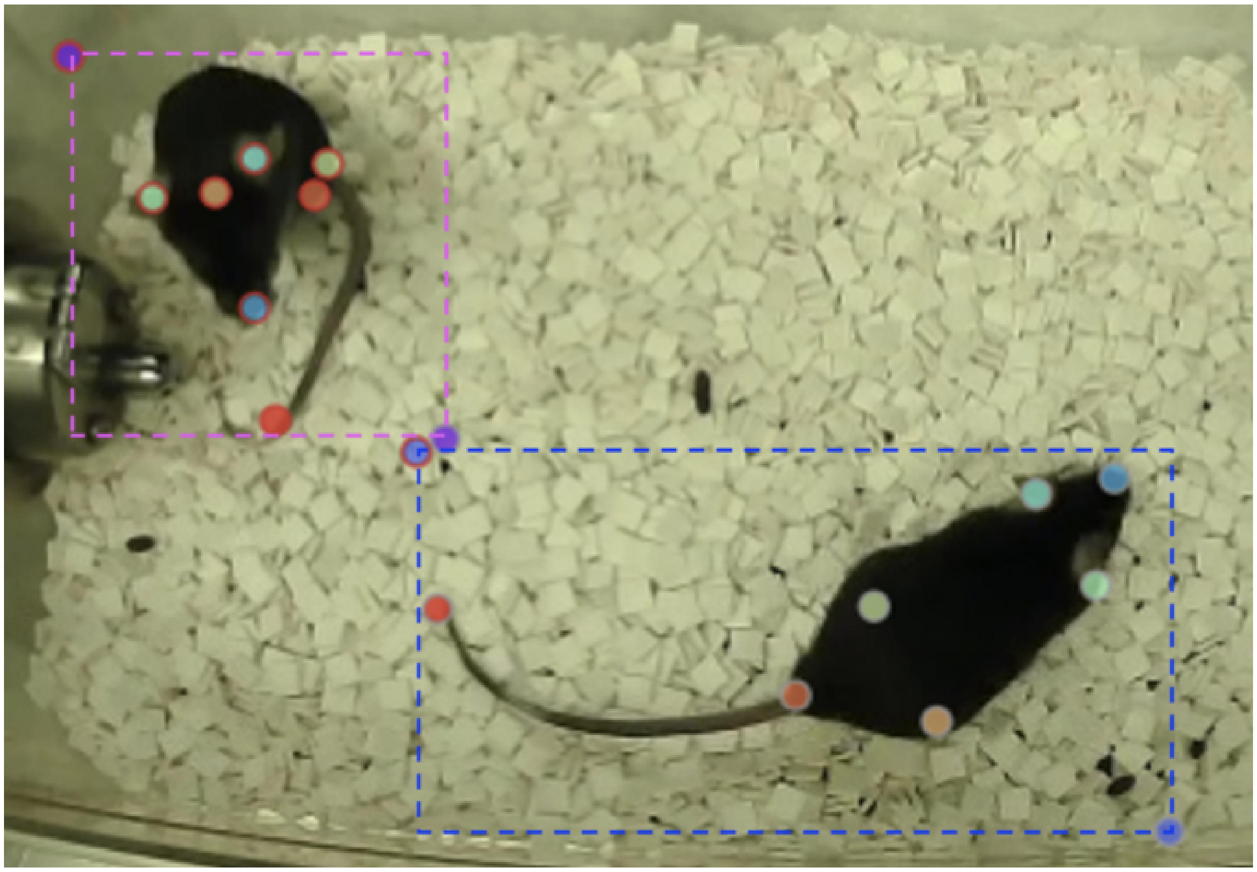
Prediction of Mice locations using Neural Network

Machine Learning HW3
Santosh Reddy Edulapalle
A20501739



All the program code and reports are uploaded to GitHub

Description:

This project aims to train a model to localize the mice in the video. We will first annotate the video frame by frame, then train a Neural Network to predict mice locations and finally evaluate the model.

Key observations:

- The inputs of the model are 7 joints' coordinates.
- The mouse location ground truth (target variable) is defined as the average of the corner coordinates of the bounding box, i.e., center = ((x1+x2)/2, (y1+y2)/2).
- Loss function and evaluate metric are both L2 loss.

