Object Detection of Animals from Camera Trap Images

Justin Kim

May 1, 2019

Abstract

In this paper, Eigenbackground method of object detection is explored as a means of object detection of animals from camera trap images in a pre-compiled dataset. Finding a way to model and subtract a common background from a sequence of images or video is becoming increasingly popular and important, because it is the first step in computer vision and object identification of non-static objects. An automated process that can differentiate between the foreground and background of images greatly reduces the human input required in processing these images. The data set that this paper is based on is from Trinity College's $//tbos/Projects/Smedley$ database, formerly a biology/ecology project concerned with compiling biodiversity data gathered in 2015. The images from this data set were obtained through camera traps, which is a motion triggered stationary camera source. The methods outlined in this paper are generalized, and they can be applied to any sequence of images from other stationary image data sources such as traffic cameras and security cameras. By using a combination method as detailed, the suggested Eigenbackground model creates a robust probability density function that makes for a forgiving object detection scheme in even the shortest data sets with only 10 images. This paper will also cover the method used to reduce false positive detection, and the traditional convolutional neural network used in object identification.

1 Introduction

Camera-traps are stationary, motion-triggered cameras that are secured to trees in the field in order to observe the animal population and biodiversity in the selected area. As the animals pass through the desired area, camera traps are motion triggered to capture short sequences of images. These camera traps can also include other sensor information such as time, humidity, light level, and GPS data. Camera-traps are becoming increasingly popular as a non-invasive and cost friendly method of data collection in the environmental science field. Due to the nature of their operation, camera-traps often generate a large volume of images. These image data sets typically range in the order of tens of thousands of images, and the task of manually processing these images is extremely time consuming. As a result, it is important to develop automated methods of object detection, so that animals can be detected and later identified with minimal human input. Likewise, Trinity College has also amassed a large folder of image data sets collected from a camera trap from a recent biology research project.

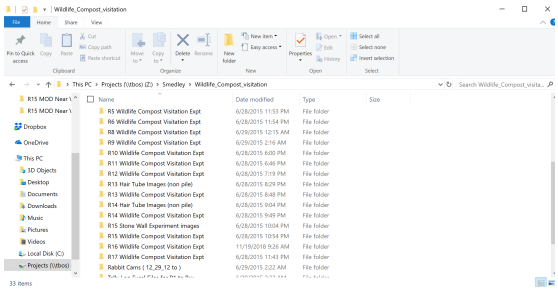


Figure 1: Tbos.Projects.Smedley Image Data Set Files

In this paper, the Eigenbackground method of object detection is explored. The Eigenbackground method uses the eigenvectors of the image data set to perform singular value decomposition and principal component analysis to construct a robust probability density function of the static portions of background. The Eigenbackground is not strictly pixel related, as it takes into account the corresponding pixel values in the form of eigenvectors, and is relatively computationally light. The following sections will detail the methods used to arrive at foreground object detection.

2 Problem Definition

After quickly browsing the image data sets available at Trinity College, it becomes clear that Trinity is in possession of tens of thousands of images generated by the motion sensitive camera traps. Traditionally, these images at Trinity has been processed manually by volunteers such as veterans and younger children. Detecting and identifying animals present in the images becomes an extremely repetitive and time-consuming task at this volume. In an effort to reduce the amount human input that goes into processing these images, the first step is to develop a method for a computer detect if there is a non-static animal present in each of the images.
The following Figure 2 displays an example of a 10-image data set. It can be

noticed that this image data set looks like a series of roughly fragmented frames of a video. To reiterate, the desired goal of this project would be to have a program that returns a corresponding 10 image sequence that contains only the foreground object present in the foreground of the images. In the case of the data set shown below, a successful object detection process would detect that there is a dog in the foreground of the images in the topmost row.

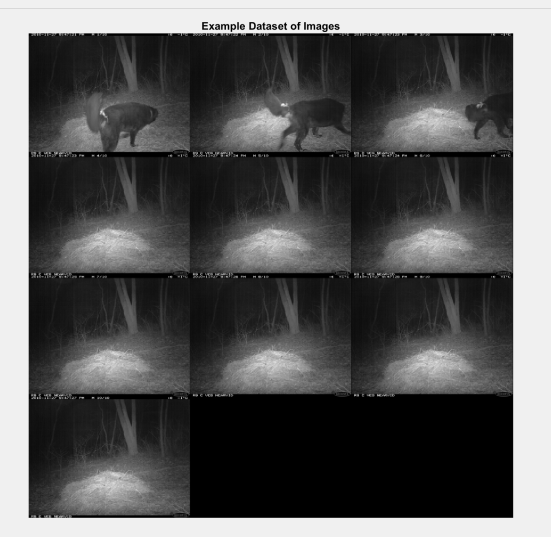


Figure 2: Example 10 Image Data Set

A difference between this project and the numerous research done on Eigenbackgrounds is that the datasets used in this research is precompiled. The data was primarily gathered without the intention of using it for object identification, and thus much of the data is unusable due to moving camera points. Furthermore, there are only a few datasets that have more than 40 consecutive frames of the same camera vantage point. This makes it hard for the Eigenbackground model to provide an adequate training set. In contrast, much of the previous research done on Eigenbackgrounds has specific datasets (usually located indoors) with as many as 200 images for the training set. In this paper, the method of Eigenbackgrounds is used as a post processing method of finding the most amenable way to identify foreground objects from a previously compiled dataset, which consists mostly in dynamic natural scenes. This paper will assume that object detection will be done not in real time, and that we have as few as 10 to 40 images for the data set.

