

**POULTRY POSE ESTIMATION BY DETECTING
KEYPOINTS WITH DEEP CONVOLUTIONAL NETWORK**

AMIR HAKIM BIN HILMI

**COLLEGE OF ENGINEERING
UNIVERSITI TENAGA NASIONAL**

2022

POULTRY POSE ESTIMATION BY DETECTING KEYPOINTS WITH DEEP
CONVLUTIONAL NETWORK

By

AMIR HAKIM BIN HILMI

Project Supervisor:

MR. DICKSON NEH TZE HOW

THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF ELECTRICAL AND ELECTRONIC ENGINEERING

COLLEGE OF ENGINEERING
UNIVERSITI TENAGA NASIONAL

2022

DECLARATION

I hereby declare that this thesis, submitted to Universiti Tenaga Nasional as partial fulfilment of the requirements for the degree of Bachelor of Electrical and Electronic Engineering, has not been submitted to any other university for any degree. I also certify that the work described herein is entirely my own, except for quotations and summaries sources of which have been duly acknowledged.

This thesis may be made available within the university library and may be photocopied or loaned to other libraries for the purposes of consultation.

14 January 20 22

Amir Hakim Bin Hilmi
EE0102953

DEDICATION

ACKNOWLEDGEMENT

ABSTRACT

Nowadays, there is an increasing awareness of acceptable farm animal welfare conditions, farm animal health, efficiency, and sustainable farm environments. Animal behavioural analysis can provide an insight into the health of these farm animals. Poultry chicken is a white meat source that has one of the highest worldwide demands. The conventional method to diagnose disease in poultry flock is through observation by a veterinarian and what the veterinarian determines, and this method is no longer viable because, in large-scale production, it requires many veterinarians to perform regular inspections, which is both time-consuming and labour-intensive and this will make it harder to detect sick poultry at an early stage. Hence, behavioural analysis which is a process that could be used to recognize sickness in poultry would greatly benefit not only the poultry farming industry but also the consumer. Diagnosing poultry diseases can be done through any indication shown by poultry's behaviour. One of the bases of behavioural analysis is accurate pose estimation. An accurate poultry pose estimation would be able to identify when the posture of the poultry is abnormal, with such data the farmers could find a way to cure or isolate the sick poultry. Hence, this paper is studying how a class of Artificial Intelligence (Deep Convolutional Network) can be utilized to accurately detect the poultry's body keypoints in a video. Poultry keypoint detection involves predicting the location of the specific body keypoints of the poultry like the centre of the body, head, tailbase, left and right leg. Accurate poultry's body keypoint detection using Artificial Intelligence is the basis to develop an accurate automated poultry health classifier. This paper will go into detail about the method used to gather the needed data to train the Deep Convolutional Network, methods to and tuned train the network to accurately predict and detect poultry's body keypoints on a video. This paper would also compare and find the best available Deep Convolutional Network that would be deemed suitable for keypoint detection on poultry using a state-of-the-art method called DeepLabCut which is a pose estimation toolbox.

CONTENTS

	Page
DECLARATION	III
DEDICATION	IV
ACKNOWLEDGEMENT	V
ABSTRACT	VI
CONTENTS	VII
LIST OF TABLES	X
LIST OF FIGURES	XI
LIST OF ABBREVIATIONS	XIII
CHAPTER 1 INTRODUCTION	1
1.1 General Background	1
1.2 Problem Statement	2
1.3 Proposed Solution	2
1.4 Objectives of the Research	2
1.5 Scope of Thesis	3
CHAPTER 2 LITERATURE REVIEW	5
2.1 Introduction	5
2.2 Keypoint Detection	5
2.2.1 <i>Definition and human application</i>	5
2.2.2 <i>Significance of Poultry Pose Estimation</i>	6
2.3 Deep Learning	7
2.3.1 <i>Convolutional Neural Network</i>	7
2.3.2 <i>CNN in Keypoint Prediction</i>	8

2.4	Related Work	9
	<i>2.4.1 The Problem with Detecting Poultry</i>	9
	<i>2.4.2 Deep Learning Technique to Detect Poultry Chicken</i>	10
	<i>2.4.3 Methods of Detecting Sick Poultry</i>	12
	2.4.3.1 Machine learning method	12
	2.4.3.2 Deep learning method	12
	<i>2.3.4 Related Work Conclusion</i>	14
	CHAPTER 3 METHODOLOGY	16
3.1	Introduction	16
3.2	Software Used	16
	<i>3.2.1 DeepLabCut</i>	16
	3.2.1.1 GUI installation with Anaconda environments	17
	<i>3.2.2 Google Colab Pro and Google Drive</i>	19
3.3	Dataset Preparation	20
	<i>3.3.1 Data Gathering</i>	20
	<i>3.3.2 Data Augmentation</i>	21
	<i>3.3.3 Labelling Data</i>	22
	3.3.3.1 Creating project	22
	3.3.3.2 Labelling convention and tool	24
3.4	Training the Deep Convolutional Network	30
	<i>3.4.1 Transfer Learning</i>	30
	<i>3.4.2 Hyperparameters</i>	31
	3.4.2.1 Learning rate	31
	3.4.2.2 Train-test split ratio	33
	3.3.2.3 Training iteration	34
3.5	Network Performance	36
	<i>3.5.1 Evaluation and the Metrics Used</i>	36
	3.5.1.1 Pixel error and confidence level	36
	3.5.1.2 Training loss	37

3.6	Keypoints Prediction on a Video	38
	3.6.1 <i>Confident level or likelihood</i>	38
	3.6.2 <i>Create Predicted Labelled Video</i>	40
3.7	Flowchart	42
	CHAPTER 4 OBSERVATION AND RESULTS	43
4.1	Overview	43
4.2	Performance of Resnet-50, Resnet-101, and Mobilenet V2	43
4.3	Testing Different Training and Test Dataset Split Ratio	45
4.4	Network Performance of Default and Custom Learning Rate	46
4.5	Custom Model Comparison	48
	CHAPTER 5 ANALYSIS AND DISCUSSION	50
5.1	Important Concept for Discussion	50
	5.1.1 <i>Overfitting</i>	50
	5.1.2 <i>Pixel Error Categorization</i>	50
5.2	Best Network Supported by DeepLabCut	51
5.3	Best Train and Test Splitting Ratio	53
5.4	Custom Learning Rate Performance	54
5.5	Best Performing Custom Model	55
	CHAPTER 6 CONCLUSIONS	59
	REFERENCES	61
	APPENDICES	65
	APPENDIX A	65
	APPENDIX B	68

LIST OF TABLES

Table No.		Page
3.1	Parameters of the config file	24
3.2	Parameter for bodyparts	27
3.3	Parameter for skeleton	27
3.4	Default dynamic learning rate	33
4.1	Default dynamic learning rate	44
4.2	Performance at 100,000 training iterations	44
4.3	Performance at 200,000 training iterations	44
4.4	Performance at 300,000 training iterations	44
4.5	Frames used for each splitting ratio	45
4.6	Performance at 100,000 training iterations	45
4.7	Performance at 200,000 training iterations	45
4.8	Performance at 300,000 training iterations	46
4.9	Default dynamic learning rate	47
4.10	Custom dynamic learning rate	47
4.11	Performance at 100,000 training iterations	47
4.12	Performance at 200,000 training iterations	47
4.13	Performance at 300,000 training iterations	47
4.14	Characteristics of CM1 and CM2	48
4.15	Custom dynamic learning rate	48
4.16	Performance at 200,000 training iterations	49
4.17	Performance at 300,000 training iterations	49
5.1	Pixel error category	51
5.2	Architecture of different Resnet variants	56

LIST OF FIGURES

Figure No.		Page
2.1	Posture comparison of sick and healthy chicken	6
2.2	Typical architecture of CNN	7
2.3	Deconvolutional layer and CNN	8
2.4	Convolutional neural network architecture	10
2.5	Feature points of a chicken	13
2.6	Chicken-pose estimation process	14
3.1	Command of installing DLC-GUI	18
3.2	Commands to launch the DLC-GUI	19
3.3	A frame of a video from the dataset	20
3.4	Original and flipped frame of the same video	22
3.5	New project interface	23
3.6	Newly created project folder	24
3.7	Labelling convention used for chicken	25
3.8	Colormap options	27
3.9	DeepLabCut labelling interface	28
3.10	Contents of labelled-data folder	29
3.11	DeepLabCut code for training dataset creation	30
3.12	Various effects of learning rate on loss	32
3.13	Default learning rate within the pose_cfg.yaml file	33
3.14	DeepLabCut code for network training	35
3.15	Example of the snapshot file	35
3.16	DeepLabCut network evaluation code	36
3.17	Output of DeepLabCut network evaluation code	37
3.18	Human labels plots against network predicted plots	37
3.19	Training loss equations	38
3.20	Human label against low confident network prediction	39
3.21	Human label against high confident network prediction	40
3.22	DeepLabCut code making keypoint prediction	40
3.23	DeepLabCut code for plotting keypoints trajectories	41
3.24	Trajectories plots	41

3.25	DeepLabCut code for creating a video with predicted keypoints	41
3.26	Output video using the trained network	42
3.27	Project flowchart	42
5.1	Train pixel error across different networks	51
5.2	Test pixel error across different networks	52
5.3	Discrepancy test-train across different networks	52
5.4	Discrepancy test-train across the different splitting ratio	53
5.5	Discrepancy test-train across the different learning rates	54
5.6	Train pixel error across different custom models	57
5.7	Test pixel error across different custom models	57
5.8	Discrepancy test-train across different custom models	58

LIST OF ABBREVIATIONS

Abbreviation	Meaning
GS	Gait Score
PLF	Precision Livestock Farming
DLC	DeepLabCut
DL	Deep Learning
DCN	Deep Convolutional Network
CNN	Convolutional Neural Network
NN	Neural Network
MAE	Mean Euclidean Error
HDF	Hierarchical Data Format
SGD	Stochastic Gradient Descent
FC	Fully Connected
CM	Custom Model

CHAPTER 1

INTRODUCTION

1.1 General Background

The future global meat consumption is predicted to increase by 70% by 2050 [1] followed by a projection of over 9.96 billion people of the world population by the year 2050. Hence, there is an increase in food security concerns which causes an increase in agricultural production. Henschion et al. [2] reported that there is an indication of increased consumption of poultry meat and poultry meat products, with a projected increase within the next decade due to preferences for white meat [3]. The growing demand has mostly been driven by urbanization and rising incomes in developing countries[4]. Acceptable animal welfare conditions, animal health, efficiency, and sustainable environmental condition have become more challenging to fulfil due to the increasing amount of chicken production [1]. Poultry flock welfare is usually assessed through mortality physiology, behaviour, and walking ability [5].

Walking ability or lameness is one of the common traits that can be used to determine the welfare of poultry [6]. Lameness is a word used to describe a range of injuries with infective and non-infective sources experienced by the poultry [7]. However, Aydin [6] stated that lameness can also be said as poultry's ability to run. Lameness in poultry is usually ranked by trained farmers and veterinarians using a gait score system. Gait score zero (GS0) would mean that there is no detectable abnormality, fluid locomotion, and furled foot when raised shown from the poultry and gait score five (GS5) would mean that the poultry is in complete lameness, either cannot walk or cannot support its weight on its leg. According to paper [8] and [9], this procedure gives a basis for future management in welfare decisions. However, using this technique visually on a large flock might lead to biased results since the evaluation might vary between different individuals [10]

1.2 Problem Statement

In recent years, poultry disease outbreaks have occurred frequently, and this is leaving a bad impact on the poultry industry. Currently, the conventional method for monitoring poultry disease is mainly carried out by manual observation of poultry posture, feathers, cockscombs, faeces, and sounds [11]. The problem with manual observation is, in large-scale production, it requires many people to perform regular inspections, which is both time-consuming and labour-intensive and this will make it harder to detect sick poultry at an early stage [12]. Hence, human surveillance has ceased to be a viable solution in livestock farming [13]. Precision Livestock Farming (PLF) has been used to solve these challenges by using efficient automated systems while at the same time maintaining animal welfare [14].

1.3 Proposed Solution

With the advancement of artificial intelligence and specifically in the branch of the deep convolutional network, this paper proposed an automated system that could detect the poultry's body keypoints using a deep convolutional network. Obtaining the body keypoints of poultry is a crucial step in developing an automated system that could detect the poultry's mobility or posture. However, this paper is focusing on posture estimation rather than mobility due to the type of data that could be gathered (will be further discussed in Chapter 3).

1.4 Objectives of the Research

Below is the list of objectives for this paper.

- i. To produce a poultry dataset that is annotated with the body keypoints.
- ii. To propose a deep learning model that can detect the keypoints of a poultry on a video.
- iii. To compare the accuracy of the proposed models to other models with different hyperparameters and backbones.

1.5 Scope of Thesis

This part will explain the flow of this thesis which consists of six chapters that begin with an introduction, literature review, methodology, observation & results, analysis of result and general discussion, and conclusion & recommendation.

The first chapter serves as an introduction to the overall paper. This chapter addresses the research's context, identifies the problem statement, defines the project objective, and briefly explains the scope of work. This chapter aims to make it clear about the significance of the problem and the research work outline.

The second chapter contains the summary of the literature review and the most advanced methods used by other researchers that is related to this project. This part will also discuss the Deep Convolutional Network alongside keypoint and posture detection for both human and poultry applications, based on which the subsequent work was done and the decisions that are made.

Chapter three review the methodology which will go through the type of software used and how it was used followed by the data preparation, the Deep Convolutional Network training process, the evaluation metrics of the trained network, and finally, the flowchart of this entire project. This chapter aims to provide a detailed guide that other researchers could use to replicate it. Lastly, this part will also talk about the advantages and the limitations of the methods used.

Chapter four contains the details regarding the observation and results. In this chapter the results presented are from a series of test that was done to find the best hyperparameters that should be used to train the best deep convolutional network model.

The purpose of chapter five is to evaluate and interpret the test results that are presented in chapter four. Here is where the choices made for the final tuned hyperparameters are further justified using the obtained results and the theory behind

the hyperparameters. This chapter also aims to discuss whether the final recommended model of the tuned network has any significant improvement than the untuned network.

Chapter six is the final chapter where the outcomes of this project are summarized. This part will also present whether the findings and results obtained have met the expectations and the project objectives. Finally, this chapter will also mention the limitations of the study and ways that it can be further improved alongside the possibilities and suggestions on future work that can be done by using the findings in this study.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter will highlight the importance and the role that keypoint detection had in the human application and followed by relating it to the reason why keypoint detection is a crucial part of poultry pose estimation with a further argument on the significance of poultry pose estimation. This chapter will also briefly focus on deep Convolutional neural networks (DCNN) and compare them to a normal Convolutional Neural Network (CNN). Finally, this chapter will review a few related works that use different artificial intelligence (AI) based methods to solve the problem statement presented in chapter 1.

2.2 Keypoint Detection

2.2.1 Definition and human application

Keypoint detection in humans involves locating parts of human limbs, or facial features. These parts help to represent the underlying object in a feature-rich manner [15]. The common application of keypoint detection is pose estimation and facial expression estimation. For any automated system that could do pose estimation on a human would also mean that the system can do posture detection. In medical terms, posture is the carriage of the body as a whole, the attitude of the body or the placement of the limbs (the arms and legs) are all aspects of posture [16]. Human posture detection enables the acquisition of the human body's kinematic properties, which is useful in a variety of applications such as assisted living, healthcare, physical exercise, and rehabilitation. Recent advances in deep learning and computer vision can substantially aid this effort.

2.2.2 Significance of Poultry Pose Estimation

Based on the research process and the literature review of other common work (will be discussed in section 2.3), there are two traits of all poultry that can be utilized to develop an automated system that can determine the poultry health state, the first is the poultry mobility or lameness which is usually judged by the ability of poultry to walk, and the second trait is the posture that the poultry is displaying. This paper is focusing on detecting the posture of a poultry chicken.

The process of determining the health of poultry by using its posture can be simplified into three main processes. The first process is to detect the important keypoints on the poultry's body which then can be used in the second process where the detected key points on the bird's body is used to estimate the bird's pose (standing, preening, running, or sitting) and finally, this pose estimation can be used to predict the bird's health. The posture of poultry could describe the health of a poultry, most used type of poultry in this paper is chicken, and sick chicken is often reluctant to walk for very long and will isolate itself and displays a depressed bird look/posture [17] and Figure 2.1 is showing a comparison image of the posture of a sick chicken (a) and a healthy chicken (b).



(a) Sick chicken posture

(b) Healthy chicken posture

FIGURE 2.1 Posture comparison of sick and healthy chicken

2.3 Deep Learning

2.3.1 Convolutional Neural Network

Convolutional Neural Network (CNN/ConvNet) is a class of Deep Learning algorithms used to evaluate visual imagery by assigning relevance (parameters: learnable weights and biases) to various aspects/objects in the image and using those parameters to distinguish between the different classes of object. One of the differentiating factors between CNN and other Deep Learning algorithms is that CNN makes use of a technique called Convolution. Convolution is a mathematical operation on two functions that yields a third function that explains how the shape of one is changed by the other [18]. Figure 2.2 is a typical architecture of a CNN.

