Fingertips Position Estimation of a Robot Hand using 3D images

Intro to Machine Learning Capstone Project

Sangwon Baek[1]

[1]Center for Data Science, New York University, New York, USA

Net ID: swb303 Email: swb303@nyu.edu

GitHub URL: https://github.com/Baeksw98/s-w-b_finger_project

December 11th, 2022

1. Overview

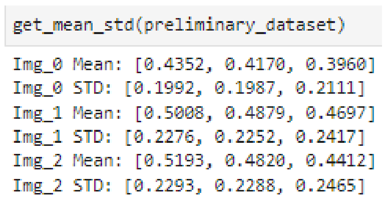
Fingertips Position Estimation is one of the popular tasks in computer vision that require substantial effort to construct accurate predictions. The primary aim of this project was to minimize the root mean squared error loss on unseen data. My model obtained a highly precise model with an RMSE of 0.00222. Although my model may not be the state-of-the-art model for position estimation, the project allowed the room for applying diverse data preprocessing techniques and deep learning modeling architectures explored throughout the course. Therefore, this report contains detailed information on methodologies utilized for fingertip estimations, the experimental results obtained from various image combinations, the discussion of my observations and realizations throughout the research, and the plans for possible implementations in the future to improve the model performance further.

2. Method

2.1 Data Loading & Transformation

The image dataset was initially loaded through the lazy loading class provided by the course instructor to avoid storing the data on the RAM or GPU. This lazy loading class was then modified to return all input values with target coordinates, Y, for the train dataset and all input values without Y for the test dataset. After loading each dataset, I split the dataset into train and validation sets by an 8:2 ratio. Then, each dataset was thrown into a data loader with a batch size of 32 (Shen 2018).

Figure 1. Report of a mean and standard deviation per each RGB Image used for image normalization



```
get_mean_std(preliminary_dataset)

Img_0 Mean: [0.4352, 0.4170, 0.3960]
Img_0 STD: [0.1992, 0.1987, 0.2111]
Img_1 Mean: [0.5008, 0.4879, 0.4697]
Img_1 STD: [0.2276, 0.2252, 0.2417]
Img_2 Mean: [0.5193, 0.4820, 0.4412]
Img_2 STD: [0.2293, 0.2288, 0.2465]
```

I computed the mean and standard deviation of the three RGB images to normalize the image dataset to the appropriate scale. On top of that, depth images were normalized by using the cv2.normalize function with depth/1000. After normalizing the RGB images, I then removed the edges by resizing the images to (240,240) from (224,224) and center cropping (224,224) from resized images. This cropping process helped improve my model’s prediction accuracy because unnecessary pixels that do not contribute much to the fingertip estimation task could be removed.

2.2. Modeling Architecture: Pretrained Model

Instead of creating my own convolutional neural network (CNN) architecture to train images, I implemented transfer learning of pretrained models tested on ImageNet (Dongest 2022). The baseline modeling architecture utilized for this project was Residual Neural Network, also known as ResNet. There exist diverse types of ResNet architecture, such as ResNet-18, ResNet-50, ResNet-101, and ResNet-152. Referring to Pytorch models documentation and the reports in ResNets thesis, ResNets with deeper layers tend to have better accuracy, so ResNet-152 was selected as my primary pre-trained model (He 2016)..

After selecting the model, a further modification was made to the ResNet architecture to fit the given images for training. For instance, the input size of the first convolution layer was changed to match the given image input dimension (3,6,9, and 12) accordingly, and the output size of the last fully connected layer changed from 1000 to 12 to get x,y, and z coordinates of four fingertips only.

2.3 Optimizer & Learning rate

Several types of optimizers were explored to determine the best-performing optimizer for the model, such as Adam, SGD, and AdaGrad. Through trial and error, Adam optimizer performed best among the three optimizers (Kingma 2014). Hence, Adam, with a learning rate of 1e-3, was used to train the model for this project. Next, I tried adding the weight decay term to the model to apply the regularization to the model. However, it aggravated the model's prediction performance, so I eventually had to remove weight decay. In addition to that, I used the StepLR learning rate scheduler, a tool that constantly lowers the learning rate per epoch, to further improve the model performance. Through trial and error, I determined that setting 0.8 Gamma and 1 step size could maximize the model's ability in finding the edges of the prediction improvement.

2.4 Model Training & Early Stopping

The dataset was randomly separated into a train and a validation set with the size of an 8:2 ratio, respectively. After each training epoch, a validation set was used to validate the model. Then I added an early stopping feature to halt the training loop to avoid the model overfitting problem (Kumar 2022)). The essential parameter that had to be tuned for appropriate early stopping was patience. I set the patience to 5 and updated the minimum validation loss per each epoch. By doing so, I designed my model to quit training when the validation loss doesn't go below the minimum validation loss throughout five epochs.

