Convolutional Neural Networks for Gesture Recognition
Calvin Kwei Hoi Tan

Convolutional Neural Networks for Gesture Recognition

Section 2 – Data Preprocessing

Section 2.2 – Understanding the dataset

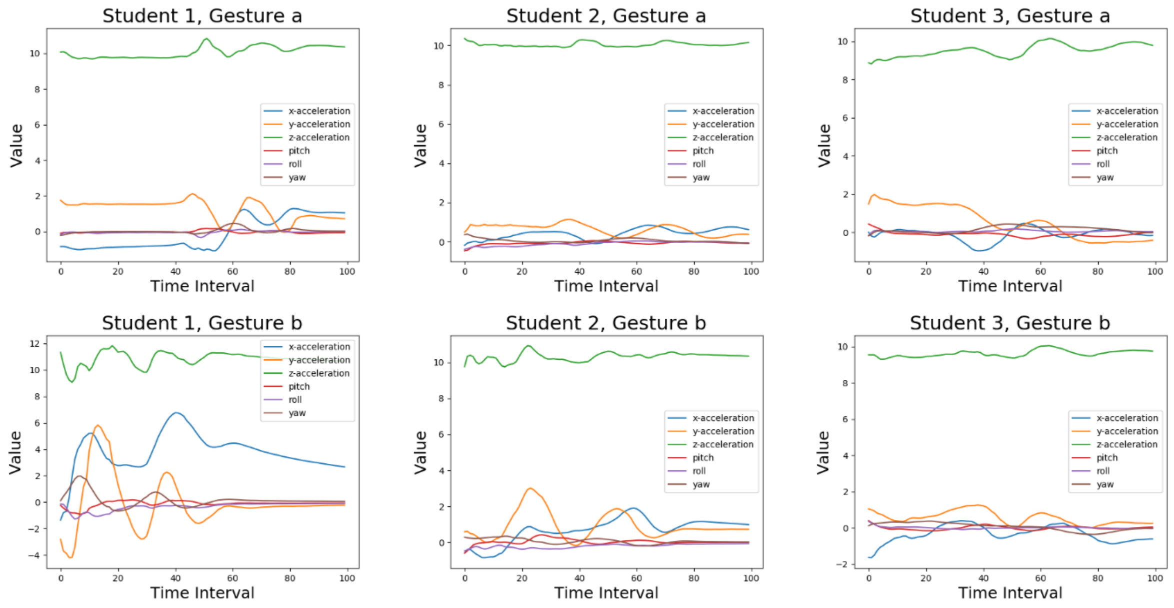


Figure 1: Sensor Values of Gestures a and b of three students

1.

Judging between gestures a and b, there is a noticeable difference in the y-acceleration sensor values (orange line). For gesture a, the y-acceleration was fairly constant for the beginning of the gesture whereas there was an immediate spike in acceleration for gesture b. Another pattern is the x-acceleration (blue line). For gesture a, it seems as if the x-acceleration decreases first before increasing (dips before rising), then plateaus whereas for gesture b, it seems as if it increases around two times (two humps). Other than that, the pitch, roll and yaw curves look fairly indistinguishable and the z-acceleration does not have much correlation between the gestures itself to conclude.

2.

Yes, my observations make sense with how I traced the letters. An example of that is gesture b. Since gesture b required the user to move their hand towards and away from them (i.e. the y-direction) in a very large motion (the “l” shape of b), it leads to a noticeable y-acceleration, which can be seen in the plots as it is bumpy. The first bump can be the user bringing the phone away from them, peaking when it reaches the furthest distance from the user and decelerating when it is brought back towards the user.

3.

Our kernels need to be 1 dimensional containing 6 entries, 1 for each recorded acceleration/rate. 1 dimensional since each of the entries are fixed to a single dimension (x, y, z, etc.) and 6 entries since there are 6 of them each gesture.

