Contents

1 Creating labelled training & test data

1.1 Collecting the data

Training is done based on ***mutual*** *and* ***self strength*** *data* (*baseline is subtracted* for model accuracy) with **multiple touches**. A total of 2705 **strength** frames were used (*2-5 fingers, no baseline subtraction, zebra display noise, typical AFE settings, multiple locations, thumb*). Both ***grounded*** *and* ***floating*** *touch* condition were recorded: 1943 grounded and 762 floating.

An example of mutual data is shown below. Note the data size: **(16, 32)**

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	92	-26	-12	53	53	57	74	26	101	74	-2	68	58	71	-38	11	16	70	68	26	33	44	28	-34	-33	51	19	-24	24	-44	28	23
1	105	4	96	110	161	172	86	46	115	103	103	114	81	100	93	64	75	45	60	10	25	22	67	49	10	11	53	-8	83	28	-12	20
2	-34	9	-1	-100	534	770	203	113	32	39	717	675	-4	4	-68	59	13	1	9	-22	19	-38	-30	-35	-46	14	8	30	-20	-28	-10	16
3	-71	-39	-45	-2	851	835	132	41	78	296	933	846	119	-12	8	3	42	36	-34	-42	-55	-54	-41	-48	-6	9	-12	42	-30	-23	-22	5
4	99	67	82	129	544	392	153	145	157	250	579	335	98	25	38	40	150	99	42	15	27	22	67	-6	-10	60	40	75	10	11	-22	11
5	19	20	126	516	265	95	23	20	14	44	-1	-30	14	2	-12	-4	55	21	-31	-64	-45	-45	10	-8	-6	2	46	25	-39	-46	0	-11
6	89	84	454	1049	659	62	13	19	22	-16	-94	-49	-10	-36	3	62	45	7	32	49	46	4	-2	-9	-49	55	3	15	22	10	36	-34
7	88	143	268	851	510	78	50	27	34	75	27	11	25	57	97	106	82	36	51	80	62	85	10	5	28	68	13	21	27	23	87	12
8	71	-54	25	12	103	104	105	93	67	44	28	-1	-39	-59	-2	-47	-15	7	-2	2	-15	-16	-45	-68	44	-9	-41	9	-46	-54	4	-20
9	27	10	-24	59	114	32	32	24	5	-27	-32	-59	-105	-67	-130	-57	-92	-60	55	56	-16	-77	-73	-55	12	-35	-52	-8	-63	-72	-19	12
10	43	112	71	82	103	151	91	104	109	82	100	106	63	65	-2	71	118	48	43	61	48	31	-24	37	53	6	-15	29	-2	-9	37	54
11	49	192	200	557	429	150	131	142	139	126	132	129	115	37	34	38	95	151	80	49	32	4	23	17	40	-17	31	15	-17	-25	12	30
12	62	53	499	933	802	33	71	28	34	35	266	613	173	54	38	81	64	30	24	38	4	18	2	-5	48	60	-18	35	-25	-8	-19	14
13	152	52	221	572	231	34	114	136	88	104	619	1069	384	101	82	114	48	72	61	26	60	57	-16	98	30	41	83	7	12	35	28	-14
14	86	6	27	58	32	18	109	2	100	101	323	741	80	77	53	25	18	54	11	17	29	24	9	23	13	23	1	36	-51	40	-16	-44
15	-70	-11	-11	17	7	-11	-97	55	-24	-92	11	52	42	30	-6	-39	-41	-66	-24	-71	20	-20	-16	-64	6	-87	-26	-30	-48	-3	-57	-19

A Matlab script will browse the data folders (for different conditions), read all Excel files (as saved by the GUI), extract all raw data matrices and finally, merge and save them into a new file as a **(2705, 512)** matrix (rows = *number of data*, columns = *flattened raw data; 16 * 32 total*).

Improvement areas:

1) Identify all difficult scenarios (e.g. *charger noise, zebra display noise, low accumulations, shipment/testing noise, baseline drift, water condition,* etc.) and **collect data for each scenario**
2) Redo 1) for **differential mode** to train the neural network directly based on it (no integration).

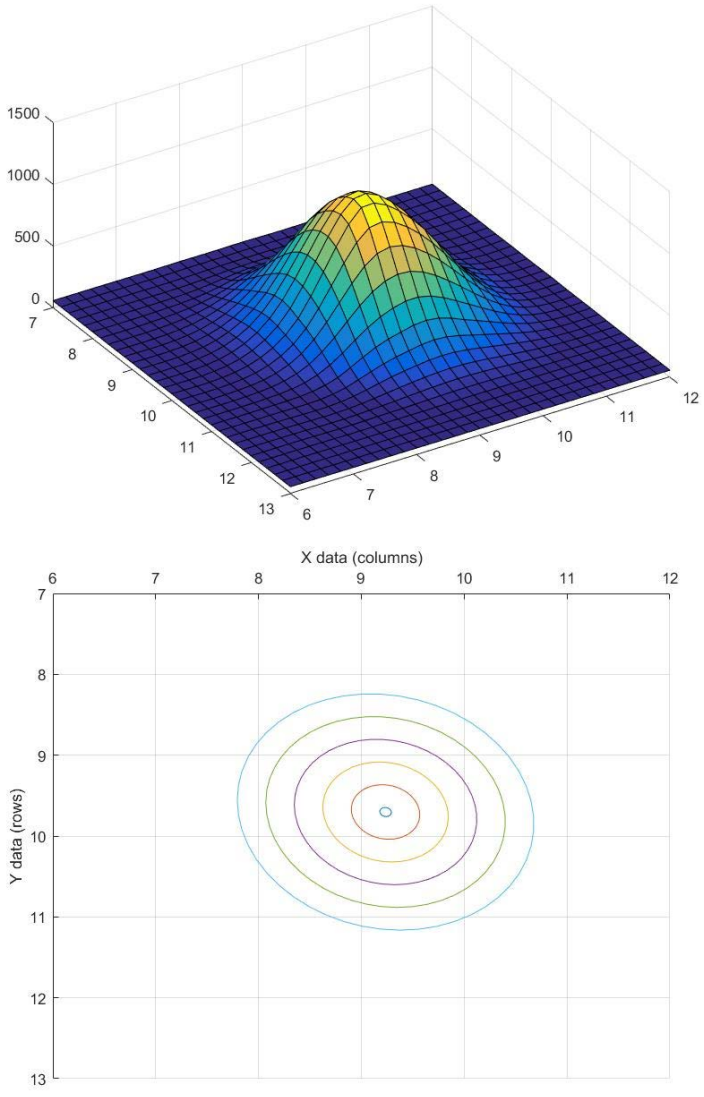
1.2 Computing the touch location(s) for *grounded touches*

