Simplifying Machine Learning Usage in Supercomputing Environments

LEO GU, MARY THOMAS

Machine learning (ML) has the potential to solve complex problems, and supercomputers can provide the huge computing power needed for ML. Although ML usage in Expanse, the current supercomputer within SDSC, is growing, the complexity associated with accessing the supercomputer is still a barrier for broad implementation. Right now, running ML programs off of a base environment requires setup of miniconda, installation of multiple packages on the said miniconda, cloning a repo to securely launch your program, and editing the path configuration variable to have the launch program. In this paper, we will explore ways to simplify security protocol access and create a tutorial that utilizes a container to standardize the process to launch an ML program. We developed a classification machine learning program on Expanse as a proof of concept for the simplification of training models on Expanse using containers. This work has been validated and added to the Expanse training material repository.

Additional Key Words and Phrases: Machine Learning, HPC, Interactive Computing, Jupyter, Jupyter Notebooks, Proxy

ACM Reference Format:
Leo Gu, Mary Thomas. 2021. Simplifying Machine Learning Usage in Supercomputing Environments. 1, 1 (October 2021), 7 pages. https://doi.org/12345

1 INTRODUCTION

Machine learning has been widely used for tasks like image recognition and self driving cars. In addition to consumer application, machine learning can also help scientists to solve complex scientific problems like finding new drugs(specific example maybe later). Complex machine learning models require high computing power to reach convergence in manageable time.

Expanse is a supercomputer currently hosted by the San Diego Super Computer Center which runs large amounts of programs. Currently it has 13 Scalable compute units, each unit containing 4 compute nodes which use 4 Nvidia V100 GPUs. This comes out to a total of 6.76 Petaflops[1]. Compared to an average home computer, which can run around 20 giga flops, the super computer is around 300000 times more powerful.

Currently, there are a large number of experienced machine learning programmers and scientists who are able to run it on their home computer or home server. However, when starting on Expanse, they find that the process to set up and run their programs and models is different from their standard computer. Due to a large security concern if a typical user is not given administrator privileges, which prevent them from root access. This restricts their ability to use sudo to install packages for the environment. Therefore, scientists must rely on third party installation tools like conda and pip, which leads to conflicting package dependencies.

Another issue is that a large portion of the training material becomes deprecated after a few months. This issue stems from package updates, which cause some functions to become entirely deprecated and replaced by new functions.

Author's address: Leo Gu, Mary Thomas, starwars0411@gmail.com.

Manuscript submitted to ACM

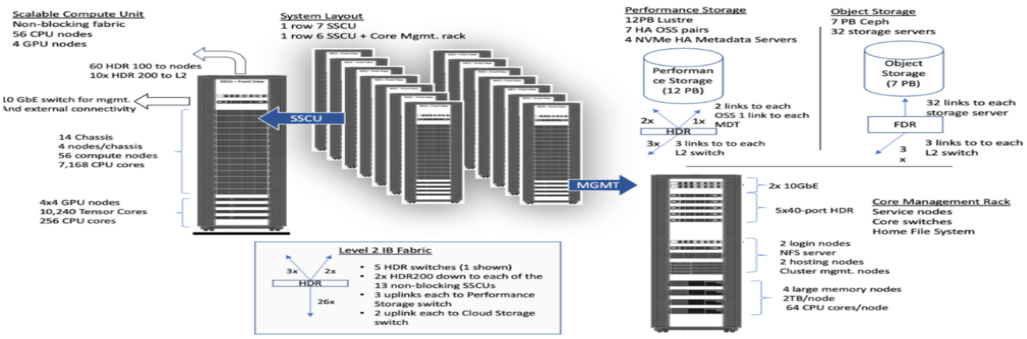


Fig. 1. Expanse System Architecture[2]

And there is no team that will update and test training material on a constant basis. This will result in a lack of example code, which users can refer to.

