Knest: Selection and Enhancement of Wildlife Photographs

Alexander deCurnou, Joseph Leavitt, Nhi Nguyen, and Jonathon Rice

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

Abstract — **Wildlife photography is a popular hobby of many people around the world. However, captured images can total in the thousands and these hobbyists must search through all of these files for desirable images, which takes a significant amount of time. In this work, we aim to shorten this period by presenting *Knest,* an end-to-end solution for selecting and enhancing bird photography. We combine machine learning methods, such as object classification and detection, as well as a number of computer vision techniques in order to automatically produce aesthetically-pleasing versions of a bird photographer's images.**

Index Terms — **deep learning, computer vision, neural networks, object detection, object classification**

I. Introduction

Wildlife photography is an extremely popular hobby for many people around the world; in the case of bird watching, there are approximately forty-five million bird observers in the United States alone [1]. A significant portion of this group of people travel away from their homes in order to pursue their hobby. Bird watchers spend billions of dollars each year on photography equipment and travel accommodations in order to find elusive subjects and capture an image of them. As such, the local economies located near popular bird watching spots benefit from the money spent on food, lodging, and transportation.

Wildlife photographers, both amateur and professional, can take a colossal number of pictures on a single excursion. Many photographers place their camera in a particular viewpoint and set the camera to take as many pictures as possible in a short time in order to capture a single image of their subject. Additionally, a subject has the potential to quickly move at the point of image capture, which leads to the presence of blur inside the image. Upon their return, one may find that the collection of images from a single excursion can number in the thousands. As a result, photographers may spend an inordinate amount of their leisure time in the sorting process alone in which they search for images that suit their particular purpose. Additionally, a subset of these people spends additional periods of time editing and manipulating their photographs. Since a wildlife photographer's trip may span several days, the total time spent processing these images can add up to an infeasible amount of a person's time, especially when one considers that the photographer may have other obligations in their life.

There exists a number of tools aimed at automatically classifying images and grouping similar images together, such as the photography suites offered by Apple and Google. However, these suites only offer human face detection and do not recognize animals, which poses a problem for wildlife photography. Furthermore, these programs do not automate the manipulation of these images, and thus, the user is required to carefully inspect each of their images in order to ascertain whether a desired subject is present. Finally, there is a noticeable absence of options for blur detection in many of these photography suites.

In this paper, we address this set of problems by creating an application that programmatically does each of these actions in turn. We present *Knest*, an end-to-end solution in which a user's original images are given as input and cropped images of a subject are given as output. Our solution consists of five stages, ordered chronologically: blur detection, object classification, object detection, image comparison, and image manipulation. At this time, *Knest* only processes images containing birds. Notably, the application can process one hundred full-size images of size 5184x3456 in approximately one half-hour and five hundred images of the same size in about two hours. Our blur detection stage makes use of the Haar wavelet transform function in order to identify what edge types are present in an image and the sharpness of each. The object classification and detection stages are powered by two separate state-of-the-art neural network architectures, trained on images from a collection of datasets. The former stage removes any images that do not contain a bird. Leftover images are then sent to the latter stage in which an attempt is made to locate the bird and its face. After the detection stage, we attempt to reduce the amount of similar image written to disk by comparing the hash values of images. Finally, image manipulation uses the location of a bird's face and its body in order to create an output image centered around the subject. To the best of our knowledge, this is the first work to combine each of these five stages in order to provide automated detection and cropping of image subjects. This usefulness of this application is not limited to solely bird photography. The project was created in a subject-agnostic manner and thus, its object recognition capabilities can easily be changed or extended by supplying a different dataset to the neural networks that comprise the classification and detection stages.

II. Proposed Approach

There are a number of solutions aimed at addressing some of the aforementioned issues; however, these solutions do not work in tandem with one another in order to provide an easy-to-use solution, which would require a user to use several different programs and spend time preparing each of them. In contrast, our application combines several components in order to address each of the common "time sinks" of the wildlife photographer and does so in a seamless and automated user experience. The application is intended for use on personal computers, although it is not limited to a certain type of architecture. As such, each component of the program was developed to be as effective yet quick as possible. Every stage is independent of one another and can be removed or enabled at will.

