Charlie Roslansky, Sam Johnson-Lacoss, Lily Haas
Anna Rafferty
Machine Learning

Final Project Writeup

Abstract

Hand labeling buildings on satellite imagery is time consuming and tedious, slowing down the process of prioritizing aid based on damages to communities after natural disasters. We propose the use of a U-net machine learning model to automate this process to increase efficiency. We use a U-net architecture to segment the images into building and not-building regions. Our model performs only marginally worse than a model that makes use of transfer learning from Densenet121, which was trained on the Imagenet dataset. We are able to segment the test images with 95.5% accuracy and an F1 score of 0.696 after 50 epochs of training.

1 | Introduction

1.1 | Problem

We are investigating the problem of segmenting images to identify where buildings exist. We have chosen this because it is the first subproblem of the contest surrounding the dataset we found. The contest for the dataset specifies that between two images, before and after a natural disaster, the machine learning algorithm should segment the image to identify building locations and classify each building based on how much damage it sustained during the disaster. The full problem would have been too much to solve for a class final project so we decided to focus on the image segmentation portion.

1.2 | Basic Approach

Our initial approach was to use some form of a convolutional neural network, with augmented data. After going through some literature, it was pretty clear that the best way to approach this problem would be with a U-Net architecture, which is described in Section 3. The input to our model is a satellite image of an area that may or may not include buildings. The intended output would match the labels in the given dataset. These are images in which each pixel has a value of 0 if it is not predicted to be a part of a building, and a value of 1 if it is predicted to be a part of a building.

1.3 | Other Approaches

The U-Net architecture is widely used for image segmentation across disciplines. The first instance of U-Net was used for segmentation in biomedical imaging and it has been widely adopted by those building small machine learning models for image segmentation. Other approaches to image segmentation include R-CNN and K-means clustering, but those approaches seemed less well-suited to our data.

2 | Dataset

2.1 | Base Dataset

The data we will be using hails from the Defense Innovation Unit's database as a part of their *xView2: Assess Building Damage* challenge. Although the challenge is over, we were still able to register for the event in order to gain access to the dataset. The XView2 challenge includes 130 gigabytes worth of image pairs organized among three tiers of prizes. We have decided to narrow our dataset to only use the initial training and testing data, as described below:

> The "train" dataset contains 2799 pairs of high-resolution RGB satellite imagery in PNG format, provided as a 7.8 GB GZIP archive. The datasets are organized into pairs of images. Each pair is

> a training instance: one image from before the natural disaster, and another image from after a disaster occurred. Filenames in the training set indicate the location and type of disaster, a numerical ID for the pair of images, and whether the image is a "pre" or "post" disaster event. (...) The test and holdout datasets each include 933 instances. Label information (polygon annotations and metadata) and ground truth targets (pixelwise values in PNG format) are also now available for Test and Holdout sets. (XView2)

This dataset contains 5598 images with 1024x1024 pixels and 3 color values per pixel, and for each image there exists a similarly-dimensioned mask. This poses two problems: firstly, such an image set would be simply impossible to work with in Colab due to its stringent limits on how many files are allowed to be open at one time; second, each image is of such a size that the parameters for any model we would create would need to contain approximately 3 million nodes at the level of the input layer (1024x1024x3). Such a dense and populous neural network would be entirely impractical to create and train given our time and processing constraints.

2.2 | Modification of Dataset

In order to make our data more manageable, we first had to reduce the resolution of each image. We attempted to load and train on our data three times: first with resolutions of 128x128, next with 256x256, and finally with 512x512. The first of the tests allowed for much faster completion of epochs in training, but fairly low accuracy in its results; the second, by far the most promising resolution, yielded relatively quick training (~20–100s / epoch) and a much higher rate of building prediction; finally, upon loading the 512x512 images both Colab and the high-powered Idealab computers exceeded their memory allotments before even the first epoch. As such, we have elected to use images of size 256x256.

