came to predicting false positives from Thule (Balanced Accuracy = 35 % as opposed to 77% for the full dataset). This suggests that our model is very prone to over-fitting to one specific location. So while our model might be good at explaining variance in the data it seems it is not appropriate for predicting new instances of false positives.

Finally, as can be seen in Table 4, we note that despite our best efforts to exclude correlated variables prior to model selection, the model still suffers from multicollinearity. The results from it should therefore be interpreted with great caution, as we can not meaningfully distinguish the contributions of each of the different predictors.

4.2 Considerations for Future Investigations

Because of the exploratory nature of this study and due to time constraints much still remains to be said about these data. Below we go more in detail with some considerations for future investigations of the data.

Making a Better Model First of all, the predictors investigated here are far from exhaustive. As is the case with any model, the one presented here is a simplification of reality and is only allowed to explore the model space provided via its predictors. There could be (and most likely are) other factors not included here which would be even better at explaining the differences between the two groups. With that in mind we comment on some predictors that might be useful include in order to to make the model better at explaining and/or predicting false positives.

As previously noted, in order to most effectively distinguish between true- and false positives we desire a model that is able to generalise across all locations, and not over-fit to one location as we saw in Table 3. One way to solve this could be to include location as a random effect in a mixed effects model. Random effects are predictors we are "not interested in", meaning that they provide systematic variance in the data, that we want the parameters of the final model to be independent of. It might therefore be useful to include location as a random variable to let the model know that we want it to specify what distinguishes between the true- and false positives *regardless* of the location. In addition, to further improve upon the location predictor, one could imagine encoding the specific longitude and latitude of images, to make the measure more precise. This should be simple enough, as time-lapse cameras such GoPros have an inbuilt GPS system.

Furthermore, location independence is not the only desirable feature of a future model. Other features that might be included as random factors could be parameters such as weather conditions, as sun, wind and rain will greatly affect the images. This effect is not trivial to encode, however. *Dryas integrifolia* have a large head and a long thin stem (see Figure 1), which can cause them to shift easily in the wind, drastically changing the visual features of the flowers.

Similarly, the time of day/month might also be of importance. The position of the sun affects the position of the flower heads, due to a phenomenon known as *phototropism*, which causes plants to rotate towards the sun in order to maximize their surface for photosynthesis (Evert and Eichhorn, 2012). In many of the areas where the research takes place, the sun never sets during the flowering months, so a rotation around the head axis might be more likely, than opening and closing of the flower. All of this to say that the features that are fed into the initial NN greatly depend on local variables, which are useful to understand in order to model the variances in detections later on.

Another important point to make, is that the model should be trained on a more balanced dataset. Here it was the case that almost all of the false positives from Thule consisted of variants of the same image seen in Figure 3b, whereas the false positives from Narsarsuaq contained a much greater variety of images. This, combined with the fact that there were simply many more images from Narsarsuaq, lead to a model that greatly over-fitted to one location.

From all of this we see that knowing the specifics of the data you are working with is very important when it comes to designing models to effectively distinguish between the true- and false positives.

The Aspect of Time 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. Autocorrelation might also be part of the reason why we have seen such a large amount of over-fitting. Future studies should strive to include time-sensitive measures in the analysis so as to avoid over-fitting and increase generalisability of the results.

Evaluating Complexity Measures A large part of this article was dedicated to the implementation of new complexity measures, in particular the Q complexity measure suggested by (Zanette, 2018). This to investigate whether the complexity of images differed systematically between true- and false positives, meaning that the measures could be used to effectively distinguish between them. However, given the considerations described in section 4.1 we cannot with certainty conclude that the measures were effective in predicting such a difference.

In Figures 8 and 10 we also see that Q seems to suffer from a ceiling effect, where many of the images have been given a complexity score of zero. Theoretically speaking this would indicate that these images are either 100% random or 100% ordered, which, given our knowledge about the data, is highly unlikely. This effect is therefore more likely to be caused by errors in the calculation of Q. In future analysis, one might therefore consider excluding instances where Q is zero in order to not bias the outcome.

Finally we will note that given the significant interactions of both Q and IC_{LS50} with other model parameters (see Table 2) we remain hopeful that the measures could be useful in future investigations of the data.

Adding Anoter Layer As a final note, we suggest a different approach to effectively distinguish between true- and false positives predictions. Such an approach might be to simply train another CNN to differentiate between the two groups. If a high enough accuracy was obtained, information from such an additional network could be added as an extra layer to the already existing MASK R-CNN to improve performance. Importantly, all the considerations we have discussed so far with regards to the input data still apply, and, one might argue, now to an even greater extent. This is because the inner workings of a NN cannot be meaningfully investigated with methods currently available. Therefore, in order to to better understand the predictions it produces, it is highly important that we gain a deeper understanding of the data that we fed into the network.

5 Conclusion

In this paper we investigated systematic differences between true- and false positives from a MASK R-CNN detecting flowers of the species *Dryas integrifolia* 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 the data.

