COSMIC[1,2] Semantic Segmentation Framework

Asher Trockman

B.S. Computer Science

University of Evansville

May 2, 2019

Sponsor: **Dr. Lukas Mandrake**

Jet Propulsion Laboratory

California Institute of Technology

Advisor: **Dr. Don Roberts**

University of Evansville

ABSTRACT

Deep space missions such as the Mars Reconnaissance Orbiter collect more data than can be sent back to Earth due to limited communications bandwidth. Machine learning algorithms can be deployed on board orbiters to prioritize the downlink of scientifically interesting images, such as those including fresh impact craters, recurring slope lineae, or dust devils. However, basic machine learning research is necessary to boost real-world performance, and numerous possible convolutional neural network architectures must be evaluated in terms of accuracy and compute requirements. A framework is designed to reduce redundant development, to standardize the algorithm testing process, and to allow developers to focus on the implementation details of novel machine learning algorithms. Three convolutional neural network implementations are included with the framework, pending use in future research.

[1]Content-based Onboard Summarization to Monitor Infrequent Change
[2]CL #18-465

List of Figures

1 System overview diagram including core classes and modules. . . . . .
2 Example experiment to test two loss functions for VGG-16. . . . . .

1 INTRODUCTION

Space agencies around the world, such as NASA, are considering new missions to Mars and beyond. The resulting space probes must operate reliably in uncertain environments. Due to communications constraints, it is impossible for humans to intervene when something goes wrong. In more common situations, it is impossible for humans to react in real-time to opportunities for scientific discovery, such as unpredictable and transient events. Moreover, science instruments can collect more data than can be downlinked or stored on board the spacecraft. Autonomous decision making done on board the spacecraft could make better use of limited communications resources by prioritizing the downlink of anomalous and scientifically interesting images [1].

However, this setting poses many challenges for modern machine learning techniques. Data sets of scientifically interesting and transient phenomena are inherently limited in size. The objects of interest are often distinguished by small details that are only revealed in high-resolution images. For example, there may be just three hundred labeled example images that are each 2000×2000 pixels. In contrast, modern deep learning techniques have been successful after seeing hundreds of thousands of much smaller images, e.g., 32×32 pixels [2].

Planned missions also do not include high performance computers, but a successful demonstration of the scientific discovery capabilities of machine learning algorithms could result in the deployment of modern laptop-grade CPUs and GPUs on board future Mars missions. Hence, the Machine Learning and Instrument Autonomy group at the Jet Propulsion Laboratory aims to develop machine learning algorithms that can find scientifically interesting features in high-resolution images of the surface of Mars [3].

This project provides a framework to help developers compare machine learning algorithms that can find scientifically interesting features in Mars imagery. It will make it easier to do basic machine learning research to improve algorithms so that they are suitable for real-life deployment. The framework will reduce redundant development and allow developers to focus on the implementation details of their specific machine learning algorithms. It will ensure that different algorithms are tested rigorously and fairly.

2 PROBLEM STATEMENT

Given the limited bandwidth of the Deep Space Network, Mars science missions generate more data than can be transmitted back to Earth. This means that a significant portion of observations from missions such as the Mars Reconnaissance Orbiter (MRO) are never

downlinked or analyzed. Additionally, it means that repeat observations are highly expensive, making observations of scientifically significant transient events such as fresh impacts and avalanches very rare [4].

The MRO has two cameras that operate in the visible spectrum: the High Resolution Imaging Science Experiment (HiRISE) and the low resolution Context Camera (CTX). Currently, scientists analyze images from the CTX to find potentially interesting features, and request that those locations be imaged with HiRISE at the next possible opportunity [5]. Scientists cannot immediately act based on the low-resolution CTX images because the orbiter has moved a significant distance by the time the signal has reached Earth. Otherwise, CTX and HiRISE images are downlinked only as bandwidth allows, depending on whether other missions are communicating via the Deep Space Network and whether or not the MRO is serving as a relay for ground-based missions.

Due to bandwidth restrictions and the relatively small amount of storage space on the MRO, it is extremely difficult to observe Martian transient events. Finding such events requires that two images be aligned and differenced. However, repeat observations are expensive, and it is impossible to store the images on board the orbiter. Scientists are interested in the dynamics of the surface of Mars and would like more data on transient events such as fresh impact craters, slope streaks, recurring slope lineae, and dust devils.

Fresh impact craters occur when asteroids or comet fragments collide with the surface of Mars, leaving a blast zone of disturbed dust and rocks [6]. Studying these craters helps to estimate the duration of surface processes on Mars, but only a few hundred large impacts have been found in homogeneous dusty regions [7]. Slope streaks occur when settled dust falls downhill to reveal the surface beneath it; scientists can learn more about the weather of Mars by monitoring these streaks [8]. Recurring slope lineae are visually similar to slope streaks, but can be seen growing during warm seasons and fading during cold seasons; the reason for their formation is not known with certainty, and more observations would be helpful [9]. Dust devils, much like those on Earth, are the result of surface-atmosphere interaction that creates thermally driven vortices, which disturb trails of bright dust on the surface [10]; observing them can also help scientists learn about the climate and weather of Mars.

