UC Davis

UC Davis Previously Published Works

Title

Automated Labeling of Electron Microscopy Images Using Deep Learning

Permalink

<https://escholarship.org/uc/item/8m19p87t>

ISBN

9781728101804

Authors

Weber, GH

Ophus, C

Ramakrishnan, L

Publication Date

2019-02-08

DOI

10.1109/MLHPC.2018.8638633

Peer reviewed

Automated Labeling of Electron Microscopy Images Using Deep Learning

Gunther H. Weber^{*†}, Colin Ophus[‡], and Lavanya Ramakrishnan^{*}

^{*}Computational Research Division, Lawrence Berkeley National Laboratory,
One Cyclotron Road, Berkeley, California 94720, Email: {GHWeber, LRamakrishnan}@lbl.gov

[†]Department of Computer Science, University of California, Davis
One Shields Avenue, Davis, California 95616

[‡]National Center for Electron Microscopy, Molecular Foundry, Lawrence Berkeley National Laboratory,
One Cyclotron Road, Berkeley, California 94720, Email: CLOphus@lbl.gov

Abstract—Searching for scientific data requires metadata providing a relevant context. Today, generating metadata is a time and labor intensive manual process that is often neglected, and important datasets are not accessible through search. We investigate the use of machine learning to generalize metadata from a subset of labeled data, thus increasing the availability of meaningful metadata for search. Specifically, we consider electron microscopy images collected at the National Center for Electron Microscopy (NCEM) at the Lawrence Berkeley National Laboratory (LBNL) and use deep learning to discern characteristics from a small subset of labeled images and transfer labels to the entire image corpus.

Relatively small training set sizes and a minimum resolution of 512×512 pixels required by the application domain pose unique challenges. We overcome these challenges by using a simple yet powerful convolutional network architecture that limits the number of free parameters to lower the required amount of computational power and reduce the risk of overfitting. We achieve a classification accuracy of approximately 80% in discerning between images recorded in two operating modes of the electron microscope—transmission electron microscopy (TEM) and scanning transmission electron microscopy (STEM). We use transfer learning—i.e., re-using the pre-trained convolution layers from the TEM vs. STEM classification problem—to generalize labels and achieve an accuracy of approximately 70% despite current experiments being limited to small training data sets. We present these predictions as suggestions to domain scientists through a web-based tool to accelerate the labeling process with the goal of further validating our approach and improving the accuracy of automatically created labels.

I. INTRODUCTION

Scientific data search is becoming critical to enable users to find relevant data and advance discoveries on domain boundaries. We need metadata that provides a context about the data to enable search on scientific data. For example, when searching images from electron microscopes, scientists may only be interested in images with certain characteristics, such as those that were recorded at “atomic resolution” or those that show “nanoparticles.” While it is possible for scientists to add such labels manually, doing so for hundreds or thousands of images would be an extremely tedious time and labor intensive process. As a consequence, adding labels is often postponed or neglected, making scientific data sets difficult or impossible to search.

The ScienceSearch project’s [1] goal is to generate metadata for scientific data sets automatically and by doing so making scientific data more easily searchable. The project considers a wide range of sources (proposals, images, file names, calendar entries, etc.) and techniques (natural language processing, deep learning, etc.). In this paper, we focus on training a convolutional neural network (CNN) using a subset of hand labeled images from the National Center for Electron Microscopy (NCEM) in order to classify unlabeled images and create metadata tags to enable search.

Specifically, this paper focuses on addressing two specific challenges in developing an approach to generalize labels. First, we require relatively large resolution (at least 512×512 pixels) to classify images accurately due to needs of the application. Second, we have a relatively small training data set size available, consisting only of hundreds of images. Current deep learning approaches rely on thousands or millions of images and thus, we need to extend current approaches for our work. We propose using a simple yet powerful CNN architecture, consisting of a few convolutional layers followed by a pooling layer and a few densely connected layers to handle large image sizes. The simple CNN also reduces the risk of overfitting for small sample data set sizes. Our method involves a series of steps that includes automated generation of labels based on knowledge of the operating modes of the microscopes that is used with transfer learning to pre-train the convolutional layers of the CNN. Our classifier returns its confidence in label prediction, i.e., the estimated probability that the tag applies to the image, which is in turn used to rank relevant images in the search. The larger automatically generated training set is also used to identify appropriate parameters for the binary classifier CNN network architecture.

Specifically, our contributions in this paper are:

- Determining the required image resolution for making accurate predictions and creating an appropriate training data set for image classification for the National Center for Electron Microscopy (NCEM).
- Developing an approach that uses machine learning to transfer label from a manually curated image data set to all images in the corpus. Most notably, our approach uses transfer learning to reduce the number of free

Fig. 1. Steps taken to implement our approach.

parameters in the convolutional neural network (CNN), thus lowering the computational complexity for learning features of individual labels as well as reducing the risk of overfitting.

- A preliminary parameter study of CNN architectures to identify the appropriate architecture for the NCEM classification task and a more detailed analysis of the chosen CNN architecture.

Figure 1 shows the steps to implement our classifier. First, we determine and discuss the appropriate image resolution for the classification task and automatically create a training data set for CNN evaluation and transfer learning (described in Section II). We use a CNN and transfer learning to generalize labels, which is described in detail in Section III. We then identify an appropriate CNN architecture and evaluate the chosen architecture more thoroughly in Section IV. Our evaluation shows that our CNN architecture is able to achieve an accuracy of 80% on automatically generated microscope operating mode validation and withheld test data sets. Furthermore, initial transfer learning experiments using the CNN with pre-trained convolution layers, achieved accuracies of 70–80% on the validation data set and approximately 68–69% on a withheld test data set. Predicted labels are subsequently presented to domain scientists through a web-based tool, for verification with the goal of speeding up the labeling process and producing sufficient additional test data for improving the accuracy of our approach.

II. BACKGROUND: DATA AT NCEM

In this section, we describe the NCEM use case and describe and motivate the resolution needs for NCEM data sets. We also describe how we use NCEM’s operating procedures to acquire automatically labeled image data to evaluate CNN architectures and create pre-trained convolution layers for transfer learning.