1.1 Galyleo and the Satellite Reverse Proxy Service

Finally, Jupyter Services like Jupyter Notebook, which a lot of the training material is written in, face a major security flaw. Currently users submit job to the slurm (job management) cluster through the HTTP protocol in Jupyter. Therefore, standard calls to a Jupyter service result in jobs being run locally and are vulnerable to outside interception and attack. To prevent users from doing this, Jupyter services now have to be launched using a system that connects to the Reverse Satellite Proxy Service, as shown in Fig. 3a. Using the standard proxy launch, users are unable to attach set environments either in the form of venv or a container, they must manually install every package in the training material to run it, which is time consuming and error prone.

In this paper, we created a proof of concept which incorporates the use of a container with a launcher of the Reverse Satellite Proxy Service called Galyleo, to show the ability and importance of generalizing training material to combat package and environment setup difficulties.

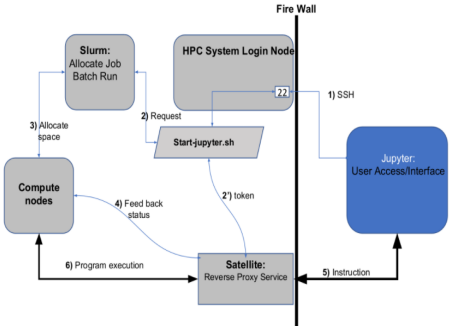
2 GENERALIZATION IN LAUNCHING AND SECURITY CONCERNS

2.1 Use of Container

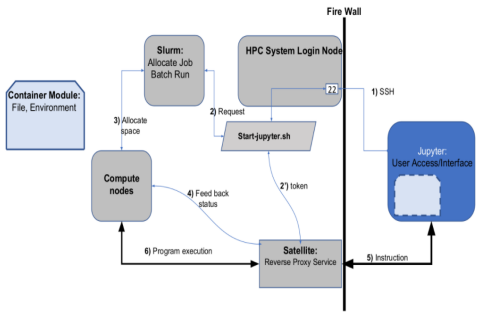
Containers are are software units that package up all the code and dependencies so that the software runs smoothly from one environment to another. They are important in the goal of this project because it is desirable to make the program run without environment issues and make it universally possible to run the program with a system that can connect to the Expanse supercomputer.

In this project, the singularity container is used to import the proper Tensorflow environment to run this program. Galyleo is an updated reverse proxy launch system that allows from the attachments of certain modules, like in this scenario singularitypro or the singularity container. With this, users can use two flags in launching the program: "–envmodules 'singularitypro'" "–sif '/cm/shared/apps/containers/singularity/tensorflow/tensorflow-latest.sif' ". This simplification of environment setup gives users the ability to use containers as a method of universal environments to set up their own machine learning models or other systems without having dependencies issues.

Fig. 2. Satellitle Reverse Proxy Services



(a) Satellite Reverse Proxy Service Standard



(b) Satellite Reverse Proxy Service With a Container

This simplification is important. Users need to be given a uniform non confusing way of launching training material. Currently, environments are very difficult to manage for new users and even advanced level users. There are package conflicts and even use of conda can result in packages downgrading themselves in trying to install other packages. Allowing for a unified environment system, plus a launch system that requires minimal environment setup, makes for easier and more unified access to training material.

Currently, we are launching our notebooks on the Satellite Reverse Proxy service. This service provides a secure way to launch Jupyter programs on the expanse supercomputer. A big issue is that Jupyter services connect by default using HTTP which is not secure. This means that is possible that unintended people can access and grab information from links [3]. Reverse proxy is a service which allows you to connect to a Jupyter service running on expanse with an HTTPS connection3a. When running the reverse proxy service, the system will generate a token or link that can then be put into the web browser with a secure HTTPS connection. Using galyleo, it is possible to run this reverse proxy service in cooperation with containers making a secured unified system run of any Jupyter service3b.

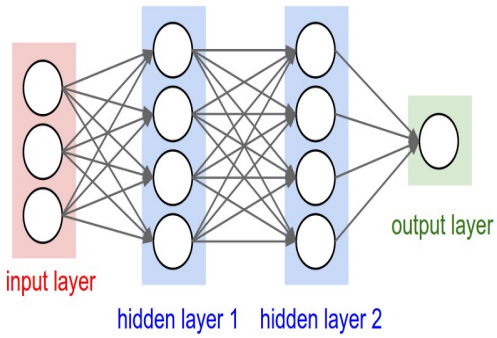
This process is currently combatting either the user manually loading the modules and attaching a container in their home environment using the previous reverse proxy launch service of start-Jupyter or just insecurely attaching a container to a Jupyter service. The first method is just messy and recquires individuals user research with the fact that none of the information to do this is written in any of the documentation. The latter method is a security concern not for only the user, but the Expanse supercomputer as well.