Deploying machine learning algorithms on board orbiters such as the MRO can result in better use of expensive and limited bandwidth by automatically identifying scientifically interesting landmarks such as these for prioritized downlink. Additionally, the output of such algorithms can be used as image "summaries" for transient detection across repeat observations. The resulting increased data throughput will increase our understanding of Martian surface processes and planetary science [4].

The performance of this strategy depends on the availability of requisite hardware on board future orbiters, on the types of features scientists would like to detect, and on the precision and recall demanded. Hence, one aspect of this project will be to create a catalog of the trade-offs between accuracy and computing power for a variety of machine learning algorithms. Currently, several different algorithms are being tested, resulting in redundant development. The project would benefit from a small framework that facilitates the experimentation and comparison of machine learning algorithms.

3 BACKGROUND

The Machine Learning and Instrument Autonomy (MLIA) Group at JPL has previously been successful in deploying machine learning algorithms for data prioritization and novelty detection on board the Earth Observing 1 spacecraft. They used a random forest to detect clouds in images so that cloud-free images could be prioritized for downlink. They also used an unsupervised salience-based algorithm to detect and prioritize anomalies based on local image context [11]. The same cloud detection strategy has been deployed on board a CubeSat [12]. The group has effective techniques for classifying landmarks [13], surface texture detection [14], and anomaly detection [15] on the ground.

During a recent internship, a deep neural network was created based on previous work [16] that can detect fresh impact craters in HiRISE images with 96±4% precision and 43±3% recall (averages and standard deviations computed from five-fold cross validation). The network was applied to find small, undocumented fresh impact craters in high-resolution HiRISE images, demonstrating that it would be useful for analyzing image data. Curriculum learning [17] and data set augmentation (as in Krizhevsky et al. [2]) helped the model to converge despite the use of an unconventionally small data set. These are relatively simple but crucial concepts that must be implemented correctly for experimental validity.

4 REQUIREMENTS & SPECIFICATIONS

The proposed framework allows various machine learning models to be easily tested and compared in their ability to detect objects of interest (such as fresh impact craters and dust devil tracks) in high-resolution images of the surface of Mars. It will help to assess the tradeoff between predictive accuracy and computational requirements for these models, which is important in deciding what model would be most appropriate for deployment on board an orbiter. Further, it helps to ensure that the models are validated rigorously by reducing inconsistencies due to redundant development and allowing developers to focus specifically on the implementation details of their model. The framework assumes that the models do semantic segmentation, that is, assigning a label to each pixel of an image.

The framework provides a common interface that the models can conform to and trains and evaluates any model that adheres to that interface. In order to evaluate models, that is, to assess their performance on a data set, the framework defines metrics for semantic segmentation, such as precision, recall, and intersection-over-union. Since the data sets have multiple labels per image, the framework provides methods to merge multiple labels into one. In order to train the models, the framework defines methods for transforming images, or performing data set augmentation: this helps models to become invariant to brightness, scaling, translation, and rotation of the input images.

The training process follows an adjustable schedule that can depend on the user-defined difficulty of training examples: this allows the developer to help the model converge with curriculum learning [17]. When the model is done training, it generates a final report. The report itself contains metrics such as precision, recall, and intersection-over-union evaluated

across the test set, in addition to the model's requirements in terms of multiply-adds and memory usage per inference.

Thousands of labels for images of the Martian surface are being collected by MLIA on the website Zooniverse [3]. The framework is able to parse these labels from Zooniverse for the four existing workflows: fresh impacts, dust devils, araneiforms, and swiss cheese. This will help to analyze labeling progress and also reduces storage requirements.

4.1 Related Developments

Many machine learning frameworks already exist. One popular option is TensorFlow [18], which allows developers to define dynamic computational graphs of differentiable functions. TensorFlow does automatic differentiation of the resulting computational graph, and the resulting gradients can be used to update the parameters of the model to eventually converge at a global optimum. TensorFlow makes it easy to run these computations on GPUs, an otherwise arduous process.

TensorFlow provides a higher-level API called "Estimators" [19], which allows for easier training, evaluation, prediction, and deployment. However, this is ill-suited to the application domain of analyzing high-resolution Mars imagery. The API makes it difficult to make low-level changes to models and training strategies, which are required for research and development at MLIA.

