Detecting Wildlife in Uncontrolled Outdoor Video using Convolutional Neural Networks

Connor Bowley[*], Alicia Andes[+],
Susan Ellis-Felege[+], Travis Desell[*]
Department of Computer Science[*]
Department of Biology[+]
University of North Dakota

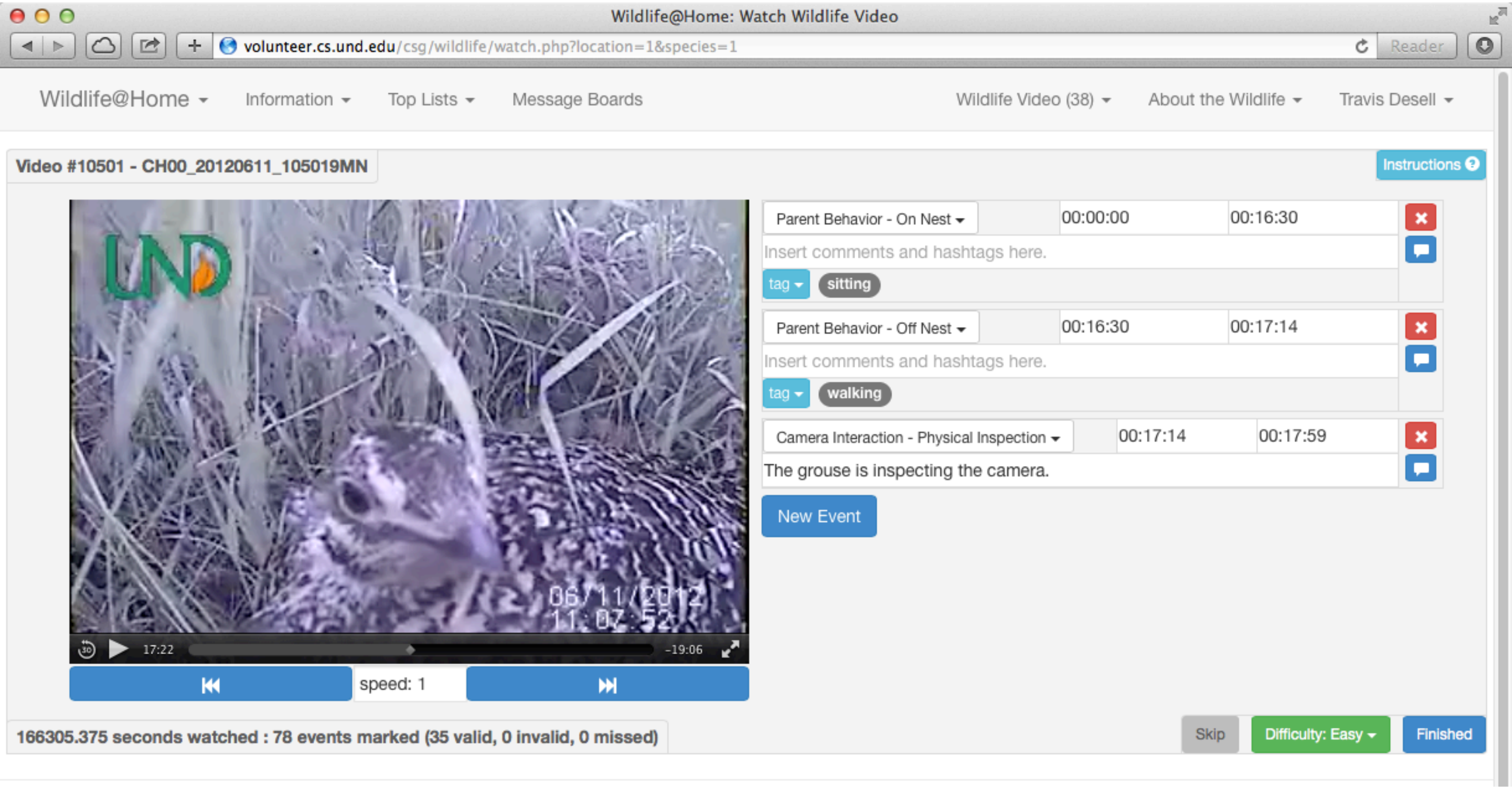
UND UNIVERSITY OF NORTH DAKOTA

Wildlife@Home

- Citizen Science project combining crowd sourcing and volunteer computing.
- Users can examine videos and images and record what happens
- They can also volunteer their computer to download videos and run algorithms over them
- There is a web portal to compare results from the users, experts, and computer vision algorithms