Although Galyleo has an easier access format on the expanse portal, which provides a simplified User Interface. However, Galyleo requires a few prerequisites and the training material currently does not include extensive information. Most developers seem to prefer using the command line during running their programs. Therefore, it is important to develop Galyleo to a more user friendly extent.

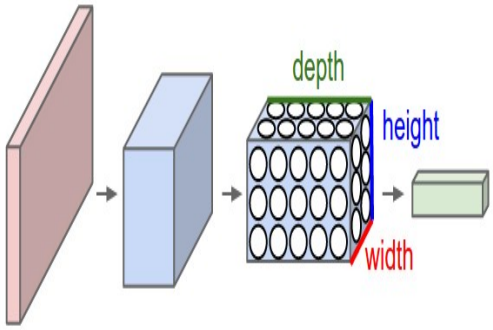
3 CREATING THE PROGRAM

3.1 Model Ideology

This model is created for existing machine learning users, so we want to exemplify the high computing power of the Expanse supercomputer while also showing the individual quirks and differences between the process on this environment compared to a home environment. This is so we first of all can show the benefits in the decrease in training



(a) Conventional 3 Layer Neural Network [4].



(b) Convolutional Neural network. Red Layer holds the image. So it would have a heigh and width of the dimensions of the image. There are 3 color channels in RGB, so the depth would be 3 [4]..

Fig. 4. Comparison Between a Convolutional Neural Network and Standard Neural Network

time and program run time, which is of a very significant degree due to hardware specifications. Also, we can show the differences in grabbing datasets and how to properly use the home directory or how to access scratch properly.

To solve the first issue, the simplest way is to have a high parameter count. High parameter counts can directly increase the amount of computing power necessary to perform high quality training. At the same time, the training material should not take an exuberant amount of time to run. Therefore, a compromise must be made. ModelNetv2 has a high parameter count in the millions but it is not too high in which a GPU node will take 10-15 seconds per layer given 100 steps per layer. If the same workload is run on the home computer, the training would take hours or days. This allows us to display the difference in computing power in a clear way without having an extremely long training time.

One more optimization is going to be the dataset. The dataset should be publicly available but also easy for users to learn how to properly install it into their home directory and use it. The dataset is publicly available on Kaggle and has a zip download. This gives the users the ability to upload it to their home directory and unzip the files via command line to access it from that point. Another user option is scratch, in which the user can ask for permission to upload it to the scratch file system and can unzip the files on their home computer, and manually upload the folder to the system.

3.2 Fundamentals of the Model

Convolutional Neural Networks are specialized neural networks that using the assumption that the input consists of images, constrain the architecture into a sensible manner with 3 dimensional neurons. Each neuron has width, height, and depth. The Height and Width are standardly the image dimensions or the dimension of the images after downsampling. The depth is either the amount of color channels or the filters that are used(Fig 4(b)).

There are three major layer types which are stacked together to form a CNN. Convolution Layers, Pooling Layers, and FC Layers. Convolution layers apply a filter by a computation between the filter and the input. The Network will earn filters that point out some visual features after the application. The amount of filters will create an activation map, and these maps are stacked are stacked together to produce an output. Pooling Layers downsize the volume along the width and height dimensions causing your neuron's input dimensions to slowly progress towards [1x1xn]. FC Layers

computes the entirety of the class scores, connected to every single volume existing in the network, compares the probabilities, and shrinks the volume down to [1x1xn] for a final output [4].

3.3 Tools

MobileNetv2 is a 53 layer convolutional neural network that is specialized in object detection [5]. It has a lower amount of parameters compared to MobileNetV1 meaning that it is faster to run. We will be using this in combination with Tensorflow, Keras, and sklearn to create our model.