We found that our model was able to predict 77% of the the variance in the data but that it did not generalize well across locations. We commented on considerations that should be taken into account when further investigating the data, including how to improve the model and suggestions for alternative methods of analysis. In addition, we commented on the usefulness of the complexity measures in combination with other image parameters and expressed our optimism that these could be useful in classifying false positives from the MASK R-CNN.

Finally, we concluded that an intricate knowledge of the data is crucial in any modelling task and that future research should take such knowledge into considerations when designing solutions to distinguish between true- and false positives.

References

Abhishek, K., Singh, M., Ghosh, S., & Anand, A. (2012). Weather Forecasting Model using Artificial Neural Network. *Procedia Technology*, *4*, 311–318. https://doi.org/10.1016/j.protcy.2012.05.047

Bansal, R., Raj, G., & Choudhury, T. (2016, November). Blur image detection using Laplacian operator and Open-CV, In *2016 International Conference System Modeling Advancement in Research Trends (SMART)*. 2016 International Conference System Modeling Advancement in Research Trends (SMART). https://doi.org/10.1109/SYSMART.2016.7894491

{Danmarks Frie Forskningsfond} (**typedirector**). (2019, March 25). *Aarets originale idé 2018*. Retrieved May 18, 2020, from https://www.youtube.com/watch?v=0Co69hfqi24&feature=youtu.be

Deeplearning.ai (**typedirector**). (2017, November 7). *C4W3L01 Object Localization*. Retrieved May 7, 2020, from https://www.youtube.com/watch?v=GSwYGkTfOKk&list=PL_IHmaMAvkVxdDOBRg2CbcJBq9SY7ZUvs

Dryas integrifolia. (2020). Retrieved May 27, 2020, from http://arcticplants.myspecies.info/file/1132

Ekici, S., & Jawzal, H. (2020). Breast cancer diagnosis using thermography and convolutional neural networks. *Medical Hypotheses*, *137*, 109542. https://doi.org/10.1016/j.mehy.2019.109542

Evert, R. F., & Eichhorn, S. E. (2012, March 9). *Raven Biology of Plants* (Eighth edition). W. H. Freeman.

Field, A. P., Miles, J., & Field, Z. (2012). *Discovering statistics using R / Andy Field, Jeremy Miles, Zoë Field.*

[Accepted: 2018-11-05T09:29:48Z]. London ; Thousand Oaks, Calif. : Sage, retrieved May 26, 2020, from http://repository.fue.edu.eg/xmlui/handle/123456789/2902

Krastev, P. G. (2020). Real-time detection of gravitational waves from binary neutron stars using artificial neural networks. *Physics Letters B*, *803*, 135330. https://doi.org/10.1016/j.physletb.2020.135330

Olsen, L. R. (2020a). *Cvms* (Version 1.0.1).

Olsen, L. R. (2020b). *Groupdata2* (Version 1.2.0).

{Python Software Foundation}. (2020). *Python* (Version 3.8.2). http://www.python.org

{R Core Team}. (2020). *R: A Language and Environment for Statistical Computing*. Vienna, Austria. https://www.R-project.org/

Vestfalen, J. (2019, November 29). *Aarhus: Hele byen er digitalt testcenter*. Retrieved May 21, 2020, from https://www.coi.dk/nyheder/2019/november/aarhus-hele-byen-er-digitalt-testcenter/

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., . . . van Mulbregt, P. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2

Yu, H., & Winkler, S. (2013, July). Image complexity and spatial information, In *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*. 2013 Fifth International Workshop on

Quality of Multimedia Experience (QoMEX), Klagenfurt am Wörthersee, Austria, IEEE. https://doi.org/10.1109/QoMEX.2013.6603194

Zanette, D. H. (2018). Quantifying the complexity of black-and-white images (D. R. Chialvo, Ed.). *PLOS ONE*, *13*(11), e0207879. https://doi.org/10.1371/journal.pone.0207879

A The Q Complexity Measure

In Figure 4 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 5.

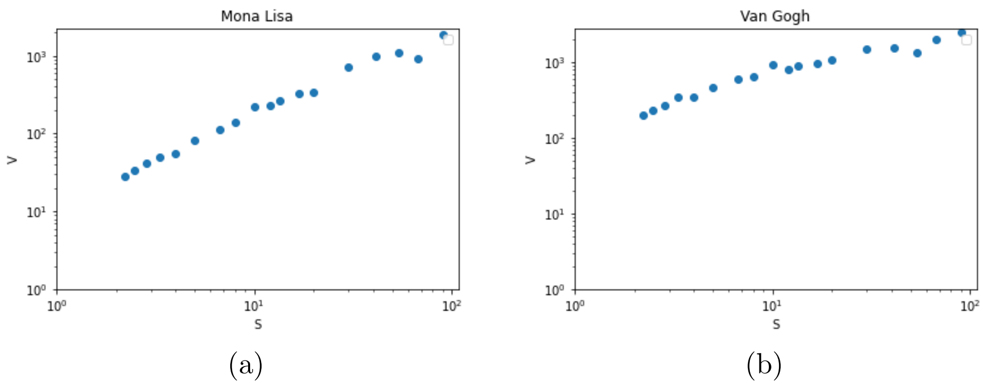


Figure 4: Variance profiles of the a) Mona Lisa and b) Van Gogh images recreated in python