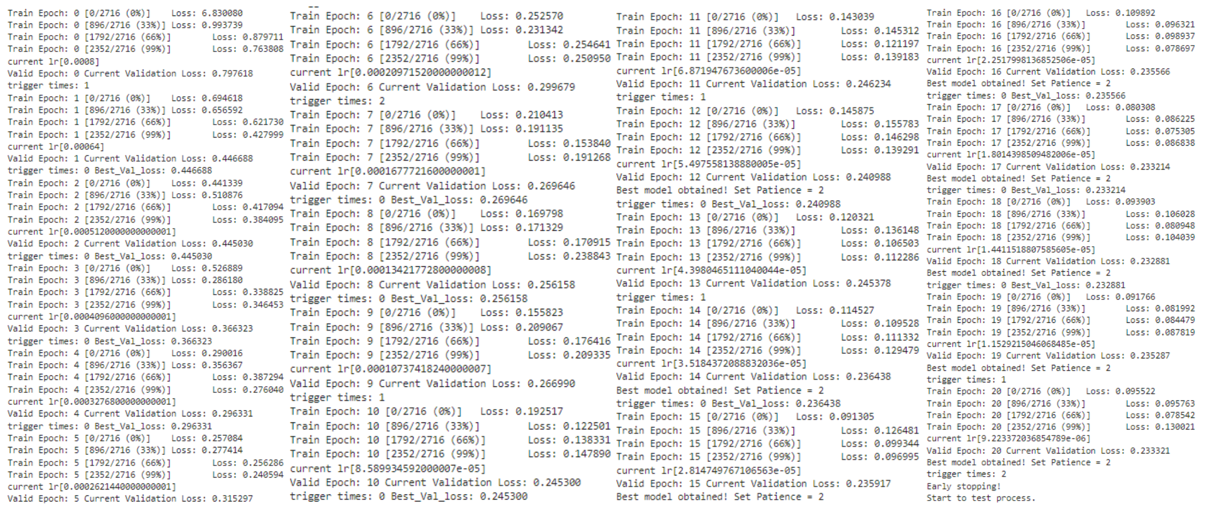
3. Experimental Results

Table 1. Overall quantitative performance of 3D image robot fingertip estimation model for each possible image combinations

Image Combinations	Kaggle RMSE score	Min Validation Loss	Min Train Loss	# of Epochs
1	0.00222	0.002333	0.001300	20
2	0.00481	0.004802	0.001037	27
3	0.00358	0.004063	0.000811	23
Depth	0.00294	0.002906	0.000591	34
1&2	0.00245	0.002544	0.000527	27
1&3	0.00234	0.002443	0.000898	18
2&3	0.00320	0.003474	0.000751	27
1&Depth	0.00223	0.002287	0.000847	21
2&Depth	0.00457	0.004378	0.001063	27
3&Depth	0.00305	0.003299	0.000849	23
1&2&3	0.00242	0.002481	0.000901	23
1&2&Depth	0.00241	0.002303	0.000969	16
1&3&Depth	0.00246	0.002308	0.000618	22
2&3&Depth	0.00320	0.003585	0.000661	19
1&2&3&Depth	0.00259	0.002579	0.005612	24

*Abbreviations: RGB, red green blue; 1, first image; 2, second image; 3, third image; Depth; depth image; RMSE, root mean squared error; Min, minimum