A. Blur Detection

Despite advances in digital photography, standardized blur detection still seems to an elusive concept. In many aspects, the practice relies on a combination of mathematical operations and requires a great deal of fine-tuning or weighting to achieve an efficient measure. We attempted to use a number of operations, specifically those involving variance or transforms.

Variance operations essentially measure the sharpness of an image by measuring the amount of rapid intensity changes in an image. Blurry images will have fewer regions of rapid intensity changes as the intensity will be spread over a greater area of pixels. This leads to a smaller spread of responses when analyzing the image, which gives a lower variance measure. In contrast, sharper images will have more regions of rapid intensity change and will subsequently lead to a higher variance measure. We tested two methods of measuring the Laplacian variance as well as a single method for calculating the Tenengrad variance. While these methods worked quite well on images with an abundance of sharp edges, e.g. birds in trees, it faltered on images where there was a marked absence of edges, such as regions of sky or water.

We also attempted to use a number of methods based on transform operations, namely calculating the mean of the fast Fourier transform (FFT) of an image and generating the

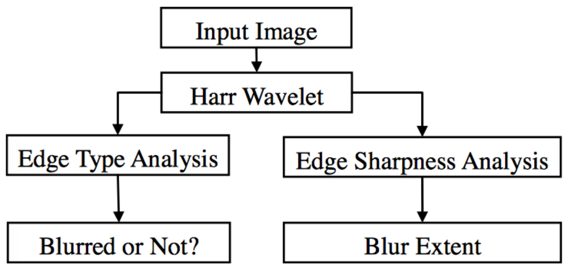

Fig. 1 The algorithm used by the blur detection stage

sum of coefficients given by the discrete wavelet transform (DWT) when processing an image. Typically, one can blur an image by calculating the discrete transform given by each of the transform operations and then multiplying it by a Gaussian function to decrease the high-frequency components of original image. Similarly, one can analyze the high-frequency components of an image to determine image sharpness. In our testing, we found that calculating the sum of coefficients given by the DWT produced results that were similar to the variance methods. In contrast, determining the mean of the FFT gave accurate results for images in which the variance methods did not; however, it was not able to detect focal blur on a consistent basis.

The aforementioned measures only do a superficial edge analysis and detection. However, by using a type of wavelet transform called the Haar wavelet transform, we are able to do analyses on both edge types and the sharpness of those edges. Our algorithm involves first doing a Haar wavelet transform on an image, which returns a wavelet approximation along with detail coefficients; a diagram detailing this process can be seen in Figure 1. Then, a second transform is done on the approximation returned by the first iteration, and a third and final iteration is done on the approximation returned by the second iteration. The details from each of the iterations are turned into edge maps, which are then separated into partitions. The local maxima of each partition represent edge intensities which can then be classified as one of three major types of edges: Dirac-Structure, Step-Structure, and Roof-Structure. Step-Structure edges can be further classified as Astep and Gstep according to the gradation of intensity changes. Blurry images are typically characterized by a lack of Dirac- and Astep-Structure edges and a decrease in the sharpness of Gstep- and Roof-Structure edges. By testing whether the amount of Dirac- and Astep-Structure edges exceeds a threshold of about five percent of all edges, we can determine in turn whether an image is blurry. Additionally, we can measure the extent of blur in an image by calculating the ratio of Gstep- and Roof-Structure edges with decreased sharpness to the total number of Gstep- and Roof-Structure edges.

B. *Object Classification*

Our object classification stage functions as the second filtering step for the *Knest* application. For each network architecture, preprocessing steps were applied to the dataset in order to encourage viewpoint-invariant representations of objects. We tried using several different network architectures in order to find the most optimal model for the project purpose. Initially, residual networks (ResNets) were used as the classification architecture as they boast state-of-the-art image classification accuracy while claiming lower complexity than other high-accuracy architectures, e.g. VGG [2]. Several modifications were made to the network model defined in [2] in order to support an image size of 448x448 pixels; this was done because we found that shrinking dataset images to a smaller size lead to a loss of important visual information that would be optimal for training. The validation accuracy of the ResNet-inspired model was calculated to be ~65%. Furthermore, the classification speed of this model was longer than what we desired for a filtering step.