The proposed framework is much higher-level than TensorFlow or TensorFlow Estimators, for example, providing metrics, data augmentation, and training techniques specifically adapted to the application domain. The user implements the model itself in a low-level framework such as TensorFlow (or TensorFlow Keras), conforming to the framework's interface. It also would be possible to implement the model in another machine learning framework such as Caffe or PyTorch. The project framework will then run the model implementation, providing all the other common functionality that is necessary.

4.2 System Requirements

Mars imagery is high-resolution, and the project requires a high degree of robustness to various input conditions, as evidenced by the recent global dust storm on Mars. Thus, data augmentation methods work "on the fly" without saving data to the disk. They would otherwise consume a prohibitive amount of storage space. Importantly, data set augmentation is faster than training the model itself, which prevents a bottleneck in the flow of data to the GPU in the training process. It does not consume an excessive amount of memory or CPU time, which is important since it may run on a shared system. The training process itself is seamlessly interruptible: the user is able to stop the process and reload its state from the disk later. The framework periodically generates reports on the model's current performance and saves them to a log file.

[3] `https://www.zooniverse.org/projects/wkiri/cosmic`

4.3 User Experience Requirements

Since the framework is intended to be useful to other machine learning researchers, it should be easy for others to understand and adopt. The framework is implemented in Python 3 because it is the preferred language of MLIA and the broader machine learning community. It includes minimal use of advanced constructs like multiple inheritance, decorators, and long list comprehensions, and it is documented so that others can modify it in the future. It is easy to extend the framework with new metrics or data augmentation techniques.

4.4 Research Requirements

Since the purpose of this work is to make it easier to study the performance of various machine learning models, the project should include three default implementations of popular convolutional neural network architectures such as VGG [20], ResNet [21], Inception [22], AlexNet [2], or UNet [23], which must be modified to do semantic segmentation [16] instead of whole-image classification. The framework currently provides VGG, ResNet, and UNet architectures. The evaluation of these architectures with the framework will provide important insights for MLIA.

5 DESIGN APPROACH

5.1 Growing a Framework

The following design is primarily the result of reading source code for many machine learning experiments, and from personal experience doing (messy and inefficient) experiments in Jupyter notebooks. For example, the proposed model interface is the result of seeing many model implementations, their commonalities, and operations typically performed on them. Classes typically replace one or more tuples that are "passed around" these experiments: for example, `LabeledImage` replaces the pair `image, label` and provides additional functionality. The framework is an attempt to reduce development effort while not getting in the way of research; it could be described as "white box" framework, because it is primarily used by subclassing the model interface, and developers benefit from understanding the internals of the components.

While building the framework, classes and modules were tested manually, often using a "dummy" model implementation. This helped to make tweaks to the initial design that were beneficial for the final product.

5.2 System Overview

The key user-facing part of the framework is the `Evaluator`, which takes a machine learning model conforming to a certain interface and the path of the data set with which to train

