5-2023

# Chicken Keypoint Estimation

Rohit Kala

Chicken Keypoint Estimation

# Chicken Keypoint Estimation

A thesis submitted in partial fulfillment
of the requirements for the degree of
Bachelors of Science in Computer Science with Honors

By

Rohit Kala
University of Arkansas
Bachelor of Science in Computer Science, 2023

May 2023
University of Arkansas

**Abstract**

Poultry is an important food source across the world. To facilitate the growth of the global population, we must also improve methods to oversee poultry with new and emerging technologies to improve the efficiency of poultry farms as well as the welfare of the birds. The technology we explore is Deep Learning methods and Computer Vision to help automate chicken monitoring using technologies such as Mask R-CNN to detect the posture of the chicken from an RGB camera. We use Meta Research's Detectron 2 to implement the Mask R-CNN model to train on our dataset created on videos of chickens in a controlled environment. We include the numeric results from different training sessions of varying datasets to showcase the improvement in the model over time. Our findings show that Deep Learning and Computer Vision technologies can effectively enhance poultry farming, and we believe that our study can serve as a foundation for future research in this field.

**THESIS DUPLICATION RELEASE**

I hereby authorize the University of Arkansas Libraries to duplicate this thesis when needed for research and/or scholarship.

**Agreed** Rohit Kala

Rohit Kala

**Refused** _____

Rohit Kala

ACKNOWLEDGEMENTS

I would like to thank Dr. Le, Minh Tran, and Chiyou Vang for their invaluable support and contributions to this project. Their dedication to this paper has been instrumental in the success of this research. I would also like to thank the committee members, Dr. John Gauch and Dr. Lu Zhang, for their invaluable feedback to improve this paper. I am also grateful to the University of Arkansas for providing us with the opportunity as well as the infrastructure to facilitate this research paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

# 1 Introduction

Poultry is an important and cheap source of protein and nutrition in the world. Therefore, it is no surprise that it is a major sector in the agricultural industry. In 2020 alone, the US produced $21.7 billion of chicken meat for consumption, which commonly comes from a breed of chicken called broilers. On top of this, chicken sales accounted for another $18.7 million excluding broilers [3]. With the increasing demand for poultry products, it is vital to ensure that the industry operates efficiently. Previously, farms manually monitored the well-being of the chickens in their farms. However, in recent times, manual monitoring of poultry such as chickens will be too time consuming to be efficient. Thus, the recent improvement in machine learning for computer vision tasks can allow us to automate the monitoring of chickens. One specific way to monitor chickens is to detect an instance of a chicken in a video by enclosing the chicken in a bounding box. In this research, we propose a methodology that utilizes machine learning algorithms to estimate the key-points of chickens, given a video from an RGB camera. We also detail the creation and testing of the dataset, which forms the basis of our research. Our study's findings provide insights into the potential benefits of using machine learning to automate poultry monitoring, paving the way for future research in this field.

## 2 Background

We will go over important background information that is relevant to the technologies used in the paper.

### 2.1 Pose Estimation

General pose estimation is a computer vision technique that involves tracking the orientation and position of objects in an image or a video. The complexity of the problem lies in the fact that objects can be occluded or obscured by other objects or by motion within a video. Additionally, another challenge in pose estimation is how an object can appear visually different in various orientations.

A specific but common problem in the domain of pose estimation is human pose estimation. To elaborate, the object we are tracking in the images and videos are humans. Human Pose Estimation has a variety of uses, with one use being action recognition. By inferring the motion of poses through a sequence of images, we can speculate what action a person is undertaking in those frames [4].

One of the best approaches to pose estimation is Deep Learning. Deep Learning methods use Convolutional Neural Networks(CNNs) to learn to detect and track objects in an image or a video. These methods have achieved state of the art results on many pose estimation benchmarks. An important part of pose estimation is detecting key-points on the object to track how the body moves over time, thus, tracking keypoints is also a primary focus of this paper. An example of a Pose Estimation algorithm for humans that utilizes CNNs is AlphaPose.