Thus, another classification model was defined using the VGG network architecture [3]. Modifications similar to those made for ResNet were made to the standard model in order to support our image size. The validation accuracy of this VGG-inspired model was similar to that of the ResNet model at ~68%. In addition to the subpar accuracy of both of these architectures, there was also a major issue of memory usage. Both of these models are quite memory intensive as they require the holding of both the convolutional layers and their hyperparameters in memory; as the models were trained, it became apparent that storing this information was intractable.

The SqueezeNet network architecture was ultimately chosen as it boasts “AlexNet-level accuracy on ImageNet with 50x fewer parameters” [4]. Additionally, the final model size of the trained network is approximately six megabytes (6 MB), which is an order of magnitude smaller than the sizes of the other network architectures and will allow for greater portability of the application.

SqueezeNet employs three different design strategies in order to preserve accuracy while significantly shrinking the total amount of parameters. The first strategy is to replace a number of 3x3 convolution filters with 1x1 filters as the latter has one-ninth of the number of parameters as the former. Downsampling further in the network has also been shown to maintain or increase classification accuracy as the activation maps used by subsequent layers will be larger and contain additional information that may be beneficial. The final strategy involves decreasing the number of input channels to the remaining 3x3 filters. By reducing the amount of unique input channels to the 3x3 filters, the total number of parameters is further reduced. This is done through the use of *Fire* modules, comprised of a layer of 1x1 convolution filters followed by a layer with a mix of 1x1 and 3x3 convolution filters.

Our implementation of the architecture differs slightly from that of the one defined in [4]. First, images are resized to 400x400 as they are passed to the network and secondly, there are only two classification classes, essentially “bird” and “not bird”. We also changed the regression optimization to use the Nesterov method in order to encourage greater exploration of potential minima in the loss function. Through the use of this architecture, we mitigate the problems of subpar classification accuracy and slower-than-desired runtime as well as the issue of extreme memory usage.

C. *Object Detection*

As arguably the most important part of the *Knest* application, we decided that accuracy should be prioritized over speed. As such, we tested a number of models claiming to have state-of-the-art detection capabilities. We attempted to define our own object detection model; however, we quickly found that achieving a working model was intractable. Thus, we made use of Google’s TensorFlow model collection to jump-start our object detection capabilities. These models are trained for as long as several months using up to 300,000 images from the COCO dataset.

The first architecture we selected was a model that implemented neural architecture search (NAS) along with a Faster Region Convolutional Neural Network (F-RCNN) architecture called NASNet. This architecture improves itself by first generating several different “child” architectures that are then trained using a dataset. It then determines the accuracy of each child network such that networks with high accuracies are further fine-tuned in order to construct the most accurate model possible. While the accuracy of the resultant model was indeed the greatest of all of the models we trained, we did not use the NASNet model as it was also the slowest of all networks available. One forward pass through the network took approximately two seconds; in contrast, the speed of a forward pass for many detection models is on the order of tens of milliseconds.

In order to mitigate the extreme amount of time taken by the NASNet model, we attempted to use a lightweight detection model in order to improve the palatability of the application response time, specifically a F-RCNN model based on Inception. However, we found that while this model was faster than NASNet by an order of magnitude, the accuracy was much lower. Thus, we desired to find a

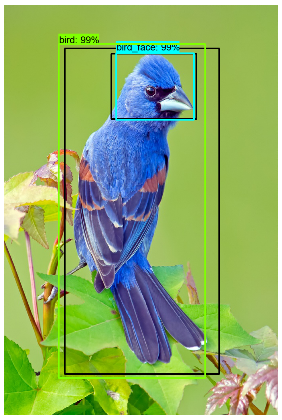

Fig. 2 Overlap of the ground-truth bounding box with the prediction returned by the detection model

model that was fast but not at the expense of detection accuracy.

Ultimately, we chose a F-RCNN model based on the ResNet-50 network architecture. We found that this model was average in speed; however, it was quite accurate in terms of object detection, as shown in Figure 2. This model was trained on a dataset of birds and their faces where applicable.