3 Problem Evaluation

While there are numerous publications and research done on the subject of background modelling and object detection, much of the work is done in a closely controlled environment. Likewise, current research on Eigenbackgrounds is primarily concerned with real time object detection of foreground objects in indoor settings, and have training image up to 200 frames. In this paper, the

dataset was preassembled, with the range of image sequences being as short as 10 frames. This paper aims to construct a forgiving, robust background model that can capture the foreground objects given the short set of data.
Once an object detection method is achieved, the final mask of the foreground objects can be compiled into a training dataset for a convolutional neural network to utilize for identification purposes. Arriving at a foreground mask is important in the training set because it greatly narrows down the set of features (pixel locations and value, edges, lines, and color drop) that the neural network must identify for each class. The implementation of object detection using Eigenbackground models will be done on Matlab, while the object identification using the detection methods will be done on Google Colab, using Python and Tensorflow.

4 Design

4.1 Eigenbackgrounds

This method builds an adaptive eigenspace that models the background, and the eigenspace model describes the possible variations in intensity values that is present in the image data set.
The Eigenbackground model first reshapes the image into column vectors. For each image i of size $[w * h]$ (width by height), the images are transformed into a $[wh * 1]$ column vector x_i. From this follows that each proceeding image is also reshaped into a column vector, and placed in the second column, the third, and so forth.
The model is taken from the data set consisting of N images. The mean image, m, of the data set is then calculated to be:

$$m = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1}$$

The mean normalized image vectors are then formed into the matrix X which has the dimension $[wh * N]$:

$$X = [x_1 - m \qquad x_2 - m \qquad \cdots \qquad x_N - m] \tag{2}$$

The columns of X all lie on a wh-dimensional space. The assumption here is that the frames are related and similar, since they are taken from a single, stationary camera source. Since the frames are similar, it is likely that these columns can be represented in a lower dimensional subspace. Following that logic, the singular value decomposition (SVD) of X is then calculated:

$$X = U \Sigma V^T \tag{3}$$

where U is a orthogonal $wh * wh$ matrix and V^T is an orthogonal $N * N$ matrix. The singular values of X are contained within the $wh * N$ diagonal matrix Σ, in decreasing order. If the first r (ranks) singular values are non-zero, while the rest is zero, then the first r columns of U gives an orthogonal basis for the column space of X. Furthermore, if the first r singular values are non-zero and the rest are sufficiently close to zero, then the first r columns of U form an

approximate basis for the column space of X.
Since our images come from a singular, stationary camera source, and because our assumption was that the consecutive images in the same data set is related and similar, the columns of X should be correlated and the singular values should decrease rapidly. Therefore, the column vectors of X can be approximated in a lower dimensional subspace by considering only the first r columns of U as a basis where $r << N$. This is also called the PCA, or the principal component analysis.

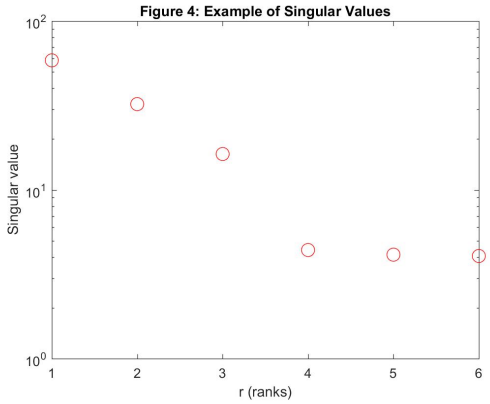


Figure 3: Singular r values of Σ for the Dataset shown in Figure 2

In pracitce, while one can construct a covariance matrix of XX^T to find the eigenvectors correpsonding to the orthogonal matrix U, in Matlab, there is the singular value decomposition function svds, which conveniently computes and stores only the first 6 ranks of SVD. This was the implementation of this design.

