Constructing a Robust Caribou Image Detection and Annotation Deep Learning System

Richard So

Terra NYC STEM Research Paper

Staten Island Technical High School

Dr. John Davis

CUNY Brooklyn College

Dr. Michael I Mandel

Abstract

Deep Learning is a subset of Machine Learning involving computers to perform tasks without explicit instruction via Deep Neural Networks (DNNs). DNNs improve themselves on a task as they receive more data examples, or ’trains’. Recent studies have used DNNs to perform Background Subtraction (BGS), the process of segmenting foreground objects from images. BGS can assist in extrapolating visual information from datasets. For instance, current investigations of caribou activity in the Alaskan wilderness are hindered by the tedious nature of manually annotating thousands of camera trap images for caribou. Thus, my project involved creating an automated caribou image detection and annotation system to aid these investigations. My caribou detection system consists of the Foreground Segmentation Network (*FgSegNet*)—one of the best performing DNN-based BGS methods—with an additional DNN that translates *FgSegNet*’s foreground masks into a numerical probability denoting the likelihood of caribou presence. To test its accuracy, I trained *FgSegNet* first, then used it to convert an image dataset into masks, which were later utilized to train and evaluate the additional DNN. This was repeated for data collected from three camera trap locations. As a result, this system achieved a mean AUC accuracy metric of 0.8503, reaching up to 0.9099 AUC in one location. For images predicted by this system to have caribou, I implemented a connected-component labeling algorithm to determine the total count and the size of each caribou. Overall, my system made significant strides in automated image labeling and could accelerate efforts in studying Alaskan wildlife activity.

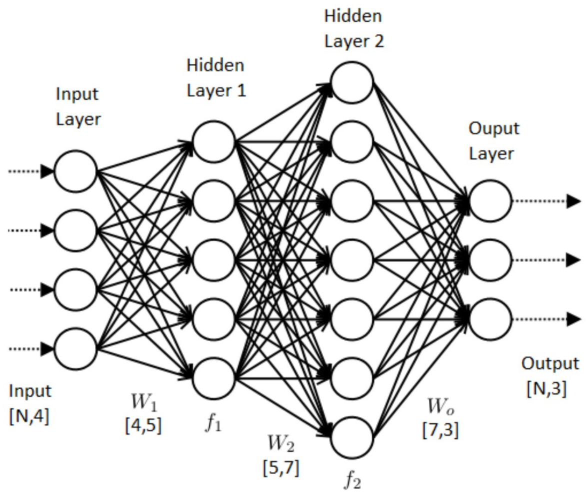
Rationale

Machine learning (ML) is the discipline involving computers to perform tasks without explicit instruction (Mayes, 2017). Instead of manually defining rules, ML-powered systems learn to recognize patterns in data by example, using such trends to perform related tasks (2017). Deep learning (DL) is a technique of applying ML which involves Deep Neural Networks (DNNs) (2017). DNNs have shown to be relatively more accurate in classification tasks compared to traditional ML approaches (2017).

Basic DNNs loosely mimic the structure of the human brain by grouping perceptrons—analogous to a biological neuron—into multiple ‘hidden’ layers and connecting all perceptrons between the layers, allowing data to pass in a single direction (see Figure 1) (Schmidhuber, 2015). Every perceptron holds numerical weights on each of its inputs to determine its output. These weights are altered when the DNN ‘trains’: a data example is inputted into the DNN, its output(s) are checked against human-provided labels, the weights are tweaked to reduce the DNN’s error, and this process repeats (2015).

Figure 1

Visual Representation of a Deep Neural Network (DNN)

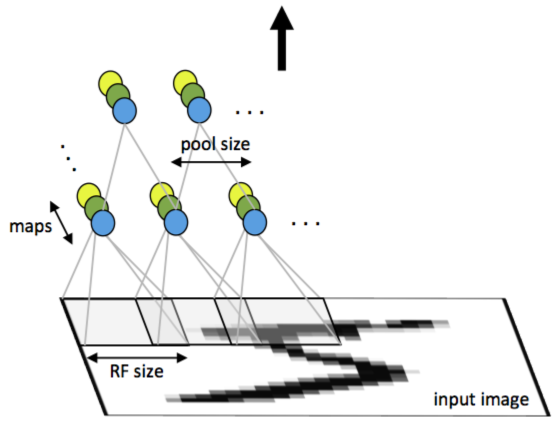


Note. Perceptrons, which resemble biological neurons, are grouped into multiple ’hidden‘ layers. This example figure consists of 2 hidden layers. In a DNN, all data flows in one direction: from the input to output layer (Ahire, 2018).

A Convolutional Neural Network (CNN) is a class of DNNs that differs noticeably in structure and is particularly effective for image data (Wu, 2017). CNN architectures include stacked convolution layers, which consist of small matrices of weights (kernels) to filter multi-dimensional input (see Figure 2). Each kernel performs a summation of an element-wise product between its weights and a patch of input, repeating this procedure for all remaining patches (2017). Through this operation, kernels extract crucial features (edges and corners) from image data (Görner, 2017). This allows CNNs to convert (or encode) images into abstract feature maps that are more efficient for classification using complementary algorithms (commonly DNNs) (2017). Thus, CNNs are widely applied for tasks involving image classification and encoding (2017).

Figure 2

Depiction of CNN Kernel Operations Across a Convolutional layer



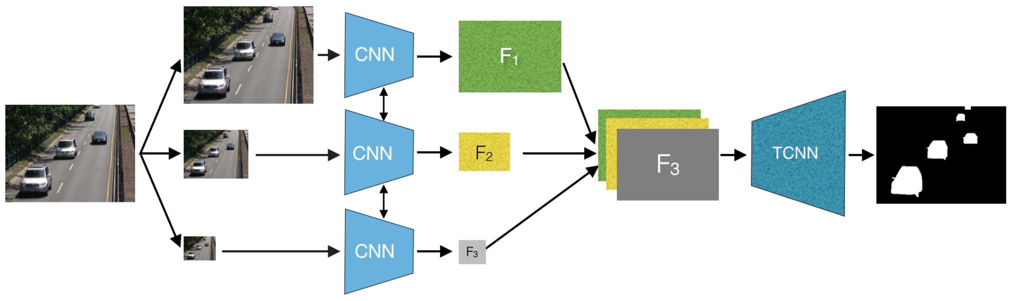
Note. The kernel traces (or convolves) through the input image, yielding a single value represented as a colored unit on the image. Multiple kernels undergo this process, producing a collection of feature maps. This layer is commonly paired with a pooling layer, which reduces the map size by a ratio—pool size—while conserving features via multiple methods (averaging, max selection, etc.) (Ng et al., 2013).

