Towards a Universal Classifier for Crystallographic Space Groups

Nouamane Laanait, Junqi Yin, Albina Borisevich
Oak Ridge National Laboratory

State-of-the-art electron microscopes produce focused electron beams with atomic dimensions that allow the capture of diffraction patterns arising from the interaction of incident electrons with nanoscale material volumes. Backing out the local atomic structure of materials requires compute- and time-intensive analyses of these diffraction patterns (i.e., convergent beam electron diffraction [CBED]). Traditional analyses of CBED requires iterative numerical solutions of partial differential equations and comparison with experimental data to refine the starting material configuration. This process is repeated anew for every newly acquired experimental CBED pattern and/or probed material.

For this dataset, we used newly developed multi-GPU and multinode electron scattering simulation codes on the Summit supercomputer to generate CBED patterns for over 60,000 solid-state materials, representing nearly every known crystal structure. The overarching goals of this data challenge are to (1) explore the suitability of machine learning algorithms in the advanced analysis of CBED and (2) produce a machine learning algorithm capable of overcoming intrinsic difficulties posed by scientific datasets.

The dataset is split across multiple HDF5 files and an accompanying Jupyter Notebook provides a detailed description on how to navigate the file structure to access the data samples and the associated materials properties. Briefly, a data sample from this dataset is given by a 3D array formed by stacking three CBED patterns simulated from the same material at three distinct material projections (i.e., crystallographic orientations). Each CBED pattern is a 2D array (512×512 pixels) with float 32-bit image intensities.

Associated with each data sample in the dataset is a host of material attributes or properties which are, in principle, retrievable via analysis of this CBED stack. These consist of the crystal space group to which the material belongs, atomic lattice constants and angles, and chemical composition, among other attributes. Of note is the crystal space group attributed (or label). All possible spatial arrangements of atoms in any solid-state (crystal) material obey symmetry conditions described by 230 unique mathematical discrete space groups.

The Data Challenge questions revolve around developing and implementing a machine learning algorithm to predict a material's space group, essentially a classification task. The dataset is, however, heavily imbalanced with regard to number of data samples per class. This imbalance is not an artifact; instead it reflects the reality that most known materials have low symmetry and as such are not uniformly distributed across the 230 space group classes.

Challenge Questions:

1. Develop a machine learning algorithm for space group classification of CBED data.
2. Implement proper machine learning techniques to overcome data/label imbalance and show how it affects the performance of the machine learning algorithm in question 1.

3. Implement a machine learning algorithm for multitask prediction of a space group, in addition to other material structural properties, and show how it affects the performance of the ML algorithm in question 1.

Notes on the challenge questions:

- A participant may choose to do questions 1 and 2 or questions 1 and 3. Completing all three questions is optional.
- Regarding approaches to question 2, our preference is for machine learning techniques (e.g., loss weighting, model ensembles, active learning, decision boundary analysis with GANs [generative adversarial networks]) in lieu of brute-force data augmentation approaches (e.g., mixup, random erasing).
- If a deep learning model is used by the participant, our preference is for the implementation to use one of the following frameworks: MXNet, Pytorch or TensorFlow.
- Our preference is for the machine learning algorithms to be implemented in one of the following languages: Python, C/C++, or Julia.