Section 2.3 – Basic Statistics

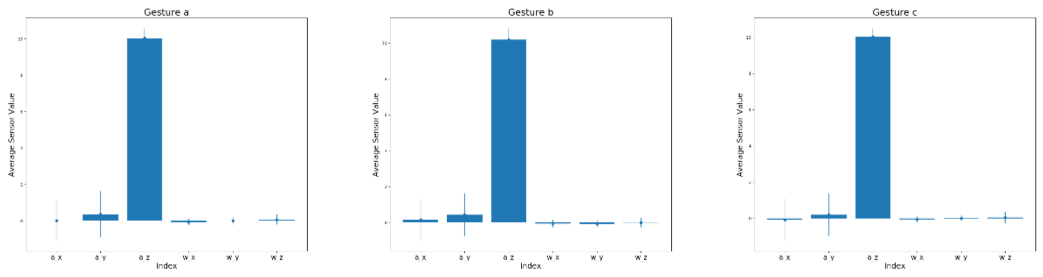


Figure 2: Average Sensor Values over Time and Gesture Instances of Letter A, B and C

1.

Looking at the bar graphs, it is very difficult if not impossible to determine which gesture is which. This is because there is a very large average sensor value of around 10 m/s^2 for the z-acceleration for each gesture (as there should be due to the force of gravity). Because of this, the resulting x-, y-accelerations, pitch, roll and yaw values are tiny in comparison to the z-acceleration. Removing this factor, there are some noticeable differences such as there is little to no average x-acceleration for gesture a whereas there is some trace for gesture b and a number in between for gesture c. For y-acceleration, gesture b seems to contain the most, which should make sense since there is a lot of movement in the y-direction, slightly less for a (since there requires the loop back up and around for a) and the least for c. Overall, however, without the given labels that '*x*' bar graph is '*x*' label, it would be very difficult to determine which gesture is which.

2.

I think it is possible that a neural network would be able to classify the gestures based on the average sensor values. This is because when averaging the sensor values, each letter was uniquely drawn in a way such that there will be subtle differences between each letter. As shown in Appendix A, each gesture has a big enough difference between the 5 average sensor values (exclude the z-acceleration since it should be constant at around 10) that it can be noticed by just glancing at the plots. Now if a neural network were to be trained on this information, the data would have a larger spread between each x-acceleration, or rather one that is more noticeable. With this large enough spread, it is possible to classify the gestures.

Section 2.5 – Train-Validation Split

1.

It is necessary to evaluate a model on a test set instead of only relying on a validation set because even the validation set has its biases. For example, when training the model on the training set, the model optimizes itself to achieve as low of a loss as possible, which in turn should increase its accuracy. This is okay, however in order to understand whether the model is actually learning and not memorizing features of the training set, the validation set is there to help guide the person determine whether or not the model is memorizing or not. In a sense, it is assumed that the model will succeed on the training set and the validation set is there to help guide the model towards better learning and not memorizing. This is indirectly training the model, which could mean the although the model is not training itself using the data, it could be producing weights such that it encapsulates both these sets without actually learning, thus an additional set, the test set, is required to test the model one last time, where it is completely unbiased since there was no prior training or validating on this set.

2.

The split chosen for the data was 0.2, or one five of the data (meaning 80% as training and 20% as validation). This was chosen since there were five gesture entries per person, which should divvy up the 4 of the 5 gestures into training and the last of into validation.

