PyTouch: A Machine Learning Library for Touch Processing

Mike Lambeta[1], Huazhe Xu[2], Jingwei Xu[3], Po-Wei Chou[1], Shaoxiong Wang[4], Trevor Darrell[2], and Roberto Calandra[1]

Abstract— **With the increased availability of rich tactile sensors, there is an an equally proportional need for open-source and integrated software capable of efficiently and effectively processing raw touch measurements into high-level signals that can be used for control and decision-making. In this paper, we present PyTouch – the first machine learning library dedicated to the processing of touch sensing signals. PyTouch, is designed to be modular, easy-to-use and provides state-of-the-art touch processing capabilities as a service with the goal of unifying the tactile sensing community by providing a library for building scalable, proven, and performance-validated modules over which applications and research can be built upon. We evaluate PyTouch on real-world data from several tactile sensors on touch processing tasks such as touch detection, slip and object pose estimations. PyTouch is open-sourced at `https://github.com/facebookresearch/pytouch`.**

I. INTRODUCTION

A fundamental challenge in robotics is to process raw sensor measurements into high-level features that can be used for control and decision-making. For the sense of vision, the field of computer vision has been dedicated to study provide an algorithmic and programmatic method of understanding images and videos. In this field, open-source libraries such as PyTorch [1], CAFFE [2], and OpenCV [3] have enabled the acceleration of research and collectively brought together commonly used techniques in each perspective domain by providing unified interfaces, algorithms, and platforms. PyTorch and CAFFE have enabled researchers to develop and scale neural networks by reducing the amount of ground work required for implementing algorithms such as backpropagation or convolution functions. Furthermore, OpenCV provided benefits such as a collection of commonly used, tested and optimized functions. These frameworks, libraries and platforms all have an underlying goal in enabling further developments in their respective fields through abstracting lower level processes into building blocks for further applications.

With the increased availability of rich tactile sensors [4], [5], [6], [7], the sense of touch is becoming a new and important sensor modality in robotics and machine learning. Sensing the world through touch open exciting new challenges and opportunities to measure, understand and interact with the world around us. However, the availability of ready-to-use touch processing software is extremely limited – this results in a high entry bar for new practitioners that want to make use of tactile sensors, which are forced to implement their own touch processing routines. We believe that similarly to computer vision, the availability of open-source and maintained software libraries for processing touch reading would lessens the barrier of entry to tactile based tasks, experimentation, and research in the touch sensing domain.

To address this challenge we introduce PyTouch – the first open-source library for touch processing that enables the machine learning and the robotics community to process raw touch data from tactile sensors through abstractions which focus on the experiment instead of the low level details of elementary concepts. PyTouch is designed to be a high-performance, modular, and easy to use library aiming to provide touch processing functionalities "as a service" through a pre-trained model collection capable of kick-starting initial experiments to allowing end applications using new or variations of existing sensors the ability to apply transfer learning in levering the performance of the library. The software library modularizes a set of commonly used tactile-processing functions valuable for various down-stream tasks, such as tactile manipulation, slip detection, object recognition based on touch, etc. Furthermore, the library aims to standardize the way touch based experiments are designed in reducing the amount of individual software developed for one off experiments by using the PyTouch library as a foundation which can be expanded upon for future experimental and research applications.

While tools such as PyTorch and CAFFE currently exist and can be applied to touch processing, precursor development is required to support the necessary algorithms for the experiment and research needs, PyTouch provides a entry point using such tools specifically catered towards the goals of touch processing. PyTouch was designed to support both entry level users beginning their first steps in tactile sensing and power users well versed in this domain. PyTouch aims to generalize to diverse tactile input devices that might differ from each other in many design choices. This effort joins the recent efforts to standardize robotics research for better benchmarks and more reproducible results. Finally, in hand with the framework, we release a set of pre-trained models which are used by PyTouch in the background for tactile based tasks.