As shown by the Figure 3, one can see that taking the SVD of the images in data set shown in Figure 2 will have results such the singular r values are rapidly decreasing, thereby reaffirming the fact that the first r vectors of U can form an appropriate subspace spanned by the columns of X.
The first r columns of U that is used will now be referred to as U_r, which is also known as the "Eigenbackgrounds". Any new image y can be projected onto the reduced subspace represented by U_r through the following equation:

$$\tilde{y} = U_r p + m \tag{4}$$

and since U_r is orthogonal (per the definition of SVD), p (the principal component of the image data set of mean normalized vector X) is obtained by performing the following calculation:

$$p = U_r^T (y - m) \tag{5}$$

Moving foreground objects do not have a significant contribution to the eigenbackground model, because they do not appear in the same location in the N number of sample images. This lack of contribution to the model may also be correlated if the moving foreground objects are small. The portion of an image containing a moving object cannot be aptly described by the eigenbackground

model, thus leaving an absolute threshold difference with the image at hand. On the other hand, the static portions of an image in the data set is well described through a linear combination of the eigenbasis vectors, thus being removed when using the absolute threshold difference. The eigenspace is an appropriate, robust model of the probability density function of the static background. To reiterate, once the eigenbackground images are obtained, as well as the mean image m, each new input image is projected onto the space spanned by the eigenbackground images in order to model the static parts of the input image that belongs to the background as deemed by the eigenbackground following Equation 4. Then, by computing and thresholding the absolute difference between the input image and the projected image, the moving objects present in the input image can be represented as follows:

$$|y_i - \tilde{y}_i| > T \tag{6}$$

Finally, the result is reshaped back into the original image format of $w*h$, where the first h elements from y is placed into the first column of F (the final image) and the second h elements into the second column of F and so forth.

4.2 Thresholding

It becomes apparent from Equation 6 that thresholding is paramount when it comes to determining whether a particular pixel can be described as being part of the foreground. Two methods of thresholding were considered: online adaptive thresholding and the Otsu's method.
The online adaptive thresholding technique is iterative, and is as follows:

$$T = \frac{\alpha \Sigma |y - \tilde{y}|}{wh} \tag{7}$$

where alpha is an arbitrary learning rate, and the threshold is determined alpha multiplied by the sum of the absolute value difference of the input image and the projected image, then dividing it by the multiplication of the scalar dimensions of the image. However, by having to find a learning rate α that works well, this defeats the whole purpose reducing human input. Therefore, the Otsu's method was determined to be a better approach to finding a threshold, which would also help decrease the amount of manual human input needed.
The Otsu's method determines the optimal threshold value for grayscale images by searching for the threshold that minimizes the intra class variance defined as the sum of the variance of two classes:

$$\sigma_\omega^2(t) = \omega_0^2(t)\sigma_0^2(t) + \omega_1^2(t)\sigma_1^2(t) \tag{8}$$

The weights ω_0 and ω_1 are the probabilities of the two classes separated by a threshold t, and σ_0^2 and σ_1^2 are variances of the two classes.
The class probability $\omega_{0,1}(t)$ is computed from the L bins of the histogram:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i) \tag{9}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i) \tag{10}$$

Otsu also shows that minimizing the intra-class variance (the goal) is the same thing as maximizing the inter class variance:

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 = \omega_0(t)\omega_1(t)[\mu_0(t) - \mu_1(t)]^2 \tag{11}$$

This can also be expressed in terms of class probabilities ω and class means μ. The class mean $\mu 0, 1, T$ is:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)} \tag{12}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)} \tag{13}$$

$$\mu_T(t) = \sum_{i=0}^{L-1} ip(i) \tag{14}$$

Finally, the proceeding relationships can be verified such as:

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T \tag{15}$$

$$\omega_0 + \omega_1 = 1 \tag{16}$$

This can be implemented easily on a computational software such as Matlab, where each calculation is done iteratively. The algorithm would first compute the histogram and probabilities of each intensity level through built in functions such as imhist(). Then, an initial $\omega_i(0), \mu_i(0)$ would be set. Then, it would step through all possible thresholds of $t = 1, \ldots 255$, while continuously updating ω_i, μ_i and compute the resulting $\sigma_b^2(t)$ from the values at each step. The desired Otsu's grayscale threshold value would correspond to the maximum $\sigma_b^2(t)$ value. This is implemented by using the built in function Otsu() or graythresh() on Matlab. For this design, the Otsu algorithm was not applied iteratively to find the threshold but globally, using the median image.

4.3 False Positive Suppression: Combination Method

A major bottleneck when dealing in image processing, especially with datasets as few as 10 to 40 image frames, is false positive detection. Given that our dataset was primarily compiled using camera traps in dynamic, cluttered natural settings, there are numerous components that can contribute negatively in false positive detection. The changes in sunlight, wind, shadows, animal disruption of the setting, can all cause the Eigenbackground model to incorrectly classify a pixel location as being part of the foreground.

In order to reduce the false positive detection, a combination method of the Eigenbackground model and the basic median image frame differencing was used. The Eigenbackground model would detect false positives across similar pixel locations in adjacent frames. To reduce this effect, the binary masks of each frame was combined at the pixel level with a logical AND operator with its two most adjacent frames. Then, each of the original binary mask was subtracted

by this logical AND mask created from its two adjacent masks. Assuming that the foreground animal (if every present in any of the frames) moved a significant amount in its pixel location, this logical operation would effectively get rid of the false positive detection due to the changes in the dynamic background.
It was noted that the median image was computationally light, and carried low weight. Using the same thresholding method, the median image frame difference would pick up any stray foreground pixels that might have be neglected, giving the Eigenbackground method support. This basic frame differenced binary mask was combined with the logical operator Eigenbackground mask using a pixel-wise OR operator. The following flowchart describes the above process.