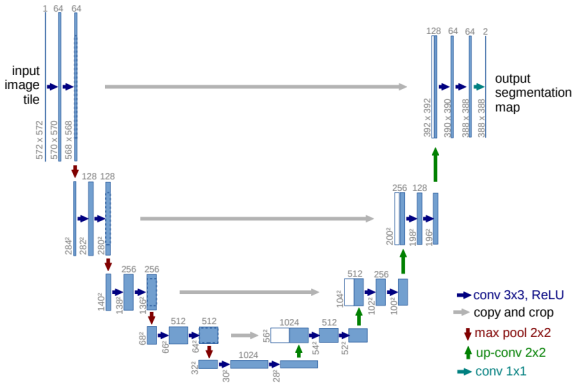
2.3 | Data Augmentation for a Robust Model

To help prevent overfitting to our training data, we augmented it in a variety of ways. For each image in the training dataset, we perform 7 types of augmentation. First, we flip the image and its mask horizontally. Next, we flip the image and its mask vertically. We also rotate the image and its mask by 90 degrees. The last augmentation that modifies both the image and its mask is cropping. The augmentation we perform exclusively on the images makes small adjustments to the brightness, gamma, and hue. This helps us combat overfitting and increases the size of our dataset greatly.

3 | Architecture — U-Net

One of the most common architectures for image segmentation is the U-Net, a modified convolutional neural network that uses a series of sliding-window convolutions decreasing in size, each one layered upon the last.

As discussed and illustrated in Ronneberger et al (2015), it specifically consists of one contracting path followed by one expanding path (pictured). The former involves a very normal convolutional neural network: it grabs a part of the image, applying two unpadded 3x3 convolutions with ReLU activations, followed by a 2x2 max pooling operation with stride 2 to cut down the dataset. After each downward step, the number of features is doubled. From there, the expanding path follows a symmetrically upward path in the network: the model upsamples its feature map and performs a 2x2 convolution to halve the number of



features that was doubled during contraction, which is then followed by 3x3 convolutions with ReLU activations. At the end of each up-convolution, the upsampled data is then recombined with the data that was not grabbed and downsampled. This symmetricality visually creates a U-shaped architecture (hence "U-net").

4 | Implementations and Results

Each of our implementations remains very similar to the original U-Net, as we wanted this exploration to really focus around the architecture. Within this architecture we created four unique implementations, three of which were modifications upon a tutorial, and the final following the abstract structure of another guide (Kibe, 2023; Mwiti, 2022). Their specific architectures and our results upon their implementations are discussed below.

For training, we randomly divided the data into 80% training and 20% validation. In training, we chose a batch size of 32, as it appeared to be the standard among the existing models that we studied both in class and for the project. For each model, we began with a learning rate of $1 \cdot 10^{-3}$, shifting down to a learning rate of $2.5 \cdot 10^{-4}$ for our second model, as we let it run for 250 epochs.

4.1 | Initial Exploration

Our first experiment was to just run the data on a U-Net model designed for segmenting images of people and clothing. After a short 10 epochs of training, our model yielded a loss of approximately 1.8 and an accuracy higher than 0.95, but the results from our initial experiment show that our model refused to predict the "building" class. We believe this is because most of the images do not have a high density of buildings, and thus the safer option for our model is to predict a building as little as possible.

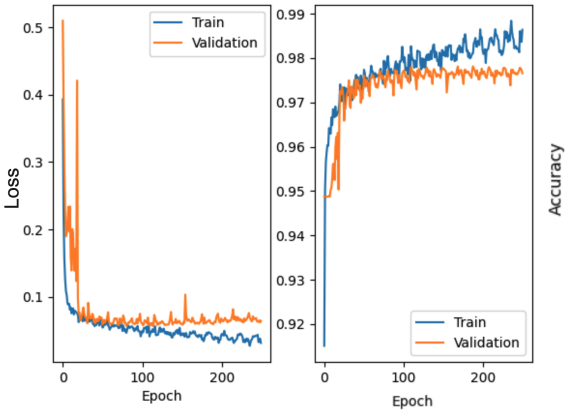
Our accuracy function for this initial exploration is the average of 0/1 losses over all pixels in the data, whereas we utilized categorical cross-entropy for our loss function. The former just gives us a simple overview of the model's performance, while the latter is the default for multiclass classification models.

