Focusing on the Big Picture: Insights into an End-to-End Systems Approach to Deep Learning for Satellite Imagery

Ritwik Gupta, Carson D. Sestili, Javier A. Vazquez-Trejo, Matthew E. Gaston
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, USA
Email: {rgupta, cdsestili, javazquez, megaston}@sei.cmu.edu

***Abstract*—Deep learning tasks are often complicated and require a variety of components working together efficiently to perform well. Due to the often large scale of these tasks, there is a necessity to iterate quickly in order to attempt a variety of methods and to find and fix bugs. While participating in IARPA's Functional Map of the World challenge, we identified challenges along the entire deep learning pipeline and found various solutions to these challenges. In this paper, we present the performance, engineering, and deep learning considerations with obtaining, processing, and modeling data, as well as underlying infrastructure considerations that support large-scale deep learning tasks. We also discuss insights and observations with regard to satellite imagery and deep learning for image classification.**

***Keywords*-satellite imagery; deep learning; software design; infrastructure design; selective classification**

I. Introduction

Identifying the functional use of facilities and land in satellite images is a problem that industry, academia, and government have explored in depth. Due to the complex and heterogeneous nature of satellite imagery [1], even within categories, classification of land use within satellite imagery is a daunting task. Furthermore, high-resolution datasets of satellite imagery are unwieldy to store, transfer, and manipulate, and performing complex learning tasks on such high-resolution data becomes complicated as systems run into storage, bandwidth, and memory issues.

IARPA, in collaboration with Johns Hopkins University's Applied Physics Laboratory (JHU APL), created a dataset of over 1 million multispectral satellite images of sites on the Earth's surface from over 200 countries [2]. This dataset contains multiple images for each site, captured over time. Each image in this dataset contains at least one example of a facility or land whose function falls into one of 62 categories such as "military facility," "barn," or "construction site." Bounding boxes identifying portions of the image that contain each instance of facility or land are also provided. Each image is accompanied with metadata providing information about each satellite image capture, such as geospatial location, date, time, etc.

Labels for each of the 62 land use categories within each bounding box were provided for the training and validation subsets of the dataset. However, labels were not provided for the test dataset. The test dataset also contained bounding boxes that did not identify entities from any of the 62 categories, thereby creating another category called "false detection." The FMoW challenge required participants to identify each land use category or false detection instance as such in the test dataset.

During this challenge, we explored a difficult deep learning task in a competitive environment with strict deadlines and computational restrictions. This challenge necessitated an infrastructure that could support fast experimentation and agile, iterative development. In this paper, we demonstrate and instrument limitations to infrastructure and deep learning methods and present our effective workarounds. We explore 1) the underlying data and feature engineering, 2) the computational infrastructure supporting deep learning, and 3) deep learning approaches, their pros and cons, and attempted solutions to problems these approaches present.

II. Dataset

IARPA, in collaboration with JHU APL, provided the challenge with the largest publicly available satellite dataset with bounding box annotations and metadata [2]. The complete dataset is over 3.6 TB, comprised of over 1 million images. All satellite imagery is provided by the DigitalGlobe constellation[1].

The dataset is split into training, validation, and testing sets. These represent 65.2%, 11.4%, and 13.8% of the dataset, respectively. The remaining 9.6% of the dataset was sequestered for challenge validation purposes and was not accessible to competitors during the competition. Labels for the bounding boxes were not provided in the testing dataset. Furthermore, labeled instances of "false detections" were not provided in the training dataset.

A. Satellite Imagery

The satellite imagery is available in two formats: 4/8-band multispectral TIFFs and 3-band compressed JPEGs. There are over 1 million images available in the overall dataset.

[1]https://www.digitalglobe.com/about/our-constellation

Figure 1. Example images from the dataset with bounding boxes manually superimposed: bounding box around a barn (left); bounding box around a lighthouse (center); example of a false detection (right).

Satellite imagery is provided for each of the 63 categories. An unlabeled dataset of satellite images, which may contain the facility/land categories or the false detection category, is provided for testing. Imagery was collected in the visible-to-near-infrared spectrum and captured over 200 countries with wide geographical and categorical diversity.

The complete multispectral TIFF dataset is over 3.5 TB and consists of a mixture of 4-band and 8-band images. In contrast, the JPEG dataset is 207 GB and consists solely of 3-band images.

Although we used both datasets in some capacity during experimentation, we mainly relied on the 3-band JPEG dataset since working with datasets with variant dimensionality (as in the 4-band or 8-band case) led to longer development times. The smaller dataset size allowed us to iterate in a much quicker fashion since data loading, transfer, and memory requirements were lowered by an order of magnitude. However, we verified our infrastructure and methods on the full multispectral TIFF dataset to ensure that our solutions scaled.

B. Metadata

The IARPA dataset provides comprehensive metadata for all images. The metadata contains information which provides further context and environmental conditions for each image. Examples of this information include geographical location, amount of cloud cover, angles of the satellite sensor, and much more. Some metadata fields are categorical while others are numerical (discrete and continuous). Not all metadata fields share the same numerical scale.