For *grounded touches*, a Matlab algorithm is used to label the data (*floating touch coordinates* are much harder to compute by an algorithm, due to the irregular mutual data, so they are processed ***manually***!). Knowing the number of touches, the peak (touch) locations are identified and a matrix is chosen around each of them. Depending on the relative distance between the touches (in order to avoid 2 touches in the same window), the window size is chosen as (3, 3), (5, 5) or (7, 7). An example of a (7, 7) window is shown below:

	C06	C07	C08	C09	C10	C11	C12
R07	-42	-28	-43	-101	-79	-69	-36
R08	-116	-108	-104	-148	-111	-24	10
R09	-81	-96	-269	-870	-496	-15	45
R10	-44	-83	-409	-1044	-800	-133	-37
R11	-16	-52	-122	-354	-251	-84	-17
R12	-51	-21	-32	-83	-135	-104	-67
R13	-82	-100	-45	-71	-81	-123	-90

The matrix values are interpreted as Z values at the intersections of rows and columns (force and sense lines in the actual application). The **coordinates** are computed relatively to the **row/column number**:

The algorithm will then try to fit a 2D gaussian (see below figures):



Extracted values: $peak\ coordinates: \left(X_{peak}, Y_{peak}, Z_{peak}\right) = (9.24, 9.7, 1177)$

$peak\ shape: \ (\sigma_1, \sigma_2) = (0.59, 0.53), \quad angle = -48.7°$

1.3 Adding self sensing data

Self sensing data is an *additional input information* which mostly helps with *classification in floating condition* (e.g. differentiating big thumb from 2 adjacent touches or differentiating touch from large water drops), but also with *regression* (e.g. coordinate computation when mutual data is very low in floating condition due to fingers present on the same line).

In self mode, we obtain two 1D arrays: *Self Sense* (SS) of shape (1, 32) and *Self Force* (SF) of shape (1, 16). From them we can generate the self data matrix (input channel X1 of shape (16, 32)) as the average of the Self Sense and Self Force values:

$$X1_{(i,j)} = \frac{SF_i + SS_j}{2}, where\ i = \overline{0,15}\ and\ j = \overline{0,31}$$

An example is shown below for the case of 3 floating touches (marked by square contours in the matrix):

		C00	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	C21	C22	C23	C24	C25	C26	C27	C28	C29	C30	C31
		63	58	64	62	72	75	92	111	98	132	114	132	128	115	120	124	131	135	123	102	79	71	58	55	19	-307	-440	-297	-524	-629	-426	-7
R00	56	-60	-57	-60	-59	-64	-65	-74	-83	-77	-94	-85	-94	-92	-85	-88	-90	-94	-96	-90	-79	-68	-64	-57	-55	-37	126	192	121	234	287	185	-25
R01	69	-66	-63	-66	-65	-70	-72	-80	-90	-83	-100	-91	-100	-98	-92	-94	-96	-100	-102	-96	-85	-74	-70	-63	-62	-44	119	186	114	228	280	179	-31
R02	144	-103	-101	-104	-103	-108	-109	-118	-127	-121	-138	-129	-138	-136	-129	-132	-134	-137	-139	-133	-123	-111	-107	-101	-99	-81	81.8	148	76.8	190	243	141	-68
R03	135	-99	-97	-100	-99	-104	-105	-113	-123	-117	-134	-124	-133	-132	-125	-127	-129	-133	-135	-129	-119	-107	-103	-97	-95	-77	86	153	81	195	247	145	-64
R04	201	-132	-129	-132	-131	-136	-138	-146	-156	-149	-166	-157	-166	-164	-158	-160	-162	-166	-168	-162	-151	-140	-136	-129	-128	-110	53.3	120	48.3	162	214	113	-97
R05	191	-127	-124	-127	-126	-131	-133	-141	-151	-144	-161	-152	-161	-159	-153	-155	-157	-161	-163	-157	-146	-135	-131	-124	-123	-105	58.3	125	53.3	167	219	118	-92
R06	214	-139	-136	-139	-138	-143	-144	-153	-162	-156	-173	-164	-173	-171	-164	-167	-169	-173	-175	-169	-158	-147	-143	-136	-134	-116	46.5	113	41.5	155	208	106	-104
R07	152	-107	-105	-108	-107	-112	-113	-122	-131	-125	-142	-133	-142	-140	-133	-136	-138	-141	-143	-137	-127	-115	-111	-105	-103	-85	77.8	144	72.8	186	239	137	-72
R08	-295	116	119	116	117	112	110	102	92.3	98.5	81.5	90.8	81.8	83.5	90.3	87.8	85.8	82	80	86	96.5	108	112	119	120	138	301	368	296	410	462	360	151
R09	-762	349	352	349	350	345	344	335	326	332	315	324	315	317	324	321	319	315	313	319	330	341	345	352	354	372	534	601	529	643	695	594	384
R10	-298	118	120	117	118	113	112	103	93.8	100	83	92.3	83.3	85	91.8	89.3	87.3	83.5	81.5	87.5	98	110	114	120	122	140	303	369	298	411	464	362	153
R11	8	-36	-33	-36	-35	-40	-41	-50	-59	-53	-70	-61	-70	-68	-61	-64	-66	-70	-72	-66	-55	-44	-40	-33	-31	-13	150	216	145	258	311	209	-0.5
R12	-382	159	162	159	160	155	154	145	136	142	125	134	125	127	134	131	129	125	123	129	140	151	155	162	164	182	344	411	339	453	505	404	194
R13	-481	209	211	208	209	204	203	195	185	191	174	184	175	176	183	181	179	175	173	179	189	201	205	211	213	231	394	460	389	502	555	453	244
R14	-99	18	20.5	17.5	18.5	13.5	12.3	3.75	-5.8	0.5	-17	-7.3	-16	-15	-7.8	-10	-12	-16	-18	-12	-1.5	10	14	20.5	22.3	40.3	203	270	198	312	364	262	53
R15	-17	-23	-21	-24	-23	-28	-29	-37	-47	-41	-58	-48	-57	-56	-49	-51	-53	-57	-59	-53	-43	-31	-27	-21	-19	-0.8	162	229	157	271	323	221	12