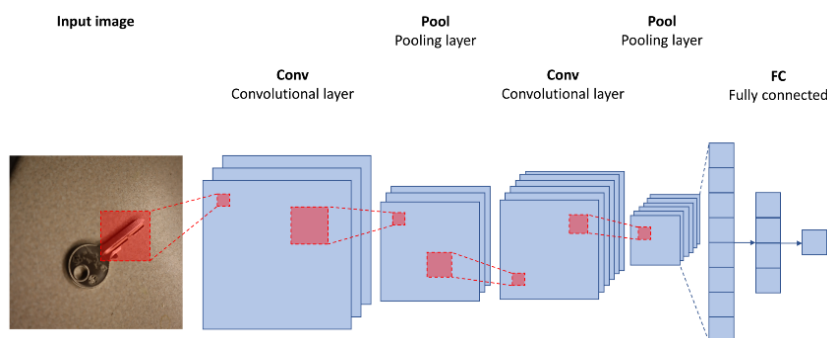


FIGURE 2.2 Typical architecture of CNN [19]

The architecture of a CNN is inspired by the structure of the Visual Cortex and is akin to the connectivity pattern of Neurons in the Human Brain. Individual neurons can only respond to stimuli in a small area of the visual field called the Receptive Field [20]. Several similar fields can be stacked on top of each other to span the full visual field. A CNN network starts with an image as input then it passed sequentially into convolutional, pooling, and fully connected (FC) layers. The convolutional layer contains a set of filters (or kernels) and the learned parameters during the training process. When the images convolve with each filter, it creates an activation map [21]. The pooling layer has a function that plays a role in reducing the spatial size of the

convolved features and this is to reduce the amount of computing power needed to process such data. The fully connected layers are the last few layers of the network, this layer performs classification tasks using the features collected by the previous layers.

2.3.2 CNN in Keypoint Prediction

CNN architecture for object classification is slightly different than the CNN architecture used for a keypoint prediction task. As an example, in DeepLabCut (a method used in this project, which will be further discussed in chapter 3) uses a combination of normal object classification CNN and a deconvolutional layer. Unlike normal object classification CNN that have a classification layer (such as the Fully Connected layers in Fig. 2.2) at its output, it was replaced with deconvolutional layers (opposite of convolutional layers) that are used to generate spatial probability densities, which represent the probability of the body being in a particular position. Figure 2.2 is showing the architecture of Resnet-50 (a variant of a CNN) with a deconvolutional layer at its output.

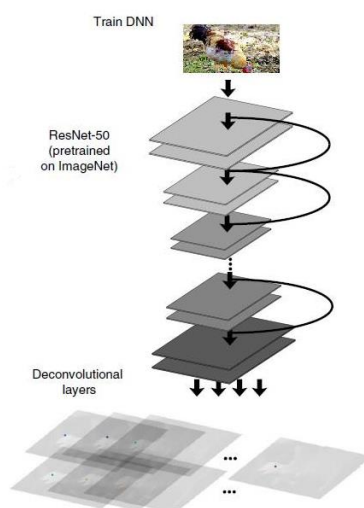


FIGURE 2.3 Deconvolutional layer and CNN

2.4 Related Work

A few theories and methods were mentioned in this section to have a better understanding of how deep learning or machine learning can help in determining the health status of poultry chickens. This section presents a posture and mobility evaluation to provide a technique for distinguishing healthy and sick birds. To monitor poultry welfare, posture, and mobility analysis is the common method as this will determine the poultry lameness and then enable us to determine their health. In some chicken farms, it has been detected that around 27% of the chicken showed poor movement and 3% of chickens were unable to move [10]. When a chicken is having locomotion problems, the bird will have low mobility and this will increase the chances of that bird having other problems such as chest soiling [22].

2.4.1 *The Problem with Detecting Poultry*

The advancement of computer vision and deep learning in the past decades has resulted in deep learning being applied more and more in animal analysis research. In some animal posture studies, some additional markers are usually placed on the animal to enhance the recognition precision [23]. However, with the precision that these markers provide, this type of system is expensive and distracts animals because they are invasive. Cameras are the common tool researchers used for monitoring the diet status, behavioural habits, and physiological changes of poultry in real-time [24]. Cameras have the advantage of not being invasive which reduces their impact on the animal's life.

The difficulty of poultry monitoring is in the foreground detection which causes by the complex background, variations in illuminations and, occlusion problems in a real poultry farm environment. When the colours and the texture of the foreground and background are similar, it is hard to segment the chicken from the background. Zhuang et al. [25] stated that early warning of changes in the health status of the chicken has always been a difficult challenge for research. Due to the nature of

the chicken that tends to stay densely stocked and like to gather very closely to each other, it is very difficult to accurately detect each chicken. Hence, it is hard to judge their health status if detecting them is a problem.

2.4.2 Deep Learning Technique to Detect Poultry Chicken

Convolutional Neural Network (CNN) is one of the most used deep learning architectures in digital image processing. It is commonly used in object detection and classification task in computer vision. CNN is a multi-layered network that can learn features of a target to perform an autonomous detection. It comprises several neural layers: convolutional, non-linear activation layer, pooling, and fully connected layers. Thus, resulting in the mapping of an input to a 1D feature vector. Figure 2.4 presents the general structure of CNN architecture [26].

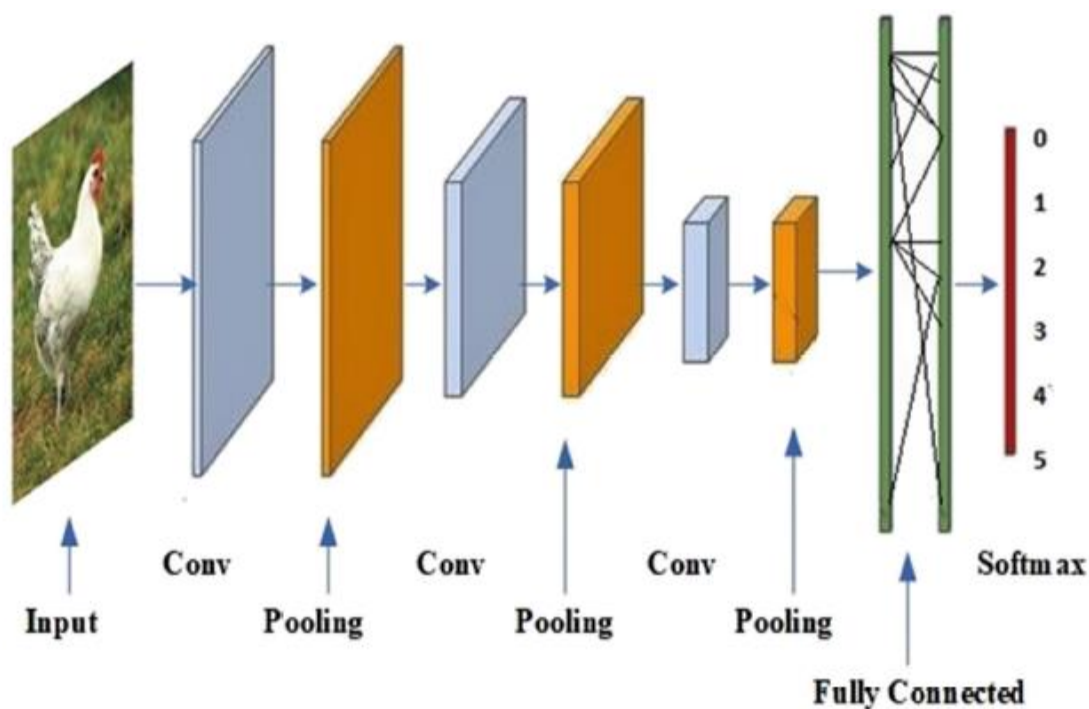


FIGURE 2.4 Convolutional neural network architecture [26]

Sergeant et al. [11] proposed a poultry tracking system based on pixel information by segmenting the outline of the broiler chicken and using the centroid segmentation algorithm. However, such a method may be hard in a dense environment where the broiler gathers very close with each other. Fang et al. [27] use the Tbroiler tracking algorithm to track a single chicken in a flock and analysed the related indexes, such as the overlap rate (OR) and the pixel error (PE). Various classical CNNs can be used in classification, such as VGGNet, Google Inception Net, NASNetMobile, ResNet, and Xception [28]. These methods provide the scope for deep learning solutions to judge the health status of broilers [12].

Single Shot MultiBox Detector (SSD) is designed for object detection in real-time. Faster Recurrent Convolutional Neural Network (Faster-RCNN) uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects and it is considered excellent in terms of accuracy. However, it runs at 7 frames per second which is far below what real-time processing needs [29]. SSD speeds up the process by eliminating the need for a region proposal network. Each bounding box can be identified as to whether it is a target or not, and regression can be used to find the exact location [12]. Using multi-scale features SSD has the advantage of being able to detect, identify targets in real-time and performs very well with overlapping targets, but the detection of small targets such as broiler chicken is not ideal.

Zhuang and Zhang [12] introduced a target detection algorithm called Improved Feature Fusion Single Shot MultiBox (IFSSD) to classify healthy and suspected sick chickens. IFSSD uses FSSD, where FSSD concatenates the features at each level and generates the feature pyramid network (FPN) while SSD only predicts the features directly at various levels and there is no connection between each level. One of the ways Zhuang and Zhang improved the FSSD is by adding the PASCAL VOC2021 dataset [30] containing 17,125 images as background images for training and to reduce the false recognition rate and improved the robustness of recognition. The detector could detect broilers and their health simultaneously at mean average precision (MAP) of 99.78%.

2.4.3 Methods of Detecting Sick Poultry

2.4.3.1 Machine learning method

The study on poultry mobility done by Nääs et al. [5] uses a machine learning approach to judge the health status of the poultry chickens based on their mobility. In the author's research, the birds were separated by their Gait Score (GS) to represent the ranges of scores, and a video recording was made. The pre-defined scores aim to be able to study the various ranges of chicken mobility. The birds were stimulated to walk on the specially built platform and the background of the recording video was a blue wall to provide a contrasting background from the birds. The video signals were then converted to an image and the centroid of the chicken body was determined. The distance moved by the broiler's centroid between a pair of consecutive frames was used to calculate the broiler speed, velocity, and acceleration. The analysed videos formed a matrix database that is used in data mining analysis.

2.4.3.2 Deep learning method

For a deep learning approach, Fang et al. [31] used deep neural networks to estimate the pose of the chicken to determine its behaviour classification. Accurate pose estimation is important for poultry behavioural analysis, and it is also able to provide an early warning method to poultry disease. First, the researchers created a map of a broiler chicken pose skeleton. As seen from Figure 2.5, there are ten custom feature points in this pose skeleton. Point 1 is the centre point of the broiler chicken [25]. Point 2 is the tail point, point 7 and 8 is the left and right eye, point 9 is the highest points of the comb, and point 10 is the tip of the beak. Points 3 and 4 represent the knee point and it is also the dividing points of colour between the legs and the feathers. Points 5 and 6 are heel points which are the joints of the phalanges.

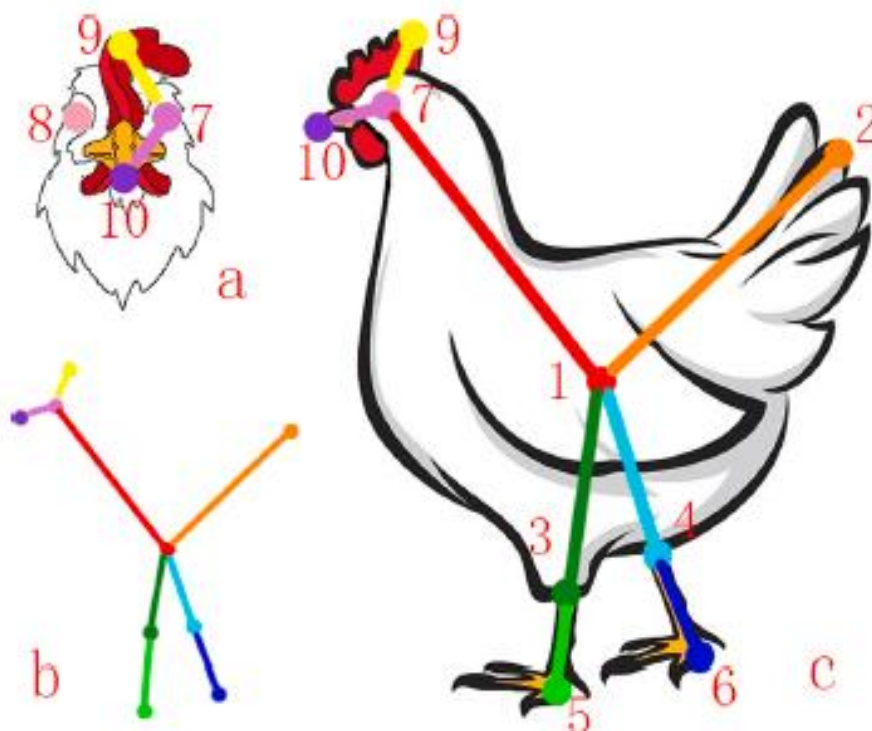


FIGURE 2.5 Feature points of a chicken [27]

The experiment was done with four K90 broilers (between 40 and 50 weeks old) in an area of $4\text{ m} \times 3\text{ m}$ to ensure that the broiler chickens have enough space for free movement while the videos were taken. A broiler chicken is feeding through and a set of water tanks within that area. The system consists of a high-definition video camera and a computer. Captured images were transmitted to a computer via a serial bus (USB) port for pose estimation and behaviour classification.

The process of chicken-pose estimation introduced by Fang et al. [31] is shown in Figure 2.6. In (a) the region of interest (ROI) in the training images were extracted including the posture of the broiler chicken. (b) showed that the feature points of the chicken body parts were marked as ROI and these features are as discussed and mentioned in the first paragraph of this section and the features points are as shown in Figure 2.5. The image and the coordinate of the marked points were then sent to the network to train the algorithm. Part (c) showed that, to the prediction of the position of

the chicken body part, the pre-trained ResNet-50 is used. ResNet-50 is a 50-layer residual network, and its parameter is obtained by pretraining on ImageNet. In (d), the spatial probability density is obtained by sampling the chicken image information using a deconvolutional layer (DL). The result is a trained network that can extract the locations of the different feature points from the test video.

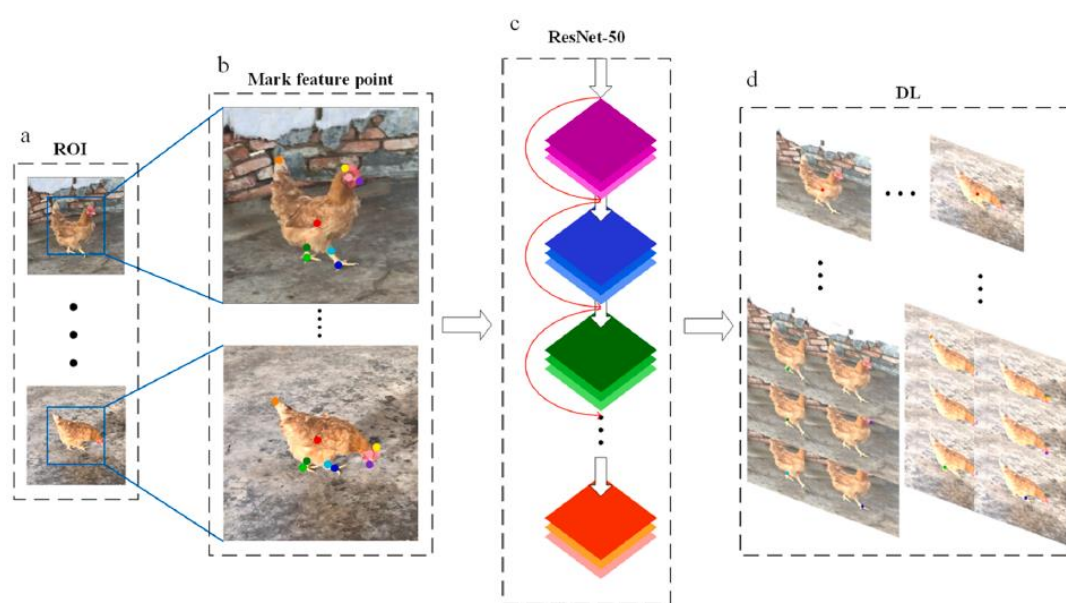


FIGURE 2.6 Chicken-pose estimation process [31]

2.3.4 Related Work Conclusion

The deep learning technique to detect poultry mentioned in section 2.4.2 is to address the problem that is presented in section 2.4.1 related to how the poultry chicken is usually gathering closely to each other in a real farm environment. In my opinion, the machine learning method proposed by Nääs et al. [5] in section 2.4.3.1 is suitable to detect the mobility of the chicken and classify their health in a controlled environment. However, in a real farm environment, the machine learning method mentioned is unsuitable because there is no confirmation that the poultry will walk through the specially built platform as in the method in section 2.4.3.1. Therefore, for a real farm environment, the deep learning method as in section 2.4.3.2 as proposed by Fang et al.

[31] is much more suitable. This is because the deep learning method can estimate the pose of the chicken and its behaviour classification rather than having to analyse the movement of the bird which will require a clear movement of chicken recorded on the camera to calculate its speed, acceleration, and velocity like the method mentioned in 2.4.3.1 rather than just having a video of a chicken doing a certain pose. Hence, this paper is focusing on a deep learning technique called deep convolution network to estimate the pose of poultry chicken by the means of keypoint detection.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter reviews the methods in detail which will start with the software that was used, how such software was installed or accessed, and how the software is relevant for this study. This chapter will also specify how the data was gathered, what manipulation was done to the data, and how the data is used. This followed by an explanation of how the artificial intelligence part of this project which is the Deep Convolutional Network (DCN) was trained and what parameters were changed to fine-tune it based on the type of dataset that was gathered. Lastly, this chapter will present the available methods that are used to evaluate the performance of the trained network and followed by the method used for inferencing using the trained network.

3.2 Software Used

3.2.1 DeepLabCut

DeepLabCut (DLC) is an open-source toolbox for a robust approach of 2D or 3D markerless pose estimation of animals performing a variety number of tasks by utilizing deep neural networks with transfer learning. DeepLabCut, developed in the year 2018 by Mackenzie and Alexander Mathis, a married couple of neuroscientists. DeepLabCut is essentially a tweaked version of DeeperCut, a neural network algorithm developed by other researchers to detect and label human poses in videos [32]. DeeperCut is good at detecting human poses as it was trained on thousands of hand-labelled frames.