III. Obtaining Data

IARPA made the dataset publicly available and ready to download via two methods: 1) Requester Pays AWS S3 buckets, and 2) P2P BitTorrent. For most competitors, using Requester Pays S3 buckets was not financially practical, as it would rougly cost $3600\,\mathrm{GB} * \$0.09/\,\mathrm{GB} = \$324$[2] to download the data. Initially, we requested data from AWS S3. However, data transfer speeds were extremely low, with estimated transfer completion times of over a month.

P2P BitTorrent proved to be a much better method of transferring the data, although it still performed slowly. With a 2x10 GbE uplink to the Internet via CMU's Parallel Data Laboratory (PDL)[3], we were able to download both the TIFF and JPEG datasets in little over one week.

BitTorrent demonstrated some limitations when downloading the data. While JHU APL hosted a seed box for the data, much of the download depended on individual contributors across the world contributing to the peer pool. However, other peers rarely provided seeding, resulting in a slower download process than otherwise expected. The CMU PDL served as a dedicated seeder for the peer pool for a few days after our completed download to aid the overall competition.

The use of the BitTorrent protocol is not supported or encouraged in many environments. In these circumstances, there may be no choice but to use S3. However, with proper data verification and data separation, using P2P protocols should not be a concern. We used a sandboxed environment to download the data and verify it (using methods such as a hash), after which we transferred it to the unsandboxed file system at large. This process kept us in compliance with our organization's rigorous security guidelines and provided us with verification that our copy of the data was intact.

IV. Data processing

Due to the highly heterogeneous nature of satellite imagery in terms of landscape, structures, cloud cover, and more, proper data processing is extremely important to learn useful features from the input data.

A. Bounding Boxes and Context

The provided metadata defines bounding boxes within the images to be classified. These bounding boxes are usually

[2]https://aws.amazon.com/s3/pricing/

[3]http://www.pdl.cmu.edu/index.shtml

tight around the area of interest (Figure 2). It would be simple to present only the bounded portion of the image to the model. However, our hypothesis was that the landscape surrounding the bounded portion of the image gave important context that would improve the representation of the overall image. Therefore, we needed a way to expand the bounding box to look at the *context*. To do this, we defined a method to create a context window and performed experiments to determine a reasonable context window size.

Figure 2. A tight bounding box around an amusement park.

We define the context window around the bounding to be dependent on a *context ratio*, C, to be

$$context\_window = \frac{C * AR}{2}$$

where AR is the aspect ratio of the image. The bounding box would be expanded to cover an extra $context\_window * width$ pixels, and by the same factor for the height (see Figure 3). A context ratio of 0 would imply no extra context window. In order to decide on a good context ratio, we performed an experiment where we trained a simple convolutional neural network on the input data while varying the context ratio. We observed that a context ratio of ˜1.5 resulted in the greatest performance (see Figure 4).

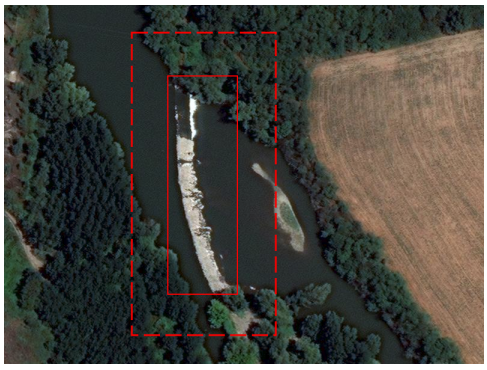

Figure 3. Example of a bounding box and the expanded context window around an instance of ”dam.”

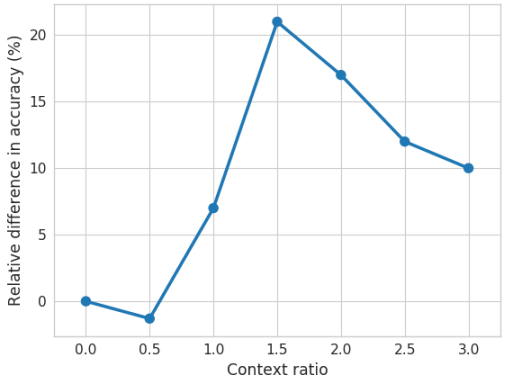

Figure 4. Relative difference in accuracy with varying context ratio.

B. Highly Variant Bounding Box Resolutions

Images within and between categories varied in resolution. While some images, such as certain instances of ”airport”, were upwards of 4000x4000 px, other categories, such as ”zoo”, had images lower than 200x200 px. Furthermore, images rarely had an aspect ratio of 1, causing aspect ratio to be a case to consider when trying to handle images of different resolutions. Convolutional neural networks have no requirement that images must be the same resolution. However, fully connected layers rely on fixed vector sizes to operate, which imposes that convolutional feature maps should be of the same dimensions. We explored multiple strategies to work around this issue.