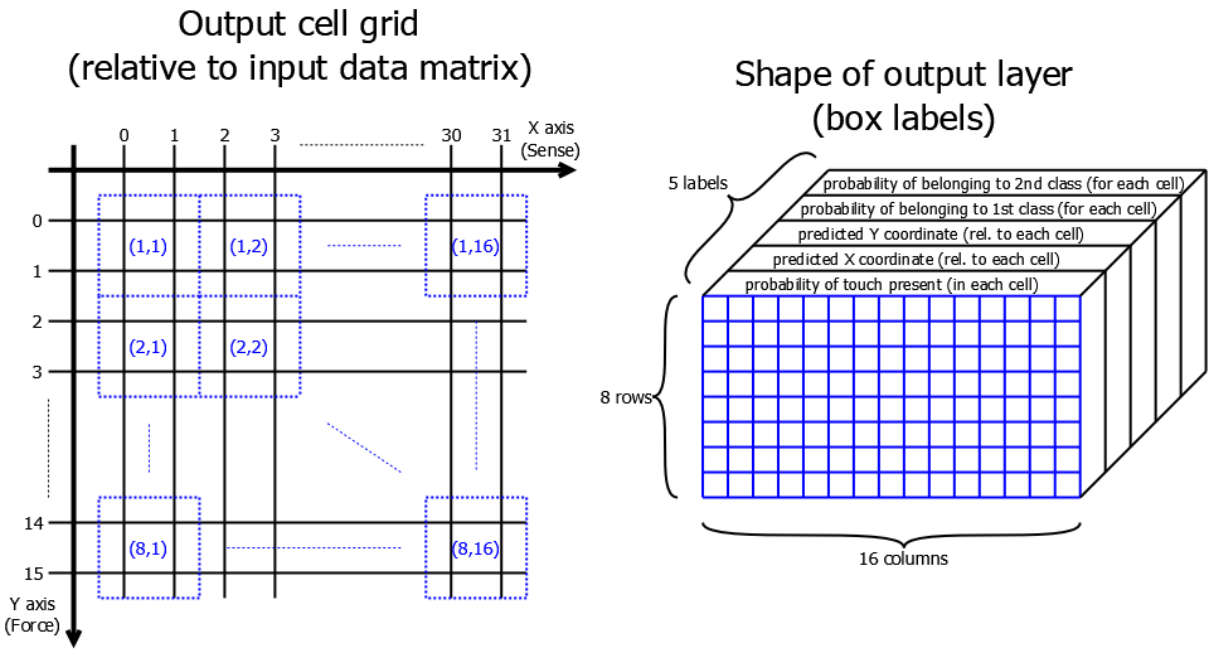
The SS and SF arrays are the very first row and column respectively (outside the self matrix).

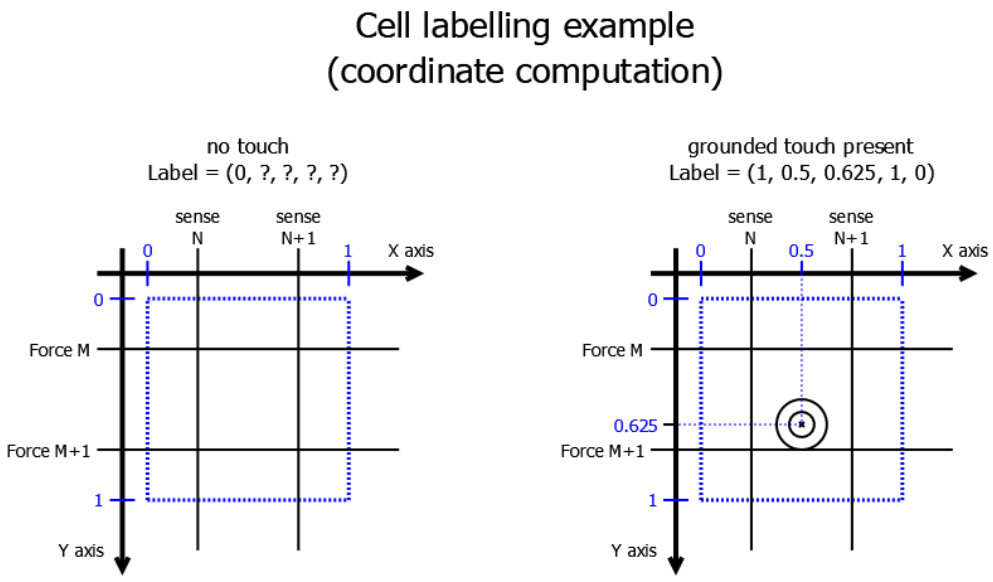
Note that the touch locations are distinguishable despite the common mode values and noise (which should naturally be filtered by weights of opposite signs in the convolutional kernels).

1.4 Creating a labelled training set

At this stage, we know the touch locations for each of the 2705 raw data frames. The way each data is labelled/stored is by creating a cell grid (see figure below) and for each cell, we have 4 labels: 1 label for *the presence of touch in the cell* ('1' if touch, '0' if no touch), 2 labels for *the coordinates* (one for *X* and one for *Y*) and 2 labels for *the touch class* ('10' if 1st class, '01' if 2nd class). The coordinates are values in the [0, 1] interval, corresponding to the relative touch location in the cell (see figure below).

Compared to previous implementations, having all labels in the [0, 1] interval will facilitate the training of the CNN (stronger gradients near 0 leading to faster learning; normalization can be omitted).





At the end of this initial labelling stage, each touch will correspond to a unique cell in the output stage. While this might seem good for discriminating *touch* from *no touch*, a problem arises when the touch is located at the edge of a cell. In this scenario, the model won't know how to distinguish the touch (it will likely classify the both adjacent cells sharing the touch as *no touch*!).

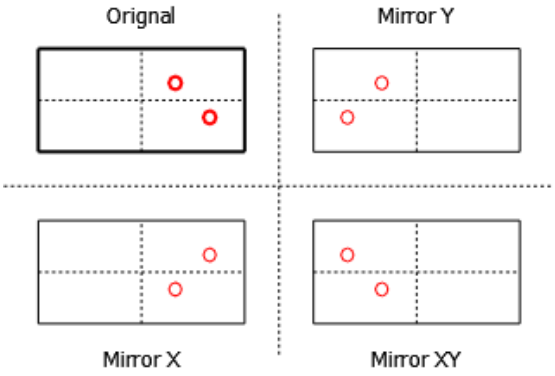
To mitigate this issue, a Matlab algorithm will *expand the touch labels* to the adjacent cell(s) if the touch is within **0.05** from it(them). Therefore, a touch may correspond to 1, 2 or 4 cells in the output grid. An algorithm called *non-maximum suppression* (described in a dedicated section below) will eliminate redundant touch predictions (so having extra "predicted" touches around a real touch is usually not an issue). Overall, this manner of labelling the touches is more robust and significantly increases the classification accuracy for touches at the grid edge (~20% of the total touch surface).