Initially, we attempted to perform chicken pose estimation using AlphaPose, a model designed to detect human postures in images and videos. We selected AlphaPose for its advanced features, such as pose flow building, which significantly improved results in videos [5]. However, AlphaPose relied on another model called YOLO to detect human instances in a given frame, and outdated documentation


**Figure 2.1**: Alpha Pose Library for Pose Estimation [1]

made it difficult for us to adapt it for the task of chicken pose estimation. Without training a YOLO model to detect chicken instances, AlphaPose could not detect chicken instances.

## 2.2 Chicken's Key-points

The primary problem in our research paper is to formulate a Key-points schema for the chicken. We ultimately concluded that it would be best to use a key-point system for the chicken that contains 9 points. These 9 points in order for an individual chicken are: the beak, the comb, back of the chicken's head, the chest, the back, the start of the chicken's tail, the end of the chicken's tail, foot 1, and foot 2. An example key-point for a chicken is shown below.


**Figure 2.2**: Example of Chicken's Key-points.

## 2.3 Neural Networks

Neural Networks are computational structures that mimic the way the human brain functions to solve complex computational problems. The basic building block of these structures are called perceptrons and closely follow the design of biological neurons. The perceptron takes in an input, then calculates the output using the perceptron's specified weight and threshold value. If the weight, $w$, multiplied by input to the perceptron exceeds the threshold, $T$, then the preceptron is said to activate. The weight determines the importance of that particular perceptron in the overall network. We organize perceptrons into layers, and at the end of each layer, we multiply all the inputs by their respective weights and then pass them through an activation function, determining the output. When we stack these layers together, we are able create a model that is able to solve more and more complex problems. The first layer into the network is commonly referred to as the input layer and the last layer is commonly referred to as the output layer. Every layer in between these two layers are referred to as hidden layers. An example of a visualization of a neural network is shown in Figure 2.3. Neural Networks have allowed us to solve previously difficult problems [6].

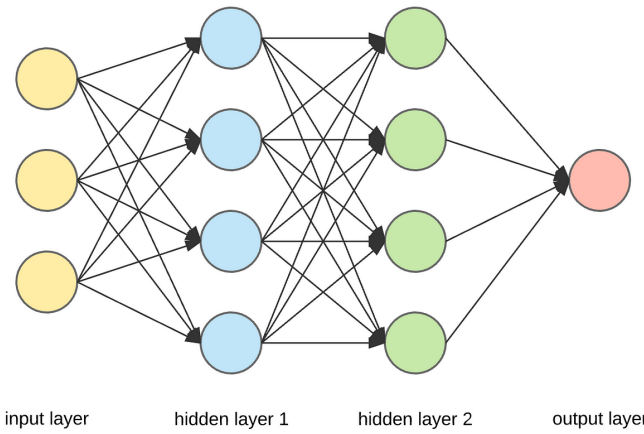


**Figure 2.3**: Visualization of a neural network [2].

## 2.4 CNNs

Convolutional Neural Networks are neural networks that perform better with inputs such as images or audio signals. These networks have three main types of layers, which are: Convolutional layers, Pooling layers and Fully Connected layers.

The convolutional layer is one of the primary building blocks of a CNN, and where a majority of the computation occurs. In this layer, a kernel matrix, $k$, is used to compute a convolution with the input image. The convolution is used to detect features at any particular part of an image. The most common size for $k$ is 3x3, however, other sizes are also applicable depending on the situation.

Pooling Layers downsample the input to reduce the number of parameters. Like the convolutional layer, the pooling layer uses a kernel matrix, $k$, to sweep across the input and apply an aggregation function to send to the output array. The most common types of pooling are average and max pooling. ReLU is a common activation function for both pooling and convolutional layers.