1) Bounding Box Rescaling: A common method of handling datasets with images of varying resolutions is to rescale all images to the same size [3]. In the case of [3], all images were downsampled to a fixed common size (256x256px). However, due to the extremely large range of resolutions in our dataset, downsampling all images to a small size would cause large images to be affected much more than already small ones. To find a middle ground, we took a sample of images and computed their mean and median resolutions (Table I). Since the mean and median differed by a large margin, we chose to rescale images to a size close to the median to ensure that a large number of images would have to undergo small transformations. Furthermore, the chosen size had an aspect ratio of 1 to ensure that the convolution math in neural networks would be simple. By skewing the aspect ratio of the images, we are forcing the models to learn an inaccurate representation of the underlying data. While this is not an ideal trade-off, it was one that did not have a considerable impact. We briefly pursued better methods to maintain aspect ratios, but had to stop due to time constraints. With more time, a proper solution could be crafted.

2) Spatial Pyramid Pooling: Spatial pyramid pooling (SPP)[4] uses a bag-of-words approach to create fixed-length

Table I
SAMPLE IMAGE RESOLUTION

	Width (px)	Height (px)	Aspect Ratio
Mean	367	289	˜1.27
Median	245	196	1.25

vectors that maintain spatial information. This lets us replace the last pooling layer in the network with an SPP layer, which allows images of any input size to be fed into the model. There are significant real world limitations to this. In [4], only images of two possible resolutions are used (i.e., all images are resized to either 180x180 or 224x224, depending on the closest dimension). However, we attempted to use this approach without any resizing, leaving bounding boxes in their original resolutions. In Keras, our framework of choice for this problem, tensor allocation on the GPU is fixed and not garbage collected until later in the execution cycle. As a result, tensors are created for every image that has a unique resolution, which results in an extremely rapid GPU memory exhaustion. Because of this, our training process terminated within a few batches as no further tensors could be allocated. A possible acceptable trade-off to using SPP in a dataset, with large variation in width and height would be to create a limited amount of possible image resolutions based on summary statistics and resize images to the closest "bucket." This approach would have ensured that our GPU could maintain the allocated tensors in memory and not distort images by a large amount. Due to time limitations, we were unable to test this methodology.

Spatial pyramid pooling has been empirically shown to boost the representation capacity of the network, leading to higher accuracy[4]. Our hypothesis is that the use of such a technique would lead to large gains in accuracy on satellite imagery, which is naturally well-fit to the problem SPP attempts to solve.

C. Different Pixel Scales

Every bounding box has an associated *ground sample distance* (GSD), a field in the metadata that gives the side length, in meters, of the square on the Earth's surface that each pixel in the image represents. A wide range of GSD scales is present in the dataset, as observed in Figure 5.

Such different scales would fundamentally change the way the data is interpreted by the models. Therefore, we created a simple scaling mechanism to normalize all bounding boxes to the same scale so that every pixel represented a 1x1 meter square on the ground. Based on the distribution of ground sample distances in Figure 5, a GSD of 1 meter is not an ideal rescaling target since the median GSD is at 2 meters. However, we chose a normalized GSD of 1 meter because it is a unit number and allowed us to use simpler calculations to perform the experiment.

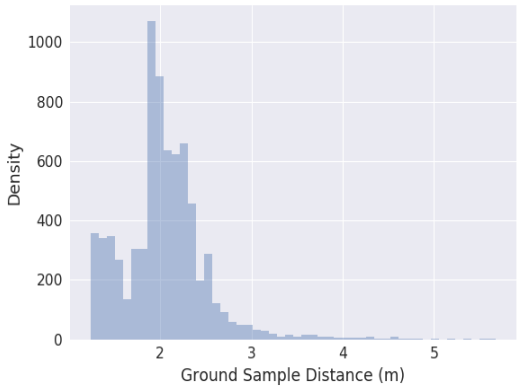

Figure 5. Distribution of ground sample distances in the dataset.

Overall, normalizing the bounding boxes to a GSD scale of 1 meter provided a marginal boost in accuracy of ˜2%. We strongly believe that normalizing to the median GSD would provide a larger boost in accuracy.

D. Data Augmentation

Data augmentation is a well explored practice to make neural networks more robust to various types of input transformations [3], [5]. We augmented both the metadata and the images to allow the models to learn a more representative function over the input space.

1) Image augmentations: We defined a set of basic transformations that would provide a large variety of alternate views on the data provided in the FMoW dataset.

- Rotations (15, 30, 45, 90, 180 degrees)
- Flips (East-West, North-South)
- Zooms (-1.5 - 1.5x)
- Channel-wise noise addition

Performing the augmentations during the training process was a time-consuming operation. On large batch sizes, we saw slowdowns in training of up to 2.5x. In order to decrease training times to perform quicker iterations, all image augmentations were preprocessed and saved to disk ahead of time. This approach provided us with a rapid training pipeline as no expensive CPU processing was performed for every image in the dataset, something which could add multiple hours onto training due to the amount of time it takes to perform these transformations.

