Making Schools Safer and Resilient at Scale in Kyrgyzstan

Martin Jiang, Finlay Piroth, Eric Newcomer, Julian Rice, Sveta Selvan, Ziyi Wang, Steven Le

California Polytechnic State University, San Luis Obispo, USA

Abstract

The Global Program for Safer Schools (GPSS) at the World Bank aims to boost large-scale investments to improve the safety and resilience of school infrastructure at risk from natural hazards and enhance the quality of learning environments for children in vulnerable countries around the world. In an effort to better assess which schools are most susceptible to damage, our team has partnered with the World Bank and the Cal Poly DxHub to identify the structural type of school buildings in Kyrgyzstan. To do this, we improved upon an existing convolutional neural network that performs classifications of the school buildings; Additionally, we implemented an improved data pipeline for providing school building images from ArcGIS to our model.

1. Overview

For our project, we improved an existing system that classifies the structural type of school buildings in under-developed countries around the world. Our focus was to improve the model's classification accuracy of school buildings in Kyrgyzstan, as well as improve the data pipeline that the model relies on.

Specifically, the goal of this project was to improve the classification accuracy of the 3 current taxonomy parameters implemented by the Spring 2020 team, being building category (P0), main structural system (P1), and height range (P2). Previously, the Keras model took a single image as input, and processed each image one at a time to classify the school building within the three structural categories. The Spring 2020 team was also manually importing images for processing, so we set up a data pipeline between the ArcGIS online photo repository and our model so that the model could learn on new data and improve the classification accuracy. We also developed improvements for the model's architecture and inputs to improve its accuracy on the target categories.

The intended users for this project are the engineers hired by the World Bank to survey the schools. By using this project, engineers require less manual work and less need of mobilization in the field. Another group of possible users is the analysts at the World Bank. Given the structural classifications, analysts will be able to predict the damages that can occur in the event of natural disasters, such as earthquakes, hurricanes, and flooding. By doing so, they are able to provide advisory information to local governments and provide improved intervention and investment plans.

1.1. Background and Related Work

This project was developed in partnership with the World Bank and the Cal Poly DxHub. The project began in Spring quarter, with a team that developed the initial ML model that we inherited. The model, built on InceptionResnetV2, was used to predict 3 building criteria, according to the GLOSI taxonomy: "building category", "main structural system", and "number of stories". They achieved 81%, 67%, and 95% accuracy in these respective categories. Our initial goal was to improve these accuracies—particularly the first two parameters, building category and main structural system. However, we discovered in our research that various bugs and misinterpretations meant that these accuracies were not actually representative of the model's performance. The issues we discovered and our solutions for them are documented in later sections of this report.

1.2. Difficulty

At the beginning of the quarter, we predicted that this project would be moderately difficult, due to the model's complex architecture and our team's lack of machine learning experience. Since the goal of our project involved making complex, fine-tuning adjustments to existing generic base models, it required a lot of time-intensive trial and error. Additionally, because many of us did not have much experience in TensorFlow or Keras, we also needed to simultaneously learn the frameworks, learn how the existing codebase worked, and expand and improve upon it. Despite that we had an existing code base to start from, it was still a challenge to understand the framework because of the lack of documentation.

We also anticipated that communication would be a challenge, since we were collaborating with people in other

time zones (including one in Singapore, a 9-hour time difference). We addressed this by having flexible meeting times and using asynchronous collaboration tools such as GitHub repositories. Despite these challenges, we were able to make meaningful progress for the project.

2. Requirements

The first requirement of our project involved improving the existing data pipeline by developing a program that pulls data (images) from ArcGIS using the ArcGIS API. This program would be run on our EC2 instance, where our model is stored. This improved the workflow because we no longer had to constantly download and import thousands of images every time we trained the model on a new EC2 instance. The next (and main) requirement of our project was to experiment with the ML model to improve its classification accuracies, making sure to document our code and thought processes along the way. Over the course of the quarter we experimented in many different ways, but the methods that we found to be most successful were sending the input in batches, using a scheduled learning rate, and data augmentation.

3. Features

The features we implemented to fulfill the project requirements are outlined in this section. First off, we changed the model's architecture and inputs to improve its accuracy on the target categories. We adjusted the model so that it took 8 images in parallel (4 facade photos, 4 diaphragm photos) instead of individual images. We also implemented a scheduled learning rate, weighted losses, and data augmentation, which had varying degrees of success for our model. To improve the existing data pipeline, we developed a program to provide convenient access to the photos stored on ArcGIS. We downloaded the images from the ArcGIS website using their API for Python and organized them in a way that our model could understand. One of the issues we faced during our project was a heavily imbalanced dataset, which led to overfitting. To address this issue, we implemented data augmentation to increase the number of images in our under-represented classes. In doing so, we were able to evenly represent the different building classes in our dataset.

