TOHOKU UNIVERSITY
Graduate School of Information Sciences

Application of Deep Neural Networks in Pose Estimation of Low Observable Objects for Tracking Bats (コウモリ追跡のための低輝度対象の姿勢推定へのディープニューラルネットワークの応用)

A dissertation submitted to the department of
Graduate School of Information Science in partial fulfillment of
the requirements for the degree of

Master of Philosophy
in
Information Science

by

Tiwat LARPVISUTTISAROJ

July 5th of, 2019

Application of Deep Neural Networks in Pose Estimation of Low Observable Objects for Tracking Bats

Tiwat LARPVISUTTISAROJ

Abstract

Animal behavioral analyses can be done in various ways, one way is to analyze images of the animal of interested and observe the behaviors. However, images analyses can be very labor intensive if the number of images is large. This thesis focuses on animal pose estimator as a tool for animal behavioral study. Deep learning is used to create a system that can work as the pose estimator. The pose estimator is designed to works well with the flying bat dataset which is introduced in the thesis. The pose estimator model gets a 3-channel image as the input and output the bat pose which is a combination of keypoints and the relationship between the keypoints.

In this study, the application of deep neural network pose estimator for the flying bat dataset introduced in this work is shown. The deep neural network pose estimator generally composes of the keypoint detector whose purpose is to predict the probability of a pixel on the input image is a keypoints and the keypoints connector which determines the relationship between keypoints. In this case, the dataset used in this study is flying bat dataset, the keypoints are both wings and the body. The dataset only has one bat in it, therefore, each keypoint is unique and the relationship can be a simple rule as the connection between the body and the wings. The essential part to create the well-performing deep neural network pose estimator is the keypoint detector, hence, the focus is narrowed down to improving the performance of the keypoint detector.

The keypoint detector is based on autoencoder model based on U-net with ResNet as the encoder backbone. With the help of data augmentation and taking advantage of time-sequential frames, the model can generalize the keypoints of the bat better. The model is tested on an image sequence and shows a respectable accuracy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pose estimation is one of a very old computer vision problem. One of the famous pose estimators that utilize classical computer vision technique prior to deep learning era is dated back to 2002[4]. In several years back, deep convolutional neural networks have become a hot topic in the research field. With more powerful hardware and a huge number of data to work with, deep learning has expanded extremely quickly. One of the fields that have become highly active in deep learning is human pose estimation. This is partly due to a handful array of human pose datasets (such as [5, 6, 7, 8]) and the promising performance that deep learning can achieve in such a task. The deep neural networks pose estimator has been shown to be increasingly reliable[9, 10, 11, 12].

Animal behavioral analyses can be done in various ways, one way is to analyze images of the animal of interested and observe the behavior. By looking at an animal image, we can easily identify what animal it is, where is it in the image, and what it is doing. With the use of high frame rate animal photography is very useful in studying the fast-moving and complex animal behavior. However, images analyses

can be very labor intensive as the amount of images increases due to the increase in data throughput of the high-speed camera. This is where pose estimation can see an application, to be a tool so that the study of such animal behavior becomes simpler.

This thesis focuses on animal pose estimator as a tool for animal behavioral study, especially for the bat dataset that is introduced in Section 3.1. The bat is a fast-moving animal and has a small body that alone makes the study of the bat behavior laborious to human. Moreover, it is also a nocturnal animal which is active in a dark environment only. All of those factors cause the studying of bat behavior even more tedious. The deep learning technique explored here is combined with data augmentation and a few loss functions are used to get the best performance out of the pose estimator model The model can achieve a respectable performance which will be shown in Chapter 4.

Chapter 2

Literature Review

2.1 Autoencoder

Autoencoder model is comprised of 2 parts; encoder and decoder as can be seen in Figure 2.1. The encoder (h) encodes the original data (y) into the encoded data ($h(y)$). The decoder (g) decoded the encoded data and gives back the representative of the original data (y'). In general, eq. 2.1 describes the process of autoencoder model.

$$y' = g(h(y)) \tag{2.1}$$

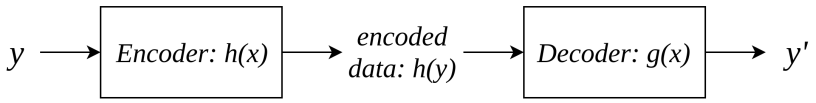

Figure 2.1: The block diagram of autoencoder where y is the original data and y' is the output of the autoencoder that represent y

Depending on the intended purpose of using autoencoder, the representative of the original data may not retain every piece of data like its original form. the au-

toencoder can retain only the essential part of the data and use that essential part to represent the original data. Some of the application of autoencoder in a communication system are error correction, data encryption, and data compression. Those application intended to make use of the encoded data in a specific environment and give back the original data from the decoder side.

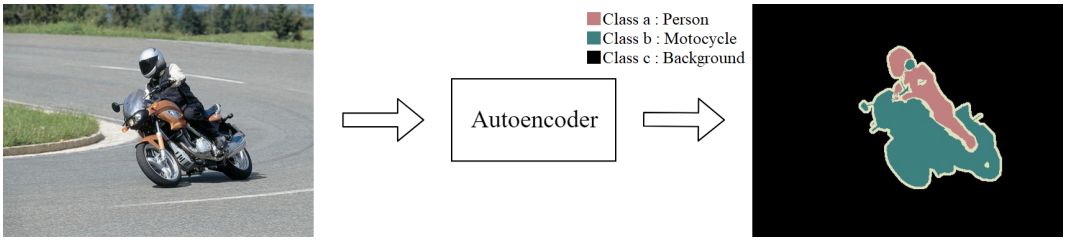

Figure 2.2: A simple illustration of semantic segmentation on an image in Pascal VOC dataset[1] where the input image goes through an autoencoder and it gives out classification of each pixels

In deep learning, autoencoder is widely used in any application. One of the earliest applications of the autoencoder is dimension reduction [13, 14] which is also designed to give the original data at the output of the decoder and the lower-dimension representative of the original data at the output of the encoder. In recent studies, the application of autoencoder in deep learning is to generalize the original data, to gain insightful information of the data as the output. This means that the output y' is not intended to be the exact copy of the original data. In principle, the autoencoder ignores insignificant details in the data and gives out the important pieces of data that can represent the original. Semantic segmentation problem where each pixel of the input image is classified into a class sees great use of autoencoder as seen in [15, 16, 17, 18, 19] which are some of the state of the art studies in semantic segmentation task that utilize autoencoder. In semantic segmentation problem, the encoder can be seen as the feature extractor that extracts important patterns/features

of the original data and the decoder uses those features to classify each element of the original data. Figure 2.2 shows a simple illustration of semantic segmentation process. The encoder in most cases is based on some existing architecture that works well in classification task such as VGG[20] and ResNet[2]. As the problem formulation of this thesis is based on semantic segmentation problem, autoencoder is the go-to architecture of this thesis.

2.2 Residual Neural Network

Residual Neural Network or ResNet is a popular neural network architecture that is used as encoder backbone for many autoencoders (as appeared in). ResNet is originally designed to show that performance degradation problem in deep neural networks can be solved by introducing residual/shortcut to deep neural networks. The structure of ResNet is shown in Figure 2.4. The study in [2] demonstrates that the residual can improve performance of deep neural networks by using ResNet to compete in ImageNet:image classification challenge[21] where ResNet gives remarkable performance on the dataset. The residual unit in Figure 2.3 can be described by eq.2.2 where y is the output of the residual unit and $\text{relu}\,(x) = \max\,(0, x)$. From the equation, it is shown that the residual unit can retain the information in the shallower layer of the neural network when the deeper layer can't find significant features in the middle layer ($F\,(x) = 0$)

$$y = \text{relu}\,(F\,(x) + x) \tag{2.2}$$

Seeing the brilliance in the design of ResNet, it is only normal that ResNet is repurposed as the encoder backbone for many other tasks as seen in [15, 22, 23, 24].

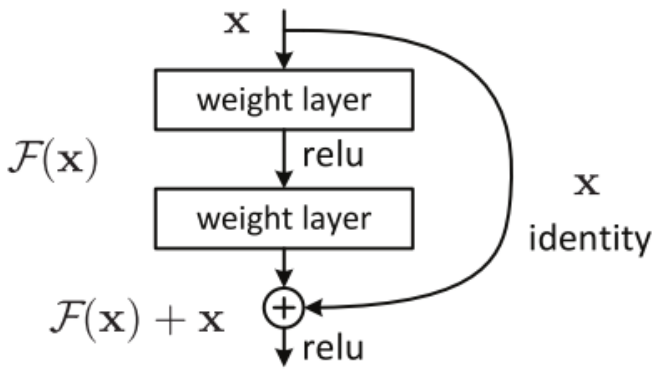

Figure 2.3: Residual unit as appeared in [2].

