University of Cape Town

EEE4114F

Digital Signal Processing

13 May 2019

# *Neural Networks Applied to Position Estimation from Stereoscopic Images*

| | |
|---|---|
| Stefan Dominicus | DMNSTE001 |
| Jonathan Oehley | OHLJON001 |

# Table of Contents

## Abstract

*This report presents an investigation into the application of neural networks for the purpose of 3D position estimation of keypoints in pairs of stereoscopic infrared images. A labeled dataset was generated using a Leap Motion Controller, and two learning algorithms were tested: support vector regression, and convolutional neural networks. SVR was poorly suited to the problem, and could not achieve accurate enough results. CNNs produced satisfactory results, after a number of hyperparameter configurations were tested. A git repository of project resources is available here.*

## Introduction

Leap Motion Inc. develop and manage a hardware, firmware and software suite that allows the real-time tracking of a user's hands - both position and gestures. The Leap Motion Controller is the hardware component of this suite which consists of a pair of infrared cameras in a small USB device. Leap Motion encourages developers to use and implement Leap Motion Technology within their projects, and provide a rich API in a variety of languages to give users comprehensive, high- and low-level control over the device.

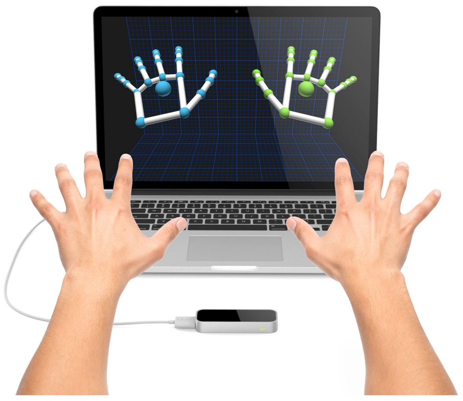


*Figure 1: Image of a Leap Motion Controller connected to a laptop, displaying the positions of the hands in the device's FOV. [1]*

The Controller consists two cameras 40mm apart, which record synchronised stereo images of objects within the visual range of the device. This data is then streamed to a computer via USB, where the Leap Motion Service, the software portion of the suite, which "applies advanced algorithms to the raw sensor data" to extract details of the hands and tools in the images [2]. Leap Motion utilise a variety of conventional image processing techniques to process the stereo image data recorded by its cameras, and thereby track the position of hands in the camera's FOV.

This report describes the investigation into using neural networks and machine learning techniques to produce similar hand tracking data from the stereo images provided by the Leap Motion Controller. Firstly, the problem statement and scope of the investigation are presented. Thereafter, the process of recording the labelled dataset from the Leap Motion Controller is explained, followed by a discussion on the methods and metrics used to evaluate the performance of the neural networks. The two approaches used are put forward, followed by a presentation of their results. These are discussed and analysed before concluding on the success and results of the investigation as a whole.

# Problem Specification

## Problem Statement

This investigation aimed to determine if a machine learning algorithm could be used to estimate the 3D position of key points within a pair of stereoscopic images.

## Background

Machine learning has become an incredibly popular choice for a number of image processing applications, largely due to the increased ubiquity of high-performance GPU implementations such as Google's Colab. An application which is closely related to this problem is that of depth map estimation from stereoscopic images. The author in [3] presented a fully convolutional neural network able to compute a depth map over an entire image, which illustrates that machine learning approaches are feasible for depth estimation applications.

Contrary to the recent hype regarding machine learning, the author in [4] presented an argument for the continued use of traditional image processing techniques, claiming that there us a wealth of information available in the image structure which human-designed methods can explicitly take advantage of. This idea is somewhat supported by [5] where a DCT transform was used as a pre-processing / feature-extraction step prior to passing the images through a CNN.

One significant challenge when reviewing the literature available on the subject is the bias towards machine learning applied to classification tasks, as opposed to regression tasks. Despite this, examples of both SVR and CNN regression were available, albeit in limited quantity [6][7].

An important development in the field has been the advent of batch learning, and particularly mini-batch learning methods. These methods allow the optimization process to iterate through many more smaller gradient descent steps, compared to non-batch methods. This results in a more robust learning process, and, particularly in the case of image processing, a significant reduction in the memory requirements for training [8].

## Scope

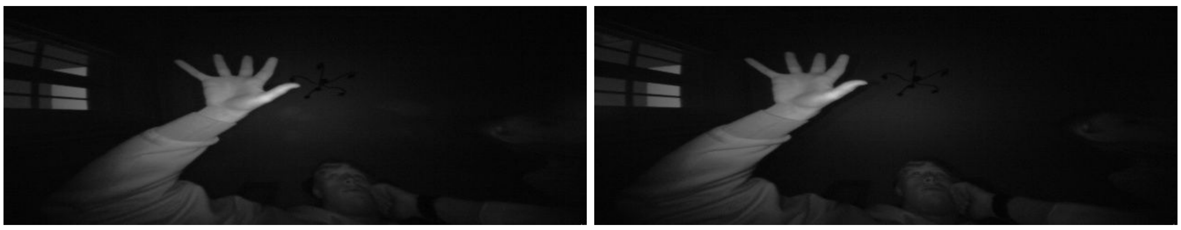
- This investigation considered two types of machine learning: SVR and CNN. No other learning algorithms were tested.
- The dataset used for training and evaluating the algorithms contained only images from the Leap Motion cameras. Each image contained only a single, unobstructed hand.
- For a single hand, six key points were identified: palm center, and the five fingertips. Each of these key points has a 3D position vector computed by the Leap Motion Controller, which is assumed to be ground truth. The machine learning predictions therefore consisted of 18 real-valued numbers representing the (x,y,z) coordinates of each key point.
- This investigation considered each frame in isolation, and did not use any form of memory to inform a current prediction based on prior predictions.