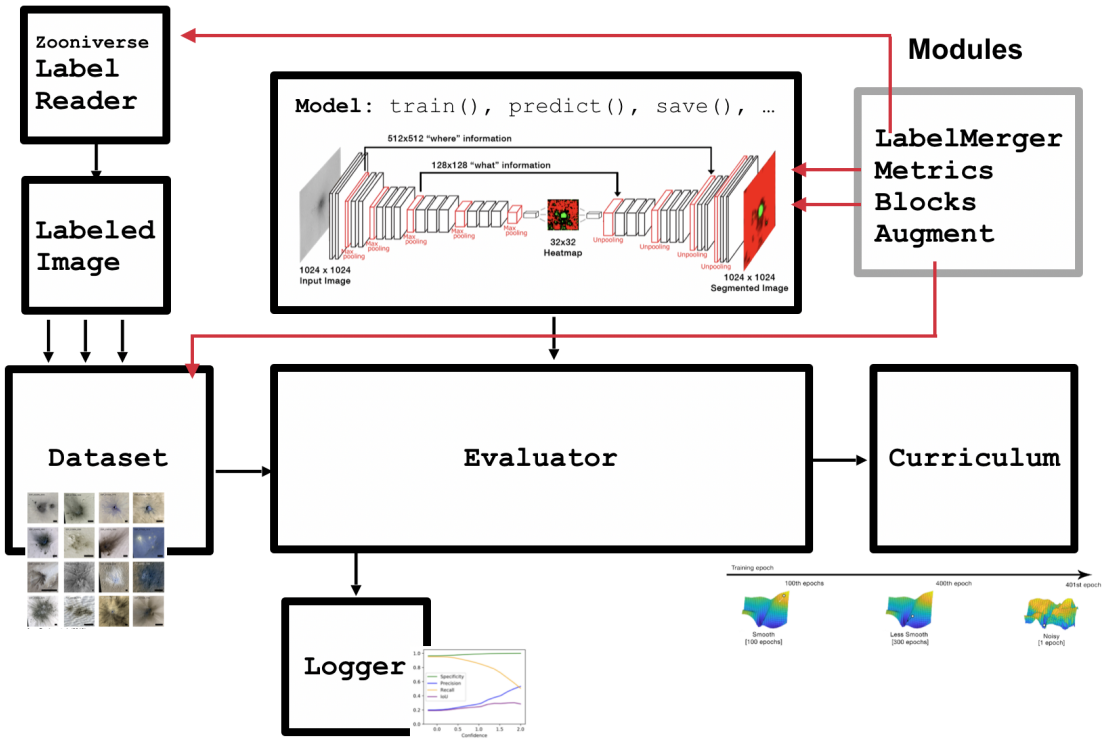

Figure 1: System overview diagram including core classes and modules.

and test the model. Thus, the `Evaluator` has a `Model` object and a `Dataset` object. It also accepts an optional list of configuration parameters.

The core of the framework is the `Curriculum` class, which is responsible for the iterative training process. Since cross validation is necessary, the `Evaluator` has many `Curriculum`s, each of which only see a subset of the larger data set. These `Curriculum`s may work in parallel on multiple GPUs. The `Evaluator` has a `Logger` for each cross validation fold (see Appendix A). Both the `Curriculum` and the `Evaluator` use the `Metrics` module for evaluating performance on the validation set or the test set respectively.

The `Dataset` serves as a large "virtual" data set that is generated on the fly to save storage space; it has many `LabeledImage`s. It may have just 300 original `LabeledImage`s, for example, but can extend this into tens of thousands of transformed images with data augmentation. The `Dataset` is also responsible for merging multiple labels into one unambiguous label and for resizing images to fit in the GPU memory if necessary. Thus, a `Dataset` uses the `LabelMerger` and `Augment` modules.

Models conforming to the required interface return predictions as `Inference` objects, which know how to print themselves in a user-friendly and informative way. The `Logger` aggregates statistics from `Inference` objects and writes them to disk throughout the training process.

5.3 Types

Although Python is a dynamically typed language, it is conceptually convenient to define some "types" that are used throughout the system. An `image` is a two-dimensional array of floats, or a grayscale image (since the existing MRO imagery is only one channel). A `label` is a two-dimensional array of integers with the same dimension as `image`; each discrete integer value corresponds to the label of the corresponding pixel in the `image`. The value 0 means that the pixel was not labelled. A `batch` is a list of images of arbitrary length. A `heatmap` is also a two-dimensional array of floats, but it is mapped to a topological color scheme when plotted; it represents the raw output of a semantic segmentation model.

5.4 Implementation

5.4.1 Model Interface

The core idea of the project is that a variety of machine learning models can be represented with a common interface. Any model conforming to this interface can then be evaluated according to a standardized procedure. Since Python does not provide interfaces by default, the framework uses abstract base classes. The interface includes the following methods:

`init`. This method initializes the computational graph for the machine learning framework of choice, such as TensorFlow. It initializes the TensorFlow-related class variables that are used in the following methods.

`heatmap(batch)`. Returns a list of `heatmap`s from the model, i.e., the model's "raw" output.

`inference(batch)`. Returns a list of `Inference`s after calling `heatmap`. Responsible for applying a threshold to `heatmap` that converts it into a discrete-valued label for comparison in `Metric`s.

`train(batch)`. Passes a batch of images through the model, computing gradients with respect to the loss function and propagating them back through the network.

`loss(batch)`. Defines the loss function for the network and sets up the TensorFlow optimizer (the dual of the computational graph defined in `init`).

`flops(w, h, c)`. Computes the number of FLOPs/multiply-adds required to make a prediction on an image of size $w \times h \times c$. The number of channels (for example, RGB) is three by default. In practice, this calls some internal TensorFlow functions to get a realistic number. It would be possible to count the number of multiply-adds based on the network architecture, but this would be assuming naive matrix multiplications and convolutions. This is used in reporting model comparisons in `Evaluator`.

`memory()`. Computes the number of trainable parameters in the network, which is proportional to the memory required to store the model.

`load(name)`. Loads a trained model from disk with the given name into the computational graph (`name` is optional).

`save(name)`. Saves a trained model to a file with the given name (`name` is optional).

The `save` and `load` methods are called by the `Curriculum` class's own save function.

In practice, this interface is implemented by `TFModel`, which provides TensorFlow-specific implementations of `memory`, `flops`, `save`, and `load`. Then, when using TensorFlow, work begins by subclassing `TFModel`. For example, `Vgg19` is a subclass of `TFModel`, which can be further extended for experimentation. Alternatively, `Dummy` is a model for testing purposes which implements `Model`.

