DETECTING FIRM PRESSES ON CAPACITIVE SENSORS USING MACHINE LEARNING

Amaury Chevoir (S251893)

ABSTRACT

Atopical interfaces, a new type of human-machine interface, asks for more complex touch information. In particular the ability to detect a firm press is needed, that is when a finger is laid on a touch sensor and presses more firmly momentarily. To reduce cost and keep the hardware simple, capacitive sensors are often used on touch controlled devices. These sensors don't give pressure, but discharge intensities. This, with other hardware characteristics, can make it complicated to see if a finger is *pressing* harder. This project explores how a neural network can be used to detect to detect presses on capacitive sensors.

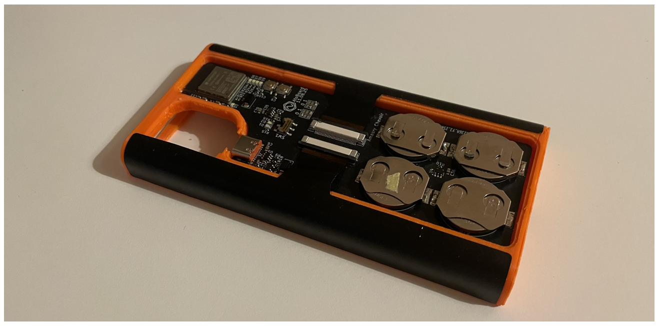

Fig. 1. Picture of the current prototype, a phone case

1. INTRODUCTION

From the video *How Atopical Interfaces work* [1], Micheal Helke:

> There are two fundamentally different types of user interface. One is topical. That is, the location you engage determines the functions you actuate, no matter what you engage it with, in particular which finger.
>
> The other type is atopical. Here the finger you engage the interface with determines the function, regardless of the location.

Atopical interfaces offer numerous advantages and could be very useful in some fields. For instance, visually impaired people could navigate a phone whithout having to press at specific locations, which can be complicated for them. The music controls of a car steering wheel could be atopical as well, to avoid reaching for a specific location.

I've been working on this interface for a year and find it really interesting. The implementation into a prototype has revealed some challenges, particularly in the finger press detection. The prototype consists of a smartphone case with the edges covered by capacitive sensors. Fingers are laid on the sides, and the goal is to trigger certain commands (taking a picture, pausing the music) by pressing more firmly momentarily. The capacitive sensors don't give the pressure applied by the fingers, but the discharge intensity in the human body, which depends mostly on the area of the contact patch. When a finger presses harder, it deforms and the contact area increases.

We can easily detect a firmer press of one finger this way. It has been first done by Apple with 3D Touch [2], and by Google *firm press*. The latter is briefly explained in this blog post [3]. In both cases the firm press is not used for a specific action, but to speed up a long press action. It makes it hard to really experience how it would feel like.

Furthermore, we're in a different situation, because all the fingers are laid at the same time, and on opposite sides. So even though only one finger is *instructed* (that is voluntarily pressed by the user), the fingers on the opposite side are pressing as well.

This makes it complicated to detect *which* finger is pressing, in particular with deterministic algorithms. These make a correct detection approximately 80% of the time, the rest being false positive, false negative, or inversions. They don't generalize well to other users. This is highly undesirable for a satisfying user experience, and machine learning could help improve this.

The goal is therefore to train a neural network on the usage data of the prototype, so that it can predict in real time if a finger is intentionally pressed. All the code for this project can be find on GitHub [4].

2. PRESS CHARACTERISTICS

Unfortunately, a press of a finger on the side of the prototype is not just its intensity increasing. Different effects take place.

As explained in the introduction, an action on one side of the prototype implies a reaction on the other side. For instance an increase in intensity from the index could be taken for an intentional press of it, while it's only the reaction to the thumb press.

But some interactions also occur on the same side. A biomechanism called "finger enslavement" links some finger pressure. It's an effect we can see while playing the piano for instance : when trying to press a key with the pinky, we sometimes press one with the ring at the same time.

We can see these two problematics in Figure 2. Even though only the pinky is *instructed*, we observe a rise in all of the fingers and the palm.

