# White Christmas: Remaking Augmented Reality Censorship from Black Mirror with Identity-Preserving Instance Segmentation

Michael Lee
Stanford University
mike21@stanford.edu

Mario Baxter
Stanford University
mariobax@stanford.edu

## Abstract

*The concept of this project is augmented reality visual censorship, as presented in the episodes White Christmas and Arkangel in Netflix's popular series, Black Mirror, recreating this science-fiction experience using modern machine learning techniques. We approached this problem by merging two popular computer vision applications, object tracking and instance segmentation, into one cohesive model by adding a masking branch to our object tracker. We trained the object tracking part of our project on the MOT 2016 dataset and trained the instance segmentation part of our project on the COCO dataset. We achieved 78.2% MOTP (a measure of precision) and 46.3% MOTA (a measure of accuracy) on the MOT dataset, and 81.1% accuracy on humans in MS-COCO with a fully-convolutional network, with 0.834 recall. In achieving this, we have created an alternative architecture for video object tracking plus segmentation that has good performance and is much faster.*

## 1. Introduction

Our social and political climate is becoming increasingly concerned with the impact that technology will have every aspect of our lives. The concept of this project is augmented reality visual censorship, as presented in the episodes White Christmas and Arkangel in Netflix's popular series, Black Mirror. In these episodes, a futuristic world is presented where people are able to visually censor anyone they choose by applying a blur over their physical shape. Our project's goal was to recreate this science-fiction experience using modern machine learning techniques.

### 1.1. Object Tracking

Object Tracking is an important computer vision problem that has a variety of applications in autonomous systems, animation, and more. The inputs to our object tracker are videos. These videos are then fed into an existing, well performing object tracker, Deepsort [4], which outputs a set of bounding boxes for each frame in the input video encapsulating detected human instances.

### 1.2. Instance Segmentation

Image segmentation has been an important computer vision milestone for many years, with applications in visual object detection, medical imaging, and more. For our project it is important to distinguish two types of image segmentation, semantic and instance. As can be seen in Figure 1, semantic segmentation classifies each region into classes, but does not differentiate between instances of the classes. Instance segmentation, which is the focus of this paper, is significantly more difficult, as it must differentiate between instances of the class. This is essential to our project as we need to be able to individually blur people and not blur every person in a video frame.

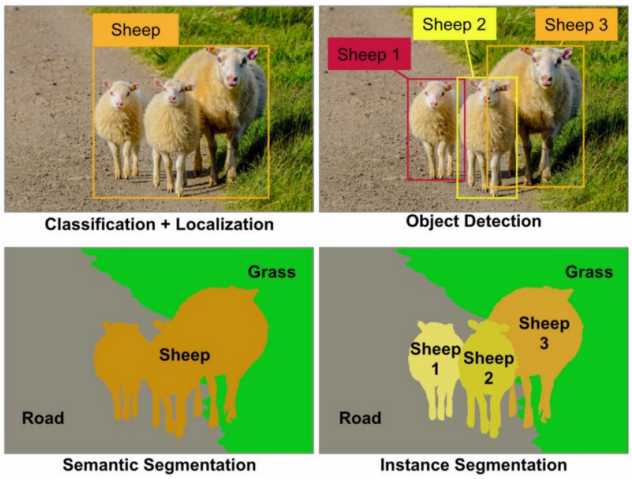


Figure 1. A comparison classification vs. detection vs. semantic segmentation vs. instance segmentation. We go one step further, combining instance segmentation plus object tracking

The inputs to our instance segmentation algorithm are images corresponding to bounding boxes outputted by our object tracker. These images are then fed into a neural

network architecture that outputs a corresponding mask for each input image.

### 1.3. Branching

Our goal of creating one, seamless model that performs both object tracking and instance segmentation of humans would not be possible without branching. FINISH!!!!!

## 2. Related Work

### 2.1. Previous Object Tracking Models

When choosing an object tracking model, we decided to look at the best performing online models on the MOT dataset. The best performing was Person of Interest (POI) [6], however, the slight increase in performance that made it outpeform its rivals made it too slow for our application, ultimately running at 10Hz. We thus decided to use the second best performing online model, DeepSort [4], which did not suffer much in performance when compared to POI partly thanks to its use of the state-of-the-art object detector, Faster-RCNN. This model was able to run at 40Hz.