Wildlife@Home

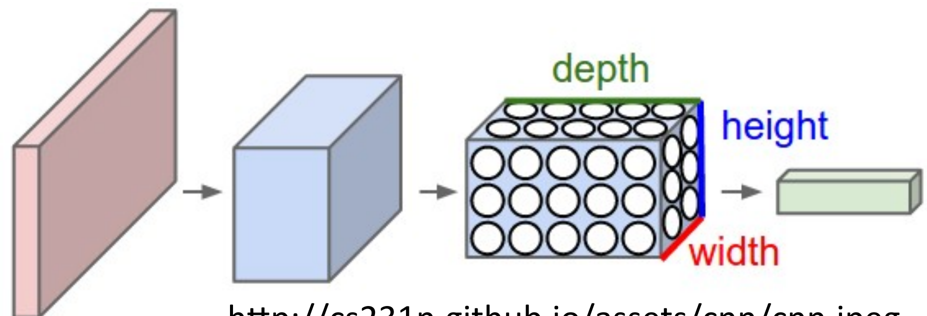
- Nest cameras
- Around 7.8 years of video time gathered over 3 years
  - Over 91,000 videos of Grouse, Interior Least Tern, and Piping Plover
  - A little over 4.5 TB
- Challenges with dataset
  - Changing weather
  - Changing lighting as day progresses, cloud cover
  - Some species are camouflaged
  - Video quality can be low



Crowd sourcing interface users can give us information about the video through. The biology experts have a similar interface.

Convolutional Neural Networks

- CNNs commonly used for image classification
- A few types of layers
  - Convolutional (has weights to be trained)
  - Activation
  - Max Pooling
  - Fully Connected



http://cs231n.github.io/assets/cnn/cnn.jpeg

- Softmax or SVM usually used at the end
- Local connections, shared weights
- Learns from labeled training data

Creating Training Data

- Images of variable sizes
- Sub-images size 32x32 used for training
- Striding process used to get sub-images
- Careful cropping needed to minimize mislabeled data

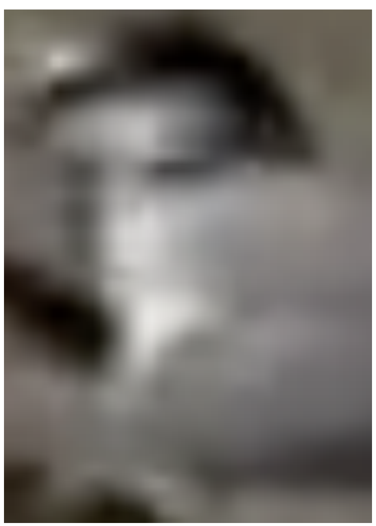
Creating Training Data

- Images of variable sizes
- Sub-images size 32x32 used for training
- Striding process used to get sub-images
- Careful cropping needed to minimize mislabeled data

Creating Training Data

- Images of variable sizes
- Sub-images size 32x32 used for training
- Striding process used to get sub-images
- Careful cropping needed to minimize mislabeled data

Creating Training Data

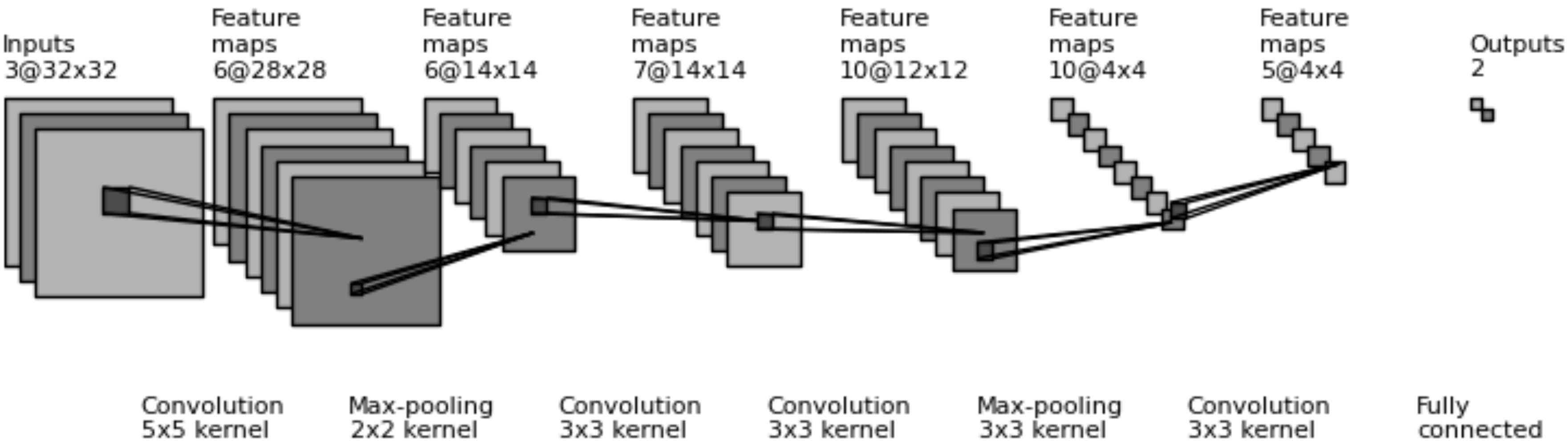


Creating and Training CNN

- Written in C++ and OpenCL
  - C++ allows distribution via BOINC
  - OpenCL allows execution on most CPUs and GPUs
- Stochastic gradient descent backpropagation
- Uses L2 regularization and Nesterov Momentum
- Weights initialized by normal distribution with mean of 0 and standard deviation of $\sqrt{2/n}$ [1]
- Two way softmax classifier
  - (tern not in frame, tern in frame)