4. Evaluation Criteria

Due to the short-term timeline and experimental nature of this project, the World Bank did not define any set of required metrics for success. Since this project was still in the beginning phases of what is hopefully a long and fruitful project, any measurable success in the classification of the building photos would be useful for the World Bank.

The main concrete metric of the model's performance that we used is accuracy (the percentage of correct predictions out of total predictions). In order to measure the accuracy of our model, we used training, testing, and validation sets for our model. By creating a validation set, we can remove the bias of model fitting. By evaluating the validation and testing accuracy (examples that the model has not seen), we can determine how well the model will be able to generalize to new inputs in the future.

Table 1. Features, Requirements, and Evaluation Criteria

	Feature	Requirement	Evaluation Criteria
API	API that can access World Bank image database	Transfer images from ArcGIS into EC2 instance	Visually inspect images
ML model	A model that can accurately classify a building given the GLOSI taxonomy	Well organized code for creating the model	Accuracy rating from model on training and validation sets
Input multiple photos	Ability to give the model multiple photos to classify in one batch	Combine data from batch of photos into component that can be passed into machine learning model	Ensuring the input photos are properly and accurately classified.

5. System Design and Architecture

One of the primary avenues we explored in an attempt to improve the ML model was changing the existing single input ML model to a multi-input ML model. This new proposed architecture is shown in Figure 1. This architecture can handle a variable amount of input photos and output a taxonomy string for the chosen taxonomy parameters. Having multiple photos being input in parallel allowed the model to make better generalizations because it had a better understanding and perspective of the building as a whole. For a high-level overview of how the ML model being trained, see Figure 2. Another objective of the project was to create a system for downloading images from ArcGIS Online. The data from ArcGIS online will be updated with new photos in the future, so the model will need to be able to access these new photos on demand. To address this, we have created a Python script to

download the desired photos from ArcGIS online using the Python API provided by ArcGIS. The dataflow we have created is illustrated in Figure 3.

Figure 1: Block Diagram for Classification Architecture

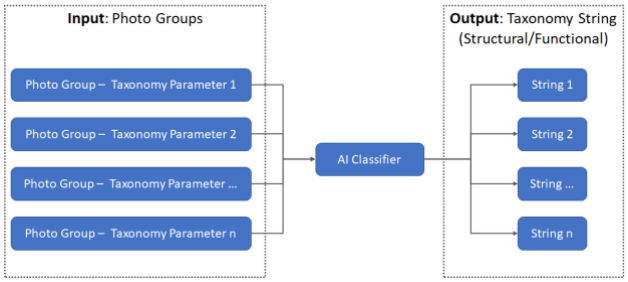


Figure 2: Model Training and Learning High-Level Block Diagram

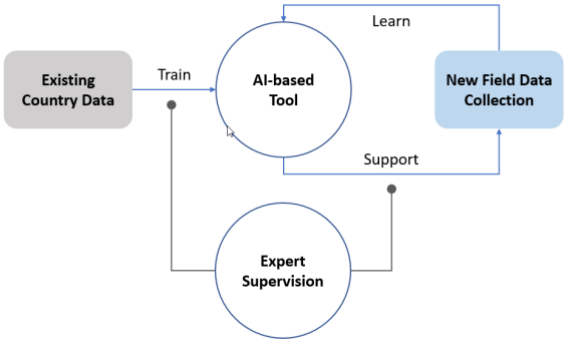
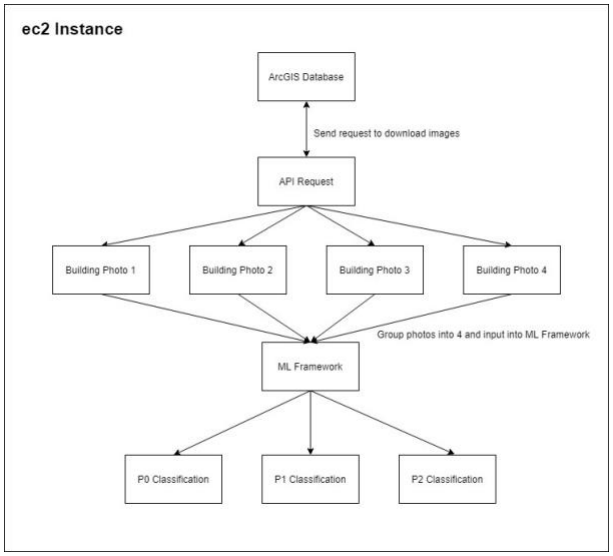