A. NCEM Metadata Requirements

NCEM is a electron microscopy user facility that operates 11 electron microscopes at the Molecular Foundry at Lawrence Berkeley National Lab. It offers a wide range of capabilities for high resolution materials characterization such as *in situ* nanoindentation, spin-polarized low-energy microscopy, or tomography [2]. NCEM provides a concrete use case for our work due to the variability of its data sets. A single microscope can produces images at a wide range of scales—ranging from atomic to lower resolutions. The type of data collected at NCEM is representative of a large number of electron microscopy facilities, in particular, but also for the

type of data collected at other experiment facilities such as light sources.

NCEM stores terabytes of micrographs, i.e., microscopy images, in millions of files, and a significant percentage of them lack metadata. Today, users store metadata in a variety of ways, e.g., in notebooks and as naming conventions of files and directories. Lack of metadata makes it harder to use the data beyond its initial use.

Our project aims to provide search over NCEM’s data set to help users to find micrographs of their interest. In this paper, we focus on using deep-learning to transfer labels from manually curated images to the entire image corpus. NCEM scientists manually assign meaningful labels to a randomly selected subset of all images produced by the center’s microscopes. These labels may be based on scale (e.g., “atomic resolution” vs. “low resolution”), microscope operation mode (“transmission electron microscopy (TEM)” vs. “scanning transmission electron microscopy (STEM)”), type of object imaged (“nanowire, nanoparticles,” etc.), etc. The exact nature of these labels is often application specific, but the general idea is for scientists to assign labels that support meaningful search queries on the images. Once scientists have labeled a sufficient number of images—with this sufficient number still being determined—our goal is to transfer these labels to all images and make the entire image corpus more easily searchable. We use deep learning to discern the features that characterize a label and transfer the label to the entire corpus of images from the center’s microscopes.

B. Appropriate Input Image Resolution

We consider images from the TEAM I microscope at NCEM, a double-aberration-corrected (scanning) transmission electron microscope (S/TEM) capable of producing images with 50 pm resolution. The microscope has a 2048×2048 slow-scan CCD camera for use in TEM imaging, and can record data with arbitrary image sizes for STEM imaging. Typically, the maximum image resolution in the collection is 4096×4096 pixels. Commonly used resolutions are 512×512 , 1024×1024 or 2048×2048 pixels.

Most image classification approaches rely on a relatively small, fixed input size. For example, the ImageNet database [3], [4] is typically scaled/cropped to 256×256 pixels with many image classification applications considering even lower resolutions. We also need to choose an appropriate, fixed input resolution for our classifier. We determine the appropriate image resolution based on the assumption that relevant features in the image, which enable an application scientist to classify images, must be preserved, i.e., that a CNN can only classify images reliably if a human domain scientist

Fig. 2. Cropped region from a microscopy image recorded at 1024×1024 scaled to lower resolutions (upper row) and spectra (FFT) of the entire image. We show all resolutions in the same size, to make images more easily comparable. The 512×512 version still preserves the overall structure of the image, while already smoothing details out. Scaling to 256×256 or 128×128 removes important details and creates a different feature set due to aliasing.

can do so as well. Considerations from the application domain suggest that the minimum viable resolution for a human to distinguish features in a typical atomic-resolution image is 512×512 pixels. Atomic positions will be visible at this size when using typical fields of view of ≈ 10 nm.

Further, we substantiate this minimum resolution requirement by considering a microscopy image with an original size of 1024×1024 and scaled down versions of this image, see Figure 2. The regular structure of these images also shows up in the fast Fourier transform (FFT), which is often used in data analysis. As the image shows, features and overall data structure are preserved—though smoothed/blurred out—when halving the resolution. Furthermore, while the exact location of the peaks in the FFT changes, the overall structure of the FFT is still similar to the original image. However, this resolution is already at the limit of resolving important features, and further reductions in resolution change the overall image structure—along with the structure of the FFT—due to aliasing artifacts. The considerations from the application domain and analyses of the image suggest a minimum resolution of 512×512 pixels as reasonable input for the classifier.

C. Automatic Creation of a Labeled Data Set

Labeling is a time-consuming process and hence, we currently have only access to approximately 300 manually labeled images, an insufficient training data set size to choose an appropriate classifier or train all layers in a convolutional neural network (CNN). Due to operational procedures at NCEM, images using two operating modes of a microscope—

transmission electron microscopy (TEM) and scanning transmission electron microscopy (STEM)—are saved using two different file formats (SER and DM), and we can label images as TEM or STEM based on the file format.

We identify all 2D images collected by the TEAM I microscope from beginning February 2015 through the end of November 2017 to create a training data set. We include all images that are larger than 512×512 pixels and exclude images from movies (as movies often consist of many largely identical images). The image subset identified this way consisted of 8681 images in STEM mode (i.e., saved as SER files) and 1662 images in TEM mode (i.e., saved as DM files).

The lower number of usable images in TEM mode limited the size of our training and validation data sets, as we were aiming to generate an even distribution of both labels to prevent the classifier learning to just predict the more common format. We select a training and validation data set by choosing 500 random TEM and 500 random STEM images each, totaling 1000 images for each training and validation data set.

We create an independent test data set for final validation using 5,915 images collected by the TEAM I microscope after creating our initial training and validation data sets (i.e., starting from December 2017). Again, we select 500 random TEM and 500 random STEM images for a total of 1000 test images.

III. LABEL GENERALIZATION APPROACH

We consider each label separately and train a CNN as binary classifier on the manually labeled images to predict whether

Fig. 3. We transfer labels by training a binary classifier for each label “X” based on the “ground truth” data set of manually labeled images. Subsequently, we run this classifier on the remaining images of the data set and add label “X” as suggestion to all images where the classifier predicts that the label applies. (Color key: Red = input data, blue = classifier/operations, orange = output data.)

the label applies to a given image or not (see Figure 3 and Section III-A). Since NCEM scientists may assign many labels to images, and we train a CNN for each label separately, computational costs can quickly escalate. Furthermore, since training data set sizes for labels are small, there is a risk of overfitting the CNN. We use transfer learning (see Figures 4, 5 and Section III-B) to lower the computational complexity for learning features and reduce the risk of overfitting. Specifically, we pre-train the convolutional layers on the much larger, automatically generated TEM vs. STEM data set from Section II-C. We then “freeze” the convolutional layers and retrain only the densely connected layers for each label. Since the densely connected layers have a relatively small number of weights compared to the convolutional layers, the computational cost for each label is significantly reduced, and HPC resources are utilized more efficiently. Using transfer learning to reduce the number of free parameters also lowers the risk of overfitting [5], [6]. Next, we describe these steps in more detail.