[1] http://cs231n.github.io/neural-networks-3/

Creating and Training CNN



Layer Type	Layer Dims	Filter/ Pool Size	Stride	Number Filters	Padding
Input	32 x 32 x 3				
Convolutional	28 x 28 x 6	5	1	6	0
Max Pooling	14 x 14 x 6	2	2		
Convolutional	14 x 14 x 7	3	1	7	1
Convolutional	12 x 12 x 10	3	1	10	0
Max Pooling	4 x 4 x 10	3	3		
Convolutional	4 x 4 x 5	3	1	5	1
Fully Connected	1 x 1 x 2				

In total 2068 weights

Running the Trained CNN

- Strided over full images similar to method used to create training data
- A prediction image is created for each frame in video to create a prediction video
- A chart is also created plotting how much of each frame is predicted to be of the positive class

Running the Trained CNN

- Each pixel in full image has a “pixel classifier”
  - Softmax output in sub-image is added into pixel classifier of each pixel in sub-image
- Sub-images may overlap and their outputs are summed into pixel classifier
- Pixel color determined using ratio of squares of pixel classifier
  - red is positive class, blue is negative class

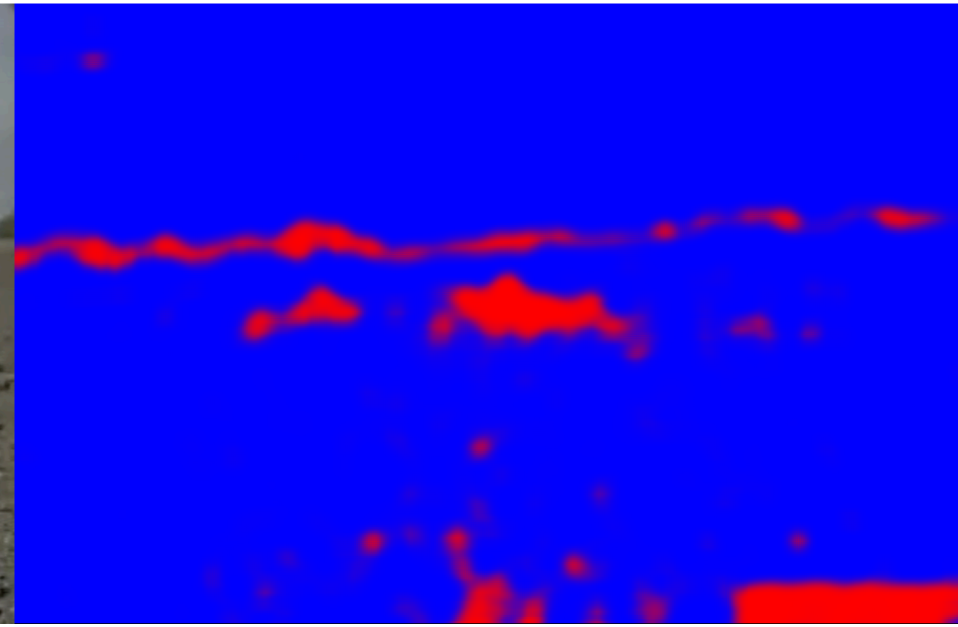
$$r = 255c_p^2 / \sum_{i=0}^{n} c_i^2 \qquad b = 255c_n^2 / \sum_{i=0}^{n} c_i^2$$

Results

- Initially trained 5 epochs over ~73,000 images from 1 video
- Ended training with accuracy of 95.6% on training data
- Run over test set of 280,000 images from 2 other videos with 82% accuracy
  - These images were not created yet during initial training
  - Videos all from same nest, so some background images might have been similar
  - 77% of errors from false positives