In this paper, we describe the architectural choices of library and demonstrate some of its capabilities and benefits through several experiments. Notably, for the first time we evaluate the performance of a machine learning model trained across different models of vision based tactile sen-

[1] Facebook AI Research, Menlo Park, USA
[2] University of California, Berkeley, USA
[3] Shanghai Jiao Tong University, China
[4] Massachusetts Institute of Technology, USA

sors, and show that this improves performance compared to models trained on single tactile sensors through the use of PyTouch itself to rapidly prototype these experiments.

The future facing goal of PyTouch is to create an extendable library for touch processing, analogous to what PyTorch and OpenCV are for computer vision, in a way which allows reliable and rapid prototyping of experiments and applications. Additionally, through the use of PyTouch as a set of touch processing building blocks and released pre-trained models we lessen the barrier to entry into touch processing while allowing researchers and experimenters to focus on the end goal of their application through proven algorithms and models supplied by the expansion of this library. We believe that this would beneficially impact the robotic and machine learning community by enabling new capabilities and accelerate research.

II. RELATED WORK

Developed fields in computer science build application and research upon frameworks. There are many frameworks existing today which bring together common paradigms used in their respective domains while providing benefit to the community. One example of such library is OpenCV [3] for the computer vision community, which is a well known and probably the most widely-used data processing library supporting computer vision topics.

Frameworks which are applicable to the domain of tactile touch sensing are far and few between or not available through open-source channels. Observing public repositories and open-source networks, we see very little open-source work pertaining to building a tactile processing framework. Although there is work in isolated instances which provide insight into specific problems such as interfacing with robotic skin and providing interfaces for applications [8], [9], there are no works which brings commonly used paradigms within the field of tactile touch sensing together or that provide general purpose tools. Previous attempts include studying friction models [10], physical property understanding [11], [12], [13], in-hand manipulation, determining physical manipulation characteristics and grasp success. [14] demonstrates individual capabilities related to tactile touch which allow for robotic learning in providing given forces to an object, determining density and texture through stirring actions, and various types of in hand or arm rotation of objects. Other work such as [15] learns robotic grasp success through an end-to-end action-conditional model based off of raw tactile sensor input data. [16] introduces TactileGCN which estimates grasp stability through a graph convolutional network. PyTouch aims at being a general purpose and open-source solution that combines useful computational tools and capabilities into a single framework and thereby proposes an agnostic library for tactile sensors regardless of their hardware or sensing modalities.

Another contribution of PyTouch is to push towards hardware standardization by providing flexible interfaces. Observing the tactile touch community, we saw common trends amongst the hardware used for acquiring touch input: barrier to entry due to cost, sparseness of available hardware arising from single one-off builds, and lack of reproducibility due to semi-closed or closed sourced designs. The DIGIT sensor [5] aimed to resolve these problems by providing a low cost, scalable and easily machine manufacturable tactile touch sensor. Amongst this, we see a pattern of individualized pieces of work in the tactile sensing field but a lack of unification that brings this work together into a library that future work can build upon. PyTouch is a step towards unifying the tactile processing field to create a proven, tested and modular open source software library available to all researchers and practitioners.

III. A LIBRARY FOR TOUCH PROCESSING

We now describe the desired specifications considered during the design of our touch processing framework, and the corresponding design choices adopted. In addition, we describe the software architecture of PyTouch, and several of its key features.

A. Desired Specifications

PyTouch aims to provide the same function in the domain of tactile touch processing through the introduction of an open-source library in which proven, tested and performance validated modules allow for scaling touch based applications, research and experimentation. This work aims to set forth a need for the unification of tactile processing based software which not only strengthens the field but also decreases the barrier to entry in designing applications and research which utilize tactile touch. As such, the main goals of PyTouch are to provide a library that is powered by pre-trained models through the "as a service" paradigm, enables transfer learning through the PyTouch for extending models for new hardware sensors, and deploys joint models which work across a variety of vision based tactile sensors. By centralizing touch processing tasks into a software library and through the models provided through PyTouch, this provides a path forward for sensor generalization. This is important due to the variance in produced hardware, image transfer layers and mechanical construction of lab-built sensors.