Section 3 – Training

Section 3.5 – Plotting

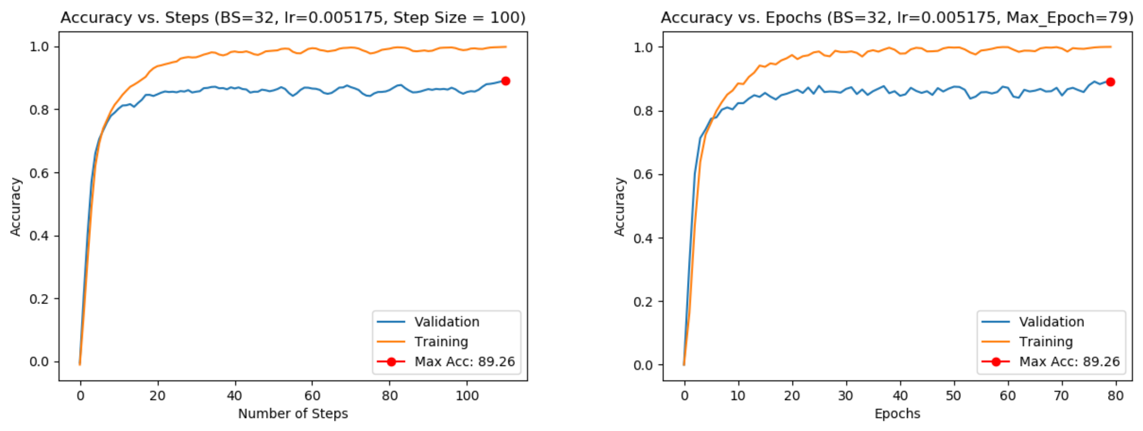


Figure 3: Training and Validation Accuracy of Model against Number of Steps and Epochs

The best validation accuracy my model achieved was 89.26.

Section 4 – Test Performance

Section 4.2 – Explanation/Source of Accuracy

Originally, the hyperparameters chosen were that of the previously assignment, batch size of 64, learning rate of 0.01, and a fairly high epoch number. Those were the hyperparameters that were known during that time. With these set, I started experimenting with different techniques.

Kernel Size and Padding

I first started off with a Kernel size of 5 and a **padding size** of **2**. This is because of what was mentioned in lecture; the kernel starts at one end and goes until the other. A padding should be chosen such that it is around half the kernel size (which was copied from assignment 1 as I had no clue of what size to use). After experimenting with this, results improved. I eventually led up to increasing my **kernel size** to **10**. I was unsure how the kernel size affected my results so I incremented it up and down (as done previously with batch sizes, learning rates, etc. in assignments 1 and 2). I came to the conclusion that a kernel size of 10 was most optimal.

Stride, and Dilation

I experimented on both stride and dilation, primarily on stride whilst working with kernel size and padding. What I noticed was that my channel size for my data was shrinking as I increased these values, and it caused a lower validation accuracy. Due to this I left both **stride** and **dilation** at its default values of **1**.

Pooling

I did not touch upon pooling as much as I would have liked. Just like kernel sizes, I matched the pooling size to that from assignment 1, thus my **pool size** is **2.**

Number of Layers

In this section, I will talk about both the number of convolutional layers as well as fully connected layers. Initially, I started with two of each, since it felt like a safe number to start with. The accuracy was not terrible, so I increased the number of layers by one each (following the advice of the meme in the assignment, more layers = better). As I increased the number of layers, specifically the convolutional layers, it kept decreasing the size of the data channels (due to the max pooling). Because of this, I could either only have a limited number of convolutional layers and max pool between each layer or do not pool after every layer. I chose the first option as it made sense to me that pooling should occur. As such, I stuck to having **3 convolutional layers**. As for the fully connected layers, I too experimented with more linear layers however the results did not prove as desirable, thus I stuck to keep it the same number as the convolutional layers (**3 linear layers**).

Activation Functions and Soft Max

The default activation function from assignment 1 was ReLu and tanh from assignment 2. I first started with ReLu since assignment 1 had a CNN. As such, the results were decent. When I started experimenting with tanh and sigmoid however, the validation accuracy did not reach nearly as high as it did with ReLu and as such, the **activation function** chosen was **ReLu**. When trying to vary the activation function between layers and whether there should be activation after each layer, I found that the results were worse than with the activation function.

In lecture, the idea of a soft max was discussed where a soft max should be applied after iterating the network, and before returning the model. As such the default option of soft max was used. After experimenting with log soft max and without any at all, it was concluded that a **log soft max** was the technique that improved the accuracy the most.