A. Using a Binary Classifier to Transfer Labels

We consider each label separately when generalizing labels. Specifically, we do not take any potential relationships between labels into account as we currently do not have an ontology for the labels that could provide information such as mutual exclusion. Next, we take the “ground truth” data set, i.e., all images labeled by domain scientists for which we know for certain whether that label applies or not. We split this data set randomly into a training (55% of the entire ground truth data) and a validation data set (45% of the ground truth data set). For a preliminary validation of our approach, we ask the domain scientists to label a small set of additional images as test data set. While these training and test data sets are currently small, our goal is to use CNN predictions as suggested labels to be confirmed or rejected by domain scientists. In future work, we plan to further validate these results subsequently based on statistics of how many predictions were accepted or rejected and/or to create a larger test set once we have more manually labeled images as result of our predictions.

Subsequently, we train a CNN as binary classifier on the training data set until we achieve a classification accuracy

on the validation data set above a user specified threshold. We use thresholds of approximately 70% to 80% for our experiments, currently manually tuning this parameter for each label. Finally, we run the trained classifier on the entire unlabeled image corpus and let it predict whether a particular label applies to a given image and import each prediction into the manual image label curation tool for confirmation (or rejection) by NCEM domain scientists.

B. Transfer Learning to Reduce Computational Complexity

Despite using a simple CNN architecture, based on a significantly simplified AlexNet [7], training is still expensive, which poses a problem when training many individual CNNs—one for each label. Furthermore, there is still the potential problem of overfitting when training this network on relatively small labeled image data set. One way to alleviate this problem is transfer learning, i.e., re-using already trained layers from another classification problem—effectively freezing their weights—and only retraining the remaining subset of layers (or replacing them with a different network) [5], [6]. A common approach freezes (and re-uses) all convolutional layers and re-trains the dense layers. For example, this approach is used to repurpose a CNN trained on ImageNet for other classification tasks, assuming that when training on ImageNet, the convolutional layers learn general image features, which are useful for a wider range of classification tasks. Thus, just retraining the dense layers responsible for a new classification leads to the desired result.

An important consideration in pre-training the convolution layers is the appropriate choice of a pre-training data set. One common approach is to use ImageNet or similar large, publicly available data sets. However, since we are using higher resolution electron microscopy images— 512×512 instead of 256×256 commonly used for ImageNet classification—and these images likely show different features than photographs, we use a different approach. Since we have a larger, automatically generated data set for predicting the operation mode (STEM or TEM), we use this data set to pre-train the convolutional layers in our architecture (Figures 4 and 5) and only retrain the dense layers for generalizing labels.

C. Computationally Efficient Implementation

In addition to reducing the risk of overfitting by reducing the degrees of freedom—we only retrain weights in dense layers, while convolution layers are fixed—this approach also lends itself to a computationally efficient implementation [6], which is critical in HPC environments. First, we train the CNN from Figures 4 and 5 on the automatically generated TEM vs. STEM data set. Subsequently, we remove the dense layers and consider the values immediately after the “Flatten” layer as output of this CNN and use them as input of a new neural network consisting only of one or more dense layers. To do so, we split the network into the convolutional part and the densely connected part, see Figure 6.

Splitting the network to achieve the effect of “freezing” the convolution layers we only need to transform all images once

Fig. 4. Transfer learning reuses the convolutional layers while replacing the dense layers. A computational efficient way to accomplish this is to consider the values after the flatten as output and use them as input for a new densely connected neural network.

Fig. 5. Transfer learning with CNN from Figure 4.

Fig. 6. An efficient implementation of transfer learning [6] splits the pipeline from Figure 5 into two parts. Using the first part of this pipeline, one precomputes bottleneck vectors for all images. These bottleneck vectors then serve as input for a neural network consisting only of one or a few densely connected layers. The confusion matrix currently shows the results for a run with a final validation accuracy of 70%, which is also the CNN model used for generating the first set of labels for ScienceSearch.

with the convolution layers to pre-compute the “bottleneck” values (see Figure 6, top panel), saving the output to a file. Subsequently, we input the bottleneck values directly into a dense neural network, using them in lieu of the original input images, and eliminate the need to duplicating this expensive operation during CNN network evaluation for training for a single label, as well as repeating it for multiple labels. As a consequence, it is possible to train only small neural network for each label, making training per label very quick and making it possible to quickly adapt to labeling results.

For each user provided label we train a simple neural network consisting of—in most cases—only one densely connected layer using these “bottleneck values” instead of images as input (see Figure 6, bottom panel). Using “bottleneck” values instead of images is computationally equivalent to fixing weights in the convolutional layers and just training the dense layer(s), but reduces computational costs. Instead of computing the results of the convolutional layers multiple times during training and for different labels, we compute them only once and re-use them multiple times, see Figure 6 (bottom).

D. Estimating probabilities

For search, in particular ranking search results, we need to estimate how “certain” the classifier is that a label applies to an image. We use the softmax function on the output of the CNN to determine the probability that a given label applies to an image. The softmax function normalizes output values such that each value is in the range zero to one and that all entries add up to one. The softmax function is generally used to translate the output of a neural network into probabilities. Applied to the output of the binary classifier, using the vector value for an output of “true” this results in a probability estimate that a label applies to the given image. In the image tagging tool, we currently suggest labels if its probability estimate is higher than 55%. In future work, as we create more data, we will conduct further experiments to determine if this threshold is appropriate.

IV. RESULTS

In our evaluations, we conduct a preliminary parameter study of different CNN architectures to identify an appropriate configuration of convolution kernel and feature vector size as well as number of dense layers and batch size (Section IV-A). We perform these experiments on Cori at National Energy Research Scientific Computing Center (NERSC). Cori, a Cray XC40, has a peak performance of about 30 petaflops. Cori is comprised of 2,388 Intel Xeon “Haswell” processor nodes, 9,688 Intel Xeon Phi “Knight’s Landing” nodes, and a 1.8 PB Cray Data Warp Burst Buffer.

We implement the CNNs in Keras and, during the parameter study, run each CNN on a single node. To determine the parameter configuration on each compute node, we run the study as an array batch job. Each node has access to its task id (`SLURM_ARRAY_TASK_ID`), which we use to select the appropriate CNN configuration for a compute node and run.