## Constraints

- This investigation was performed for a university project, and as such the time dedicated to the task was limited by other course work.
- The models were initially trained on laptop CPUs, and later the CNN model was ported to train on a Google Colab GPU. No other optimized hardware was used to increase training speed.
- All models tested in this investigation were constructed and trained from scratch. Despite the advantages of transfer learning, the additional complexity of its implementation was beyond the scope of this investigation.
- A dataset of approximately 4500 labeled pairs of images was generated, and used to train all the models discussed in this investigation. Data augmentation was not used due to the complications of augmenting the labels as well.
- The authors had no prior experience in this field. All design decisions were informed through the literature included in this report (both referenced and in the bibliography), and through consultation with Prof. A. Mishra.

# Data Generation

Since this investigation was focused on a very specific data source (the Leap Motion Controller), and there aren't any publically available datasets of this type, a new labeled dataset first had to be generated. This was done using python, and interacting with the Leap Motion Controller via the device's developer API. This allowed access to the raw images captured by the device each frame, as well as the wealth of tracking information providing the coordinates of each bone and joint in each hand present in the field of view. The coordinates of the six points of interest to this investigation were saved to a CSV file, which would act as the labels for the raw images, which were also saved.



| **FrameID** | **Right** | **PalmX** | **PalmY** | **PalmZ** | **...** | **PinkyX** | **PinkyY** | **PinkyZ** |
|---|---|---|---|---|---|---|---|---|
| 133684 | TRUE | 94,1 | 176,6 | -77,8 | | 152,7 | 169,8 | -132,7 |

*Figure 2: An example frame recorded from the Leap Motion Controller. Each frame consists of a pair of stereoscopic infrared images as shown, and a set of labels calculated by the device's API. The dataset used in this investigation consisted of approximately 4500 such frames.*

This process was set to run as an event listener callback, which triggered on each time the Leap Motion API produced the next frame. This was typically at a rate of approximately 50fps. A sample dataset, and all of the python code used are included in the project git repository. Above is an example of a single frame, which consists of a pair of images, and a set of labels. The dataset used for the remainder of this investigation consisted of approximately 4500 such frames.

# Evaluation Criteria

The performance of each model was evaluated in three ways. Each time a model was to be tested, a batch of predictions was obtained, based on unseen frames from the dataset. The model's predictions were then compared to the labels for the same test data, in order to determine the pointwise error for each frame. The set of labels and pointwise errors were saved as a CSV for each model, which was later used to plot the error distributions discussed below. Plotting was done using this Google Colab python notebook. Note that while the models were all trained using a Mean Squared Error objective function, the error distribution comparisons evaluate the error (not MSE).

## Overall Accuracy

This evaluation simply considers the pointwise error on the set of test frames, and determines the spread. This is useful as a high-level measure of how accurate the model's predictions are, but does not provide any insight into what aspects of the problem the model may be struggling with. Nonetheless, it is a useful measure of performance for quantifying the model's accuracy on any given point.

## Accuracy by Axis

Perhaps the most critical aspect of this problem is that of depth estimation. Afterall, feature position in 2D is relatively straightforward given a 2D image. The fact that this problem includes a third dimension in the output, while still relying on 2D input data, results in the z-axis prediction accuracy being particularly important for determining how well the model has actually learned the desired behaviour. For this reason, each model's error was evaluated per axis, giving an indication of how well the model could predict position in each direction.

## Palm vs Fingers Accuracy

The final evaluation criterion aimed to determine how the model's accuracy varied as a function of feature size. This evaluation compared the pointwise error in predicted palm position with that of each of the fingertips. The goal of this comparison was to determine if feature size played a role in prediction accuracy. I.e. if a model is more accurate when predicting the position of larger features (palm) versus smaller features (fingertips).

# Problem Approach

## Support Vector Regression (SVR)

Support Vector Machines (SVM) are a class of supervised learning methods that are used for classification and regression. SVMs are mostly seen and often applied in classification problems where the SVM attempts to find a separating line, or hyperplane in higher dimensions, between the data of the classes. The application of SVMs to solving such a problem is called Support Vector Classification (SVC). The method of SVC can be extended for use in solving regression problems (SVR). [7]

The package chosen to implement SVR was scikit-learn. This provided an easy install through the use of pip. The next stage taken was an attempt to develop a trained SVR model. No variation of hyperparameters was considered initially, just a proof of concept was constructed.

Initially unnoticed, it was found that scikit-learn's implementation of the SVR algorithm scales with a complexity of between O(n^2) and O(n^3) with regard to feature set. The implication of this is that the SVR is not well suited to image processing, which has a large number of inputs both in size and quantity. Additionally, SVR doesn't support any form of batch processing or a partial fit method. On top of this, the SVR algorithm has a single fixed output. Therefore, it was required to wrap the algorithm in scikit-learn's Multiple Output Regressor class. This does mean that for every output a new SVR model is trained i.e. 18 models will be needed to train to generate the 18 coordinates of the hand to track.