PyTouch is designed in mind to cater to first time and power users of the tactile touch community. High level abstractions of common goals are provided such as, but not limited to: “am I touching an object”, “is the object slipping”, or “what is the touch contact area of the object”. Diving deeper, the library can be used to perform analysis techniques such as augmenting the input sensor data, modifying the models used for inference, providing motion and optical flow outputs, or transfer learning techniques for extending the library to non-vision based sensors.

B. PyTouch Architecture

Pre-trained Models. PyTouch is built upon a library of pre-trained models which provide real-time touch processing functionalities. The pre-trained models are downloaded after initializing the library with the desired sensor, an example follows in Lst. 1. Currently, the library supports the DIGIT [5], OmniTact [6] and GelSight [4] sensors, but the

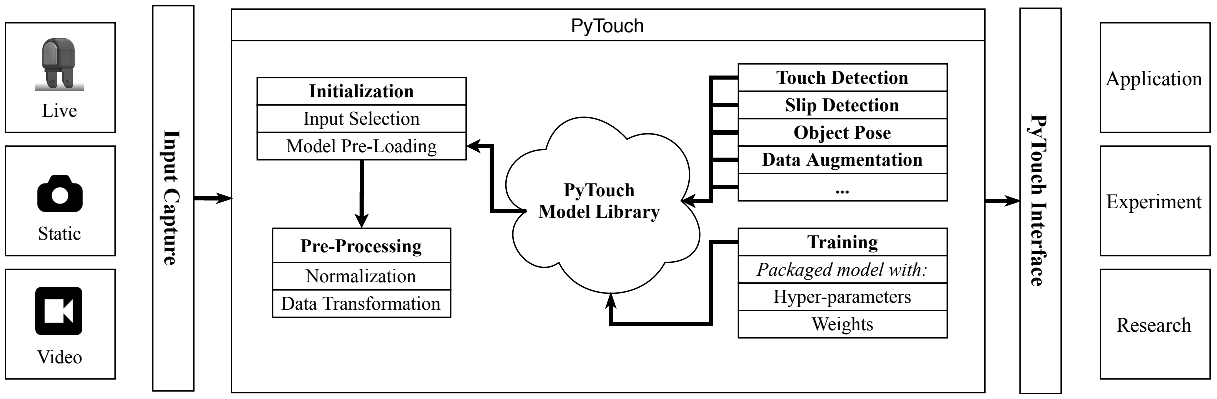

Fig. 1: PyTouch high level architecture where tactile touch processing is delivered to the end application ”as a service” through released pre-trained models.

Listing 1: Instantiating PyTouch with DIGIT sensor to detect touch and slip. PyTouch reduces the amount of code required to do advanced tasks such as these.

```
import pytouch as pt

digit_pt = pt.init(pt.devices.DIGIT, pre_loads={pt.models.
    TOUCH, pt.models.SLIP})

digit = ... # Connect and initialize DIGIT sensor
frame = ... # acquire frames from DIGIT sensor
touch, certainty = digit_pt.is_touched(frame)
slipping = digit_pt.is_slipping(frame)
print(f"Touching object: {touch}, with certainty: {certainty};
    Object Slipping: {slipping},")
```

library can be used to train models using data from other vision or non-vision based tactile sensors. At the current state, PyTouch provides the following functions: contact classification, slip detection, contact area estimation, and interfaces for training and transfer learning. By developing this library with modularity and scalability in mind, we plan to extend features and provide new models ”as a service” to the community and supporting new additions from the community itself. Additionally, deploying these models ”as a service” aids in the experimentation of different approaches to the same task by swapping pre-trained models for another without changing high-level software. This allows for performance benchmarking of real-world experiments for the purposes of creating a tactile task baseline.

Sensor Abstraction. PyTouch can be used in a number of ways such as, providing pre-captured video streams, static images, or live video captured from an input device. The library is instantiated with a known hardware device which in the background provides inference and functionality based on the hardware specific pre-trained models. The set of supported devices is extendable through the library as well, and they may be initialized by supplying a custom configuration file with models respective to that device. PyTouch provides the training routines for each library capability. Once trained, the weights can then be used through PyTouch for different hardware input devices without changing the application software.