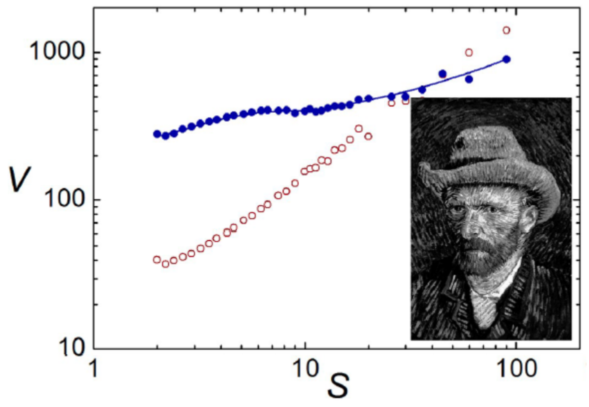


Figure 5: Original variance profiles of the Mona Lisa (red) and Van Gogh (blue) images. Image is an exempt from (Zanette, 2018)

B   Interaction Plots

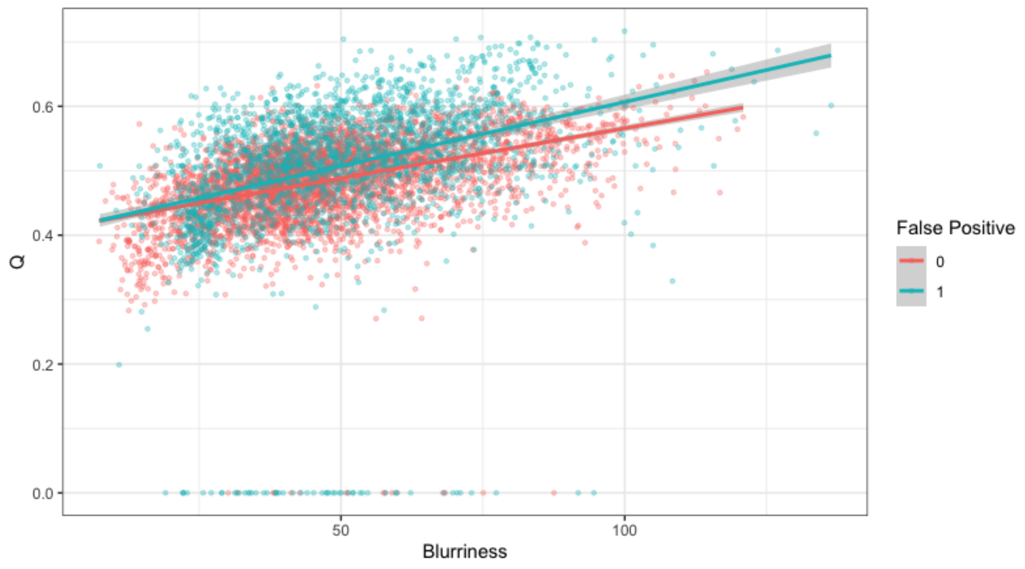


Figure 6: Interaction effect between the parameters Blurriness and Q

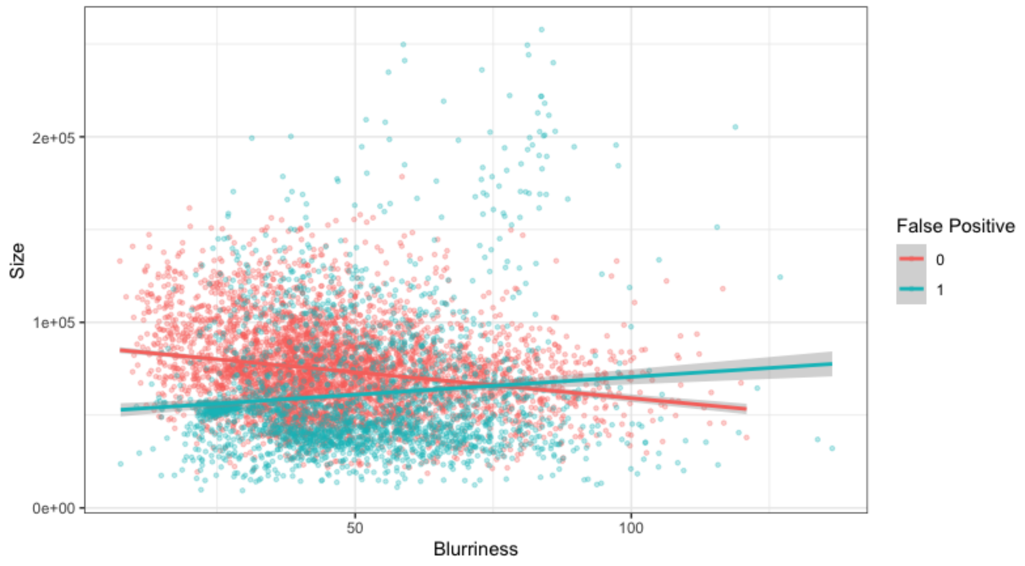


Figure 7: Interaction effect between the parameters Blurriness and Size