Thus it can already be seen that SVR is not well suited to this kind of problem, however for a comparative measure, the development of such a network continued. Upon implementation it was found that due to the large size of the input that it was not possible to train the SVR model on the full 9000 image dataset. This was due to the fact that the algorithm required over 16Gb of RAM to generate the model which the computer on which the algorithm was run could not provide. Through an iterative process, it was found that the maximum training set possible was 15% of full dataset. This is a very small training set only 600 images and the produced results reflect this fact.

A SVR implementation was attempted on Google Collab, but the RAM constraints were only more restrictive and hence training was done locally. Additionally, SVR does not support GPU  or TPU acceleration and thus Collab would not have provided any significant speed-up either.

Very little information was found on how to make informed decisions on the hyperparameters of the SVR model. While there is little literature on how to manually tune SVC class problems, even these present a trial and error approach. Additionally, it was not immediately clear what effect each of the hyperparameters of the model would adjust. However, in literature it was found that the most often varied parameters are the gamma and penalty term, C, in orders of magnitude [10]. Therefore, a number of models were trained by individually varying these parameters by factors of ten around their default values to attempt to improve the model to some degree.

## Convolutional Neural Network (CNN)

All the CNN models tested in this investigation were implemented using Keras, a python wrapper for TensorFlow [11]. The initial tests were performed on a PC, which allowed for much of the initial debugging to be done quickly on a local machine. However, the resource limitations soon caused problems with training speed and memory requirements. In an effort to get by with local resources, the models were set to train using batch learning, and used a custom implementation of Keras's sequence generator to limit the number of images in RAM at one time [12]. This was successful, and allowed small models to be trained in less than a few hours.

Despite early successes with the small, locally trained models, the local computational limits were soon reached, and more processing power was needed before the network complexity could be increased. The model generation code was ported to run on Google Colab, using GPU hardware acceleration. This resulted in a training speed increase in excess of 100x, compared to local training, and allowed for much more complicated networks to be trained and evaluated in a matter of minutes.

To take full advantage of Colab's speed, this notebook was created to streamline the processes of designing, training, saving, importing and evaluating various models. For the purposes of this investigation, six different model structures were tested, each one a simple variation of the others, allowing the hyperparameters to be tuned systematically. So that a fair comparison could be made, all models were trained and evaluated on the same datasets, and used the same optimizer and number of epochs.

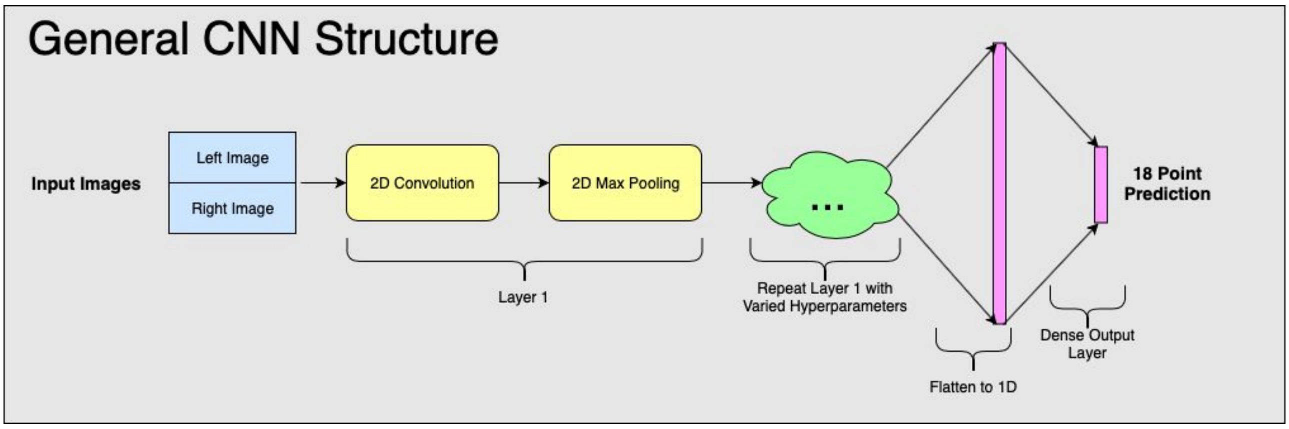


*Figure 3: General structure used for each of CNN models tested. Each model varied the number of layers, the number of kernels per layer, and the degree of max pooling. The output of the last convolutional layer was then flattened, and densely connected to the 18 output nodes.*

The figure above shows the basic layer structure used in all six of the models. Each model varied either the number of layers, the number of convolutional kernels per layer, or how aggressive the max pooling was. Since the Colab notebook was integrated with Google Drive, each of the models could easily be saved once trained, along with any evaluation results. Each model could also be recalled for comparison at a later stage if necessary.

# Results and Analysis

## Support Vector Regression (SVR)

The results of the SVR models were less than optimal and significantly worse than those produced by the CNN architecture which are present below. A numerical summary of the statistical analysis of the performance of the SVR models under parameter variation is shown in Appendix C. The nature of the results meant that the only evaluation criteria from those laid out above, that is sensible to apply to the SVR model results is the overall accuracy of the models. For comparative purposes, the overall error distribution of one of the models is presented in the Figure below.