Results

Original Image

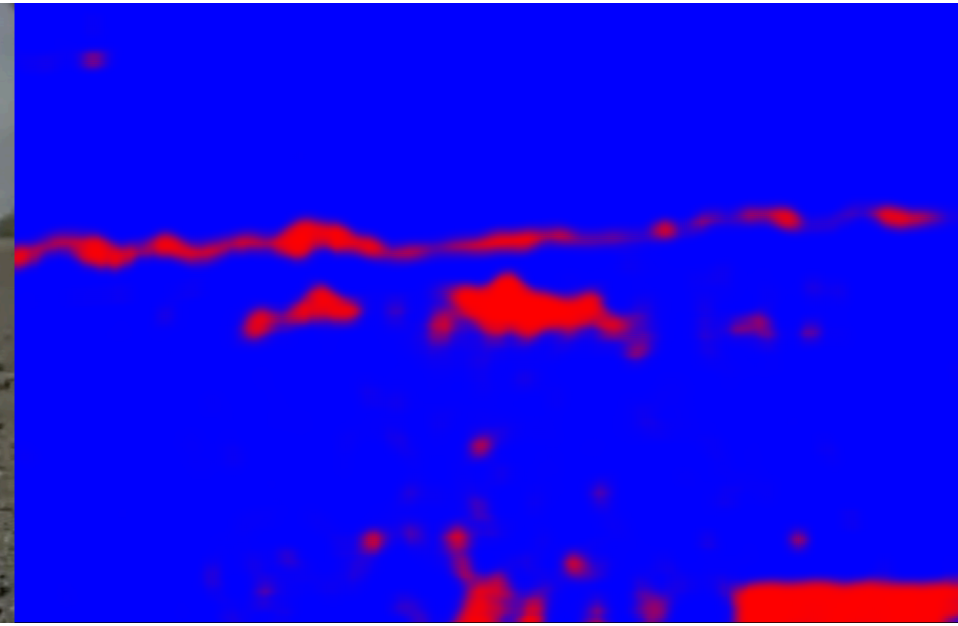


After Initial Training

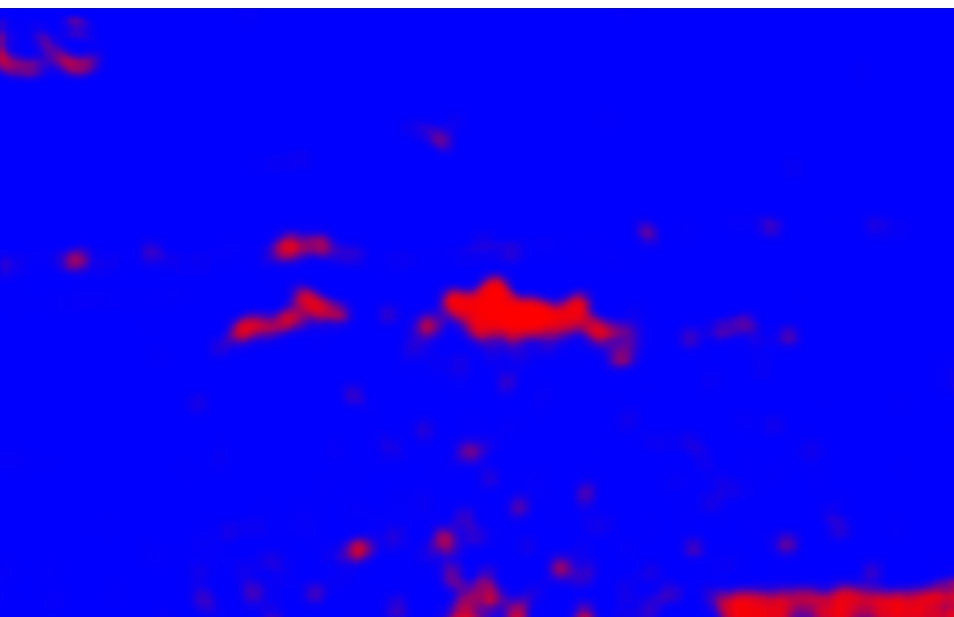
Extra Training

- Misclassification prompted extra training on CNN
- New training set of approx. 17,000 images
  - 69% negative
  - Mostly of trees and ground stubble
  - Positive examples were reused from original training set