D. *Image Comparison*

Using a standardized Haar wavelet hashing library, we developed a custom algorithm to reduce the amount of similar images in any given set. Minimizing runtime, we aimed to only visit each image once. We begin comparing the input set of images by setting the first image as the comparison standard. The hash for that standard is calculated and kept in memory. Since we wish to limit the number of similar images to three, in addition to the comparison standard, we only wish to allow two more images. As such, the hash of the succeeding image and its difference with the standard is calculated. If the difference was at or below a certain value, then the succeeding image is deemed similar to the comparison standard. If the count of images similar to the standard is not yet two, then the image is accepted; otherwise, it is ignored. This pattern will continue until a subsequent image is deemed dissimilar. At that point, the current working image becomes the new comparison standard, the count of similar images is reset, and the process is repeated for the new standard. This continues until all images have been compared.

E. *Image Manipulation*

The final stage of the application is the image manipulation state in which filtered images are cropped in such a way that emphasizes any birds detected. This is done using the bounding boxes outputted by the object detection stage.

The algorithm first calculates the center of the image from its dimensions. Then, given a bounding box for a bird from the set of bird face boxes returned from the detection stage, determine the center of that particular box and calculate the distance between the box center and the center of the image. If the center of this box is closer to the image center than all preceding boxes, then store the box center as the most central face box. Then, determine the extrema of all bird bounding boxes. From those extrema, calculate the difference between the largest and smallest extrema for both the x- and y-components. These differences are then multiplied by the square root of a scaling factor in order to enforce the inclusion of some of the surrounding environment in the final image. The resultant height and width are then divided in half to calculate the amount that needs to be added and subtracted from the most central face coordinates to determine the final crop area dimensions. Before cropping the image, the crop dimensions are checked to ensure that they are within the bounds of the original image. Finally, the image is cropped and written to disk.

Through the use of this algorithm, we enforce an aesthetic style in final images that centers the viewpoint around the eye of a bird, regardless of the amount of birds in the original image. In images with a single bird, the algorithm will use the center of the face bounding box, which serves as an approximation for the eye location, as the center for the final image. It then scales the area of bird bounding box by a scaling factor in order to preserve an area of the environment and uses this new scaled area as the viewpoint of the final image. In images with multiple birds, the algorithm selects the bird bounding box coordinates that are closest to each of the original image boundaries as the dimensions of the salient area. After the determination of this area, the image is manipulated using the same process as single-bird images.

III. Implementation Details

A. *Datasets*

We used a number of datasets for training our neural network models. Our main dataset, provided by our sponsor, was comprised of their own personal photographs collected from several years of wildlife excursions. The object classification model was originally trained using labeled, resized versions of these images; this did not lead to a consistent classification of images as the details of the salient object are lost in the resize operation and the surrounding environment becomes the salient object in each image. To mitigate this issue, we manually annotated over three thousand images from this dataset and cropped the objects accordingly. After checking these images for extremely dissimilar height and width, these cropped images were then resized to our standard size of 400x400 pixels. We found these new images to be much more effective in accurately training the model.

For the object detection model, we used the NABirds dataset, compiled and provided by the Cornell Lab of Ornithology. This dataset is comprised of ~48,000 annotated photographs of birds that are commonly observed in North America [5]. The images are annotated across more than 550 visual categories, including beak, wing tip, and nape. This allowed us to define meta-categories, e.g. bird face, that enabled the fulfillment of one of our core requirements. From the standard set of annotations, a derivative set containing the coordinates for birds and their faces was created. The object detection model was then trained using this derivative set. Additionally, by using the bird boxes defined in the annotations, cropped versions of each of the 48,000 images were generated and used to further train the object classification network.

B. *Parameters and Settings*

For the blur detection stage, images are resized to 1024x1024 pixels. We set the threshold for local maxima consideration to thirty-five; this was done as a result of a recommendation given by the authors of [6] as they state that humans have difficulty detecting edges with intensities below thirty. Additionally, the threshold for edge type analysis was set to zero as a non-blurry image will likely have a non-zero amount of Dirac- and Astep-Structure edges.