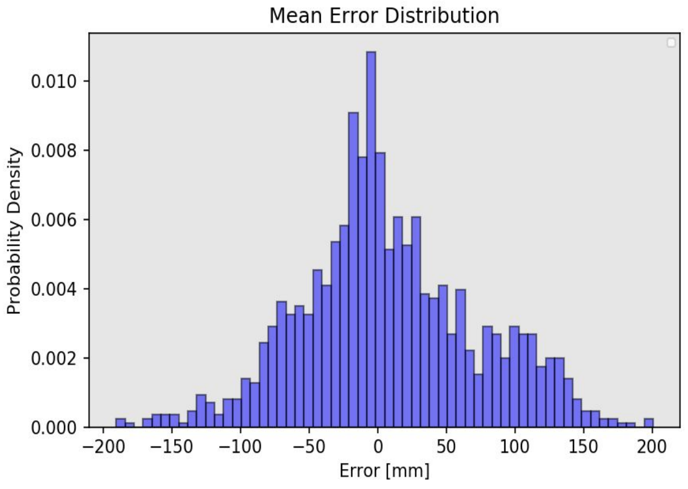


*Figure 4: Overall Error Distribution of the most precise SVR model (model 9)*

On average the SVR models have little variation between the models which were trained using different hyperparameters. On average the models produced a mean estimation error of 7.43 mm across the prediction of the coordinates of all points , however the mean standard deviation of these predictions was 65.63 mm. In a cuboidal tracking space with length of 600mm, this is a very significant error which shows that SVR is not the optimal solution to this problem.

There are a number of likely reasons why the results achieved by the SVR models were poor. The first and most definite reasons is that the training set size was of a very small size for a machine learning problem such as this one. Additionally, since the network also only sees the training data once, there is no ability of the network to make incremental changes to the weights or structure of the network.

The hyperparameter variation that was done in training the models did not produce a significant differences. There could be a number of reasons for this. Firstly, that the training set size does not allow for the effect of the hyperparameters to properly present themselves. Additionally, the naive variation of these parameters may have been suboptimal, meaning that either the parameters may not have been varied enough or that additional or different parameters should have been put under test. Either way, the time constraints of this project prevent additional investigation into this issue, however, techniques and line of improvement are laid out in detail in the Further Development section.

## Convolutional Neural Network (CNN)

The problem approach section explained the way in which each of the models was iteratively designed and tested. This section will present the results of the model which performed the best over the three evaluation criteria. This model consisted of 4 convolutional layers, with 16, 32, 64, & 128 3x3 kernels respectively, and used 2x2 Max Pooling after each convolutional layer. A detailed summary of the network structure is included in Appendix B.

The figure below shows the error histograms related to each evaluation criteria. These results were plotted for each model and used to determine which had performed the best. The table below lists the mean and standard deviation of each histogram shown. The scale of these errors is remarkably small when compared to the working volume of the Leap Motion Controller (approximately 600x600x600mm [2]).

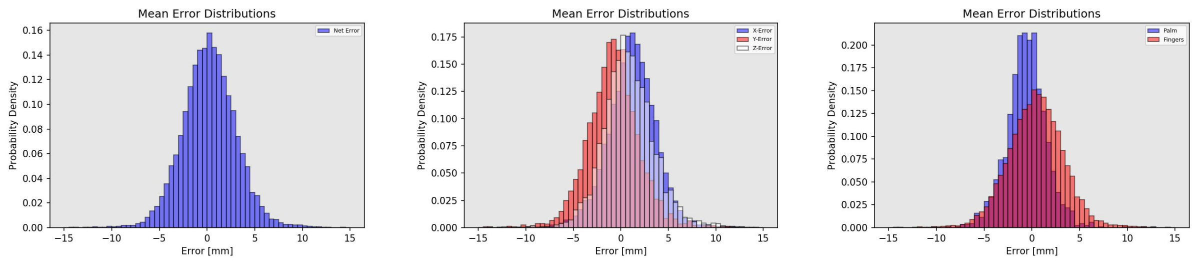


*Figure 5: Error analysis of the best performing prediction model, according to the three evaluation criteria. The histograms provide an intuitive interpretation of the model's error characteristics. The table below lists the mean and standard deviation of each histogram, for a more quantitative analysis.*

| | Overall [mm] | X [mm] | Y [mm] | Z [mm] | Palm [mm] | Fingers [mm] |
|---|---|---|---|---|---|---|
| **Mean** | 0.234 | 1.11 | -0.775 | 0.831 | -0.581 | 0.420 |
| **Standard Deviation** | 3.18 | 3.46 | 2.90 | 2.77 | 2.45 | 3.28 |

*Table 1: Summary of statistical error metrics for the best model's performance. The mean and standard deviation are listed for each of the histograms shown in the previous figure.*

The model's overall error presents a very promising result. The predictions are very accurate, as shown by the small mean error of 0.234 mm per coordinate. The model's precision is also very impressive - the histogram shows that the majority of predictions are within 5 mm of the correct value.

The X, Y, Z error analysis presents a remarkable trend - that the z-axis predictions exhibit the tightest error spread, and a mean deviation similar to that of the x- and y-axes. As discussed in the evaluation criteria section, this result is particularly important for understanding how well the model has been applied to the problem, since a core part of this problem is depth estimation. The results for this model show that it is no worse at predicting depth than it is at predicting xy-position.