However, to predict the label on a different species, the training state must be done from the start again. Hence, what the Mathises has done is to tweak the DeeperCut and pretrained it on ImageNet, which is a huge image database for image classification.

Now the DeeperCut algorithm is given a basic visual system that can tell apart between different images such as a chicken and car. Now that it can tell apart from different species of animal, the network is ready to recognize more specific body parts of the animals and predicts its keypoints.

In this project, the DeepLabCut toolbox was used in two different methods to achieve different goals. The first methods are by using the DeepLabCut graphical user interface (GUI) to label the extracted frames from the collected video dataset which is also the first objective of this project. The second way DeepLabCut was utilized is by using its Python command to train and evaluate the deep convolutional network and this is where the second and third objectives of this project are attempted to be met.

The reason for choosing DeepLabCut instead of training a deep neural network from scratch is because DeepLabCut is beginner friendly as it provides a clear and comprehensive pathway on how to tune the deep neural network that is supported by DeepLabCut and how to adapt it to the type of dataset that is relevant to this project. Even so, that does not mean DeepLabCut is the perfect tool and without its limitation. The main trade-offs are that because of the tight integration between each step of training the deep neural network, only a limited number of the deep neural network is supported, and it is also lacking in the ability to modify the layers of the deep neural network itself which could be a huge drawback for seasoned deep learning engineer.

3.2.1.1 GUI installation with Anaconda environments

To install the DeepLabCut graphical user interface (GUI), the installation was done through Anaconda virtual Environment as recommended by the developer of DeepLabCut itself. Anaconda is a Python package built by Continuum Analytics that comes preloaded with several major Python libraries that is related to data science task. The main benefit of installing DeepLabCut GUI through Anaconda is because Anaconda enables the creation of an environment that is isolated and not affected by the operating system or the administrative libraries of the local hardware which could

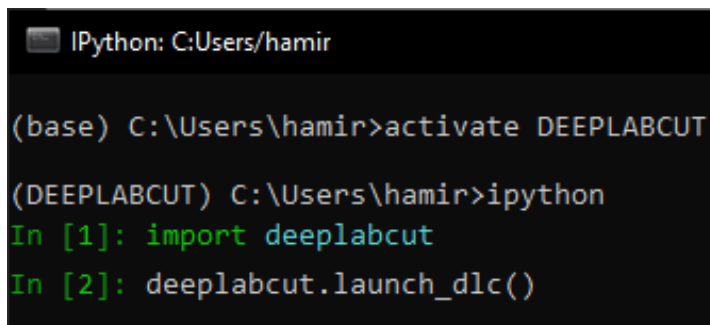
cost compatibility problems between the different Python libraries [33]. The Anaconda environment can be downloaded directly from any browser.

The compatibility problem mentioned did happen initially when installing DeepLabCut directly from the command prompt of the local personal computer. However, by using Anaconda environments, such a problem was overcome. The DeepLabCut GUI is only used in this project for frame extraction and keypoint labelling purposes. The training and evaluation process can be done using the GUI itself but with limited features lacking such as the hyperparameters manipulation.

The version of DeepLabCut GUI installed on the Windows 10 personal computer is the central processing unit (CPU) version as the personal computer used for this project is not equipped with any supported graphic processing unit (GPU). The commands shown in Figure 3.1 are run on the anaconda prompt (already pre-installed with Anaconda) is to install the DeepLabCut GUI. Figure 3.2 is the set of commands used to get into the DeepLabCut environment and launch the GUI from the Anaconda Prompt. From Figure 3.2, we could see that before entering the DeepLabCut environment, on the topmost left side of the terminal, within the bracket is showing the word base, which would mean that the system is not yet entering the DeepLabCut environment. Based on Figure 3.2, after activating the DeepLabCut environment the word base will change into DeepLabCut to indicate that the system has entered the DeepLabCut environment, and only after entering this environment the DeepLabCut GUI can be launched.

```
C:\Users\hamir>cd desktop
C:\Users\hamir\Desktop>cd dlc
C:\Users\hamir\Desktop\dlc>git clone https://github.com/DeepLabCut/DeepLabCut.git
C:\Users\hamir\Desktop\dlc>cd DeepLabCut/conda-environments
C:\Users\hamir\Desktop\dlc\conda-environments> conda env create -f DEEPLABCUT.yaml
```

FIGURE 3.1 Command of installing DLC-GUI

A screenshot of a terminal window with a black background and white text. The window title is 'IPython: C:\Users\hamir'. The terminal shows the following sequence of commands and outputs:

```
(base) C:\Users\hamir>activate DEEPLABCUT
(DEEPLABCUT) C:\Users\hamir>ipython
In [1]: import deeplabcut
In [2]: deeplabcut.launch_dlc()
```

FIGURE 3.2 Commands to launch the DLC-GUI

3.2.2 Google Colab Pro and Google Drive

Google Colaboratory or also known as Google Colab is a product of Google's research with Jupyter notebook. Google Colab is a cloud computing service that is allowing developers, students, or researchers to build and run Python programmes directly from their browser. The first reason that Google Colab is an excellent platform to code from is that many widely used machine learning libraries are supported or pre-installed by Colab and can be easily loaded into the notebook and most importantly Google Colab will also support the DeepLabCut library. The second advantage of using Google Colab, for a deep learning project that works with a large amount of data such as this project, Google Colab can work with all the saved files in Google Drive, and because of this feature both codes and the data is safe and easy to backup and restore in the case of saving error. Finally, the biggest advantage of using Google Colab is having the ability to utilize Google graphic processor unit (GPU) for training the model which is also the recommended way to train the network by the developer of DeepLabCut.

The reason that GPU is better at training a deep learning network than the CPU is that the GPU is better at parallel computing when compared to the CPU [34]. Parallel computing is a type of computing architecture in which numerous processors work together to do a series of smaller tasks that are broken down from a bigger, more difficult problem, and this is crucial when training a deep convolutional network with a large amount of data. However, using Google Colab does have a few drawbacks

such as every time starting a new test or session, the DeepLabCut library must be re-installed albeit with a single line of code. The biggest drawback when using Google Colab is that users can only stay connected to Google's computing resources such as the GPU and the runtimes with the maximum duration of 24 hours for the paid version called Colab Pro (the version that was used for this project).

3.3 Dataset Preparation

3.3.1 Data Gathering

The type of data that is needed to train a keypoint detection AI that uses a deep convolutional network is in the form of a short video of poultry and the type of poultry that this paper is focusing on is chicken. This is because out of all available poultry meat sources such as turkey, duck, quail, and many others, chicken is the most consumed poultry meat [35]. The attribute of this short video is that it would mainly consist of one main chicken in a frame and this chicken will do various kinds of poses such as roaming, pecking, or scratching. Before the augmentation process, a total of 19 short videos were gathered that are well-suited with the attributes that were mentioned beforehand. These short videos were collected from various websites available online such as Pixabay, Pexels, YouTube, and Shutterstock. Figure 3.3 is showing a frame of a video that was obtained from Pixabay.



FIGURE 3.3 A frame of a video from the dataset

3.3.2 Data Augmentation

Data Augmentation is a term that refers to a range of measures for expanding the size or quality of training datasets so that robust Deep Learning models can be generated using them [36]. Without data augmentation, only 19 short videos were gathered and from these 19 short videos only 20 frames (default number that the software in the next section will use) from each video will be used and that will total to 380 frames that are available to use for training. Do note that, the Deep Convolutional Network only uses frames that are extracted from each video in the dataset and not the video itself. When comparing the number of frames that is available for this paper to one of the works that were in the literature review, C. Fang et al. [31] uses 556 frames and only use five to six frames from each video, and with that, it is clear that without data augmentation, this project video dataset is small and not diverse. The meaning of not diverse here is when the deep convolutional network was trained using too much frame from the same video which would cause the deep convolutional network to recognize a smaller pool of features.

If this project were to follow C. Fang et al. [31] number of training frames per video, this project will have a significantly lower total number of frames that are available to train the deep convolutional network. Hence, the dataset of this project must go through an augmentation process to expand the number of videos available which would mean an increase in the total number of training frames while also decreasing the number of frames per video needed to be used for training to increase the diversity of the dataset and increase the pool of features that the network could recognize. One augmentation process that is relevant to the type of content that the video dataset of this project has, is the horizontal flipping process and this process is done manually using an online tool accessible from any browser called Animaker. Flipping the image protects the image's features while rearranging the pixels [37]. With the augmentation process, the video dataset was increased from 19 to 28, and not all videos undergo the flipping process. Figure 3.4 shows two frames that were extracted from an original and a flipped of the same video that is in the video dataset.



(a) Original frame



(b) Horizontally flipped frame

FIGURE 3.4 Original and flipped frame of the same video

3.3.3 Labelling Data

3.3.3.1 Creating project

The data was labelled using the DeepLabCut (DLC) graphical user interface (GUI) which was mentioned in section 3.2.1.1. However, before the labelling process, the first step of using the DLC GUI is to create a project by filling in the details needed to create the project. The DLC interface when creating a project is shown in Figure 3.5 alongside an example of the needed details that are already been filled in. The details needed to create a project using the DLC GUI would be the name of the project, the name of the experimenter (the labeller), and the directory file where the project (the labelled frames) would be saved at.

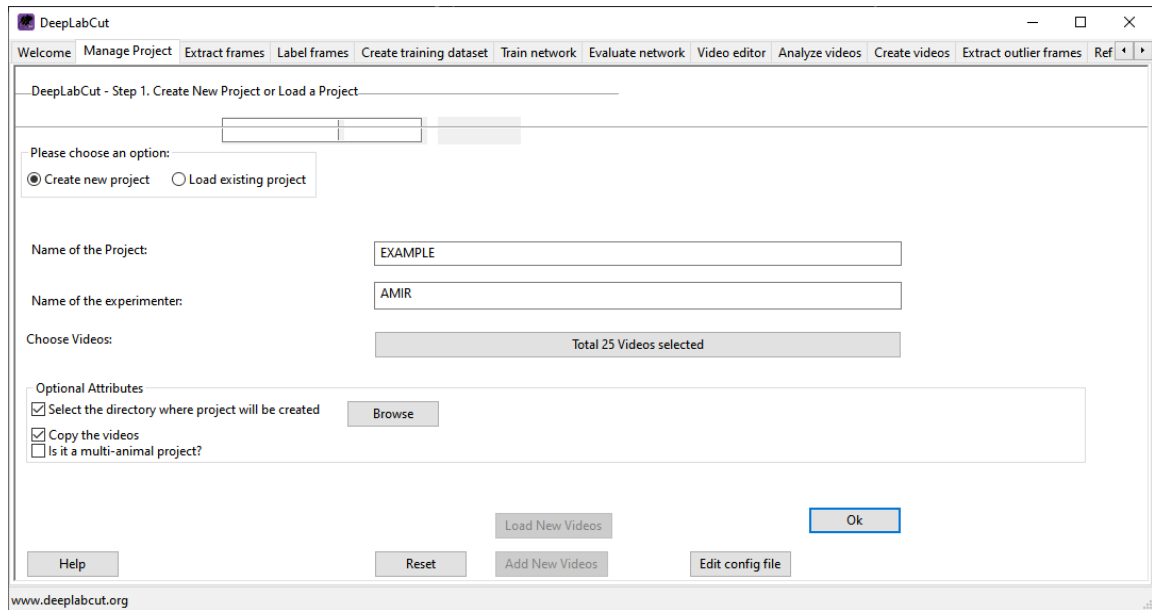


FIGURE 3.5 New project interface

It is also important to note that this is the point where the collected short videos of poultry are selected into DeepLabCut and from the example shown in Figure 3.5, 25 videos were loaded onto DLC. The checkbox for multi-animal detection is left unchecked as this project is only focusing on single animal detection and the box for copy video is checked for ease of use when inferencing. When the project is created successfully, the project file will contain four folders and one configuration file as shown in Figure 3.6. The `dlc-models` file in Figure 3.6 is where the trained network alongside the trained weights and parameters is saved, the `labelled-data` and the `training-datasets` file is where the extracted frame is saved and where the dataset is split into training and test dataset respectively. The project file will also copy the selected videos from the dataset into the project folder so that it is easier to access the videos later for the inferencing process. Lastly, the config file is where the crucial parameters related to the training process and the dataset label can be manipulated.

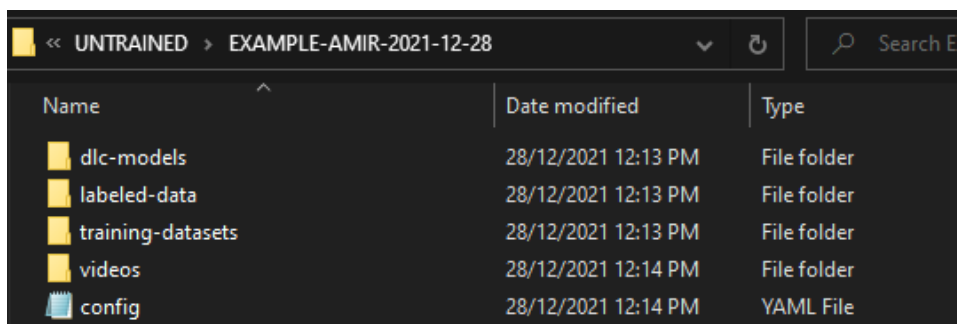


FIGURE 3.6 Newly created project folder

3.3.3.2 Labelling convention and tool

After the project is successfully created, a few parameters within the config.yaml file were changed to adapt for this project dataset. The config.yaml file contains and determines the different parameters for creating the training set file and evaluation results. A few parameters that were changed and that are relevant to the dataset creation are the numframes2pick, bodyparts, skeleton, and the description of these important parameters is shown in Table 3.1.

TABLE 3.1 Parameters¹ of the config file

Parameters	Description
numframes2pick	An integer that specifies the number of frames extracted from each video in the dataset. The default is 20.
bodyparts	A list containing the points to be tracked.
skeleton	The connection between each tracked body part.
colormap	The colour scheme of each tracked body part.
dotsize	The size of the marker when labelling. Default value is 12.
alphavalue	The transparency of the plotted labels. Default value is 0.5.
TrainingFraction	Two-digit floating-point number in the range [0-1], the ratio used to split the dataset into train and test. Default value is 0.95.
default_net_type	Specifies which pre-trained model to use. Default value is Resnet-50

¹ The complete list of parameters of the config.yaml file and its description is included in Appendix A.2.

The first important parameter that is changed is the `numframes2pick`, which specifies the number of frames to be extracted from each video in the dataset. As mentioned in the section about the data augmentation (section 3.3.2), the lower the number of frames used from each video while also maintaining the quantity of the total number of frames used for training will make the dataset more diverse which would help the deep neural network to recognize a larger pool of features. Hence, this parameter (`numframes2pick`) that was used in the series of the test presented in the next chapter is lower than 20 which is the default number set by DeepLabCut.

The next two related crucial parameters for the dataset preparation are the `bodyparts` and the `skeleton` parameters. The `bodyparts` parameter within the config file specifies the bodyparts that this project is tracking. This project is tracking five main body parts of poultry chicken which are the center body of the chicken, the head, the tail base, the left, and the right leg. If we were to compare the number of feature points tracked by Fang et al. [31] in their paper which is 10 (see Fig 2.3 from chapter 2), they are tracking double the feature points that are tracked by this project. Figure 3.7 is showing how the tracked body parts were labelled for this project.

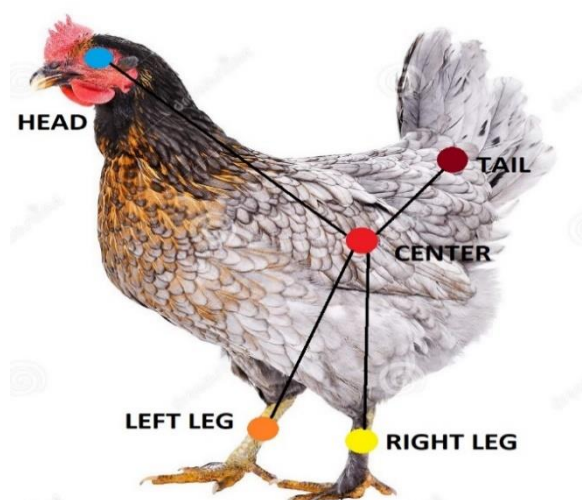


FIGURE 3.7 Labelling convention used for chicken

The reasoning behind choosing a significantly lesser number of feature points to track in this project is because this project has significantly less resource in terms of the available dataset and time constraints to properly hand-labelled those feature points for each extracted frame than compared to the resource available (especially in the number of personnel available to assist in hand-labelling the dataset) in the paper mentioned earlier [31]. The main risk if this project were to follow the paper mentioned earlier is that, without any access to an expert that can accurately identify the various body parts of poultry in a variety of backgrounds and postures, there is a risk of labelling inconsistency which may contribute to higher error in predicting the location of the chicken's body keypoints. Thus, this project is tracking only the body parts of the chicken that is easy to identify and label.

Table 3.2-3.3 is showing the differences between the default and the custom parameters that were used to replace the default parameters for both bodyparts and skeleton. The custom parameter for the bodyparts in Table 3.2 is how the body parts of the chicken that this project is trying to track were defined. The custom skeleton parameter shown in Table 3.3 is how the connection between the different body parts of the chicken was defined. The skeleton parameter is providing the deep neural network the information on how each of the tracked body parts is connected. Do note that both the dotvalue and alphavalue parameter were not changed and remained at the default value as stated in Table 3.1 earlier while the colormap parameter that was commonly used in this project is rainbow and plasma, the colormap parameter is the parameter that specifies the colour scheme of each label for each body parts (see Fig 3.8). However, for the TrainingFraction (a value that specifies the splitting ratio between the train and test dataset) and the default_net_type (the pretrained network that was used) parameter is dependant on the case of each test that was done and presented in chapter 4. For each series of tests, the TrainingFraction and the default_net_type parameter that was used will be specified and justified based on the different aims of each test.