Our object classification network modeled after SqueezeNet uses an image size of 400x400 pixels, a stride length of two, and sixty-four filters with a size of three. We use Xavier initialization and the exponential linear unit (ELU) as an activation function for each of the convolution layers. The number of filters in each *Fire* module is incremented by sixteen for the squeeze layers and sixty-four for the expand layers; the sizes in numerical form are [16, 32, 48, 64] for squeeze layers and [64, 128, 192, 256] for expand layers. For the regression layer, we use the Nesterov optimization function with a learning rate of 0.001 with a decay of 0.96 applied every one hundred (100) training steps. Finally, we use a dropout probability of 0.5 in order to prevent overfitting.

We adjusted a number of parameters in the standard ResNet-50 model provided by Google's TensorFlow team. The standard model is fully convolutional, which allows for images of different sizes to be used. As such, we adjusted the minimum and maximum dimensions to 300 and 1600 pixels, respectively; this enabled us to use all of our training images, which increases the accuracy of object localization. We enabled the use of dropout layers in the model as well and set the probability to 0.5. In order to speed up our training time, the evaluation configuration was adjusted to use a random sampling of 4,000 training examples for a maximum of four evaluations instead of a sampling of 8,000 examples for a maximum of ten. Furthermore, a random horizontal flip was added as an additional data augmentation option as a way to enforce viewpoint-invariant representations.

There are a few settings that are used in the image comparison and manipulation stages. For the former, we resize any images larger than 1000x1000 pixels to that size as a speed optimization. For the actual comparison, we set the difference threshold between the hashes to twenty, as we found that any higher values allowed objectively dissimilar images. Additionally, we retain only two images that are similar to the comparison standard as a way to reduce storage usage. The sole parameter of the image comparison stage is the scaling factor, which is set to three, i.e. the final crop area is three times larger than the area containing the bird.

C. *Hardware*

Training of our object classification and detection models was done on an NVIDIA GTX 1080 Ti, and a system with sixteen gigabytes (16GB) of RAM and an Intel Core i7-8700 CPU with a clock speed of 3.2GHz. Since this application was developed for use on a personal computer, all timings were measured on a mid-2014 MacBook Pro with eight gigabytes (8GB) of RAM and an Intel Core i5 with a clock speed of 2.6GHz.

IV. Results

A. *Network Accuracy*

In regard to classification, our SqueezeNet model achieved a validation accuracy of ~95%, a feat was attained multiple times through each of our dataset variations. We were pleased with this result as it appears that the only false positives that are returned by the model correspond to images containing objects that are similar to birds, e.g. flying insects. The detection model inspired by ResNet-50 did very well, achieving a mean average precision (IoU @ 0.5) of .9611 when presented with 8000 unseen images from the NABirds dataset.

B. Run-time Efficiency

We compute the response time for each of the stages in the application. All measured times are averaged through fifty runs of the full application, given the same input. After the initial resize operation (an almost instantaneous operation), both blur detection and object classification take about fifty milliseconds (50 ms) per image, which essentially appears to be instantaneous to the user. Object detection takes approximately twelve seconds (12 secs) per image; this is because the network convolves over the entire image in order to detect and localize any salient objects that may be present. For image comparison, an image is resized and then hashed; this operation takes about a half-second. Calculating the difference between image hashes is considered to be instantaneous. Finally, the speed at which the final cropped images are written to disk depends on a number of factors, namely final image size and the write speed of the underlying hardware; however, on our hardware, we were able to measure the write speed to be approximately one second (1 sec) per image on average.

VII. Conclusion

We presented an end-to-end solution for the automated selection and enhancement of wildlife photography. By combining machine learning and computer vision techniques, we were able to create an application that generates aesthetically-pleasing images from a user's collection of photographs.

Acknowledgement

The authors wish to acknowledge the support and sponsorship of Dr. Gary T. Leavens. Additionally, the help provided by Michael Powell and Dr. Mark Heinrich was integral to the success of this project.

References

[1] 2016 National Survey of Fishing, Hunting, and Wildlife-Associated Recreation, U.S. Fish & Wildlife Service.

[2] Deep Residual Learning for Image Recognition, He et al.

[3] Very Deep Convolutional Networks for Large-Scale Image Recognition, Simonyan and Zisserman.

[4] SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size, Iandola et al.

[5] NABirds, Cornell Lab for Orinithology

[6] Blur Detection for Digital Images Using Wavelet Transform, Tong et al.