1.5 Data augmentation

A cheap way of increasing the training data set is "*data augmentation*". Basically, this means *creating new labelled data* from existing ones by geometrical *transformations* (translation, rotation, mirroring) taking into account the nature and symmetry of the data. *Note*: data augmentation can also be done by adding noise, but this isn't always effective and the noise must be properly characterized.

For the touch application, the input data matrix may have specific *patterns on the rows* (constant, linear or parabolic noise shape for a given time slot along one Force line; e.g.: display noise) or *on the columns* (low frequency or beat frequency noise along the Sense lines where a touch is present; e.g.: charger noise, lamp noise). Therefore, the rows and columns can't be rotated or taken out of sequence.

The only viable transformation is the *mirroring* (also taking into account that a touch can be present in any quadrant of the input matrix), which generates 3 new views for every input matrix, resulting in *4x more training data* (see below).



A Matlab script will augment the data giving a total of *10820 labelled input data* (*7772* for *grounded* touch and *3048* for *floating* touch).

Finally, the 10820 data will be *permutated* (to randomize the order) and split into a *training* set (used in training the neural network model) and a *testing* set (used to evaluate the model accuracy on previously unseen data): **80%** (8656 data) used for training and **20%** (2164 data) used for testing.

To conclude, we now have the following data to be used in training and testing the CNN:

```
number of training examples = 8656
number of test examples = 2164
X_train shape: (8656, 16, 32, 2)
Y_train shape: (8656, 8, 16, 5)
X_test shape: (2164, 16, 32, 2)
Y_test shape: (2164, 8, 16, 5)
```

2 Creating and training the CNN model

2.1 The CNN architecture

The model does both the classification and the regression, so it **predicts the touch presence, the touch coordinates and the touch class** (e.g. grounded, floating, etc.).

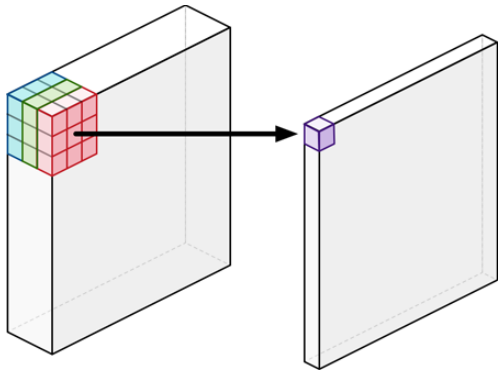
The chosen CNN architecture is summarized below:

Layer		Kernel					Output shape			Parameters		
#	name	W	H	Cin	Cout	stride	W	H	Cout	weights	bias	MACs
1	INPUT						16	32	2			
2	CONV1	3	3	2	8	1	16	32	8	144	8	80784
	ReLU											
	POOL	2	2			2	8	16	8			
3	CONV_DW1	3	3	8	1	1	8	16	8	72	8	11016
	CONV2	1	1	8	16	1	8	16	16	128	16	19584
	ReLU											
4	CONV_DW2	3	3	16	1	1	8	16	16	144	16	22032
	CONV3	1	1	16	32	1	8	16	32	512	32	78336
	ReLU											
5	CONV4	1	1	32	5	1	8	16	5	160	5	24480
										Total:		
										1245		**236232**

Note: Cin and Cout refer to the number of input and output channels respectively.

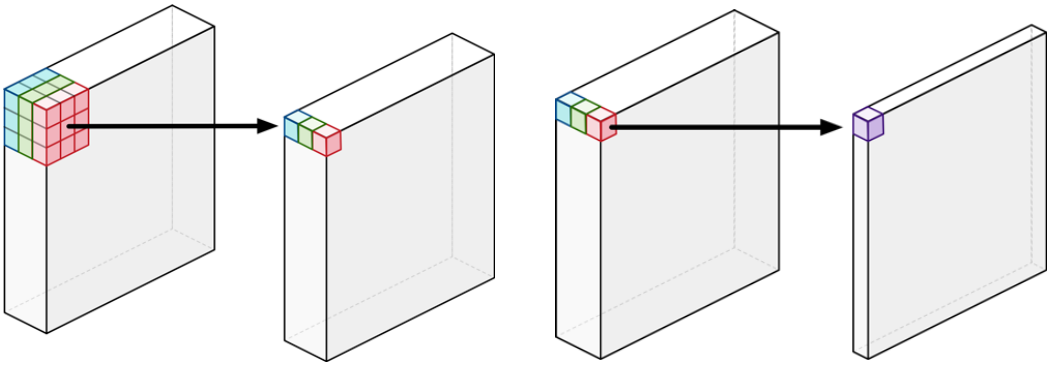
There are 5 layers (1 input layer and 4 convolutional layers). The input consists of 2 channels (mutual and self matrices) of size (16, 32).

The convolutional layers are of 2 types: standard *convolution over volumes* (CONVi) and *depthwise separable convolution* (CONV_DWi). A standard convolution is shown below for an input with 3 channels (Cin = 3) and a kernel of (3, 3, 3, 1):



The resulting point is the dot product of the 27 input elements (highlighted in RGB) with the convolution kernel (a 3x3x3 volume not shown). Multiple output channels (Cout) for the convolution kernel mean that multiple 3x3xCin volumes are used and each of them will generate a “slice” (channel) like the one shown above. Stacked together, the output will have Cout such channels.

The disadvantage of the regular convolution is that it will combine every input element with every kernel element, being computationally wasteful, since not all combinations are needed/relevant. A depthwise separable convolution is much lighter in terms of parameters, with little loss in information (compared to a standard convolution). It is illustrated below:



It consists of 2 parts: a *depthwise* convolution (seen on the left) and a *pointwise* convolution (seen on the right). A depthwise convolution doesn't combine the input channels, performing the convolution on each channel separately (for the purpose of filtering each channel, performing edge detection, etc.). It is followed by a pointwise convolution (which is a regular convolution, with a 1x1xCin kernel), which will recombine the input channels as a weighted sum in order to create new features. To create more output channels, Cout such pointwise kernels are stacked.