Raw video data cannot be used for model training. So, we have been provided with 125 images of random mouse samples in which some images are with a single mouse and some are with 2 mice for which, we need to annotate:

- 7 joints: "nose", "left ear", "right ear", "left hip", "right hip", "tail base", "tail end".
- 2 corners, "left top" and "right bottom"

To help with these annotations, we are using a tool called DeepLabCut. After finishing the annotations of all 125 images, we saved those annotations in a CSV file (**A20501739_CollectedData_annotation.csv**) and uploaded them to the annotation result folder.

Data Analysis:

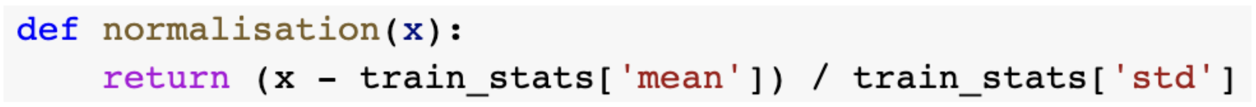
Most of the images don't show proper mice which makes it difficult to annotate all 7 joints perfectly, instead, they are in inappropriate positions making it difficult to annotate all joints. So, we got some NA values in the CSV file.

Since this is a regression kind of problem (here we are not actually locating mice, instead we are predicting their location which is center = ((x1+x2)/2, (y1+y2)/2)),

Steps:

1. Converted all the NA to 0.
2. Renamed the Pandas DF columns so that I can further split the data according to Mouse1 and mouse2.

3. Next, The top 3 rows were information about the type of values that are going to be present in that column and it is a string, for this reason, the entire column is stored as string type. So, I deleted those top three columns. And in the later part, I converted those object-type columns to float type.
4. Now, I added 4 new columns with the target values calculated by taking an average of the top left and bottom right (x,y) for mouse1 and mouse2.
5. Now, I divided the main DF into two sub-DF with mouse1 and mouse2 data separately.
6. If we observe here, our goal is to find the center value(target x,y) for both the mouse, so technically, the coordinates are different but entire features are the same, so, I appended the mouse2 to the mouse1 making it a single DF of mouse data. So, now we can make a single Neural Network to predict the target x,y
7. I'm splitting the data into 70-30% for training and testing. From training data, I further split a small ratio of data for validation which I will use for predicting new results (unseen data)
8. Normalization is an important step to be done before providing input to the neural networks. So, I normalized data using its mean and SD.



```
def normalisation(x):
    return (x - train_stats['mean']) / train_stats['std']
```

9. Later, I popped the target columns (x,y) from the input data set and converted them to NumPy arrays, and stored them in the target dataset.

After completing my data analysis, we will proceed to model building.

Model Building:

I used Keras Functional API for my model because it is much more flexible than sequential API. The functional API can handle models with non-linear topology, shared layers, and multiple inputs or outputs.

Since we have two outputs, (multi-output regression model), I preferred this Functional API to create multiple deep layers specific to each output, as compared to a sequential model where I need to create layers in a sequence.

Here, I created 2 dense layers for each output and used ReLu as an activation function.

The Rectified Linear Unit is the most commonly used activation function in deep learning models. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as f(x)=max(0,x)

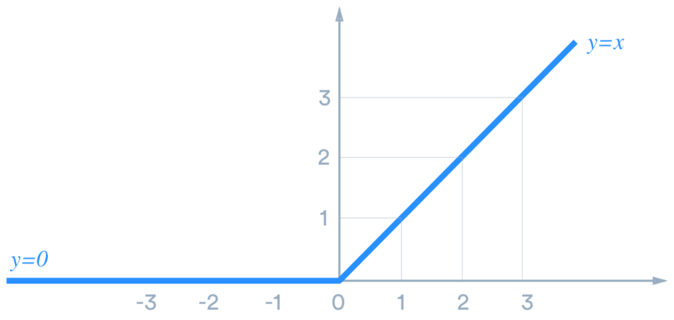


Image Source

Model Compilation:

Initial Learning rate = 0.001

The optimizer I used is Adam (Adaptive Moment Estimation). As this method is really efficient when working with a large problem involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of "gradient descent with the momentum" algorithm and the "RMSP" algorithm.

I used L2 loss for both loss function and evaluation metrics.

L2 loss, also known as Squared Error Loss, is the squared difference between a prediction and the actual value, calculated for each example in a dataset. The aggregation of all these loss values is called the cost function, where the cost function for L2 is commonly MSE (Mean of Squared Errors).