In contrast, a Transposed Convolutional Neural Network (TCNN) acts as a decoder: it decreases the abstraction of high-level representations of data (Dumoulin & Visin, 2016). TCNNs have transposed convolution layers that similarly use kernels as CNNs do. However, rows and columns of zeros are inserted between the inputted data before the kernel operations, resulting in an output that is higher in dimension but lower in abstraction (2016).

Background Subtraction (BGS) is the process of segmenting foreground objects from image data (Zeng et al., 2019). BGS algorithms are widely used for object detection and serve as a foundation for real-world object classification (2019). Recent studies have employed the use of DNNs/CNNs on the task of BGS. One of these studies was made by Lim and Keles (2018), who created the Foreground Segmentation Network, or *FgSegNet*. *FgSegNet* is currently among one of the best-performing BGS methods, achieving an average F-Measure score of 0.9758 on the CDnet2014 dataset and 0.9770 on a recent literature review made by Bouwmans et al. (2019).

Figure 3

Visual Representation of FgSegNet's Architecture



Note. FgSegNet is a feedforward Neural Network comprised of 3 CNNs in parallel and a single TCNN. Its output is a probability mask (shown at the rightmost portion of the figure) denoting the likelihood of each pixel in the image containing a foreground object (Lim & Keles, 2018).

FgSegNet is an encoder-decoder, multi-scale CNN model, using 3 CNNs (encoders) and a TCNN (decoder) component in its architecture (see Figure 3) (2018). The model accepts one image frame as input and outputs a probability mask (grayscale image) representing the likelihood of a foreground object present in each pixel. First, the model downscales the inputted image to three ratios. These are passed into 3 CNNs, all of which are similar in architecture. The outputs are then upscaled to match the dimensions of the largest output. Finally, these are passed into the TCNN, which converts the feature maps into a probability mask with original dimensions (2019; 2018).

The alarming rates of ecosystem degradation caused by human disturbance over the past few decades have led to a collective effort by the scientific community to monitor this phenomenon more closely (Buxton et al., 2018). One of the main technologies deployed to observe an environmental setting is the camera trap (2018). Camera traps (or autonomously triggered cameras) can confirm the presence of ground animals, as they are activated to record only when movement is detected near its proximity (2018). They can be deployed remotely for long spans of time, making camera traps appealing for monitoring ecosystems (2018).

There is currently an ongoing study on caribou activity in the Alaskan wilderness via camera trap analysis, led by Scott Leorna at the University of Alaska Fairbanks (S. Leorna, personal communication, August 18, 2020). Leorna aims to measure the amount of impact that human activity has had on the caribou population and migration patterns. Despite the benefits that camera traps provide, manually annotating thousands of collected images for target objects is very tedious and time-consuming. Thus, this project investigates the possibility of automating foreground caribou detection and annotation to aid current and future wildlife investigation efforts.

Research Goal / Hypothesis

The goal of this research project is to implement a robust system that is able to automatically flag and annotate images consisting of foreground objects, especially caribou. Specifically, camera trap images from the Alaskan wilderness (see Appendix A)—shared by the University of Alaska Fairbanks—will be used for training and testing this implementation. This system will utilize the state-of-the-art *FgSegNet* model as its backend for performing BGS, along with complementary algorithms to determine the final probability of foreground presence and annotate the count and size of foreground caribou from its outputs. It is hypothesized that this system will perform well against ground-truth labels determined by Leorna and his team: *FgSegNet* has performed very accurately in recent competitions and reviews, thus, the model will similarly yield respectable results (Bouwmans et al., 2019; Wang et al., 2014).

The independent variable of this experiment will be the architecture of this image flagging system. The architecture of this system is established to have two complementary algorithms assisting *FgSegNet* to detect and annotate caribou within image data, however, I will evaluate multiple complementary algorithms and modifications of *FgSegNet* in this study. The dependent variable will be the accuracy of the system in correctly predicting images that have foreground objects/caribou. The system's accuracy will be measured using the F_1 score, as well as the Receiver Operating Characteristic (ROC) and Area Under the ROC Curve (AUC). The image data used and the hardware/software environment where the system is tested will remain controlled throughout the experiment.

Methods

Training *FgSegNet*

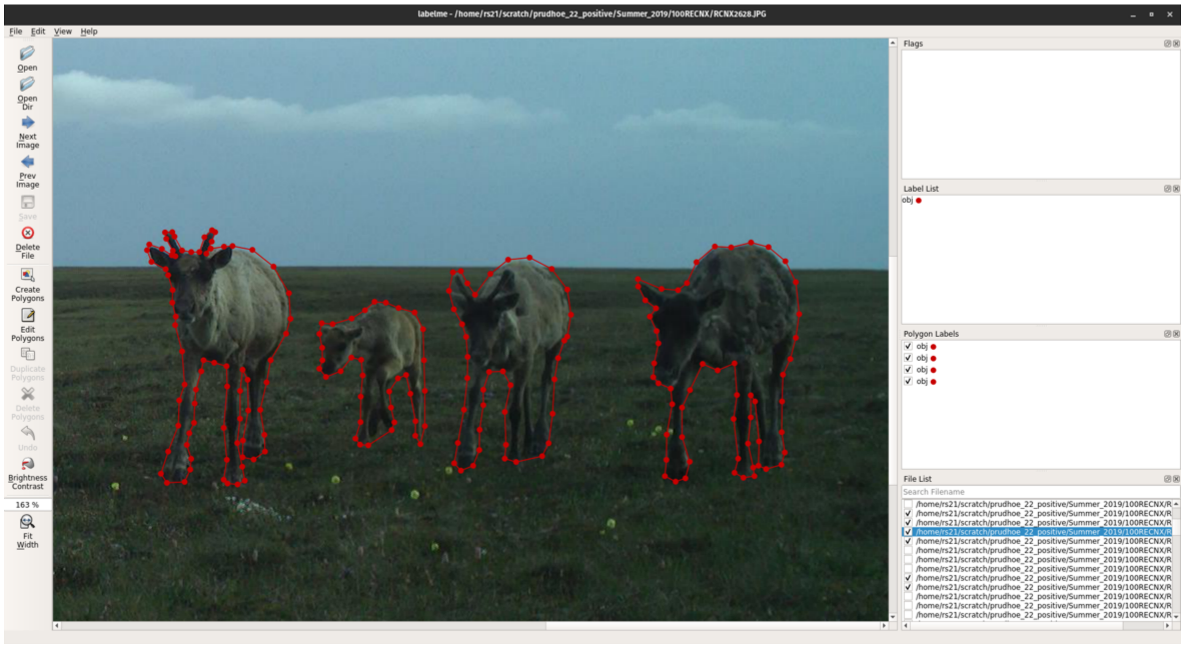
FgSegNet was tested on three different camera trap locations located in the Alaskan wilderness site, code-named: `prudhoe_12`, `prudhoe_15`, and `prudhoe_22` (see Appendix A). The three locations were chosen at random, after omitting locations with camera trap malfunctions. The camera traps at each location were deployed during the same timeframe from May to September 2019. The camera traps are also identical in model and specification: the Reconyx Hyperfire 2 Covert camera trap was used for deployment, which captures images in 2048 x 1440 pixel resolution.