Train-time augmentation allows us to apply a random combination of transformations to each image whereas preprocessing augmentations provides a fixed set of transformations to feed through the model. A lower amount of combinations, in the case of preprocessing, results in lower generalization. This is a trade-off which must be examined carefully. It is possible to preprocess all possible combinations, but this requires an extremely large amount of storage, something we did not have. In our case, we

were able to use the time saved in this part of the pipeline to discover methods that led to much larger increases in accuracy than train-time augmentation.

Table II
EFFECTS OF IMAGE AUGMENTATIONS

Transformation	Model Accuracy
None	42%
Rotations	53%
Rotations + Flips	69%
Rotations + Flips + Zooms	72%
Rotations + Flips + Zooms + Noise	73%

2) Image processing: With the heavy number of data transformations that we perform on a large set of data, we had to choose a fast and performant library that could perform the necessary transformations. We considered a few Python libraries to use, and compared two in the end: PIL and OpenCV.

We ran three main tests to compare performance between PIL and OpenCV. The tests were run on 300 images, averaged over 5 runs.

- **Test 1:** Load an image, blur it, and flip
- **Test 2:** Load an image, rotate by 45 degrees
- **Test 3:** Load an image, rescale to a fixed size

The comparison results are shown in Figure 6. Overall, PIL performed slower than OpenCV, especially on flips, which account for a considerable portion of our augmentations. While PIL provides an easy-to-use API, our main focus was getting maximum performance from the data processing pipeline. When dealing with a large amount of data, the disparity between the frameworks quickly adds up to a noticeable difference in time. Since our processing pipeline was relatively simple and did not require complicated, detailed image manipulation, the slightly more verbose API of OpenCV was not an issue.

The overall trade-off is decided by the size of the dataset. On a smaller dataset, the differences in time between PIL and OpenCV will not add up to a noticeable amount of time spent just processing the image. However, on the Functional Map of the World dataset, the small differences in processing add up to over an hour. Roughly,

$$\frac{(3.1-1.5)\,\text{seconds}}{300\,\text{images}} * 1,000,000\,\text{images} * \frac{1\,\text{hour}}{3600\,\text{seconds}} \approxeq 1.5\,\text{hours}$$

As evident, the extra processing time quickly adds up to an extra hour and half per model training run, which is a sizable time sink to the goal of quick iterations. Therefore, we chose OpenCV to implement our transformations.

3) Metadata augmentations: As an added challenge, IARPA introduced noise into their testing metadata. Therefore, in order to truly be robust to test-time input fluctuations, we had to ensure that the metadata was augmented to account for some uniform noise range around the true data.

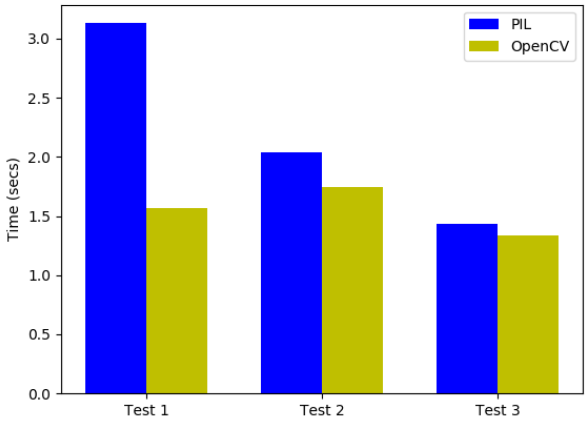

Figure 6. Comparison between PIL and OpenCV.

Therefore, we randomly created vectors of noise sampled from a uniform distribution with various ranges and added them to the normalized metadata vectors. Additionally, we added noise to the time and date in the metadata by sizable amounts to account for differences in time of day and year.

Since these are simple operations, these augmentations were left to be performed at training time. Unlike images, the metadata are simple vectors containing less than 50 elements. Because operations on these vectors are computationally simple, we did not incur any noticeable increase in training time.

Overall, metadata augmentation provided us with a ˜2-3% increase in test accuracy.

E. Infrastructure Limitations and Solutions

As a small research and prototyping lab, the CMU SEI Emerging Technology Center has general purpose hardware that is reconfigured and utilized for many different tasks as needed. To accommodate the FMoW challenge's unique and large computational requirements on our varied compute infrastructure, we identified and engineered clever trade-offs that provided significant boosts to our performance.

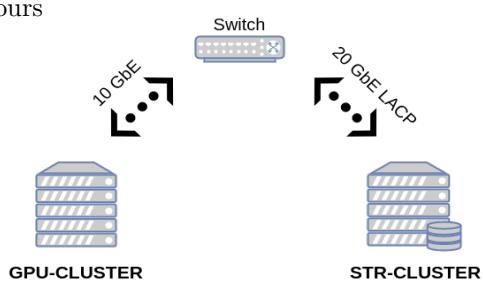

Figure 7. Simplified infrastructure diagram.