TABLE 3.2 Parameter for bodyparts

Default bodyparts	Custom bodyparts
bodyparts: - bodypart1 - bodypart2 - bodypart3 - objectA	bodyparts: - center - head - tail - leftleg - rightleg

TABLE 3.3 Parameter for skeleton

Default skeleton	Custom bodyparts
skeleton: - - bodypart1 - bodypart2 - - objectA - bodypart3	skeleton: - - center - head - - center - tail - - center - leftleg - - center - rightleg



FIGURE 3.8 Colormap options

To prepare the dataset, DeepLabCut will extract a certain number of frames from each video in the dataset based on the numframes2pick that was specified within the Config file and labelled individually using the DeepLabCut labelling GUI. The labelling interface that was used for this project is shown in Figure 3.9. To label a frame, the cursor is placed at the point of the chicken's body that is intended to be marked followed with a right click of mouse, this will place the cursor at the intended point and to adjust to position of the marker, the marked point can be drag by holding

the right click of the mouse onto the desired point. The sequence of labelling the chicken's body parts is following the custom bodyparts parameter that is stated in the Config file (see Tab 3.2).

After finishing the labelling process on one frame, the same process is repeated for the remaining extracted frames by clicking the next button shown in the GUI (see Fig 3.9) and when frames from one video are done the process is to move onto the next frames of another video until all the frames from all the loaded training video are done being labelled. From the labelling GUI, there are a few tools and features that were used to assist the labelling process such as the zoom and pan feature which is useful when trying to be very accurate when placing the label onto the chicken's body. When the labelling process is done, the folder within the project file (as in Fig. 3.6) named labeled-data will contain multiple folders according to the number of videos used and within the video folder, it will contain the extracted frames in .png format, the coordinate of the labelled marker in an excel sheet and a .h5 file which is a Hierarchical Data Format (HDF) which contain multidimensional arrays data on the labelled markers (see Fig. 3.10 for of example of the labelled dataset folder). Finally, the entirety of the project folder is exported onto Google Drive to work with Google Colab.

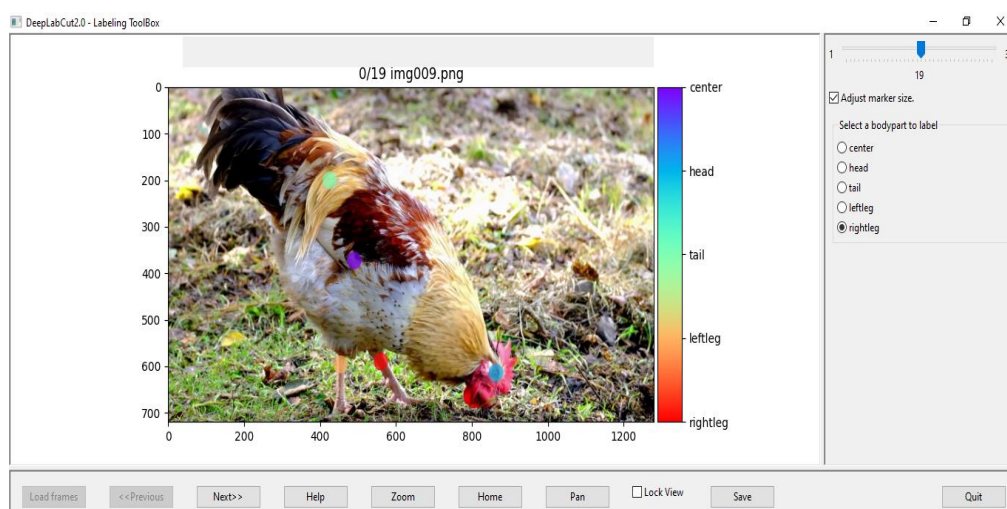


FIGURE 3.9 DeepLabCut labelling interface

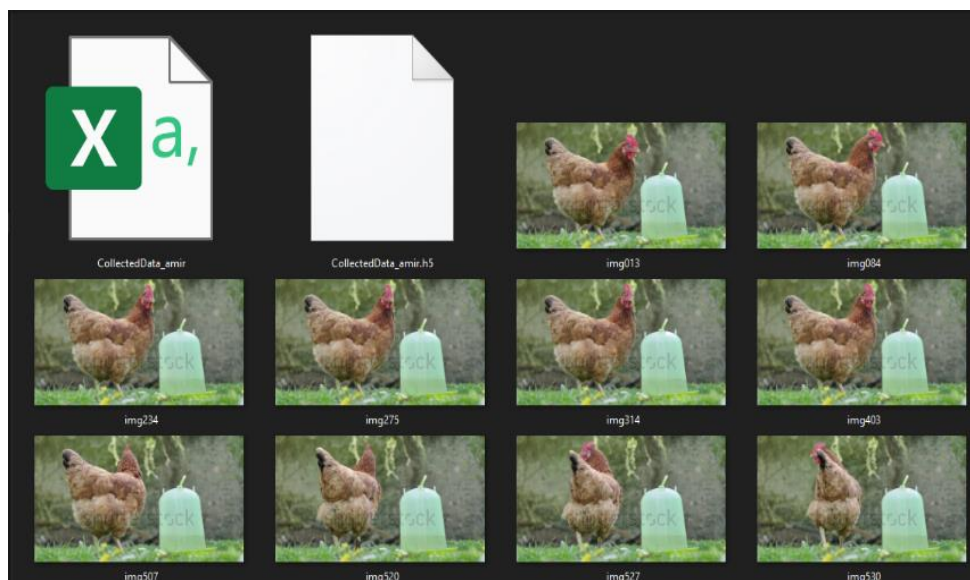
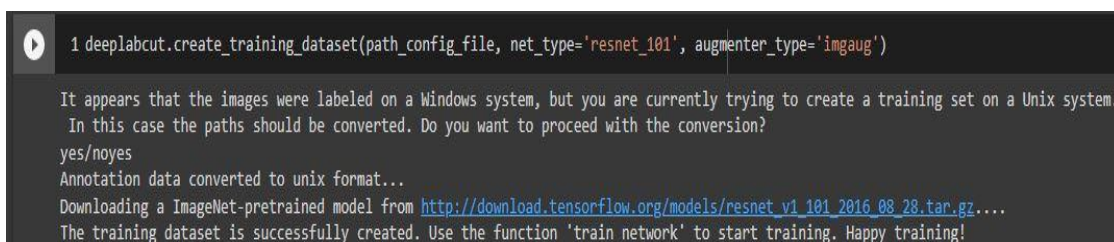


FIGURE 3.10 Contents of labelled-data folder

Before initiating the Google Colab notebook, a few Python's third-party libraries must be installed such as the DeepLabCut, TensorFlow, and OpenCV library followed by changing the runtime type to connect to Google's GPU and finally, the Google Colab must be connected and mounted to the Google Drive where the project folder (as in Fig. 3.6) is exported. To view the necessary installation, and the proper method to summon the project folder alongside the method to connect Colab with Google Drive, view this GitHub² repository created for this project to avoid cluttering this thesis with installation codes. After all, the needed library is installed and the project folder configuration is done, the DeepLabCut code shown in Figure 3.11 is where the labels from the extracted frames are merged into a single .h5 file, and do note that only videos that are included within the config.yaml (as in Fig. 3.6) is used to create this dataset, the code in Figure 3.11 is also where the frames dataset is split into train and test datasets, and the frames are chosen randomly based on the TrainingFraction parameters (as in Table 3.1) that are stated within the config.yaml

² The necessary installation and folder configuration: https://github.com/amrhkm/POULTRY-POSE-DEEPLABCUTV2/blob/main/startingwith_DeepLabCut.ipynb
The dataset is also available to be accessed by the public, see Appendix B.1

file. Using the code in Figure 3.11, conversion is also done to the dataset to make it compatible with a Unix system (Google Colab) and during this process, the pretrained network that is intended to be used (as an example in Fig 3.11, pretrained Resnet-101 is downloaded) is downloaded to prepare for the training process using the training dataset created earlier. To conclude about the dataset, the creation of this poultry keypoints dataset is to fulfil the first objective of this project, which is to prepare a dataset that is annotated with the poultry's body keypoints.



```

1 deeplabcut.create_training_dataset(path_config_file, net_type='resnet_101', augmenter_type='imgaug')
It appears that the images were labeled on a Windows system, but you are currently trying to create a training set on a Unix system.
In this case the paths should be converted. Do you want to proceed with the conversion?
yes/noyes
Annotation data converted to unix format...
Downloading a ImageNet-pretrained model from http://download.tensorflow.org/models/resnet\_v1\_101\_2016\_08\_28.tar.gz...
The training dataset is successfully created. Use the function 'train_network' to start training. Happy training!

```

FIGURE 3.11 DeepLabCut code for training dataset creation

3.4 Training the Deep Convolutional Network

3.4.1 Transfer Learning

Transfer learning is the main method that is adapted for this project, transfer learning is a deep learning method that is repurposing an already developed deep learning model (done by other researchers), created and trained to work on a specific task to work on a different set of tasks that then what it was intended to do (this project). This project can be classified as a computer vision task and to be able to develop a competent deep convolutional neural network from the ground up will require a lot of time resources, computing power, data, and most importantly the knowledge and skills needed to tune and modify those deep neural network layers.

Paper [38] stated that in deep learning, transfer learning only works if the model features learned in the first task are general and all of the supported models by DeepLabCut that is used in this project are all pretrained on a general dataset called ImageNet that contains over 14 million annotated image with 1000 different classes.

By training the supported model using the ImageNet dataset, the robustness of the supported models is increased and this is further supported in paper [39], which uses DeepLabCut to estimate the pose of horses. The paper mentioned earlier [39], has done a test that shows using DeepLabCut with a pretrained model is more robust and resulted in less error than the ones that are trained from scratch. In this project, three pretrained models or network that is supported by DeepLabCut is tested to find the suitable network that can detect keypoints on a chicken (results are presented in chapter 4, test 4.2).

3.4.2 Hyperparameters

Hyperparameters are configuration variables whose values will determine how the network is trained and these values are required for estimating the model (network) parameters. The prefix hyper is to show that they are the 'top-level' parameters that control the learning process and the model parameters that come from it. Hyperparameters are external to the model as the values are not saved within the model and these hyperparameters must be set manually before the training process, hyperparameters are independent of the dataset as the hyperparameters are not learned from the dataset. The chosen hyperparameters will determine the speed and the accuracy of the training process and to obtain the most suitable hyperparameters, the values are estimated through hyperparameters tuning and in this project, the hyperparameters are tuned through a series of trial-and-error tests (presented in chapter 4) and the default values of the hyperparameters that were set by DeepLabCut is used as a reference point on improving the model.

3.4.2.1 Learning rate

The learning rate is a hyperparameter that governs how quickly a model updates its parameters in response to the expected error. A very low learning rate may result in a long training process as it will need more training iterations which will also consume more memory capacity resources with a risk that the training process may become

stalled, whereas a value too large may result in learning a set of weights too fast which could cause an unstable training process [40]. Thus, finding a suitable learning rate is crucial and challenging. Figure 3.12 is presenting the different scenarios that are possible when configuring the learning rate parameter.

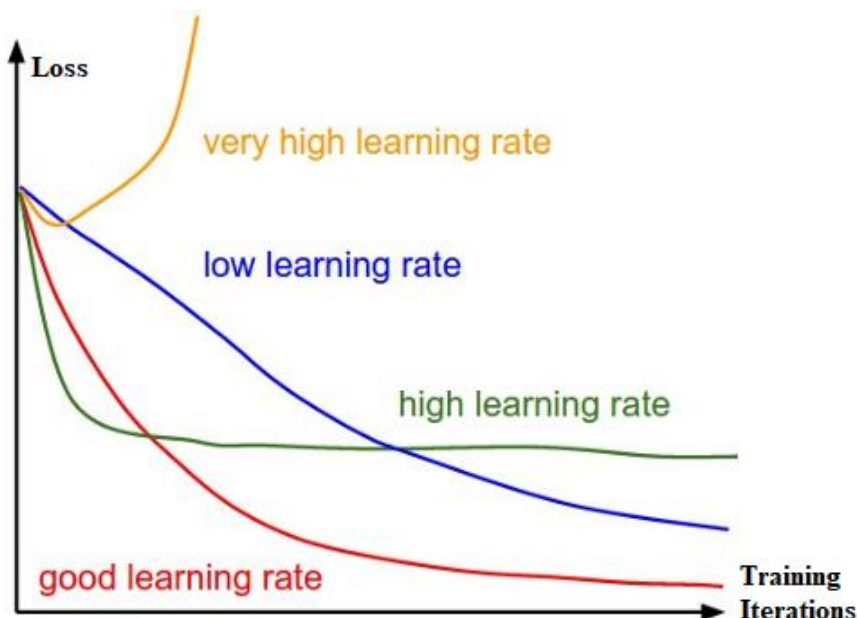


FIGURE 3.12 Various effects of learning rate on loss [41]

Within DeepLabCut, to alter the default pre-set learning rate, it must be done through a text file named `pose_cfg.yaml` located within the `train` subdirectory within the `dlc-models` directory of the project folder (as in Fig. 3.6). The `pose_cfg.yaml` file³ contains some parameters that control the training process of the network. Figure 3.13 is showing how the learning rate is stated within the `pose_cfg.yaml` file and Table 3.4 is showing the meaning of the learning rate parameter that is shown in Figure 3.13. From Table 3.4 the default learning rate set by DeepLabCut changes continuously based on the current training iterations until around one million training iterations. Hence, a custom learning rate is tested (results in chapter 4.4) to find the suitable learning rate based on the network and the training iterations used for this project.

³ The full available parameters within the `pose_cfg.yaml` file is shown in Appendix A.3

```

multi_step:
- [0.005, 10000]
- [0.02, 430000]
- [0.002, 730000]
- [0.001, 1030000]

```

FIGURE 3.13 Default learning rate within the pose_cfg.yaml file

TABLE 3.4 Default dynamic learning rate

Learning Rates	Training iterations range
0.005	0<iterations<10,000
0.02	10,001<iterations<430,000
0.002	430,001<iterations<730,000
0.001	730,001<iterations<1,030,000

3.4.2.2 Train-test split ratio

The train and test splitting ratio determines how much of the total dataset is used for training and testing and after splitting these datasets, there would be a training dataset and test dataset. The training dataset is fed into the pretrained model for the training process and this training process is done to fit the pretrained deep convolutional network model to the labelled chicken frames dataset. The test dataset is not used at all during the training process which would mean that the pretrained model never learns the features or the label from the frames that are in the test dataset.

The test dataset is only used after the training process has been completed and during this process, all the updated parameters (such as the weights and biases) that the network obtained from the training process is used to make keypoints prediction on the test dataset and the accuracy of the model on the test dataset would present a rough estimate on how accurate is the performance of the trained network/model against new and unseen data which could also mean how accurate the model is for a real-world environment implementation. In DeepLabCut the splitting ratio is

determined using the `TrainingFraction` parameter (as in Table 3.1) stated within the `config.yaml` file and the commands used within Google Colab to split the dataset are shown in Figure 3.11 earlier.

3.3.2.3 Training iteration

As mentioned earlier, this project is using transfer learning to make use of other researchers' trained networks and adapt their networks to fit the objectives of this project. Hence, the training process in this project is referring to the process where the training dataset that is specific to this project is fed into the deep convolutional network and during this process, the network will observe the labels and features present in the training dataset and use the observed features to update its pretrained parameters such as their weights and biases to fit the task of this project. In layman's terms, training is a process of finding the suitable weights and biases that are good for the task.

Weights and biases are called deep neural parameters which are different from hyperparameters. Parameters are internal to the network and the value of these parameters is not set manually and unlike hyperparameters whose value will stay the same throughout the entire training process, parameters are learned and are constantly updated during the training process through each training iteration. The parameters, unlike the hyperparameters they are dependent on the dataset, and the final value of these parameters will determine the performance of this network on unseen data (test dataset or real-world application).

In DeepLabCut, the term one training iteration would mean a cycle where the network has seen (learn its features and label) all the frames within the training dataset once. The number of iterations said by the developer of DeepLabCut where the loss will start to converge (when the loss has plateaued) is more than 200,000 to 400,000 iterations [42]. Hence, this project used the range of the number of iterations said by the developer of DeepLabCut where the loss will start to converge. To train the pretrained network with the labels in the training dataset, the command in Figure 3.14

was run on Google Collab and the number of iterations used is based on the aim of the test presented in chapter 4. The full available parameters for the codes in Figure 3.14 are available in this GitHub repository⁴ under the network training section. During the training process snapshots of the current iterations are automatically uploaded into the training subdirectory of dlc-models file (as in Fig 3.6).

```

1 #TRAIN THE PRETRAINED MODEL WITH THE LABELLED CHICKEN FRAMES DATASET
2 deeplabcut.train_network(path_config_file, shuffle=1, displayiters=100, saveiters=100)

iteration: 4000 loss: 0.0221 lr: 0.005
iteration: 4100 loss: 0.0219 lr: 0.005
iteration: 4200 loss: 0.0200 lr: 0.005
iteration: 4300 loss: 0.0207 lr: 0.005
iteration: 4400 loss: 0.0206 lr: 0.005
iteration: 4500 loss: 0.0237 lr: 0.005

```

FIGURE 3.14 DeepLabCut code for network training

These snapshots contain the current state of the trained network parameters at a particular training iteration (see Fig 3.15 for the example of the snapshot file at training iteration 300,100) and because this project was saved in Google Drive that has a limit to the number of memories it can store, only the 10 most recent training iterations snapshots are saved within the dlc-models folder and the rest is transferred to the bin. Some of the useful snapshots that is in the bin were restored to project folder for the evaluation process. These snapshots are crucial for evaluating the network performance at a certain training iteration (as the test presented in chapter 4).