These are some of the advantages of using ResNet as the backbone backbone:

- ResNet gives a relatively good performance with relatively low computational cost compared to other architecture[25].
- The encoder can benefit from knowledge transfer of ResNet trained on ImageNet Dataset in initializing the autoencoder model.
- Popular deep learning platforms make it easy to create a model using ResNet architecture as the backbone.
- It is easier to focus on designing decoder when the great encoders are readily avaiable.

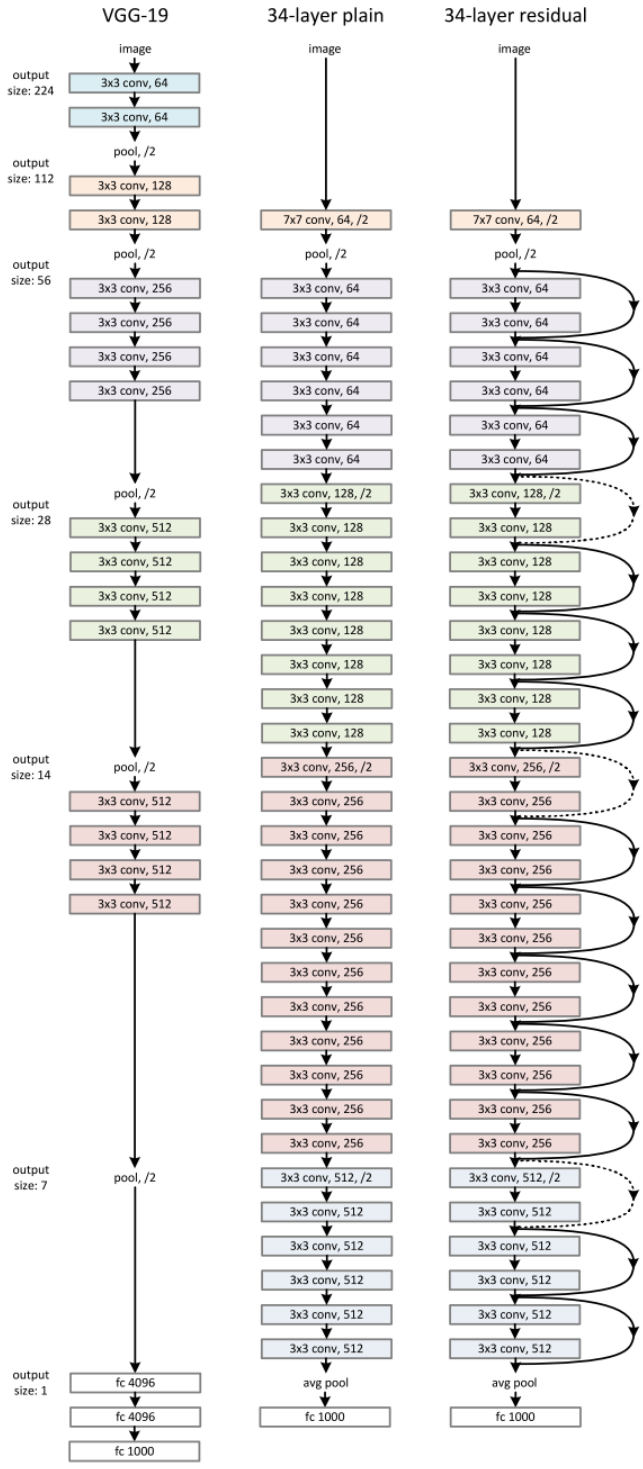

Figure 2.4: ResNet architecture is illustrated in comparison to VGG and plain neural network without residual in [2] where dash line shows increase in dimensions.

2.3 Softmax Function

Softmax function is used as the output function of the deep learning model in this study. The purpose of using Softmax function as the output function of a deep learning model is to map the output of the model to a probability distribution over the predicted output class. The predicted probability of the element k of a system outputs that belongs to class c is described as $P[c|k]$ where $\sum_c P[c|k] = 1$ and $y_{c,k}$ is the output of the deep learning model at element k before applying Softmax function. Softmax function is defined in eq. 2.3.

$$P\left[c|k\right] = \frac{\exp y_{c,k}}{\sum_c \exp y_{c,k}} \tag{2.3}$$

2.4 Weigted Cross-Entropy Loss Function

Cross-entropy loss is a loss function that is widely used in classification problem. The weighted cross-entropy loss is a class-dependent weighted version of cross-entropy loss. The weighted is included in normal cross-entropy function as there is the need to set a penalty to the class that is dominant in the dataset which is a big problem in the data with the unbalanced class problem. The rationale is that when one class is very dominant (the background class is most cases), it primarily contribute to the total value of loss; in this case, the value is larger than 150,000 times compared to the crucial target class. With that much-unbalanced loss, the optimization will try optimizing only for the dominant class, causing the minority class to be left unoptimized and this causes the model to perform badly with the minority class. By lowering the weight of the dominant class, the loss caused by the class is reduced which, in turn, affects

the optimization of the model less and the optimizer can focus on other classes. The result is the model that is less focus on one class and can perform well in every class.

The weighted cross-entropy loss function is described by eq. 2.4 where c denotes a class, w_c is class dependent weight. $q(c|k)$ (described in eq. 2.5) is the ground truth probability that the element k of a system outputs belongs to class c, and $p\,(c|k)$ is the predicted probability of the element k to belongs to class c. It is the loss function that is widely used in the classification task and semantic segmentation task.

$$H\left(p(c|k), q(c|k)\right) = -w_c \times q(c|k)\log\left(p(c|k)\right) \tag{2.4}$$

$$q(c|k) = \begin{cases} 1.0, & \text{if the element } k \text{ actually belongs to class } c; \\ 0.0, & \text{otherwise.} \end{cases} \tag{2.5}$$

2.5 Euclidean Distance

Euclidean distance is the distance between 2 points in Euclidean space. Euclidean distance described by eq. 2.6 where $\boldsymbol{a} = (a_1, a_2, ..., a_n)$ is the vector of coordinate $(a_1, a_2, ..., a_n)$ in euclidean space of n dimensions, and $\boldsymbol{b} = (b_1, b_2, ..., b_n)$ is the vector of coordinate $(b_1, b_2, ..., b_n)$ in euclidean space of n dimensions.

$$|\boldsymbol{a}, \boldsymbol{b}| = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2.6}$$

Chapter 3

Methodology

3.1 The Dataset

The dataset composes of 1394 labeled images of a bat flying in a dark chamber. The image is captured by HXC40NIR Baumer near-infrared camera which is capable of 357 FPS maximum. There are 2 cameras placed in front of the chamber where the bat flies with equal distance to the chamber (one camera on the left and the other on the right). the images size is 1024×1024 pixel^2 grayscale image. The sample of an image captured on the left camera is shown in Figure 3.1 and the sample of an image captured on the right camera is shown in Figure 3.2.

The data is labeled with 3 keypoints of the bat which are left wing, body, and right wing. The data labeling process is done by employing 2 humans to select the location of the keypoints. The samples of the actual location of the labeled key-points on the images are shown in Figure 3.3

The labeled dataset comes from 1500 sequential frames of the flying bat where frame number 8 to 102 from both cameras are labeled and separate for validation

test. The number of the validation test dataset is 190 images in total (95 images from each camera). The samples of the test set are shown in Figure 3.4 where the figure shows the possible location of the bat in the test set.

The dataset used in training of deep learning model is labeled from frame number 141 to 1500 from both cameras. The labeling process is done on every three frames instead of every frame as done with the test set (i.e. frame number 141, 144, 147, ..., 1500 are labeled). There are 910 images used as the training set in total (455 images from each camera). The samples of the training set are shown in Figure 3.5, Figure 3.6, Figure 3.7, and Figure 3.8 where the figures show the possible location of the bat in the training set.