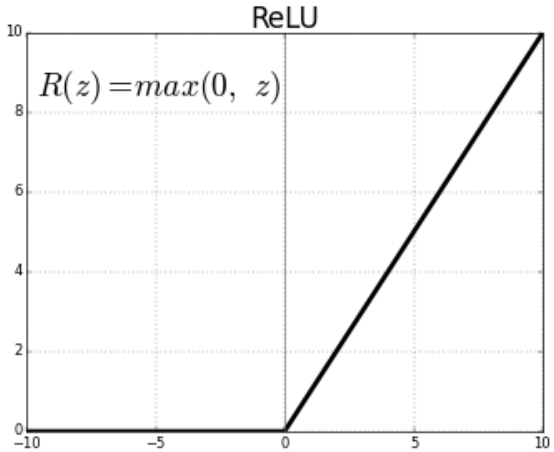
This will reduce the number of parameters (compared to a standard convolution) by a factor of:

$$parameter\_reduction\_factor = \frac{1}{C_{out}} + \frac{1}{f^2}$$

where "Cout" is the number of output channels and "f" is the kernel rank (3 in our case). For 8 output channels, the reduction factor is ~0.24, meaning a **~75% parameter reduction**!

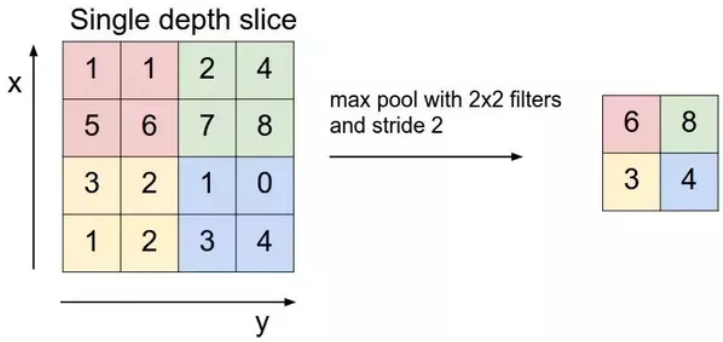
Note: preceding each convolution, zero-padding is used (size of 1) in order to maintain the input shape (the stride is always 1 for convolutions).

The *ReLU (Rectified Linear Unit)* operation happens at the output of each layer, with the exception of the last one. The ReLU function is shown below:



Note: The ReLU in between the depthwise and the pointwise convolutions is optional (*not used* currently, as it was observed that information is lost and overall accuracy degraded).

There is also a POOL (Max Pooling) operation after the first convolution for the purpose of scaling down the size by a factor of 2 in each direction. It outputs the max of each 2x2 patch of the input:



One final thing to note on the architecture choice is the progression of the depth (number of output channels): *2 (input) → 8 → 16 → 32 → 5 (output)*. While the input and output are determined by the application, the size of the hidden layers is determined by 3 factors: *exponential progression* (we want to derive more and increasingly complex features: from edge detection in the 1st hidden layer to pattern/shape detection in the 3rd layer), *complexity requirement* (more and deeper layers will always give better accuracy, provided we have enough training examples) and *implementation restrictions* (finally, the hardware and processing time will be the main roadblocks, so limiting the weights to a reasonable number (~1000) is a must).

2.2 Training the model

To train the model, we need to define a *loss (cost) function* which incorporates and weighs the different parts of the model: the touch presence probability, the predicted touch coordinates and the predicted touch class.

The total loss, as well as the weighted partial losses are defined as:

$$\mathcal{L}_{total} = \lambda_{touch} \cdot \mathcal{L}_{touch\_present} + \lambda_{no\_touch} \cdot \mathcal{L}_{touch\_not\_present} + \lambda_{coord} \cdot \mathcal{L}_{coord} + \lambda_{class} \cdot \mathcal{L}_{class}$$

$$\mathcal{L}_{coord} = \sum_{i=0}^{15} \sum_{j=0}^{31} 1_{i,j}^{touch} \cdot \left[\left(x_{i,j} - \widehat{x_{i,j}} \right)^2 + \left(y_{i,j} - \widehat{y_{i,j}} \right)^2 \right]$$

$$\mathcal{L}_{class} = \sum_{i=0}^{15} \sum_{j=0}^{31} 1_{i,j}^{touch} \cdot \sum_{c=0}^{1} (p_c - \widehat{p_c})^2$$

$$\mathcal{L}_{touch\_present} = -\sum_{i=0}^{15} \sum_{j=0}^{31} 1_{i,j}^{touch} \cdot (1 - \widehat{p_t})^{\gamma} \cdot \log(\widehat{p_t})$$

$$\mathcal{L}_{touch\_not\_present} = -\sum_{i=0}^{15} \sum_{j=0}^{31} 1_{i,j}^{no\_touch} \cdot \widehat{p_t}^{\gamma} \cdot \log(1 - \widehat{p_t})$$

$$\lambda_{touch} = 4, \qquad \lambda_{no\_touch} = 0.8, \qquad \lambda_{coord} = 0.5, \qquad \lambda_{class} = 0.5, \qquad \gamma = 4.5$$

Note that there are 2 main types of loss functions being used: *mean squared error* (also known as $\mathcal{L}_2$ loss; typically used for linear regression) and *focal loss* (a variant of cross entropy; typically used for logistic regression).

Since we mainly want to compute the loss for relevant values (e.g. *where a touch is located*), two logic coefficients are introduced:

$$1_{i,j}^{touch} = \begin{cases} 1, & is\ a\ touch\ is\ labelled\ at\ (i,j) \\ 0, & if\ there\ is\ no\ touch\ label\ at\ (i,j) \end{cases}$$

$$1_{i,j}^{no\_touch} = \neg 1_{i,j}^{touch}$$

The *coordinate loss* measures the $\mathcal{L}_2$ loss at the touch locations (i,j) between the labelled coordinates $(x_{i,j}, y_{i,j})$ and the predicted coordinates $(\widehat{x_{i,j}}, \widehat{y_{i,j}})$.

The *class loss* is the same as the coordinate loss, between the class labels (for 2 classes, they can be (1, 0) or (0, 1)) and the predicted labels $(\widehat{p_0}, \widehat{p_1})$.

The *touch present* or *not present* losses are log-losses which heavily penalize the model whenever a significant mis-classification takes place: a touch is missed (label is 1, but $\widehat{p_t} \ll 1$) or wrongly detected (label is 0, but $\widehat{p_t} \gg 0$). The weights are chosen to balance the occurrence of "touch" and "no touch" labels (out of 128 total output cells, most are labelled as "no touch") and to even out the *precision* and *recall* (accuracy metrics described below).