FgSegNet is scene specific, meaning that it generates a finer-tuned model particular for each camera trap location. It allows the option of training with either 50 or 200 images from each location, although this parameter can be overwritten to allow for variable training data to be utilized. The training set must include the raw images, along with corresponding binary (black-and-white) masks representing only the foreground objects. All binary masks for the training sets were manually created by the researcher.

To generate the training set, all images labeled to have at least one foreground object (caribou, vehicle, etc.) were first collected from the dataset. Then, 200 images were randomly sampled from the collection to be masked—or manually extract foreground from. *Labelme* is an open-sourced graphical image annotation software, which was primarily used for manually masking foreground objects from this project's camera trap images (Wada, 2016). *Labelme* stores the mask as a set of coordinates outlining all corners of the polygon shapes comprising the mask. A Python script was created to convert this data format into a binary mask. This process is repeated for each camera trap location, resulting in approximately 600 images being manually masked in total.

Figure 4

Screenshot of the Labelme Masking Utility



labelme - /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/RCNX2628.JPG

Polygon Labels:
- ✓ obj ●
- ✓ obj ●
- ✓ obj ●
- ✓ obj ●

File List:
```
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
✓ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
☐ /home/rs21/scratch/prudhoe_22_positive/Summer_2019/100RECNX/R
```

Note. Labelme allows for a streamlined process in manual image masking. For each image, I mark 'polygons' of each foreground object by denoting points which border each object. All pixels in the marked polygons are converted to white (1) and the others black (0), yielding a binary mask.

The extent to which a larger training set would assist the model was also experimented by adding 600 negatively labeled images—data examples which do not contain any foreground objects—into the training set. It was hypothesized that a training set of mixed examples would help *FgSegNet* to generalize the image data used for the system, despite deviating from the training procedures that *FgSegNet*'s authors had documented (Lim & Keles, 2018). Thus, *FgSegNet* was trained and compared on two training sets: 200 positively labeled images, and 800 images with mixed labels. Because of hardware GPU memory limitations, the width and height dimension of each image was reduced by half for use with *FgSegNet* (2048 x 1440 to 1024 x 720 pixels). *FgSegNet* generated 3 different models, corresponding to each camera trap location. These models were stored locally as HDF5 data files and can be reused to output a foreground mask given an input image of the same 1024 x 720 pixel dimensions. *FgSegNet*'s models do not produce an exact binary mask, but a probability mask denoting the likelihood of each pixel belonging to the foreground (closer to 1) or background (closer to 0).

Determining Foreground Presence From *FgSegNet*'s Outputs

Because one of the main objectives is to determine the presence of foreground objects, multiple approaches were tested to bridge *FgSegNet*'s generated masks to a single probability denoting the existence of a foreground object. The first approach was to use a simple "thresholding" algorithm, which accepts the summed pixel probabilities of *FgSegNet*'s mask as its input. From these sums, the algorithm searches for a single (threshold) value that most accurately distinguishes summed probabilities of positively labeled masks from summations of negatively labeled masks. The second approach was to use a Convolutional Neural Network (CNN) classifier, which accepts the whole probability mask as its input. A custom architecture was utilized for this experiment, which consists of four convolutional layers complemented with three max-pooling layers, followed by three fully connected (hidden) layers (see Appendix B). This CNN was implemented using the *PyTorch* framework on the Python programming language.

To test both approaches, I generated a set of *FgSegNet*-generated probability masks for each location; this set of masks was used to train and evaluate each approach. First, *FgSegNet* was used to generate all remaining images (of a location) labeled to have foreground objects. The exact number of remaining positively labeled images varied for each location, ranging from approximately 500 to 900 in count. Next, to better represent the proportion of positive to negative data in the full dataset, twice the respective number of positive images were sampled. No data in this generated set were used to train *FgSegNet*. Then, this set was partitioned into a 80-20 split, with 80% of the data for training the approach, and the remaining to quantitatively evaluate it.

Measuring the System's Accuracy

The accuracy of the whole system in caribou image detection is fully dependent on the accuracy of the tested approaches, which are in turn partially dependent on *FgSegNet*'s performance. It should be noted that *FgSegNet* was not evaluated by itself specifically on its foreground extracting accuracy in this study. To measure the detection accuracy of each approach, I utilized metrics beyond simply finding the ratio of correct predictions to total data examples (raw accuracy).

First, the outputs of the approach on the test set were checked against the actual (ground truth) labels to find the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) counts with a threshold of 0.5 on the output probabilities. These counts were then used to calculate the F_1 score—the harmonic mean of the calculated precision (TP / [TP + FP]) and recall (TP / [TP+FN]) (Sasaki, 2007). F_1 was among one of the preferred metrics for this study because TN count is not a factor in its measurement. The datasets used for experimentation contain a majority of TN examples, which would inflate other accuracy metrics, except F_1. F_1 score ranges from 0 (worst) to 1 (best).