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

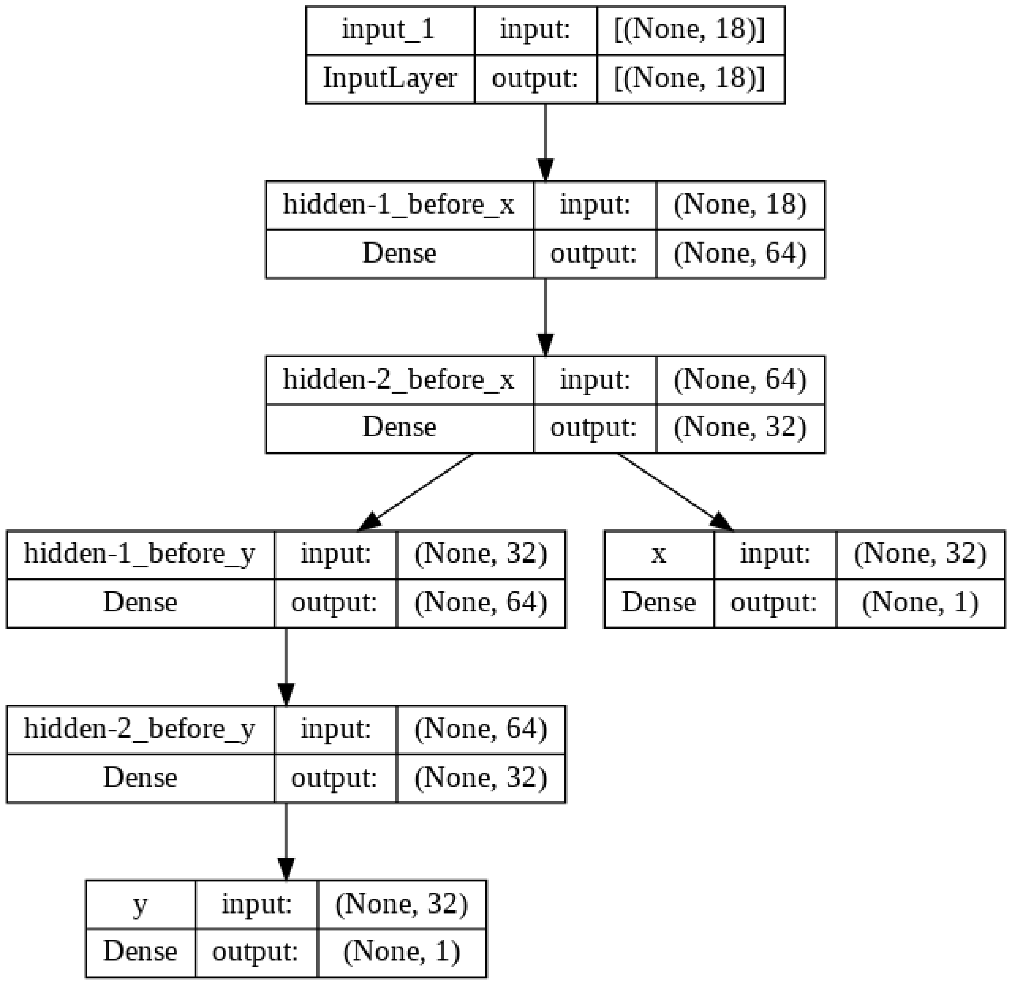
Image Source

Model architecture:

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_8 (InputLayer)           [(None, 18)]         0           []

 hidden-1_before_x (Dense)      (None, 64)           1216        ['input_8[0][0]']

 hidden-2_before_x (Dense)      (None, 32)           2080        ['hidden-1_before_x[0][0]']

 hidden-1_before_y (Dense)      (None, 64)           2112        ['hidden-2_before_x[0][0]']

 hidden-2_before_y (Dense)      (None, 32)           2080        ['hidden-1_before_y[0][0]']

 x (Dense)                      (None, 1)            33          ['hidden-2_before_x[0][0]']

 y (Dense)                      (None, 1)            33          ['hidden-2_before_y[0][0]']