### 2.2. Previous Instance Segmentation Models

Our final version of our instance segmenter was influenced by the work done by J. Long et al. They were the first to develop a fully convolutional end-to-end image segmenter (FCN) [2]. FCN takes an image with an arbitrary size and produces a segmented image with the same size. Initially, the authors modify well-known architectures such as AlexNet, VGG16 and GoogLeNet to have a non fixed input size as well as replacing all the fully connected layers by convolutional ones. The main challenge they addressed was deconvolution, or the act of creating an output with a larger size than the input. This allows the network to be trained on a pixel-wise loss.

### 2.3. Previous Object Tracking and Segmentation Models

The only reasonably-well performing, all-in-one, object tracker and segmenter that we have found is MaskTrack R-CNN [5]. This algorithm essentially adds a tracking branch to the Mask R-CNN architecture. However, this method is very slow and computationally expensive as it is tied to a slow object detection architecture. In this paper we flip this architecture. Instead, we start with a multiple object tracking base (DeepSort) and add a masking branch to generate masks on the outputted bounding boxes.

## 3. Data

We are using two datasets for our project, MS-COCO [1] and the Multiple Object Tracking (MOT) dataset [3], which includes only human examples.

As a preprocessing step, we first extracted all the human-only examples from the very large MS-COCO dataset. We padded these human-only examples for the purposes of testing our first baseline model (SVM). However, for our final model, these paddings were not necessary. We then partitioned the MS-COCO dataset into train (64115 examples), validation (2693 examples) and test (1k examples) sets.

We used the MOT dataset to train an existing, well-performing, multiple object tracker (DeepSort) as well as the underlying object detector (Faster R-CNN). We split this dataset into two: a training set consisting of 7 videos and a validation/test set containing another 7 videos. The data necessary for training the tracker is not enormous as it already comes with a pre-trained object detector, Faster R-CNN.

### 3.1. Dataset Examples

In this subsection, we provide image examples of samples from the MS-COCO and MOT dataset, providing a visualization of both the input and the ground truth layered over the input.

#### 3.1.1 MS-COCO

Figure 2. COCO Input Example

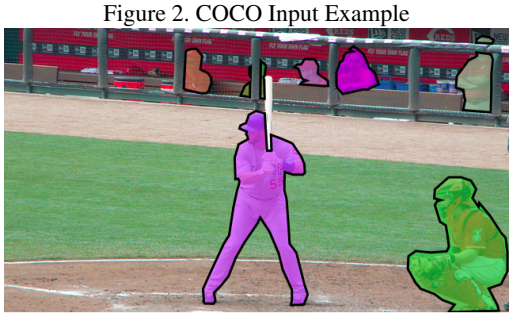

Figure 3. COCO Ground Truth Example

#### 3.1.2 MOTS

Input and GT EX:

Figure 4. MOT Input Example

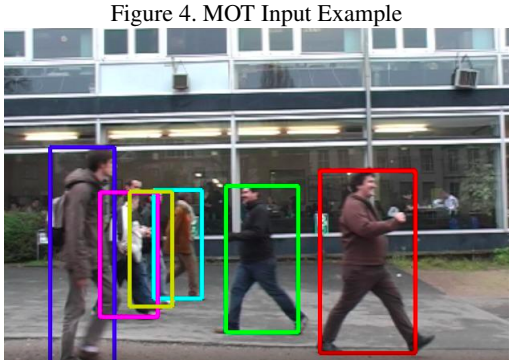

Figure 5. MOT Ground Truth Example

# 4. Methods

## 4.1. Baseline 1 (SVM)

Our original, most basic implementation of the instance segmenter was using a Support Vector Machine with a Gaussian Kernel. An SVM is a discriminative classifier formally defined by a separating hyperplane. In other words, an SVM outputs an optimal hyperplane which categorizes new examples. In order to compute non-linear hypotheses, a kernel is necessary. A kernel is anything that outputs the same value as a dot product between two feature vectors, without any constraints on the dimension of these feature vectors. For this baseline, we used the Gaussian kernel, which is expressed as:

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-z)^2/2\sigma^2}$$

Where x and z are 2 examples and $\sigma$ is a hyperparameter. We implemented masking with our SVM by outputting per-pixel values between 0 and 1. These outputs determined our model's confidence that the pixel at hand was part of a human mask.

Initially, our inputs consisted of a single pixel (R, G, B). However, we quickly realised that this was a bad way of implementing a model as it did not take any spacial considerations into account. In other words, we were expecting our model to outuput wether or not a given pixel was part of a mask without taking into account what the neighboring pixels looked like. In order to solve this, we created a radius of neighboring pixels and fed all of them into our SVM. This method performed substantially better.