Figure 3.1: A sample of the dataset captured on the left camera and the location of the bat (in red circle) with the actual bightness level

Figure 3.2: A sample of the dataset captured on the right camera and the location of the bat (in red circle) with the actual bightness level

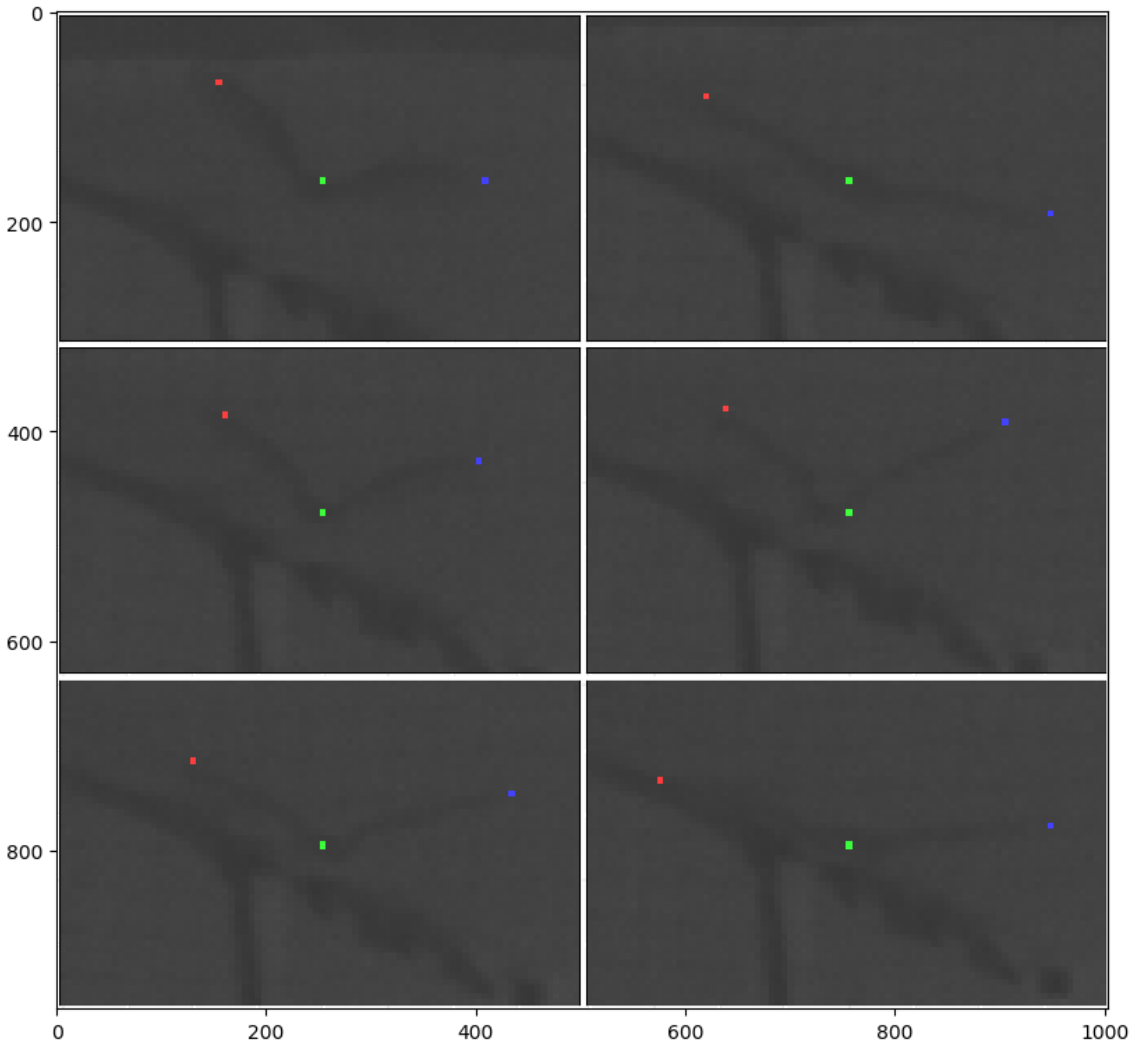

Figure 3.3: The labeled keypoints location is shown on the actual image that the points are labeled with. The image is cropped, zoomed, and increased the brightness for better visualization. The cropped image is centered at the green point which locates the body keypoint, the red dot is the right wing, and the blue dot is the left wing.

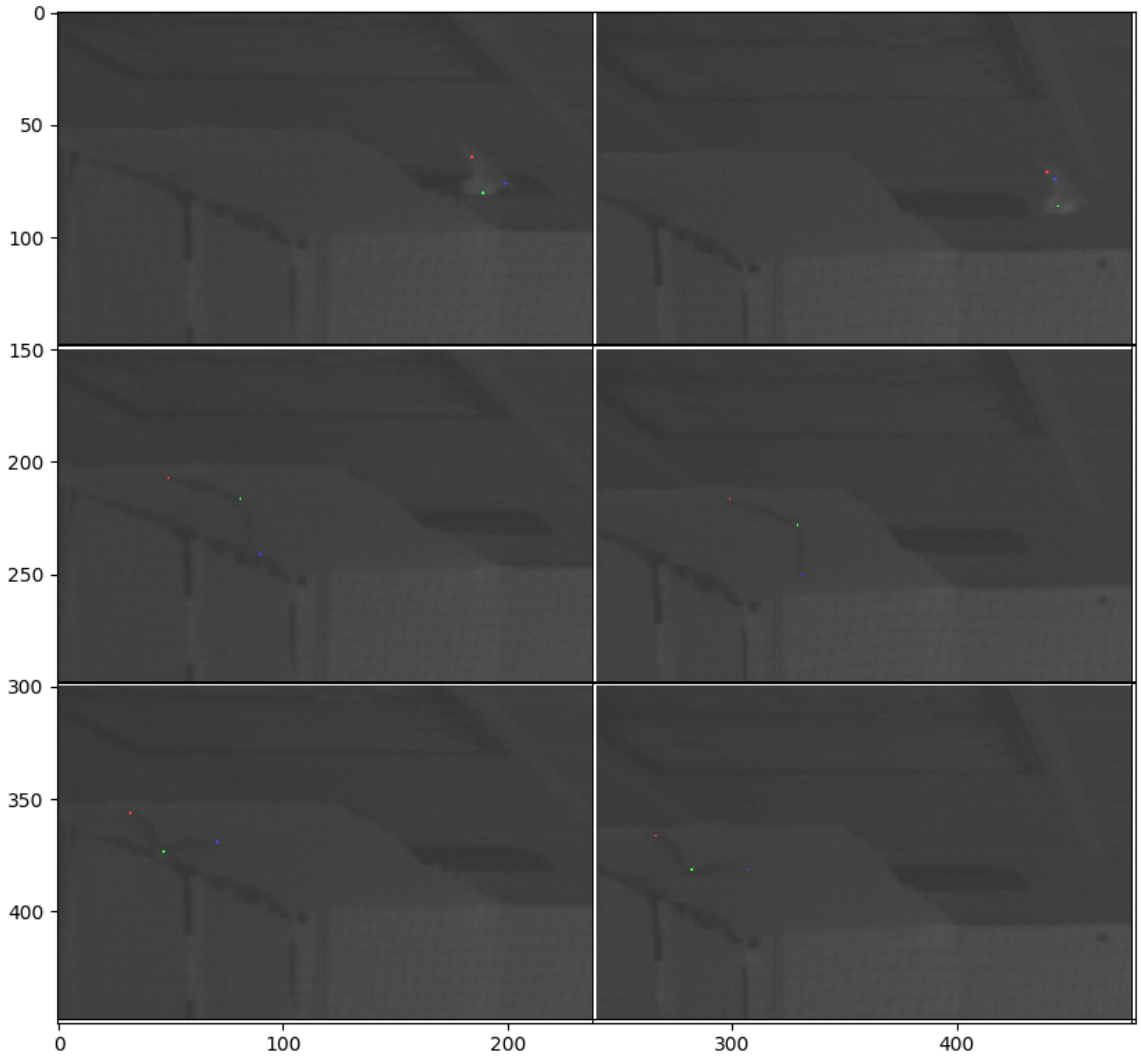

Figure 3.4: The labeled key-points location is shown on the actual image from the test set. The image is cropped, zoomed, and increased the brightness for better visualization. The green point locates the body key-point, the red dot is the right wing, and the blue dot is the left wing. The image shows the location of the bat in frame number 8, 56, and 102 in order of top to bottom.