Tactile Task Abstractions. PyTouch provides implementations of commonly used features in the tactile touch domain. The aim of the library is to reduce friction when experimenting with hardware implementations and robotic workspaces utilizing touch. Two interfaces are provided in initializing the library for use in applications. The first interface, Lst. 1, abstracts all internal modules into a single module while the second interface allows for individual module level access to each task for power users.

With each interface, an input device is registered which loads the appropriate pre-trained models. These pre-trained models are downloaded during initialization or when the appropriate class is instantiated in the background. The registered input device provides defaults as to which pre-trained models to use, and defines the internal data transformation to the models. Additionally, PyTouch supports extending the input devices for custom hardware through adding an additional input device module, or by specifying the custom device module which allows the user to provide custom pre-trained models. PyTouch is built upon a proven and widely used machine learning framework, PyTorch [1].

Model and Network Agnostic. One of the crucial benefits of PyTouch is the abstraction of high-level function calls which allows end-users to integrate their own models specific to experiments, sensors, or use our pre-trained models and perform fine-tuning. PyTouch introduces standardization of learning-based methods through the application programming interface of being able to swap models as needed, or change them through initialization parameters. PyTouch abstracts the low level details in order to allow new users to experiment through models served by the PyTouch team, or through the community. This ultimately allows PyTouch to be agnostic to model support in supporting conventional deep CNN models and through non-conventional learning based methods. Furthermore, this abstraction enables new users to touch processing to begin experimentation without having knowledge of model, network and parameter design choices but rather to focus on the experimentation itself.

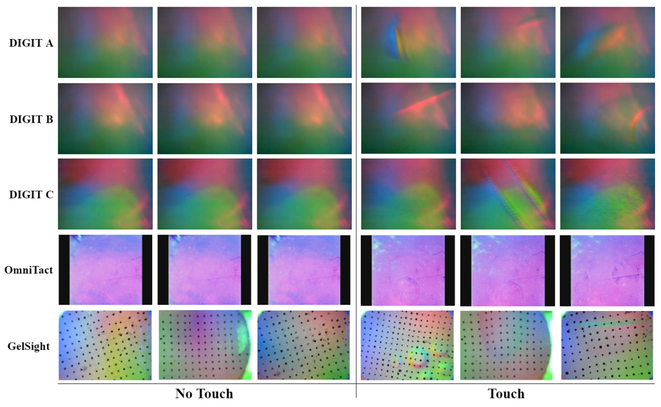

Fig. 2: Examples of data used to train our touch prediction models. The dataset include data across several DIGITs, OmniTact, and GelSight sensors showing different lightning conditions and objects of various spatial resolutions.

IV. EXPERIMENTAL RESULTS

We now provide several examples which outline the advantages in using the PyTouch library. Predominantly, the flexibility of the library to work with different vision-based tactile sensors such as DIGIT [5], OmniTact [6] and GelSight [4]. Secondly, the inherent design of the library to enable capabilities through the "as a service" paradigm. PyTouch releases pre-trained models which are tested, validated and supports additions via the community to extend library capabilities.

A. Touch Detection

A first functionality that PyTouch provides is touch detection. This is a basic functionality which is often used as a subroutine in more complicated manipulation sequences. However, training high-accuracy touch detection models can be difficult and require extensive data collections. Here we show that using one DIGIT sensor, we can determine if an object is being touched across multiple variations of that same sensor. The robustness of the library depends on the generalization of the model and due to manufacturing variances in the sensors and image transfer layers, no one sensor is exactly the same.

Formulation. Touch detection is formulated as an image classification task where given an input image $\mathbf{X}$, we assign a label y which is 0 for data without touch and 1 for data with registered touch.

Datasets. The dataset for DIGITs was collected touching random household and office objects of various spatial resolutions. This dataset was split by unique device serial numbers in contact, with over overall 2278 samples from 3 different sensors. The OmniTact dataset was provided by the authors of [6] and consisted of 1632 random presses of a single object. The GelSight samples were extracted from the dataset provided in [15], which consisted of before and after grasp images from two GelSight sensors. We used only a subset of the total dataset for a total of approximately 1000 images. The input images collected for DIGIT are images with size $240 \times 320 \times 3$, OmniTact images are $480 \times 640 \times 3$ and GelSight images are $1280 \times 960 \times 3$.