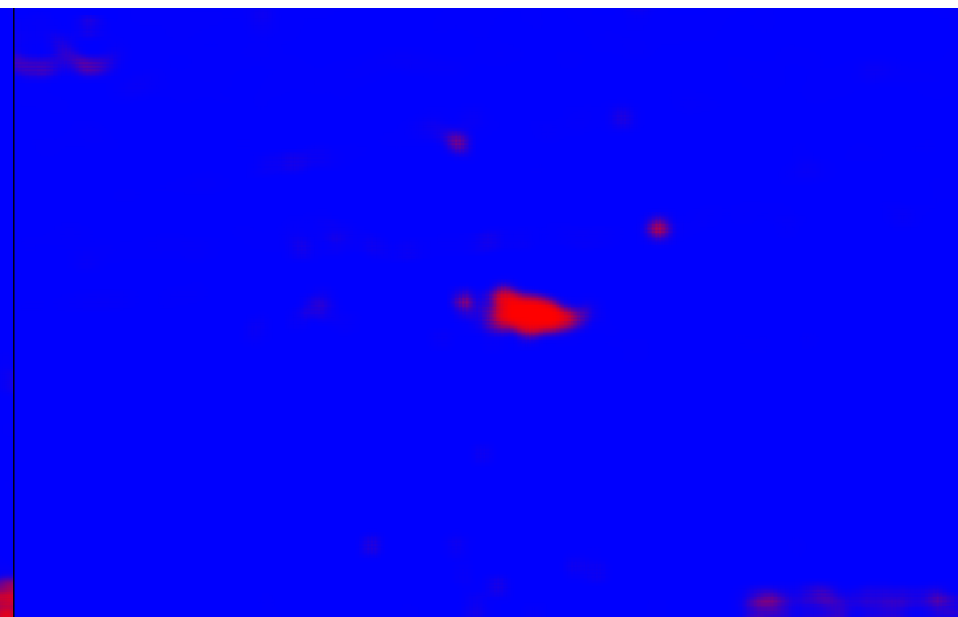
Original Image



After Initial Training

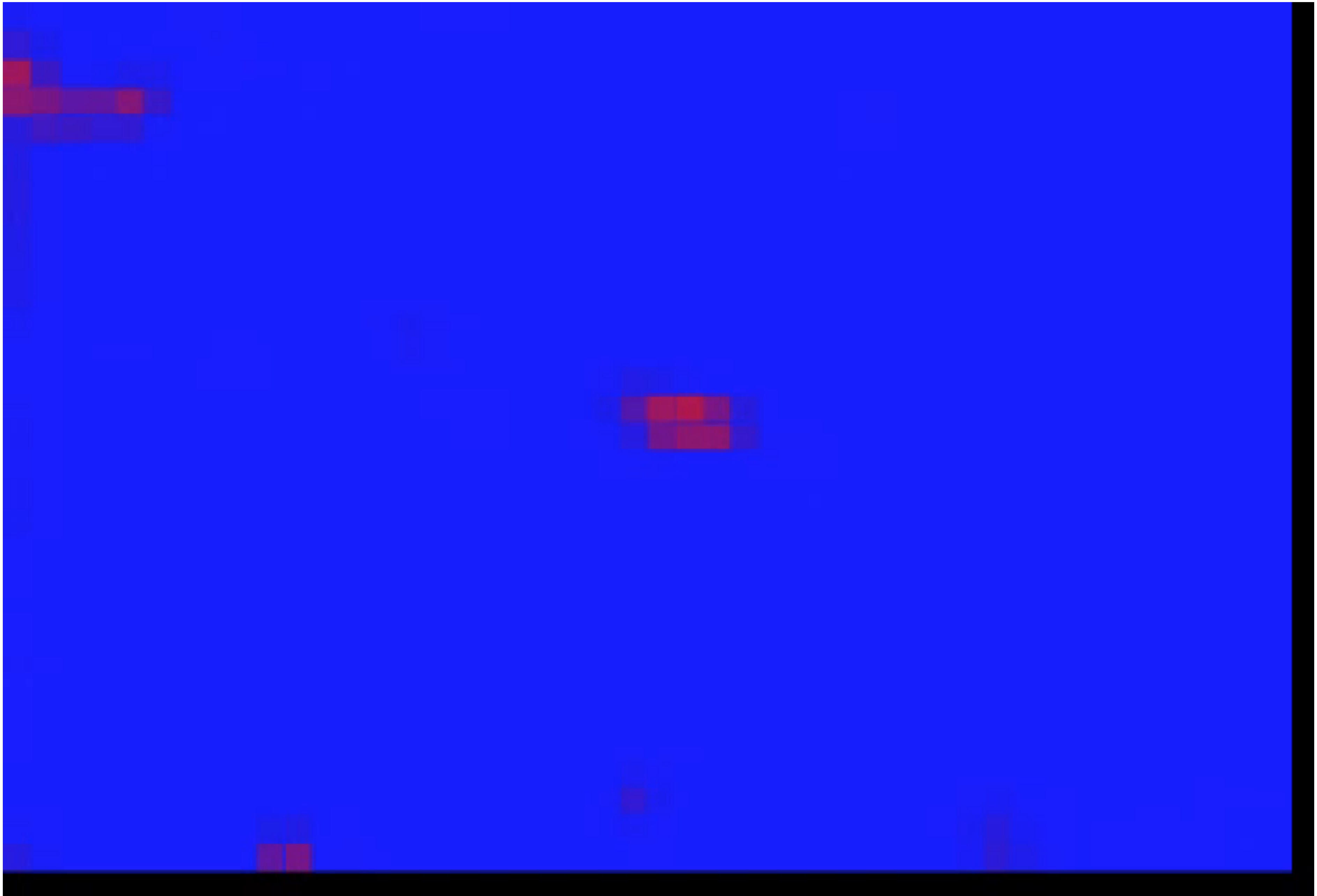


After 2 extra epochs



After 4 extra epochs