Average pooling layers first split the input into rectangular regions. It then computes the average value for each of these rectangular regions to fill the output array. Max pooling is similar to average pooling in the way that it splits the input into rectangular regions, however, rather than taking the average of the rectangular region, it takes the maximum value to fill the output array.

The final type of layers in the CNNs are fully-connected layers, which are just regular perceptron layers. These layers are used to compute the final output of the model for tasks such as classification. Thus, they usually have softmax functions as the activation function. [7]

### 2.4.1 RCNN

RCNN is a classic CNN pipeline for object detection developed by Ross Girshick et al. The RCNN takes in an input image, then extracts 2000 region proposals using a selective search algorithm. Then, it computes features for each

proposal using a large CNN. Finally, the model classifies each region using class-specific SVMs [8].

### 2.4.2 Fast RCNN

Despite the success of RCNN, it had many drawbacks. One of the drawbacks was that training was a multi-stage pipeline, thus it took a lot of work to train a model up. Moreover, training a model was computationally expensive in space and time as there were various pipelines to train. Finally, object detection in real time was not possible as the object detection was slow [9].

Fast RCNN improves on this pipeline. Initially, an input image is split into regions of interests, which are input to a convolutional network. Each RoI is pooled into a fixed size feature map, which are then mapped to a feature vector per RoI by fully connected layers. The network has two output vectors per each RoI. The first output vector are the softmax probabilities for class scores, whereas the second output vector are the bounding box for the objects.

### 2.4.3 Faster RCNN

The Faster R-CNN framework consists of two main components: the RPN and the Fast R-CNN detector. The RPN generates object proposals by sliding a small network over the convolutional feature maps extracted from the input image. At each sliding window location, the RPN predicts a set of object proposals (bounding boxes) and their objectness scores, which represent the likelihood of the box containing an object. The object proposals generated by the RPN are then fed into the Fast R-CNN detector, which is similar to the object classification component in RCNN. The Fast R-CNN detector uses a CNN to extract features from the proposed regions, and then uses these features to classify the region and refine its bounding box location [10]. One of the reasons why faster R-CNN is faster is due to sharing the same convolutional layers between the RPN and the fast R-CNN detector.

### 2.4.4 Mask-RCNN

Mask RCNNs are built on top of Faster RCNNs by adding another branch that detects and extracts the segmentation mask of the object detected. Thus, we can use Mask-RCNN for the task of object segmentation.

## 3 Approach

We encountered issues with AlphaPose when attempting to detect instances of chickens in our images. Thus, we switched to Detectron 2, a model developed by Meta's FAIR team. Detectron 2 employs the mask R-CNN model to perform image segmentation and key-point detection. One significant advantage of using Detectron 2 is that it uses the Region Proposal Network (RPN) to propose bounding boxes for objects rather than relying on another model. The RPN employs a mechanism called anchors to suggest a bounding box region, followed by an attention mechanism that generates the objectness score, indicating the likelihood of an object's presence within a specific bounding box. The model then passes this information on to downstream tasks such as classification and segmentation. Because Detectron 2 integrates the RPN network into its pipeline, the model can learn to detect chicken instances in our images as we train it on our key-point estimation dataset.

While Mask R-CNNs are primarily designed for mask segmentation, they are highly adaptable to key-point detection tasks. In the pipeline, each key-point $K$ is encoded as a mask, and the model predicts $K$ masks for each image, corresponding to the points annotated in the dataset [11]. The Detectron 2 model further reduces training time by utilizing architectures such as ResNet and FPNs as its backbone.

To optimize our model's performance, we used a learning rate of 0.001 and trained it for 24,000 epochs on our dataset, using Stochastic Gradient Descent as the optimizer to minimize the loss function. We had a batch size of 2 images per GPU, although we only used one RTX 3070ti, effectively resulting in a batch size of 2.