Model \ Sensor	DIGIT [5]	OmniTact [6]	GelSight [4]
Single models (no reference)	95.5 ± 1.2	98.4 ± 1.1	93.7 ± 1.2
Single models (with reference)	95.8 ± 1.3	98.5 ± 1.1	N/A
Joint model (no reference)	96.1 ± 1.0	99.1 ± 0.4	98.3 ± 0.6
Joint model (with reference)	96.2 ± 1.1	99.5 ± 0.3	N/A

TABLE I: Classification accuracy [%] of touch detection (mean and std) using cross-validation ($k = 5$). The joint models are trained with data from all three sensors: DIGIT, OmniTact and GelSight.

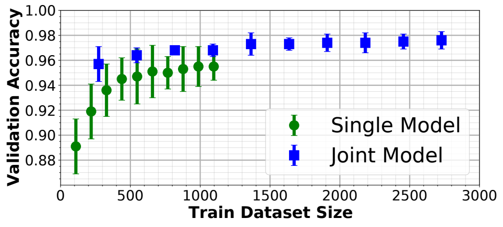

Fig. 3: Cross validation ($k = 5$) accuracy with varying train dataset size for single and joint models. With the same amount of data, training a joint model using data across multiple sensors (i.e., DIGITs, OmniTact and GelSight) results in better model performance compared to training from a single sensor.

Training and Architecture. The touch detection capability is based off of a pre-trained ResNet-18 model. Two modalities are used for training, the first is where images are provided to the model without a reference and the second is where the image is concatenated with a reference no-touch image. The reference is unique to each sensor which is a frame where the sensor is not touching any object. The input data is normalized and down-sampled to 64×64. The data was split to ensure an equal distribution of device serial numbers. For training with a reference no-touch image, a unique reference image was provided to the model and concatenated to each input image. Finally, a joint dataset was created to include all devices in order to show model robustness and generalization.

Results and Ablative Study. A model specific to each sensor and a joint model comprising of all sensor data across each unique sensor was trained. We show in Table I that the joint model performs better across a 5-fold cross validation test. The model trained for the OmniTact sensor shows higher performance than the DIGIT model, however, observing the data in the OmniTact dataset, sample images shown in Fig. 2, we see that the breadth of sample uniqueness is minimal. We also show the effects of constraining the train dataset sizes ranging from 10% to 100% of the total dataset size in Fig. 3 that creating a joint model which uses multiple sensors results in better performance while using a similar amount of data in the training dataset. To our knowledge, this is the first study on multiple sensors manufacturers and sensor variants to show benefit of a touch processing library.

Furthermore, we show a direct benefit of using PyTouch

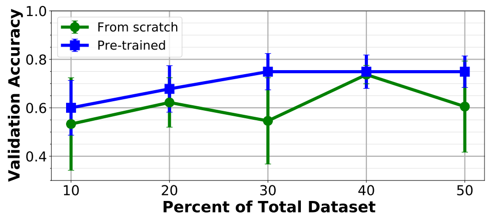

Fig. 4: Cross validation ($k = 5$) accuracy with varying train dataset size when training a model from scratch or starting from the pre-trained joint model. The validation is performed on dataset of size 482 from a DIGIT sensor. Using the joint model to fine-tune against a new dataset results in better model performance with lower dataset sizes compared to training a the model from scratch for the same wall-clock time of $30\,\mathrm{s}$. Thus PyTouch can provide benefit to end applications when using the "as a service" pre-trained models.

in regards to fine-tuning a smaller dataset acquired from a new sensor on the joint model. A large dataset was collected from a new sensor and then constrained from 50% to 10% of the initial size. The results on the touch detection task show that using a pre-trained model can both increase the final performance and the training wall-clock time when using our pre-trained model and then fine-tuning, compared to a model trained exclusively from new data. We ablate this experiment over different dataset size to show how the advantages are more pronounced the smaller the dataset used for fine-tuning is. In Fig. 4 we show the results of the fine-tuned model compared to the model trained from the dataset alone for the same wall-clock time of $30\,\mathrm{s}$. Furthermore, we shown in Fig. 5 that introducing new sensors using PyTouch as a base results in better performance with less samples through transfer learning from the released pre-trained models PyTouch provides.