The final histogram is a comparison of palm position accuracy vs fingertip accuracy, according to the third evaluation criterion. This result illustrates quite well that the concern regarding feature size and prediction accuracy is well founded. The histogram and statistical metrics suggest that the model was able to make more precise predictions for the palm position than for the fingertip positions.

# Further Development

There are a number of ways in which this investigation could be improved. These are discussed in two sections: improvements to the way this investigation was carried out, and further experiments or approaches which could be used to extend the scope of the investigation.

## Methodological Improvements

Within the SVR development section, while it was found that SVR was ill-suited to the task at hand, there were a number of ways that the model generated could have been improved. First, and foremost to improving the results of the estimator would be to run it on a platform with enough RAM to run the training with a larger dataset. Additional, with the size of the feature set, linear-SVR may be able to achieve better results simply due to the fact that it has O(n) complexity instead of between O(n^2) and O(n^3).

With the hyperparameter variation that was described in the problem approach above, there was additional room for varying the gamma and C parameters. As can be seen in the results described, the variation in these parameters produced differences in the performance of the predicted outputs. Therefore, it follows that there is are more optimal values of these parameters. There were also additional free parameters available to vary in the SVR function. These were not varied due to a lack of information on both the range of values to vary them over but also the effect they would have on the output. One of these such parameters that is described in scikit-learn's documentation is epsilon [13]. Additionally, the kernel choice will significantly affect the output of the estimator which was not changed from the default 'rbf' kernel.

It was found late in the project development that scikit-learn has functions to tune the hyperparameters of an estimator. This, however, essentially requires the estimator to be trained with each combination of hyperparameters chosen - a lengthy process given the size of the feature set for this project. Such a function would likely be the quickest way to search the hyperparameter space for an efficient and significantly more optimal combination of hyperparameters.

Convolutional neural networks presented very promising results, and their performance was highly satisfactory given the scope and limitations of this investigation. However, given the knowledge gained to this point, two improvements are suggested to reduce the prediction error. First, a larger dataset could be generated for training the model. This is a fairly standard desire in supervised machine learning, but bigger datasets are not always available. However, this problem is unique in that labeled data can be generated on demand. The second suggested improvement is to increase the model complexity. It was found in testing that models with more layers, and a higher layer complexity, usually performed better than simpler models. The main limitation here is the additional computational resources required, although this could be mitigated by building the model to run using Google's TPU hardware acceleration, which should provide a performance improvement over GPUs.

## Future Investigation

The results of this investigation could be continued in any number of directions. Two possibilities are discussed here, which the authors found particularly exciting, but which were beyond the scope of this investigation.

### Real-Time Prediction

The purpose of this extension is to determine if the prediction models generated in this investigation are computationally efficient enough to serve predictions at a comparable rate to the Leap Motion Controller (approximately 50fps). So far, prediction tests run using Google Colab GPUs would suggest that this rate of predictions is feasible, but these tests could also be considered to be under 'ideal' conditions - the images are already stored in RAM, and the model runs a batch of uninterrupted predictions, as opposed to handling a prediction as an event interrupt. The significance of these differences can only be speculated at this point, but further investigation may yield interesting results.

### Real-Time / Adaptive Learning

The desire to investigate adaptive learning stems from the fact that the Leap Motion Controller generated a labeled dataset in real-time. It is proposed that future investigations may try to implement a learning model which trains in real-time, while simultaneously, or at least pseudo simultaneously, serving predictions. This may be able to provide an interactive experience of how the model learns, as it may be able to watch as the model gets progressively better and better as more real-time data is acquired and learned. The additional complexities of this approach are far beyond the scope of this investigation, and cannot be commented on. However, advances in adaptive learning techniques will likely have an impact much more far-reaching than what has been presented in this investigation so far.

# Conclusion

This report began by introducing the context of the investigation before defining the problem statement. Thereafter, a summary of relevant and supporting literature is presented before the scope and constraints of the investigation are defined. The process undertaken to record a labelled dataset is then documented. Before the approaches to the problem are detailed, the methods and metrics for evaluating the performance of the implementation solutions are stipulated. Finally, the results of the investigation are presented before the methods to further develop the project described.

This investigation concluded that it is possible for a machine learning algorithm to be used to estimate the 3D position of key points within a pair of stereoscopic images. However, it was found that SVR models are not realistically applicable within this problem space. The CNN implementation achieved an overall accuracy of 0.234 mm in the prediction of key points with a standard deviation of 3.18mm. Given the scope and limitations of this investigation, these results are highly acceptable, however there is scope for improvement given additional time and resources.

## References

[1] M. Rabbit, "Leap Motion, Proprioception, and the Jenga Test", *Lumo Interactive*, 2019. [Online]. Available: https://lumointeractive.com/lumo-daily-inspiration/2016/2/17/leap-motion-proprioception-and-the-jenga-test. [Accessed: 13- May- 2019].

[2] A. Colgan, "How Does the Leap Motion Controller Work?", *Leap Motion Blog*, 2019. [Online]. Available: http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/. [Accessed: 12- May- 2019].