Figure 2. Training process of best performing model with RMSE of 0.00222 (first image only)



```
Train Epoch: 0 [0/2716 (0%)]    Loss: 6.830080
Train Epoch: 0 [896/2716 (33%)] Loss: 0.993739
Train Epoch: 0 [1792/2716 (66%)]        Loss: 0.879711
Train Epoch: 0 [2352/2716 (99%)]        Loss: 0.763808
current lr[0.0008]
Valid Epoch: 0 Current Validation Loss: 0.797618
trigger times: 1
Train Epoch: 1 [0/2716 (0%)]    Loss: 0.694618
Train Epoch: 1 [896/2716 (33%)] Loss: 0.656592
Train Epoch: 1 [1792/2716 (66%)]        Loss: 0.621730
Train Epoch: 1 [2352/2716 (99%)]        Loss: 0.427999
current lr[0.00064]
Valid Epoch: 1 Current Validation Loss: 0.446688
trigger times: 0 Best_Val_loss: 0.446688
Train Epoch: 2 [0/2716 (0%)]    Loss: 0.441339
Train Epoch: 2 [896/2716 (33%)] Loss: 0.510876
Train Epoch: 2 [1792/2716 (66%)]        Loss: 0.417094
Train Epoch: 2 [2352/2716 (99%)]        Loss: 0.384095
current lr[0.0005120000000000001]
Valid Epoch: 2 Current Validation Loss: 0.445030
trigger times: 0 Best_Val_loss: 0.445030
Train Epoch: 3 [0/2716 (0%)]    Loss: 0.526889
Train Epoch: 3 [896/2716 (33%)] Loss: 0.286180
Train Epoch: 3 [1792/2716 (66%)]        Loss: 0.338825
Train Epoch: 3 [2352/2716 (99%)]        Loss: 0.346453
current lr[0.0004096000000000001]
Valid Epoch: 3 Current Validation Loss: 0.366323
trigger times: 0 Best_Val_loss: 0.366323
Train Epoch: 4 [0/2716 (0%)]    Loss: 0.290016
Train Epoch: 4 [896/2716 (33%)] Loss: 0.356367
Train Epoch: 4 [1792/2716 (66%)]        Loss: 0.387294
Train Epoch: 4 [2352/2716 (99%)]        Loss: 0.276040
current lr[0.0003276800000000001]
Valid Epoch: 4 Current Validation Loss: 0.296331
trigger times: 0 Best_Val_loss: 0.296331
Train Epoch: 5 [0/2716 (0%)]    Loss: 0.257084
Train Epoch: 5 [896/2716 (33%)] Loss: 0.277414
Train Epoch: 5 [1792/2716 (66%)]        Loss: 0.256286
Train Epoch: 5 [2352/2716 (99%)]        Loss: 0.240594
current lr[0.0002621440000000001]
Valid Epoch: 5 Current Validation Loss: 0.315297
Train Epoch: 6 [0/2716 (0%)]    Loss: 0.252570
Train Epoch: 6 [896/2716 (33%)] Loss: 0.231342
Train Epoch: 6 [1792/2716 (66%)]        Loss: 0.254641
Train Epoch: 6 [2352/2716 (99%)]        Loss: 0.250950
current lr[0.00020971520000000012]
Valid Epoch: 6 Current Validation Loss: 0.299679
trigger times: 2
Train Epoch: 7 [0/2716 (0%)]    Loss: 0.210413
Train Epoch: 7 [896/2716 (33%)] Loss: 0.191135
Train Epoch: 7 [1792/2716 (66%)]        Loss: 0.153840
Train Epoch: 7 [2352/2716 (99%)]        Loss: 0.191268
current lr[0.0001677721600000001]
Valid Epoch: 7 Current Validation Loss: 0.269646
trigger times: 0 Best_Val_loss: 0.269646
Train Epoch: 8 [0/2716 (0%)]    Loss: 0.169798
Train Epoch: 8 [896/2716 (33%)] Loss: 0.171329
Train Epoch: 8 [1792/2716 (66%)]        Loss: 0.170915
Train Epoch: 8 [2352/2716 (99%)]        Loss: 0.238843
current lr[0.00013421772800000008]
Valid Epoch: 8 Current Validation Loss: 0.256158
trigger times: 0 Best_Val_loss: 0.256158
Train Epoch: 9 [0/2716 (0%)]    Loss: 0.155823
Train Epoch: 9 [896/2716 (33%)] Loss: 0.209067
Train Epoch: 9 [1792/2716 (66%)]        Loss: 0.176416
Train Epoch: 9 [2352/2716 (99%)]        Loss: 0.209335
current lr[0.00010737418240000007]
Valid Epoch: 9 Current Validation Loss: 0.266990
trigger times: 1
Train Epoch: 10 [0/2716 (0%)]   Loss: 0.192517
Train Epoch: 10 [896/2716 (33%)]        Loss: 0.122501
Train Epoch: 10 [1792/2716 (66%)]       Loss: 0.138331
Train Epoch: 10 [2352/2716 (99%)]       Loss: 0.147890
current lr[8.589934592000007e-05]
Valid Epoch: 10 Current Validation Loss: 0.245300
trigger times: 0 Best_Val_loss: 0.245300