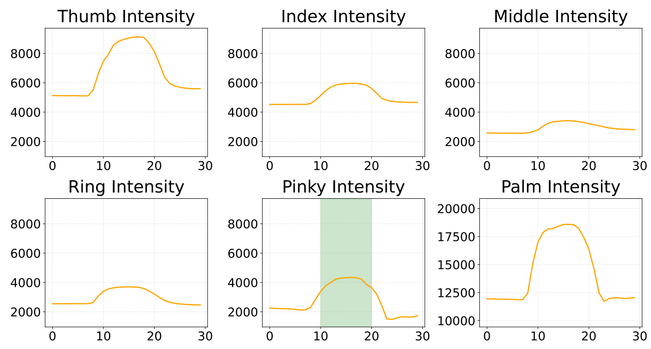

Fig. 2. Recording of a pinky press. Green part shows when the pinky is considered pressed

All of these patterns, interactions and irregularities are hard to predict with a deterministic algorithm, while a neural network is very well suited for this type of problems.

3. DATASET AND TRAINING

The data gathered contains windows of frames like we can observe for the pinky press in Figure 2. It was recorded at 20Hz with a custom application. The labels were added in real-time using an **event-driven method** : a metronome was ringing every 2 seconds, and the user had to press for the duration of the ring (0.5 second).

From this recording of thousands of frames, we want blocks of a few in input.

To simplify the recording, it was done finger by finger, so the dataset is composed of class blocks (50 presses of the thumb, then 50 presses of the index and so on). It is dangerous, because if we train in this order, the last class to be trained is going to erase what was learned with those before. Therefore we want to choose random indices during the training.

Here is how a frame looks like:

$$\underbrace{\texttt{8397;6532;3778;2944;2920;19293}}_{\text{intensities}}\texttt{;}\underbrace{\texttt{0;1;0;0;0}}_{\text{labels}}$$

We can see that the intensities can be high, up to 25000 to be precise. Yet a neural network doesn't train well on such big inputs. That's why I'm by dividing by 25000 so they are between 0 and 1.

Another aspect of the dataset is that the positive labels are not as present as negative ones. We have presses one fourth of the time, for only one finger out of five. Consequently only a 20th of the labels are positive, which creates class imbalance. But by trying different loss functions, like weighted or focal ones, I was suprised to see that it was still a standard binary cross entropy which was giving me the best results on most models.

4. MODEL CHARACTERISTICS AND IMPLEMENTATION

Before explaining the different models used in the project, it's important to emphasize that their goal will be to predict in real time wether a finger is pressing. It asks for very light architectures, with a reduced number of parameters. Furthermore, for better reactivity, I would like them to run directly on the microcontroller of the prototype. One of the leading solution for this is Tensor Flow Lite, so I used Tensor FLow for the models, to make it easier to migrate later.

5. EVALUATING PERFORMANCE

To evaluate the performance of the models, we can test them on a validation dataset, and compare the predictions with the real labels. We could then show these results in a confusion matrix, but because we have a very important proportion of negative labels (0), and consequently a very important proportion of true negatives, I discarded them and looked only at true positives, false positives and false negatives.

The performance is defined as the proportion of true positives in these three categories. But I quickly realized that this metric wasn't really representative of the real experience a user would have. If the label is wrong but only by being one frame (50ms) too early or too late, the user wouldn't consider it as an issue. In fact looking at wrong predictions showed me it was often in this situation.

I therefore landed on a "tolerant" performance evaluation, which would accept one frame shift in the label.

6. BASELINE MODEL

To evaluate the base performance, it's always good to start with a baseline model. It can help us determine what is important in the dataset or in which way to go for some parameters

I chose for this a simple Fully Connected Neural Network, which takes 30 frames as input (30x6), flatten them, and makes them go trough two layers (32 and 16 units), and then outputs the 5 probabilities.