# 4 Experimental Results

### 4.0.1 Data Acquisition

**How to collect raw video data**

Our dataset was based on videos collected at the University of Arkansas Division of Agriculture research house. To capture the movement of a chicken walking along a designated walkway, we set up a regular RGB camera. The camera recorded the chicken's movements as it walked.

### 4.0.2 Data Annotation

The tool we used to annotate the data is a free online annotation tool called Imglab. We exported the annotation data to an XML format file, which we then had to convert to a COCO format JSON file since Detectron 2 works well with it. The data in the XML file included the bounding box data, i.e., the top left X and Y coordinates, the width and the height and finally the area of the bounding box. Additionally, the annotations contained various image details such as image name and picture resolution. Finally, our annotations also contained the X and Y coordinates for each visible key-point. When we convert the XML file to a COCO JSON file, we add another variable for each key-point that is the visibility flag. The value for this variable is 0 if the key-point is not visible, 1 if the key-point is visible but occluded, and 2 if the key-point is fully visible [12].

### 4.0.3 Metrics

We will go over the metrics used to gauge the performance of a model after training.

### Precision

The first metric we need is the Precision score, shown in equation 4.1. Here, TP indicate true positive predictions by the model, and FP represents the false positive predictions by the model. The precision gauges what proportion of positive identifications was actually correct [13].

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

### Recall

The next metric we use is the Recall score, shown in equation 4.2. Here, TP indicate true positive predictions by the model, and FN represents the false positive predictions by the model. The recall measures what proportion of actual positives was identified correctly [13].

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

### Precision-Recall Curve

The precision recall curve is computed by calculating the Precision and Recall at various intersection over union thresholds in the case of bounding boxes. The evaluator computes the precision and recall between the ground truth and model predictions at each threshold. The area under this graph is commonly referred to as the Average Precision(AP) metric, which is what we use to evaluate the performance of the model in this paper for bounding boxes. Key-points evaluation is similar, but rather than using the precision-recall for the IoU thresholds, Detectron 2 uses OKS values as a threshold for precision and recall to calculate the AP values [14].

### 4.0.4 Implementation Details

Our first dataset consisted of annotated images of an annotated toy chicken to test to see if the Detectron 2 model is capable of detecting the chicken. Our tests showed that the model was indeed capable of detecting the chicken. However, this initial annotation contained around 47 key-points (pictured below), which would mean our model would require more data to predict each key-points accurately. Additionally, the posture did not make much sense on this initial model compared to how the skeletal structure of the chicken would look. Therefore, we made the decision to simplify the model to 9 points that more closely resembled the skeletal structure of the chicken.


**Figure 4.1**: Toy chicken containing 47 keypoints

Once again, we created a new dataset on the toy chicken which contains the new set of points that we discussed earlier in the chicken posture section. We carefully selected the points such that they were visible from most view points and marked them with a blue dot on the toy chicken to allow for easier annotation. Then, we shot a video on an iPhone 13 and slowly rotated the toy chicken to gather various view points so that we can test the efficacy of the model at different

orientations of the chicken. An image of the new toy chicken annotations can be seen below.

**Figure 4.2**: Toy chicken containing 9 keypoints

One thing we noticed with the toy chicken dataset after the initial training was that the model could potentially be learning the location of the blue points rather than the intrinsic structure of a chicken. Thus, to alleviate these fears, we need to remove the blue points from the training dataset. To do so, we make use of the OpenCV library. We first convert the image from an RGB scale to an HSV scale. This allows us to select all pixels in the image such that they fall within a range of colors. We then had to find a lower bound and an upper bound of colors that allowed us to select the location of each pixel that contained any shade of blue. The lower bound had the HSV value of (75, 26,40) and the upper bound contained the HSV value (145, 255, 255). Note that these values are using the OpenCV HSV scale. With this range, we could locate each pixel in the image that contained blue. Now, we apply a blur in a 101x101 square around that point. We then iterate through each of the blue points and apply this blur, thereby removing

the blue points. In Figure 3.4, we showcase this with a red square, which denotes the area that the blur will use to average the color to replace the blue pixel. The end result is shown in Figure 3.5