We evaluate the “winning” CNN architecture more thoroughly (Section IV-B) and conduct tests for using machine learning to transfer labels (Section IV-C) and run further tests

of this architecture on a NVIDIA GPUs. Our test machine has twelve Intel(R) Xeon(R) CPU E5-2603 v4 running at 1.70GHz each with 64 GB main memory and five¹ NVIDIA Tesla K80 GPUs with 12 GB memory per GPU. We implement the selected CNN architecture in TensorFlow using the Estimator API, to make efficient use of the resources. We also reduce the batch size to 25 samples to avoid running out of memory during training.

A. Identifying Appropriate Network Parameters

Our first experiment’s goal is to identify an appropriate CNN architecture, using the comparatively simple CNN architecture shown in Figure 7 as basis and varying the following parameters:

- convolution layer kernel size,
- convolution layer feature vector length,
- pooling layer kernel size,
- number of dense layers as well as number of neurons in each dense layer, and
- batch size.

Since electron microscopy images differ significantly from photographic images (and optical microscopy images), our primary goal in this parameter study is to identify appropriate configuration of the convolution layers of the network. In particular, we focus on determining an appropriate kernel size and number of filters/feature maps.

First, we run the same parameter sweep twice, testing all possible combinations of the following parameter choices:

Convolutional layers (three layers, eight configurations):

- Kernel sizes of 32, 16 and 8 with 64 features each.
- Kernel sizes of 64, 32 and 16 with 64 features each.
- Kernel sizes of 32, 16 and 8 with 128 features each.
- Kernel sizes of 64, 32 and 16 with 128 features each.
- Kernel sizes of 32, 16 and 8 with 256 features each.
- Kernel sizes of 64, 32 and 16 with 256 features each.
- Kernel size of 32 with 256 features, Kernel size of 16 with 128 features and kernel size of 8 with 64 features.
- Kernel size of 64 with 256 features, Kernel size of 32 with 128 features and kernel size of 16 with 64 features.

Pooling layer (two configurations):

- Kernel sizes of 2 or 4.

Dense layers (five configurations):

- 1 layer with 512 neurons
- 1 layer with 1024 neurons
- 1 layer with 2048 neurons
- 2 layers: 1024 and 512 neurons
- 2 layers: 2048 and 1024 neurons

Batch size (two configurations):

- 32 or 64 samples per batch.

Testing all possible combinations results in 160 ($8 \times 2 \times 5 \times 2$) different CNN configurations. For regularization, we use a

single dropout layer with a dropout probability of 0.1%. For the first sweep, we use a maximum wall clock time of 8 hours per run. Overall, 69 out of 160 runs finished; the remaining runs either did not finish all epochs or failed due to running out of memory or other errors (e.g., internal TensorFlow errors). For the second parameter sweep, we increase the maximum wall clock time of 12 hours per run, resulting in 94 runs finishing. We note that at the time when we were running this parameter study, the Keras/TensorFlow installation at NERSC was not fully optimized and did not use all cores on a node efficiently. As a result, training times for neural networks are very long with a single training epoch often taking about one hour, but in some configurations taking up to three hours. Thus, we restrict our tests to five epochs per configuration.

Tables I and II summarize the result of this parameter sweep, running five training epochs for each configuration. To restrict the number of CNNs to be compared, we only show CNN architectures that achieve a classification accuracy of at least 75% on the validation data set during any epoch. The tables show both the final validation accuracy at the end of the final epoch as well as the maximum accuracy achieved during any epoch. Final validation accuracy for the same CNN configuration fluctuates significantly between the two runs. For example, configuration 16 achieves a final validation accuracy of 51.3% in the first run and of 72.1% during the second run. This is likely due to the small number of training epochs where the optimization process has not converged. However, considering the maximum validation accuracy achieved during any training epoch is more stable—configurations 16, 18, 34 and 70 only vary by approximately 3%. While these runs are insufficient to decide on the best CNN configurations, they make it possible to fine-tune the CNN parameter search space. For example, in most cases a batch size of 32 achieves comparable or better results than a batch size of 64, and we use a fixed batch size of 32 for the second run to focus our search on other parameters, most notably more configurations for the convolution layers. Furthermore, most configurations with a feature vector size of 256 for all convolution layers result in out of memory errors. In the second run, we switch to a decreasing feature vector size for the subsequent convolution layers when the initial feature vector size is 256.

Next, we run a second sweep of configurations with the goal of identifying simpler CNN architectures that still achieves a sufficient classification accuracy:

Convolutional layers (three layers, nine configurations):

- Kernel sizes of 16, 8 and 4 with 64 features each.
- Kernel sizes of 32, 16 and 8 with 64 features each.
- Kernel sizes of 64, 32 and 16 with 64 features each.
- Kernel sizes of 16, 8 and 4 with 128 features each.
- Kernel sizes of 32, 16 and 8 with 128 features each.
- Kernel sizes of 64, 32 and 16 with 128 features each.
- Kernel size of 16 with 256 features, Kernel size of 8 with 128 features and kernel size of 4 with 64 features.
- Kernel size of 32 with 256 features, Kernel size of 16 with 128 features and kernel size of 8 with 64 features.

¹The machine has six GPUs, but we restrict our use to five GPUs by setting `CUDA_VISIBLE_DEVICES`. The machine is a shared resource and we cannot block all GPUs.

Fig. 7. Convolutional neural network (CNN) architecture for classifying microscopy images into TEM vs STEM, which is based on AlexNet [7]. The input is a 512×512 (gray scale) image. The CNN consists of three convolution layers followed by a max pooling layer and one or two densely connected layer(s). The output layer consists of two neurons corresponding to the two classifications. In a parameter study we experiment with different kernel and feature vector sizes for the convolution layers and different configuration of dense layers.

ID	Convolution Layers (#features, kernel size)	Pooling Size	Dense Layers	Batch Size	Status	Max Time per Epoch	Validation Accuracy Best	Final
16	[(64, 32), (64, 16), (64, 8)]	2	[1024]	32	Finished	2121.0s	0.754	0.513
18	[(128, 32), (128, 16), (128, 8)]	2	[1024]	32	Finished	3648.0s	0.758	0.758
34	[(128, 32), (128, 16), (128, 8)]	2	[1024, 512]	32	Finished	3610.0s	0.758	0.758
50	[(128, 32), (128, 16), (128, 8)]	2	[2048]	32	Finished	3637.0s	0.756	0.527
70	[(256, 32), (128, 16), (64, 8)]	2	[2048, 1024]	32	Timeout (Epoch 5)	6285.0s	0.751	0.751
144	[(64, 32), (64, 16), (64, 8)]	2	[2048, 1024]	64	Finished	2115.0s	0.754	0.623
146	[(128, 32), (128, 16), (128, 8)]	2	[2048, 1024]	64	Finished	3568.0s	0.765	0.765

TABLE I

CNN CONFIGURATIONS THAT ACHIEVE A VALIDATION ACCURACY OF AT LEAST 75% DURING ANY EPOCH FOR THE FIRST RUN OF FIRST PARAMETER STUDY.