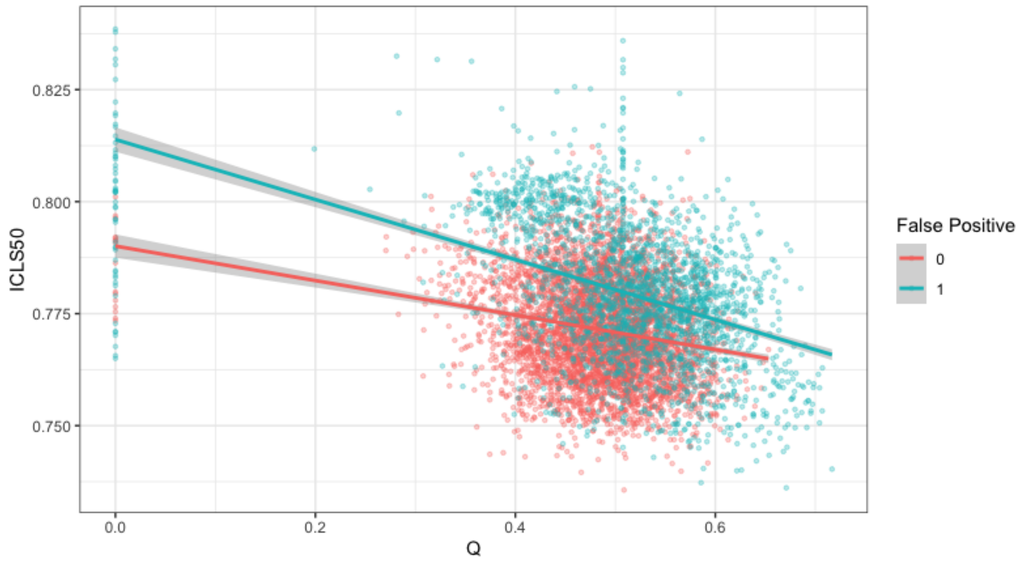


Figure 8: Interaction effect between the parameters Q and ICLS50

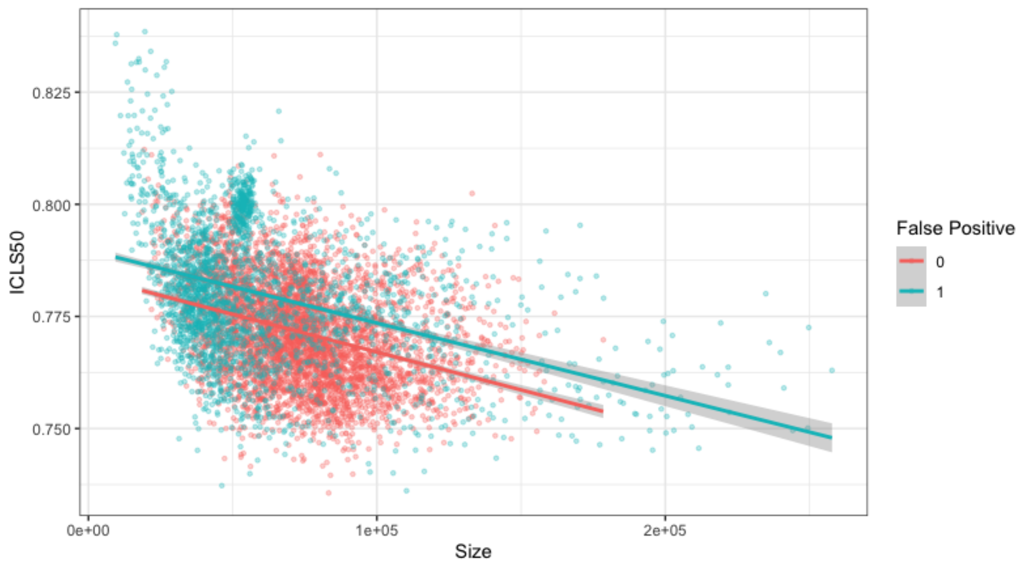


Figure 9: Interaction effect between the parameters Size and ICLS50

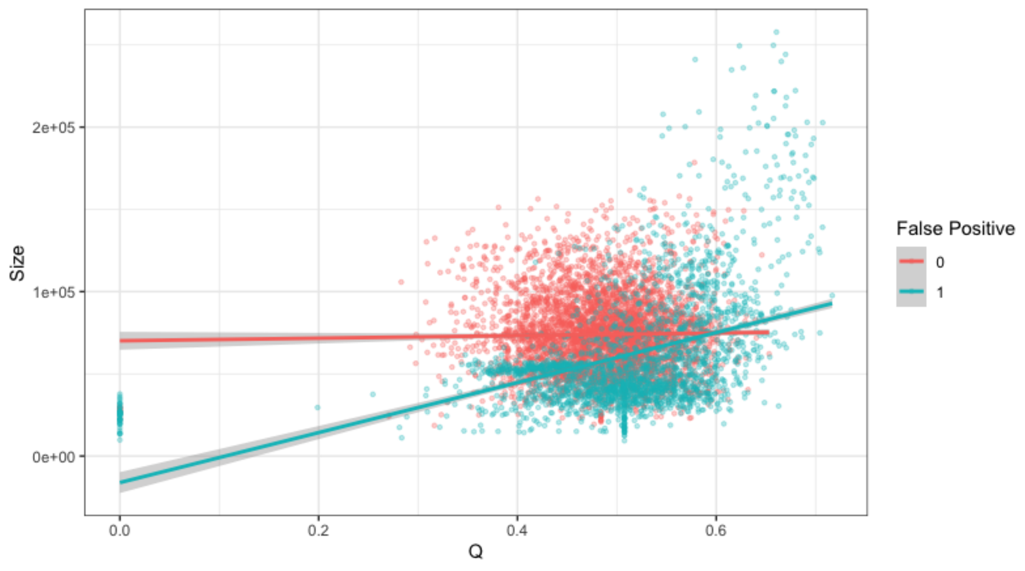


Figure 10: Interaction effect between the parameters Q and Size

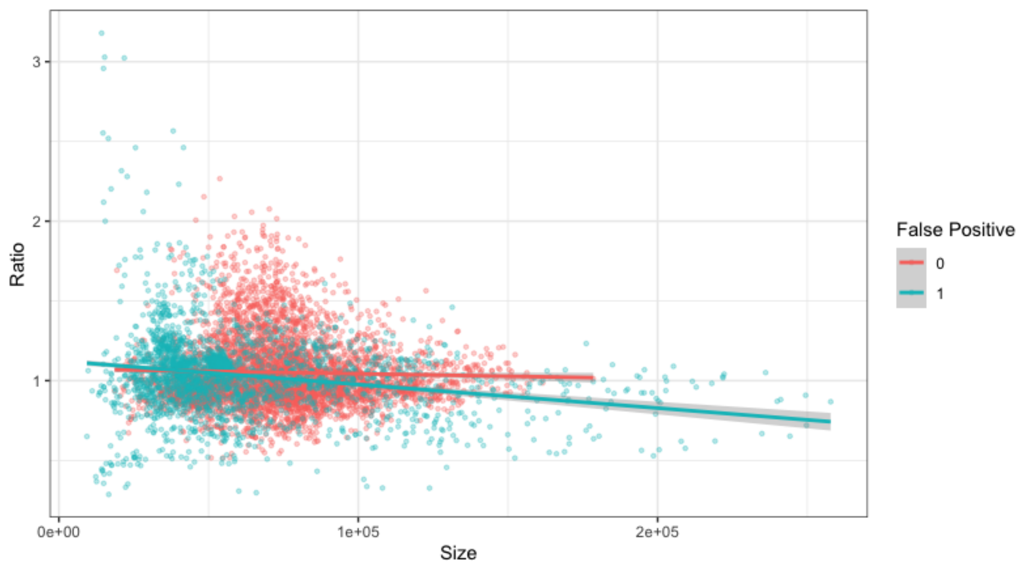


Figure 11: Interaction effect between the parameters Size and Ratio (height/width)

C Multicollinearity Checks

Table 4: multicollinearity checks for predictors of the highest scoring model

	VIF	1/VIF
blur	84.63	0.01
Q	6544.67	0.00
size	4715.35	0.00
ICLS50	97.29	0.01
ratio	9.19	0.11
blur:Q	132.37	0.01
blur:size	22.39	0.04
Q:ICLS50	5627.61	0.00
size:ICLS50	4126.69	0.00
Q:size	157.25	0.01
size:ratio	35.07	0.03