1) System Specification: All data storage, processing, and learning was done on two servers that the CMU SEI self-hosts. Since most machines in our computing lab are multi-tenant, we identified these two servers to be single-tenant for the duration of the competition. We will refer to these machines as GPU-CLUSTER and STR-CLUSTER, for a general-purpose GPU compute cluster and storage server, respectively. Mainly used for deep learning applications, GPU-CLUSTER is a fast, compute-heavy server that lacks a large amount of storage. STR-CLUSTER is a large storage server meant to hold and transfer data over the network and is not meant for large compute operations. Since this problem required both a large computational load as well as large storage requirements, a careful interfacing of these two systems was necessary.

GPU-CLUSTER:

- CPU: 2x Intel® Xeon® E5-2640 v4 @ 2.40GHz
- Memory: 8x 32GB (256GB) DDR4-2400MHz ECC
- Storage: 2x Intel® SSD 240GB, 2.5in SATA 6Gb/s, RAID 1 (HW)
- GPU: 8x Nvidia GeForce GTX 1080 (8GB GDDR5X, 2560 CUDA cores)

STR-CLUSTER:

- CPU: 2x Intel® Xeon® E5620 @ 2.40GHz
- Memory: 12x 8GB (96GB)
- Storage: 8x Intel HDD 2TB, RAID 6 (HW)

STR-CLUSTER and GPU-CLUSTER are connected via an Ethernet switch. STR-CLUSTER has a 2x10 GbE connection to the switch in LACP mode, for an effective 20 GbE data pipe. However, GPU-CLUSTER has a 10 GbE connection to the switch, meaning the overall system has an effective max throughput of 10 GbE (See Figure 7).

2) Slow writes and deletes: Our storage server was set up to be a general-purpose, read-operation oriented server. With deep learning tasks that include heavy data augmentation and image processing with frequent temporary file writes, this is not an ideal setup. Existent hardware RAID 6, also known as double-parity RAID due to its use of two parity stripes on each disk, on STR-CLUSTER, presented a significant challenge as RAID 6 is not meant to be used in a heavy write setup. Due to existent data on the server that was not removable, we had to identify and work around many limitations of this system.

STR-CLUSTER was mounted on GPU-CLUSTER via an NFS mount. The RAID 6 hard drive setup, high hard disk latency, and less-than-ideal network bandwidth usage resulted in slow random create and delete operations. While random creates and deletes were at an unacceptable threshold (see Figure 8, random reads were at a reasonable level. We decided to take full advantage of our highly performant random reads and created a system to work around our high create workload. By creating a 100 GB RAM disk on GPU-CLUSTER, we were able to stream data from STR-CLUSTER into the RAM disk via sequential block reads (which were ˜100x lower latency), perform heavy creates on RAM disk, and almost instantly delete the relevant files. With careful engineering and tuning, we developed a system to seamlessly stream files and get a speedup of ˜5x on our overall filesystem operations.

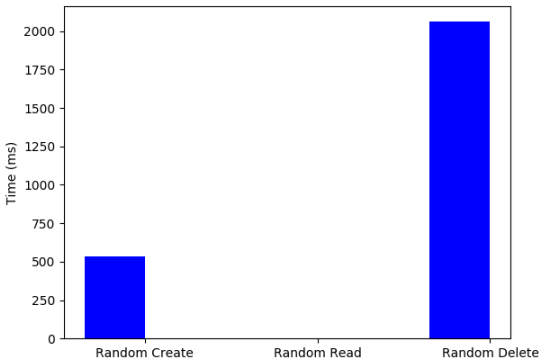

Figure 8. Disk latency on random access creates, reads, and deletes.

V. Deep Learning Discussion

While working on this problem, we attempted to develop and use many different deep learning and machine learning techniques. In doing so, we observed some interesting behaviors and discovered some intriguing problems that are worth discussing.

A. Pre-trained Models

Initializing models with weights trained on large datasets such as ImageNet has been shown to be unreasonably effective [6], [7]. It was previously believed that the massive size and diversity of ImageNet created a network that learned general features, but this was shown to be a flawed hypothesis [8]. In our experiments, ImageNet pre-training demonstrated dramatic improvements in accuracy. However, we hypothesized that pre-training on a dataset more similar to our actual satellite imagery dataset would result in better overall accuracy compared to ImageNet.

We used a simple VGGNet [9] convolutional neural network with no pre-trained weights as a baseline model. The model was trained on the IARPA dataset, and the resultant accuracy was logged. We repeated the experiment with VGGNet pre-trained on ImageNet as well as DeepSat [10] datasets. Figure 9 shows the results of this experiment. As expected, using pre-trained weights resulted in a massive gain in accuracy as opposed to not pre-training at all. Furthermore, we observed a ˜5% gain in accuracy when using DeepSat to pre-train the model as opposed to ImageNet. Due to the limited amount of trials and detailed methodology,

we are cautious about making claims about the veracity of this solution, especially with regard to deeper, complicated models. Given more time, we would like to further explore the use of DeepSat and compare the differences in accuracy, precision, and recall with models pre-trained on other datasets. Related work in transfer learning gives credence to this idea [11].

