Improving Disaster Relief Efforts with Satellite Imagery Analysis

Metis Project 5
Gregory Lull

Design / Intro

Objective: Improve natural disaster relief efforts by creating a model that analyzes satellite imagery for damage and recommends different areas based on damage extent.

Originally when I started this project I was only interested in image classification as a tool really. I thought it would be cool to identify buildings on a map, and I've always wondered how Google Maps does it, and how can I zoom in on almost anywhere in the world and see rendered version of that area.

Initially I wasn't getting much results for the first 1 ½ weeks, the dataset I had required some data wrangling, and my MVP wasn't even on satellite imagery. Keras didn't have a built-in UNet model, and the tutorial I was following on was examining salt sedimentation. So really my MVP was looking at how to predict salt deposits, and that was part of my presentation in the 1on1 with instructors.

Once I had a better understand of what UNet was doing, and cNN's in general by watching about 6 lectures from Stanford's Image Classification lectures (https://www.youtube.com/playlist?list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv), I started creating the code to use the Satellite imagery from spacenet, and that's when I ran into all sorts of data wrangling issues.

Data / Methodology

Quick note on hardware: Nvidia RTX 2070 GPU with CUDA 10.2, AMD Ryzen 2700x 8 core cpu, 32gb RAM, 1TB SSD, Ubuntu 18.04, Keras 1.1x.

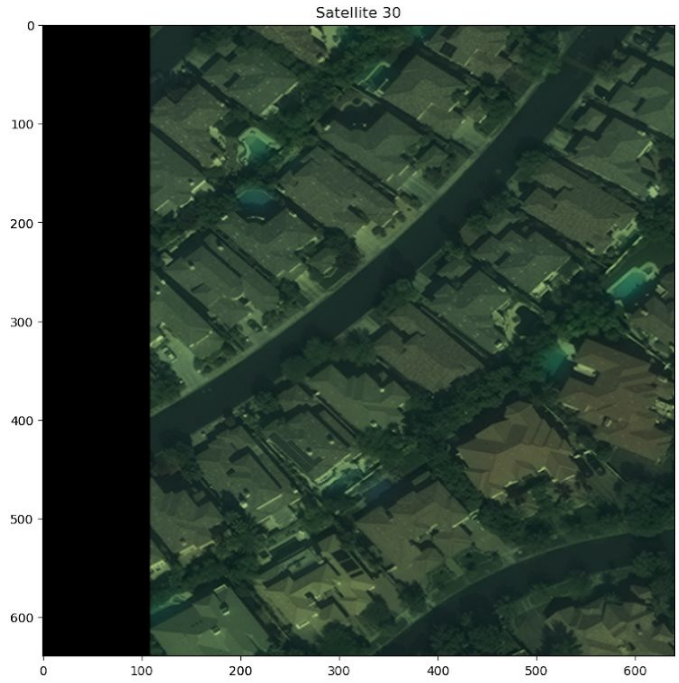
Primary Data was satellite imagery of Las Vegas n=4000 resolution=640x640x3 pixels 200mx200m area from SpaceNet with geo-coordinates for buildings that could be converted into a mask (https://spacenetchallenge.github.io/). Spacenet offers these challenges every couple of years, and the dataset I was looking at was from a couple years back, the code from tutorials didn't work anymore, and the site itself is a little bit outdated so I had to learn a couple of things before I even got started:

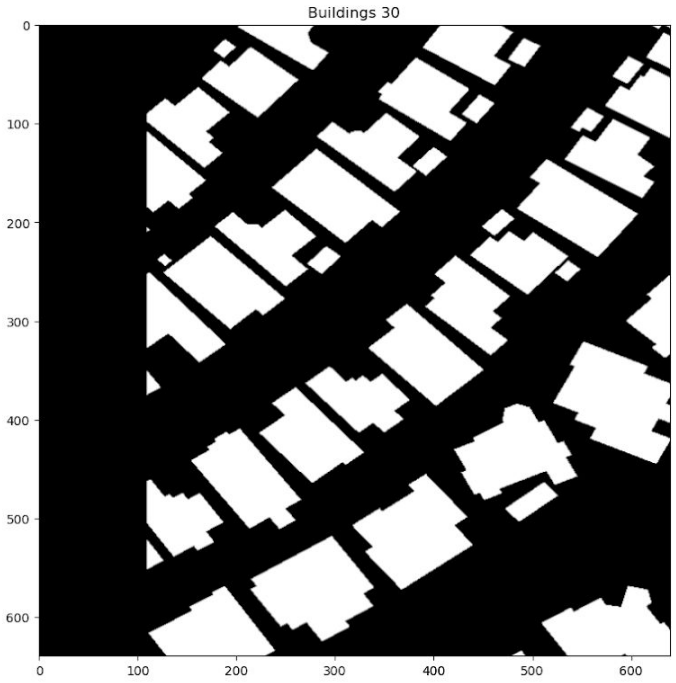
1. S3 bucket and aws-cli command line to explore and download the dataset
2. Spending hours online to find a way to view geotiff files, which ended up requiring a program called QGIS
3. Spending a couple hours more to find out how to load the .tif files into my python workspace. Load_image from keras, imread from skimage, and PIL all didn't work, there were errors that could not be bypassed. I eventually found a comment in a stackoverflow answer that mentioned using tifffile https://scikit-image.org/docs/dev/api/skimage.external.tifffile.html