Note that $\widehat{p_t}$ (the touch presence probability being output) is actually taken as the *sigmoid* of the output channel **Y0** (the (8,16) matrix of the output layer, containing the probabilities of touch). This function is shown below:

$$\widehat{p_t} = sigmoid(Y0) = \frac{1}{1 + e^{-Y0}}$$

With the cost function defined, the evaluation metric(s) also needs to be defined. The chosen metrics are *precision* and *recall* being defined as:

$$precision = \frac{TP}{TP + FP}$$

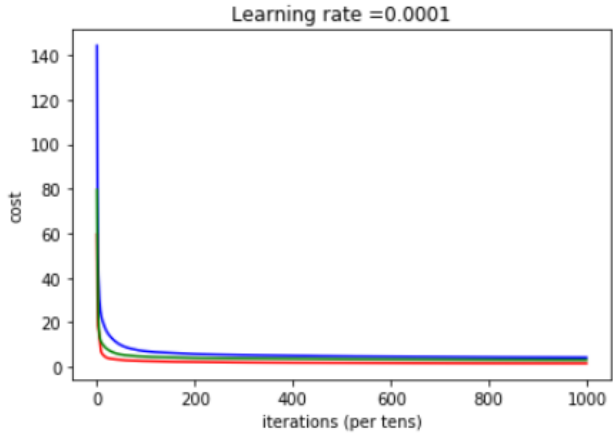
$$recall = \frac{TP}{TP + FN}$$

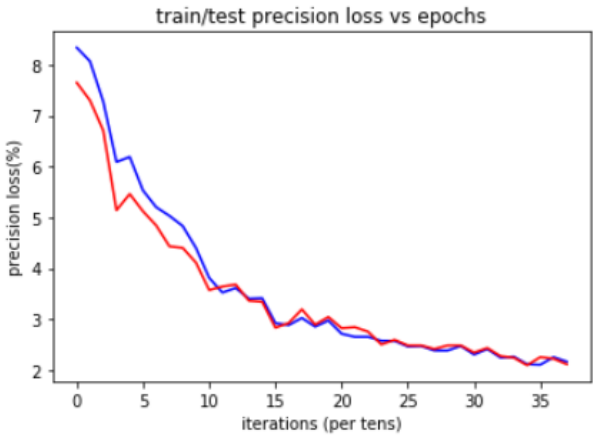
where TP = true positives (touches labelled and correctly detected), FP = false positives (touches not labelled, but wrongly detected), FN = false negatives (touches labelled, but not detected).

This makes the errors more interpretable and gives a better sense of the expected accuracy of the model. One final metric is the mean coordinate error in the TP case (even when a touch is correctly detected, it can be more or less close to the labelled coordinate). On the next page, the training progress and curves are shown (the X axis of the plots is 1-1000, the number of training iterations).

Both the precision and recall on the test set are **~98%** and the mean distance error is **~0.19** of the input grid size (meaning the distance between 2 adjacent lines, which is ~4mm). The 2 values below the plots (~99.5%) represent the percentage of output cells being correctly predicted (total of 8·16·10820 grid cells).

```
Cost after epoch 960: 4.205599
Cost after epoch 965: 4.212965
Cost after epoch 970: 4.201228
Cost after epoch 975: 4.189545
 1. Train set precision = 0.9784
 1. Train set recall = 0.9798
 1. Train set error distance = 0.192886896171
 2. Test set precision = 0.9789
 2. Test set recall = 0.9821
 2. Test set error distance = 0.195609549225
Cost after epoch 980: 4.203187
Cost after epoch 985: 4.183526
Cost after epoch 990: 4.187286
Cost after epoch 995: 4.185519
```






```
Train Accuracy: 0.995486
Test Accuracy: 0.995473
```

2.3 The detection model: non-max suppression

The input data shape is **(16, 32)**. After passing it through the CNN, we obtain an output of shape **(8, 16, 5)**, where the last dimension contains (*probability of touch, predicted X coordinate, predicted Y coordinate, probability of 1st class, probability of 2nd class*). A Python algorithm will now **remove all redundant elements** (some of them may point to the same touch, but **we only want the highest likelihood to dictate the coordinates**).

- Since each of the (8, 16) output elements has its own probability the low-likelihood elements are removed (e.g. below 50% confidence).
- The highest probability element is chosen and the predicted X_coord and Y_coord values are stored.
- The **Euclidean distance** between the highest confidence coordinates and the remaining coordinates (after the first filtering, based on confidence threshold) is computed. If the distance is **smaller than a threshold** (e.g. **2**, since 2 touches can't be within 2 adjacent nodes), the **respective elements are discarded**.

This algorithm is illustrated in the figure below (only the relevant part of the raw data is shown).

There are 3 possible touch locations, in boxes **(1, 1)**, **(1, 2)** and **(3, 2)**. The highest confidence box (box (1, 2), with 100% confidence) has the **coordinates of the first touch: (2.61, 0.96)**. Any other predicted touch within a **radius of 2** will be **discarded**. This is the case of the box (1, 1): it's within a distance of 1.**69** from the first touch location.

The second predicted touch (box (3,2), with 58% confidence) is further than 2 from the first, so it will be kept. The algorithm stops after 2 detected touches, since there are no more high confidence boxes.

Raw data:

	0	1	2	3	4	5	6	7
0	155	493	681	726	389	88	49	-3
1	559	1086	1115	1088	742	251	131	106
2	124	845	821	616	140	-7	-56	-86
3	-200	282	498	105	-107	-127	-161	-161
4	-130	-159	83	364	-27	-81	-120	-90
5	-48	-88	437	951	244	18	-56	-53
6	46	16	168	893	211	48	-7	-8
7	-124	-81	-99	62	-33	-128	-138	-165

Predicted X coord:

	1	2	3	4
1	0.95	2.61	0	0
2	0	0	0	0
3	0	2.88	0	0
4	0	0	0	0

Predicted Y coord:

	1	2	3	4
1	1.28	0.96	0	0
2	0	0	0	0
3	0	5.09	0	0
4	0	0	0	0

Probability of touch:

	1	2	3	4
1	69	100	0	0
2	0	0	0	0
3	0	58	0	0
4	0	0	0	0

distance = 1.69

3 Evaluation of the trained model on more difficult cases

3.1 Five fingers – grounded touch (*Note: with 2nd version*)

The same example of 5 fingers that was used to test the 1st implementation, is now used to illustrate the improvement of the 2nd version.