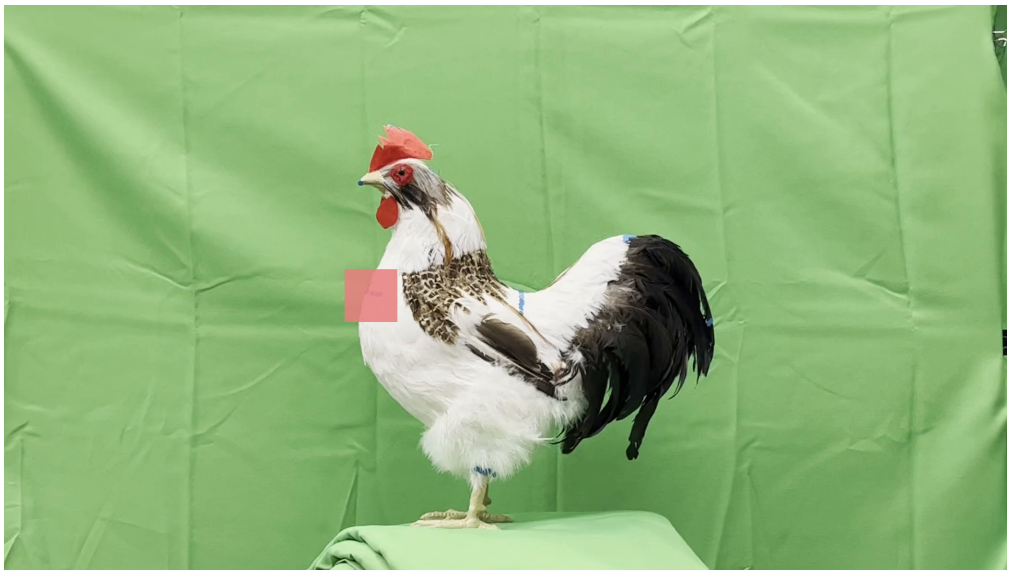

**Figure 4.3**: A visualization of the region of blurring

In addition to the OpenCV library method discussed above, we also tried to use Llama Cleaner, a model that removes an input mask and replaces the mask cleanly within an image. Our hope was that by passing in the mask of the blue points that we collect using the OpenCV library in the previous step into this model, we could remove the blue points cleanly. However, this did not pan out for unknown reasons, as the model simply output a very blue version of the same image. Thus, we trained the model on the new and blurred toy chicken dataset to test if the model is able to infer the key-points correctly, which it managed to do. This gave us the confidence that the model would be able to predict properly on a dataset of real chickens in a controlled environment. We first created a test dataset of 66 images, which will allow us to get the AP scores of both the bounding box prediction as well as the key-point predictions. Next, we created a training dataset

**Figure 4.4**: Example image of blurred dataset

of sizes 44 images as well as 100 images to compare the performance of the model on varying dataset sizes. Then, we train the model on both of these datasets as well as the blurred toy chicken dataset to compare the results.

### 4.0.5 Performance

We will go over the results of the various training sessions. Our testing set consists of 66 images of a real chicken in a controlled environment that is not present in the training data set. This is what we use to determine the efficacy of any trained model. We will first showcase the model's AP metrics, then we will showcase the model's performance on an example images to showcase how well model predicts the posture of a chicken.

The first model we trained was the toy chicken model to determine how effective the model is when trained only on the dataset of the toy chicken. The results are shown below in Table 4.1.