ID	Convolution Layers (#features, kernel size)	Pooling Size	Dense Layers	Batch Size	Status	Max Time per Epoch	Validation Accuracy Best	Final
16	[(64, 32), (64, 16), (64, 8)]	2	[1024]	32	Finished	2110.0s	0.786	0.721
18	[(128, 32), (128, 16), (128, 8)]	2	[1024]	32	Finished	3603.0s	0.784	0.784
20	[(256, 32), (256, 16), (256, 8)]	2	[1024]	32	Finished	7772.0s	0.753	0.749
34	[(128, 32), (128, 16), (128, 8)]	2	[1024, 512]	32	Finished	3589.0s	0.752	0.546
66	[(128, 32), (128, 16), (128, 8)]	2	[2048, 1024]	32	Finished	3629.0s	0.764	0.764
68	[(256, 32), (256, 16), (256, 8)]	2	[2048, 1024]	32	Finished	7830.0s	0.752	0.752
70	[(256, 32), (128, 16), (64, 8)]	2	[2048, 1024]	32	Finished	6279.0s	0.768	0.768
80	[(64, 32), (64, 16), (64, 8)]	2	[512]	64	Finished	2107.0s	0.759	0.616
115	[(128, 64), (128, 32), (128, 16)]	2	[1024, 512]	64	Timeout (Epoch 4)	10662.0s	0.768	0.512

TABLE II

CNN CONFIGURATIONS THAT ACHIEVE A VALIDATION ACCURACY OF AT LEAST 75% DURING ANY EPOCH FOR THE SECOND RUN OF THE SECOND PARAMETER STUDY.

- Kernel size of 64 with 256 features, Kernel size of 32 with 128 features and kernel size of 16 with 64 features.

Pooling layer (four configurations):

- Kernel sizes of 2, 4, 8 or 16.

Dense layers (5 configurations)

- 1 layer with 256 neurons
- 1 layer with 512 neurons
- 1 layer with 1024 neurons
- 1 layer with 2048 neurons
- 2 layers: 1024 and 512 neurons

Batch size (1 fixed configuration):

- Fixed at 32 per batch.

Testing all possible parameter combinations results in 180 ($9 \times 4 \times 5$) CNN configurations. For regularization, we again used a single dropout layer with a dropout probability of 0.1%. Table III shows the results of this parameter study on Cori at NERSC, where each run had a maximum wallclock time of 18 hours per CNN configurations. Overall, 65 runs out of 180 finished. 55 of the remaining configurations did not complete all training epochs (e.g., runs 22, 31, 119 and 166, which

timed out during earlier epochs) and 60 configurations ran out of memory.

While this parameter sweep does not provide us with the extensive overview and the definite results we are aiming for, it nonetheless provides us with valuable information about possible CNN architectures that could achieve a satisfactory classification accuracy on the training data set. Most notably, two configurations (#100 and #168) achieve a validation accuracy greater than 80%. The considerable discrepancy between final and best validation accuracy indicates that most CNNs have not converged after training for 5 epochs, yet, limits on computation time available to the project prevented us from training all CNN architectures over substantially longer time. As a consequence, we decided to identify a CNN that achieved a satisfactory validation accuracy and continue experiments on GPUs for a faster turn around of results. In particular, we want to determine, how many training epochs are necessary for the CNN to converge and how changing the dropout probability affects convergence. While CNN configuration #168 achieves the best validation accuracy with 82.6%, configuration #100 comes in as close second with a validation accuracy of

ID	Convolution Layers (#features, kernel size)	Pooling Size	Dense Layers	Batch Size	Status	Max Time per Epoch	Validation Accuracy	
							Best	Final
9	[(64, 16), (64, 8), (64, 4)]	4	[256]	32	Finished	1950.0s	0.762	0.688
10	[(64, 32), (64, 16), (64, 8)]	4	[256]	32	Finished	6107.0s	0.784	0.784
15	[(256, 16), (128, 8), (64, 4)]	4	[256]	32	Finished	6860.0s	0.794	0.794
19	[(64, 32), (64, 16), (64, 8)]	8	[256]	32	Finished	6071.0s	0.754	0.627
21	[(128, 16), (128, 8), (128, 4)]	8	[256]	32	Finished	3898.0s	0.768	0.535
22	[(128, 32), (128, 16), (128, 8)]	8	[256]	32	Timeout (Epoch 5)	13372.0s	0.755	0.622
31	[(128, 32), (128, 16), (128, 8)]	16	[256]	32	Timeout (Epoch 5)	13439.0s	0.753	0.54
45	[(64, 16), (64, 8), (64, 4)]	4	[512]	32	Finished	1980.0s	0.771	0.771
54	[(64, 16), (64, 8), (64, 4)]	8	[512]	32	Finished	1909.0s	0.756	0.551
57	[(128, 16), (128, 8), (128, 4)]	8	[512]	32	Finished	3862.0s	0.766	0.506
82	[(64, 32), (64, 16), (64, 8)]	4	[1024]	32	Finished	6255.0s	0.768	0.685
96	[(256, 16), (128, 8), (64, 4)]	8	[1024]	32	Finished	6735.0s	0.755	0.745
99	[(64, 16), (64, 8), (64, 4)]	16	[1024]	32	Finished	1924.0s	0.805	0.596
100	[(64, 32), (64, 16), (64, 8)]	16	[1024]	32	Finished	6073.0s	0.815	0.815
117	[(64, 16), (64, 8), (64, 4)]	4	[1024, 512]	32	Finished	2020.0s	0.784	0.784
119	[(64, 64), (64, 32), (64, 16)]	4	[1024, 512]	32	Timeout (Epoch 3)	19629.0s	0.759	0.759
123	[(256, 16), (128, 8), (64, 4)]	4	[1024, 512]	32	Finished	6886.0s	0.793	0.793
127	[(64, 32), (64, 16), (64, 8)]	8	[1024, 512]	32	Finished	6072.0s	0.769	0.55
135	[(64, 16), (64, 8), (64, 4)]	16	[1024, 512]	32	Finished	1944.0s	0.772	0.752
141	[(256, 16), (128, 8), (64, 4)]	16	[1024, 512]	32	Finished	6843.0s	0.767	0.767
166	[(128, 32), (128, 16), (128, 8)]	8	[2048]	32	Timeout (Epoch 5)	13440.0s	0.789	0.789
168	[(256, 16), (128, 8), (64, 4)]	8	[2048]	32	Finished	6888.0s	0.826	0.826

TABLE III

CNN CONFIGURATIONS THAT ACHIEVED A VALIDATION ACCURACY OF AT LEAST 75% DURING ANY EPOCH FOR THE SECOND PARAMETER STUDY.