Figure 3: Data Flow inside EC2 Instance



6. Implementation

The code for our project was written in Python 3. Images and image metadata are read and manipulated using standard Python data science libraries, including OpenCV, NumPy, Pillow, Matplotlib, and Pandas. Additionally, we used the ArcGIS API for Python in order to download images to our dataset, in a specified format.

To perform our image classification, we used the machine learning framework Keras with a TensorFlow (version 2.2) backend. These tools are standard when it comes to image classification and have worked well for our project.

We are using GitHub as our version control system and most of us are using Visual Studio Code as our IDE. We were each provided with an AWS login so that we could initiate an EC2 server instance, allowing us to connect to this server and run the model with all of the input data on our local machines. We used g4dn.12xlarge instances, which have 4 powerful GPUs, to train our models.

6.1. Data Acquisition

In order to download images for our dataset, we used the ArcGIS API for Python. This API allowed us to interface with our database of school building images in the country of Kyrgyzstan. Using each school's Global ID, we were able to make individual API calls for each building. We would then extract the images from the response body of these calls and organize them into a folder structure that our model could understand.

6.2. Data Preprocessing

Once we have downloaded the images, we are ready to load them into our model. Because loading all images at once is too memory intensive, we utilized the data generator created by previous quarters' teams to load images on demand. A key improvement we made in this area is that addition of data augmentation.

Data augmentation is the use of methods such as random cropping, flipping, and addition of noise to create images that are visually similar, but are different enough that they might appear as different images to the model. This is primarily used to force the model to be able to generalize well, instead of memorizing the training dataset. However, we further utilized data augmentation to somewhat correct for data imbalances in the classes, by augmenting building images that belonged to a rare class many more times than building images from common classes. Specifically, we augmented a number of times inversely proportional to the

frequency of each class (capped at 30 augmentations).

This resulted in good improvements in the data imbalances, as can be seen in the following table:

Table 2: Top three most common classes in each category, by their percentage share of the dataset

Category	Unaugmented	Augmented
P0	89.7, 6.5, 2.5	67.9, 15.6, 10.8
P1	51.6, 21.5, 9.2	32.9, 13.7, 12.9
P2	67.8, 25.1, 6.9	66.8, 22.2, 10.2

Although it would be possible to achieve perfect parity in the dataset, it would result in a massive increase in the size of the dataset, which would be too computationally expensive unless we resorted to deleting images from overrepresented classes. Additionally, because images might belong to a rare class in P0 but a common class in P2, for example, it is difficult to achieve perfect balance. Thus, our method balances the need for imbalance-compensating augmentation against the speed and memory pitfalls of an over-complex augmentation method. Further methods to address imbalanced data are discussed in later sections.

6.3. Model Architecture and Training

[1]Figure 4: Main Model Architecture with Xception base

The main improvement we made to the model's architecture was adjusting the model to take multiple photos in parallel. The model now receives 8 photos in total, 4 building side photos and 4 diaphragm photos. We also experimented with the InceptionResnetV2, Xception, and custom models as the base model architecture.

We run each photo through the base model and group them into building side images and diaphragm images. From there we learn from each output and combine them into one layer for side images and one for diaphragm images. P2 (building height range) is related only to the building side photos, and thus we learn this output directly from those images. P0 and P1 are learned from a further combination of the side and diaphragm layers. Because class P1 is related to P0, P1 is learned from the combination of the regular output and P0's output. P3 and P4 are also output by the model, though we are not focusing on those categories yet, so they do not contribute to the loss.

Another change we implemented, to address the imbalanced dataset, was the optional use of a weighted loss. This loss works by weighting the loss of the model's prediction in each category inverse-proportionally to how common the true class is in that category. For example, if 90% of the P0 labels were Class A while only 10% of the labels were Class B, the loss (i.e., the penalty for a wrong prediction) for Class B examples would be 10x that for class A. Thus, the model has to focus on each class the same amount, regardless of the number of examples in it, in order to minimize the loss. This change actually decreased the model's accuracy, though this is to be expected, since if the model is focusing less on very common classes, it will get them wrong more often, and thus the total number of inaccurate predictions increases. However, we believe the model is learning more useful and valuable predictions, because it has to actually learn the differences between classes instead of guessing the most common one each time.