After implementing some data augmentation for our training set and running for another 10 epochs, we now achieve sub-0.16 loss with an accuracy of over 0.95, and the model has begun to predict certain areas of each image as buildings (example above). Although the classification is not nearly perfect yet, these results mirror that of similar segmentation models at only 10 epochs of training. Since this model is entirely not our own, we decided not to run it on the test data before modification.

4.2 | Improvements Upon the Model

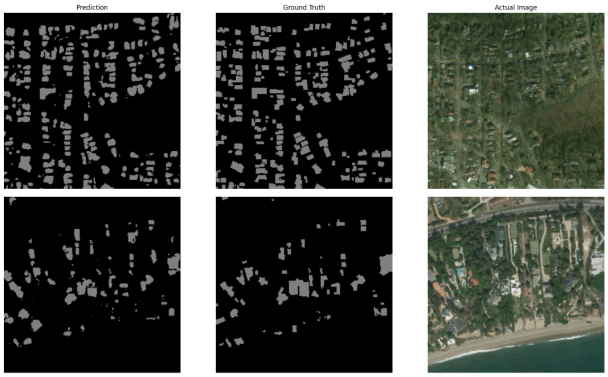
The first improvement we made was to move from Google Colab to the powerful machines in the IdeaLab, allowing us to train our models much more quickly and be able to experiment with them more.

We made multiple iterative modifications upon the original implementation, starting with a modification of the number of classes used. The original architecture, built for segmentation of clothes, offered the ability to segment 59 classes. We changed this to only two classes (as our initial



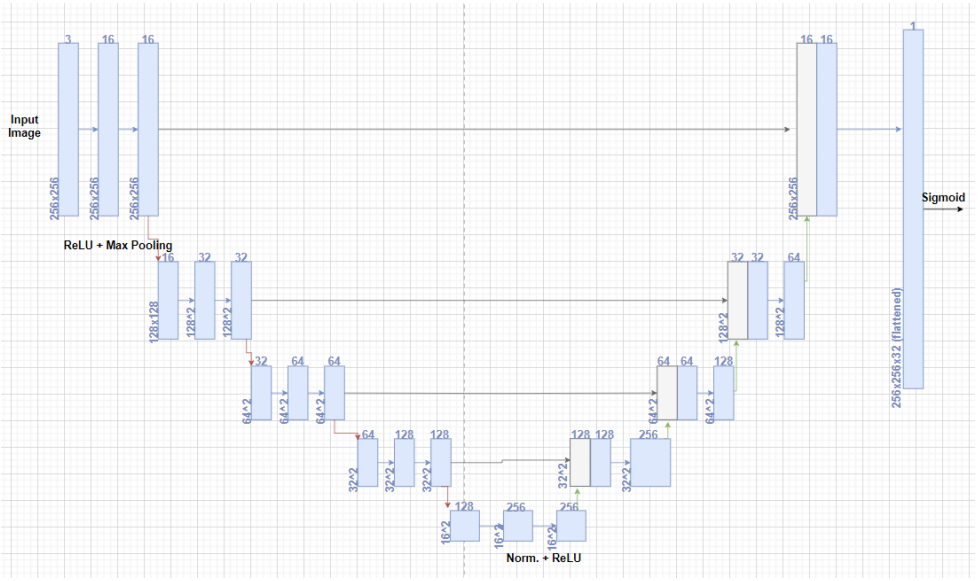
belief was that “building” and “nothing” were two separate classes). Other than this assumption, we also modified the contracting path so that it may be trainable in order to move further away from its initial weights which were plucked from ImageNet, which has a vastly different dataset from ours.

This assumption was quite worthwhile, as both our loss and accuracy found themselves much more optimal than before. We trained this model for 250 epochs, though its performance began to plateau between epochs 20 and 50. Overall, the validation accuracy increased to approximately 0.975, which did a decent job segmenting buildings. Finally, our model yielded a loss of 0.069 and an accuracy of 0.970 on our testing dataset, thus setting the gold standard for any model to follow. We can also see from its predictions in comparison with the first model that it is much better at predicting densely-populated instances of the building class.