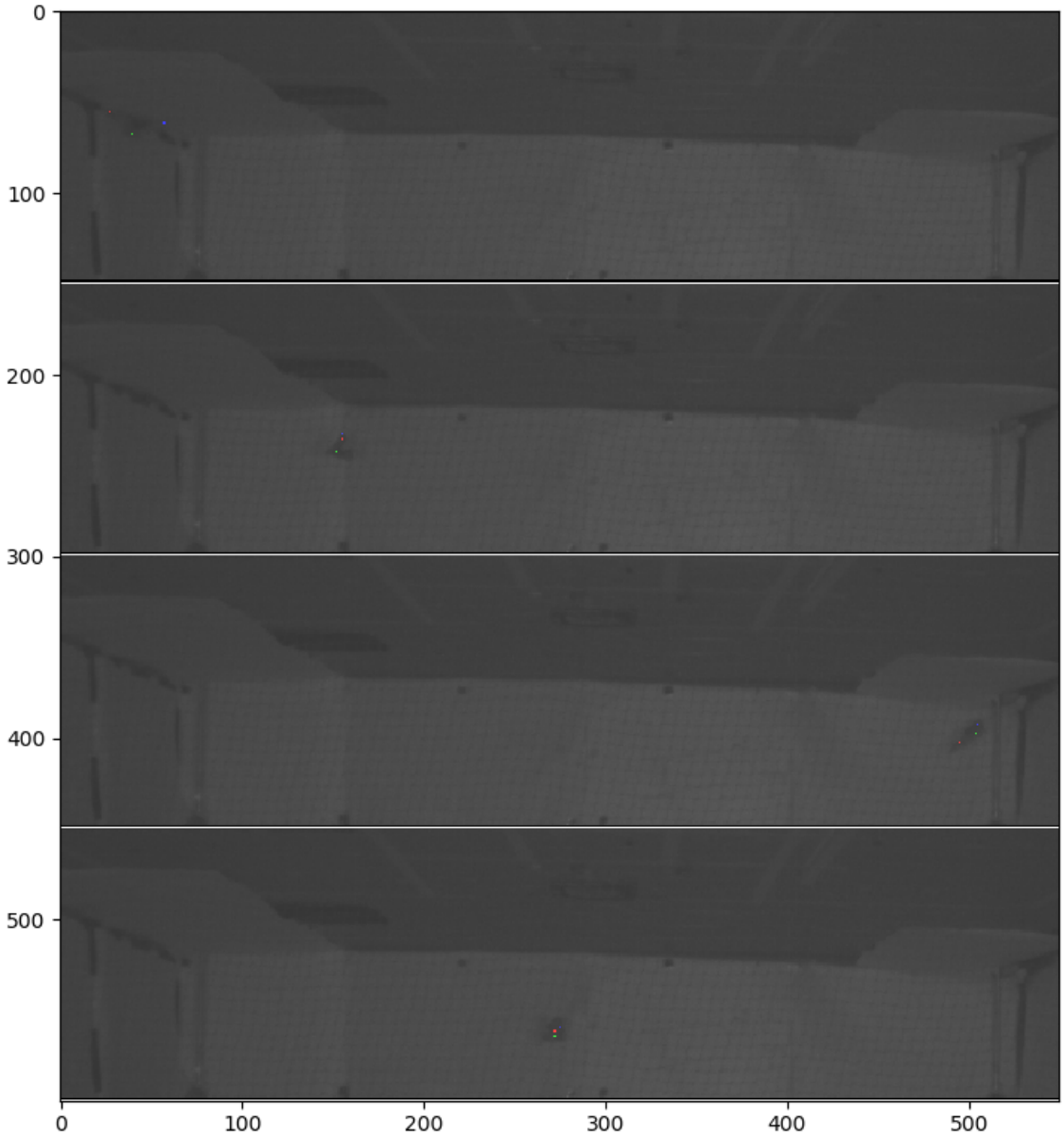

Figure 3.5: The labeled key-points location is shown on the actual image captured from the left camera from the training set. The image shows the location of the bat in frame number 201, 300, 621, and 786 in order of top to bottom. The green point locates the body key-point, the red dot is the right wing, and the blue dot is the left wing. The image is cropped, zoomed, and increased the brightness for better visualization.

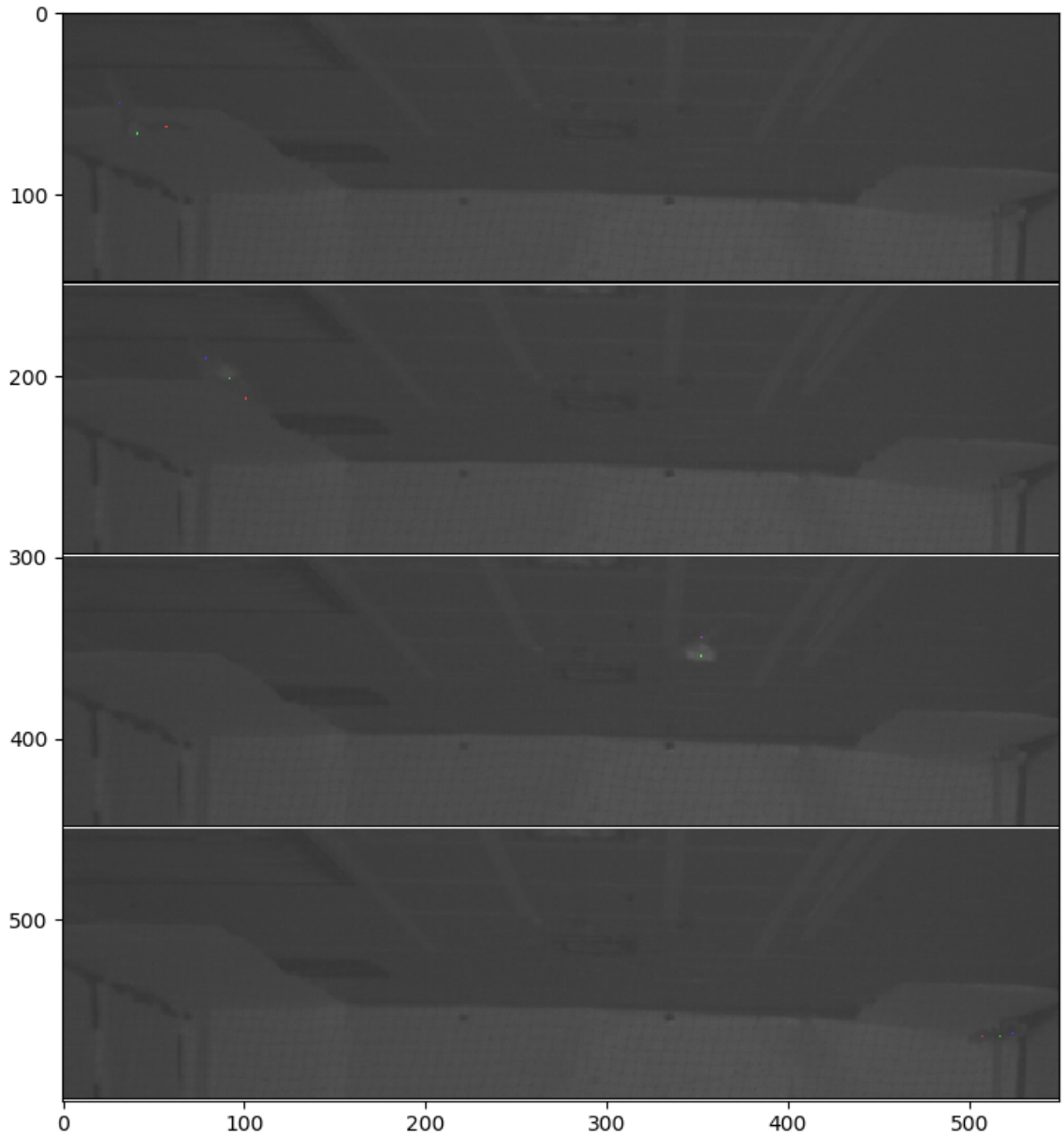

Figure 3.6: The labeled key-points location is shown on the actual image captured from the left camera from the training set. The image shows the location of the bat in frame number 1161, 1191, 1281, and 1500 in order of top to bottom. The green point locates the body key-point, the red dot is the right wing, and the blue dot is the left wing. The image is cropped, zoomed, and increased the brightness for better visualization.