[3] "Deep learning for depth map estimation from stereo images : MachineLearning", *Reddit.com*, 2019. [Online]. Available: https://www.reddit.com/r/MachineLearning/comments/465w29/deep_learning_for_depth_map_estimation_from/. [Accessed: 12- May- 2019].

[4] A. Kendall, "Have We Forgotten about Geometry in Computer Vision?", *Home*, 2019. [Online]. Available: https://alexgkendall.com/computer_vision/have_we_forgotten_about_geometry_in_computer_vision/. [Accessed: 12- May- 2019].

[5] *Scss.tcd.ie*, 2019. [Online]. Available: https://www.scss.tcd.ie/Rozenn.Dahyot/pdf/IMVIP2017_MatejUlicny.pdf. [Accessed: 12- May- 2019].

[6] A. Rosebrock, "Keras, Regression, and CNNs - PyImageSearch", *PyImageSearch*, 2019. [Online]. Available: https://www.pyimagesearch.com/2019/01/28/keras-regression-and-cnns/. [Accessed: 12- May- 2019].

[7] Scikit-learn.org. (2019). *1.4. Support Vector Machines — scikit-learn 0.21.0 documentation*. [online] Available at: https://scikit-learn.org/stable/modules/svm.html#svm-regression [Accessed 12 May 2019].

[8] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/. [Accessed: 12- May- 2019].

[9] V. Dibia, "How to Build a Real-time Hand-Detector using Neural Networks (SSD) on Tensorflow", *Medium*, 2019. [Online]. Available: https://medium.com/@victor.dibia/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-tensorflow-d6bac0e4b2ce. [Accessed: 12- May- 2019].

[10] S. Patel, "Chapter 2 : SVM (Support Vector Machine) — Coding", *Medium*, 2019. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-coding-edd8f1cf8f2d. [Accessed: 13- May- 2019].

[11] "Home - Keras Documentation", *Keras.io*, 2019. [Online]. Available: https://keras.io. [Accessed: 13- May- 2019].

[12] "sdcubber/Keras-Sequence-boilerplate", *GitHub*, 2019. [Online]. Available: https://github.com/sdcubber/Keras-Sequence-boilerplate/blob/master/Keras-Sequence.ipynb. [Accessed: 12- May- 2019].

[13] "sklearn.svm.SVR — scikit-learn 0.21.0 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html. [Accessed: 12- May- 2019].

# Bibliography

## Documentation

"Camera Images — Leap Motion Python SDK v3.2 Beta documentation", *Developer-archive.leapmotion.com*, 2019. [Online]. Available: https://developer-archive.leapmotion.com/documentation/python/devguide/Leap_Images.html. [Accessed: 12- May- 2019].

"6. Dataset loading utilities — scikit-learn 0.21.0 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/datasets/index.html. [Accessed: 12- May- 2019].

"Imageio's user API — imageio 2.5.0 documentation", *Imageio.readthedocs.io*, 2019. [Online]. Available: https://imageio.readthedocs.io/en/latest/userapi.html. [Accessed: 12- May- 2019].

"Imageio usage examples — imageio 2.5.0 documentation", *Imageio.readthedocs.io*, 2019. [Online]. Available: https://imageio.readthedocs.io/en/stable/examples.html. [Accessed: 12- May- 2019].

"pandas.read_csv — pandas 0.24.2 documentation", *Pandas.pydata.org*, 2019. [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html#pandas.read_csv. [Accessed: 12- May- 2019].

"Python SDK Documentation — Leap Motion Python SDK v2.3 documentation", *Developer-archive.leapmotion.com*, 2019. [Online]. Available: https://developer-archive.leapmotion.com/documentation/v2/python/index.html. [Accessed: 12- May- 2019].

"leapmotion/leapuvc", *GitHub*, 2019. [Online]. Available: https://github.com/leapmotion/leapuvc. [Accessed: 12- May- 2019].

L. Motion, "Introducing LeapUVC: A New API for Education, Robotics and More - Leap Motion Blog", *Leap Motion Blog*, 2019. [Online]. Available: http://blog.leapmotion.com/leapuvc/. [Accessed: 12- May- 2019].

"scikit-learn: machine learning in Python — scikit-learn 0.21.0 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 12- May- 2019].

## CNN

A. Deshpande, "A Beginner's Guide To Understanding Convolutional Neural Networks Part 2", *Adeshpande3.github.io*, 2019. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/. [Accessed: 12- May- 2019].

K. Rusek and P. Guzik, "Two-stage neural network regression of eye location in face images", *Multimedia Tools and Applications*, vol. 75, no. 17, pp. 10617-10630, 2014. Available: 10.1007/s11042-014-2114-z [Accessed 13 May 2019].

"Deep Leaning | Computer Vision | Convolutional Neural Network (CNN)", *YouTube*, 2019. [Online]. Available: https://www.youtube.com/watch?v=ZKc98POgNUY. [Accessed: 12- May- 2019].

"Convolutional Neural Networks - Practical with Keras", *YouTube*, 2019. [Online]. Available: https://www.youtube.com/watch?v=BcEapJEKz3M. [Accessed: 12- May- 2019].

## SVR

"1.4. Support Vector Machines — scikit-learn 0.21.0 documentation", *Scikit-learn.org*, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/svm.html#svm-regression. [Accessed: 12- May- 2019].

"LibSVM", *Csie.ntu.edu.tw*, 2019. [Online]. Available: https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf. [Accessed: 12- May- 2019].