All of these packages are built into a singularity container that is accessible by any expanse user. When the container is launched in galyleo, any user should have all the package dependencies necessary to run it.

3.4 Data

Our data set consists of 3100 images of different types of balls. We can split this into three subsets of a training dataset with 2860 images, a validation data set with 120 images, and a testing dataset with 120 images. This turns out to be a 92.25: 3.87 : 3.87 ratio. These 3100 images are split into 24 different classes. There is a large variety of balls like paintballs, water polo balls, and brass balls. There is not an equal amount of images per set. For example, there are 32 paintballs but 143 billiard balls. For this data set to run, all the images have to be resized to the same target size of (224,224)

3.5 Model

MobileNetv2 is general purpose computer vision neural networks designed for mobile devices, it only has only about 75 percent accuracy[[5]]. MobileNetV2 is our base model, but accuracy from the base alone is not sufficient. So we add a Global Average Pooling layer first, which averages each feature map in the classification and feeds it to the softmax layer after. Because there is no parameter that needs to be optimized, this method will help reduce overfitting over just a standard FC Layer. Following this step, the rest of the model follows a dense loop until we reach the softmax layer. For this model, there are around 3 million parameters , and 740,000 of them are trainable.

3.6 Optimization

To evaluate the model performance for each layer, we compare the predicted output of the model to the desired output. Cross-entropy loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual data. The goal of training is continuously decreasing loss function until it reaches the upper limit (converge). The gradient of loss function decrease will determine the training time. To train 740000 parameters, it may take a very long time to converge the data if a wrong method is used. We have studied several optimization methods:

Adagrad is an algorithm for gradient-based optimization that adapts the learning rate to the parameters, performing greater updates for infrequent and smaller updates for frequent parameters.

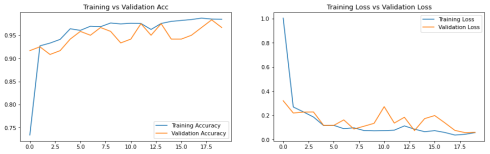
RMSProp is an adaptive learning rate method, which takes the average running rate of the previous average and the current gradient. RMSProp divides the learning rate by an exponentially decaying average of squared gradients.

Adaptive Moment Estimation(Adam) computes adaptive for each parameter by keeping an exponentially decaying average of past gradients[6].

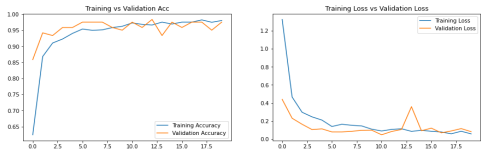
We have compared the above 3 optimization methods. Balance training cost vs iterations, Adam is the best and we selected it for our model. We have used a standard categorical cross entropy for loss function, and run 20 epochs with 100 steps per epoch.

4 RESULTS

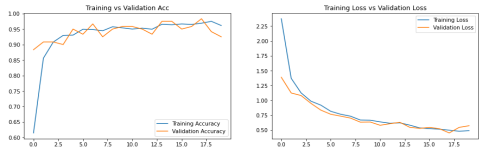
4.1 Graphs



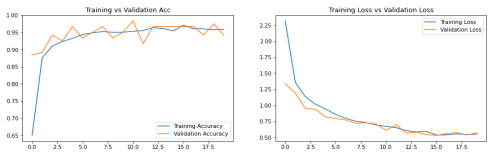
(a) Model with Dropout Layers



(b) Model with K Fold Cross Validation



(c) Model with Kernel Regularization



(d) Model with Kernel Regularization + Dropout Layers

Fig. 5. Results with Different Strategies

4.2 Discussion

After we finish training the model, we will perform an accuracy test on the validation and test images. Ideally, we would expect the same accuracy on the test image as the training image. However, two common issues were usually reported on training modeling: underfitting or overfitting. When the accuracy on the training is too low, the model can not perform image recognition on the test data, namely the model is underfitting training data. This usually occurs due to that model not powerful enough with too few parameters or training time is not sufficient or the model is not converging. The reverse is that a model fits too perfectly with training data, but the model performs poorly on the test data for image recognition [4]. In this case, the model is overfitting.