81.5%. We continue further studies with configuration #100, since (i) the difference in accuracy is well below fluctuations between the same CNN architecture in multiple runs, and (ii) configuration #100 requires less memory and fit more easily on a GPU. This configuration, shown in Figure 8 uses three convolution layers with kernels sizes of 16, 8 and 4 and a feature vector size of 64. The convolutional layers are followed by a max pooling layer with a kernel size of 16 and a single dense layer consisting of 1024 neurons. We note that finding the optimal CNN configuration would require a higher maximum wallclock time for individual runs. Furthermore, it may require distributing the CNNs for configurations that ran out of memory over multiple nodes. Finally, an improved parameter study would require training over more epochs. We are planning to re-run the parameter study with recent optimizations in the TensorFlow/Keras environment at NERSC.

B. Detailed Examination of the Selected CNN

Based on the parameter study described in the previous section, we focus on the CNN architecture shown in Figure 8 since it achieves a satisfactory validation accuracy while having relatively low memory requirements. We run further tests of this architecture on NVIDIA GPUs, reimplementing the selected CNN architecture in TensorFlow using the Estimator API to make efficient use of these resources. However, we still have to reduce the batch size to 25 samples to avoid running out of memory during training.

For regularization (i.e., to avoid overfitting) we use a single dropout layer between the fully connected layer and the output layer. Figure 9 shows the convergence of this CNN for different dropout probabilities (0.0001, 0.001, 0.01, and 0.1). Overall, CNN training accuracy was most consistent for a dropout rate of 0.01, which we used for subsequent training.

True Label \ Predicted Label	Predicted Label	
	TEM	STEM
TEM	449	51
STEM	141	348

TABLE IV

CONFUSION MATRIX FOR TEM VS STEM CLASSIFICATION USING THE CNN ARCHITECTURE SHOWN IN FIGURE 8.

For the TEM vs STEM classification task, we achieved a classification accuracy of 79.4% on the separate validation dataset. Applying the resulting CNN to all images collected by the microscope achieves a similar accuracy. (Note: We use smaller training and validation datasets to have the same number of TEM and STEM images for training and validation.) Table IV shows the confusion matrix for STEM vs. TEM classification task. Rows correspond to the ground truth while columns to the predicted labels. The matrix shows that 449 and 348 images are correctly classified as TEM or STEM respectively. Only a very small number of STEM images (51) is incorrectly classified as TEM, while there are more TEM images (141) that are incorrectly classified as STEM. Finally, we run the CNN on a withheld test data set of 1000 images collected since December 2017, i.e., after we created training and test data set and identified the CNN architecture. On this withheld test data set, our CNN achieves a classification accuracy of 86.5%. While we were careful not to include images already used during training, it is possible that the higher classification accuracy of the withheld test set is due to it being more similar to the training set than the validation set. The higher accuracy may also be due to relatively high variance due to small training set sizes. We plan to further

Fig. 8. The convolutional neural network architecture used after parameter study for transferring labels: Three convolution layers with kernel sizes of 16, 8 and 4 and a feature vector length of 64 are followed by a pooling layer with kernel size 16. The output of the pooling layer is flattened and used as input for a single densely connected layer comprised of 1024 neurons. The output of the network consists of two neurons for true and false.

Fig. 9. Convergence for different dropout rates over multiple training epochs: The image shows classification accuracy on the validation data set (y-axis) as a function of number of training epochs (x-axis).

investigate why the accuracy on the withheld test set is higher than on the validation data set.

C. Transferring Learning Tests

Now that we have identified appropriate parameters for our CNN architecture and examined this architecture in great detail, we use it for transfer learning. To ensure that our general transfer learning implementation works and that the pre-computed bottleneck values are correct, we first re-train a neural network on the TEM vs. STEM classification problem, using the bottleneck values as input (Section IV-C1). Subsequently, we show initial results of using transfer learning for generalizing labels (Section IV-C2).

1) *Validation of the Bottleneck Model:* To test the accuracy of transfer learning, we first perform a simple “sanity” check. We use the bottleneck values of all images of the TEM vs. STEM data set and train a new neural network consisting of a dense layer with 1024 neurons and an output layer with two neurons on the TEM vs. STEM classification problem. As mentioned in Section III-B this is equivalent to “freezing” the weights in the convolution layers and retraining the densely connected layers. As expected, the accuracy achieved after

retraining is close to 80%. We train over 22 epochs and achieve a validation accuracy of 81.3% using dropout rate of 0.2.

2) *Transferring Labels:* We subsequently run experiments using this model to generalize labels. We use a data set of 351 distinct manually labeled images with each image having a subset of labels out of a total of 121 labels. We chose the labels for which we had the most images: “atomic resolution” (applies to 198 images), “low resolution” (applies to 99 images), and “nanoparticles” (applies to 95 images).

We use 55% of the images for training and the remaining 45% for validation. We use a convolutional network consisting of two hidden dense layers with 2048 and 1024 neurons, followed by an output layer. We train this network for each label until the validation accuracy is above 80%. For the “atomic resolution” label, we achieve a training accuracy of 88.1% and a validation accuracy of 80.4% after 72 training epochs. For the “low resolution” label, we achieve a training accuracy of 89.1% and a validation accuracy of 80.4% after 73 training epochs. For the “nanoparticles” label, we achieve a training accuracy of 92.7 and a validation accuracy of 89.9% after 101 training epochs. We run the resulting classifier on all images to predict labels and import them in the manual tagging tool to be confirmed or rejected by domain scientists.