5.4.2 Data Objects

Thousands of high-resolution images of scientifically interesting landmarks on the surface of Mars are being labelled on the website Zooniverse, which allows citizen scientists to make contributions to science by labelling data. Moreover, each image has up to 11 separate semantic (per-pixel) labels from users who may have substantial disagreements.

`LabeledImage`. The most important data object in the framework is the `LabeledImage`, which represents a high-resolution image and its corresponding labels. Each image has an ID that can be traced back to a larger source image from the MRO. Each image exists in its own directory on disk along with its labels.

First, the `LabeledImage` provides an interface to this directory. It stores the `image` and its `label`s in memory, but does not load them from disk until the raw data is required. Importantly, the `LabeledImage` provides functions to get the `image` and `label`, which seamlessly fixes two problems: The images are usually too large to fit in the GPU memory, so `get_image` resizes and caches the `image`. There are multiple labels per image, but there can only be one label for training and testing purposes; hence, `get_label` uses another module to compress all of the `label`s into a single `label`, which is then cached for later use. The resized images and merged labels are not written to the disk, as the specific sizes and merging techniques can change frequently.

The `LabeledImage` also reads a difficulty value from disk, which is chosen by the user at an earlier stage of the development pipeline. This difficulty value is used for the `Curriculum`, which will be defined below.

`Inference`. This class holds the output (or "inference") of a model. It includes a `LabeledImage`, a `heatmap` from the model, and a `label` obtained by applying a threshold to the `heatmap`. It also includes the inference time in milliseconds. This class is used in the `Logger` as well as for viewing model progress, e.g., in a Jupyter notebook. It is printable and saveable for qualitative analysis by the user.

5.4.3 Modules

In Python, a *module* is essentially a collection of functions under a common name. It is analogous to declaring static methods in a utility class. Modules provide standalone functionality. MLIA prefers modules to classes when only one instance of the class is necessary.

`Augment.` Deep learning algorithms typically require millions of training images for good performance. When transfer learning [24] from a pre-trained model (see Appendix A.3), only thousands of images are generally required [2]. However, in some cases, MLIA has less than a thousand images. This small data set can be extended by transforming the examples in various physically plausible ways. Fresh impacts, for example, come in various shapes, sizes and colors. Hence, the data set can be extended by changing the scale, rotation, brightness, position, and noise level of these images. This is called "data augmentation" [2]. This module provides a variety of image transformation functions, such as `rotate_and_shift(degrees, dx, dy)`, `zoom(scale factor)`, `brightness(slope, intercept)`, `noise(sigma)`, and `mixup(im1, im2)`. Each function takes a `LabeledImage` and returns a `LabeledImage`. Importantly, each function has a GPU and CPU implementation: there might not always be an extra GPU to use for data augmentation, in which case the framework can resort to parallel processing on CPUs. One limitation is that the GPU implementation relies on TensorFlow, which is not ideal if the framework is used with another machine learning library.

`Dataset.` Loads the data set from disk according to a provided path, e.g., creates a list of `LabeledImage`s. `Dataset` is also responsible for transforming the images en masse in the case of CPU parallel processing.

`LabelMerger.` This class solves the *multi-label* problem with two key strategies.

`rate(labs).` The first strategy involves merging labels based on inter-rater agreement, which results in a discrete-valued label that can be used to compute metrics such as precision, recall, and intersection-over-union. This strategy is parameterized by an agreement percentage κ, in which the label is chosen which has more than κ percent of raters in agreement. If there is no consensus, the pixel is left unlabelled.

`melt(labs).` The second strategy "melts" the labels into one high-information content label with "soft" (or real) values. The strategy is parameterized by a "temperature" T, which is similar to that defined in Hinton's paper on knowledge distillation [25]. Unlike Hinton's paper, the values in one pixel of the image can spill over into adjacent pixels, e.g., a kind of spatial knowledge distillation. In practice, this amounts to a Gaussian blur of the merged label with standard deviation T.

Each function in `LabelMerger` takes a list of `label`s and returns a single `label`.

`Metrics.` This class provides the metrics to be used by the `Evaluator`, such as precision, recall, accuracy, F1 score, and intersection-over-union. More metrics may be defined, such as those used in Google AI's paper [26] on semantic segmentation of tissue samples. Each metric takes an `Inference`; it uses the ground truth `label` from the `LabeledImage` and the inferred `label` to compute the metric. Each metric returns a score between 0 and 1 inclusive. Some metrics may not be defined for all inputs, in which case they return *NaN*.

`LabelReader.` This class is responsible for loading labels from the Zooniverse data dump, which is a CSV file. On Zooniverse, we currently have four workflows: fresh impacts, dust devils, araneiforms, and swiss cheese. The first two workflows have been completed, and the next two have tens of thousands of labels each. This class parses the CSV file, filters out

only specified recent workflow version numbers, and creates labels based on the labeling tool name and polygons provided. This makes it possible to eliminate an intermediate step of generating labels and storing them on disk. Alternatively, the module itself can be run to generate the labels on disk. `Dataset` resorts to generating the label in memory if it is not found with its corresponding image on disk.