Using the Matlab labelling algorithm, we get the following labels of the 5 touches:

	touch 1	touch 2	touch 3	touch 4	touch 5
X_coord	10.38	3.16	4.53	3.21	10.84
Y_coord	2.83	6.22	2.86	11.93	13.04

Using the 2nd version mode (see figure on the next page), we get the following predictions:

	touch 1	touch 2	touch 3	touch 4	touch 5
Probability	100	100	100	100	100
X_coord	10.27	3.22	4.57	3.10	10.80
Y_coord	2.85	6.22	2.76	12.04	12.94

The predictions have **very high confidence** (basically 100%), and are **very close to the labelled coordinates.**

Overall, the 2nd version is a big leap in accuracy compared to the original implementation and can go **to the next steps**:

- **evaluation** of C implementation on M4 emulator: **computation time**, **code size**, **accuracy** (*2F separation, jitter/precision under very noisy conditions*, etc).
- **reinforcement learning** based on **more data** with **more accurate labels.**
- **training** for detection of **new classes** (e.g. *floating finger*, *water differentiation*) and making use of **additional input features** (e.g. *self-sensing data*) and **pre-processing** (e.g. *baseline subtraction*).
- training a **recurrent neural network** to have temporal behaviour (use previous touches' information to make better predictions).
- add high-level output features, like **gesture/pattern detection.**

3.2 Five fingers – floating touch (*Note: with 5th version*)

The case of 5 fingers (using the 5th version) is exemplified on the last figure. Since the body is not properly grounded, the touches are "floating", as can be observed by the presence of negative elevations. The grid elements containing the real touch centres are highlighted in black, while the corresponding predictions are highlighted in red.

Note that while all 5 touches are correctly detected, the probabilities are closer to 50% (mainly due to the cost function change from mean square error to focal loss). However, this has no impact the overall accuracy (e.g. a simple thresholding or softmax would make the probabilities closer to 0/100%, but the probability ordering and half-point would be unchanged). The focal loss heavily penalizes large misclassifications (e.g. label 1 being predicted as <0.5), tolerating small ones (e.g. 1 predicted as 0.8).

Raw data:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	92	-26	-12	53	53	57	74	26	101	74	-2	68	58	71	-38	11	16	70	68	26	33	44	28	-34	-33	51	19	-24	24	-44	28	23
1	105	4	96	110	161	172	86	46	115	103	103	114	81	100	93	64	75	45	60	10	25	22	67	49	10	11	53	-8	83	28	-12	20
2	-34	9	-1	-100	534	770	203	113	32	39	717	675	-4	4	-68	59	13	1	9	-22	19	-38	-30	-35	-46	14	8	30	-20	-28	-10	16
3	-71	-39	-45	-2	851	835	132	41	78	296	933	846	119	-12	8	3	42	36	-34	-42	-55	-54	-41	-48	-6	9	-12	42	-30	-23	-22	5
4	99	67	82	129	544	392	153	145	157	250	579	335	98	25	38	40	150	99	42	15	27	22	67	-6	-10	60	40	75	10	11	-22	11
5	19	20	126	516	265	95	23	20	14	44	-1	-30	14	2	-12	-4	55	21	-31	-64	-45	-45	10	-8	-6	2	46	25	-39	-46	0	-11
6	89	84	454	1049	659	62	13	19	22	-16	-94	-49	-10	-36	3	62	45	7	32	49	46	4	-2	-9	-49	55	3	15	22	10	36	-34
7	88	143	268	851	510	78	50	27	34	75	27	11	25	57	97	106	82	36	51	80	62	85	10	5	28	68	13	21	27	23	87	12
8	71	-54	25	12	103	104	105	93	67	44	28	-1	-39	-59	-2	-47	-15	7	-2	2	-15	-16	-45	-68	44	-9	-41	9	-46	-54	4	-20
9	27	10	-24	59	114	32	32	24	5	-27	-32	-59	-105	-67	-130	-57	-92	-60	55	56	-16	-77	-73	-55	12	-35	-52	-8	-63	-72	-19	12
10	43	112	71	82	103	151	91	104	109	82	100	106	63	65	-2	71	118	48	43	61	48	31	-24	37	53	6	-15	29	-2	-9	37	54
11	49	192	200	557	429	150	131	142	139	126	132	129	115	37	34	38	95	151	80	49	32	4	23	17	40	-17	31	15	-17	-25	12	30
12	62	53	499	933	802	33	71	28	34	35	266	613	173	54	38	81	64	30	24	38	4	18	2	-5	48	60	-18	35	-25	-8	-19	14
13	152	52	221	572	231	34	114	136	88	104	619	1069	384	101	82	114	48	72	61	26	60	57	-16	98	30	41	83	7	12	35	28	-14
14	86	6	27	58	32	18	109	2	100	101	323	741	80	77	53	25	18	54	11	17	29	24	9	23	13	23	1	36	-51	40	-16	-44
15	-70	-11	-11	17	7	-11	-97	55	-24	-92	11	52	42	30	-6	-39	-41	-66	-24	-71	20	-20	-16	-64	6	-87	-26	-30	-48	-3	-57	-19