Train Epoch: 11 [0/2716 (0%)]   Loss: 0.143039
Train Epoch: 11 [896/2716 (33%)]        Loss: 0.145312
Train Epoch: 11 [1792/2716 (66%)]       Loss: 0.121197
Train Epoch: 11 [2352/2716 (99%)]       Loss: 0.139183
current lr[6.871947673600006e-05]
Valid Epoch: 11 Current Validation Loss: 0.246234
trigger times: 1
Train Epoch: 12 [0/2716 (0%)]   Loss: 0.145875
Train Epoch: 12 [896/2716 (33%)]        Loss: 0.155783
Train Epoch: 12 [1792/2716 (66%)]       Loss: 0.146298
Train Epoch: 12 [2352/2716 (99%)]       Loss: 0.139291
current lr[5.497558138880005e-05]
Valid Epoch: 12 Current Validation Loss: 0.240988
Best model obtained! Set Patience = 2
trigger times: 0 Best_Val_loss: 0.240988
Train Epoch: 13 [0/2716 (0%)]   Loss: 0.120321
Train Epoch: 13 [896/2716 (33%)]        Loss: 0.136148
Train Epoch: 13 [1792/2716 (66%)]       Loss: 0.106503
Train Epoch: 13 [2352/2716 (99%)]       Loss: 0.112286
current lr[4.398046511040044e-05]
Valid Epoch: 13 Current Validation Loss: 0.245378
trigger times: 1
Train Epoch: 14 [0/2716 (0%)]   Loss: 0.114527
Train Epoch: 14 [896/2716 (33%)]        Loss: 0.109528
Train Epoch: 14 [1792/2716 (66%)]       Loss: 0.111332
Train Epoch: 14 [2352/2716 (99%)]       Loss: 0.129479
current lr[3.518437208832036e-05]
Valid Epoch: 14 Current Validation Loss: 0.236438
Best model obtained! Set Patience = 2
trigger times: 0 Best_Val_loss: 0.236438
Train Epoch: 15 [0/2716 (0%)]   Loss: 0.091305
Train Epoch: 15 [896/2716 (33%)]        Loss: 0.126481
Train Epoch: 15 [1792/2716 (66%)]       Loss: 0.099344
Train Epoch: 15 [2352/2716 (99%)]       Loss: 0.096995
current lr[2.81474976710656З3e-05]
Valid Epoch: 15 Current Validation Loss: 0.235917
Best model obtained! Set Patience = 2
Train Epoch: 16 [0/2716 (0%)]   Loss: 0.109892
Train Epoch: 16 [896/2716 (33%)]        Loss: 0.096321
Train Epoch: 16 [1792/2716 (66%)]       Loss: 0.098937
Train Epoch: 16 [2352/2716 (99%)]       Loss: 0.078697
current lr[2.251799813685250бe-05]
Valid Epoch: 16 Current Validation Loss: 0.235566
Best model obtained! Set Patience = 2
trigger times: 0 Best_Val_loss: 0.235566
Train Epoch: 17 [0/2716 (0%)]   Loss: 0.080308
Train Epoch: 17 [896/2716 (33%)]        Loss: 0.086225
Train Epoch: 17 [1792/2716 (66%)]       Loss: 0.075305
Train Epoch: 17 [2352/2716 (99%)]       Loss: 0.086838
current lr[1.801439850948200бe-05]
Valid Epoch: 17 Current Validation Loss: 0.233214
Best model obtained! Set Patience = 2
trigger times: 0 Best_Val_loss: 0.233214
Train Epoch: 18 [0/2716 (0%)]   Loss: 0.093903
Train Epoch: 18 [896/2716 (33%)]        Loss: 0.106028
Train Epoch: 18 [1792/2716 (66%)]       Loss: 0.080948
Train Epoch: 18 [2352/2716 (99%)]       Loss: 0.104039
current lr[1.4411518807585605e-05]
Valid Epoch: 18 Current Validation Loss: 0.232881
Best model obtained! Set Patience = 2
trigger times: 0 Best_Val_loss: 0.232881
Train Epoch: 19 [0/2716 (0%)]   Loss: 0.091766
Train Epoch: 19 [896/2716 (33%)]        Loss: 0.081992
Train Epoch: 19 [1792/2716 (66%)]       Loss: 0.084479
Train Epoch: 19 [2352/2716 (99%)]       Loss: 0.087819
current lr[1.1529215046068485e-05]
Valid Epoch: 19 Current Validation Loss: 0.235287
Best model obtained! Set Patience = 2
trigger times: 1
Train Epoch: 20 [0/2716 (0%)]   Loss: 0.095522
Train Epoch: 20 [896/2716 (33%)]        Loss: 0.095763
Train Epoch: 20 [1792/2716 (66%)]       Loss: 0.078542
Train Epoch: 20 [2352/2716 (99%)]       Loss: 0.130021
current lr[9.223372036854789e-06]
Valid Epoch: 20 Current Validation Loss: 0.233321
Best model obtained! Set Patience = 2
trigger times: 2
Early stopping!
Start to test process.
```