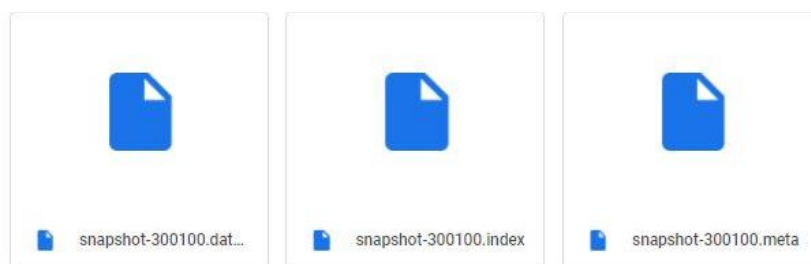


FIGURE 3.15 Example of the snapshot file

⁴ Parameters of DLC training code <https://github.com/DeepLabCut/DeepLabCut/wiki/DOCSTRINGS#>

3.5 Network Performance

3.5.1 Evaluation and the Metrics Used

3.5.1.1 Pixel error and confidence level

For object classification task using any variants of the neural networks (NN), the most common metrics used to evaluate such network is accuracy and error and this easier to do for a classification process because the ground truth for each object that the neural network is tasked to identify is available [43]. However, for a keypoint detection task on a video, such ground truth is hard to provide, and the only ground truth that this project is able to provide is the individually manual label that is plotted onto the chicken's body on the extracted frames dataset. Hence, Using DeepLabCut, the main way that the trained network was evaluated by computing its mean average Euclidean error (MAE) or simply put, the average distances in pixels between the manual label (the ground truth) that is done during the labelling process in chapter 3.3.3 and the label predicted by the trained network [42]. This average distance between the manual and predicted labels are called pixel error and this evaluation process was done to both train and test dataset and therefore, there will be a pixel train error and pixel test error for a network.

Figure 3.16-3.17 is showing the DeepLabCut code that was used on Google Collab to evaluate the trained network and the output of the code. If the plotting parameter were set to true as in Figure 3.16, it will plot the network predicted label against the manual label as in Figure 3.18 where the human label is plotted as (+), and the network predicted label is plotted as a dot. Notice that in the output of the evaluation code there is a parameter call p-cutoff, this parameter is called a confidence level threshold and is used to increase the accuracy during inferencing and will be discussed in detail in chapter 3.6.1.

```
deeplabcut.evaluate_network(config_path, plotting=True)
```

FIGURE 3.16 DeepLabCut network evaluation code

```

256it [00:42, 5.99it/s]Analysis is done and the results are stored (see evaluation-results) for snapshot: snapshot-300000
Results for 300000 training iterations: 80 1 train error: 2.77 pixels. Test error: 11.28 pixels.
With pcutoff of 0.6 train error: 2.77 pixels. Test error: 7.69 pixels
Thereby, the errors are given by the average distances between the labels by DLC and the scorer.
Plotting...

```

FIGURE 3.17 Output of DeepLabCut network evaluation code

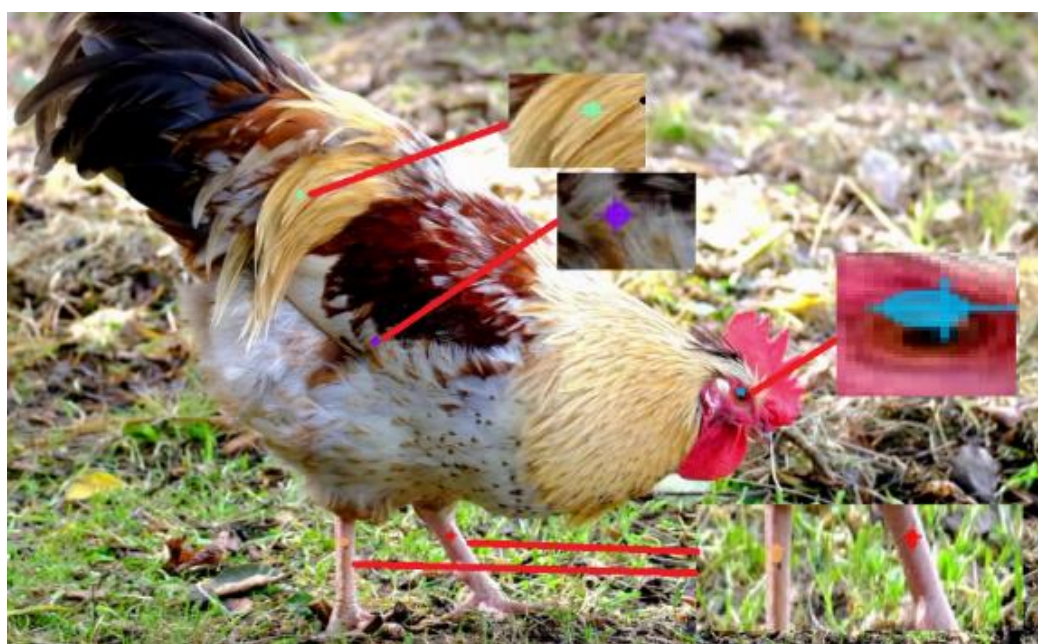


FIGURE 3.18 Human labels plots against network predicted plots

3.5.1.2 Training loss

The training loss in DeepLabCut is using the standard cross-entropy loss function shown in equation 1 of Figure 3.19 to predict the pixel-wise class probability and the location refinement error (a form of location refinement for the final coordinate) via Huber loss with a weight of 0.05 loss was used to regress predicted location to actual location using equation 2 in Figure 3.19 with ($\delta = 1$) [43]. These losses were combined and optimized by Stochastic Gradient Descent (SGD) to obtain the tuned parameters for a network that is suitable for a chicken's body keypoint detection task.

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$$

Equation 1: Cross Entropy Loss

$$L_{\delta}(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 & \text{otherwise.} \end{cases}$$

Equation 2: Huber Loss for Regression: note $f(x)$ denotes predicted values

FIGURE 3.19 Training loss equations [43]

3.6 Keypoints Prediction on a Video

This section will go into detail on the DeepLabCut codes and how DeepLabCut was used to do the chicken's body keypoints prediction for a video of a chicken (will work with a video that is within the dataset or not) using the trained network. Using a trained network to do prediction on a video that is not within the dataset (unseen data) is called inferencing. Ideally, this project aimed to achieve a network that is good at inferencing because if a network is good at inferencing this would make the network even more suitable for a real-world environment which would add more value to this study.

3.6.1 Confident level or likelihood

As mentioned in chapter 3.5.1.1, the main metrics used to evaluate the trained network is the pixel error or the MAE, this MAE is saved alongside a parameter called a confidence level or likelihood in a comma-separated file. The confidence level is a positive integer value that is stating the level of confidence that network had in making a particular keypoint prediction for each of the tracked chicken's body parts. The range of this confidence level value is starting from zero which is a very low confident prediction to one which is a very high confident prediction. Using this confidence level, DeepLabCut is showing its strength if the network is sufficiently trained, it will have a probabilistic output of the scoremap that can be used as guide to reliably report when a body part is visible within the frame [42].

From chapter 3.5.1.1 and Figure 3.17, there is parameter called p-cutoff which is a confidence level threshold and by default this value is set to 0.1, this p-cutoff parameter is available within the config.yaml file (see appendix A.2). This parameter is used to increase the accuracy of the network by reducing the keypoint prediction that has a low confident value. The way the p-cutoff parameter work is that if a prediction has a confidence level higher than the P-cutoff ($\text{confident} > \text{p-cutoff}$) that would mean that it is a confident prediction. However, if a prediction confidence level is lower than the P-cutoff ($\text{confident} < \text{p-cutoff}$), that would mean that the prediction has a low level of confidence. Hence, if a keypoint prediction has a confident level below the p-cutoff, the network will eliminate the prediction thus reducing the pixel error and during keypoint prediction on a video, the prediction with a confident level below the p-cutoff will not be shown. Using the network evaluation code in Figure 3.17 we could see the differences between the network prediction that has a high confident level and low confident level against the human label (see Fig. 3.20-3.21). From Figure 3.20-3.21, the human label is in the plotted as (+) and the predicted label that has high confident ($\text{confident} > \text{P-cutoff}$) are plotted as a dot (.) and the prediction label that has lower confident level ($\text{confident} < \text{p-cutoff}$) are plotted as an (X).

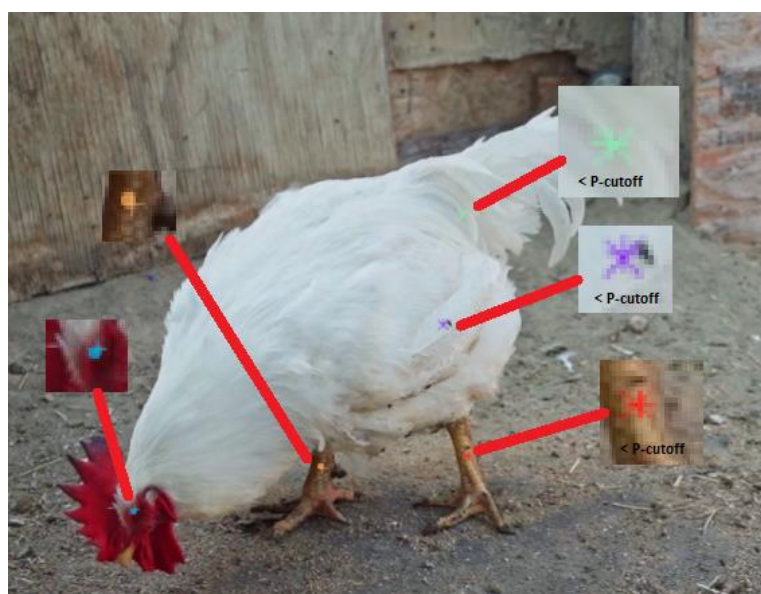


FIGURE 3.20 Human label against low confident network prediction

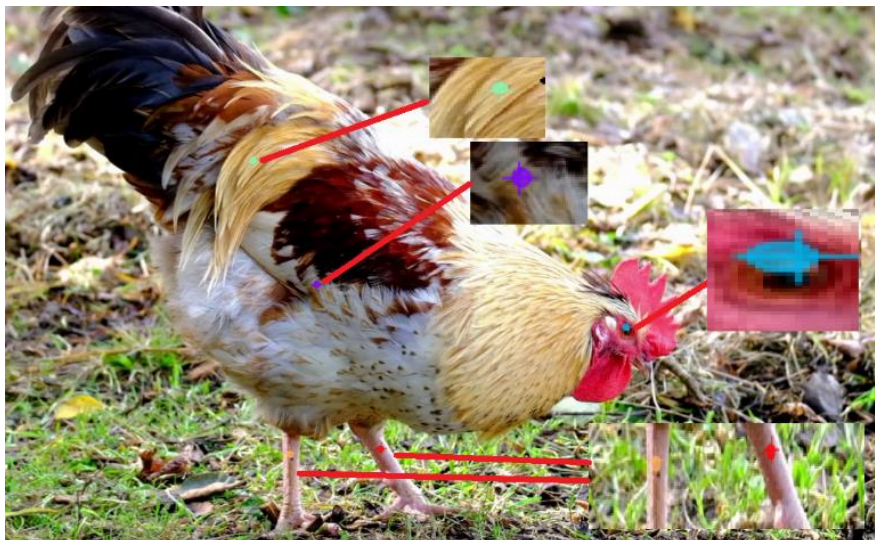


FIGURE 3.21 Human label against high confident network prediction

3.6.2 Create Predicted Labelled Video

The code shown in Figure 3.22 was used to make keypoint predictions using the trained network. The output of the code from Figure 3.22 is saved within the videos file of the project folder (see Fig. 3.6) and the output is stored as a Multi-index Pandas array that is stored in an efficient Hierarchical Data Format (HDF) which contains the name of the network used, names of each tracked body parts, (x,y) label position in pixels, and the confidence level for each frame per body part.

```
deeplabcut.analyze_videos(path_config_file, videofile_path, videotype=mp4)
```

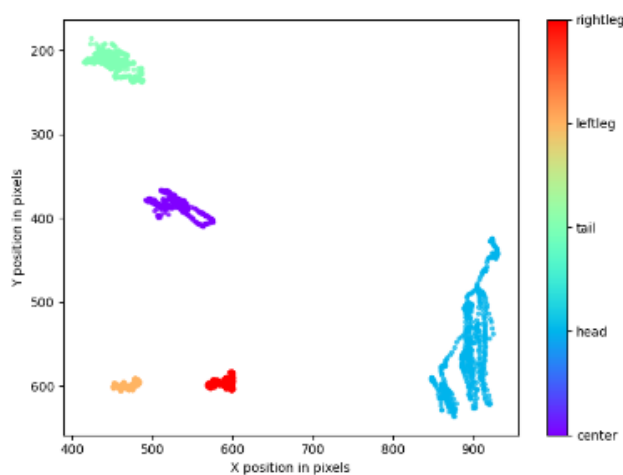
FIGURE 3.22 DeepLabCut code making keypoint prediction

The code in Figure 3.23 is used to analyse the trajectories of the predicted keypoints and the output of this line of codes is saved at plot-trajectories within the videos folder. Using this line of code, the movement of the predicted keypoints throughout the whole duration of the video is plotted onto an XY-axis (as in Fig. 3.24 (a)) alongside the confidence level/likelihood of each body parts throughout the entire

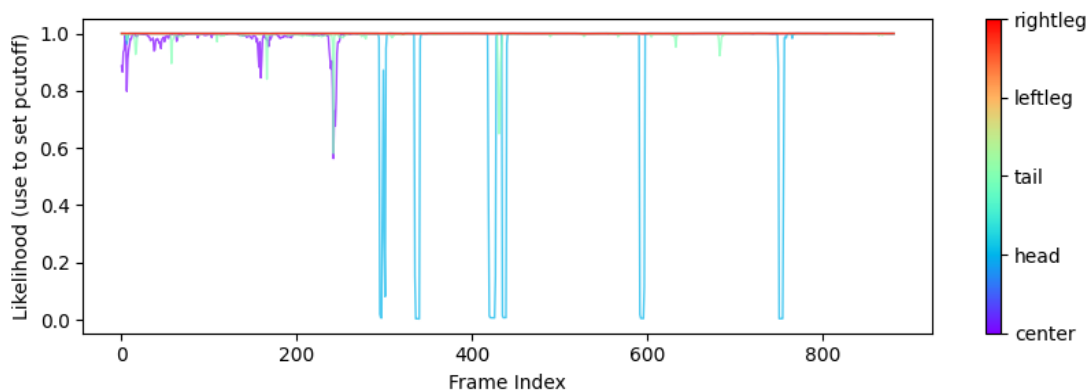
video timeframe (as in Fig. 3.24 (b)). Finally, to create a video using the keypoints that is predicted by the trained network, the code in Figure 3.25 was used and an example of a frame of the output video is shown in Figure 3.26.

```
deeplabcut.plot_trajectories(path_config_file, videofile_path, videotype=mp4)
```

FIGURE 3.23 DeepLabCut code for plotting keypoints trajectories



(a) Keypoint movements across frames



(b) Confidence level across the frames

FIGURE 3.24 Trajectories plots

```
deeplabcut.create_labeled_video(path_config_file, videofile_path, videotype=mp)
```

FIGURE 3.25 DeepLabCut code for creating a video with predicted keypoints



FIGURE 3.26 Output video using the trained network

3.7 Flowchart

The flowchart below acts as a summary of the entire methodology chapter of this project.

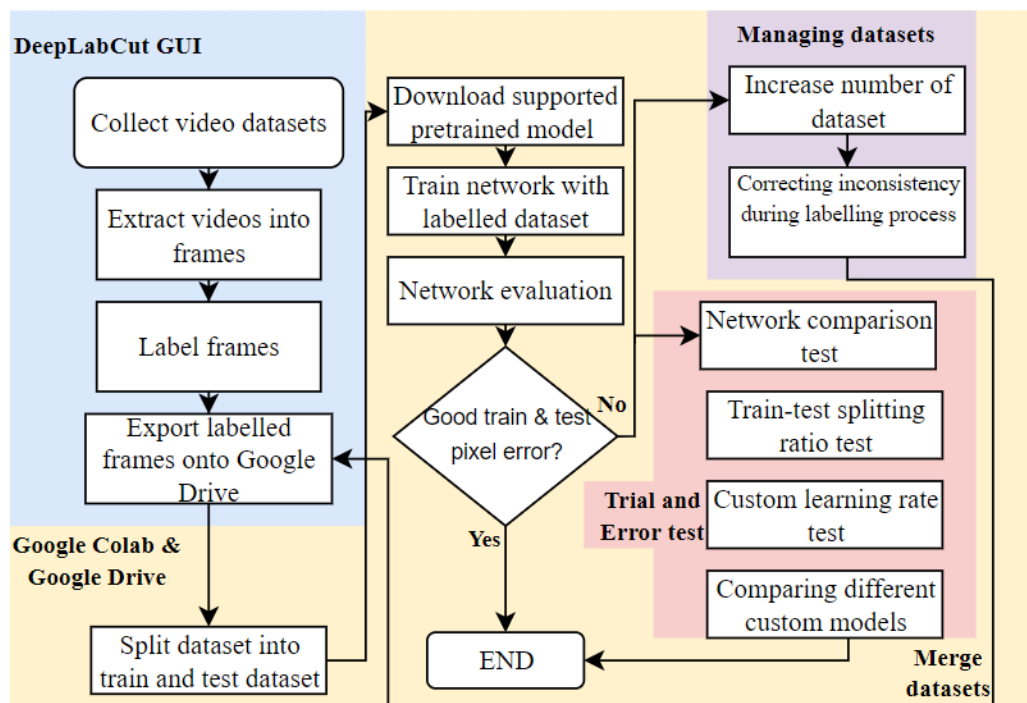


FIGURE 3.27 Project flowchart

CHAPTER 4

OBSERVATION AND RESULTS

4.1 Overview

This chapter comprises the results achieved from carrying out a series of tests along with a detailed explanation for each case hyperparameter that was changed. This chapter aims to find the best hyperparameters (as discussed in chapter 3.4.2) and to analyze and evaluate the performance of the trained network under certain hyperparameters. This chapter will make use of the metrics and methods explained in chapter 3.5.