==================================================================================================
Total params: 7,554
Trainable params: 7,554
Non-trainable params: 0
__________________________________________________________________________________________________
```

Here we can see our model's hidden layers and their respective connections.

Model architecture plot:


input_1 | input: | [(None, 18)]
InputLayer | output: | [(None, 18)]
hidden-1_before_x | input: | (None, 18)
Dense | output: | (None, 64)
hidden-2_before_x | input: | (None, 64)
Dense | output: | (None, 32)
hidden-1_before_y | input: | (None, 32)
Dense | output: | (None, 64)
x | input: | (None, 32)
Dense | output: | (None, 1)
hidden-2_before_y | input: | (None, 64)
Dense | output: | (None, 32)
y | input: | (None, 32)
Dense | output: | (None, 1)

This is our Model's architecture plot, we can clearly see the hidden layers and their respective connections to previous layers.
x,y are our output dense layers.

Model Training and Results:

We ran the model for 100 epochs with batch_size =1 and here are our results.

```
loss: 198.94058227539062
x_loss: 111.2213134765625
y_loss: 87.71926879882812
x_rmse: 10.546151161193848
y_rmse: 9.365856170654297
```

Fig: Model Evaluation metrics.

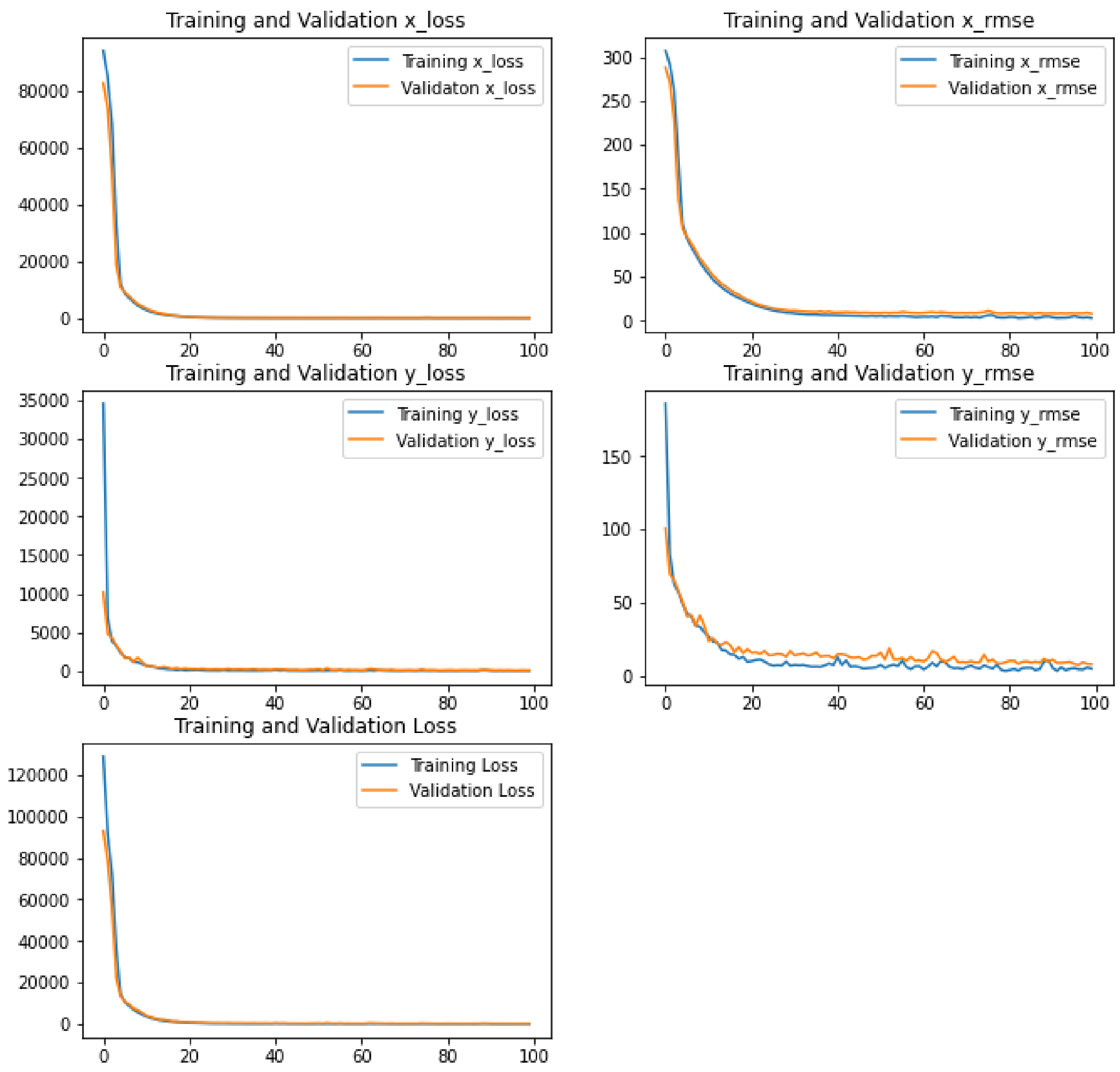
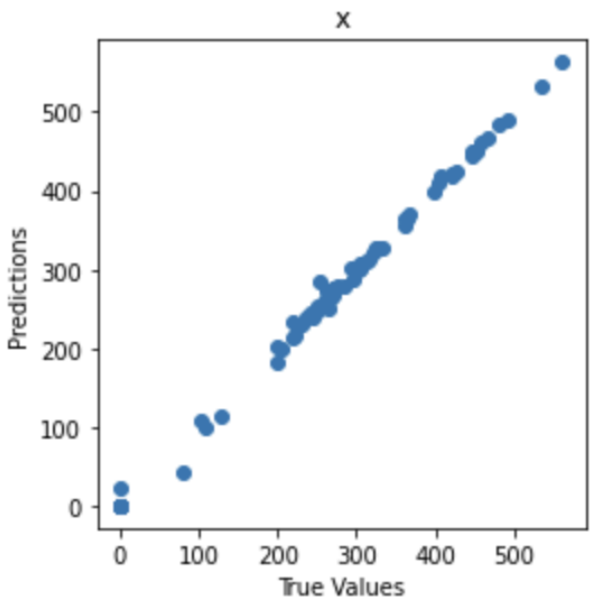


Fig: Plot of model evaluation metrics.

Predictions:

We tried to predict results with our model and our accuracy for predicted results is good.



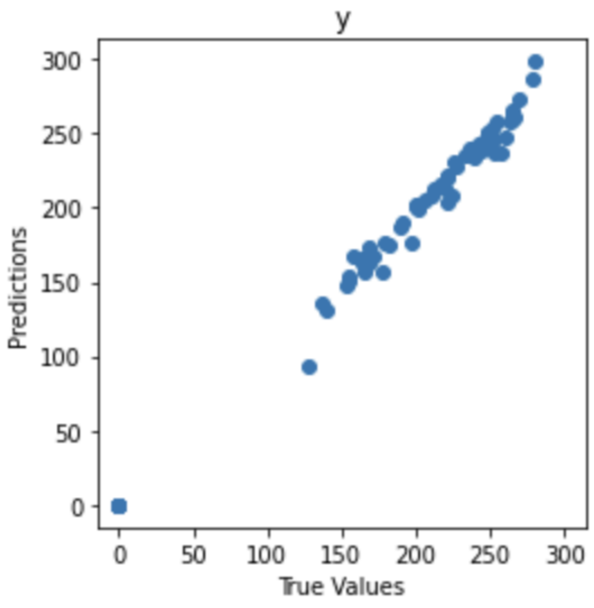
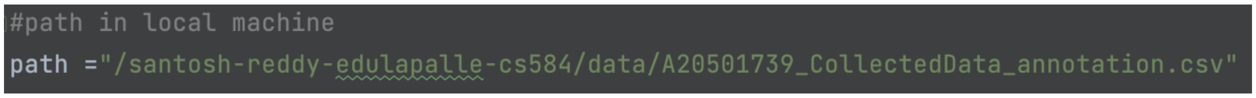


Fig: Predicted vs True values for x and y targets.

Instructions to run the program:

All the program code and reports are uploaded to GitHub

- Please unzip and open the **santosh-reddy-edulapalle-cs584** folder.
- Please open the subfolder **data** to view the annotation dataset named **A20501739_CollectedData_annotation.csv**
- Please open the subfolder **src** and open the notebook **ML_HW3.ipynb.**
- Run the notebook to see the results.



```
#path in local machine
path ="/santosh-reddy-edulapalle-cs584/data/A20501739_CollectedData_annotation.csv"
```

Conclusion:

We have successfully trained a Fully Connected Neural Network to predict the location of mice by taking its 7 joint annotations as inputs. We have got an MSE of around 111 for x and 87 for y, considering our model mean data lies in that range, we assume this is a good value. Also, our prediction results y_hat were good as shown in the graph.

Reference

Thank You