From here, we attempted to create a third model that had a nearly identical architecture, changing the model to only segment for one class, and modifying the loss function to use Binary Cross-Entropy, a modified loss function that explicitly handles classification of binary data, which meets our needs much more specifically. This attempt failed miserably, as our model decided to never predict buildings. This is seemingly due to a local optimum being located at “all weights to 0.”

4.3 | Original Implementation



The final iteration of our model was an original implementation, while still referencing the U-Net structure that had shown promising results. The above diagram represents this architecture — the largest differences from the original U-Net architecture are the dimension sizes and the final output. Between each convolutional layer, we add a layer of dropout for generalization. After the final upscaling to the input dimensions, we include a flattened layer with a sigmoid activation function, thus translating to an

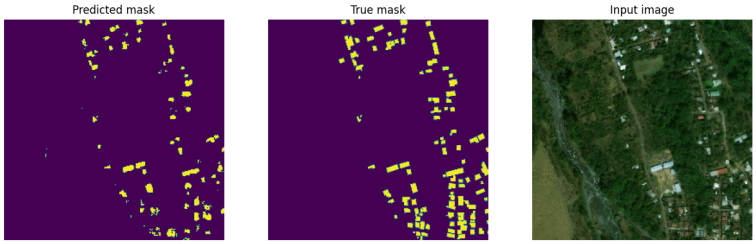
activation map of values between zero and one corresponding to the probabilities of a building on each pixel.

Another significant change that was made for this model was the implementation of a custom loss function. Instead of categorical cross-entropy, the loss function that we used was a modified version of F1-score that is differentiable (Haltuf, 2018). F1-score combines two metrics — precision and recall — that pay closer attention to the rate of false positives and false negatives in predictions. Precision is the proportion of outputs labeled as positive that were actually positive. Recall is the proportion of actual positives that were correctly labeled as positive. See the following formulas for precision, recall, and F1-score:

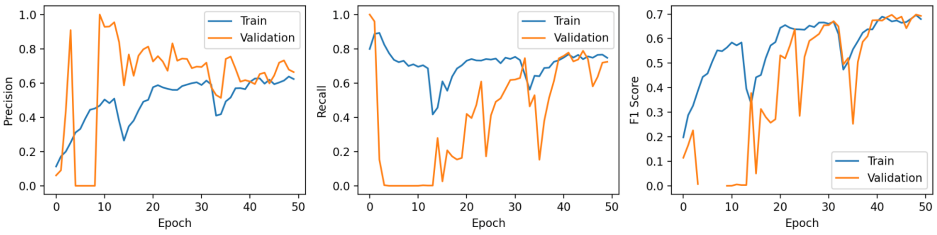
$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \qquad Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

$$F1 = 2\left(\frac{Precision \cdot Recall}{Precision + Recall}\right)$$

Typically, a high precision and low recall are signs of a model that is hesitant to classify things positively, and a low precision and high recall are signs of a model that is very quick to classify things positively. Our hope was that this loss function (which is modified to use probabilities so that it can be used for gradient descent) would give better results than categorical cross-entropy, as the increased focus on false positives and negatives would put a higher penalty on an overly cautious prediction. In the context of this challenge in particular, we believe that it is much more important to avoid false negatives than false positives — a false negative would miss a building that might need aid, while a false positive would maybe waste responders' time.



After training this model for 50 epochs, we tested it and had a loss of 0.578, accuracy of 0.955, precision of 0.654, recall of 0.747, and F1-score of 0.696. These are promising results, as an F1-score of 0.7 is generally considered to be good. While the accuracy was slightly lower, this model does do a decent job of catching most buildings, only 50 epochs into training. As can be seen from the graphs above, the F1-score is still increasing at 50 epochs. Although the current implementation performs well, it could see substantial improvement with more time to train. This may not be our best-performing model, but it is more explainable and we can better estimate when it has stopped improving.