`Blocks.` Provides reusable components of neural networks, making model implementations more readable. This includes convolution, pooling, and transposed/strided convolution layers, as well as the residual block for the implementation of ResNet.

5.4.4 Classes

`Curriculum.` Curriculum learning was introduced in 2009 by Bengio et al. [17] and remained largely unstudied until recently. At JPL, curriculum learning was independently rediscovered as an effective method to learn hard decision boundaries by building up "knowledge" from easy examples.

The proposed system needs to discriminate between fresh impact craters and other landmarks that look similar to fresh impact craters to the untrained eye. Moreover, there are far more of these difficult non-impact examples than actual impacts. The result is that a classifier typically is unable to learn what a fresh impact looks like without the assistance of a "curriculum".

The simplest example of a curriculum is to split the training process into three stages: easy, medium, and hard difficulty. The "easy" stage has images with clear, well-defined fresh impacts. The "medium" stage introduces some challenging transformations of these images, such as scaling and the introduction of random noise. The "hard" stage introduces images that look very similar to fresh impact craters but really are not, such as dust devils or araneiforms (dark fan-shaped deposits of dust caused by the sublimation of CO_2 below an icy surface).

This type of manually defined curriculum is supported by the presence of "difficulty" labels in each `LabeledImage` object. However, there are recently proposed techniques (ca. 2018) to automatically structure the curriculum [27], e.g., based on a second model (a "MentorNet" [28]) that ranks `LabeledImage`s according to perceived difficulty.

In this framework, the `Curriculum` class is responsible for training the model and ordering the training examples by using one of multiple strategies; it is the "engine" of the framework, deciding when to stop training based on evaluating `Metric`s on a validation set, when to write log files, and when to save the models to disk for back up. Hence, a `Curriculum` has a `Model` and a `Dataset`. The curriculum itself has `save` and `load` methods so that the training process can be stopped and started when necessary, e.g., due to resource limitations.

In summary, `Curriculum` is the inner loop of the training process.

`Evaluator.` This class is the "outer loop" of the training process, deciding how to split a `Dataset` into train/test/validation sets and performing cross validation. It runs one or many

`Curriculums` on a `Model` and logs the final performance metrics, i.e., graphs of performance and statistics on runtime/memory usage. It catches interruptions and coordinates calls to `Curriculum`'s `save` and `load` methods when necessary.

`Logger`. This class is responsible for creating graphs and recording statistics, e.g., by processing `Inference` objects. The `Evaluator` has a `Logger` that is shared with its `Curriculum`s. The `Logger` is not a module because it retains state about the current cross validation fold.

5.5 Third-Party Subsystems

The project uses several open source components: NumPy [29] is used extensively throughout the project for its convenient N-dimensional array objects and operations. OpenCV [30] is used to provide CPU-bound affine transformations of images, such as scaling, translation, and rotation. TensorFlow [18] provides GPU-bound transformations of images. Keras [18], which is included with the latest version of TensorFlow, provides a more user-friendly API for specifying machine learning models. Three convolutional neural network architectures are implemented in TensorFlow.

5.6 Model Implementations

Three fully-convolutional neural network architectures (for semantic segmentation) are provided with the framework to faciliate research: VGG-16 (and VGG-19), UNet, and ResNet. They can easily be subclassed, for example, to change loss functions. By looking at the implementation details of the parent class, it is possible to reduce the size of models by setting certain parts of the TensorFlow graph to the identity function. After upgrading to TensorFlow 2.0 Alpha for the Keras API, these model implementations were broken, reporting zero trainable parameters. After much trial and error, it was discovered that the layers of the network must be initialized separately from the function that calls the network on an input tensor. It was necessary to look at source code for TensorFlow to make up-to-date implementations of `flops` and `memory`.

6 RESULTS

The proposed design fulfills the requirements and specifications of the project. Machine learning models conforming to the interface proposed can be passed to the `Evaluator` in a single line of code with minimal configuration overhead. This ensures that the models can be tested easily and consistently as required. The `LabelMerger` module used by `Dataset` provides the functionality necessary to solve the problem of having multiple labels per image. The `Augment` module defines methods to extend the data set and promote invariance to various transformations. The `Metrics` module satisfies the requirement for standardized

metrics such as precision, recall, and intersection-over-union. The proposed interface guarantees that every model can report its compute and memory requirements; implementations for these functions have been provided for TensorFlow.