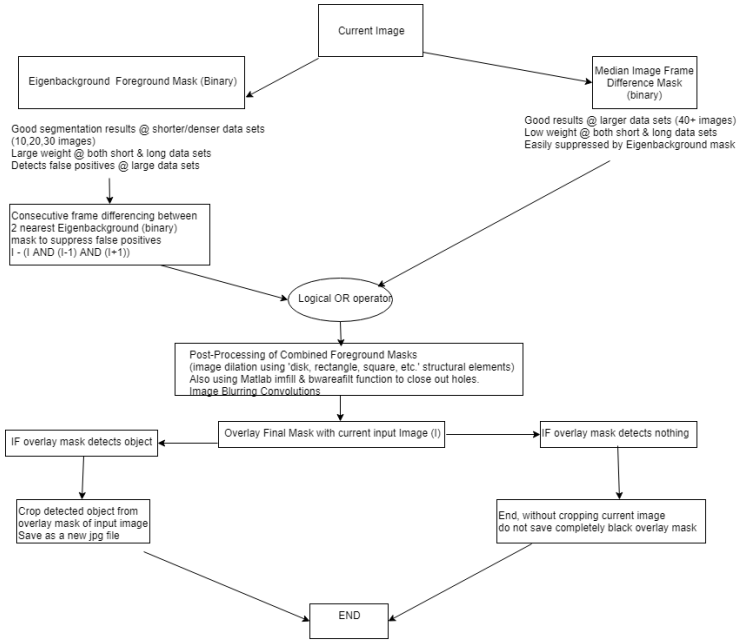


Figure 4: Flowchart of the Combination Method

By following this process, this design would effectively take short data sets with frames as few as 10, and result in a binary mask containing only the features of the foreground animals (if present), and leave the rest of the background as completely black.

4.4 Convolutional Neural Network

A convolutional neural network is a machine learning neural network system, that is widely used in image recognition tasks. In simplified terms, a convolutional neural network takes an input array, and tries the learn the relationship between the image and some target data (classification). This section will go over the architecture of a standard convolutional neural network (CNN), and how each layer contributes to the model.

Input Layer: The input image is placed into this layer. It can be a 2D(grayscale)

or 3D(RGB) image. The difference in the two type of inputs are the expected kernel shapes that the CNN will use to convolve the images. Inputs to a CNN tend to work best when they are of certain predetermined sizes. The aforementioned Eigenbackground object detection model provides the predetermined dimensions of the input images as needed.

Convolutional Layer: Intuitively, the purpose of a convolutional layer is to detect the features of the images such as lines, color drops, edges, shapes, and textures. Any of the features that the layer has learned at this point can also be recognized at any later part of the neural network. A convolutional layer works by applying certain kernels to each of the pixel values of the input image. Different convolutional kernels (or filters) will arrive at different features (vertical v. horizontal edges, etc). The result of this convolution (multiplication and summation) of pixel values is the obtainment of new features from the original input image. Usually some information is lost along the edges due to convolution kernel sizes, as it cannot apply vector multiplication along the edges where there are no adjacent pixel values. In this design, a kernel size of $[3x3]$ was used.

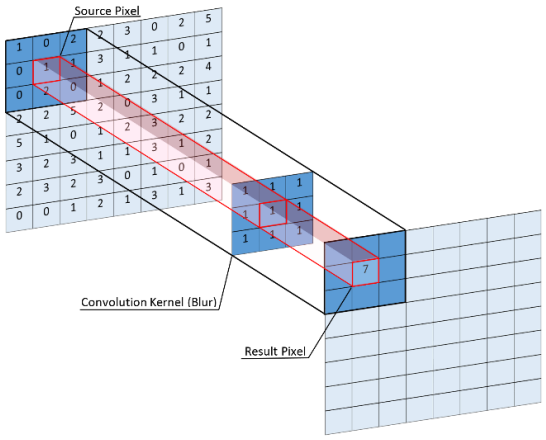


Figure 5: Convolutional Layer

Non-linearity: The "non-linearity" is not a distinct layer of the CNN. It is a part of the convolutional layer, and the operation is done on the output of the neurons. However, this data is not taken forwards (linearly) but rather non-linearly, such that it will help with operations later on. The non-linear activation function that is applied in this design in called rectified linear unit, also known as ReLu. ReLu is an activation function that is defined as the positive part of an argument. In our case, the negative pixel values resulting from the convolution is set to zero, while the positive values remain the same. This result is not taken forwards into the next layer, but is kept as information for back propagation. ReLu is traditionally used because it does not require expensive computation, and has been shown to speed up gradient descent algorithms.