Layer Sizes

Originally, I had my layer sizes to be double, quadruple, etc. of the output size in the convolution layer (somewhat identical to that of assignment 1, where it goes from 10 to 20), and in reverse in the linear layers. As usual, I experimented with these sizes and found that the larger these layers are, the more accurate the model was (as well as the larger it got). The larger the size, the longer it took to compute the model. Knowing such, I could keep increasing the sizes of my layers and potentially have better results but would sacrifice space and time. I settled for sizes of **input_size -> 158 -> 312 -> 520** for the **convolution layers** (all multiples of 26, the output size except for 158, as I accidentally added 2 extra) and **208 -> 104 -> 26** for the **linear layers.**

Batch Normalization, Dropout

I found that **normalizing** the **batch** size **prior to soft-maxing** the model provided better results. I had experimented with batch normalization between activation functions, prior to the layers, after layers, and after the activation functions, all with little to no improvement. Likewise can be said for dropout. For dropout, since my model had a high training accuracy (100% for a majority), dropout zeroes entries, which does not really benefit as much as I believed it could be.

Optimizer

I first started with the Stochastic Gradient Descent optimizer, as it was what was used in the previous assignment, however after hearing about **Adam** from classmates, I tried this optimizer on my model and it worked better.

Batch Size, Learning Rate and Epochs (revisited)

With all the prior parameters set, I started to fine-tune these hyperparameters. I first started with batch size. I went from 32 to 64 to 128. Since 64 was my default, I compared to results to that one. With a batch size of 32, I noticed that the accuracy was better than that of a batch size of 64, whereas with a batch size of 128, my model was stuck with an accuracy of around 0.04, which is as good as randomly guessing, thus I chose my **batch size** to be **32**. For learning rate, I also varied it and found that the most optimal one was **learning rate** = 0.005175. Finally, for epochs, I realized that since my model will always overfit, I should try to epoch for as long as possible to reduce the loss function and see whether there was improvement. And with that in mind, I tried that and my model occasionally increased in validation accuracy even though the training accuracy was 1. As such, I set **epoch** such that it was high enough to allow overfitting.

Seed

The last parameter was the seed. Initially, I tried it with a seed of 0, and incremented it by 1 each time. Since there was no real science to this, I kept trying for the next 20 seeds and found that **seed 6** was the most successful.

Z-Acceleration and k-Cross

Although these two features were not implemented, I did try and attempt to use these. First, the z-acceleration. I attempted to drop the z-acceleration, as to think that it was not necessary since it should be a constant 10 (constant force of gravity) however my results were worse than with the feature included. I concluded that this was because since the users providing the data did not have the device perfectly horizontal, there was some z-acceleration in the motions and caused this feature to be important. It can also be seen in each letter, as not all of them have an average value of 10. For k-Cross validation, I tried to implement it, thinking it was a method in which I train my model k times however it was a validation method, not meant for retraining the model, so this idea was abandoned.

Summary of features:

3 conv layers

158->312->520

kernel size 10

padding 2

stride 1 (default)

pool = 2

dilation = 1 (default)

relu activation

3 FCL

208->104->26

batch normalization between LAST layer and SOFTMAX

relu activation

log_softmax

SEED = 6

LR = 0.005175

BS = 32

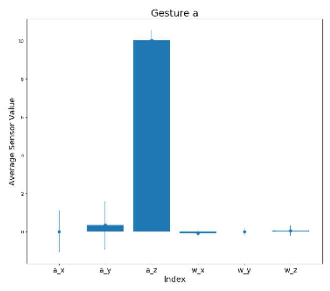
Section 4.4 – Submission Instructions

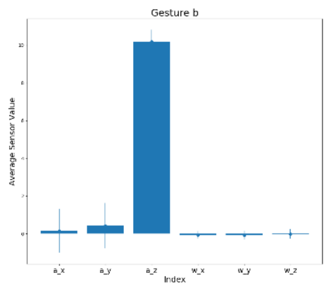
3.

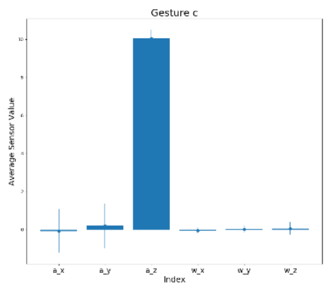
The hyperparameters are all set, all that is required is that there is a GPU since the model saved and the one being trained was done on a GPU. To test, uncomment the test() call in main.py and to train, uncomment the train() call. csv2numpy and normalize_data were commented since it was repetitive to redo what is not necessary, however if it is the first time trying the model, uncomment and run such lines.