The idea of curriculum learning is central to the framework, and the `Curriculum` class guarantees that it is easy for developers to use curriculum learning to train their models. The required report can be generated by the `Logger` class.

The system is fast and efficient, and data augmentation does not bottleneck the training process: `LabeledImage`s are not loaded into memory and processed until they are required, and they are cached after processing for quick access. Bottlenecks are avoided through `Augment`'s provided CPU and GPU-based transformation functions. Transformations on the GPU are fast and do not significantly slow down the training process. In the case of CPU transformations, `Dataset` parallelizes the required transformations on multiple CPUs to keep pace with the GPU-based training process.

The proposed framework also is simple and easy to use for other developers to understand and adopt. While the framework internally uses "complex" programming constructs, namely multiple inheritance and decorators, developers are not required to use these when making new experiments with the framework; the cost in readability is offset by savings in reusability. See Figure 2 for an example experiment implemented with the framework; while the VGG-16 model itself is quite large, it is easy to read both examples to determine the intent of the experiment. The default configuration of `Evaluator` can be inspected to find details on training; moreover, the logged output shows performance on training, testing, and validation sets, as well as performance requirements in FLOP/s and memory usage or trainable parameters.

Modules are used in place of classes when possible, which is the preference of MLIA. Adding new metrics or data augmentation techniques is as easy as adding the function to `Augment` or `Metrics`.

Figure 2: Example experiment to test two loss functions for VGG-16.

```
class Vgg16_Dice(Vgg16):
    name = 'VGG16/dice'
    def loss(self, x, y):
        yhat = self.heatmap(x)
        return dice_loss(y, yhat)

Evaluator(Vgg16_Dice(),
    Dataset('./fresh_impacts'))
```

(a) Dice loss

```
class Vgg16_MSE(Vgg16):
    name = 'VGG16/mse'
    def loss(self, x, y):
        yhat = self.heatmap(x)
        return mse(y, yhat)

Evaluator(Vgg16_MSE(),
    Dataset('./fresh_impacts'))
```

(b) Mean-squared error loss

6.1 Experimental Results

At the time of writing, the requirement to evaluate three machine learning models in the application domain has not been fulfilled. However, one experiment has been *replicated* using the framework. Previously, a VGG-19 fully-convolutional neural network was used to achieve acceptable precision and recall on the fresh impacts task; this involved curriculum learning and data augmentation. The framework, which provides the same model and functionality, was used to replicate this result successfully. This, at least, is evidence that the framework fulfills its basic goals.

7 CONCLUSION

The framework designed makes it easier to do basic machine learning research, reducing redundant development and allowing researchers to focus on the implementation details of their models. Hopefully, this framework will accelerate the pace of the machine learning research that is necessary for COSMIC. While the module used for label parsing has been sent to JPL to help with label analysis, the framework as a whole has not been used there. It remains to be seen if it will be adopted.

8 FUTURE WORK

The framework facilitates "traditional" machine learning experiments: essentially, training on a training set and testing on a held-out test set. However, this is not a realistic setting to evaluate model performance; it serves as a *lower bound* on performance. In reality, it should be possible to incorporate *domain knowledge* into the prediction pipeline. For example: fresh impacts are quite rare, and it is unlikely to see many of them in succession over a large area. If the model finds hundreds of fresh impacts in a few hundred square kilometers, this prediction is likely anomalous; for example, it could be a field of araneiforms with deposits instead of fresh impacts. Similarly, swiss cheese generally only forms in polar regions, so the "prior" on this landmark should be a function of the orbiter's position. Incorporating this knowledge into the testing process is possible, but it remains a long-term goal for the project. More realistic testing scenarios might push forward research and development, as has been the case for machine learning historically.

One requirement was to test three models with the framework. Only one model, VGG-19, has been tested; moreover, this was a "sanity check" instead of original research. In the (near) future, ResNet and UNet should be tested as planned.

References

[1] Ari Jónsson, Robert A Morris, and Liam Pedersen. Autonomy in space: Current capabilities and future challenge. *AI magazine*, 28(4):27, 2007.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] High performance spaceflight computing overview. `https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20180003537.pdf`.

[4] Lukas Mandrake. Cosmic - capturing onboard summarization to monitor image change. 2017.

[5] Hirise releases new public suggestion page: "hiwish". `https://hirise.lpl.arizona.edu/releases/hiwish.php`.

[6] Impact processes. `https://www.uahirise.org/science_themes/impact.php`.

[7] IJ Daubar, AS McEwen, S Byrne, MR Kennedy, and B Ivanov. The current martian cratering rate. *Icarus*, 225(1):506–516, 2013.

[8] Slope streaks on a dusty planet. `https://www.uahirise.org/ESP_040386_1915`.

[9] Seasonal flows on warm martian slopes. `https://www.uahirise.org/sim/science-2011-aug-4.php`.