$$f(x) = x^+ = max(0, x) \tag{17}$$

Pooling Layer: The pooling layer's purpose is to make sure that the subsequent

layers of the CNN are able to identify a larger-scale detail than simple curves and edges. The pooling layer operates by merging pixel regions in the convolved image together, thereby shrinking the image. This stage takes another kernel of a specified size (in our case $[2x2]$), and passes it over the entire image, like the convolution. A $[2x2]$ kernel will effectively half the size of the convolved image. In this design, *maxpooling* was used, which merged the kernel with the maximum value. The convolution and maxpooling are done in the same layer

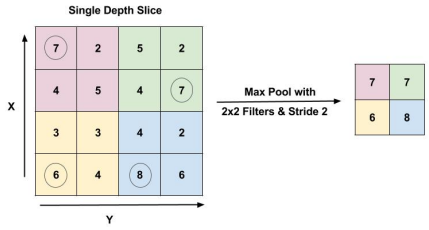


Figure 6: Max-Pooling Layer

in the CNN. Users can build multiple convolutional/pooling layers, and in this design, a total of 4 convolutional layers were used.

Fully Connected (Dense) Layer: After detecting the high level features through the application of multiple convolutions and pooling, a fully connected layer is attached to the end of the network. This is often called the hidden layer, because it is notoriously uninterpretable, and works like a "black box". This layer takes an input volume (the output of the final convolution/pooling layer) and outputs it to an N dimensional vector, where N is the number of desired classes. Each number in this N dimensional vector represents the probability of a certain class. The total of these outputs is always 1. Increasing the number of neurons in this layer allows the fully connected layer to perform many different combinations of features, and it correspondingly creates a more complex non-linear function that represents the feature space. The number of nodes in this layer can be whatever the user wants, and is not dictated by any previous dimensions. This is where the actual processing is done via a system of weighted 'connections' between the feature parameters and the fully connected neurons in the layer. In this design, only 1 hidden layer was implemented with 256 neurons.

Output Layer: The output layer consists of a number of nodes which have a high value if they are activated. In a binary classification problem, this could either be 1 or 2 neurons. In this design, only 1 output neuron was used to determine whether an image could be classified as either an animal or a human. If the output was greater than 0.5, the output neuron was activated and the image was classified as a human, otherwise it was not activated and the image classified as an animal. Since this design is for binary classification between animals and humans, the sigmoid activation function was used. The sigmoid

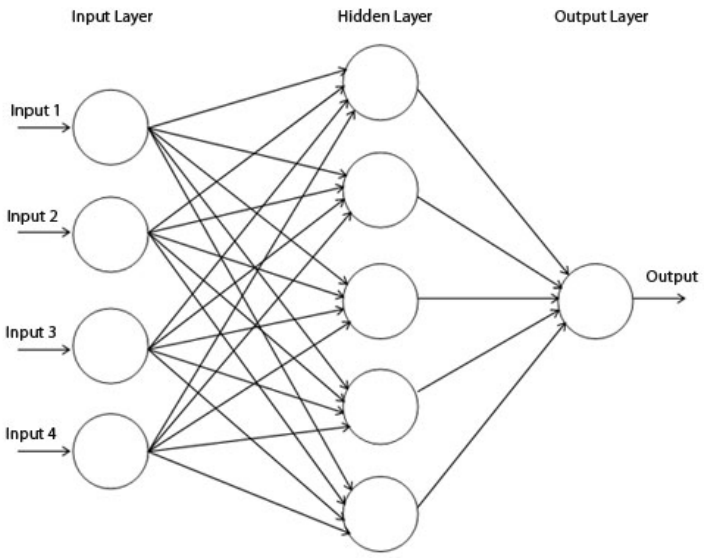


Figure 7: 1 Hidden Layer with 1 Output Neuron

function is defined as:

$$S(x) = \frac{1}{1 + e^{-x}} \tag{18}$$

The output neuron will be a scalar value between 0 and 1, and the activation of the output neuron will be determined using the midpoint value of 0.5

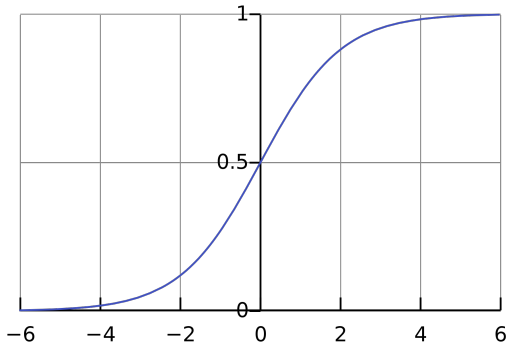


Figure 8: Sigmoid Function

In reference to Figure 8, the input layer displayed there would be the output of feature parameters generated by the multiple convolutional layers. The hidden layer would have 512 circles representing the neurons, each processing the weighted sums of the parameters from the layers preceding it. The output layer is accurately displayed as having 1 output neuron, which would either be activated or not depending on what the classification value of the input image is.