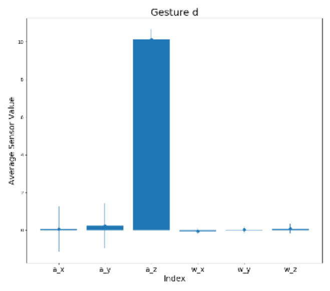
Appendix A:

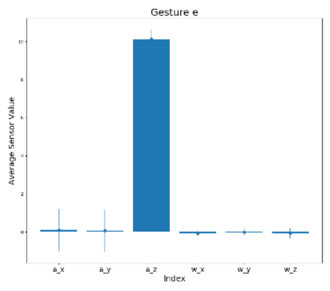
Average Sensor Values for Gestures A to Z

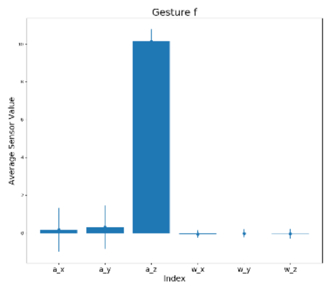


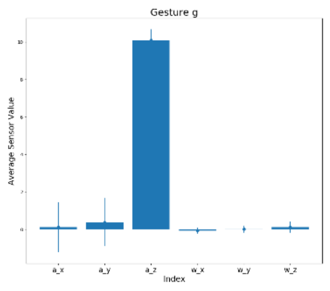


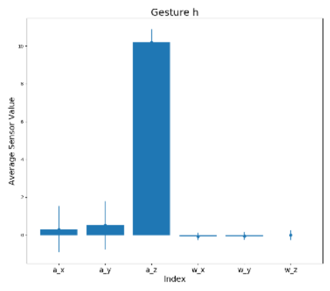


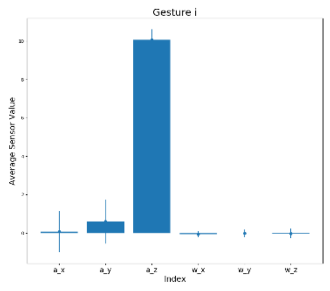


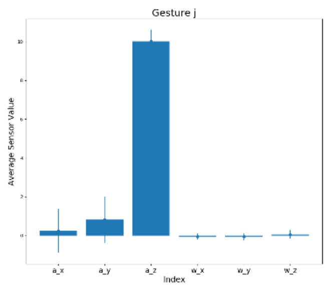


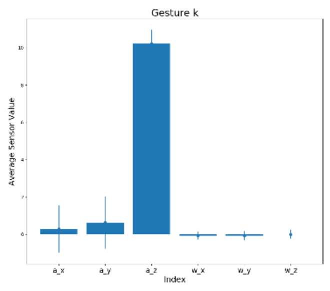


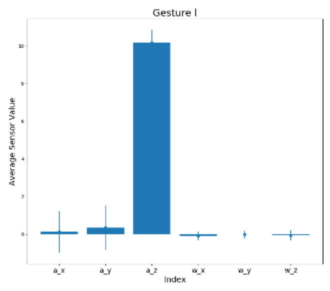


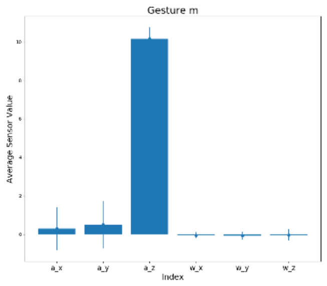


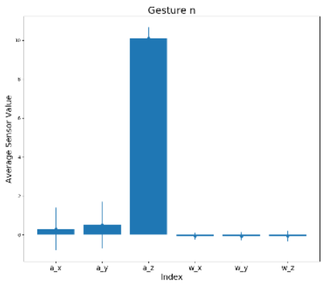


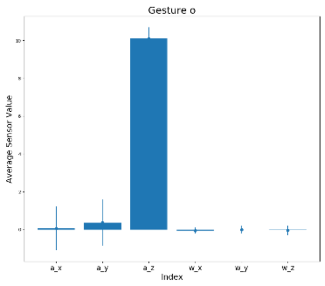


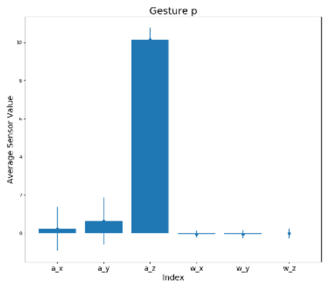


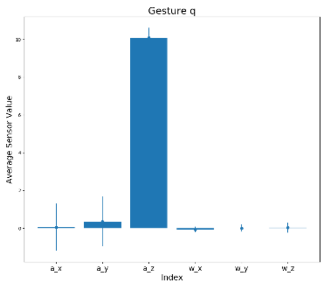


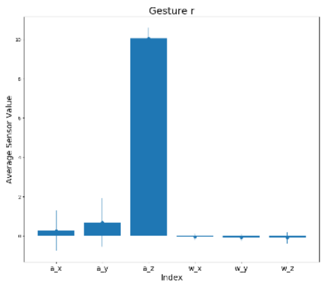

Gesture m
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

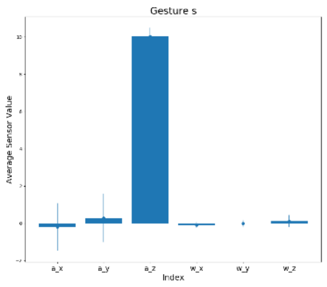

Gesture n
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

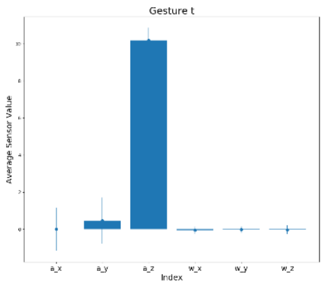

Gesture o
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