Finally, we selected 100 random images—using CNN label predictions to conduct multiple draws until all labels were represented in the selections—and asked our NCEM co-author to manually assign the labels “atomic resolution,” “low resolution” and “nanoparticles” to these images. After discarding 12 images that are already present in our training or validation data set, we obtain an independent test data set of 88 labeled images. On this test data set, our CNNs achieves a classification accuracy of 69.3% for the labels “atomic resolution” and “low resolution,” as well as a classification accuracy of 68.2% for the label “nanoparticles.”

V. RELATED WORK

Image classification is a common application of machine learning and there exist a wide range of benchmark data sets for evaluating new classifiers. One of the simplest data sets is the MNIST data set [8], [9] of handwritten digits, which requires only a very simple CNN for classification and is consequently used in many machine learning tutorials. The

ImageNet data set [3], [4] is currently the major benchmark for image classification. It is the basis of the “Large Scale Visual Recognition Challenge (ILSVRC),” which has given rise to a multitude of convolutional network architectures for image classification, including AlexNet [7], ZF Net [10], VGG Net [11], GoogLeNet [12] and Microsoft ResNet [13]. We base our architecture loosely on AlexNet [7]. More recent CNN architectures winning this challenge achieve higher classification accuracy, but are too complex for our case due to the amount of training data and the required higher image resolution.

Some approaches use machine learning to create textual descriptions of image contents [14]. In sciences, such as biology, there are ongoing efforts exploring content based image search, e.g., for microscopy images [15], as well as combining images with other data sets, such as ontologies [16]. Araujo et al. [17] use convolutional neural networks for content based image retrieval (CBIR).

Various scientific domains have used machine learning in recent times, including biology [18], detecting supernovae in images [19]–[21] as well as other classification tasks in astronomy [21]. Other examples of feature mining in scientific data sets include beams in particle accelerators [22]–[24], burning regions in combustion simulations [25], [26], overdensities (corresponding to stars, galaxies or clusters of galaxies) in cosmological simulations [27], extreme weather events in climate simulations [28] and pore structures in porous media [29].

In science applications, for example for image classification on biomedical data [30], lack of sufficiently sized training data is often an issue. The Zooniverse project [31] employs citizen scientists to generate labeled data sets for applications of machine learning to science. However, with scientific data, detailed expert knowledge is often necessary for classification tasks, thus limiting the number of people who can reliably label data leading to very small data set sizes of ground truth data.

TensorFlow [32] is a machine learning/deep learning framework by Google that supports efficient implementation of neural networks on the CPU as well as GPUs. Keras [33] provides an abstract, high level layer on top of TensorFlow and other machine learning backends and makes it easier to prototype and test complex neural network architectures. Graphics Processing Units (GPUs) have also been used for scalable machine learning algorithm frameworks [34] and deep learning². However, there is also work on scalable machine learning for high performance computing/supercomputers. The PNNL Machine Learning Toolkit for Extreme Scale (MaTeX)³ provides implementations of machine learning and data mining algorithms that are suitable for workstations, cloud computing and supercomputers.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we present an approach that uses deep learning to transfer labels from manually labeled images to the entire

image corpus. This method achieved a classification accuracy of approximately 80% on an automatically created data set for detecting microscope operating mode. Experiments with three labels achieved an accuracy of 68–69% for predicting labels. While this accuracy is already promising and useful for accelerating the manual labeling process by suggesting labels to domain scientists, we already have extensive plans to improve on this work and improve prediction accuracy significantly.

The limited amount of manually labeled micrographs makes a thorough evaluation of label generalization accuracy difficult. In this paper, we report the accuracy on a small validation and a small withheld test data set to gain an estimate on the reliability of this process. We will collect expert feedback using the data curation and tagging tool and use the number of accepted vs. rejected labels as additional evaluation metric for the quality of our predictions.

While automatic labeling as TEM vs. STEM enabled us to have a reasonably sized training data set, 3000 images for training and validation and testing is still a small size compared to other domains. We will create a larger training and validation data set by considering images from other microscopes at the center, and explore which standard data augmentation techniques are suitable for micrographs, i.e., which techniques do not distort important image characteristics. Using this larger data set, we will run more extensive parameter studies (more training epochs, more runs per configuration, using multiple nodes for a single CNN) at NERSC, making use of recent improvements of the machine learning software stack at the center and using a more efficient random hyper-parameter search [35].

ACKNOWLEDGMENTS

This work is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR) and resources used at the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility, supported by the Office of Science of the U.S. Department of Energy, both under Contract No. DE-AC02-05CH11231. Work at the Molecular Foundry is supported by the Office of Science, Office of Basic Energy Sciences, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

We would like to thank Gonzalo Rodrigo-Alvarez (former postdoctoral scholar at LBNL) and Yisha Sun (former summer student at LBNL/NERSC) and Katie Antypas for useful discussions and technical advice. We thank NCEM’s staff for their help in providing data and insights that made this work possible.

REFERENCES

- [1] G. P. Rodrigo, M. Henderson, G. H. Weber, C. Ophus, K. Antypas, and L. Ramakrishnan, “ScienceSearch: Enabling search through automatic metadata generation,” in *Proceedings eScience 2018*, 2018, accepted for publication.
- [2] NCEM Capabilities & Tools. [Online]. Available: <http://foundry.lbl.gov/facilities/ncem/expertise.html>