Training: With the completion of the CNN architecture, the model is trained so that it can start predicting on future testing datasets. The training process is called back propagation, and largely works in four distinct sections: forward pass, loss function, backward pass, and weight update.
First, during the forward pass, our labeled data (training image) is passed through the entire network. In the first training example, the weights on the hidden layer neurons would be randomly initialized. The network, with its current random weights, is not able to look for significant features, and cannot make a reasonable conclusion about what classification the image may be.
Then, it the loss function component of training comes in. In this design, since it is a binary classification problem, the binary cross-entropy loss function was used, which is defined as:

$$H_p(q) = -\frac{1}{N}\sum Ni = 1y_i log(p(y_i) + (1 - y_i)log(1 - p(y_i)) \tag{19}$$

Where y is the label, and $p(y)$ is the predicted probability of the point being true for all N points.
The loss will be extremely high for the first couple of training images, since the weights will not be updated well enough for significant improvement in prediction. The desired outcome is for the predicted label to match the training label. Minimizing the loss is the way to optimize the problem in calculus terms, and finding out which inputs/weights on the neurons that most directly contributed to the loss of the neural network becomes the next task.
The backward pass computes the rate of the loss in accordance to the weights, and in this design, the RMSprop optimizer was used. RMSprop optimizer is a form of gradient descent algorithm, which looks to minimize the loss caused by the predictions on the training dataset by following the derivative of loss according to weight, until loss is no longer decreasing (or at the minimum value). Once the backward pass is completed, the weight update occurs, replacing the old weights on the neurons with newer ones that are calculated to make the prediction results more accurate on the training dataset such that the prediction labels match the truth.
The process of forward pass, loss function, backward pass, and parameter update is one training iteration. The program repeats this process for a fixed number of iterations for each set of images (also called batches). In this design, a batch size of 128 images were used for a duration of 15 epochs with 8 steps in each epoch.

5 Results

5.1 Eigenbackgrounds/Combination Method

The Combination Method utilizing the Eigenbackground Method arrived at satisfactory results for the short training datasets. Below are some of the datasets alongside the binary masks created for each of the frames.

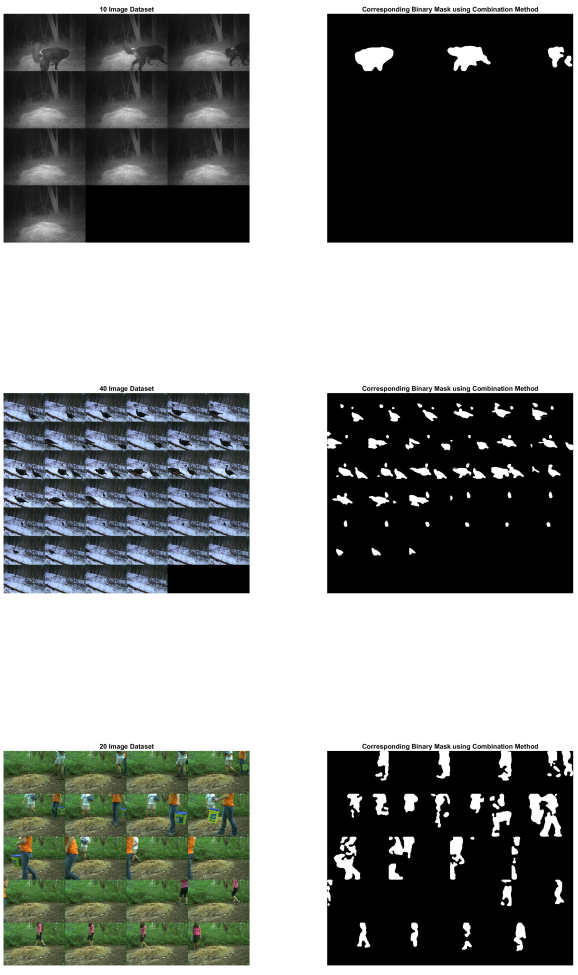
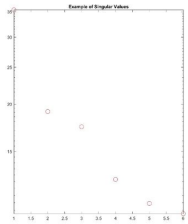
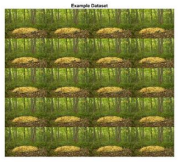


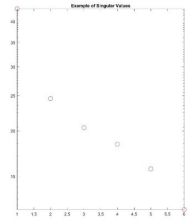
Figure 9: Datasets and Corresponding Binary Foreground Masks

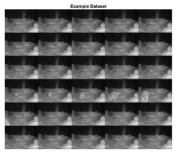
Clearly, the Combination method succeeds in capturing all forms of foreground objects, and also rejects the image frames when there are no moving objects by outputting a completely black frame.
Referring to Figure 10 below, the second table shows a higher false negative rate, but it was inevitable using the combination method that if a small animal did