However, F_1 only determines the accuracy of the approach in one threshold (0.5). Therefore, I used the Receiver Operating Characteristic (ROC) curve—a TP rate versus FP rate graph plotted over multiple thresholds—and AUC (Area Under the ROC Curve) score to better generalize the overall accuracy of the approach/system. The *scikit-learn* Python library was used for finding both the ROC curve and the AUC score. AUC score ranges from 0 to 1 (a higher value is better), with 0.5 denoting the outcome due to random predictions.

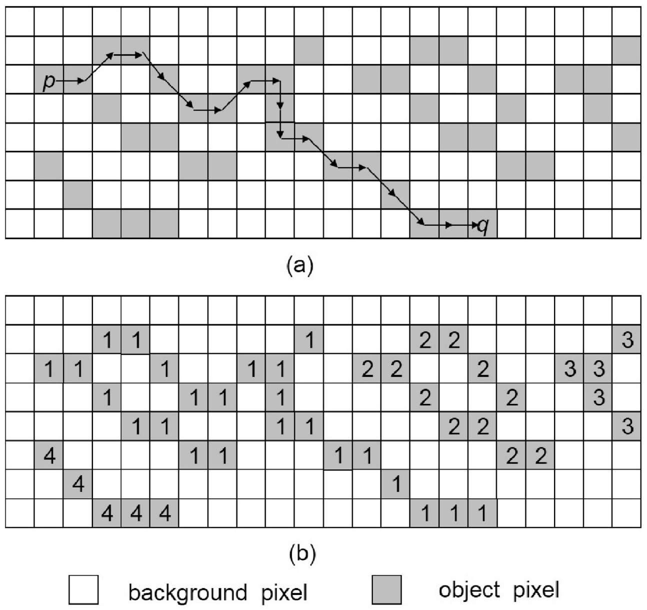
Developing a Caribou Annotation Algorithm

Another main objective of this study was to extrapolate information on caribou count and size based on *FgSegNet*'s outputs. The algorithm to achieve this relies upon connected-component labeling, which fundamentally involves reading through each pixel to determine sets of neighboring nonzero pixels within a binary image (see Figure 5) (He et al., 2017). First, this algorithm performs a threshold of 0.5 on the mask, meaning that all pixel values inheriting a probability greater than or equal to 0.5 were converted to a value of 1, while the rest were reduced to 0. Connected-component labeling was performed on the thresholded mask, yielding all connected components found on the mask. Labeled components that were less than

20 pixels in size (the size of image examples is 1024 x 720 pixels, amounting to 737,280 total pixels per image) were neglected to reduce the amount of false positives. Then, a bounding box is created to enclose all remaining labeled components. This feature could potentially be used to estimate the caribou's width and height.

Figure 5

Depiction of Connected-Component Labeling



Note. As the procedure traces through every pixel on the image, it determines all neighboring pixels that are foreground (or nonzero object pixels as denoted on the figure). It then assigns the pixel an existing label from its foreground neighbors, or a new label if none of its neighbors are foreground (He et al., 2017).

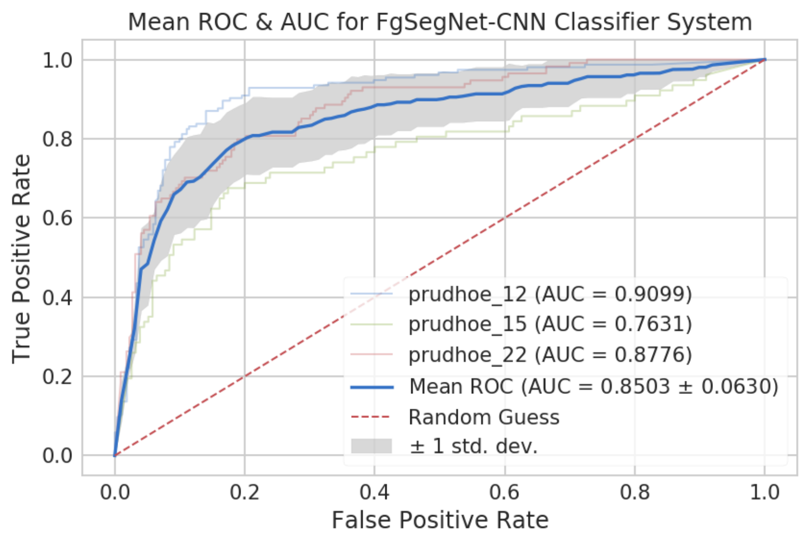
This algorithm would be utilized on masks that were classified to contain a caribou or foreground object from the previous algorithm, and operates on one mask at a time. The algorithm was implemented via a Python script and depended on the *OpenCV* library, which provides a large selection of functions for computer vision-related tasks. Specifically, the algorithm contained the *OpenCV* methods `cv2.connectedComponents()` and `cv2.boundingRect()` to perform connected-component labeling and create bounding boxes of resulting labeled components, respectively.

Results

Two approaches to predict the presence of caribou using probability masks generated by *FgSegNet* had been evaluated: a pixel thresholding algorithm, and a CNN classifier. Attempting the simple threshold algorithm on the training set yielded a raw accuracy score (the number of correct predictions divided by the number of data examples) of 0.6658 and 0.6572 in locations `prudhoe_12` and `prudhoe_15`, respectively. On the other hand, the CNN classifier averaged an AUC accuracy metric of 0.8503 and a mean F_1 score of 0.7151 across all locations(see Figure6). The highest accuracy metrics achieved by the CNN classifier are 0.9099 AUC and 0.8143 F_1 score in `prudhoe_12` (see Figure 7). On the contrary, the lowest metrics achieved are 0.7631 AUC and 0.6107 F_1 in `prudhoe_15` (see Figure 8). It took approximately 1 minute to train 1 epoch for the CNN classifier with a batch size of 4, and less than 15 seconds for the model to evaluate on a set of 300 images (on average, as the amount of images varied for each camera trap location tested).