I then wrote code to create buildings masks from geo coordinates, the library is called Solaris and had some tutorials online that took geojsons and original images as inputs and created masks as output.

For the convolutional neural network I used UNet, which is very popular for image segmentation and has lots of guides. Unfortunately keras does not have this model builtin, but I was able to find an online source with code and trained it on original images and masks (source for original code is in src/utilities/unet.py).

When I started training I would consistently get out of memory errors, or it would be very slow, and at Roberto's suggestion I created code to split 640x640 images into 25 128x128 images which worked out and sped up the training. This splitting resulted in 100,000 images and masks (4000 * 25). I also configured an Ubuntu 18.04 desktop with a Nvidia RTX2070 GPU to train on which reduced training time by 90%+ compared to my laptop, but with this much data it still ends up being about 2 hours to train the model on the full data set. This is an example of what the split and re-stitched image looks like with its mask counterpart:

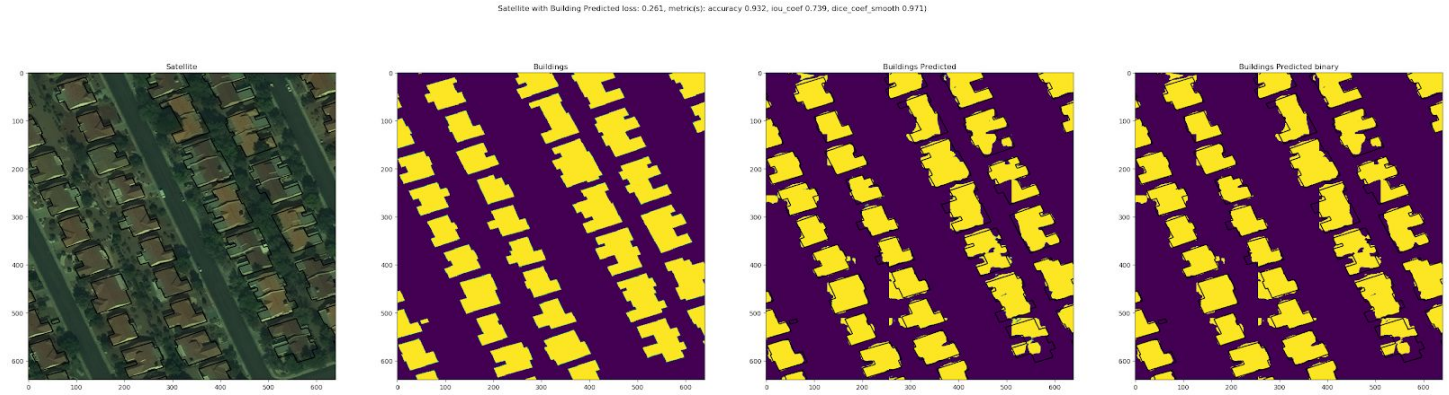




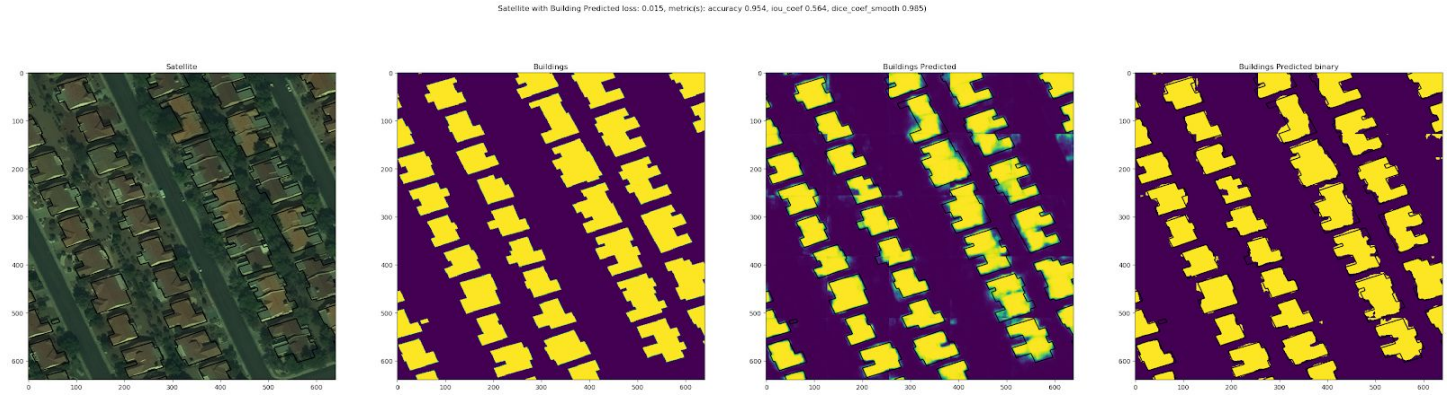
Analysis

The original metric I used was accuracy, and the loss metric was 'binary_crossentropy' which honestly performed quite well visually. I think it's probably because while the masks are imbalanced, they all still have a good amount of foreground imagery. I ended up looking into Dice, and IoU (intersection over union) and I think it would require running the training for longer than I have time for unfortunately. It takes about 70 minutes to train on just half the dataset. For these images the second from the left is the ground truth, the 3rd is softmax probabilities (0-1), and the far right is displaying the predicted with a threshold of 0.5 with the geojson boxes.

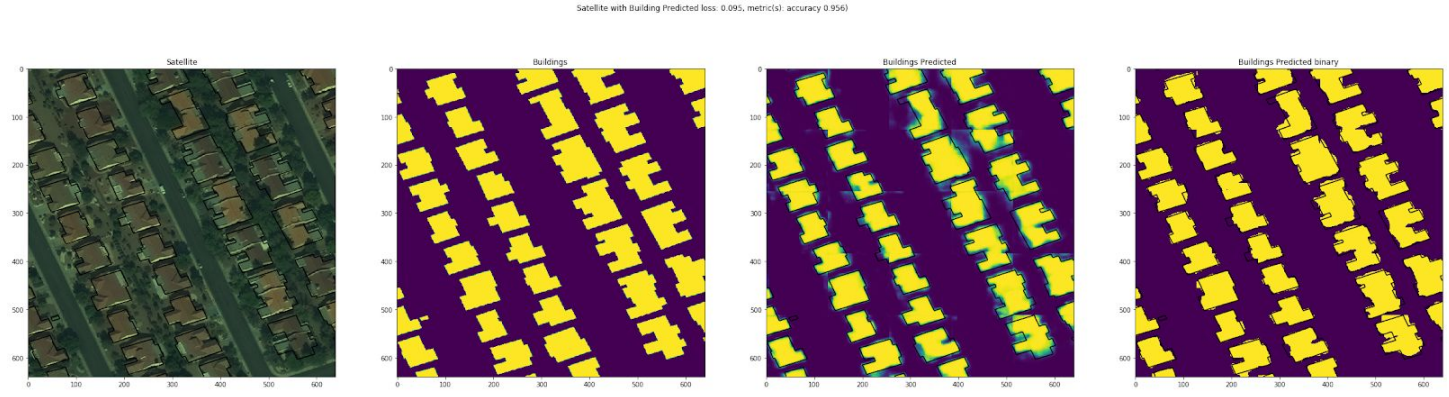
Using IoU:



Using Dice:

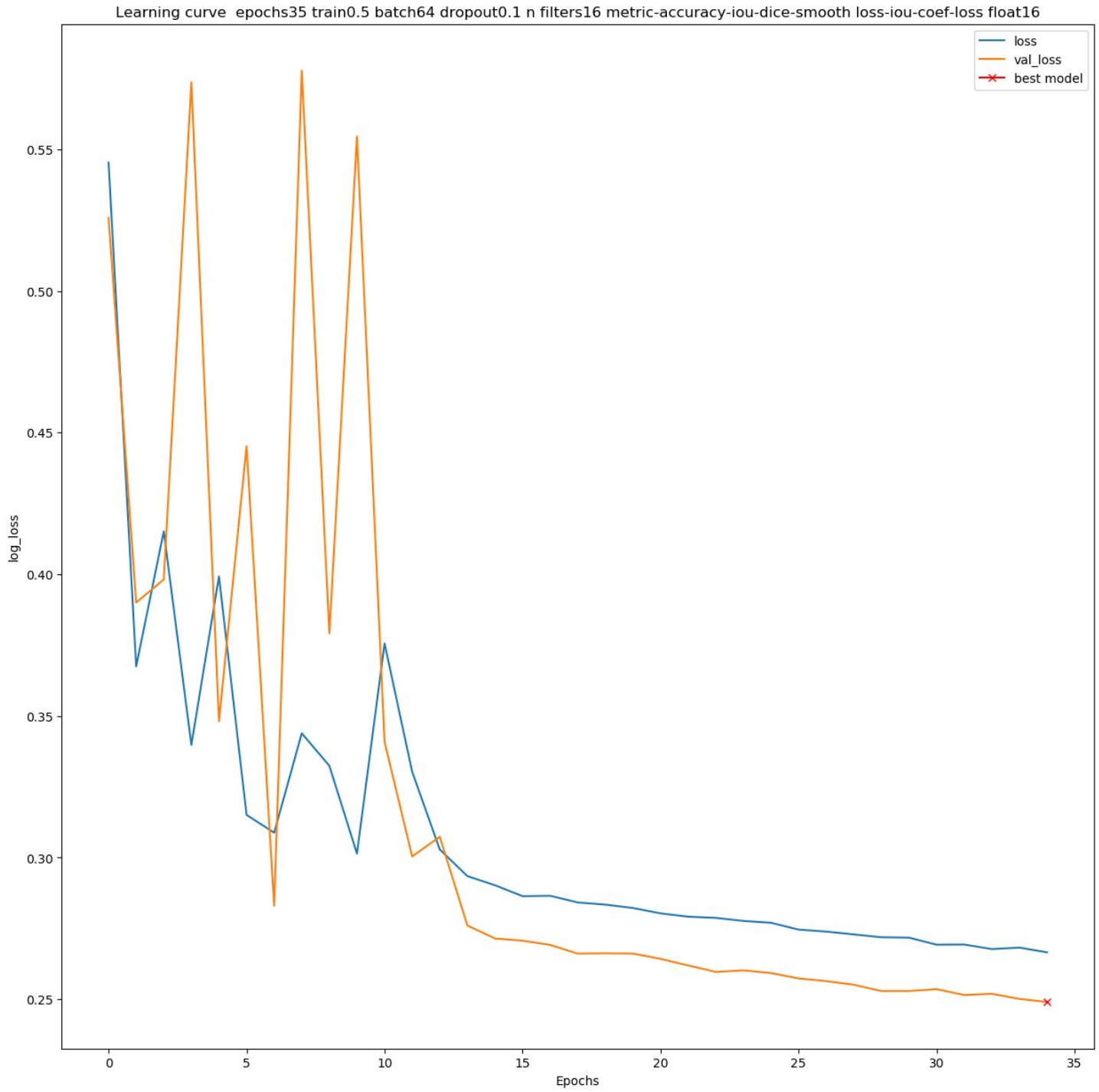


Using accuracy:

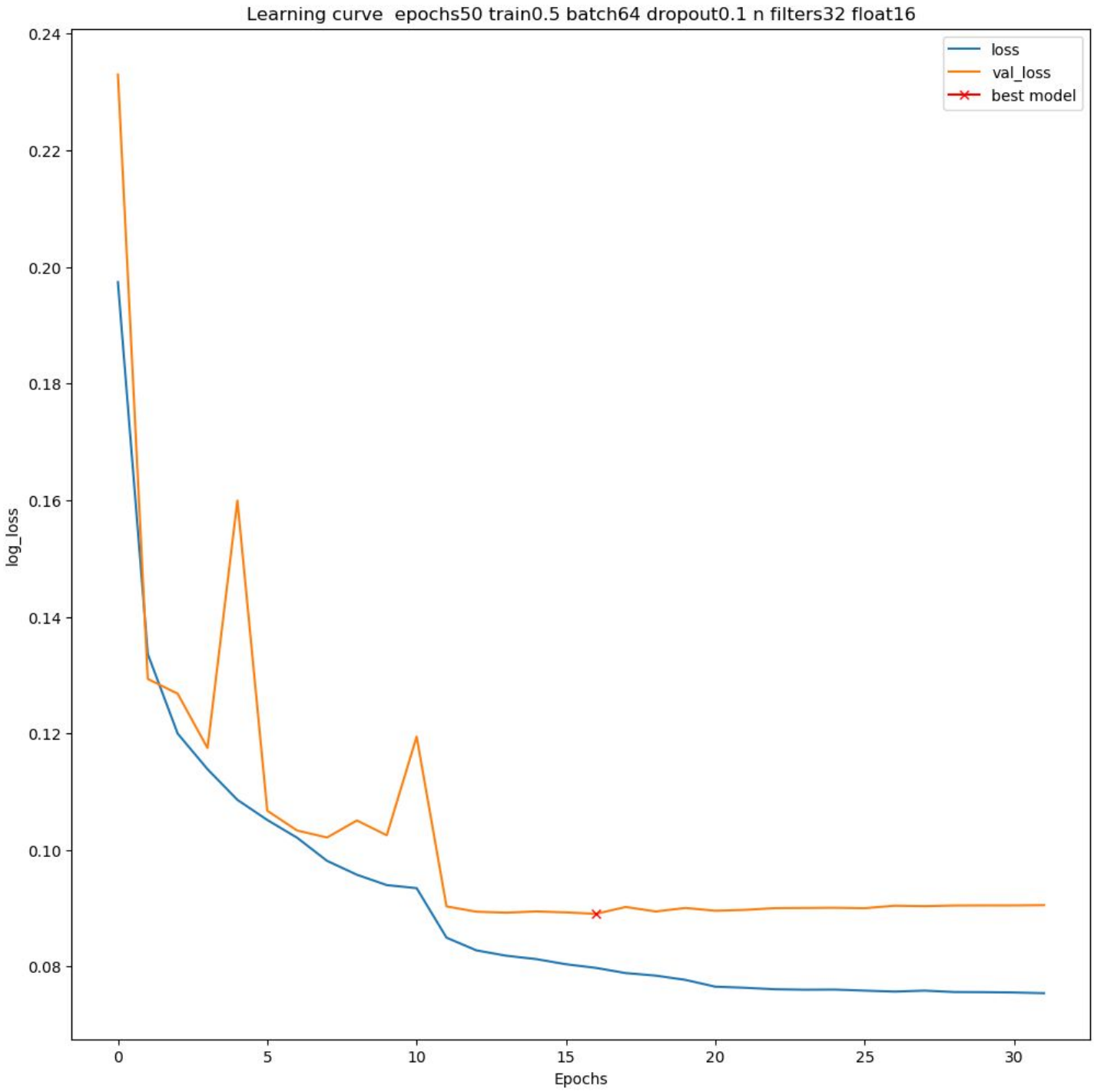


For the learning curve I had initially used 100 epochs with 20 patience, but the training was just way too slow, and I am not well versed in tuning the learning parameter, so after some initial test runs it seems like the most gain was within the first 30 - 50 epochs and that's where I set it at.

For IoU it looks like I can let it run for much longer, and as it takes quite a while to converge.



For Accuracy on the other hand the elbow and tapering off is around 20 epochs.



Results / Conclusions

This turned out to be much better than I expected and is pretty exciting. The time it takes to read the images from disk (loading 1000 images into memory and then doing some minor processing is about 6 minutes), each of these images are about ~0.015 square miles, and using a trained model to predict ~1100 of these images takes 25 seconds, so in 60 seconds this model can predict about 40 square miles. Seattle's only 83 square miles, so in about 20 minutes this model can roughly assess how the entire city of Seattle is fairing post natural disaster.

Future Work

1. Look at the Roads dataset from spacenet, this could train a model to recognize roads imagery. This combined with the buildings set could be very useful in identifying if roads are blocked when considering the logistics of getting relief into a disaster zone.
2. Look at the xView2 dataset which contains before and after images of areas that have experienced a natural disaster. This dataset has geojson coordinates as well and it's a currently live challenge, but I unfortunately do not have the bandwidth to look at it.
3. Having an app that can create a heatmap of areas that have experienced some sort of damage and then weigh it with known priorities such as hospitals, schools, homes for the elderly, shelters, etc.