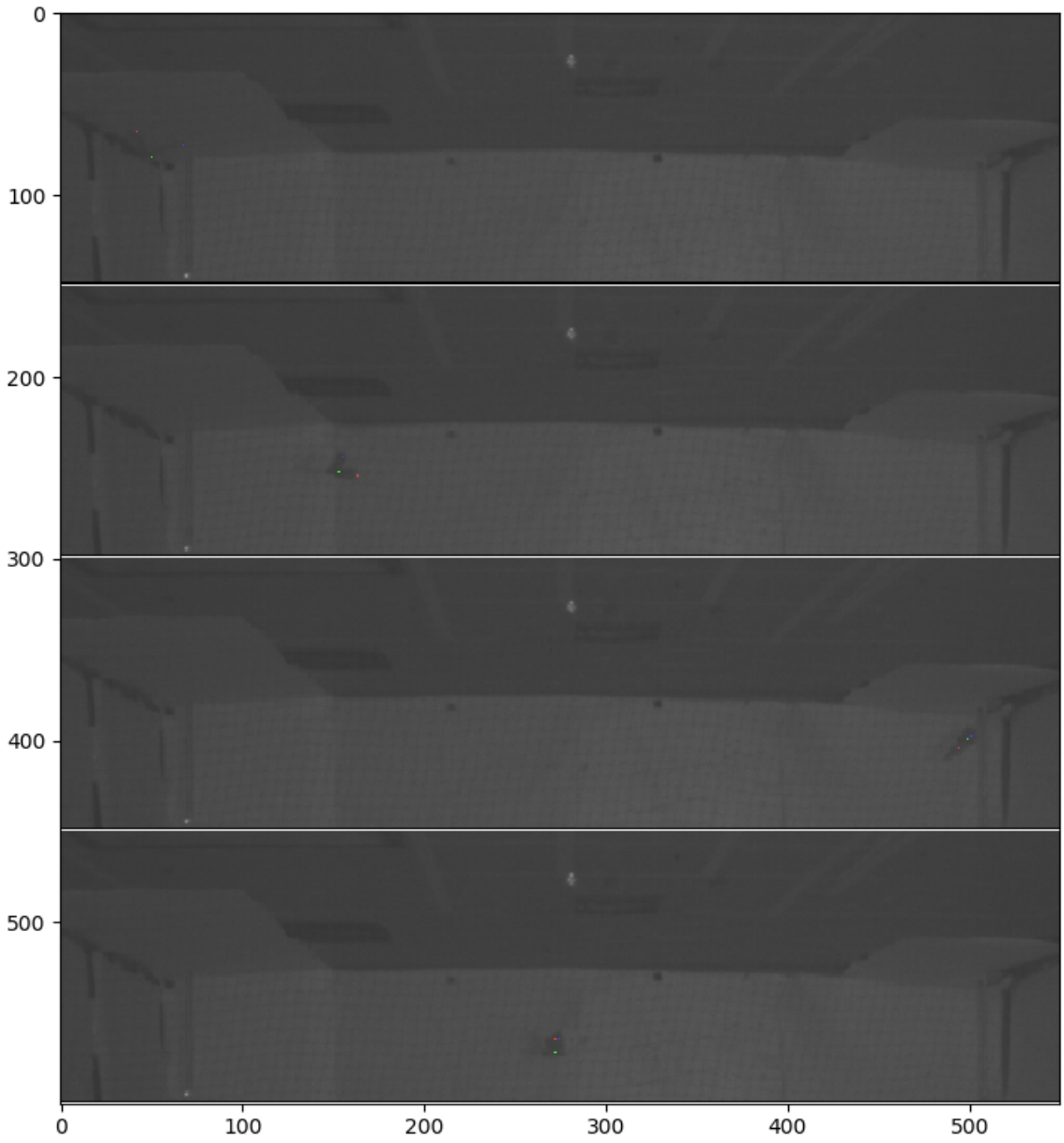

Figure 3.7: The labeled key-points location is shown on the actual image captured from the right camera from the training set. The image shows the location of the bat in frame number 201, 300, 621, and 786 in order of top to bottom. The green point locates the body key-point, the red dot is the right wing, and the blue dot is the left wing. The image is cropped, zoomed, and increased the brightness for better visualization.

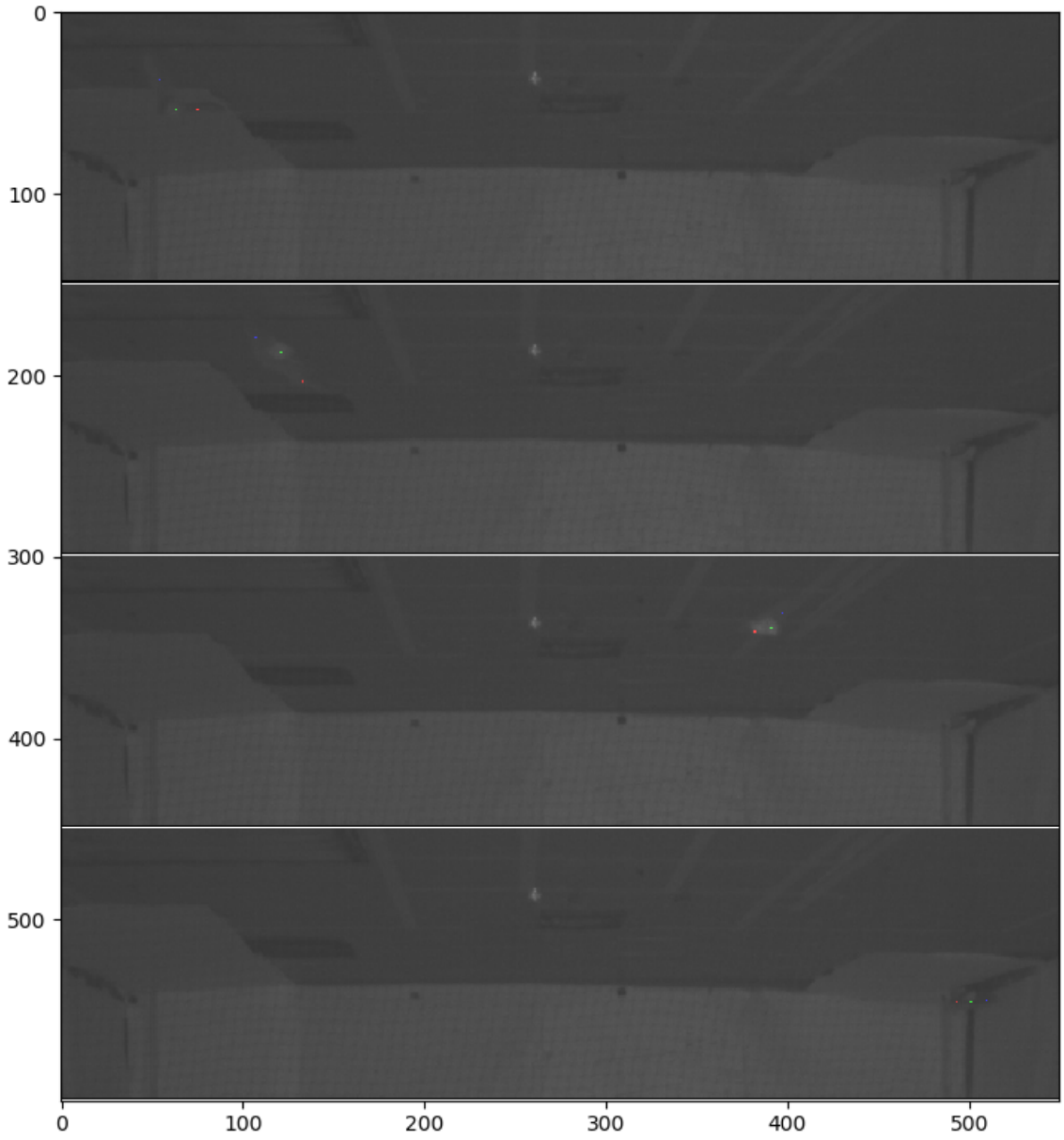

Figure 3.8: The labeled key-points location is shown on the actual image captured from the right camera from the training set. The image shows the location of the bat in frame number 1161, 1191, 1281, and 1500 in order of top to bottom. The green point locates the body key-point, the red dot is the right wing, and the blue dot is the left wing. The image is cropped, zoomed, and increased the brightness for better visualization.