Figure 6

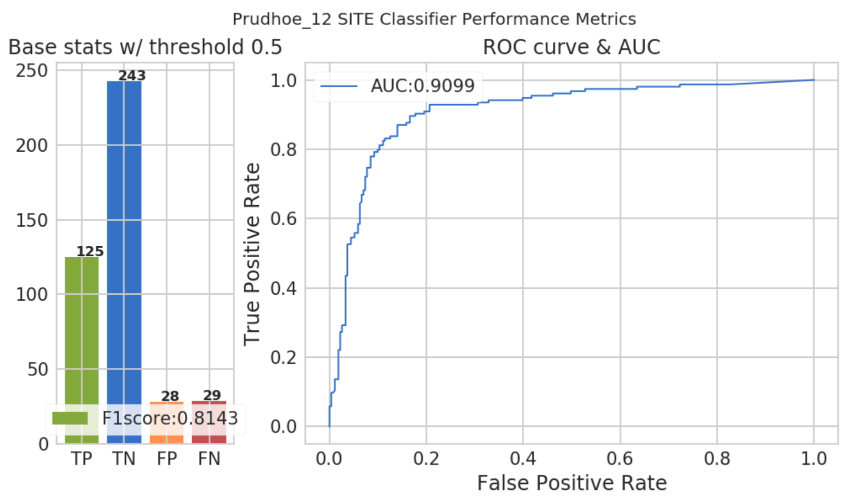
ROC Curves and AUC Scores for FgSegNet-CNN System



Note. The system achieved a mean AUC of 0.8503 with a standard deviation of 0.0630 across all three tested locations.

Figure 7

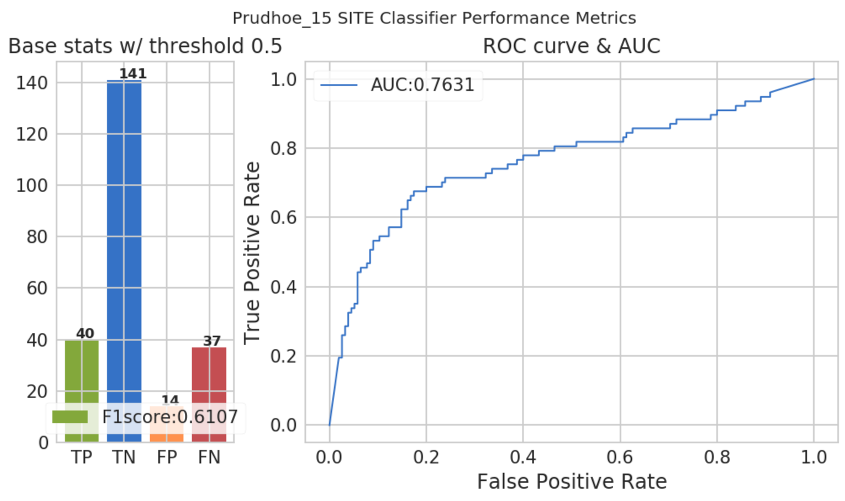
Performance Metrics of FgSegNet -CNN System in prudhoe_12



Note. The left bar graph shows the true positive(TP), true negative(TN), false positive(FP), and false negative(FN) counts, along with the F_1 score. The right graph shows the ROC curve and the AUC score. The system performed the best at prudhoe_12, achieving a 0.9099 AUC and a 0.8143 F_1 score.

Figure 8

Performance Metrics of FgSegNet -CNN System in prudhoe_15

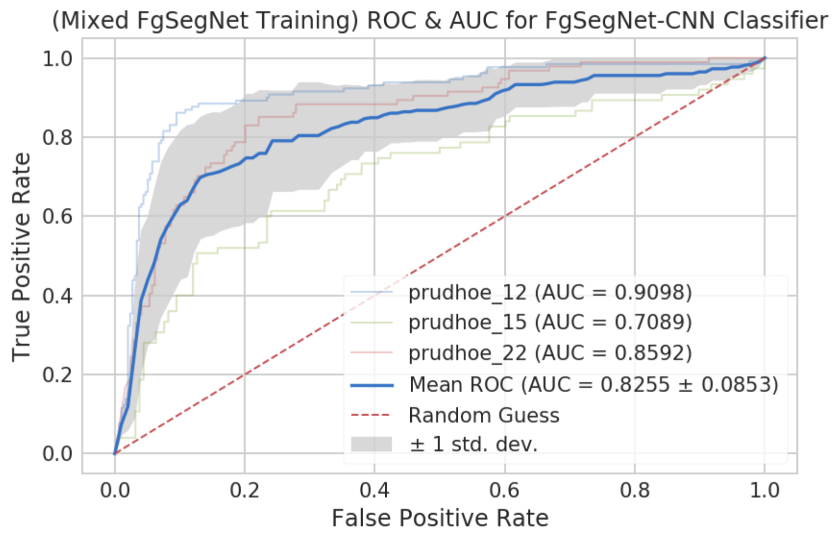


Note. The layout of this figure is retained from Figure 7. The system performed the worst at this location, achieving a 0.7631 AUC and a 0.6107 F_1 score.

This study also attempted a variation of the *FgSegNet* component: training the model on a combination of positively and negatively labeled images within the dataset. The modified system scored an average AUC of 0.8255 and F_1 score of 0.6406, which denotes lower accuracy compared to the system without *FgSegNet* training modifications (see Figure 9). Based on these results, the hypothesis in which a larger training set would aid in the accuracy of the system was rejected.

Figure 9

ROC Curves and AUC Scores for System with Modified FgSegNet Training Data

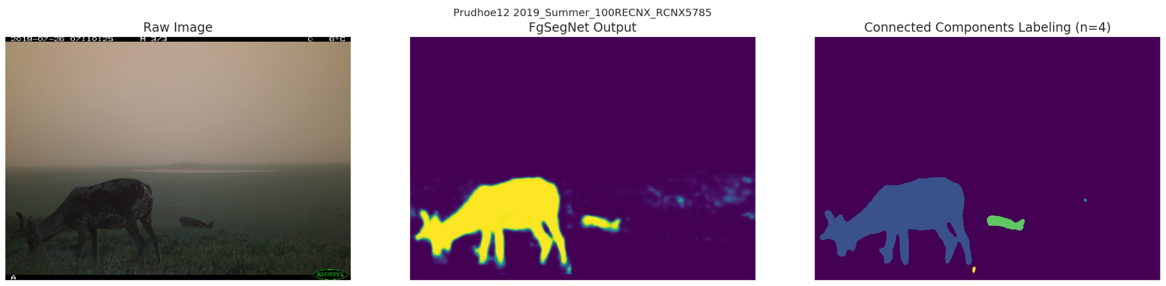


Note. The modified system achieved a mean AUC of 0.8255 with a standard deviation of 0.0853, concluding that it performed worse than the unmodified variant.