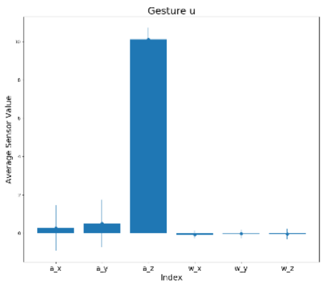

Gesture p
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

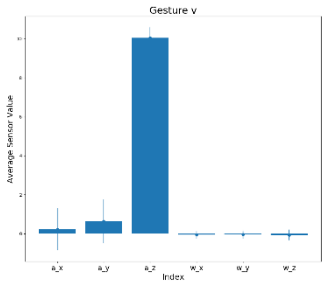

Gesture q
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

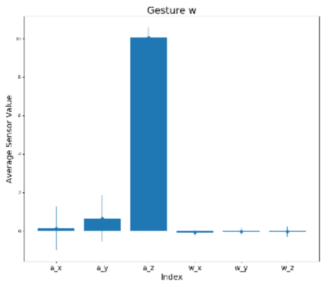

Gesture r
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

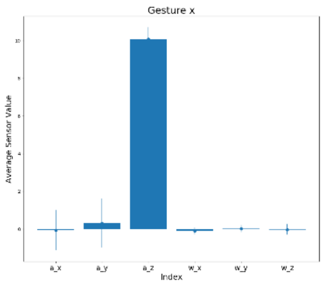

Gesture s
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

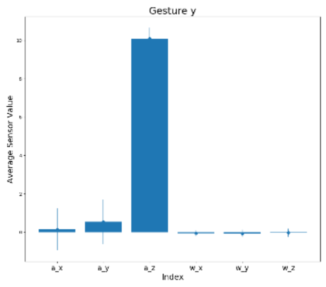

Gesture t
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index

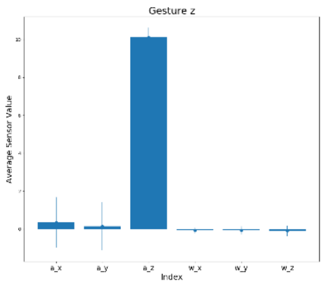

Gesture u
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index


Gesture v
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index


Gesture w
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index


Gesture x
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index


Gesture y
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index


Gesture z
Average Sensor Value
a_x
a_y
a_z
w_x
w_y
w_z
Index