3.2 Model

The objective of this study is making a deep learning pose-estimator model that works well with the flying bat dataset. The pose, in this case, is defined as the location of the keypoints and their relationship;

- the keypoints, in this case, are the locations of the left wing, the body, and the right wing,
- the relationship of the keypoints, in this case, is which bat the keypoints belongs to and its relationship between each point.

Most of the pose-estimation works have 2 components: body-part keypoint detector whose purpose is to detect the keypoints and body-parts connector whose purpose is to determine the relationship between each keypoints. The focus is narrowed down to improving body-part keypoint detector as it is the key for well-performing pose-estimator on the dataset. The body-part keypoint detector takes an RGB image of size $Height \times Width \times 3$ as the input and predicts 3 body-part keypoints (left wing, body, and right wing). The predicted output is of size $Classes \times Height \times Width$ where $Classes$ are left wing, body, and right wing, and background. Using Softmax function as described in Section 2.3, the probability output at (row x, column y) location that belongs to each body-part classes naturally sums to 1.0 as shown in eq. 3.1.

As the input is the image of $Height$ rows and $Width$ columns, $\boldsymbol{X}$ indicates the row number where $\boldsymbol{X} = \{x | x \in \mathbb{Z}, x \in [0, Width - 1]\}$, and $\boldsymbol{Y}$ indicates the column number where $\boldsymbol{Y} = \{y | y \in \mathbb{Z}, y \in [0, Height - 1]\}$. The predicted probability of the location (x, y) to belong in class $c \in \{\text{left wing, body, right wing}\}$ is presented as

$P[\boldsymbol{C} = c|\boldsymbol{Y} = y, \boldsymbol{X} = x]$ in eq. 3.1.

$$\sum_{c} P[\boldsymbol{C} = c|\boldsymbol{Y} = y, \boldsymbol{X} = x] = 1.0 \tag{3.1}$$

The dataset only has one bat, therefore each body-part keypoints are unique, therefore, the body-part connector can link each part with the simple rules as following:

- Left wing to body,
- Body to the right wing.

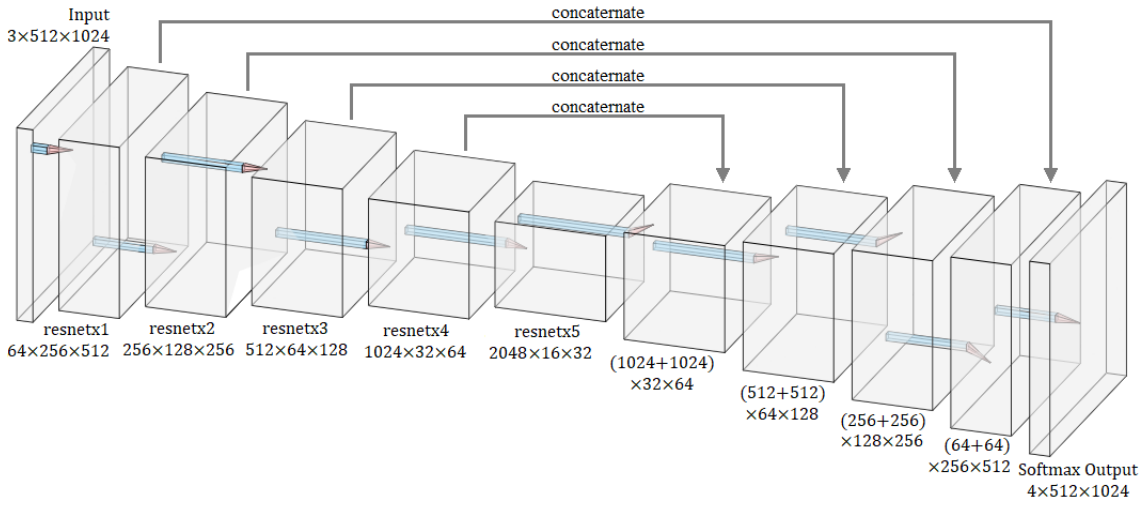

Figure 3.9: The model used in this study has U-Net like skip connections. The figure is generated from [3].

Figure 3.9 shows a simple illustration of the body-part keypoint detector used in this study. The body-part keypoint detector is based on the autoencoder model where ResNet described in Section 2.2 is used as the encoder backbone. The decoder part is based on U-net[26]. U-Net utilizes not only the deeper level features but also the lower level features which are said to be helpful for size-invariant model[27].

3.3 Validation Test Process

The test dataset as described in Section 3.1 is used only for the validation test process. The data are not used in the model training process so that the model can show how well the model can represent the dataset. Euclidean Distance described in Section 2.5 is used as the testing metric in this study. Using the aforementioned data, the outputs of the model using the test dataset as the input are compared with the ground truth location of each human-labeled body-part keypoints. The distance in pixel between the most probable body-part keypoints and the ground truth location is measured. The average distance is, then, compared to show if the model achieves good performance on the dataset.

Using the same definition of the predicted probability that coordinate (x, y) belongs to class c in eq. 3.1, the most probable of the locaton that belongs to class c is defined as $\boldsymbol{U}_c$ in eq. 3.2.

$$\boldsymbol{U}_c = \left\{ (x,y) | P[\boldsymbol{C} = c | \boldsymbol{Y} = y, \boldsymbol{X} = x] = \max_{x',y'} P[\boldsymbol{C} = c | \boldsymbol{Y} = y', \boldsymbol{X} = x'] \right\} \tag{3.2}$$

3.4 Deep Neural Network Model Training: Optimizer, Training Set, and Image Preprocessing

The model is trained using the training set described in Section 3.1. The input images are cropped to 512×1024 pixel^2 to reduce the computational time and remove human on the scene. The human in the scene is the only movable object other than the bat. The movable object other than the bat can give the model false information as the model may rely on the human posture and predict the body-part keypoints

relative to the movable object. The grayscale image is also converted into an RGB image to match the color channels to the input of the model.

To initialize the training, ResNet backbone is pretrained with ImageNet dataset. The model is trained by batch training with a batch size of 4. The data augmentation is random crop and horizontally flip. The image is padded with a constant value before being cropped to get the same image size back. The crop can be seen as position translation instead of just cropping. The input image is also randomly horizontally flipped, the probability of an image being flipped is 50%. The model is trained batch by batch until all of the training data is used, this means one epoch of training is finished. After that, the next epoch starts with the same procedure.

L2-regularized Stochastic Gradient Descent with Nesterov Momentum is used in the optimization of the model.

3.5 Train The Deep Learning Model with Cross-Entropy Loss

The model is trained using the training target of the same size as the input image times the number of classes; $512 \times 1024 \times 4$). Each location on the output belongs to only one class, i.e. the probability of the specific human-labeled class on that position is 1.0 and the probability of other classes are 0.0.

In the same way, Eq. 3.4 describes the probability of the training target at row y, column x to be class c, $P[\boldsymbol{C} = c|\boldsymbol{Y} = y, \boldsymbol{X} = x]$. $\boldsymbol{X}$ indicates the row number where $\boldsymbol{X} = \{x|x \in \mathbb{Z}, x \in [0, 1023]\}$, $\boldsymbol{Y}$ indicates the column number where $\boldsymbol{Y} = \{y|y \in \mathbb{Z}, y \in [0, 447]\}$, and $\boldsymbol{C}$ indicates the classes where

$\boldsymbol{C} = \{c|c \in \{\text{left wing, body, right wing, background}\}\}$. From the previously defined classes, the human-labeled body-part keypoints can be described in Eq. 3.3 where the labeled location of class $c \in \boldsymbol{C}$ (named $\boldsymbol{V}_c$) is a set of vector containing row number (y_k) and column number (x_k) that indicates the location of each point k that belongs to the class. Each class c has n_c number of points that belongs to the class, i.e., $\boldsymbol{K} = \{k|k \in \mathbb{Z}, k \in [0, n_c - 1]\}$.

$$\boldsymbol{V}_c = \{(x_k, y_k)\,|k \in \boldsymbol{K}, x_k \in \boldsymbol{X}, y_k \in \boldsymbol{Y}\} \tag{3.3}$$

$$P[\boldsymbol{C} = c|\boldsymbol{Y} = y, \boldsymbol{X} = x] = \begin{cases} 1.0; & \text{if } c \in \boldsymbol{C} \text{ and } (x, y) \in \boldsymbol{V}_c, \\ 0.0; & \text{otherwise} \end{cases} \tag{3.4}$$

Since the human-labeled body-part keypoints are 1×1 pixel^2 each, on one image, the number of labeled body-part keypoints is 3 pixel^2 (one each) and the number of background class is $448 \times 1024 - 3$ pixel^2 (458749 in total). Figure 3.10 shows the 1-point representation of each body-part keypoints on a sample of the dataset. The figures show how small those points are compared to the background classes.