6.4. Obstacles and Implementation Issues

The project has posed some large obstacles that took some time to overcome. Most of us had no prior experience with artificial intelligence or machine learning so it has taken time for all of us to get up to speed. Additionally, it took time to learn how to use the ArcGIS API for Python; Once we learned how to use it, we were able to download images for our dataset in the correct format. The codebase that we were given was poorly documented and tested in some areas, leading to confusion and frustration. One notable instance of this was with our image loading function. The previous group's implementation took the input photos and heavily distorted them when resizing to the point of being unrecognizable as actual buildings before putting them into the model. We only realized this recently when we were trying to implement data augmentation. Fixing this problem alone increased the accuracy of our model by an average of 7% in each category.

Another notable constraint was memory. When we tried experimenting with batch size, we ran into memory

[1] Model Architecture, from Prototypes and Implementation

allocation and space issues. This was because each individual sample of data was actually 8 600x800 pixel images. With some experimentation, we determined the largest batch size we can use is 4, which actually corresponds to 32 images per batch.

Perhaps the largest issue within trying to improve the model accuracy was the heavily imbalanced data set. In some of the categories, the most prevalent class consisted of about 90% of the dataset, which led to the model overfitting and struggling to classify rarer building types. In other datasets, the most common classification consisted of about half of the dataset.

7. Validation and Evaluation

In order to measure the accuracy of our model, we used training, testing, and validation sets for our models. By creating a validation set at the beginning of the model building process we can evaluate how much our model is overfitting during training, by seeing the difference between the training and test losses and accuracies. The main criterion we used for evaluation is accuracy. This gives us the percent of successful classifications and gives the end-users an overview of the performance of our model.

8. Conclusions

At the beginning of the quarter, our team was tasked with improving an existing system that classifies the structural integrity of school buildings in under-developed countries around the world. Our focus was to improve the model's classification accuracy of school buildings in Kyrgyzstan, as well as improve the data pipeline that the model relies on. Throughout the quarter, we encountered a number of interesting challenges. These challenges included sparse documentation, broken image input in the model, and the fact that most members of the team were beginners to artificial intelligence and neural networks. Despite these challenges, we were able to meet most of the goals we set out to accomplish from the beginning; These accomplishments include an improved data pipeline, fixed image input, and improved classification for the parameters P0 (building category) and P2 (building height range).

The biggest thing we learned was a dual lesson in software engineering and AI: For software engineering, you must always verify the codebase you are given actually works; for AI, make sure you know your data well. The code we inherited boasted an accuracy of 81%, 67%, and 95% in the three categories we are focused on predicting. However, we discovered after a while that the dataset was severely imbalanced, so much so that the model was simply guessing the most common class every time to achieve the stated accuracies. Later in the quarter, while implementing data augmentation, we discovered that there was also a bug in the way image data was being loaded into the model, which distorted the images beyond all recognition.

The majority of our work this quarter went toward addressing imbalances and incongruencies in the data, instead of building novel ML models, as well as completing the development of the multi-input model architecture. The following is a summary of our achievements:

- Implemented ArcGIS API
- Created multi-input model
- Fixed image loading functions
- Improved the learning rate scheduling and other hyperparameter tuning
- Implemented data augmentation
- Added weighted loss

8.1. Future Work

In the future, there are various ways the system could potentially be improved. Some possible features that we identified are the ability to use different machine learning models to improve classification of images. Also, additional image augmentation types could be considered, such as contrast and brightness. Other possible fields of experimentation that could be looked into are using different base architectures, along with adjusting the number of parameters or the depth of the model.

Acknowledgments

Our team would like to thank Jingzhe Wu from the World Bank and Elise St. John from the Cal Poly DxHub for all of their guidance and support throughout this project. We would also like to thank Professor Kurfess for pairing our team with the Global Program for Safer Schools (GPSS) initiative and for his guidance on this project.

References

[1] "GLOSI Taxonomy Guide," *GPSS – Global Program for Safer Schools*, Oct. 2019.

[2] "Data Augmentation | TensorFlow Core," *TensorFlow*. https://www.tensorflow.org/tutorials/images/data_augmentation.

[3] Gandhi, Arun. "Data Augmentation: How to Use Deep Learning When You Have Limited Data." *AI & Machine Learning Blog*, AI & Machine Learning Blog, 6 Aug. 2019, nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.