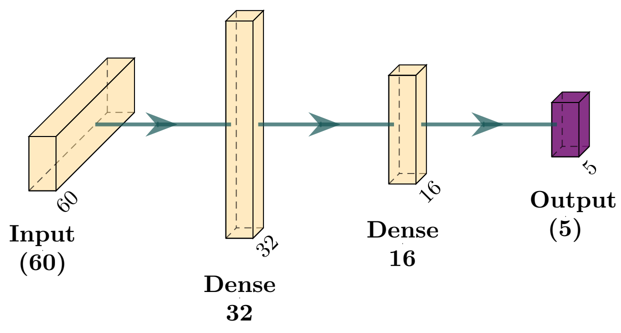

Fig. 3. The baseline model

6.1. Finding the right window length

At that time it was not clear how many past frames we should look at to detect a press. To find out, I trained the baseline model with different window lengths, from 1 to 40 frames (0.05s to 2s). The results are shown in figure 4.

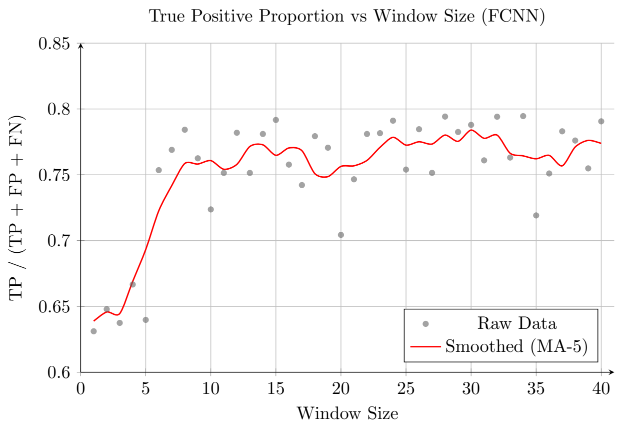

Fig. 4. Performance depending on the window length

As we can see the performance increases quickly after 5 frames, but stabilizes right after. Taking too much frames is not going to improve performance, but rather make the model very big. As we want to keep it light, 10 frames seems to be a good compromise and will be used for the following models.

6.2. Results

At the beginning I was getting about 80% in performance, but after tweaking the width of each layer, I obtained around 90% (with the architecture mentionned). It's above what I was getting with deterministic algorithm, but the model is very big for my constraints, with more than 2500 parameters.

7. CONVOLUTIONAL NEURAL NETWORK

In the hope of improving performance, and maybe reduce some parameters which could be redundant on multiple frames, I implemented a CNN. To detect a temporal feature in the previous frames, this architecture is well suited. It's going to consider the frames in the window similarly through its filters. Even though the data is two dimensionnal (fingers x frames), we don't want to make a 2D convolution as it would mix neighboring fingers. But as explained in the section 2, the interactions are between all the fingers and not only the neighboring ones. Therefore, the convolutions are going to be only in the time dimension.

I used two convolutionnal layers : the first with two filters, to capture simple features, like sudden rise or convexity of the curve, and the second with four filters to keep information after the MaxPooling.

Then, for the model to use interactions between fingers for a better prediction, I added a dense layer of 16 in width. The architecture can be seen in Figure 5.

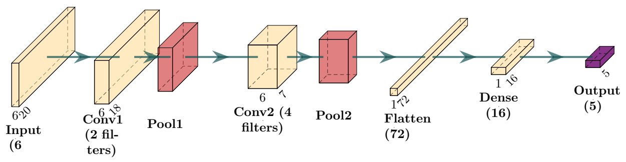

Fig. 5. Architecture of the CNN model

This model performs a little worse than the FCNN, with about 80% in performance. This can be slightly improved with more depth, but it would add too much parameters. Moreover, the point of this model it to show in which aspects a CNN shines and in which it leaves to desire. It leds to the following architecture.

8. RESIDUAL CNN

A press is caracterized by a quick rise in intensity (temporal feature), but also by the current intensities being higher than usual (instantaneous feature). The first type of features is analyzed well by convolutionnal layers, and the second type by dense layers. Moreover, the second type doesn't need a lot of frames because it's only looking at what are the intensities at the instant of the prediction.

To take advantage of these two features, I thought of an architecture where the complete window is processed through convolutions layers, and the last input frames are added to the output, like a residual network. Then, all these features are processed through dense layers to output the probabilities.