Since this is essentially semantic segmentation problem with classes unbalanced problem, weighted cross-entropy loss described in Section 2.4 is used. The result of the training will be shown in the next chapter.

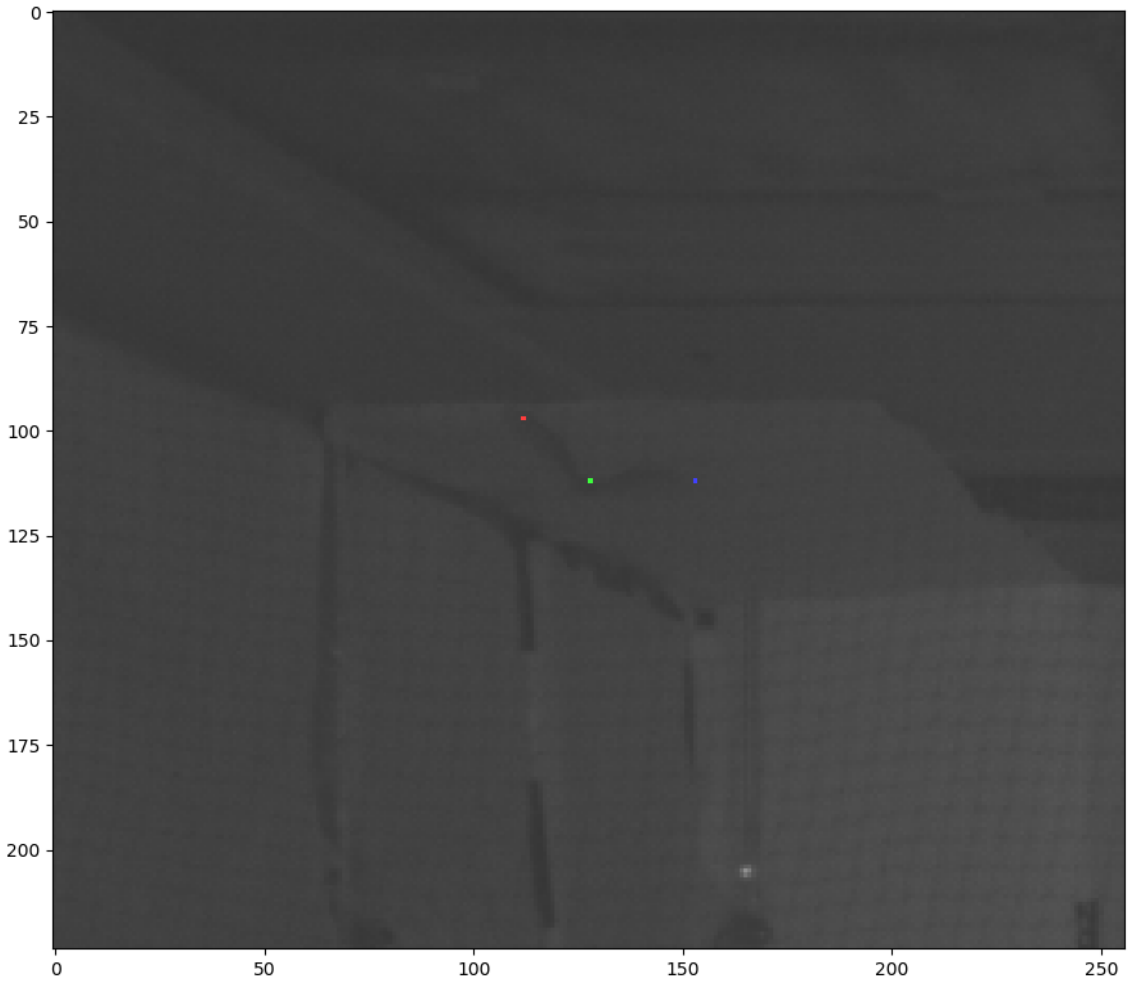

Figure 3.10: 1-point labeled body-part keypoints is displayed on a sample of the dataset that is cropped to 1/8 of the size of the full image. This shows how small the total labeled body-part keypoints area compared to the background area. The green dot is the location of the body, the red dot is the location the right wing, and the blue dot is the location of the left wing.

Chapter 4

Result and Discussion

The model described in Section 3.2 is trained with the data described in Section 3.1. Before discussing the result, the parameters for data augmentation described in Section3.4 are shown in Table 4.

Table 4.1: Parameters commonly used in the data augmentation is showm in this table.

Parameter name	value
Crop size	512×1024
Zero padding in every side	$100 pixels$
Horizontal Flip Probability	50%
Time-sequential Frames	t-4 and t-7

4.1 Training Result of The Deep Learning Model Trained with Cross-entropy Loss

The model is trained with the method described in Section 3.5. The class dependent weight of the weighted cross-entropy is described in Eq. 4.1 where c is the class and w_c is the class dependent weight. The rationale behind the weight is balancing the influence of each class by compensating all class to have the same $w_c \times$ pixel number.

Table 4.2: Parameters commonly used in every model training of this chapter is showm in this table.

Parameter name	magnitude
Learning rate	0.0007
Nesterov momentum weight	0.9
L2 regularization weight	0.03
Learning rate decay rate	0.1
Learning rate decay every	3 epochs

$$w_c = \begin{cases} 1; & c = \text{left wing}, \\ 1; & c = \text{body}, \\ 1; & c = \text{right wing}, \\ \dfrac{1}{1024 \times 448 - 3}; & c = \text{background} \end{cases} \tag{4.1}$$

The training loss is shown in Figure4.1 and the accuracy for the training data is shown in Figure 4.3. Both figures shows that the model can be trained and perform well on the test set and it should also perform as well on the data with high correspondance to the test set. The validation

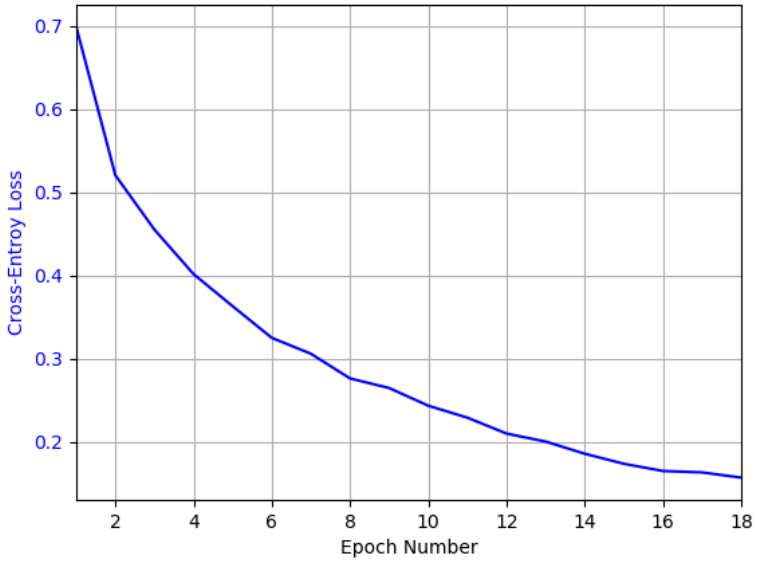

Figure 4.1: The resulting loss of training the model with the method in Section 3.5 decays over the training epoch naturally. The loss shows limited improvement after around 50 epochs of training.