Multiple images in the dataset that were both labeled to have caribou by the system (*FgSegNet* and CNN classifier combination) and were not used to train itself were qualitatively evaluated. Figure 10 shows the results of the system's annotation algorithm in an example image, along with the *FgSegNet* mask that was used as a reference to establish the connected components. This algorithm includes instructions to draw bounding boxes on each connected component, which allows the calculation of the pixel dimensions of each (see Figure 11).

Figure 10

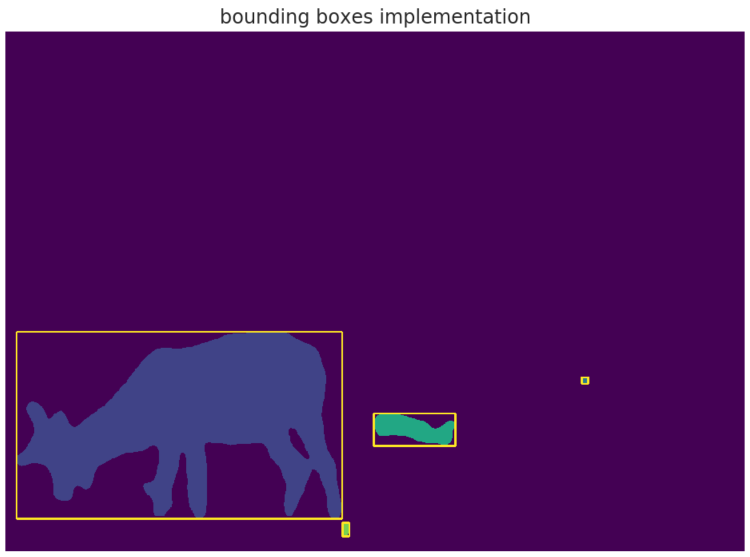
Representation of the System's Annotation Process


Prudhoe12 2019_Summer_100RECNX_RCNX5785
Raw Image
FgSegNet Output
Connected Components Labeling (n=4)

Note. The leftmost image is the raw camera trap image example, the center image is the *FgSegNet*-produced foreground probability mask, and the rightmost image is the result of the implemented connected-component labeling algorithm.

Figure 11

Demo of the Bounding Boxes Implementation

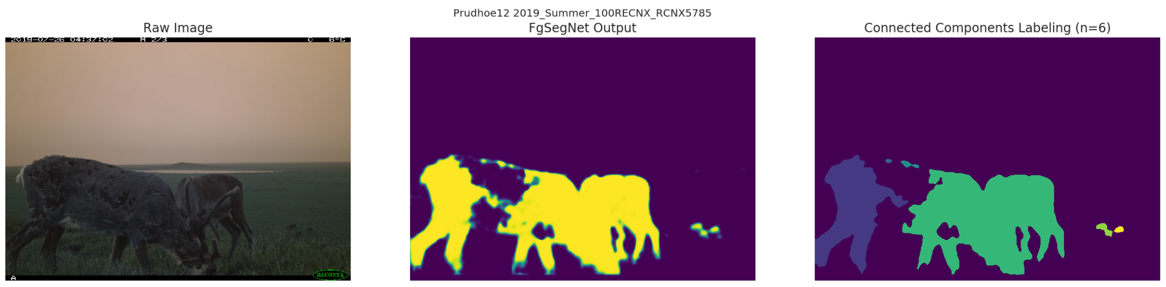


Note. The reference image is the same example as Figure 10. These boxes can be used to calculate the pixel width and height of each caribou, and assist in counting the amount of caribou in the data example.

However, the system's annotation algorithm did not perform as desired in other cases. For instance, there are multiple instances in which parts of the same caribou were labeled as multiple, different entities. In another edge case, two caribou were determined as one object due to how both were oriented relative to the camera's view. Figure 12 shows an instance where both edge cases were present and the algorithm performed incorrectly.

Figure 12

Example of Incorrect Labeling of the System's Annotation Algorithm


Prudhoe12 2019_Summer_100RECNX_RCNX5785
Raw Image
FgSegNet Output
Connected Components Labeling (n=6)

Note. In this example, one caribou (left) was separately labeled as three different entities. The right label of the actual caribou was merged with another caribou behind it. Despite these shortcomings, three caribou from the distance were still correctly labeled.

Discussion / Conclusion

The system ultimately has performed respectably well in the task of detecting caribou from raw camera trap image data. The accuracy metrics of our system are lower than what *FgSegNet* had achieved in the CDnet2014 dataset, however, this is reasonable given the multiple limitations which this experiment encountered. For instance, there may have been inadequate training data for the CNN classifier to train on, preventing it from properly generalizing the given data. On the other hand, the CNN may have encountered an instance of overfitting—when the model begins memorizing training data instead of learning data trends—when training. These issues may be fixed by altering the training process for the *FgSegNet*-CNN system, or changing the architecture of the CNN classifier. Limitations also do exist within the dataset: false negatives (e.g. images that do contain foreground objects were erroneously labeled to have none) might have existed within the labeled images used for training and evaluating both *FgSegNet* and the CNN classifier, thus varying the system performance across the locations (and likely explaining the dramatic performance drop on location `prudhoe_15`. Additionally, I was only able to qualitatively determine the annotation algorithm's performance because the dataset lacked entries denoting the amount of caribou present in each image.