4.2 Performance of Resnet-50, Resnet-101, and Mobilenet V2

Resnet-50, Resnet-101, and Mobilenet V2 is the pretrained network supported by DeepLabCut. All three of the tested networks have a different architecture which could be an advantage or disadvantage to the type of dataset this project is using. Hence, this test aims to find the best-supported network that is suitable for the chicken's keypoint detection task of this project. To find the best performing network, a few hyperparameters were set constant throughout all three networks.

Below are the hyperparameters that were constant across all three tested networks and the result for this test is presented in Table 4.2-4.4. All three of the tested networks were evaluated at three different training iterations which are at 100,000, 200,000, and 300,000 iterations. Used 80% of the frame dataset for training and 20% for testing from a total frame of 256 frames. Used Stochastic Gradient Descent (SGD) as default network optimizer that was set by DeepLabCut. Finally, this test used the default dynamic learning rate set by DeepLabCut (see Table 4.1).

TABLE 4.1 Default dynamic learning rate

Learning Rates	Training iterations range
0.005	0<iterations<10,000
0.02	10,001<iterations<430,000
0.002	430,001<iterations<730,000
0.001	730,001<iterations<1,030,000

TABLE 4.2 Performance at 100,000 training iterations

Network	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Resnet-50	0.0049	4.19	16.8	12.61
Resnet-101	0.0041	7.56	16.77	9.21
Mobilenet V2	0.0045	8.22	32.54	24.32

TABLE 4.3 Performance at 200,000 training iterations

Network	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Resnet-50	0.0038	3.70	11.77	8.07
Resnet-101	0.0029	3.22	11.62	8.40
Mobilenet V2	0.0032	5.10	26.44	21.34

TABLE 4.4 Performance at 300,000 training iterations

Network	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Resnet-50	0.0031	3.4	12.72	9.32
Resnet-101	0.0028	2.92	10.52	7.60
Mobilenet V2	0.0025	3.82	18.35	14.53

4.3 Testing Different Training and Test Dataset Split Ratio

The test in this section was done using the Resnet-101 (see Chapter 5.2 for the reasoning behind this choice) network under the default dynamic learning rate (as in Table 4.1) and the default network optimizer (SGD) set by DeepLabCut to find the best train and testing dataset split ratio that can be used on the final recommended Model, the number of frames used for each splitting ration is shown in Table 4.5. The network was evaluated at three different training iterations to obtain a better correlation between the pixel error and the dataset splitting ratio over the increasing number of training iterations. Table 4.6-4.8 shows the results of this test.

TABLE 4.5 Frames used for each splitting ratio

Training: Test	Training frames	Test frames	Total frames
95: 5	243	13	256
80: 20	205	51	256
70: 30	179	77	256

TABLE 4.6 Performance at 100,000 iterations

Training: Test	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
95: 5	0.0042	7.38	15.34	7.96
80: 20	0.0041	7.56	16.77	9.21
70: 30	0.0034	4.06	11.94	7.88

TABLE 4.7 Performance at 200,000 iterations

Training: Test	Training loss	Train pixel error (pixels)	Test Pixel error (pixels)	The discrepancy of Test-Train pixel error
95: 5	0.0030	3.93	16.88	12.95
80: 20	0.0029	3.22	11.62	8.40
70: 30	0.0027	3.02	10.10	7.08

TABLE 4.8 Performance at 300,000 iterations

Training: Test	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
95:5	0.0025	3.57	14.82	11.25
80:20	0.0028	2.92	10.52	7.60
70:30	0.0023	2.31	10.05	7.74

4.4 Network Performance of Default and Custom Learning Rate

This test aims to find the best custom learning rate that is suitable for the labelled chicken dataset that is used in this project and compare it with the default learning rate set by DeepLabCut using the best network, and the best train and test splitting ratio (70:30) obtained from the test 4.2 and 4.3 earlier. The default and custom learning rates used in this test are presented in Table 4.9-10 respectively and the results are presented in Table 4.11-13.

The custom learning rate that is used in this test is a slight modification of the default learning rate. From Table 4.9, we could see that the training iteration range for the default learning rate is larger as the range stopped at around one million iterations, and to cater to the learning rates with the number of training iterations used in this project, a modification of the default learning rate is done (as in Table 4.10). Due to the short range of training iterations being used in this test, the final learning rate in the last row of Table 4.10 was increased from 0.002, 0.001 (as in the last two rows of Table 4.9) to 0.005.

The reason that this project is not testing up to a million training iterations (as in the default learning rate in Table 4.9) is that to train a network for 300,000 iterations using DeepLabCut and Google Colab would take an average of 11 hours. Hence, training a network until a million iterations for each test would not be practical with the time constraints of this project and a million iterations is a relatively large number of training iterations compared to the size of the dataset that this project has (256 frames).

TABLE 4.9 Default dynamic learning rate

Learning Rates	Training iterations range
0.005	0<iterations<10,000
0.02	10,001<iterations<430,000
0.002	430,001<iterations<730,000
0.001	730,001<iterations<1,030,000

TABLE 4.10 Custom dynamic learning rate

Learning Rates	Training iterations range
0.005	0<iterations<10,000
0.02	10,001<iterations<214,400
0.005	214,401<iterations<300,000

TABLE 4.11 Performance at 100,000 iterations

Learning rate type	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Default	0.0034	4.06	11.94	7.88
Custom	0.0037	3.00	9.38	6.38

TABLE 4.12 Performance at 200,000 iterations

Learning rate type	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Default	0.0027	3.02	10.10	7.08
Custom	0.0025	2.92	9.02	6.1

TABLE 4.13 Performance at 300,000 iterations

Learning rate type	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
Default	0.0023	2.31	10.05	7.74
Custom	0.0021	2.74	9.61	6.87

4.5 Custom Model Comparison

In this test, two custom models each using different custom hyperparameters were tested. This test aims to find the best custom model that this study can produce. The first custom model is the same custom model used in the previous test 4.4 and for this test, it is called custom model 1 (CM1). The second custom model used here is called custom model 2 (CM2) and it almost uses the same hyperparameters as CM1 with the major difference is that CM2 is using Resnet-50 instead of Resnet-101. Table 4.14 is summarising the differences in characteristics of CM1 and CM2. Both custom models were evaluated at 200,000 and 300,000 iterations. The result of this test is presented in Table 4.16-17.

TABLE 4.14 Characteristics of CM1 and CM2

Characteristics	CM1	CM2
Network	Resnet-101	Resnet-50
Learning rate	Custom learning rate (as in Table 4.15)	Custom learning rate (as in Table 4.15)
Optimizer	SGD (default)	SGD (Default)
Final training iterations	300,000	300,000
Training: Test	70:30	80:20

TABLE 4.15 Custom dynamic learning rate

Learning Rates	Training iterations range
0.005	0<iterations<10,000
0.02	10,001<iterations<214,400
0.005	214,401<iterations<300,000

TABLE 4.16 Performance at 200,000 iterations

Custom model	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
CM1	0.0025	2.92	9.02	6.1
CM2	0.0037	3.18	7.83	4.65

TABLE 4.17 Performance at 300,000 iterations

Custom model	Training loss	Train pixel error (pixels)	Test pixel error (pixels)	The discrepancy of Test-Train pixel error
CM1	0.0021	2.74	9.61	7.74
CM2	0.0023	2.77	7.69	4.92

CHAPTER 5

ANALYSIS AND DISCUSSION

5.1 Important Concept for Discussion

5.1.1 *Overfitting*

Overfitting is a concept in data science and overfitting occurs when the trained network has a significantly better accuracy at predicting on the training dataset (the dataset that is used to teach the network) and very low accuracy on the test dataset (unseen data). Overfitting will cause the network to be bad at generalization on new unseen data which is an important factor to consider if one were to deploy the network for real-world environment use. Thus, overfitting in this discussion can also be said as having very a high discrepancy between the test and train pixel error.

5.1.2 *Pixel Error Categorization*

Pixel error is the average distance between the hand-labelled (manual label) and the label predicted by the trained network. Thus, a good and accurate keypoint prediction network would have a low average distance between the human label and the predicted label and can also be said as having low pixel error. As of the time that this thesis is written, no formal documentation can be found on the range of pixel error that can be considered as good or bad. However, from experience and observation of the inferencing process (the video output with the predicted keypoints) using any of the trained networks in this project and from reading online forums from GitHub and Stack Overflow, the author would describe the characteristic of the network prediction based on pixel error range in Table 5.1. The first network used the default learning rate (refer Table

TABLE 5.1 Pixel error category

Pixel error range	Categorization	Description
Below 3.7	Good	Would accurately detect all keypoints and no noticeable jitter (repeatedly showing and not showing the keypoint predictions)
Between 3.7 and 5.0	Mediocre	Slightly less accurate prediction but with noticeable jitter
Above 5.0	Bad	Incorrect prediction or not showing any prediction at all

5.2 Best Network Supported by DeepLabCut

From test 4.2 (see Table 4.2-4), we could see Resnet-101 has the lowest train and test pixel error for all the training iterations that it was evaluated on (see Fig 5.1-2). This would mean under the default hyperparameters presented in the second paragraph of chapter 4.2, Resnet-101 has the best performance. Even though after 300,000 iterations, Resnet-101 managed to achieve 2.92 for test and 10.52 for train pixel error which is the lowest pixel error out of the three networks that were tested, this does not that the pixel error that Resnet-101 managed to obtain in this test is good.

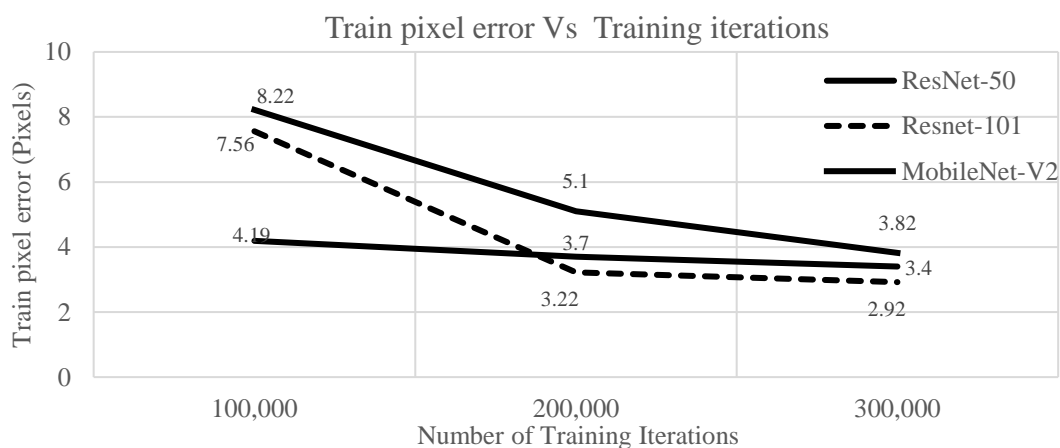


FIGURE 5.1 Train pixel error across different network

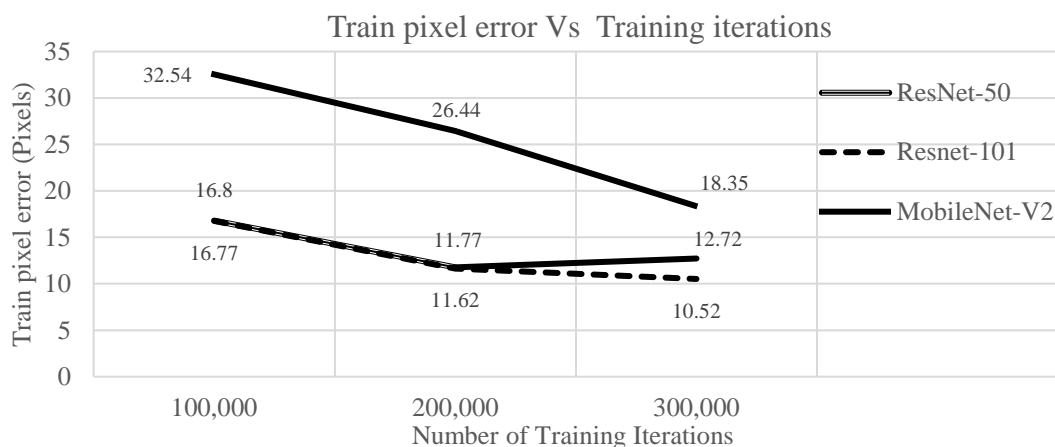


FIGURE 5.2 Test pixel error across different network

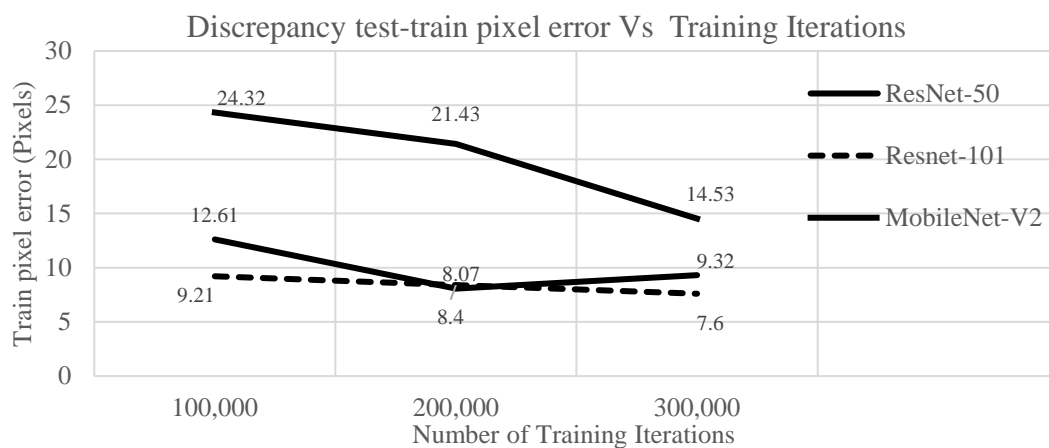


FIGURE 5.3 Discrepancy test-train across different network

Using the pixel error categorization method shown in Table 5.1 earlier, we could see that in test 4.2, Resnet-101 managed to achieve a train pixel error of 2.92 (see Fig. 5.1) after 300,000 training iterations which this value can be considered as a good pixel error. However, for test pixel error Resnet-101 only managed to achieve a pixel error of 10.52 (see Fig. 5.2) and based on the categorization method mentioned earlier this is a bad pixel error. Since the train pixel error is significantly better than the test pixel error, it can be deduced that under the default hyperparameters mentioned in chapter 4.2, Resnet-101 overfits. However, the main reason Resnet-101

was chosen to be the best network out of Resnet-50 and Mobilenet V2 is that it overfits less than all the other tested networks and this can be seen by Resnet-101 being able to achieve a 7.6 discrepancy between the test and train pixel error after 300,000 training iterations which are well below than the discrepancy achieved by the other networks (see Fig. 5.3). Hence, the tests in chapters 4.3 and 4.4 is using Resnet-101 to find the best hyperparameters based on the aim of each corresponding test.

5.3 Best Train and Test Splitting Ratio

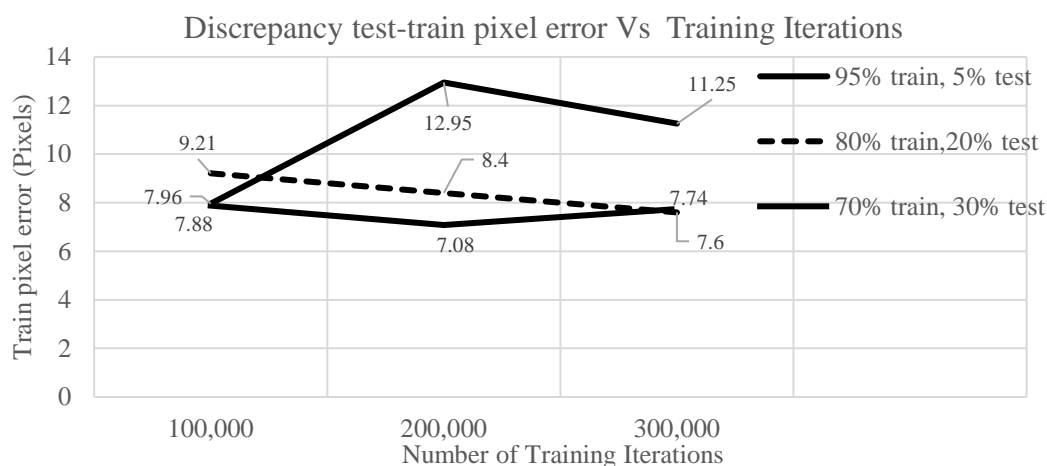


FIGURE 5.4 Discrepancy test-train across the different splitting ratio

This section of the discussion will discuss the results presented in chapter 4.3. The test in chapter 4.3 aimed to find the best training and test dataset splitting ratio based on their corresponding performance using the hyperparameters presented in chapter 4.3. Figure 5.4 is showing the discrepancy trend between the pixel error of the test and train dataset. From Figure 5.4, when using the 95:5 splitting ratio, the discrepancy value at 200,000 iterations almost doubled from 100,000 which is from 7.96 to 12.95. This would suggest that using 95% of the dataset for training would result in much faster overfitting and this implication is further justified by the discrepancy value that the 95:5 ratio arrived at after 300,000 iterations (11.25) which is double the discrepancy of other splitting ratios (also at 300,000 iterations) that was tested.