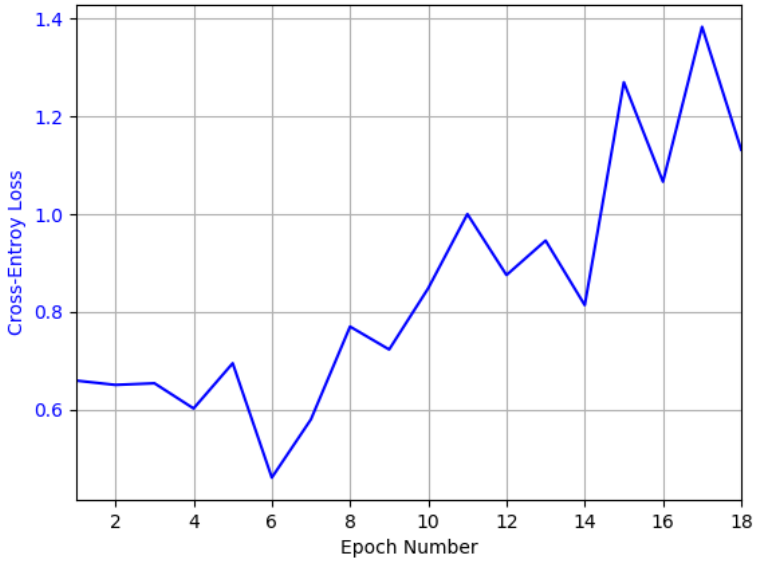

Figure 4.2: This image shows the loss of testing the model with the testing data described in Section 3.3. The model is trained with the method in Section 3.5. The validation loss shows an increasing trend which is contrast with the validation accuracy which improves over training epoch.

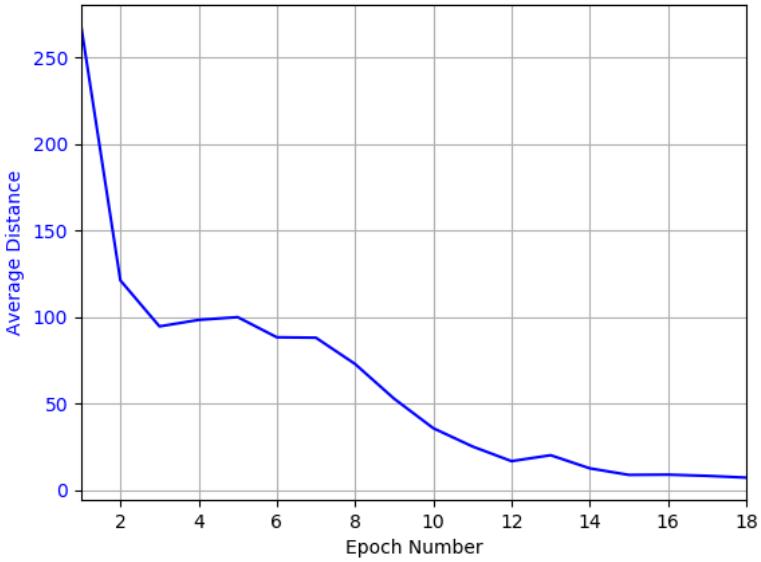

Figure 4.3: The result accuracy (distance in pixels) of the training set during the training with the method in Section 3.5 shows that the model perform well on the trainig data.

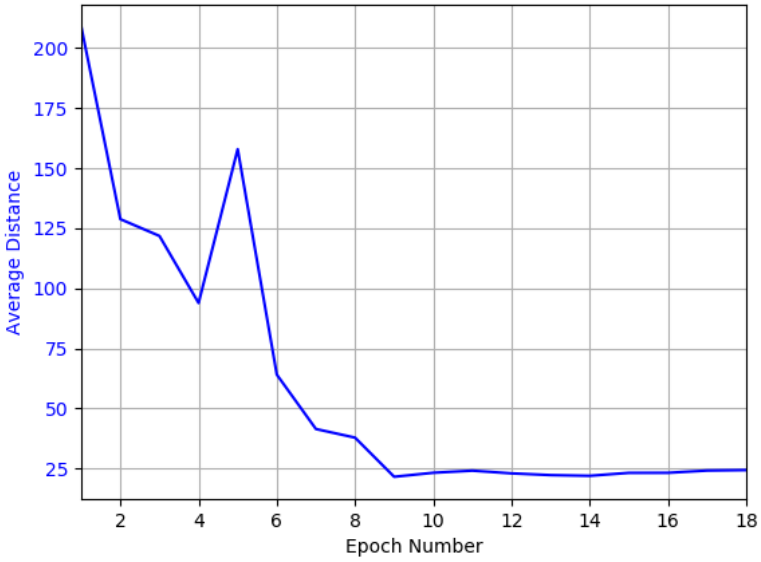

Figure 4.4: The accuracy (distance in pixels) of the trained model using the method in Section 3.5 shows a moderately high .

4.2 Future improvement

The result shows that the combination of class dependent weight and time-sequential input data can give an acceptable error when the result outlier is removed. However, the model is only trained on the dataset with one bat and the keypoints connector can structurally only works with one bat.

Furthur improvement can be done to this model by designing a neural network keypoints connector which can distinguish between 2 or more instances in the same class and assign the right keypoint to the associate instances.

Bibliography

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] ALEXANDER LENAIL. Nn-svg: Publication-ready nn-architecture schematics. http://alexlenail.me/NN-SVG/AlexNet.html.

[4] Greg Mori and Jitendra Malik. Estimating human body configurations using shape context matching. In Anders Heyden, Gunnar Sparr, Mads Nielsen, and Peter Johansen, editors, *Computer Vision — ECCV 2002*, pages 666–680, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[5] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7297–7306, 2018.

[6] Albert Haque, Boya Peng, Zelun Luo, Alexandre Alahi, Serena Yeung, and Li Fei-

Fei. Towards viewpoint invariant 3d human pose estimation. In *European Conference on Computer Vision (ECCV)*, October 2016.

[7] Sam Johnson and Mark Everingham. Clustered pose and nonlinear appearance models for human pose estimation. In *Proceedings of the British Machine Vision Conference*, 2010. doi:10.5244/C.24.12.

[8] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[9] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter V. Gehler, and Bernt Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. *CoRR*, abs/1511.06645, 2015.

[10] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *CoRR*, abs/1605.03170, 2016.

[11] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[12] Yilun Chen, Zhicheng Wang, Yuxiang Peng, Zhiqiang Zhang, Gang Yu, and Jian Sun. Cascaded pyramid network for multi-person pose estimation. *CoRR*, abs/1711.07319, 2017.

[13] Jordan B. Pollack. Implications of recursive distributed representations. In *Pro-*

ceedings of the 1st International Conference on Neural Information Processing Systems, NIPS'88, pages 527–536, Cambridge, MA, USA, 1988. MIT Press.

[14] Jordan B. Pollack. Recursive distributed representations. *Artif. Intell.*, 46:77–105, 1990.

[15] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *CoRR*, abs/1802.02611, 2018.

[16] Huikai Wu, Junge Zhang, Kaiqi Huang, Kongming Liang, and Yizhou Yu. Fastfcn: Rethinking dilated convolution in the backbone for semantic segmentation. *CoRR*, abs/1903.11816, 2019.

[17] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, Dec 2017.

[18] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, June 2015.

[19] Tsung-Yi Lin, Piotr Dollár, Ross B. Girshick, Kaiming He, Bharath Hariharan, and Serge J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[21] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[22] Brandon Forys, Dongsheng Xiao, Pankaj Gupta, Jamie D Boyd, and Timothy H Murphy. Real-time markerless video tracking of body parts in mice using deep neural networks. *bioRxiv*, 2018.

[23] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *CoRR*, abs/1606.00915, 2016.

[24] Xiao-Jiao Mao, Chunhua Shen, and Yu-Bin Yang. Image restoration using convolutional auto-encoders with symmetric skip connections. *CoRR*, abs/1606.08921, 2016.

[25] Simone Bianco, Rémi Cadène, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *CoRR*, abs/1810.00736, 2018.

[26] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[27] Peiyun Hu and Deva Ramanan. Finding tiny faces. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1522–1530, 2017.

Acknowledgments

I would like to express gratitude to Professor Koichi HASHIMOTO, Associate Professor Shingo KAGAMI, Professor Takayuki OKATANI, and Naoya CHIBA for guiding and giving me advice throughout this study. I also would like to Professor Shizuko HIRYU, Fujioka EMYO, Hase KAZUMA, Saori SUGIHARA, and Ilya ARDAKANI for providing me the dataset used in this study. Without the data, this study cannot begin in the first place.

A special thanks goes to Wasu WASUSATEIN and Shintaro TAKAYAMA for helping me label the data. The help in labeling save me a lot of time so that I can focus more on improving the work. I also would like to acknowledge with much appreciation to Ilya ARDAKANI and Engkarat TECHAPANURA for exchanging ideas to improve this work with me. Finally, I would also like to thank my friends who comfort me when I was stressed.

I have to appreciate all the support and guidance given by everyone. Without all those who support me, it is not possible for me to finish this thesis.