The PyTouch frame is designed with the goal to support real time experimentation with input into the touch detection model resulting in an average performance of $5.89 \pm 0.46\,\mathrm{ms}$. Tactile input sensors are most commonly seen with input capture rates of 24 to 60 frames per second which places each frame between 17 and $41\,\mathrm{ms}$. This enables PyTouch for use in real time applications which require high frame rates of up to $140\,\mathrm{fps}$.

B. Contact Area Estimation

When grasping an object with a single or multiple sensor it is useful to know the surface contact area of the object against the sensor. We show an example of this in Fig. 6 where a number of object contact areas are estimated with single and multiple point features using a DIGIT [5] tactile sensor. The contact area estimated is not referenced to a ground truth, but rather provides the areas of contacts and estimates the direction of contact with respect to the surface of the touch sensor.

PyTouch provides the contact area estimation capability for many input formats such as raw sensor input, video and static images. Additionally, providing centroid information and estimated size of the semi-major and semi-minor axis of the fitted ellipse.

C. Library Experimentation

There are a number of parameters available to physical hardware sensor design used in the field of tactile touch sensing and due to the cumbersome nature of hardware development, progress and iteration is often slow and costly. Often, it is valuable to experiment with physical based parameters prior to building a full prototype to reduce the number of iteration cycles. PyTouch proposes a module which allows in the experimentation of these physical based parameters against previous benchmarks obtained by the joint and single sensor models. We show in Fig. 7 that for the DIGIT [5] sensor which relies on RGB lighting that a physical hardware change to using monochromatic light results in less performance compared to chromatic light.

D. Slip Detection

Slip detection is crucial for many robotics tasks e.g.in-hand manipulation, grasping. In PyTouch, we incorporate a slip detection module as a pre-trained model. We also provide APIs for training customized models. The module design and implementation are described as follows.

Formulation. We formulate this task as a video classification task: given an image sequence $\mathbf{X} = \{x_1, \cdots, x_T\}$, we assign a label y from a discrete set $\mathbb{Y}$. We additionally label the slip starting and ending timestamp t_s and t_e for extracting shorter sub-sequences. Specifically, the x_t is an RGB image with size $240 \times 320 \times 3$, the label y is a binary label with 1 representing slip and 0 representing non-slip. We note that both rotational and translational slip are labeled as slip without distinction.

Dataset. We collected 40 objects with a large spectrum of surface finishes, roughness, and features. This can be thought of as "texture-ness" from the heavily textured pen holder to an almost textureless plastic bottle. For each object, we collected 10 slip sequences and 10 non-slip sequences. Each sequence is consist of 128 frames in total. All the data are collected by using DIGIT sensor to slide or touch an object with 30 Hz frequency [5].

Training and Architecture. We used ResNet-18 and ResNet-18-based 3D convolutional neural network as our architecture. For both architecture, we concatenate all the images as input. We use cross-entropy loss as the objective of the binary classification task. We follow the standard approach to normalize the input data, downsample the image to 112×112 and use BatchNorm for better performance. We use Adam optimizer with learning rate 0.001. Since it is video classification, we compare the results in two data split setting. The first setting is "split by sequence" where we use 20% of the data from each object as test sequences. The second split is "split by objects" where we withhold 20% objects as test data. We note that the latter is much harder than the former due to the need of generalization.