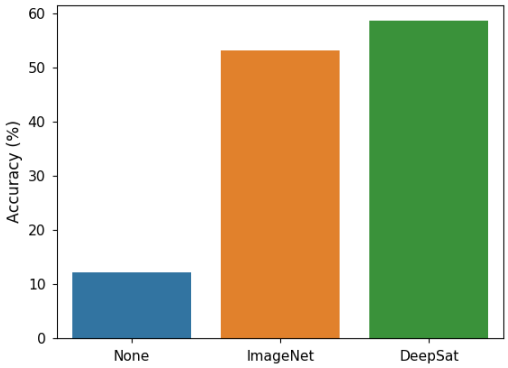

Figure 9. Comparison of accuracy with different pre-training datasets.

B. False Detections

A sizable part of this challenge was accurately classifying false detections. Incorrectly handling false detections – the 4th most populated class in the dataset – significantly affected overall precision and recall. Samples of false detections were not provided in the training data, but they were present in validation and testing data. In literature, this problem is known as selective classification or a reject option [12], [13]. We had multiple options for handling this problem.

1) Data mixing: A simple solution was to take a very limited subset of false detections provided in the validation dataset and use them to train the model. However, due to the low availability of data, this solution would have led to limited generalization and higher confusion across the entire network. [2] used this approach to achieve an acceptable gain in accuracy. As such, data mixing should only be used when larger amounts of data are available to provide a comprehensive look at false detections.

2) Random cropping: [2] do not provide a detailed explanation of how false detections were labeled in the dataset. Short of the proper methodology to create false detections, an option would be to randomly crop images containing other classes in regions that are not enclosed by existent bounding boxes. This would provide us with a false detection dataset as big as we required and would give us greater generalizability across the network. However, a significant shortcoming of this method is that a random crop may encompass an instance of another valid category. As a result, we may create "false detections" that are mislabeled.

After attempting these possible solutions individually as well as a combination of the two, we compared the precision and recall between not handling the false detection problem and our attempted solutions. Overall, the attempted solutions resulted in a greater confusion across all categories as opposed to not attempting a solution at all. Given more time and effort spent developing these methods further, we speculate that these techniques would show greater positive effect on precision and recall.

C. Neural Network Architecture Design

1) Deep learning: Many popular deep learning frameworks now exist in the open-source community for research and production purposes. TensorFlow[4], Keras[5], and PyTorch[6], and many others are popular frameworks with large open-source and industrial backings. In choosing a framework, we required one that fulfilled the following criteria:

- Ease of creating custom architectures
- Ability to write custom loss functions and optimization strategies
- Simple GPU acceleration

Architectures: TensorFlow and Keras are popular deep learning frameworks created and supported by Google. Both are static computation graph frameworks, which means that a model's structure and dataflow are defined ahead of time and cannot be modified during runtime. Conversely, PyTorch is a dynamic computation graph framework, which means that the architecture of the model can be changed during runtime. Models that we wanted to create were either all inherently static graphs, or we could easily implement the dynamic nature of the computation with some creative engineering. Out of the three, Keras presents a very straightforward and concise API, although it comes with a learning curve. All of these points made a good argument in favor of Keras, but PyTorch was also familiar since its API is straightforward and "Pythonic". TensorFlow, which Keras can use as a backend, presented an API that is verbose in many ways, especially when creating deep models that stack many layers. Since Keras can directly work with TensorFlow when needed, we preferred to use Keras where possible.

Custom functions: TensorFlow, Keras, and PyTorch all provide ways to implement custom mathematical concepts into models. Implementing novel functions in TensorFlow or Keras is difficult because of their rigid API and the requirement to use a static computation graph. On the other hand, PyTorch is extremely simple to use to create new functions via its direct, NumPy-like API and define-by-run structure.

[4]https://www.tensorflow.org/
[5]https://keras.io/
[6]http://pytorch.org/

GPU acceleration: TensorFlow and Keras both provide simple, built-in libraries to use GPU acceleration for training models. With limited boilerplate and infrastructure setup, all GPU acceleration is abstracted away from the user. While this abstraction makes TensorFlow and Keras extremely easy to use, it takes away all fine control from the user if they wish to control their own GPU memory usage. Furthermore, PyTorch provides the ability to the user to define which portions of code will use the GPU. This makes it more complicated to provide acceleration, but allows the user to granularly control GPU usage. In addition, Keras, TensorFlow, and PyTorch all provide transparent APIs to utilize multiple GPUs to enable massive data parallelization.

Overall, Keras provided us the flexibility and ease of use that we required. The custom functions that we wanted to implement were not overly complicated, and with some careful thinking and working within the framework's constraints, we would be able to create the functionality required. Furthermore, a baseline model provided by [2] was also implemented in Keras, providing us an easy reference to base our code on. We heavily considered PyTorch but preferred the plug-and-play nature of Keras instead. Had we approached this problem with complex, novel architectures, PyTorch would have been the preferred choice.