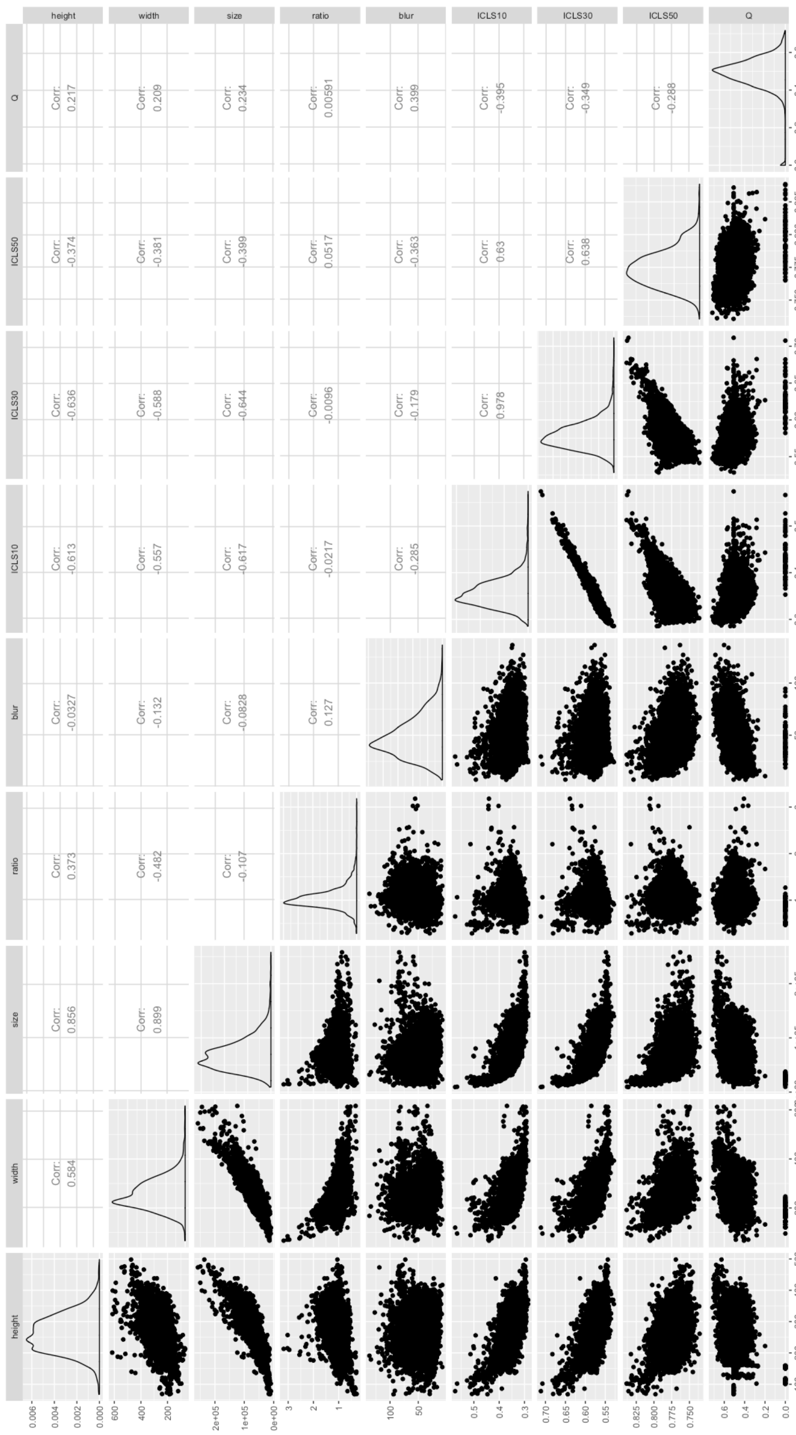


Figure 12: Multicollinearity matrix for the metadata parameters

D Python Code

```
################################################################
########################## IMPORT PACKAGES ####################
################################################################

from imutils import paths
import cv2
import numpy as np
from keras.preprocessing.image import load_img, img_to_array
from scipy.integrate import quad
from jax import grad
import pandas as pd
import math
import os
import re
from scipy.interpolate import BSpline
from scipy import interpolate

################################################################
########################## DEFINE FUNCTIONS ####################
################################################################

def location_scout(filename):
    if "NARS" in filename:
        location = "NARS"
    else:
        location = "THUL"
    return location

def label_false_positives(path):
    if "false" in path:
        label = "1"
    else:
        label = "0"
    return label

def create_L_boxes(img, L):
    #creates non-overlapping boxes of size LxL
    L_boxes = []

    #loop over pixels in image to create L-boxes:
    for r in range(0, img.shape[0], L):
        for c in range(0, img.shape[1], L):
            L_box = img[r:r+L,c:c+L]
            L_boxes.append(L_box)

    return L_boxes

def calcuate_box_variance(box, mean_gb):
    gi_variance =[]

    for gi in box.flat:

        variance = (gi - mean_gb)**2

        gi_variance.append(variance)

    box_variance = np.sum(gi_variance)

    return box_variance

def resize_image(img, resize_scales, L):
    resized_images = []
    for scale in resize_scales:
        resized_width = int(img.shape[1] * scale /100)
        resized_height = int(img.shape[0] * scale /100)

        #making sure the image can be devided into LxL boxes
        if resized_width%L == 1:
            resized_width += -1
        if resized_height%L == 1:
            resized_height += -1

        dims = (resized_width, resized_height)

        resized_image = cv2.resize(img, dims, cv2.INTER_NEAREST)

        resized_images.append(resized_image) #add to list

    return resized_images

def gray_level_mean_variance(normalized_gray, resize_scales, L):

    V_of_S = []

    resized_images = resize_image(normalized_gray, resize_scales, L)

    #divide images into L_boxes and calculate mean variances
    for i, img in enumerate(resized_images):

        L_boxes = create_L_boxes(img, L)

        Vbs = []

        for box in L_boxes:
            #calculate the mean grey level, mean_gb, of the box
            box_sum = cv2.sumElems(box)
            mean_gb = (1/L**2)*box_sum[0] #eq 2
            #print(f"mean gb = {mean_gb}")

            #calculate the sample variance of the gray level, Vb, of the box
            box_variance = calcuate_box_variance(box, mean_gb)
            #print(f"box var = {box_variance}")
            Vb = (1/(L**2-1))*box_variance #eq 1
            #print(f"Vb = {Vb}")
            Vbs.append(Vb)

        #calculate gray-level mean variance V, over all boxes:
        V = ((L**2)/(img.shape[0]*img.shape[1]))*np.sum(Vbs) #eq 3

        #print(f"V = {V}")

        S = L*normalized_gray.shape[0]/img.shape[0] #eq 4

        V_of_S.append((V,S, resize_scales[i]))

    return V_of_S