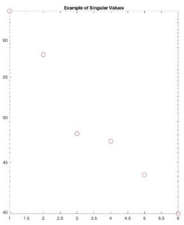


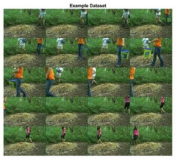
Detection vs. Truth	0 (Truth)	1 (Truth)
0 (Detected)	11	0
1 (Detected)	0	9
Accuracy (%)	100	
False Positive Rate (%)	0	
False Negative Rate (%)	0	



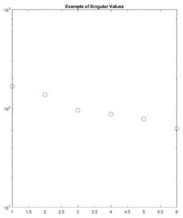


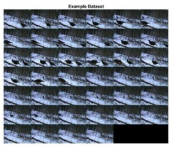
Detection vs. Truth	0 (Truth)	1 (Truth)
0 (Detected)	8	2
1 (Detected)	0	20
Accuracy (%)	86.66667	
False Positive Rate (%)	0	
False Negative Rate (%)	20	





Detection vs. Truth	0 (Truth)	1 (Truth)
0 (Detected)	1	0
1 (Detected)	0	19
Accuracy (%)	100	
False Positive Rate (%)	0	
False Negative Rate (%)	0	





Detection vs. Truth	0 (Truth)	1 (Truth)
0 (Detected)	7	0
1 (Detected)	0	33
Accuracy (%)	100	
False Positive Rate (%)	0	
False Negative Rate (%)	0	

Figure 10: Results of the Combination Method

not move significantly enough over three frames it could possibly be rejected from the foreground classification. However, the results are clear in showing that false positive detection was clearly suppressed over all of the datasets.

The results also display the versatility of the combination method in terms of the contexts of the datasets. Across the datasets shown in Figure 10 varies across time of day, size of foreground animals, number of foreground animals, and length of datasets. The only common characteristic across the datasets shown was the dynamic, cluttered natural setting. This method could easily be implemented to camera sources observing less dynamic settings such as indoor security cameras, and even traffic cameras.

Quantifiably, the lowest accuracy rate is above 86%, with most of the datasets reaching peak performance at 100% accuracy. These foreground binary masks were overlaid with the input images to create the dataset for the CNN.

5.2 Convolutional Neural Network

The training datasets assembled through the Combination method was put into a Dropbox folder so that it may be used with Google Colab, Python, and Tensorflow on an interactive online interface.

```
[6]  #printing total number of animals and humans in each file
     print('total training animals images:', len(os.listdir(train_animals_dir)))
     print('total training humans images:', len(os.listdir(train_humans_dir)))

     total training animals images: 350
     total training humans images: 263
```

Figure 11: Dropbox folder containing Training Set Images

In Figure 12, the convolutional neural network architecture is described. The input image dimensions are prescribed in the first layer. The network contains 4 convolutional layers, each applying 4 different types of kernels of size 3 by 3. Each convolutional layer is followed by a (max) pooling layer that applies a size 2 by 2 kernel across the image. Each of the convolutional layers also have the non-linear ReLu activation function to help speed up the gradient descent algorithm (RMSprop Optimizer in this case). Finally, the hidden layer contains 256 neurons, and only 1 output neuron is set since that's all that is needed for a simple binary classification problem.

```
[10] #add convolutional layers, and flatten the final result to feed into the densely connected layers
     #since we are doing binary classification (animals v humans) we are going to use sigmoid function (0 v 1)

     model = tf.keras.models.Sequential([
         #the input shape is the desired size of the image with 3 bytes of color
         #first covolution
         tf.keras.layers.Conv2D(4, (3,3), activation = 'relu', input_shape = (300, 300,3)),
         tf.keras.layers.MaxPooling2D(2,2),
         #second convolution layer
         tf.keras.layers.Conv2D(4, (3,3), activation = 'relu'),
         tf.keras.layers.MaxPooling2D(2,2),
         #third convolution layer
         tf.keras.layers.Conv2D(4, (3,3), activation = 'relu'),
         tf.keras.layers.MaxPooling2D(2,2),
         #fourth convolution
         tf.keras.layers.Conv2D(4, (3,3), activation = 'relu'),
         tf.keras.layers.MaxPooling2D(2,2),
         #fifth convolution
         tf.keras.layers.Conv2D(4, (3,3), activation = 'relu'),
         tf.keras.layers.MaxPooling2D(2,2),
         #flatten the results to a DNN
         tf.keras.layers.Flatten(),
         #512 neuron hidden layer (Dense == neurons)
         tf.keras.layers.Dense(256, activation = 'relu'),
         #only 1 output neuron, this will contain a value from 0-1 where 0 is class 'animals' and 1 is the "other" a.k.a class 'humans'
         tf.keras.layers.Dense(1, activation= 'sigmoid')
     ])
```