"3.2. Tuning the hyper-parameters of an estimator — scikit-learn 0.21.0 documentation", Scikit-learn.org, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html. 6[Accessed: 12- May- 2019].

"Simple Tutorial on SVM and Parameter Tuning in Python and R | HackerEarth Blog", *HackerEarth Blog*, 2019. [Online]. Available: https://www.hackerearth.com/blog/machine-learning/simple-tutorial-svm-parameter-tuning-python-r/. [Accessed: 12- May- 2019].

"Chapter 2 : SVM (Support Vector Machine) — Theory", *Medium*, 2019. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. [Accessed: 12- May- 2019].

"Support Vector Machines(SVM) — An Overview", *Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/https-medium-com-pupalerushikesh-svm-f4b42800e989. [Accessed: 12- May- 2019].

## General

S. Yegulalp, "What is TensorFlow? The machine learning library explained", *InfoWorld*, 2019. [Online]. Available: https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html . [Accessed: 12- May- 2019].

"From raw images to real-time predictions with Deep Learning", *Towards Data Science*, 2019. [Online]. Available: https://towardsdatascience.com/from-raw-images-to-real-time-predictions-with-deep-learning-ddbbda1be0e4. [Accessed: 12- May- 2019].

"Importing Data with Pandas' read_csv()", *DataCamp Community*, 2019. [Online]. Available: https://www.datacamp.com/community/tutorials/pandas-read-csv. [Accessed: 12- May- 2019].

"Displaying real-time webcam stream in IPython at (relatively) high framerate", *Medium*, 2019. [Online]. Available: https://medium.com/@kostal91/displaying-real-time-webcam-stream-in-ipython-at-relatively-high-framerate-8e67428ac522. [Accessed: 12- May- 2019].

"Transfer Learning with TensorFlow", *Colab.research.google.com*, 2019. [Online]. Available: https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/image_feature_vector.ipynb?linkId=59649094. [Accessed: 12- May- 2019].

# Appendix A - Project Resources

Full Dataset
https://drive.google.com/drive/folders/1wb6qhpLN83UDz3FA7Qgl6HiDNFrouV0l?usp=sharing