Despite these drawbacks, the system had made significant progress in automating image labeling for wildlife investigations in the Alaskan wilderness. It is impressive for the system to achieve up to 0.901 AUC accuracy in caribou detection only with a total of approximately 2500 images in the training set (this includes data needed to train *FgSegNet*, and the CNN classifier). The system has also shown its practicality for real-world scenarios through its fast training and evaluation speeds, meaning that this system would not inconvenience annotators and wildlife researchers who choose to utilize it in the near future. Through experimenting with the system, I have additionally discovered the possible necessity to complement an established BGS algorithm (*FgSegNet*) with a CNN classifier to achieve reliable foreground presence prediction.

Looking into the future, improvements to the system at this current state are unlikely to exist with *FgSegNet*, but instead with the CNN classifier and the annotation algorithm. Certain specifications of the CNN's architecture, including the arrangement of convolutional layers and the kernel size, along with general training parameters (e.g. learning rate, optimizer function, number of epochs, etc.) could be fine-tuned for faster or more accurate performances. Moreover, a more detailed dataset containing caribou counts for each raw image example would have allowed for a more comprehensive evaluation of the system's annotation feature. Nonetheless, this system and the process of constructing it would serve as a solid baseline for future advances in automating image labeling and annotation, which will significantly assist efforts in understanding remote wildlife activity, most notably caribou in the Alaskan wilderness.

References

Ahire, J. B. (2018). *The artificial neural networks handbook: Part 1*. https://medium.com/coinmonks/the-artificial-neural-networks-handbook-part-1-f9ceb0e376b4

Bouwmans, T., Javed, S., Sultana, M., & Jung, S. K. (2019). Deep neural network concepts for background subtraction: A systematic review and comparative evaluation. *Neural Networks*, *117*, 8–66.

Buxton, R. T., Lendrum, P. E., Crooks, K. R., & Wittemyer, G. (2018). Pairing camera traps and acoustic recorders to monitor the ecological impact of human disturbance. *Global Ecology and Conservation*, *16*, e00493.

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Görner, M. (2017). Tensorflow, keras, and deep learning, without a phd. https://cloud.google.com/blog/products/gcp/learn-tensorflow-and-deep-learning-without-a-phd

He, L., Ren, X., Gao, Q., Zhao, X., Yao, B., & Chao, Y. (2017). The connected-component labeling problem: A review of state-of-the-art algorithms. *Pattern Recognition*, *70*, 25–43. https://doi.org/https://doi.org/10.1016/j.patcog.2017.04.018

Lim, L. A., & Keles, H. Y. (2018). Foreground segmentation using convolutional neural networks for multiscale feature encoding. *Pattern Recognition Letters*, *112*, 256–262. https://doi.org/https://doi.org/10.1016/j.patrec.2018.08.002

Mayes, J. (2017). Machine learning 101. https://goo.gl/5Wd2vy

Ng, A., Ngiam, J., Foo, C. Y., Mai, Y., Suen, C., Coates, A., Maas, A., Hannun, A., Huval, B., Wang, T., & Tandon, S. (2013). *Convolutional neural network*. http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork

Sasaki, Y. (2007). The truth of the f–measure. *Manchester: School of Computer Science, University of Manchester*.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, *61*, 85–117.

Wada, K. (2016). labelme: Image Polygonal Annotation with Python.

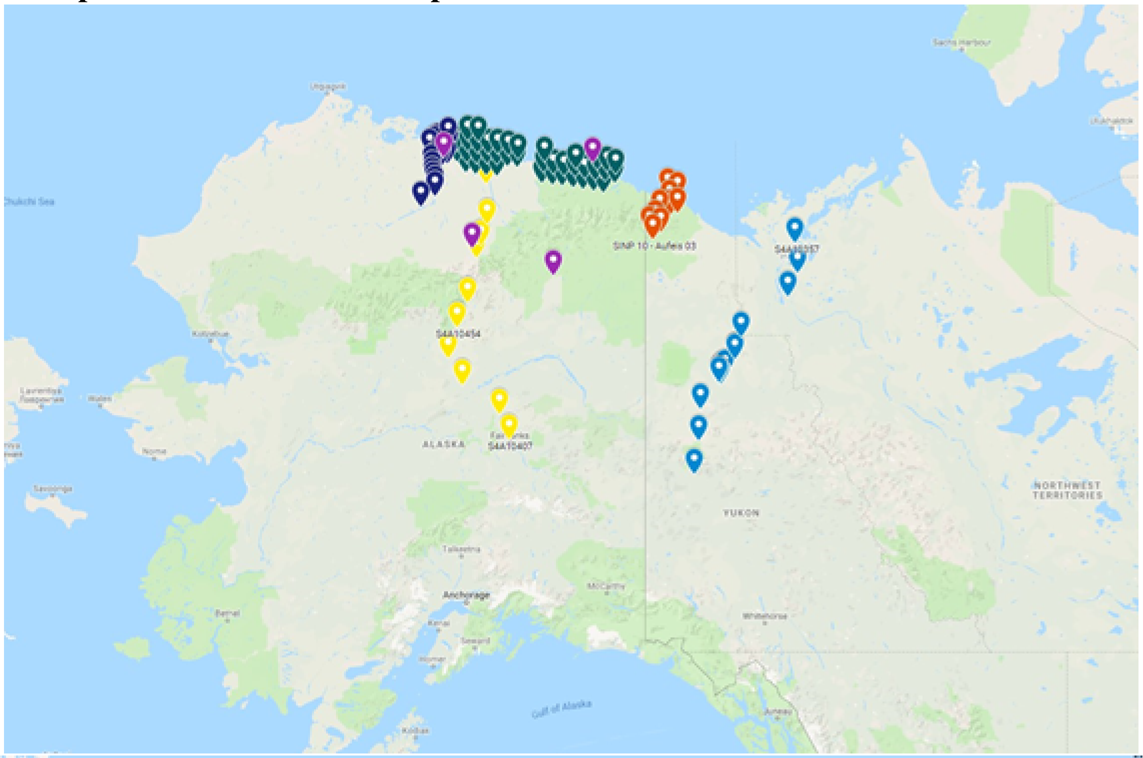
Wang, Y., Jodoin, P.-M., Porikli, F., Konrad, J., Benezeth, Y., & Ishwar, P. (2014). Cdnet 2014: An expanded change detection benchmark dataset. *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 387–394.