Probability of touch:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	100	0	0	100	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	100	0	0	0	100	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Predicted X coordinate:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	4.57	0	0	10.27	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	3.22	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	3.10	0	0	0	10.80	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Predicted Y coordinate:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	2.76	0	0	2.85	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	6.22	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	12.04	0	0	0	12.94	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Input mutual raw data:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	3	34	42	67	40	43	16	-11	29	-1	38	33	33	-2	21	-2	-1	37	31	-21	-35	-8	-5	34	-25	-19	-17	-6	30	-75	-31	-77
1	33	16	-14	23	54	35	-2	-1	25	61	38	27	33	22	11	24	-13	1	21	37	19	32	39	14	33	23	11	24	-20	-55	-41	-55
2	69	-6	3	54	6	25	-8	-3	30	-3	32	41	-20	-19	-1	-8	-63	12	27	14	36	57	51	86	148	87	45	39	-40	-62	-67	-61
3	17	8	31	28	12	7	-154	-191	-12	37	12	13	22	29	19	-54	-209	-70	-85	-222	-40	61	277	890	980	171	55	41	-142	-300	-315	-241
4	17	59	20	37	34	-48	-235	-233	-49	-10	62	-14	17	17	15	-84	-300	-76	-35	-148	8	45	594	1169	890	116	1	-16	-172	-423	-448	-335
5	55	53	10	35	44	20	-165	-179	13	52	6	52	19	3	9	-46	-268	-58	159	844	632	67	118	313	190	94	1	-8	-146	-361	-386	-285
6	40	29	22	48	2	-13	-184	-214	34	39	34	12	22	12	-1	-61	-282	-70	660	1141	527	40	-31	-168	-145	50	43	14	-122	-325	-373	-270
7	32	13	4	24	20	11	-184	-186	-10	9	-6	38	-14	40	39	-79	-302	-24	284	349	63	92	-63	-234	-157	-4	25	26	348	671	229	-158
8	-62	-43	-45	-21	-27	-31	-73	-78	-58	-16	-6	-40	-59	0	-41	-180	-537	-199	-196	-470	-176	-61	-203	-485	-414	-105	-55	32	776	820	852	414
9	-38	27	-11	23	3	171	691	694	146	-32	-34	2	-23	-32	-11	-284	-677	-293	-324	-648	-318	-109	-277	-663	-552	-123	-55	-60	86	538	604	718
10	9	43	-15	17	70	135	719	697	99	21	56	-17	-34	50	-30	-147	-406	-153	-244	-407	-244	-44	-152	-429	-327	-55	-13	-34	-192	24	443	461
11	85	107	97	127	110	159	217	197	115	121	92	81	52	74	74	73	8	27	32	-17	34	38	20	-57	-25	33	37	10	-6	-66	31	37
12	4	62	88	40	47	103	65	45	52	7	57	16	55	51	55	103	300	89	10	-81	-3	2	-12	-13	-56	5	8	19	-8	-82	-60	-48
13	58	68	94	40	35	65	-111	-125	14	69	49	12	49	39	71	645	1202	543	-30	-229	-57	0	-60	-231	-164	-9	12	-53	-174	-296	-354	-218
14	37	31	40	89	43	22	-90	-126	56	75	48	72	45	42	68	520	1093	333	-24	-166	-61	5	-40	-142	-125	-66	-50	-6	-91	-300	-288	-220
15	47	57	46	89	33	34	16	-10	26	39	62	76	33	84	54	104	143	89	-24	-52	-17	-25	-28	-42	-41	-34	-18	-16	-29	-122	-102	-14

Input self raw data:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	78	54	102	77	94	90	194	230	119	139	47	64	84	84	34	140	242	118	96	247	150	32	70	153	80	30	4	0	124	188	164	108
1	144	121	168	144	161	156	260	297	186	206	114	131	150	151	100	207	309	185	162	314	216	99	136	220	147	97	70	66	190	254	231	174
2	200	176	223	198	216	212	315	352	240	260	168	186	205	206	155	262	364	240	217	368	272	154	192	274	202	152	126	122	246	310	286	230
3	323	300	346	322	340	335	438	476	364	384	292	310	328	330	278	386	488	364	340	492	395	278	315	398	326	276	249	245	369	433	410	353
4	416	392	439	414	432	428	531	568	456	476	384	402	421	422	371	478	580	456	433	584	488	370	408	490	418	368	342	338	462	526	502	446
5	390	366	413	388	406	402	505	542	430	450	358	376	395	396	345	452	554	430	407	558	462	344	382	464	392	342	316	312	436	500	476	420
6	359	336	382	358	376	371	474	511	400	420	328	346	364	366	314	422	524	400	376	528	431	314	351	434	362	312	285	281	405	469	446	389
7	304	280	328	303	320	316	420	456	345	365	273	290	310	310	260	366	468	344	322	473	376	258	296	379	306	256	230	226	350	414	390	334
8	426	402	449	424	442	438	541	578	466	486	394	412	431	432	381	488	590	466	443	594	498	380	418	500	428	378	352	348	472	536	512	456
9	561	538	584	560	578	573	676	714	602	622	530	548	566	568	516	624	726	602	578	730	633	516	553	636	564	514	487	483	607	671	648	591
10	380	356	404	379	396	392	496	532	421	441	349	366	386	386	336	442	544	420	398	549	452	334	372	455	382	332	306	302	426	490	466	410
11	162	138	185	160	178	174	277	314	202	222	130	148	167	168	117	224	326	202	179	330	234	116	154	236	164	114	88	84	208	272	248	192
12	212	189	236	212	229	224	328	365	254	274	182	199	218	219	168	275	377	253	230	382	284	167	204	288	215	165	138	134	258	322	299	242
13	239	216	262	238	256	251	354	392	280	300	208	226	244	246	194	302	404	280	256	408	311	194	231	314	242	192	165	161	285	349	326	269
14	153	130	176	152	170	165	268	306	194	214	122	140	158	160	108	216	318	194	170	322	225	108	145	228	156	106	79	75	199	263	240	183
15	26	3	50	26	43	38	142	179	68	88	-4	13	32	33	-18	89	191	67	44	196	98	-19	18	102	29	-21	-48	-52	72	136	113	56

Touch presence:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	12	16	20	22	13	13	19	24	13	4	1	2	19	27	26
1	13	15	19	20	25	17	15	15	12	7	4	62	53	12	20	21
2	17	16	20	15	16	20	18	14	16	47	32	72	51	6	7	18
3	12	9	8	10	16	17	12	12	15	65	25	2	2	8	31	27
4	14	8	23	59	40	11	11	10	6	9	5	7	5	26	65	61
5	19	13	35	55	35	15	9	7	4	18	14	16	11	19	22	24
6	19	17	23	17	17	15	6	34	72	30	19	11	11	6	5	9
7	14	12	9	9	14	11	4	40	56	8	15	18	14	15	17	12

Detected touches:

	1	2	3	4	5
prob.	72	72	65	65	59
X	15.7	23.3	29.3	18.9	6.41
Y	13.4	3.41	8.2	5.76	9.53

X coordinates:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	23.2	23.3	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	23.3	23.2	0	0	0
3	0	0	0	0	0	0	0	0	0	18.9	0	0	0	0	0	0
4	0	0	0	6.41	0	0	0	0	0	0	0	0	0	0	29.3	29.2
5	0	0	0	6.47	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	15.7	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	16.1	0	0	0	0	0	0	0

Y coordinates:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	4.17	3.86	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	3.41	3.64	0	0	0
3	0	0	0	0	0	0	0	0	5.76	0	0	0	0	0	0	0
4	0	0	0	9.53	0	0	0	0	0	0	0	0	0	0	8.2	8.37
5	0	0	0	9.53	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	13.4	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	13.4	0	0	0	0	0	0	0

Grounded touch:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	2	7	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	101	58	0	0	0
3	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0
4	0	0	0	3	0	0	0	0	0	0	0	0	0	0	1	0
5	0	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0

Floating touch:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	99	93	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	42	0	0	0
3	0	0	0	0	0	0	0	0	78	0	0	0	0	0	0	0
4	0	0	0	98	0	0	0	0	0	0	0	0	0	0	98	101
5	0	0	0	98	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	98	0	0	0	0	0	0	0