## 4.2. Baseline 2 (NN)

After training our SVM, we soon realised that the complexity of our model would have to increase drastically if we were going to get relatively good accuracy. We thus decided to transition to using a Neural Network. A neural network is a combination of nodes that perform 2 functions: a linear one involving the parameters of the model which are learnt through training as well as a non-linear function that guarantees that the complexity of the model grows with the number of layers.

The architecture that we chose included input and output layers of size 12,288 (64x64x3), as well as 10 hidden layers, each with 500 neurons each. As our loss function we used binary cross entropy with logits and used binary accuracy as our metric. We also used the Adam optimization algorithm and the ReLU activation function with a sigmoid activation function at the end. The network was trained for 30 epochs.

## 4.3. DeepSort

DeepSort applies appearance information to Simple Online and Realtime Tracking (SORT) to improve its performance. SORT is a pragmatic approach to multiple object tracking with a focus on simple, effective algorithms. Ultimately, this allows DeepSort to track objects through longer periods of occlusion, dramatically increasing the preservation of identities.

To incorporate motion information, DeepSort uses two distance metrics. The first is the (squared) Mahalanobis distance between predicted Kalman states and newly arrived measurements:

$$d^{(1)}(i,j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i)$$

Where the projection of the ith track distribution into measurement space is $(y_i, S_i)$ and the jth bounding box detection is $d_j$. The second metric is the smallest cosine distance between the ith track and jth detection in appearance space:

$$d^{(2)}(i,j) = min\{1 - r_j^T r_k^{(i)} | r_k^{(i)} \in R_i\}$$

Where $R_k = \{r_k^{(i)}\}_{k=1}^{L_k}$, $L_k = 100$ and $r_j$ is the appearance descriptor. Finally, both metrics are put together using a weighted sum:

$$c_{i,j} = \lambda d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j)$$

## 4.4. Fully Convolutional Networks (FCN)

Convolutional Neural Networks are comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully

connected layers as in a standard multilayer neural network. A convolutional layer is one which has kernels, or filters, instead of traditionally fully-connected layers.

Fully Convolutional Networks have been shown outperform the state-of-the-art in semantic segmentation. The key insight is to build fully convolutional networks, without any fully connected layers, that take input of arbitrary size and produce correspondingly-sized output with efficient inference and learning. This is achieved partly thanks to the definition of a skip architecture which combines semantic information from a deep, coarse layer with appearance information from a shallow, fine layer to produce accurate and detailed segmentations.

Our FCN was comprised of 53 convolutional layers as well as a bilinear upsampling layer. Our hyperparameters included a learning rate of 1e-3 and a weight decay of 5e-5. We used the Adam optimization algorithm and the ReLU activation function on every node to speed up the learning as well as Batch Norm as a regulariser.

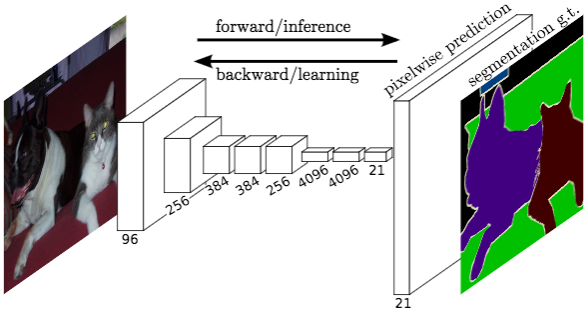


Figure 6. FCN Architecture

### 4.5. Branching Overview

In order to build a full pipeline for emulating the White Christmas censorship, we needed to output the boxes from the multiple object tracker to masking 'branches.' These branches would be trained separately, and then multiple algorithms tested for each one, combining into a pipeline that would add masking functionality onto the multiple object tracker.

## 5. Results

### 5.1. Evaluation Metrics

We evaluated the performance of our models by 3 metrics: accuracy, precision and recall. More specifically, we used the following formulas:

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$Precision = \frac{TP}{(TP+FP)}$$

$$Recall = \frac{TP}{(TP+FN)}$$

Where TP is the number of true positives, TN the number of true negatives, FP the number of false positives and FN the number of false negatives.

### 5.2. Table

| Model | Accuracy | Precision | Recall | Speed |
|---|---|---|---|---|
| SVM 1 pixel FOV | 54.17% | - | - | - |
| SVM 5 pixel FOV | 56.05% (3 image overfit, 89.3%) | - | - | - |
| SVM 15 pixel FOV | 51.20% | - | - | - |
| 10-Layer Neural Network | 61.38% | 0.6012 | 0.6315 | 106 FPS |
| FCC Network w/ DenseNet | 81.10% | 0.7008 | 0.8341 | 35 FPS |

Figure 7. Table of performance, including binary accuracy, precision, recall, and speed. Some values omitted due to SVM infeasibility.