Next, we will look at the model where we used 44 images to train the model.

| | AP | AP50 | AP75 |
|---|---|---|---|
| Bounding Box | 4.51 | 27.25 | 0.03 |
| Keypoint | 0.11 | 1.01 | 0.00 |

**Table 4.1**: Results of toy Chicken Dataset.

The model's AP metrics are shown in Table 4.2.

| | AP | AP50 | AP75 |
|---|---|---|---|
| Bounding Box | 66.22 | 97.93 | 82.58 |
| Keypoint | 69.49 | 98.02 | 72.44 |

**Table 4.2**: Results of 44 images Chicken Dataset.

Finally, we will look at the model where used 100 images to train the model. The model's AP metrics are shown in Table 4.3.

| | AP | AP50 | AP75 |
|---|---|---|---|
| Bounding Box | 71.28 | 100.00 | 89.85 |
| Keypoint | 78.35 | 98.02 | 94.49 |

**Table 4.3**: Results of 100 images Chicken Dataset.

Now we shall showcase how model perform on two test pictures. The input images are shown in Figure 4.1.

The toy model's output is shown in Figure 4.2. In the first picture, it was able to roughly predict a bounding box, however, it could not predict many key-points. In the second picture, it was unable to detect the chicken at all.

Next, we will showcase the model trained on the 44 image dataset in Figure 4.3. As shown below, it is able to accurately predict and bound where the chicken is, however, it predicted a couple of key points incorrectly in the first picture.

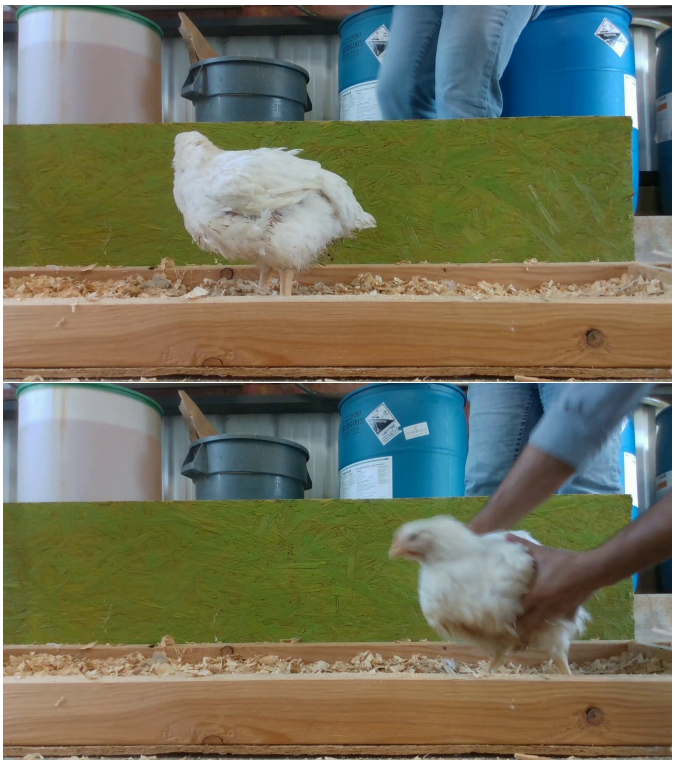

**Figure 4.5**: Input Images to showcase model performance

Finally, we will showcase the model trained on the 100 image dataset in Figure 4.4. It is able to predict most of the key-points in the first image well and had equal performance on the second image to the 44 image dataset.

As anticipated, the 100-image dataset outperformed the rest as it contained a greater variety of chicken images captured from different angles. Despite the toy chicken model having 373 images, there were several drawbacks in the dataset we curated. Firstly, the chicken in the toy model was relatively static, resulting in a limited number of poses the model could learn. As a result, the model encountered difficulties when predicting the movements of a real chicken in a controlled environment. Secondly, the toy chicken was of a different breed than the real chicken, leading to the need to consider more chicken species to improve the model. Additionally, detecting the start and end tail of the real chicken species proved difficult