def ramp(x):
    return np.maximum(0, x)

def Q_complexity(glmv):

    #take log of V and S
    v = [math.log(i[0]) for i in glmv]
    s = [math.log(i[1]) for i in glmv]
    #find limits for integral
    Smax = max(s)
    Smin = min(s)

    #reverse to increasing order for Bspline()
    v.reverse()
    s.reverse()

    #convert to arrays for splrep
    x = np.array(s)
    y = np.array(v)

    #find vector knots for Bspline
    t, c, k = interpolate.splrep(x=x, y=y, s= 0, k=4)
    spl = BSpline(t, c, k)

    dvds = BSpline.derivative(spl)

    integral = quad(lambda s: ramp(1-(0.25*dvds(s)**2)), Smin, Smax)

    Q = (1/(Smax-Smin))*integral[0] #eq5

    return Q

def ICLS_complexity(img, path, compression):

    uncomp_size = os.path.getsize(path)

temp_compressed = cv2.imwrite(f'temp_compres{compression}.jpg', \
img, [int(cv2.IMWRITE_JPEG_QUALITY), compression])
```

```
        comp_size = os.path.getsize(f'temp_compres{compression}.jpg')

        os.remove(f'temp_compres{compression}.jpg') #remove the temp image from disk

        CR = uncomp_size/comp_size #eq1

        ICLS = 1/CR #eq2

        return ICLS

    def extract_meta_data(data_folder):

        metadata = []

        for dirpath, _, files in os.walk(data_folder):
            for i,file in enumerate(files):
                if file.endswith(".jpg"):

                    path = os.path.join(dirpath,file)
                    #print(f"{dirpath} {i}/{len(files)}")
                    print(path)

                    filename = file #extract filename from path

                    img_no = re.findall('[\d]+(?=.JPG)',filename)[0]

                    location = location_scout(filename)

                    label = label_false_positives(path)

                    image = cv2.imread(path) #import image
                    shape = image.shape #height, width and number of dimensions

                    size =  shape[0] * shape[1] #height * width

                    ratio = shape[0] / shape[1] #height/width

                    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

                    blur = cv2.Laplacian(gray, cv2.CV_64F).var()

    normalized_gray = cv2.normalize(gray, None, alpha=0, \
    beta=255, norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_32F)

                    ICLS10 = ICLS_complexity(image, path, 10)

                    ICLS30 = ICLS_complexity(image, path, 30)

                    ICLS50 = ICLS_complexity(image, path, 50)

    glmv = gray_level_mean_variance(normalized_gray, \
    (3,4,5,7,10,12,15,17,20,25,30,40,50,60,70,80,90), 2)

                    Q = Q_complexity(glmv)

    img_tupple = (filename, img_no, shape[0], shape[1], \
    size, ratio, blur, location, label, ICLS10, ICLS30, ICLS50, Q)

                    metadata.append(img_tupple)

        return(metadata)

    ###############################################################
```

```
########################## RUN SCRIPT ######################
##############################################################

data_folder = "./data/"
my_metadata = extract_meta_data(data_folder)

#convert to csv file and save
metadata = pd.DataFrame(my_metadata, columns = ["filename", \
"img_no", "height", "width", "size", "ratio", "blur", "location", \
"false_pos", "ICLS10", "ICLS30", "ICLS50", "Q"])
metadata.to_csv('metadata.csv')
```