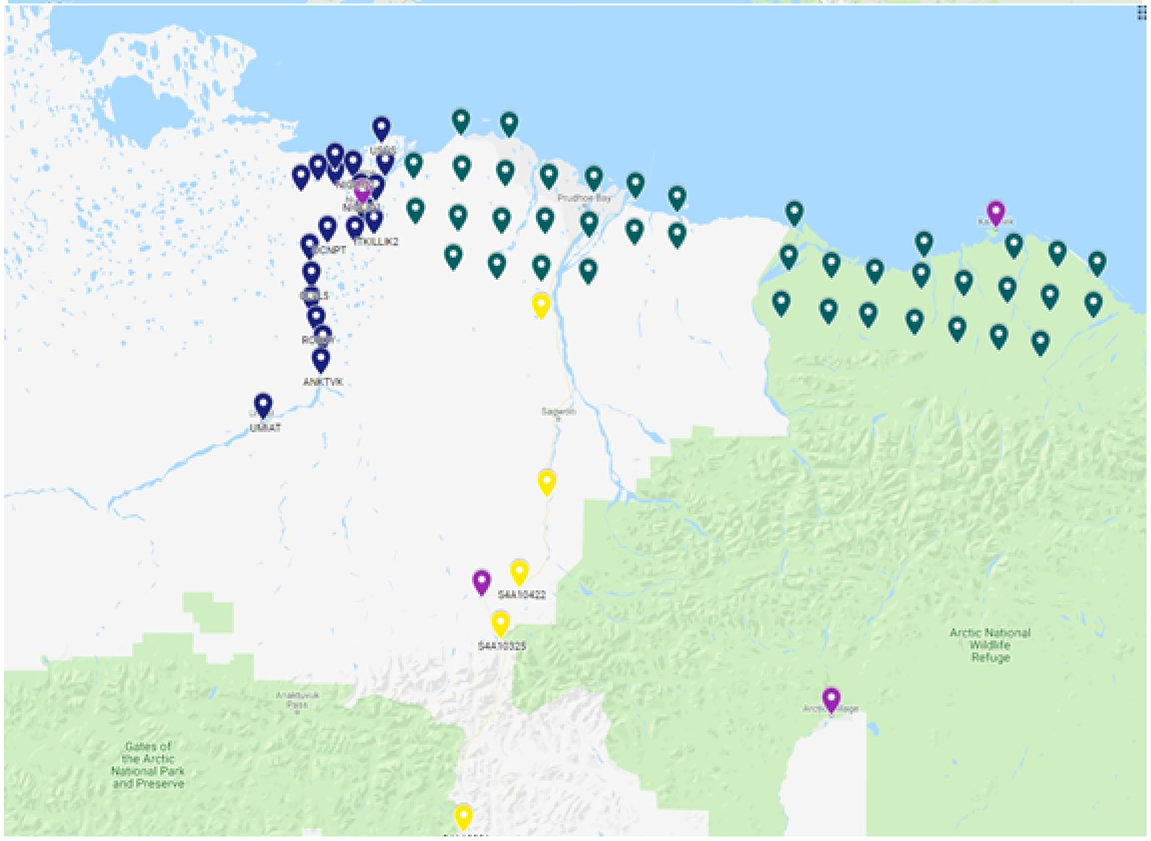
Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*, 23.

Zeng, D., Zhu, M., & Kuijper, A. (2019). Combining background subtraction algorithms with convolutional neural network. *Journal of Electronic Imaging*, *28*(1), 013011.

Appendix A

Maps of Various Camera Trap Locations Established in the Alaskan wilderness




ANKTVIK
UMIAT
Prudhoe Bay
S4A10422
S4A10325
Arctic National Wildlife Refuge
Gates of the Arctic National Park and Preserve
Arctic Village

Appendix B

Code Definition of CNN Classifier in Python using *PyTorch* [1]

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
...
def accuracy(outs, labels):
    # check raw accuracy for threshold @ 0.7
    outs_th = outs >= 0.7
    # return ratio of outputs >= 0.7 to total evaluated outputs
    return torch.tensor(torch.sum(outs_th == labels).item() / len(outs))

class ModelBase(nn.Module):
#       training step
    def train_step(self, batch):
        xb, labels = batch
        labels = labels.view(-1, 1)
        outs = self(xb)
        loss = F.binary_cross_entropy(outs, labels)
        return loss

#       validation step
    def val_step(self, batch):
        xb, labels = batch
        labels = labels.view(-1, 1)
        outs = self(xb)
        loss = F.binary_cross_entropy(outs, labels)
        acc = accuracy(outs, labels)
        return {'loss': loss.detach(), 'acc': acc}
```

[1] The full code script can be found at https://github.com/richardso21/SERP2021-BGSUB.

```
#      validation epoch (avg accuracies and losses)
    def val_epoch_end(self, outputs):
        batch_loss = [x['loss'] for x in outputs]
        avg_loss = torch.stack(batch_loss).mean()
        batch_acc = [x['acc'] for x in outputs]
        avg_acc = torch.stack(batch_acc).mean()
        return {'avg_loss': avg_loss, 'avg_acc': avg_acc}
...
class Custom_NPT(ModelBase):
    # custom-defined model w/o pretrained weights
    # no use of batch norm
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 8,kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(8, 16, kernel_size=3, padding=1)
        self.pool1 = nn.MaxPool2d(2)
        self.conv3 = nn.Conv2d(16, 16, kernel_size=3, padding=1)
        self.pool2 = nn.MaxPool2d(2)
        self.conv5 = nn.Conv2d(16, 8, kernel_size=3, padding=1)
        self.pool3 = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(8*128*90, 4096)
        self.fc2 = nn.Linear(4096, 512)
        self.fc3 = nn.Linear(512, 1)

    def forward(self, xb):
        out = F.relu(self.conv1(xb))
        out = F.relu(self.conv2(out))
        out = self.pool1(out)
        out = F.relu(self.conv3(out))
        out = self.pool2(out)
        out = F.relu(self.conv5(out))
        out = self.pool3(out)
        out = torch.flatten(out, 1)
        out = F.relu(self.fc1(out))
        out = F.relu(self.fc2(out))
        out = self.fc3(out)
        out = F.sigmoid(out)
        return out

model = Custom_NPT() # initialize the defined model
...
```