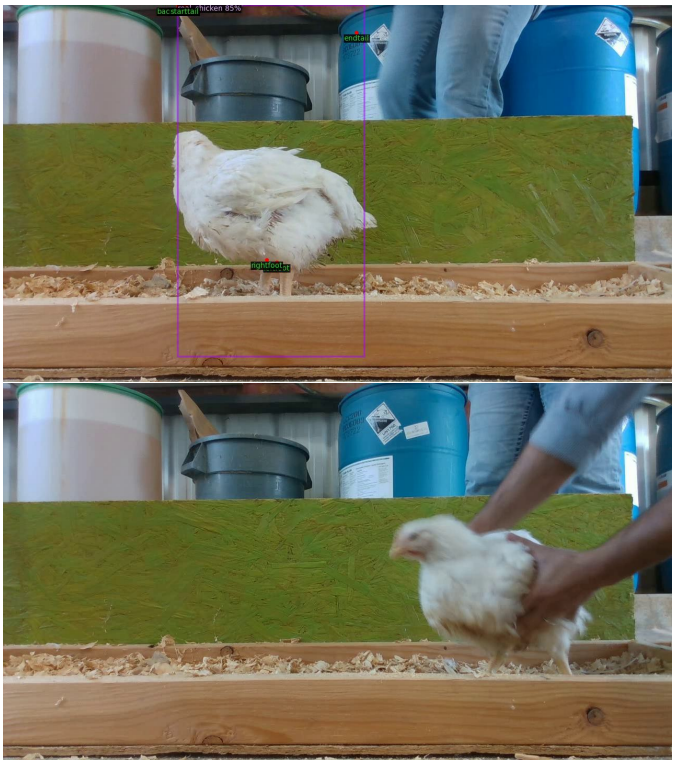

**Figure 4.6**: Toy model performance

for the model, and it could benefit from more training data with instances of these points. The models struggled with these points because they were often occluded in the training dataset due to factors such as the chicken operator's hand, motion blur, or being out of the frame. Improving the dataset with more instances of the start and endtail points will allow the model to better predict these points for chickens