As can be seen in Table 1, the lowest RMSE score of 0.00222 was obtained for the first RGB (top-down view) image only and the second lowest RMSE score of 0.00223 was obtained for the first RGB and depth images combined. Through trying multiple combinations of images, a first RGB image with around 20 epochs of training can guarantee a highly accurate finger estimation among all possible image combinations.

As shown in Figure 2, the validation steadily decreases until Epoch 10, then slows down as it approaches its peak. Here I intentionally set the threshold for the best model as 0.24 and changed the patience to 2 when achieved. This implementation allowed my model to only linger in the training loop for a short time because I could observe that the model stopped improving after attaining a particular RMSE score. Another interesting observation was that the model required some time to reflect the acquired validation loss when applied to the test set. Perhaps this observation could be further explored to better understand the learning and testing process of deep learning models.

4. Discussion

Since this project was open-ended with a single goal of obtaining the lowest RMSE score, there existed many different approaches that we could have applied to get the best-performing image estimation model. To get the finest model, I also went through various trial and error processes to identify the right hyperparameter combinations for obtaining the minimum RMSE score.

Although rotations and flips are well-known as popular data augmentation techniques in the image classification task, these techniques didn't seem feasible in the image estimation task because a slight variation to the image results in coordinates shifts that could significantly aggravate the RMSE score. To test this intuition, I randomly rotated, horizontally, and vertically flipped the images. As shown through a decrease in RMSE score, I decided not to implement the above transformations.

Another essential procedure was identifying the adequate scale of target variables. This step was necessary because the model wouldn't train and converge well when the scale was too low. Hence, the target was multiplied by 100 before processing into a loss function. To offset this effect for test predictions, I then divided by 100 when computing the predictions into a submission file. Indeed, this linear scaling helped the model learn and accurately capture the loss more efficiently so that the desired RMSE score could be achieved with fewer epochs.

Model overfitting refers to when the model starts memorizing the noise and pattern that occurs when the model gets overly trained on the dataset and is too complex (Carremabs 2018). Such an overfitting problem can be highly detrimental to the model performance because the model becomes incompetent against unseen data. Therefore, identifying an adequate strategy to minimize the overfitting issue was vital for attaining stable model performance in the project (Kumar 2022). My model could overcome such overfitting issue by implementing early stopping during the model training process and applying a lower learning rate per each epoch to slower the training process. These implementations seemed to be the critical characteristic that made my model have superior estimation performance than the models' of many other classmates.

5. Future Work

Although I could obtain fairly accurate RMSE results from this project, there still exists room for further exploration if more time is permitted. One possible adaptation could be exploring the compatibility of new state-of-the-art models. For example, a pretrained model may perform better than the ResNet, the main structure of my model. Another possible option could be identifying more suitable hyperparameters by trying the following options:

1. trying different learning rates of the optimizer
2. increasing the patience number for an early stopper
3. trying different gamma for the learning rate scheduler
4. applying different normalization and transformation techniques to data.

Finally, this fingertip estimation problem looks similar to the hand pose estimation problem, which is one of the most popular topics in computer vision. One of their techniques is to create a 2D/3D ground truth heat map and use that heatmap as the reference point to train the images in a supervised manner. Unfortunately, since this proposed solution is out of the scope of our course, I aim to explore these new methodologies for image estimation as a follow-up project.

References

1. Carremabs, Bert. “Handling Overfitting in Deep Learning Models.” *Medium*, Towards Data Science, 23 Aug. 2018, https://towardsdatascience.com/handling-overfitting-in-deep-learning-models-c760ee047c6e. Accessed 10 Dec. 2022.
2. Donges, Niklas. “What Is Transfer Learning? Exploring the Popular Deep Learning Approach.” *BuiltIn*, BuiltIn, 25 Aug. 2022, https://builtin.com/data-science/transfer-learning. Accessed 10 Dec. 2022.
3. He, Kaiming, et al. “Deep Residual Learning for Image Recognition.” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, https://doi.org/10.1109/cvpr.2016.90.
4. Kingma, Diederik P, and Jimmy Lei Ba. “Adam: A Method for Stochastic Optimization.” *ArXiv*, https://doi.org/https://doi.org/10.48550/arXiv.1412.6980. Accessed 10 Dec. 2022.
5. Kumar, Bijay. “PyTorch Early Stopping + Examples.” *PythonGuides*, PythonGuides, 22 Mar. 2022, https://pythonguides.com/pytorch-early-stopping/. Accessed 10 Dec. 2022.
6. Shen, Kevin. “Effect of Batch Size on Training Dynamics.” *Medium*, Mini Distill, 20 June 2018, https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e. Accessed 10 Dec. 2022.