The network that used the 80:20 and 70:30 splitting ratio can be said as the least overfitted network. However, when comparing the average discrepancy⁵ between 70:30 and 80:20, 70:30 has a much lower average discrepancy of 7.57 than the average discrepancy of 80:20 which is 8.40. Thus, it could be said that the network that used the 70:30 ratio has a lower average discrepancy which resulted in a performance that overfits less than other networks that used the 80:20 and 95:5 ratio. From this test also, it could be concluded that using a large network such as Resnet-101 alongside a high amount of the dataset for training would make the network itself overfits very early on in the training process. Since using the 70:30 split ratio managed to get the best result, the next test presented in chapter 4.4 is using the 70:30 split ratio to compare the performance of a network that used custom learning rate and the network that used DeepLabCut default learning rate. From Table

5.4 Custom Learning Rate Performance

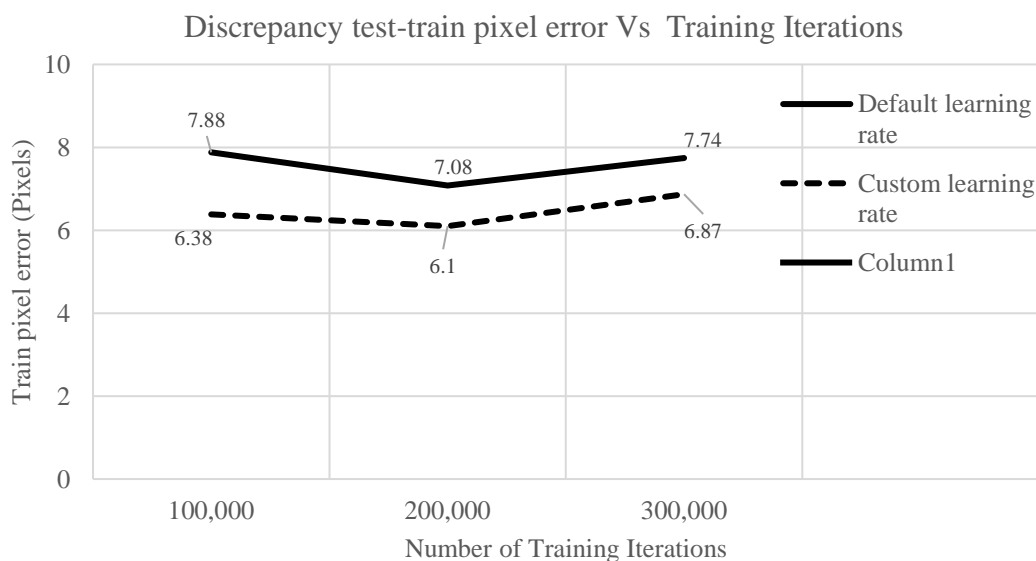


FIGURE 5.5 Discrepancy test-train across different learning rates

⁵Average discrepancy is total of the discrepancy (from each iteration) and divide by three.

Average discrepancy (70:30) = $(7.88+7.08+7.74)/3 = 7.57$

Average discrepancy (80:20) = $(9.21+8.40+7.60)/3 = 8.40$

Average discrepancy (95:5) = $(7.96+12.95+11.25)/3 = 10.72$

This section of the discussion will discuss the result presented in chapter 4.4 which is a test to compare and observe the effect of using custom and default learning rates on the overfitting situation that occurs in the first two tests that were presented in chapter 4. In the test presented in chapter 4.4 two network/models were tested under the same hyperparameters (presented in chapter 4.4) with the exemption of their learning rate. To see the default and custom learning, refer to Table 4.9-10 within chapter 4.4. From Figure 5.5, both networks had the smallest discrepancy between the pixel error of the test and train dataset during the 200,000 iterations. Figure 5.5 also shows that when both networks arrived at the 300,000-training iteration, the overfitting situation for both networks is worsened, and this showed by a slight increase in the discrepancy.

From Table 4.13, both networks managed to achieve a train pixel error that can be categorized as good (based on the categorization method in Table 5.1). However, both networks perform poorly on the test dataset as both networks obtain a test pixel error that is categorized as bad. This would suggest that while using the custom learning rate as in Table 4.10, may improve on the overfitting problem, the test pixel error is still large and cannot be considered as good. Based on test 4.4, it can be concluded that the custom learning has resulted in better results than the default learning rate for the kind of dataset that Resnet-101 is trained with. However, the peak performance of both (where it overfits less) networks is when the training iterations is less than 300,000.

5.5 Best Performing Custom Model

In the last test done in chapter 4.5, the network comparison was done between two custom models. The first model named custom model 1 (CM1) uses the same model that uses the custom learning rate in chapter test 4.4. The second custom model named (CM2) has a slight difference in that it used Resnet-50 as the network, and it also used 80:20 training and test split ratio. The full characteristics of CM1 and CM2 alongside the used hyperparameters are available in Table 4.14. From the first test (chapter 4.2) and by observing the trend in Figure 5.1-3, Resnet-50 and Resnet-101 has an almost

similar performance in terms of their train and test pixel error and the test-train discrepancy. However, during the first test Resnet-101 was deemed as the best network due to the lowest value of test-train discrepancy which is an indicator that Resnet-101 overfits less than Resnet-50.

TABLE 5.2 Architecture of different Resnet variants [44]

layer name	34-layer	50-layer	101-layer
conv1	$7 \times 7, 64$, stride 2		
	3×3 max pool, stride 2		
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$
conv5_x	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	average pool, 2048-d fc		

In the second and the third test, it has been shown that training Resnet-101 using a 70:30 dataset splitting ratio and using the custom learning rate (as in Table 4.10) has managed to lower the discrepancy between the test and train pixel error. However, the main problem that persisted with Resnet-101 throughout all the tests is that its test pixel error or the performance of Resnet-101 on new unseen data has been bad with the lower test pixel error for a model that used Resnet-101 being 9.02. Hence, in the last test Resnet-50 was chosen to be the comparison network to use against Resnet-101 because Resnet-50 has an almost similar architecture but with a much simpler architecture than Resnet-101. The complex network has a higher capacity to process data and as claimed by Jason. B [45], a model can overfit because the network has the capacity to do so. Table 5.2 is showing the difference in the architecture of three variants of ResNet.

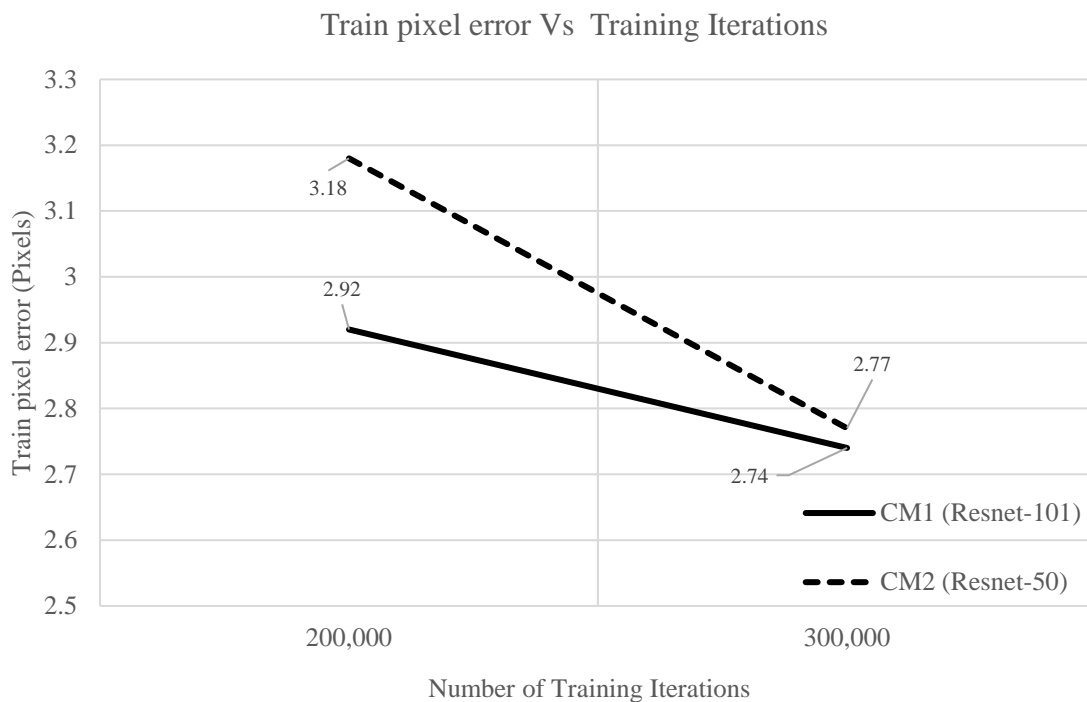


FIGURE 5.6 Train pixel error across different custom models

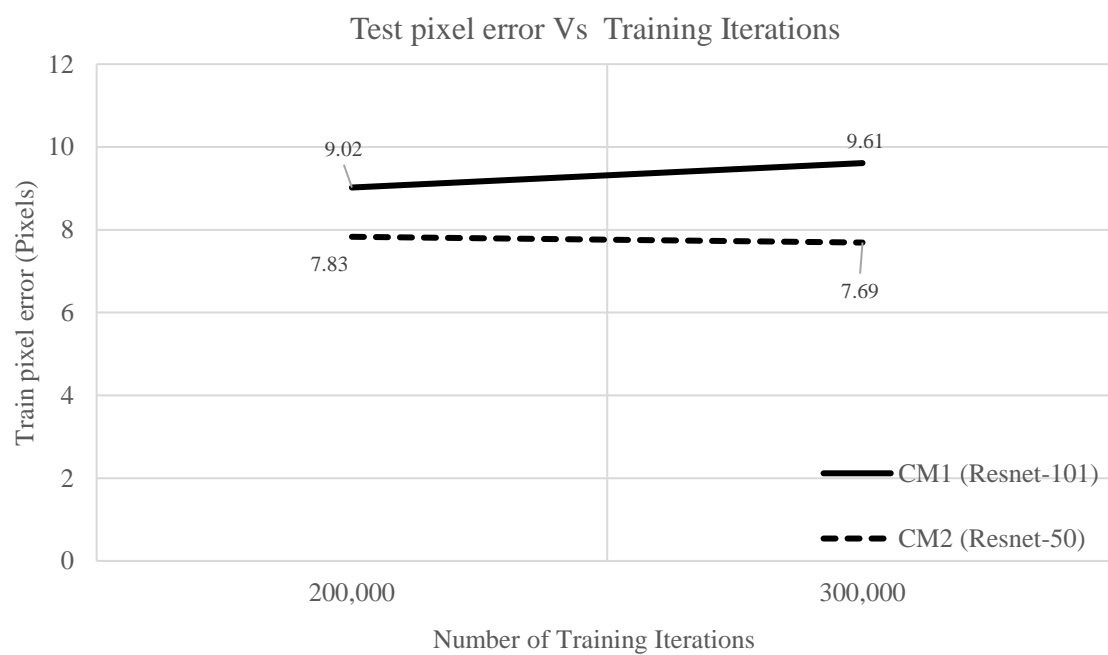


FIGURE 5.7 Test pixel error across different custom models

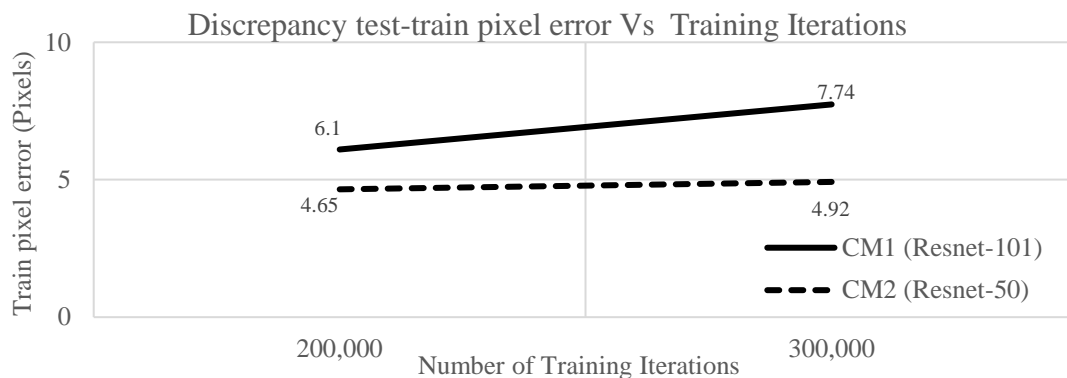


FIGURE 5.8 Discrepancy test-train across different custom models

Based on the results of the last test (see Fig. 5.6), all of the train pixel errors for both CM1 and CM2 can be considered as good. For the test pixel error, both networks obtained a pixel error that is categorized as bad however after 300,00 iterations CM2 managed to achieve the lowest test pixel error from all the test that was conducted (see Fig. 5.7). Based on Figure 5.8, not only that CM2 manage to achieve the lowest test pixel error in this project when moving from 200,000 to 300,000 training iterations CM2's already low discrepancy value of test-train pixel error only increased by 0.27. On the other hand, CM1 already has a higher discrepancy value at 200,000 iterations, and when assessed at 300,000 that value increased by 1.64.

This is confirming two claims, the first claim made in discussion chapter 5.4 which says that for the type of chicken keypoint dataset that is was trained on, the peak performance of Resnet-101 in this project is around 200,000 and the second claim that was presented earlier is that complex network (Resnet-101) is easier to overfit because it has the capacity to do so. The second objective of this project is to propose a model that can do poultry keypoint prediction on a video. Hence, the CM2 is proposed however the current performance of CM2 is not accurate on new data but performs well on training data. Chapter 6 will talk about recommendations for any future work of the methods that could be adopted tuned the CM2 model further.

CHAPTER 6

CONCLUSIONS

To conclude, this thesis has gone through various methods that used different classes of Artificial Intelligence to estimate the health state or posture estimation of a poultry chicken. A method of pose estimation called DeepLabCut was adapted for this project and the series of steps taken to train a keypoint prediction neural network were also presented in detail. This project has managed to produce a poultry chicken dataset (available online, see appendix B.1) that is annotated with the body keypoints, which consist of around 29 videos of chicken and 256 labelled frames to fulfil the first objective of this project. Moving on, to fulfil the second objective, this project has managed to propose a deep learning model named CM2 (see chapter 5.5) that was tuned through a series of trial-and-error tests using a deep convolutional network that can detect poultry's body keypoints on a video however as mentioned in the final part of chapter 5 this model managed to get good accuracy on the training dataset but have a poor performance on new unseen data. The series of tests presented in chapter 4 is where the performance of the proposed model was compared to different networks under various hyperparameters to fulfil the third and last objective of this project which is to compare the accuracy of the proposed models to other models with different hyperparameters and backbones.

Based on the experience of using DeepLabCut, a pose estimation toolbox, there are a few limitations that the author experience, the first is the lack of accessibility to modify the network layers, the ability to add a dropout layer to the CM2 model network might increase its test accuracy and eliminate the overfitting problem. The dropout layer is a method to prevent the model from overfitting which will drop random neurons (randomly removing some of the learned weight and biases). Lacking the ability to change the neural network layers may be a huge

drawback for an experienced deep learning engineer. The second limitations are that it lacks available metrics that can be used to evaluate the model performance, the metrics available while evaluating the network performance with DeepLabCut is the pixel error and the training losses. Having an overall accuracy of the keypoint prediction or a learning curve would be a great insight to have when tuning the hyperparameters. Lastly, to finely tune the hyperparameters with DeepLabCut, one must use either use a trial-and-error test or tune it based on intuition and experience and this can be improved by others or developers of DeepLabCut who could work on making DeepLabCut compatible with a hyperparameter optimization algorithm such as Bayes search or Grid search rather than random search (trial and error) that was adapted for this project. To close off this chapter, this poultry keypoint detection project using a deep convolutional network is a basis of future work where others could implement the keypoint detection model with another model that could accurately predict the health of the poultry based on the estimated posture detected by the keypoint detection model.