²<https://developer.nvidia.com/deep-learning-software>

³<http://hpc.pnl.gov/matex/> and <https://github.com/matex-org/matex>

- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the Conference on Computer Vision and Pattern Recognition 2009 (CVPR09)*, 2009.
- [4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [5] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, pp. 1–647–1–655. [Online]. Available: <http://dl.acm.org/citation.cfm?id=3044805.3044879>
- [6] Google. (2018) How to retrain an image classifier for new categories. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.
- [8] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
- [9] Y. Lecun, C. Cortes, and C. J. Burges. (1999) The MNIST database of handwritten digits. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [10] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Springer International Publishing, 2014, pp. 818–833.
- [11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: <http://arxiv.org/abs/1409.1556>
- [12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015.
- [13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016.
- [14] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 3156–3164.
- [15] B. H. Cho, I. Cao-Berg, J. A. Bakal, and R. F. Murphy, "Omero searcher: content-based image search for microscope images," *Nature methods*, vol. 9, no. 7, pp. 633–634, 2012.
- [16] Y. A. Kourmpetis, A. D. van Dijk, and C. J. ter Braak, "Gene ontology consistent protein function prediction: the falcon algorithm applied to six eukaryotic genomes," *Algorithms for Molecular Biology*, vol. 8, no. 1, p. 1, 2013.
- [17] F. H. Araujo, R. R. Silva, F. N. Medeiros, D. D. Parkinson, A. Hexemer, C. M. Carneiro, and D. M. Ushizima, "Reverse image search for scientific data within and beyond the visible spectrum," *Expert Systems with Applications*, vol. 109, pp. 35–48, 2018.
- [18] P. Baldi and S. Brunak, *Bioinformatics: The Machine Learning Approach*, 2nd ed., ser. Computation and Machine Learning. Bradford, 2001.
- [19] S. Bailey, C. Aragon, R. Romano, R. C. Thomas, B. A. Weaver, and D. Wong, "How to find more supernovae with less work: Object classification techniques for difference imaging," *The Astrophysical Journal*, vol. 665, no. 2, p. 1246, 2007. [Online]. Available: <http://stacks.iop.org/0004-637X/665/i=2/a=1246>
- [20] S. Bailey, C. Aragon, R. Romano, R. Thomas, B. Weaver, and D. Wong, "Object classification at the nearby supernova factory," *Astronomische Nachrichten*, vol. 329, p. 292, Mar. 2008.
- [21] N. M. Ball and R. J. Brunner, "Data mining and machine learning in astronomy," *International Journal of Modern Physics D*, vol. 19, no. 07, pp. 1049–1106, 2010. [Online]. Available: <http://www.worldscientific.com/doi/abs/10.1142/S0218271810017160>
- [22] O. Rübel, Prabhat, K. Wu, H. Childs, J. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. weber, P. Messmer, H. Hagen, B. Hamann, and E. W. Bethel, "High Performance Multivariate Visual Data Exploration for Extremely Large Data," in *SuperComputing 2008 (SC08)*, Austin, Texas, USA, November 2008, IBNL-716E.
- [23] D. Ushizima, C. Geddes, E. Cormier-Michel, E. W. Bethel, J. Jacobsen, Prabhat, O. Rübel, G. Weber, B. Hamann, P. Messmer, and H. Hagen, "Automated detection and analysis of particle beams in laser-plasma accelerator simulations," in *Machine Learning*, Y. Zhang, Ed. In-Teh, Feb 2010, pp. 367–389, ISBN 978-953-307-033-9.
- [24] D. Ushizima, O. Rübel, Prabhat, G. Weber, E. W. Bethel, C. Aragon, C. Geddes, E. Cormier-Michel, B. Hamann, P. Messmer, and H. Hagen, "Automated Analysis for Detecting Beams in Laser Wakefield Simulations," in *2008 Seventh International Conference on Machine Learning and Applications, Proceedings of IEEE ICMLA'08*, Dec. 2008, pp. 382–387, IBNL-960E.
- [25] P.-T. Bremer, G. H. Weber, J. Tierny, V. Pascucci, M. S. Day, and J. B. Bell, "Interactive Exploration and Analysis of Large Scale Turbulent Combustion Using Topology-based Data Segmentation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 9, pp. 1307–1324, 2011, IBNL-5921E.
- [26] P.-T. Bremer, G. H. Weber, V. Pascucci, M. S. Day, and J. B. Bell, "Analyzing and tracking burning structures in lean premixed hydrogen flames," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 2, pp. 248–260, Mar./Apr. 2010, IBNL-2276E. doi:10.1109/TVCG.2009.69. In press.
- [27] A. Knebe, S. R. Knollmann, S. I. Muldrew, F. R. Pearce, M. A. Aragon-Calvo, Y. Ascasibar, P. S. Behroozi, D. Ceverino, S. Colombi, J. Diemand, K. Dolag, B. L. Falck, P. Fasel, J. Gardner, S. Gottlöber, C.-H. Hsu, F. Iannuzzi, A. Klypin, Z. Lukić, M. Maciejewski, C. McBride, M. C. Neyrinck, S. Planelles, D. Potter, V. Quilis, Y. Rasera, J. I. Read, P. M. Ricker, F. Roy, V. Springel, J. Stadel, G. Stinson, P. M. Sutter, V. Turchaninov, D. Tweed, G. Yepes, and M. Zemp, "Haloes gone mad: The halo-finder comparison project," *Monthly Notices of the Royal Astronomical Society*, vol. 415, no. 3, pp. 2293–2318, 2011. [Online]. Available: <http://mnras.oxfordjournals.org/content/415/3/2293.abstract>
- [28] Prabhat, O. Rübel, S. Byna, K. Wu, F. Li, M. Wehner, and E. W. Bethel, "TECA: A Parallel Toolkit for Extreme Climate Analysis," in *Third Workshop on Data Mining in Earth System Science (DMESS 2012) at the International Conference on Computational Science (ICCS 2012)*, Omaha, Nebraska, June 2012, IBNL-5352E.
- [29] D. M. Ushizima, D. Morozov, G. H. Weber, A. G. Bianchi, J. A. Sethian, and E. W. Bethel, "Augmented topological descriptors of pore networks for material science," *IEEE Transactions on Visualization and Computer Graphics (Proc. IEEE Vis 2012)*, vol. 18, no. 12, pp. 2041–2050, 2012, IBNL-5964E.
- [30] G. Litjens, T. Kooi, B. E. Bejnordi, A. A. A. Setio, F. Ciompi, M. Ghafoorian, J. A. van der Laak, B. van Ginneken, and C. I. Săânchez, "A survey on deep learning in medical image analysis," *Medical Image Analysis*, vol. 42, pp. 60–88, 2017. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1361841517301135>
- [31] R. Simpson, K. R. Page, and D. De Roure, "Zooniverse: Observing the world's largest citizen science platform," in *Proceedings of the 23rd International Conference on World Wide Web*, ser. WWW '14 Companion. New York, NY, USA: ACM, 2014, pp. 1049–1054. [Online]. Available: <http://doi.acm.org/10.1145/2567948.2579215>
- [32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [33] F. Chollet, "Keras," <https://github.com/fchollet/keras>, 2018.
- [34] N. Lopes and B. Ribeiro, "Gpuml: An efficient open-source gpu machine learning library," *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 355–362, 2011.
- [35] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, Feb. 2012.