[10] Dust devil etch-a-sketch. `https://www.uahirise.org/ESP_013751_1115`.

[11] Kiri L Wagstaff, Alphan Altinok, Steve A Chien, Umaa Rebbapragada, Steve R Schaffer, David R Thompson, and Daniel Q Tran. Cloud filtering and novelty detection using onboard machine learning for the eo-1 spacecraft. 2017.

[12] Alphan Altinok, David R Thompson, Benjamin Bornstein, Steve A Chien, Joshua Doubleday, and John Bellardo. Real-time orbital image analysis using decision forests, with a deployment onboard the ipex spacecraft. *Journal of Field Robotics*, 33(2):187–204, 2016.

[13] Kiri L Wagstaff, Julian Panetta, Adnan Ansar, Ronald Greeley, Mary Pendleton Hoffer, Melissa Bunte, and Norbert Schörghofer. Dynamic landmarking for surface feature identification and change detection. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):49, 2012.

[14] David R. Thompson, Dmitriy Bekker, Brian Bue, Greydon Foil, Kevin Ortega, and Kiri L. Wagstaff. Texturecam. `https://github.com/davidraythompson/texturecam`, 2014.

[15] Kiri L Wagstaff, Nina L Lanza, David R Thompson, Thomas G Dietterich, and Martha S Gilmore. Guiding scientific discovery with explanations using demud. In *AAAI*, 2013.

[16] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[17] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[18] Tensorflow. `https://www.tensorflow.org`.

[19] Tensorflow estimators. `https://www.tensorflow.org/guide/estimators`.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[22] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[23] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[24] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[25] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[26] Yun Liu, Krishna Gadepalli, Mohammad Norouzi, George E Dahl, Timo Kohlberger, Aleksey Boyko, Subhashini Venugopalan, Aleksei Timofeev, Philip Q Nelson, Greg S Corrado, et al. Detecting cancer metastases on gigapixel pathology images. *arXiv preprint arXiv:1703.02442*, 2017.

[27] Adithyavairavan Murali, Lerrel Pinto, Dhiraj Gandhi, and Abhinav Gupta. Cassl: Curriculum accelerated self-supervised learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6453–6460. IEEE, 2018.

[28] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Regularizing very deep neural networks on corrupted labels. *arXiv preprint arXiv:1712.05055*, 2017.

[29] Numpy. `http://www.numpy.org`.

[30] Opencv library. `https://opencv.org`.

[31] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, NY, USA:, 2001.

9 BIOGRAPHY

Asher Trockman is an aspiring artificial intelligence researcher. He is starting his Ph.D. in the Computer Science Department at Carnegie Mellon University in the fall of 2019, where he will probably study deep learning and mathematical optimization. Previously, he was an intern in the Machine Learning and Instrument Autonomy group at the NASA Jet Propulsion Laboratory, and before that, in the Institute for Software Research at Carnegie Mellon University. He has published several papers on empirical software engineering (ICSE'18, ICSE'19, MSR'18, MSR'19) with emphasis on large-scale data analysis. He has lived in Evansville, Indiana all his life and received a B.S. in Computer Science and Mathematics from the University of Evansville. In his free time, he is a coffee enthusiast and indisputably makes the finest five-ounce cappucinos in the state of Indiana.

A MACHINE LEARNING TERMINOLOGY

A.1 Training and Testing

The data set is often broken into three sets: the training, validation, and test sets. For example, the training set may be 70% of the data set, the validation set 15%, and the test set 15%. More specifically, the training set may be 210 images, the validation set 45 images, and the test set 45 images. The investigator may then train numerous models with different parameters on the training set, choosing the one with the best performance on the validation set. The best model is then evaluated on the test set to estimate its real-world performance, or its *generalization error*. Essentially, it is unfair to test machine learning models on data that they have already seen, so some portion of the data set must be held out for fair testing [31].

A.2 Cross Validation

Cross validation is another technique to estimate the generalization error of a model. It is typically used when the data set is relatively small, for example, less than a thousand images. In k-fold cross validation, the data set is split into k evenly sized "folds", or subsets. For example, 5-fold cross validation with 300 images would involve 5 folds with 60 images each. Then, for each fold i, the model is trained on the remaining $k-1$ folds and tested on the i^{th} fold. The k test scores are then averaged to estimate the generalization error. For small data sets, this gives a more accurate estimate of real-world performance than simply splitting the data set once [31].

A.3 Transfer Learning

Transfer learning refers to fine-tuning a pre-trained machine learning model on a new data set. For example, it is common to start with a large neural network that has been trained on the ImageNet data set, which consists of millions of labelled images from the web. This complex model is then trained on a smaller data set, such as fresh impact craters. Essentially, knowledge for one task be "transferred" to another similar domain [24].