2) Use symlinks where possible: Keras provides a built in data loading utility called *ImageDataGenerator*[7]. However, *ImageDataGenerator* requires a fixed directory structure, which caused us to move data around to different directory structures depending on the data pipeline. Moving data is an operation that takes time dependent on the underlying storage medium, filesystem, and size of the data itself. Ideally, moving data should be limited as much as possible so time is not wasted, but in certain cases, it is unavoidable. In this use case, we incurred frequent moves of every image up and down the directory structure. It would have taken considerable engineering to recreate the functionality of *ImageDataGenerator* to fit our directory structure, so we decided to use symlinks to create an efficient way to move data around.

By using slow symlinks, which hold the absolute path to the data, we created an ephemeral file and directory structure that could be manipulated rapidly without shuffling entire images around. Creating the symlinks was a fast, one-time operation. Symlinks were on the order of sub-100 bytes (depending on file path length), compared to 10 MB for images. Although this method still incurred some disk latency, we were able to move entire symlinks in large blocks, providing us a ˜4x speedup compared to moving entire images around.

3) Architecture effects: Our neural network models used multiple convolutional neural networks in parallel as components, including both a network pre-trained on ImageNet and an initially untrained network. These two networks function as simultaneous feature extractors, and we combine their features by concatenating their outputs and feeding the result to a network of several dense layers. The motivation behind this choice is that while the network that was pre-trained on ImageNet can already extract features from natural images, the initially untrained network has the opportunity to learn to extract features from the problem-specific dataset of satellite images.

[7]https://keras.io/preprocessing/image/

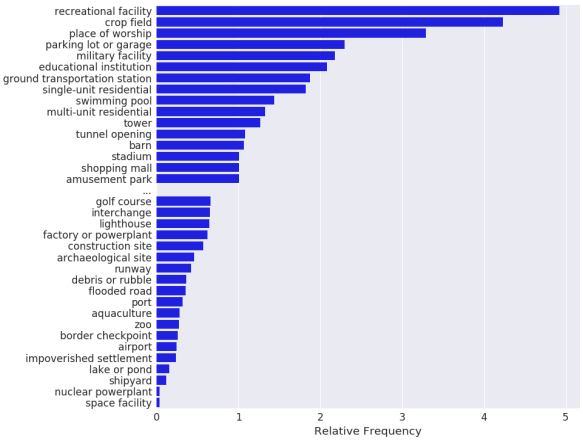

Figure 10. Relative frequencies of classes in dataset (with some omitted).

To optimize the satellite image feature extractor subnetwork, we experimented with various convolutional neural network architectures, including VGGNet [9], ResNet [14], and DenseNet [15]. We found that the best performance occurred with the DenseNet161 architecture, whose performance we compare against other architectures in Table III. We note that the comparison between architecture *families* (e.g., between ResNet architectures and DenseNet architectures) may not be fair because of the different numbers of parameters in each model, and that it is possible, for example, that a larger ResNet architecture could have outperformed the DenseNet architectures that we tried. However, ResNet is inefficient in its parameters usage [15], and thus a larger model was unable to fit on our GPUs.

Table III
EFFECTS OF NEURAL NETWORK ARCHITECTURES

Architecture	Validation Accuracy	Challenge Score
VGG16	53.1%	472774.00
ResNet50	67.3%	578552.21
DenseNet121	75.3%	614221.42
DenseNet161	78.7%	664645.25

D. Class Imbalance and Optimization

Many classes were over-represented in the dataset, while others were severely underrepresented (Figure 10), leading to a classic imbalanced class learning problem. There are

many strategies to accommodate imbalanced classes [16], [17], [18].

Over-sampling and under-sampling were implemented to modest improvements in accuracy [19]. Skewed batches were a larger problem, in which certain categories were completely missing for multiple batches. Since batch sizes were not always greater than the number of categories, we implemented a strategy in which categories were randomly sampled from the dataset with probability

$$P = \log \frac{1}{P(category)} * P(category)$$

We also implemented a guarantee that a given category would be represented within $batch\_size/63$ batches. This strategy did not have the gains in accuracy that we expected. However, it did stabilize our otherwise irregular training process, in which the network would randomly initialize in a state where the loss would keep increasing or not leave a local minimum. This sampling process ensured that our model converged at a relatively smooth rate.

VI. Insights

A. Multiple-Instance Classification

The FMoW challenge featured a unique problem. Each satellite image in the training data contained exactly one bounding box, but images in the test set contained multiple bounding boxes, each to be classified by our algorithm.

The naive approach involves simply classifying each bounding box in a testing image independently. A smarter approach would take into consideration the relationship between bounding boxes in a testing image and their labels, since bounding boxes that are in the same image are necessarily geographically close on Earth. Such an approach would involve heuristic reasoning about what land usages are likely to be found near each other. For example, one could guess that it is unlikely for a golf course to be near an archaeological site. It would therefore be desirable to make a classifier that assigns low probability to the event that two bounding boxes in the same image would have the "golf course" and "archaeological site" labels. Since the training dataset contains only one bounding box per image, assigning low probability to image combinations would be impossible to do purely through supervised learning. Possible solutions include generating surrogate secondary bounding boxes during training but asking the network to classify only the bounding box for which the true label is known, and creating a multiple-instance classifier equipped with a hand-engineered (not learned) matrix of expected class coincidences.