4.4 | Tangential Observations

When working with and trying to improve our models we tried a number of different loss functions including Sparse Categorical Cross Entropy (SCCE), Binary Cross Entropy, and a custom F1 score loss function. We found that SCCE and F1 scoring were the most resistant to imbalances in our

data. We printed metrics for precision and recall when experimenting with the binary classification and we found that our recall approached zero when the model learned that the density of buildings on the images tended to be low, but their use in the custom F1 loss function yielded a more robust model which approaches the performance of SCCE. We could not read these metrics with SCCE because it does not classify binarily, but we did see a higher accuracy and printed predicted masks, which showed that the model was predicting that buildings existed. This means SCCE is a bit more of a black box in terms of precision and recall, but it did perform better and there were indications that it was in fact predicting the existence of buildings despite the lack of those metrics; however, F1 scoring seems to perform almost as well and adds an extra layer of explainability to its observed behavior.

5 | Discussion and Future Work

5.1 | Limitations

Pure 0/1 accuracy is a wonderfully simple visualization of our performance, but it expects a similar amount of each class in each image. As our dataset consisted only of high-resolution images taken from orbit, dividing each pixel into either “building” or“anything else” pixel causes the latter of these classes to appear in vastly larger quantities than the former. Judging our model by accuracy is not enough, for the difference between a perfectly-segmented image and an image that predicts no buildings whatsoever is about 5% accuracy-wise.

As noted and displayed in section 4.3, our fourth model was not trained for long enough to both locate and fit itself to a local optimum. If we were able to explore this architecture, perhaps our final endeavor would be able to surpass the performance of our second model, thus yielding a more accurate and more explainable model. However, time constraints dictated that we could not.

5.2 | Future Work

Due to time constraints, we decided not to attempt the part of the challenge that had to do with classifying the damage done to each building. If we had extra time, we would continue experiments with our various models trying to predict damage to buildings in addition to trying to segment the image. It is unclear if we would need to compute this by comparing the pre- and post-disaster images or if the model would be able to learn to classify the damages without a before and after comparison.

Given additional time and resources, we also would modify our various models to train on the full dataset without compressing the images. This would be an interesting experiment but we do not know that the machines we currently have access to could handle the computations needed to complete this task.

5.3 | Ethical Concerns

One of the sponsors of the competition surrounding this dataset was the United States Department of Defense. This means there is a non-zero probability that any of the open source entries could be used to help the army determine damage done to locations before and after attacks. This can reduce the manpower needed to verify the success of attacks like bombings and missiles. Anyone who contributed to this competition would have to decide if they want to contribute to the US military’s agenda. Additionally, since many of the entries are open source, they could be used by other governments for purposes outside of the agreement.

5.4 | Conclusion

Through this project, we explored different ways of segmenting images with a U-Net architecture and its variations. It is clear that this is a complex problem that necessitates a sophisticated model, with computing power being a significant bottleneck. Another factor to consider is the way that loss and accuracy are computed — in a situation with strongly skewed classes, a model might just learn that

skewed distribution and nothing else. Taking this into account, we were able to create a model using U-Net to achieve an accuracy of about 0.955, that did a decent job finding the buildings in a given image and could have done even better with more training. Importantly, this model looked past the skewed distribution of classes, and was able to avoid too many false negatives.

6 | Bibliography

Chollet, F. (2023, June 19). Keras documentation: [KerasCV] Image segmentation with a U-Net-like architecture. https://keras.io/examples/vision/oxford_pets_image_segmentation/

Haltuf, M. (2018, October 17). Best loss function for F1-score metric. Kaggle. https://kaggle.com/code/rejpalcz/best-loss-function-for-f1-score-metric

Kibe, K. (2023, April 5). Deep Learning for Image Segmentation with TensorFlow. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2023/04/deep-learning-for-image-segmentation-with-tensorflow/

Module: Tf | TensorFlow v2.14.0. (n.d.). TensorFlow. Retrieved November 18, 2023, from https://www.tensorflow.org/api_docs/python/tf

Mwiti, D. (2022, December 18). Create U-Net from scratch (Image segmentation with U-Net with Keras and TensorFlow). Machine Learning Nuggets. https://www.machinelearningnuggets.com/image-segmentation-with-u-net-define-u-net-model-from-scratch-in-keras-and-tensorflow/

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation (arXiv:1505.04597). arXiv. http://arxiv.org/abs/1505.04597

Team, K. (n.d.). Keras documentation: API reference. Retrieved November 19, 2023, from https://keras.io/api/

XView2. (n.d.). Retrieved November 18, 2023, from https://xview2.org/download