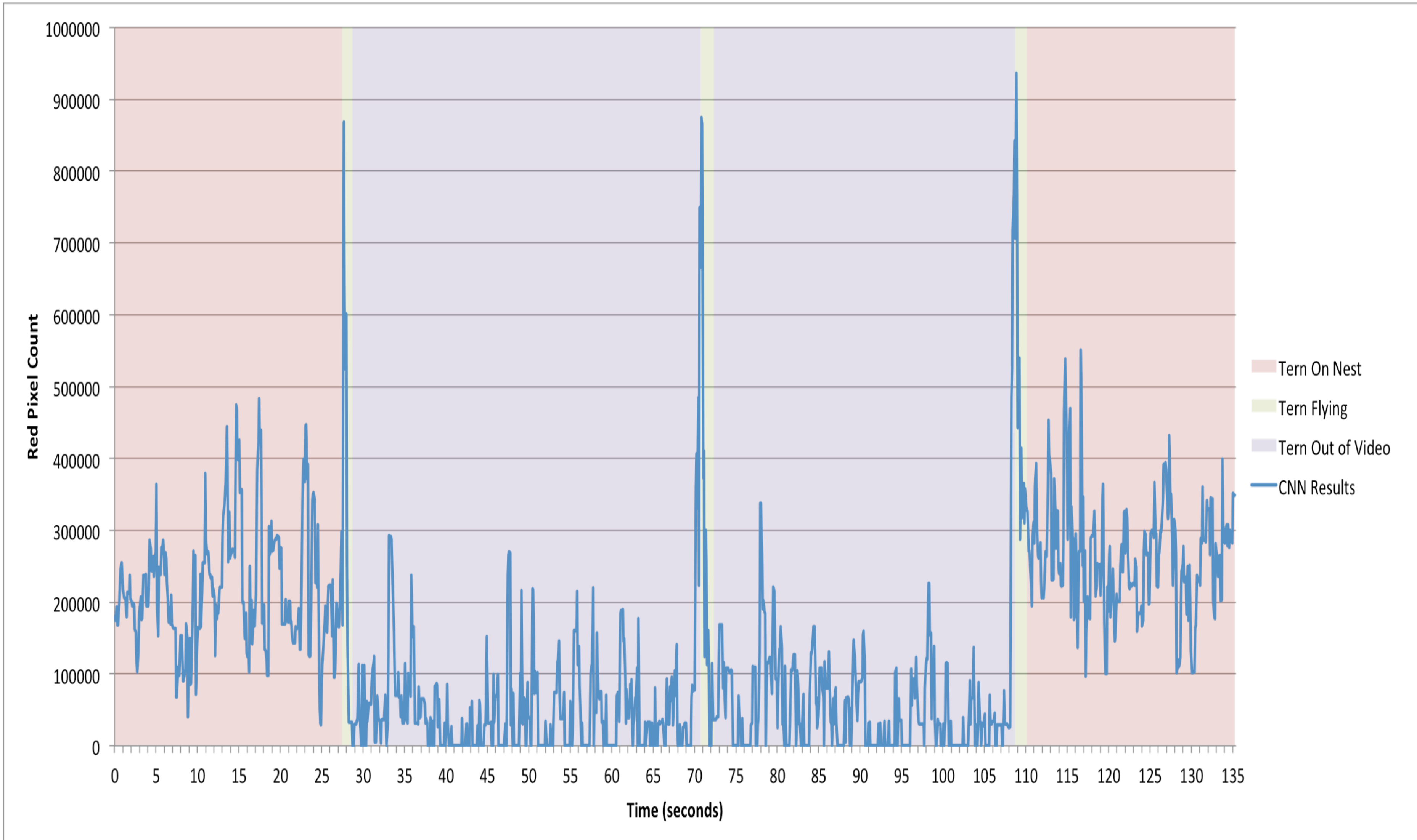
Prediction Video



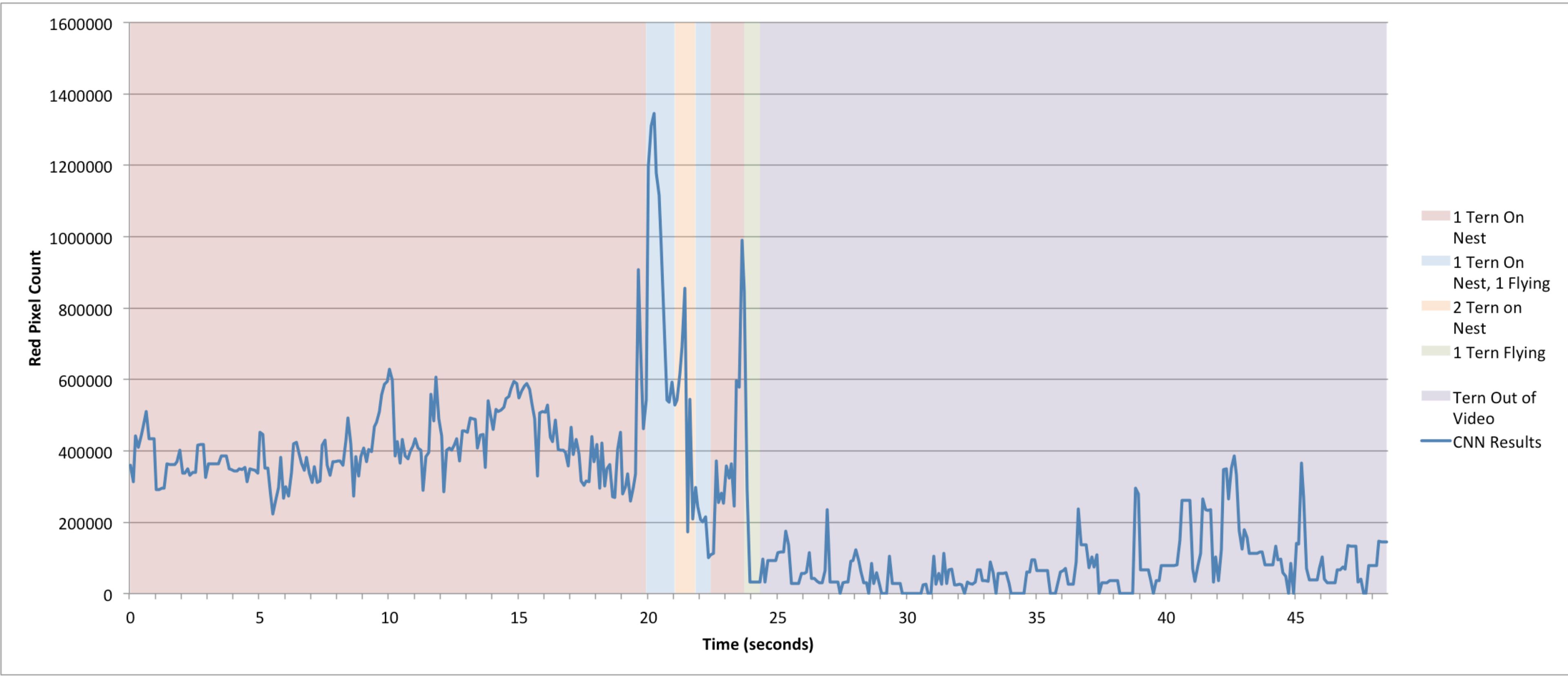
Tracking when a tern is in the frame

- Charts were made tracking how much of the image is comprised of red (positive class) pixels
- Easy to see some trends across whole video
- Difficult to classify frame by frame
- Difficult to classify more complex events