E R Code

```
Load packages

setwd("/Users/isalykkehansen/Desktop/Git/Data-Science-Exam-2020/analysis")
library(pacman)
p_load(tidyverse, cvms, groupdata2, knitr,
doParallel, lmerTest, bbmle, car, GGally, xtable)
set.seed(1991)
‘‘‘

#Data wrangling

metadata <- read.csv("metadata.csv")

#find mean Q values for false and true positives
Q_data <- filter(metadata, Q != "NA") %>%
  group_by(false_pos) %>%
  summarise(Q_mean = mean(Q))
```

```
#assign the means to the 73 imgs with missing Q values:
data <- metadata %>%
  mutate(Q = ifelse(is.na(Q) & false_pos ==1, Q_data$Q_mean[2], Q)) %>%
  mutate(Q = ifelse(is.na(Q) & false_pos ==0, Q_data$Q_mean[1], Q))


#Multicollinearity check

#look at multicollenearity btw variables
multicol <- data %>%
  select(4:14, -c(location, false_pos))

X<-multicol
ggpairs(X)



#Cross validation

#find all possible combinations of predictors
model_formulas = combine_predictors("false_pos",
c("size", "ratio", "blur", "ICLS50", "Q"), max_interaction_size = 2)
#run CV in parallel
registerDoParallel(7)

cv1data <- data
#create folds for cvms (only run once to find best models)
cv1data <- fold(data, k = 10, cat_col = 'false_pos',
 id_col = 'X')

CV <- cross_validate(cv1data, model_formulas,
 fold_cols = '.folds',
 family = 'binomial',
 REML = FALSE,
 parallel = TRUE)

#write_csv(CV[1:15], "CV_size_ratio_blur_ICLS50_Q_2way.csv")


#arrange the models in order - best on top
arranged_BA = arrange(CV, desc(`Balanced Accuracy`))

#show the whole model and only one metric
select_definitions(arranged_BA, additional_includes = "Balanced Accuracy")

#extract the model formulas of the best 100 models
best_model_formulas = reconstruct_formulas(arranged_BA, topn = 100)


#CV of best 100 models

cv2data <- data

#create folds for repeated cvms
cv2data <- fold(cv2data, k = 10, cat_col = 'false_pos',
 id_col = 'X', num_fold_cols = 5)

#cross validate on the folds
CV2way <- cross_validate(cv2data, best_model_formulas,
 fold_cols = c('.folds_1', '.folds_2', '.folds_3', '.folds_4', '.folds_5'),
 family = 'binomial',
 REML = FALSE,
 parallel = TRUE)
```

```
#inspect the models
arranged_BA = arrange(CV2way, desc(‘Balanced Accuracy‘))
top100 <- select_definitions(arranged_BA,
additional_includes = c("Balanced Accuracy", "AUC"))
top_10 = reconstruct_formulas(arranged_BA, topn = 10)

best_model <- as.data.frame(arranged_BA[1,2:9])
print(xtable(best_model, type = "latex"), file = "best model.tex")



#model evaluation

topmodel <- glm(false_pos ~ blur * Q + blur * size + ICLS50 * Q + ICLS50 * size
+ Q * size + ratio * size, family = "binomial", data = data)

anova(topmodel)
summary(topmodel)

vif = as.data.frame(vif(topmodel))
column <- (’1/vif’ = 1/vif[,1])
vif_stats <- cbind(vif, column)
print(xtable(vif_stats, type = "latex"), file = "vif_stats.tex")

model_coefs <-summary.glm(topmodel)$coefficients
model_coefs <-as.data.frame(model_coefs) %>%
  select(-’z value’)
model_coefs$Estimate <- round(model_coefs$Estimate, 2)
round(model_coefs, 5)

print(xtable(model_coefs, type = "latex"), file = "model_coefs.tex")
```

```
logisticpseudoR2s <- function(logisticmodel) {
  deviance <- logisticmodel$deviance #extract model deviance
  nulldeviance <- logisticmodel$null.deviance #extract baseline model deviance
  modelN <- length(logisticmodel$fitted.values) #compute sample size
  R.l <- 1 - deviance/nulldeviance  # Hosmer and Lemeshow's R2 is computed
  R.cs <- 1- exp(-(nulldeviance-deviance)/modelN) # Cox and Snell R2
  R.n <- R.cs / (1 - (exp(-(nulldeviance/modelN)))) # Nagelkerke R2
  cat("Pseudo R2 for logistic regression\n")
  cat("Hosmer & Lemeshow's R2    ", round(R.l,3), "\n")
  cat("Cox and Snell's R2    ", round(R.cs,3), "\n")
  cat("Nagelkerke's R2    ", round(R.n,3), "\n")
}

logisticpseudoR2s(topmodel)
sum(residuals(m10, type = "pearson")^2)



#Predictive checks across locations



loc_0 <- data %>% dplyr::filter(location == "NARS")
loc_1 <- data %>% dplyr::filter(location == "THUL")

# Best model from cross-validation
model <- glm(false_pos ~ blur * Q + blur * size + ICLS50 * Q + ICLS50 * size
+ Q * size + ratio * size, family = "binomial", data = loc_0)

# Get predictions on the loc_0 data
predictions <- data.frame(
  "prediction" = predict(model, newdata = loc_1, type="response"),
  "target" = loc_1$false_pos
)


# Evaluate predictions
eval <- evaluate(predictions,
 target_col="target",
 prediction_cols="prediction",
 type="binomial")

Nars_vs_Thul <- as.data.frame(eval[1:9])
print(xtable(Nars_vs_Thul, type = "latex"), file = "Nars_vs_Thul.tex")

#Visualising interaction terms

data <- mutate(data, false_pos = as.factor(false_pos))
# Setting up the building blocks
basic_plot <- ggplot(data,
   aes(x = size,
   y = ratio,
   color = false_pos)) +
  theme_bw() +
  labs(x = "Size",
   y = "Ratio",
   color = "False Positive")

# Colored scatterplot
basic_plot +
  geom_point(alpha = .3,
 size = .9) +
  geom_smooth(method = "lm")
```