GitHub
https://github.com/Jonathan-Oehley/EEE4114F_Project

## Appendix B - Detailed CNN Structure

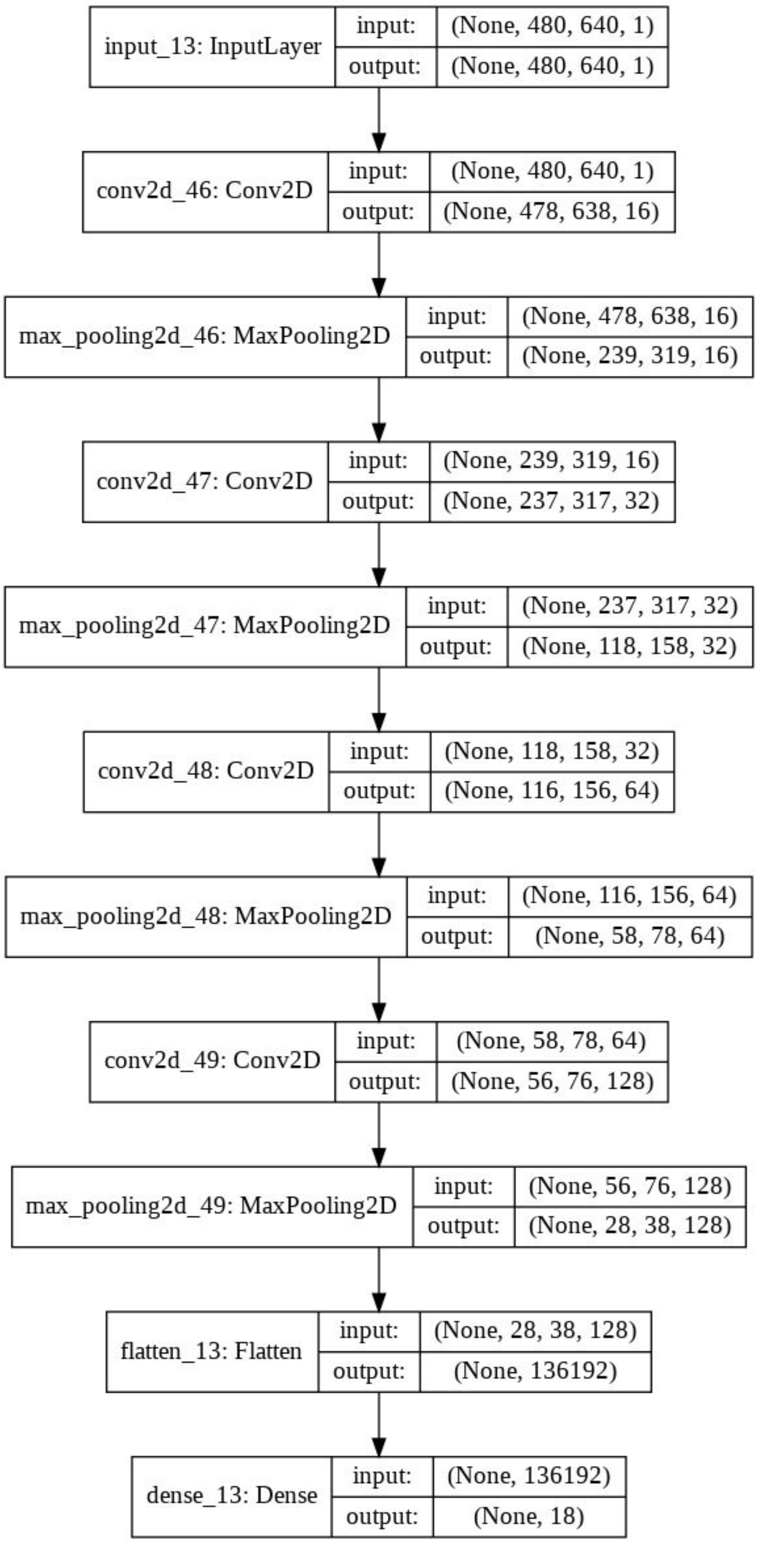

input_13: InputLayer
input: (None, 480, 640, 1)
output: (None, 480, 640, 1)
conv2d_46: Conv2D
input: (None, 480, 640, 1)
output: (None, 478, 638, 16)
max_pooling2d_46: MaxPooling2D
input: (None, 478, 638, 16)
output: (None, 239, 319, 16)
conv2d_47: Conv2D
input: (None, 239, 319, 16)
output: (None, 237, 317, 32)
max_pooling2d_47: MaxPooling2D
input: (None, 237, 317, 32)
output: (None, 118, 158, 32)
conv2d_48: Conv2D
input: (None, 118, 158, 32)
output: (None, 116, 156, 64)
max_pooling2d_48: MaxPooling2D
input: (None, 116, 156, 64)
output: (None, 58, 78, 64)
conv2d_49: Conv2D
input: (None, 58, 78, 64)
output: (None, 56, 76, 128)
max_pooling2d_49: MaxPooling2D
input: (None, 56, 76, 128)
output: (None, 28, 38, 128)
flatten_13: Flatten
input: (None, 28, 38, 128)
output: (None, 136192)
dense_13: Dense
input: (None, 136192)
output: (None, 18)

*Figure 6: Model Structure of the best performing CNN model. See the CNN section of Results and Discussion*

# Appendix C - Detailed SVR Results

The discussion of these results is presented here

*Table 2: Summary of statistical mean of the various models implemented in SVR.*

| Model | Gamma | C | Overall [mm] | X [mm] | Y [mm] | Z [mm] | Palm [mm] | Fingers [mm] |
|---|---|---|---|---|---|---|---|---|
| **1** | 3.26E-05 | 0.1 | 0.620 | 7.623 | 7.378 | 5.954 | -1.556 | 18.617 |
| **2** | 3.26E-05 | 1 | 0.620 | 7.623 | 7.378 | 5.954 | -1.556 | 18.617 |
| **3** | 3.26E-05 | 10 | 0.803 | 7.693 | 7.461 | 5.702 | -1.437 | 19.016 |
| **4** | 3.26E-06 | 0.1 | 0.620 | 7.623 | 7.378 | 5.954 | -1.556 | 18.617 |
| **5** | 3.26E-06 | 1 | 0.620 | 7.623 | 7.379 | 5.954 | -1.556 | 18.617 |
| **6** | 3.26E-06 | 10 | 0.805 | 7.691 | 7.461 | 5.692 | -1.436 | 19.017 |
| **7** | 3.26E-07 | 0.1 | 0.626 | 7.552 | 7.330 | 5.960 | -1.539 | 18.415 |
| **8** | 3.26E-07 | 1 | 0.721 | 7.567 | 7.364 | 5.931 | -1.480 | 18.501 |
| **9** | 3.26E-07 | 10 | 1.677 | 7.896 | 7.740 | 5.818 | -0.766 | 19.017 |

*Table 3: Summary of statistical mean of the various models implemented in SVR.*

| Model | Gamma | C | Overall [mm] | X [mm] | Y [mm] | Z [mm] | Palm [mm] | Fingers [mm] |
|---|---|---|---|---|---|---|---|---|
| **1** | 3.26E-05 | 0.1 | 47.64 | 67.30 | 65.81 | 72.47 | 37.20 | 79.62 |
| **2** | 3.26E-05 | 1 | 47.64 | 67.30 | 65.81 | 72.47 | 37.20 | 79.62 |
| **3** | 3.26E-05 | 10 | 47.65 | 67.33 | 65.83 | 72.46 | 37.26 | 79.62 |
| **4** | 3.26E-06 | 0.1 | 47.64 | 67.30 | 65.81 | 72.47 | 37.20 | 79.62 |
| **5** | 3.26E-06 | 1 | 47.64 | 67.30 | 65.81 | 72.47 | 37.20 | 79.62 |
| **6** | 3.26E-06 | 10 | 47.64 | 67.32 | 65.82 | 72.45 | 37.25 | 79.61 |
| **7** | 3.26E-07 | 0.1 | 47.62 | 67.28 | 65.79 | 72.45 | 37.19 | 79.62 |
| **8** | 3.26E-07 | 1 | 47.47 | 67.14 | 65.65 | 72.32 | 37.03 | 79.45 |
| **9** | 3.26E-07 | 10 | 46.19 | 65.81 | 64.31 | 71.04 | 35.64 | 77.90 |