Results of Running Trained CNN over Simple Video



Results of Running Trained CNN over More Complex Video

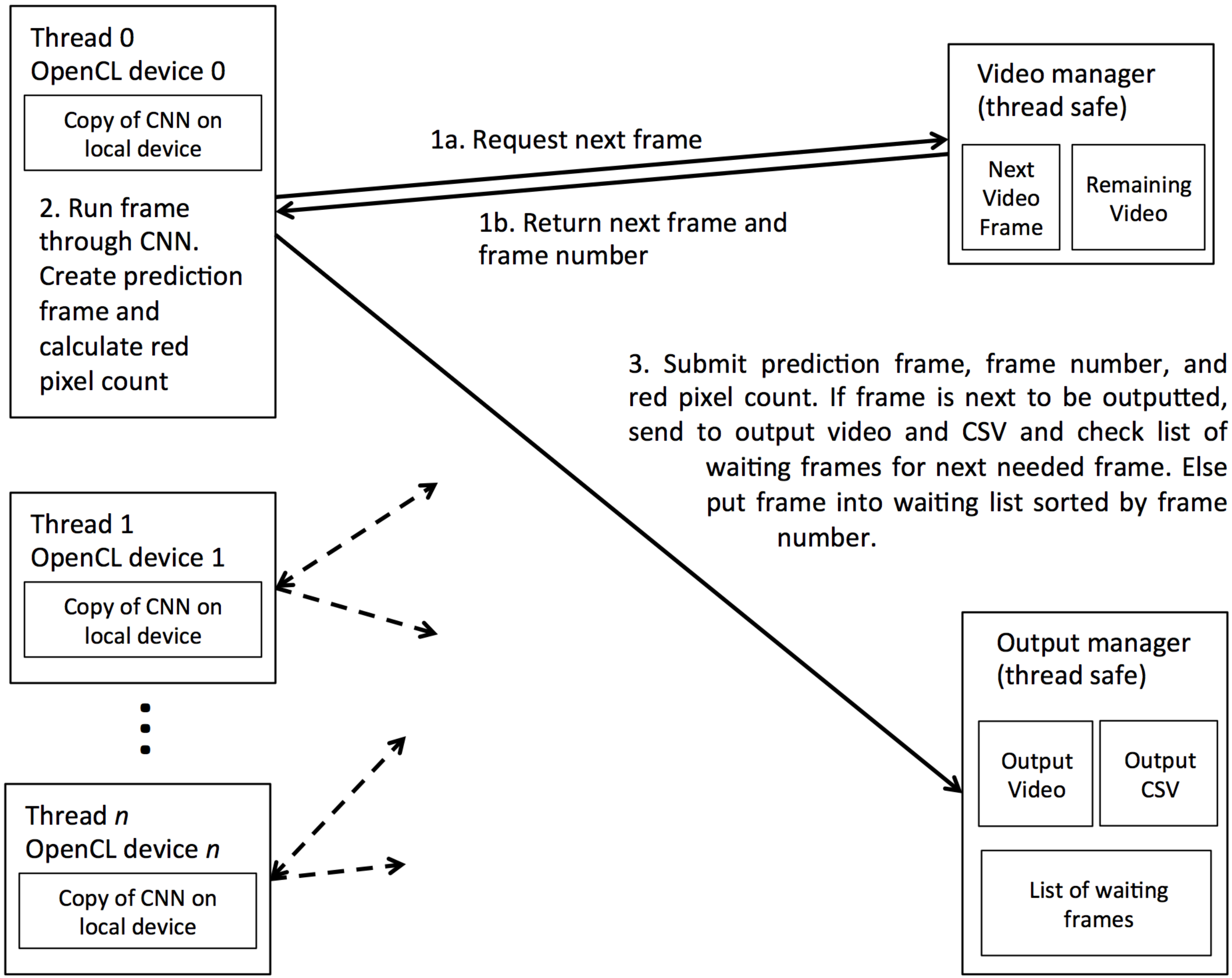


Improving Performance

- Many computers have multiple OpenCL capable devices.
  - Exp. A CPU and a GPU
- Runtime performance can be increased by using multiple devices simultaneously
- Some devices may be faster or slower than others

Improving Performance

- Work stealing approach
- Copy of CNN on each device
- Each device requests one frame at a time from Video manager
- Once finished, the results are submitted to Output manager
  - Frames that come out of order are buffered until they are next to be outputted


Thread 0
OpenCL device 0
Copy of CNN on local device
2. Run frame through CNN. Create prediction frame and calculate red pixel count
1a. Request next frame
1b. Return next frame and frame number
Video manager (thread safe)
Next Video Frame
Remaining Video
3. Submit prediction frame, frame number, and red pixel count. If frame is next to be outputted, send to output video and CSV and check list of waiting frames for next needed frame. Else put frame into waiting list sorted by frame number.
Thread 1
OpenCL device 1
Copy of CNN on local device
Thread *n*
OpenCL device *n*
Copy of CNN on local device
Output manager (thread safe)
Output Video
Output CSV
List of waiting frames

Performance Results

Computer	Devices	Time (h:mm:ss)	Seconds/Frame
Mac Pro	1 GPU	48:07	5.12
Mac Pro	CPU	32:01	3.41
Mac Pro	2 GPUs	27:34	2.93
Mac Pro	CPU and 1 GPU	20:45	2.21
Mac Pro	CPU and 2 GPUs	17:27	1.86
MacBook Pro	GPU	1:17:02	8.20
MacBook Pro	CPU	35:06	3.73
MacBook Pro	CPU and GPU	26:03	2.77

These results are from running on 56 seconds of video (564 frames) with a stride of 15 in both directions.

Future Work

- Get more training data
  - Grouse and Piping Plover
  - Crowd source creation of training data
- Full implementation with BOINC for distributed running over entire dataset
- Larger sizes than 32x32
- Speed improvements to CNN code since submission warrant testing of larger networks
- Better algorithms to determine if frames contain wildlife or if it is noise
  - CNN over output?
  - Blob detection on output?

Resources

- Code on Git
  - https://github.com/Connor-Bowley/neuralNetwork
  - Commit 8d95bf087cde7483c4984fc4891778f5280381fc (May 24, 2016)
- Videos available via Wildlife@Home Data Release
  - http://csgrid.org/csg/wildlife/data_releases.php

Acknowledgements

We appreciate the support and dedication of the Wildlife@Home citizen scientists who have spent significant amounts of time watching video. This work has been partially supported by the National Science Foundation under Grant Number 1319700. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. Funds to collect data in the field were provided by the U.S. Geological Survey.

Thanks!

Questions?

http://csgrid.org/csg/wildlife

connor.bowley7@gmail.com