For our first model, its accuracy to training data is about 95 percent, however it performed badly on the test image dataset. Overfitting is due to the fact that the training sample count is small and does not represent all test samples. With large training parameters, about 740000 in our model, the model can fit to training data too well with sufficient machine time, but can not be generalized. To prevent overfitting, the best way is to increase the training image numbers to represent all possible general user cases. To achieve this, it will require a very large amount of training images, which requires high effort on labeling and long computing time. This is not practically possible. The other way to prevent overfitting is to simplify the model for the training data, namely reduce the number of learnable parameters. However, trimming 740000 parameters manually is also a very difficult task to perform. We reduced epochs amount or steps per epoch, which reduced the computing time but the final model accuracy dropped significantly.

The first method to systematically reduce the model is to randomly drop out the output features of the layer during training(dropout layers). The drop out rate is a fraction of features, which can be set. During validation or test, the output will not be dropped out. In this way, the active output during the test will be more than training. Fig 5a shows model accuracy and loss function with dropout layers, we observed that the overfitting does decrease. When increasing the drop rate from 0.2 to 0.5, the overfitting reduces further. However, we also observed that model accuracy decreases significantly below 0.93. Therefore, this method is not the most efficient and needs to be used in combination with other strategies.

One reason for overfitting is that there are large sample distribution differences between the training data and the test data. K-Fold Cross Validation split the data into K segments. During training, K-1 segments are used as the training data and 1 segment as the test data. There are K combinations of training and test data. The process will repeat K times, each segment is used in a different turn for test data so that all data have been shuffled through and cross validated. We set K=10, and perform training 10 times. However, the overfit did not seem to get better in our case, as shown in Fig **??**. The loss function of validation is below loss of training at the beginning, but it gets higher than training loss as epochs increases.

Finally, The other way to simplify the model is to force the weights to small values, which makes the weight distribution less variation, more regular. This is called Weight Regularization or Kernel Regularization. This is achieved by adding a cost to the loss function associated with large weight. There are two common approaches: L1 the cost adder is proportional to the absolute value of weight; L2 the cost adder is proportional to the square of weight. We chose L2 kernel regularization because it applies a higher penalty to the large weight. Fig 5c shows the use of kernel regularization which turns out with still some overfit but less of a harm on accuracy.

The model that was used finally is shown in Fig 5d. We used a combination of Kernel Regularization and Dropout layers to come up with a 95 percent accuracy on a withheld test data set.

5 CONCLUSION

By implementing singularity containers in Expanse, and providing users the instructions on the launch, we have successfully demonstrated that creating a model using a unified system on high performance computers is possible for user training, which simplified user security access. In addition, having training material fit in containers is also preferred by users, because the containers would contain run codes and package versions so that training material would never be truly deprecated.

We also built a tutorial image classification machine learning program on Expanse. The program includes the flow of preparing data, building models, optimizing execution, and reducing overfitting. We explored ways to simplify security protocol access and create a tutorial that utilizes a container to standardize the process to launch an ML program. This work has been validated and added to the Expanse training material repository.

ACKNOWLEDGMENTS

The author would like to thank Dr. Mary Thomas for mentoring me in the Research Experience for High School Students and CIML institute for teaching me about the in depth behind running programs on expanse.

REFERENCES

[1] SDSC. Expanse Home Page. https://www.sdsc.edu/services/hpc/expanse/, 2021. Accessed: 2021-08-06.
[2] Mary Thomas. Expanse 101. https://hpc-training.sdsc.edu/expanse-101/Expanse_Aggregate.html, 2021. Accessed: 2021-08-015.
[3] Scott Sakai and Mary Thomas. Satellite Reverse Proxy Service. https://github.com/sdsc-hpc-training-org/satellite, 2021. Accessed: 2021-08-14.
[4] Keras. Stanford CS231n Convolutional Neural Network Explanation. https://keras.io/api/applications/mobilenet/, 2021. Accessed: 2021-08-06.
[5] Stanford. MobileNetv2. https://cs231n.github.io/convolutional-networks/, 2021. Accessed: 2021-08-06.