REFERENCES

- [1] D. Berckmans, “General introduction to precision livestock farming,” pp. 6–11, 2017, doi: 10.2527/af.2017.0102.
- [2] M. Henchion, M. Mccarthy, V. C. Resconi, and D. Troy, “Meat consumption : Trends and quality matters,” *MESC*, vol. 98, no. 3, pp. 561–568, 2014, doi: 10.1016/j.meatsci.2014.06.007.
- [3] S. Asia, *OECD-FAO Agricultural Outlook 2017-2026*. 2017.
- [4] FAO, “Gateway to poultry production and products,” *Food and Agriculture Organizations of United States*, 2018. <http://www.fao.org/poultry-production-products/products-processing/en/>.
- [5] I. D. A. Na, N. Duarte, R. F. Gonc, L. A. De Lima, H. Ungaro, and J. M. Abe, “Lameness prediction in broiler chicken using a machine learning technique,” no. xxxx, 2020, doi: 10.1016/j.inpa.2020.10.003.
- [6] A. Aydin, “Development of an early detection system for lameness of broilers using computer vision,” *Comput. Electron. Agric.*, vol. 136, pp. 140–146, 2017, doi: 10.1016/j.compag.2017.02.019.
- [7] B. H. Thorp and S. R. Duff, “Effect of exercise on the vascular pattern in the bone extremities of broiler fowl.,” *Res. Vet. Sci.*, vol. 45, no. 1, pp. 72–77, Jul. 1988.
- [8] M. S. Dawkins, C. A. Donnelly, and T. A. Jones, “Chicken welfare is influenced more by housing conditions than by stocking density.,” *Nature*, vol. 427, no. 6972, pp. 342–344, Jan. 2004, doi: 10.1038/nature02226.
- [9] M. S. Dawkins, “Behaviour as a tool in the assessment of animal welfare **,” vol. 106, no. 2003, pp. 383–387, 2006.
- [10] T. G. Knowles *et al.*, “Leg Disorders in Broiler Chickens : Prevalence , Risk Factors and Prevention,” no. 2, pp. 1–5, 2008, doi: 10.1371/journal.pone.0001545.
- [11] D. Sergeant, R. Boyle, and M. Forbes, “Computer visual tracking of poultry,” vol. 21, pp. 1–18, 1998.
- [12] X. Zhuang and T. Zhang, “ScienceDirect Detection of sick broilers by digital image processing and deep learning,” *Biosyst. Eng.*, vol. 179, pp. 106–116, 2019, doi: 10.1016/j.biosystemseng.2019.01.003.
- [13] C. Okinda *et al.*, “ScienceDirect A machine vision system for early detection and prediction of sick birds : A broiler chicken model,” *Biosyst. Eng.*, vol. 188, pp. 229–242, 2019, doi: 10.1016/j.biosystemseng.2019.09.015.

- [14] “Recent Advances in Precision Livestock Farming,” vol. 1, no. 2.
- [15] P. Sayak, “Keypoint Detection with Transfer Learning,” *Keras*, May 02, 2021. https://keras.io/examples/vision/keypoint_detection/ (accessed Dec. 22, 2021).
- [16] C. Melissa, “Medical Definition of Posture,” *MedicineNet*, Mar. 29, 2021. <https://www.medicinenet.com/posture/definition.htm> (accessed Dec. 22, 2021).
- [17] G. Damerow, *The Chicken Health Handbook: A Complete Guide to Maximizing Flock Health and Dealing with Disease*, 2nd editio. Storey Publisher, 2015.
- [18] M. Manav, “CNN for Deep Learning | Convolutional Neural Networks,” *Analytics Vidhya*, May 01, 2021. <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/> (accessed Jan. 12, 2022).
- [19] S. Bartosz, “How Convolutional Neural Network works | by Bartosz Szabłowski | Towards Data Science,” *Towards Data Science*, Nov. 16, 2020. <https://towardsdatascience.com/how-convolutional-neural-network-works-cdb58d992363> (accessed Jan. 12, 2022).
- [20] S. Sumit, “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science,” *Towards Data Science*, Dec. 16, 2018. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> (accessed Jan. 12, 2022).
- [21] G. Carneiro, J. Nascimento, and A. P. Bradley, “Deep Learning Models for Classifying Mammogram Exams Containing Unregistered Multi-View Images and Segmentation Maps of Lesions,” *Deep Learn. Med. Image Anal.*, pp. 321–339, Jan. 2017, doi: 10.1016/B978-0-12-810408-8.00019-5.
- [22] C. A. Weeks, “lameness in broiler chickens,” no. June 2014, 2002, doi: 10.1136/vr.151.25.762.
- [23] O. H. Maghsoudi and A. Spence, “Superpixels Based Marker Tracking Vs . Hue Thresholding In Rodent Biomechanics Application,” pp. 209–213, 2017.
- [24] S. A. Mehdizadeh, D. P. Neves, M. Tschärke, I. A. Nääs, and T. M. Banhazi, “Image analysis method to evaluate beak and head motion of broiler chickens during feeding,” *Comput. Electron. Agric.*, vol. 114, pp. 88–95, 2015, doi: 10.1016/j.compag.2015.03.017.
- [25] X. Zhuang, M. Bi, J. Guo, S. Wu, and T. Zhang, “Development of an early warning algorithm to detect sick broilers,” *Comput. Electron. Agric.*, vol. 144, no. November 2017, pp. 102–113, 2018, doi: 10.1016/j.compag.2017.11.032.
- [26] C. Okinda *et al.*, “Artificial Intelligence in Agriculture A review on computer vision systems in monitoring of poultry : A welfare perspective,” *Artif. Intell. Agric.*, vol. 4, pp. 184–208, 2020, doi: 10.1016/j.aiia.2020.09.002.
- [27] C. Fang, J. Huang, K. Cuan, and X. Zhuang, “ScienceDirect Comparative study on poultry target tracking algorithms based on a deep regression network

- Region of Interest,” *Biosyst. Eng.*, vol. 190, no. 2016, pp. 176–183, 2020, doi: 10.1016/j.biosystemseng.2019.12.002.
- [28] C. Google, “Xception: Deep Learning with Depthwise Separable Convolutions,” pp. 1800–1807, 2017, doi: 10.1109/CVPR.2017.195.
- [29] J. Hui, “SSD object detection: Single Shot MultiBox Detector for real-time processing,” *Jonathan Hui Medium*, 2018. [https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06#:~:text=SSD is designed for object,what real-time processing needs.\(accessed Feb. 25, 2021\).](https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06#:~:text=SSD is designed for object,what real-time processing needs.(accessed Feb. 25, 2021).)
- [30] M. Everingham, L. Van Gool, C. K. I. Williams, and J. Winn, “The PASCAL Visual Object Classes (VOC) Challenge,” pp. 303–338, 2010, doi: 10.1007/s11263-009-0275-4.
- [31] C. Fang, T. Zhang, H. Zheng, J. Huang, and K. Cuan, “Pose estimation and behavior classification of broiler chickens based on deep neural networks,” *Comput. Electron. Agric.*, vol. 180, no. October 2020, p. 105863, 2021, doi: 10.1016/j.compag.2020.105863.
- [32] Y. Ed, “DeepLabCut: A Game-Changing Tool for Tracking Animal Movements - The Atlantic,” *The Atlantic*, Jul. 28, 2018. <https://www.theatlantic.com/science/archive/2018/07/deeplabcut-tracking-animal-movements/564338/> (accessed Dec. 30, 2021).
- [33] G. Yufeng, “Which Python package manager should you use? | by Yufeng G | Towards Data Science,” *towards data science*, Oct. 17, 2017. <https://towardsdatascience.com/which-python-package-manager-should-you-use-d0fd0789a250> (accessed Dec. 30, 2021).
- [34] W. Jeon, G. Ko, J. Lee, H. Lee, D. Ha, and W. W. Ro, “Chapter Six - Deep learning with GPUs,” in *Hardware Accelerator Systems for Artificial Intelligence and Machine Learning*, vol. 122, S. Kim and G. C. Deka, Eds. Elsevier, 2021, pp. 167–215.
- [35] Z. Sui, D. Raubenheimer, and A. Rangan, “Consumption patterns of meat, poultry, and fish after disaggregation of mixed dishes: Secondary analysis of the Australian National Nutrition and Physical Activity Survey 2011-12,” *BMC Nutr.*, vol. 3, no. 1, pp. 1–12, Dec. 2017, doi: 10.1186/S40795-017-0171-1/TABLES/5.
- [36] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, pp. 1–48, Dec. 2019, doi: 10.1186/S40537-019-0197-0/FIGURES/33.
- [37] T. Aysegul, “Top 13 Data Augmentation Techniques: Comprehensive Guide,” *AI Multiple*, Nov. 29, 2021. <https://research.aimultiple.com/data-augmentation-techniques/> (accessed Dec. 23, 2021).
- [38] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features

- in deep neural networks?,” *Adv. Neural Inf. Process. Syst.*, vol. 4, no. January, pp. 3320–3328, Nov. 2014, Accessed: Jan. 04, 2022. [Online]. Available: <https://arxiv.org/abs/1411.1792v1>.
- [39] A. Mathis *et al.*, “Pretraining boosts out-of-domain robustness for pose estimation,” *Proc. - 2021 IEEE Winter Conf. Appl. Comput. Vision, WACV 2021*, pp. 1858–1867, 2021, doi: 10.1109/WACV48630.2021.00190.
- [40] B. Jason, “Understand the Impact of Learning Rate on Neural Network Performance,” *Machine Learning Mastery*, Jan. 25, 2019. <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/> (accessed Jan. 06, 2022).
- [41] L. Fei-Fei, K. Ranjay, and X. Danfei, “CS231n Convolutional Neural Networks for Visual Recognition,” *Stanford CS231n Course Notes*. <https://cs231n.github.io/neural-networks-3/#add> (accessed Jan. 08, 2022).
- [42] T. Nath, A. Mathis, A. C. Chen, A. Patel, M. Bethge, and M. W. Mathis, “Using DeepLabCut for 3D markerless pose estimation across species and behaviors,” *Nat. Protoc.*, vol. 14, no. 7, pp. 2152–2176, 2019, doi: 10.1038/s41596-019-0176-0.
- [43] K. Konstantin and L. Molly, “A survey of Deep CNNs for Mouse Paw Location,” *Stanford*.
- [44] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 770–778, Dec. 2016, doi: 10.1109/CVPR.2016.90.
- [45] B. Jason, “How to Avoid Overfitting in Deep Learning Neural Networks,” *Machine Learning Mastery*, Dec. 17, 2018. <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/> (accessed Jan. 13, 2022).

APPENDICES

APPENDIX A

List of DeepLabCut Commands, File Parameters [42]

A.1 Summary of DeepLabCut Commands

Operation	Command
Open IPython and import DeepLabCut (Step 1)	<code>ipython</code> <code>import deeplabcut</code>
Create a new project (Step 2)	<code>deeplabcut.create_new_project('project_name', 'experimenter', ['path of video 1', 'path of video2', ...])</code>
Set a <code>config_path</code> variable for ease of use (Step 3)	<code>config_path = '/yourdirectory/project_name/config.yaml'</code>
Extract frames (Step 4)	<code>deeplabcut.extract_frames(config_path)</code>
Label frames (Steps 5 and 6)	<code>deeplabcut.label_frames(config_path)</code>
Check labels (optional)(Step 7)	<code>deeplabcut.check_labels(config_path)</code>
Create training dataset (Step 8)	<code>deeplabcut.create_training_dataset(config_path)</code>
Train the network (Step 9)	<code>deeplabcut.train_network(config_path)</code>
Evaluate the trained network (Step 11)	<code>deeplabcut.evaluate_network(config_path)</code>
Video analysis and plotting results (Step 11)	<code>deeplabcut.analyze_videos(config_path, ['path of video 1 or folder', 'path of video2', ...])</code>
Video analysis and plotting results (Step 12)	<code>deeplabcut.plot_trajectories(config_path, ['path of video 1', 'path of video2', ...])</code>
Video analysis and plotting results (Step 13)	<code>deeplabcut.create_labeled_video(config_path, ['path of video 1', 'path of video2', ...])</code>
Refinement: extract outlier frames (Step 14)	<code>deeplabcut.extract_outlier_frames(config_path, ['path of video 1', 'path of video 2'])</code>
Refine labels (Step 15)	<code>deeplabcut.refine_labels(config_path)</code>
Combine datasets (Step 16)	<code>deeplabcut.merge_datasets(config_path)</code>

A.2 Summary of Config.yaml Parameters

Box 1 | Glossary of parameters in the project configuration file (*config.yaml*)

The *config.yaml* file sets the various parameters for generation of the training set file and evaluation of results. The meaning of these parameters is defined here, as well as referenced in the relevant steps.

Parameters set during the project creation

- **task:** Name of the project (e.g., mouse-reaching). (Set in Step 1; do not edit).
- **scorer:** Name of the experimenter (set in Step 1; do not edit).
- **date:** Date of creation of the project. (Set in Step 1; do not edit).
- **project_path:** Full path of the project, which is set in Step 1; edit this if you need to move the project to a cluster/server/another computer or a different directory on your computer.
- **video_sets:** A dictionary with the keys as the full path of the video file and the values, `crop` as the cropping parameters used during frame extraction. (Step 1; use the function `add_new_videos` to add more videos to the project; if necessary, the paths can be edited manually, and the `crop` can be edited manually).

Important parameters to edit after project creation

- **bodyparts:** List containing names of the points to be tracked. The default is set to `bodypart1`, `bodypart2`, `bodypart3`, `objectA`. Do not change after labeling frames (and saving labels). You can add additional labels later, if needed.
- **numframes2pick:** This is an integer that specifies the number of frames to be extracted from a video or a segment of video. The default is set to 20.
- **colormap:** This specifies the colormap used for plotting the labels in images or videos in many steps. Matplotlib colormaps are possible (https://matplotlib.org/examples/color/colormaps_reference.html).
- **dotsize:** Specifies the marker size when plotting the labels in images or videos. The default is set to 12.
- **alphavalue:** Specifies the transparency of the plotted labels. The default is set to 0.5.
- **iteration:** This keeps the count of the number of refinement iterations used to create the training dataset. The first iteration starts with 0 and thus the default value is 0. This will auto-increment once you merge a dataset (after the optional refinement stage).

If you are extracting frames from long videos

- **start:** Start point of interval to sample frames from when extracting frames. Value in relative terms of video length, i.e., `[start=0, stop=1]` represents the full video. The default is 0.
- **stop:** Same as `start`, but specifies the end of the interval. Default is 1.

Related to the Neural Network Training

- **TrainingFraction:** This is a two-digit floating-point number in the range `[0-1]` used to split the dataset into training and testing datasets. The default is 0.95.
- **resnet:** This specifies which pre-trained model to use. The default is 50 (user can choose 50 or 101; see also Mathis et al.¹²).

Used during video analysis (Step 13)

- **batch_size:** This specifies how many frames to process at once during inference (for tuning of this parameter, see Mathis and Warren²⁷).
- **snapshotindex:** This specifies which checkpoint to use to evaluate the network. The default is `-1`. Use `all` to evaluate all the checkpoints. Snapshots refer to the stored TensorFlow configuration, which holds the weights of the feature detectors.
- **pcutoff:** This specifies the threshold of the likelihood and helps to distinguish likely body parts from uncertain ones. The default is 0.1.
- **cropping:** This specifies whether the analysis video needs to be cropped (in Step 13). The default is `False`.
- **x1, x2, y1, y2:** These are the cropping parameters used for cropping novel video(s). The default is set to the frame size of the video.

Used during refinement steps

- **move2corner:** In some (rare) cases, the predictions from DeepLabCut will be outside of the image (because of the location refinement shifts). This binary parameter ensures that those points are mapped to a user-defined point within the image so that the label can be manually moved to the correct location. The default is `True`.
- **corner2move2:** This is the target location, if `move2corner` is `True`. The default is set to `(50, 50)`.

A.3 Summary pose_cfg.yaml Parameters

Box 2 | Parameters of interest in the network configuration file, *pose_cfg.yaml*

Please note, there are more parameters that typically never need to be adjusted; they can be found in the default *pose_cfg.yaml* file at https://github.com/AlexEMG/DeepLabCut/blob/master/deeplabcut/pose_cfg.yaml.

- `display_iters`: An integer value representing the period with which the loss is displayed (and stored in *log.csv*).
 - `save_iters`: An integer value representing the period with which the checkpoints (weights of the network) are saved. Each snapshot has >90 MB, so not too many should be stored.
 - `init_weights`: The weights used for training. Default: `<DeepLabCut_path>/Pose_Estimation_Tensorflow/pretrained/resnet_v1_50.ckpt`. For ResNet-50 or 101, -- this will be automatically created. The weights can also be changed to restart from a particular snapshot if training is interrupted, e.g., `<full_path>-snapshot-5000` (with no file-type ending added). This would re-start training from the loaded weights (i.e., after 5,000 training iterations, the counter starts from 0).
 - `multi_step`: These are the learning rates and number of training iterations to perform at the specified rate. If the users want to stop before 1 million, they can delete a row and/or change the last value to be the desired stop point.
 - `max_input_size`: All images larger with size `width × height > max_input_size * max_input_size` are not used in training. The default is 1500 to prevent crashing with an out-of-memory exception for very large images. This will depend on your GPU memory capacity. However, we suggest reducing the pixel size as much as possible; see Mathis and Warren²⁷.
- The following parameters allow one to change the resolution:
- `global_scale`: All images in the dataset will be rescaled by the following scaling factor to be processed by the convolutional neural network. You can select the optimal scale by cross-validation (see discussion in Mathis et al.¹²). Default is 0.8.
 - `pos_dist_thresh`: All locations within this distance threshold (measured in pixels) are considered positive training samples for detection (see discussion in Mathis et al.¹²). Default is 17.

The following parameters modulate the data augmentation. During training, each image will be randomly rescaled within the range `[scale_jitter_lo, scale_jitter_up]` to augment training:

- `scale_jitter_lo`: 0.5 (default).
- `scale_jitter_up`: 1.5 (default).
- `mirror`: If the training dataset is symmetric around the vertical axis, this Boolean variable allows random augmentation. Default is `False`.
- `cropping`: Allows automatic cropping of images during training. Default is `True`.
- `cropratio`: Fraction of training samples that are cropped. Default is 40%.
- `minsize`, `leftwidth`, `rightwidth`, `bottomheight`, `topheight`: These define dimensions and limits for auto-cropping.

APPENDIX B

B.1 Collected videos, output videos (predicted by CM2), and labelled dataset link:

https://unitedumy-my.sharepoint.com/:f:/g/personal/ee0102953_student_uniten_edu_my/ErQn9OZZyuFEuQ4XCLW4N4QBGJ6bKzLdwJhIsie7LEu8Fg?e=sx2HYC

B.2 Links to this project GitHub repository which contains the installation codes, link to the dataset, and codes to the series of tests (please do refer to the file that contains the codes as well that is submitted through teams)

<https://github.com/amrhkm/POULTRY-POSE-DEEPLABCUTV2>