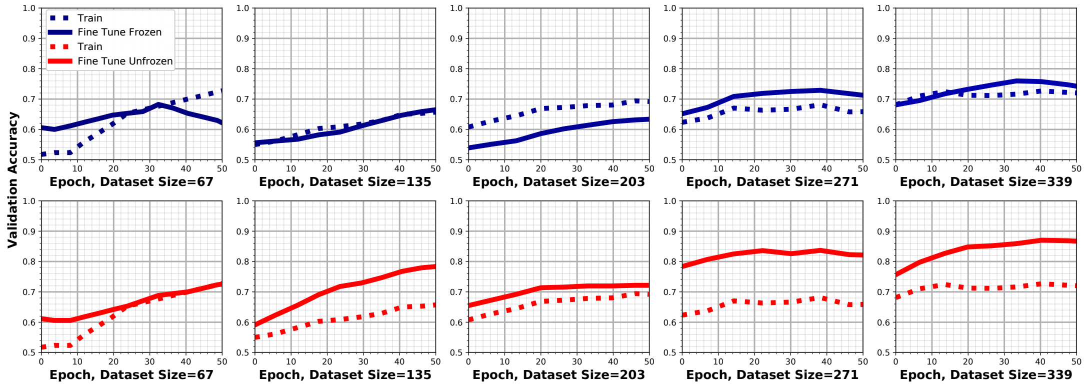

Fig. 5: Comparison of model trained just from the dataset collected from the GelSight [4] sensor (”Train”) against model trained using the pre-trained joint model and fine-tuning the last 3 layers (”Fine Tune Unfrozen”) or just the last layer (”Fine Tune Frozen”). The results show using the pre-trained and fine-tuning the last 3 layers model significantly improve the performance of the models. This suggests that the pre-trained model learned useful features that can generalize to different sensors.

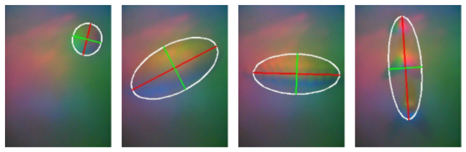

Fig. 6: Examples of contact area estimations using 4 objects: glass ball, pen, cable and fork

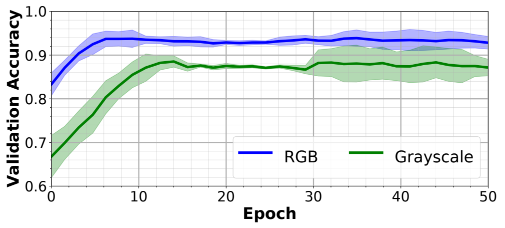

Fig. 7: DIGIT [5] sensor loss in performance when comparing monochromatic light to RGB lighting

Results and Ablative Study. In Table II, we show the classification accuracy with all 128 frames or 12 frames as inputs. We notice that the original ResNet-18 [17] outperforms 3D conv net. We attribute this to the high variance of slip velocity in the dataset that confuse the 3D convolutional kernel. We also find that the accuracy on “split by object” is lower than “split by sequence”, which confirms our claim about the requirement for better generalizability in the former split. In the same table, we also extract a shorter sequence with 12 frames (approximately 400 milliseconds) as in [18] from the starting timestamp t_s to t_{s+12}. We find the performance drops slightly to around 90%.

	Split by Sequences [%]		Split by Objects [%]	
	$len = 12$	$len = 128$	$len = 12$	$len = 128$
ResNet-18	90.6	97.9	89.2	96.3
3D Conv	N/A	91.7	N/A	81.2

TABLE II: Slip detection accuracy averaged over all trials with different architectures, number of frames and dataset split methods.

V. CONCLUSION

As the field of touch sensing evolves, a need arises for advanced touch processing functionalities capable of efficiently extracting information from raw sensor measurements. Prior work provided some of these functionalities, but typically ad-hoc – for single sensors, single applications, and not necessarily with reusable code (e.g., open-source). To better support and grow the tactile sensing community, we believe that is necessary the creation of a software library which brings together advancements in the touch processing field through a common interface. To address this need, we introduce PyTouch, a unified library for touch processing. Our goal with this library is to enable academics, researchers and experimenters to rapidly scale, modify and deploy new machine learning models for touch sensing. In this paper, we detailed the design choices and the corresponding software architecture of this library. PyTouch is designed with open-source at the core, enabling tactile processing ”as-a-service” through distributed pre-trained models, and a modular and expandable library architecture. We demonstrate several of the feature of PyTouch on touch detection, slip and object pose estimation experiments using 3 different tactile sensors, and we show that a unified tactile touch library provides benefit to rapid experimentation in robotic tactile processing. We open-source PyTouch at `https://github.com/facebookresearch/pytouch`, and aim in continuing the evolution of this library to further the field of touch processing.