In the end, we decided to perform the naive, independent bounding-box classification due to time constraints.

Preliminary results from ongoing research demonstrates that more intelligent multiple-instance classification will increase the precision and recall across the categories of satellite imagery classification.

B. Importance of Data Engineering and Infrastructure Planning

After comparing both accuracy and benchmark results from models run on raw data versus models run on processed data, it is clear that careful data engineering and processing is extremely important for tasks of this scale. Furthermore, it shows that well-planned infrastructure, *even on the scale of two machines*, provides a large boost to accuracy and the ability to iterate over methodologies rapidly. It is essential for machine learning researchers and engineers to understand the systems they are working on, their limitations, and other real-world influences on the problem at hand.

VII. Conclusion

We periodically submitted our solutions to FMoW's challenge page. Over the course of the competition, our team obtained a ranking of #8 out of the 69 competing teams. We ended the competition ranked #27 due to our inability to submit results because of CMU's holiday schedule, and late submissions and data science competition specific ensembling strategies, common in online data science challenges, by other teams.

In this paper, we have identified limitations in an end-to-end deep learning task pipeline. Namely, we discuss limitations in obtaining, storing, processing, and modeling a large dataset of satellite imagery. We highlight the importance of maintaining a fast and efficient pipeline at all steps to ensure that tasks can be explored with quick iteration speeds and stability. Furthermore, we discuss limitations and possible solutions in relation to deep learning for image classification and present a few areas of future research.

Acknowledgment

We gratefully acknowledge the support of Jonathan Chu, Nathan VanHoudnos, Scott McMillan, Oren Wright, and Hollen Barmer. This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. References herein to any specific commercial product, process, or service by trade name, trade mark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by Carnegie Mellon University or its Software Engineering Institute. Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University. DM18-0176.

REFERENCES

[1] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," pp. 823–870, mar 2007. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/01431160600746456

[2] G. Christie, N. Fendley, J. Wilson, and R. Mukherjee, "Functional Map of the World," *arXiv preprint*, nov 2017. [Online]. Available: http://arxiv.org/abs/1711.07846

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, pp. 1–9, 2012. [Online]. Available: https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, jun 2015. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23

[5] H. Hosseini and R. Poovendran, "Deep Neural Networks Do Not Recognize Negative Images," *arXiv preprint*, pp. 1–3, 2017. [Online]. Available: https://arxiv.org/pdf/1703.06857.pdf

[6] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1717–1724.

[7] D. Marmanis, M. Datcu, T. Esch, and U. Stilla, "Deep Learning Earth Observation Classification Using ImageNet Pretrained Networks," *IEEE Geoscience and Remote Sensing Letters*, vol. 13, no. 1, pp. 105–109, jan 2016. [Online]. Available: http://ieeexplore.ieee.org/document/7342907/

[8] M. Huh, P. Agrawal, and A. A. Efros, "What makes ImageNet good for transfer learning?" *arXiv preprint*, 2016. [Online]. Available: https://arxiv.org/pdf/1608.08614.pdf

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations (ICRL)*, pp. 1–14, 2015. [Online]. Available: https://arxiv.org/pdf/1409.1556.pdf

[10] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. Nemani, "DeepSat - A Learning framework for Satellite Imagery," *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 1–10, 2015. [Online]. Available: http://arxiv.org/abs/1509.03602

[11] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, oct 2010. [Online]. Available: http://ieeexplore.ieee.org/document/5288526/

[12] Y. Geifman and R. El-Yaniv, "Selective Classification for Deep Neural Networks," *Advances in Neural Information Processing Systems*, no. 30, pp. 4885–4894, 2017. [Online]. Available: https://papers.nips.cc/paper/7073-selective-classification-for-deep-neural-networks.pdf

[13] M. Nadeem, J. Zucker, and B. Hanczar, "Accuracy-Rejection Curves (ARCs) for Comparing Classification Methods with a Reject Option." *Machine Learning in Systems Biology*, vol. 8, pp. 65–81, 2010.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *CoRR*, vol. abs/1512.0, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[15] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely Connected Convolutional Networks," *CoRR*, vol. abs/1608.0, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[16] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002. [Online]. Available: https://www.jair.org/media/953/live-953-2037-jair.pdf http://www.jair.org/papers/paper953.html

[17] C. Drummond and R. Holte, "C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling," *Workshop on Learning from Imbalanced Datasets II*, no. May, pp. 1–8, 2003.

[18] Y. Tang, Y.-Q. Zhang, N. V. Chawla, and S. Krasser, "SVMs Modeling for Highly Imbalanced Classification," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 39, no. 1, pp. 281–288, 2009. [Online]. Available: https://www3.nd.edu/ nchawla/papers/SMCB09.pdf

[19] N. V. Chawla, "Data Mining for Imbalanced Datasets: An Overview," in *Data Mining and Knowledge Discovery Handbook*. Boston, MA: Springer US, 2009, pp. 875–886. [Online]. Available: http://link.springer.com/10.1007/978-0-387-09823-4_45