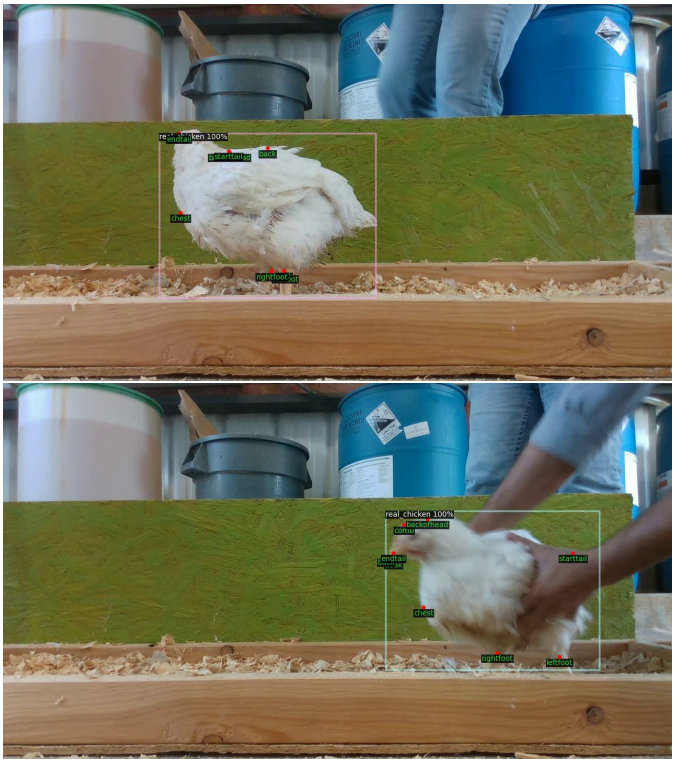


**Figure 4.7**: 44 Image model performance

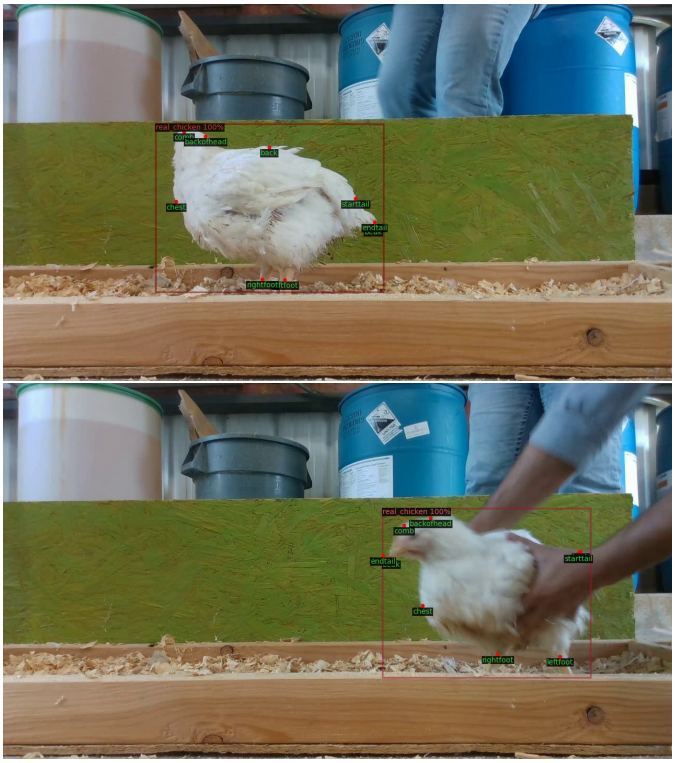


**Figure 4.8**: 100 Image model performance

## 5 Conclusion

The objective of this paper was to investigate whether advancements in Deep Learning and Computer Vision methods could enhance poultry monitoring. In pursuit of this, we curated a dataset that included annotations of diverse chicken poses in a controlled environment, enabling us to evaluate the efficacy of conventional human pose estimation algorithms. The Mask R-CNN model, implemented by Meta's Detectron 2 application, was the algorithm of choice for this study. Our findings suggest that the model performed reasonably well in predicting the chicken's posture with a moderate training data size. However, to improve the model's ability to detect chickens, we could explore techniques such as data augmentation or increase dataset quality to detect multiple chicken at once or even different chicken breeds. We envision that future research will continue to build on key-point estimation techniques to accomplish a broader range of tasks such as unique chicken identification or detecting the welfare of chicken. Ultimately, our goal is to create a more efficient and humane way of monitoring poultry rearing that benefits the birds' well-being.

## Bibliography


[1] "Human pose classification with movenet and tensorflow lite." [Online]. Available: https://www.tensorflow.org/lite/tutorials/pose_classification

[2] A. Dertat, "Applied deep learning - part 1: Artificial neural networks," Oct 2017. [Online]. Available: https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6

[3] "Poultry production and value usda." [Online]. Available: https://www.nass.usda.gov/Publications/Todays_Reports/reports/plva0421.pdf

[4] L. Song, G. Yu, J. Yuan, and Z. Liu, "Human pose estimation and its application to action recognition: A survey," *Journal of Visual Communication and Image Representation*, vol. 76, p. 103055, 04 2021.

[5] Y. Xiu, J. Li, H. Wang, Y. Fang, and C. Lu, "Pose flow: Efficient online pose tracking," *CoRR*, vol. abs/1802.00977, 2018. [Online]. Available: http://arxiv.org/abs/1802.00977

[6] "What are neural networks?" [Online]. Available: https://www.ibm.com/topics/neural-networks

[7] "What are convolutional neural networks?" [Online]. Available: https://www.ibm.com/topics/convolutional-neural-networks

[8] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[9] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[10] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[11] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[12] "Common objects in context." [Online]. Available: https://cocodataset.org/format-data

[13] "Classification: Precision and recall nbsp;—nbsp; machine learning nbsp;—nbsp; google developers." [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

[14] "Common objects in context." [Online]. Available: https://cocodataset.org/keypoints-eval