ACKNOWLEDGMENT

We thank Omry Yadan for insightful discussions and assistance to integrate Hydra [19], Frederik Ebert, Stephen Tian and Akhil Padmanabha for providing the OmniTact dataset, and Joe Spisak for helping with community building and project vision.

References

[1] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[2] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[4] W. Yuan, S. Dong, and E. H. Adelson, "Gelsight: High-resolution robot tactile sensors for estimating geometry and force," *Sensors*, 2017.

[5] M. Lambeta, P.-W. Chou, S. Tian, B. Yang, B. Maloon, V. R. Most, D. Stroud, R. Santos, A. Byagowi, G. Kammerer, D. Jayaraman, and R. Calandra, "DIGIT: A novel design for a low-cost compact high-resolution tactile sensor with application to in-hand manipulation," *IEEE Robotics and Automation Letters (RA-L)*, vol. 5, no. 3, pp. 3838–3845, 2020.

[6] A. Padmanabha, F. Ebert, S. Tian, R. Calandra, C. Finn, and S. Levine, "Omnitact: A multi-directional high-resolution touch sensor," *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[7] A. C. Abad and A. Ranasinghe, "Visuotactile sensors with emphasis on gelsight sensor: A review," *IEEE Sensors Journal*, vol. 20, no. 14, pp. 7628–7638, 2020.

[8] S. Youssefi, S. Denei, F. Mastrogiovanni, and G. Cannata, "Skinware 2.0: a real-time middleware for robot skin," *SoftwareX*, vol. 3, pp. 6–12, 2015.

[9] ——, "A real-time data acquisition and processing framework for large-scale robot skin," *Robotics and Autonomous Systems*, vol. 68, pp. 86–103, 2015.

[10] H. Culbertson, J. J. Lopez Delgado, and K. J. Kuchenbecker, "The penn haptic texture toolkit for modeling, rendering, and evaluating haptic virtual textures," 2014.

[11] Y. Gao, L. A. Hendricks, K. J. Kuchenbecker, and T. Darrell, "Deep learning for tactile understanding from visual and haptic data," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 536–543.

[12] A. Burka, S. Hu, S. Helgeson, S. Krishnan, Y. Gao, L. A. Hendricks, T. Darrell, and K. J. Kuchenbecker, "Proton: A visuo-haptic data acquisition system for robotic learning of surface properties," in *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2016, pp. 58–65.

[13] A. Burka, S. Hu, S. Krishnan, K. J. Kuchenbecker, L. A. Hendricks, Y. Gao, and T. Darrell, "Toward a large-scale visuo-haptic dataset for robotic learning," in *proceedings of the Workshop on the Future of Datasets in Vision at Computer Vision and Pattern Recognition (CVPR)*, 2015.

[14] B. Belousov, A. Sadybakasov, B. Wibranek, F. Veiga, O. Tessmann, and J. Peters, "Building a library of tactile skills based on fingervision," 2019.

[15] R. Calandra, A. Owens, D. Jayaraman, W. Yuan, J. Lin, J. Malik, E. H. Adelson, and S. Levine, "More than a feeling: Learning to grasp and regrasp using vision and touch," *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3300–3307, 2018.

[16] A. Garcia-Garcia, B. S. Zapata-Impata, S. Orts-Escolano, P. Gil, and J. Garcia-Rodriguez, "Tactilegcn: A graph convolutional network for predicting grasp stability with tactile sensors," in *International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8.

[17] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[18] J. Li, S. Dong, and E. Adelson, "Slip detection with combined tactile and visual information," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7772–7777.

[19] O. Yadan, "Hydra - a framework for elegantly configuring complex applications," Github, 2019. [Online]. Available: https://github.com/facebookresearch/hydra