### 5.3. Baseline 1 (SVM)

Due to the slow training of our SVM and since this was not going to be our final model, we decided to cut short the number of examples in order to see wether or not our dataset had something to be learnt. We found that after training on only 1000 examples, and not accounting for overfitting, our SVM implementation achieved an accuracy of 54.17% when using a pixel neighbor redius of 1, 56.05% with a radius of 5 and 51.20% with a radius of 15.

Below is an image of an overfitted example by our SVM where we achieved an accuracy of 89.3%:

Input, Ground truth and Prediction

Figure 8. 3-image overfitted example on 5-pixel FOV SVM with 89% accuracy. Pixel-by-pixel evaluation.

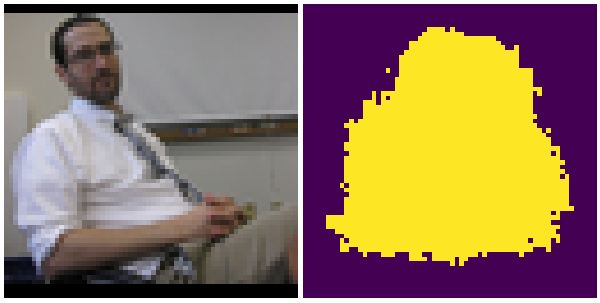

Figure 9. 10 Layer NN. Left two images - average example of input/output pair for wide image. Right two – average input/output for narrow. Demonstrates 'blobbiness' of prediction.

### 5.4. Baseline 2 (NN)

We were able to train our Neural Network on the entire COCO dataset and achieved an accuracy of 61.38%, a precision of 0.6012 and a recall of 0.6315. Fully-connected neural networks are not known for performing well on image data and thus some outputs looked like the network was simply spitting out noise corresponding to where the person would most likely be within a given bounding box. Upon noticing this, we decided to transition to using networks that included convolutional layers.

### 5.5. Fully Convolutional Network

We trained our FCN on the entire COCO dataset and achieved an accuracy of 81.14%, a precision of 0.7008 and a recall of 0.8341. This was a significant increase in accuracy when compared to its non-convolutional peer. Aside from the numerical increase, there was also a substantial increase in the subjective visual performance of the model. Instead of outputting noise corresponding to where the person would most likely be within a given bounding box, our FCN gave intelligent predictions with masks that resembled those of real humans. Below are two examples from the same video and person five frames apart:

Figure 10. FCN input/mask pairs of man, from sequences 5 frames apart

### 5.6. DeepSort

We trained our object tracker, DeepSort, on the human-only MOT dataset. After training this model, we achieved the same metrics on unseen examples as reported on the MOT website: 78.2% MOTP (a measure of precision)and 46.3% MOTA (a measure of accuracy). The most common errors that we have found after training DeepSort occur when there is occlusion, bad lighting and when humans are close together. Below is a prediction on two frames from the same video from our object tracker:

Figure 11. Output of DeepSort on a random frame

## 6. Conclusion and Future Work

In this paper, we surveyed some traditional techniques for masking, including SVM and deep neural networks. However, we found that these techniques outputted weak results. The neural network, however, did clearly learn general location of the person in the image. Overall, we were impressed by the quality of the results obtained by branching a version of FCN off of DeepSort. We additionally saw that failure cases of the FCN were mostly robust to our application, which can be seen in a high recall value of 0.834. Additionally, as can be seen in our table of results, we observed high speed in all models. As a full pipeline, we have succeeded in creating an alternative, highly efficient architecture for processing video object tracking plus segmentation, especially if precision is unneeded. Our most pressing next steps would be to try and get our algorithm to work in real time, obtaining video input from a camera and rendering at object detection and instance segmentation at over 20Hz.

## References


[1] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft COCO: Common Objects in Context. *arXiv e-prints*, page arXiv:1405.0312, May 2014.

[2] J. Long, E. Shelhamer, and T. Darrell. Fully Convolutional Networks for Semantic Segmentation. *arXiv e-prints*, page arXiv:1411.4038, Nov 2014.

[3] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler. MOT16: A Benchmark for Multi-Object Tracking. *arXiv e-prints*, page arXiv:1603.00831, Mar 2016.

[4] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[5] L. Yang, Y. Fan, and N. Xu. Video Instance Segmentation. *arXiv e-prints*, page arXiv:1905.04804, May 2019.

[6] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan. POI: Multiple Object Tracking with High Performance Detection and Appearance Feature. *arXiv e-prints*, page arXiv:1610.06136, Oct 2016.