Figure 12: Convolutional Neural Network Architecture

In Figure 13, the parameters are shown as they go through the entirety of the convolutional neural network. As expected, each convolution takes 2 pixels out (1 from x axis, 1 from y) and each maxpooling layer halves the dimensions. The third dimension is described by the different types of convolution kernels applied to the input, which in this case is 4.

```
[11]  #the model.summary() method call prints a summary of the neural network
      model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 298, 298, 4)       112
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 149, 149, 4)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 147, 147, 4)       148
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 73, 73, 4)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 71, 71, 4)         148
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 35, 35, 4)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 33, 33, 4)         148
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 4)         0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 4)         148
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 4)           0
_________________________________________________________________
flatten (Flatten)            (None, 196)               0
_________________________________________________________________
dense (Dense)                (None, 256)               50432
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 51,393
Trainable params: 51,393
Non-trainable params: 0
_________________________________________________________________
```

Figure 13: Parameter dimensions throughout the neural network

In Figure 14, the loss and accuracy of the training model is displayed as the model achieves higher accuracy with each proceeding epoch.

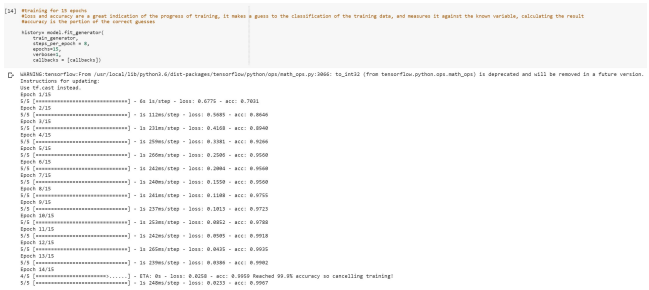


Figure 14: CNN Training

The following images in Figure 15 show the resulting feature space presented by the first convolutional layer (top figure) as opposed to the fourth convolutional layer (bottom figure).
While the additional features shown within the bottom half of Figure 15 may be meaningless to the human eye, it represents the deeper connections within the training dataset that represent more than just simple lines, patterns, textures, edges, and color drops. These are the parameters that the model uses to identify each class, and these are the parameters that get fed into the hidden layer such that they may be used as weighted sums to make the predictions.

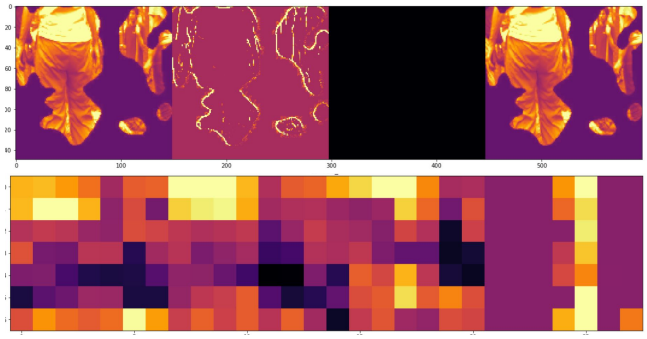


Figure 15: Feature Space represented at different layers

6 Conclusion

In sum, by performing singular value decomposition and principal component analysis, the Eigenbackground model can create a robust probability density model that can effectively determine if a pixel location is a static portion of the background given an appropriate lower dimensional basis.
Furthermore, given the dynamic nature of the cluttered background scene in the majority of the camera trap images, and the short range in image dataset size, the Combination method utilizing logical operators can provide an adequate false positive suppression, leading to a more accurate foreground object detection. This method works across multiple variations in context, including various foreground object number and size, time of day, and length in dataset. By arriving at a masked image containing only the relevant foreground objects, the images can then be used to compile a training data set for a convolutional neural network, a machine learning process used largely for computer vision tasks. The CNN can be further improved by varying parameters such as number of kernels, kernel size, pooling types, number of convolutional layers, number of hidden layers, number of hidden layers and number of neurons.

References

[1] Andre P 32133. "Relationship between SVD and PCA. How to Use SVD to Perform PCA?" Cross Validated, stats.stackexchange.com/questions/134282/relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca134283.

[2] Deshpande, Adit. "A Beginner's Guide To Understanding Convolutional Neural Networks." A Beginner's Guide To Understanding Convolutional Neural Networks – Adit Deshpande – CS Undergrad at UCLA ('19), adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/.

[3] Gan, Ming-Gang, et al. "Moving Object Detection Algorithm Based on Three-Frame-Differencing and Edge Information." Journal of Electronics Information Technology, vol. 32, no. 4, 2010, pp. 894–897., doi:10.3724/sp.j.1146.2009.01202.

[4] Joubert, Niel. "Background Modelling and Subtraction for Object Detection in Video." Final Year Project 2009, applied-maths.sun.ac.za/ wbrink/students/NJoubert2009.pdf.

[5] Moroney, Laurence. "Introduction to TensorFlow for Artificial Intelligence, Machine Learning, and Deep Learning." Coursera, Coursera, www.coursera.org/learn/introduction-tensorflow.

[6] Wang, Lei, et al. Adaptive Eigenbackground for Dynamic Background Modeling. Department of Automation, Tsinghua University, link.springer.com/content/pdf/10.1007/978-3-540-37258-5$_7$4.*pdf*.