For reasons explained in the next section, it's very hard to predict correctly the 10% of labels which were falsely predicted by the baseline model. The goal was therefore on this model to get similar performance, while reducing significantly the number of parameters.

After some tweaking, the specific architecture that gave me the best results can be seen in Figure 6.

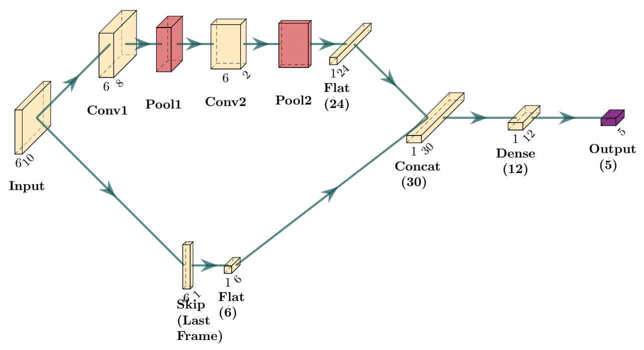

Fig. 6. Architecture of the Residual CNN model

It enabled going from the 2500 parameters of the baseline model, to 650 parameters, while keeping the same performance (90%). I tried making both FCNN and CNN with this few parameters, and the performance were below 70%. It shows that this architecture is really well suited for this problem.

9. DISCUSSION ON IMPROVEMENTS

I still have some false labels, and it's interesting to look at them to see what the model is missing. Those are most of the time in "weird" presses, for instance a very slow presses which didn't happen much in the training dataset.

To detect better those "weird" presses, the dataset should be expanded by more presses like this, which can still happen even if they are rarer. Currently, due to the method used to record data, we have a lot of very standard presses.

For the next version of this machine learning method, I'll try to gather more diverse data as it will noticeably improve the detection.

10. CONCLUSION

This project shows success in the use of neural networks to predict finger presses on capacitive sensors. Understanding the problem is a key to find the right architecture, especially when the size of the model is restricted.

I'm happy with results I got, as the model beats deterministic algorithms while still having room for improvement. I discovered machine learning this semester and wouldn't have thought being able to create something myself to answer a specific problem. I thank Jes Frellsen for accepting my project and giving me some pointers to get started.

11. REFERENCES

The book *Understanding Deep Learning* [5] served as reference for the theory, as well as the DTU course in which this project took place: 02456 Deep Learning [6], by Jes Frellsen.

[1] Micheal Helke, "How atopical interfaces work," `https://design-research.ch/explainer-video`, Accessed: 2025-10-26.

[2] Apple Inc., "Force touch," `https://en.wikipedia.org/wiki/Force_Touch`, Accessed: 2025-10-26.

[3] Google Research, "Sensing force-based gestures on the pixel 4," `https://research.google/blog/sensing-force-based-gestures-on-the-pixel-4/`, 2020, Accessed: 2025-10-26.

[4] Amaury Chevoir, "Deep learning project dtu," `https://github.com/Ora-ng3/deep-learning-project-dtu`, 2025, Accessed: 2025-12-06.

[5] Simon J.D. Prince, *Understanding Deep Learning*, MIT Press, 2023.

[6] Jes Frellsen, "02456 deep learning," DTU Compute Course, 2025.

DECLARATION OF USE OF GENERATIVE AI

This declaration **must** be filled out and included as the **final page** of the document. The questions apply to all parts of the work, including research, project writing, and coding.

- I/we have used generative AI tools: [yes]

If you answered *yes*, please complete the following sections. List the generative AI tools you have used:

- Copilot with Gemini 3 Pro

Describe how the tools were used:

What did you use the tool(s) for?

The tools were mostly used to generate scripts so visualize models results and test performance. They were also used to generate scaffolds of the training script, making it easier to focus on architecture optimization. In line completion was also used.

At what stage(s) of the process did you use the tool(s)?

Mostly during the implementation/coding part, and more rarely to ”brainstorm” by letting the tools suggest improvements on architectures.

How did you use or incorporate the generated output?

Either via in line completion or Copilot agent mode in VS-Code, which directly writes code in the editor