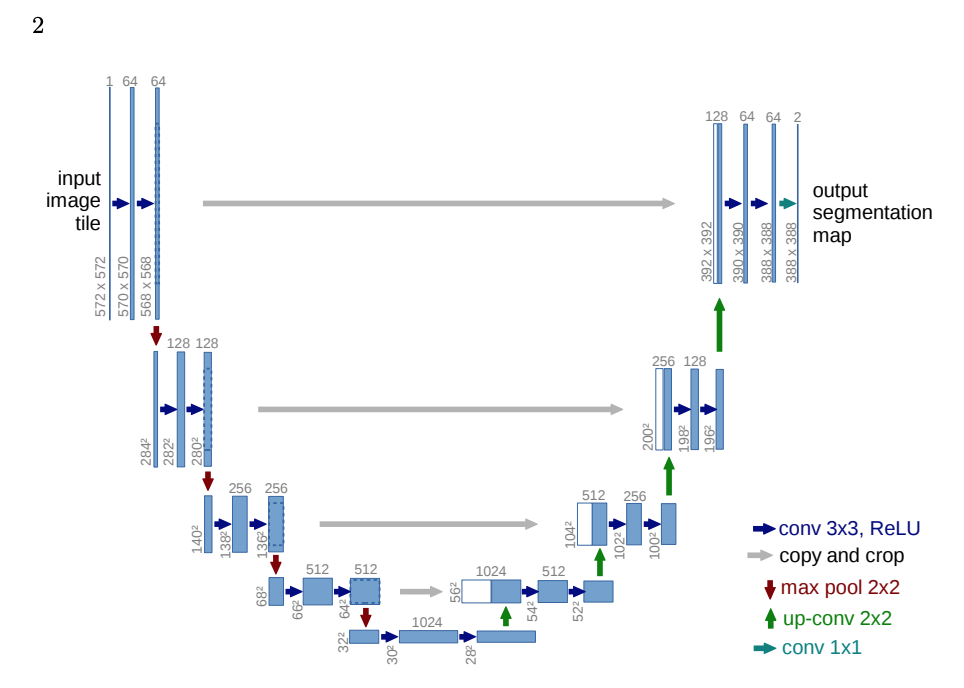
**Automated Extraction of Building Polygons from Satellite Imagery**

From the beginning of the project itself, all the research I did showed that the best way to go for this sort of problem is using a U-Net architecture. The idea was initially used in the problem of determining cell boundaries in bio-medical imagery. This is the paper for it: <https://arxiv.org/abs/1505.04597>



As can be seen from the diagram, the model is split into two parts. The left side is the encoder does convolutions, which in successive layers cover a larger context of the image, which also adding exponentially more filters. In the right side, the decoder then uses skip connections from parallel layers on the encoder to revive details that were lost during convolution, using a technique called deconvolution. The number of filters decrease on every layer in the decoder until be a segmentation map at the end.

The problem in that paper was very similar to the problem of detecting building polygons. There were also other research papers on similar problems, most of which recommended this solution. For the model, I did not want to reinvent the wheel, and found an implementation of the UNet model that was very customizable which was very helpful. The link to the implementation is: <https://github.com/jakeret/tf_unet>

I tried various variations of the UNet model throughout the project. Some modifications included:

1. Changing the number of layers and filters in the model
2. Swapping the encoder for a pre-trained encoder like the Resnet101 on image classification problems
3. Different input image dimensions

Model I:

Architecture: U-Net with low number of filters (16 on first layer) and 3 layers

Data: Bangalore data (Lepton): around 5 km2, 8000 tiles, buildings were merged in the masks

Training parameters: epochs=10, batch-size=32

Train-val-test split: 70:15:15

Ran on Google Cloud ML Engine, too approximately 12 hours

Got decent masks, but with low confidence. Could not extract polygons at all. Lots of incorrect areas in the output (masks): tree cover, mud roads, etc.

The optimizer used was the same throughout with a few experiments here and there. The Adam optimizer was best one for this purpose. The learning rate was kept at 0.001 based on the recommendations of the people who wrote the paper on the Adam optimizer. The learning parameters were played around with sometimes. Mostly the epochs and batch sizes were changed to best use the server time available to us.

The loss function was one of the areas where the choices were very spread out. The basic loss function for a segmentation problem is the binary cross entropy loss. But we also want to have some form of Intersection-over-union loss function which would treat the outputs as sets, instead of every pixel as independent. For this the two main ones were the jaccard index loss, and the dice loss. I was unsure about which was is the best, but I went with the jaccard index loss. Eventually I used a mixed cross entropy and jaccard index loss function.

Model II:

Spent a lot of time here and revamped the entire code base to make it easy to try new things quickly. Also got a lot of training data in terms of other parts of Bangalore, and a dataset downloaded from one of the COCO online challenges regarding similar mapping problems. It contained data from American cities. I was looking for ways to improve the model. A lot of papers recommended that we should use a pre-trained encoder that has been successful in image classification problems, and the best results were seen with the ResNet encoders (<https://arxiv.org/abs/1512.03385>). So incorporated that into the model. This made the model very heavy in terms of the number of parameters, and on the first attempt to run it on the ML Engine, the expected finish time was in 100 days. So, had to stop that job. Once we got the server, the code ran on it in around 10 hours for 50 epochs. The results improved a lot. Polygons could be extracted now but still a lot of merging of touching buildings. For this model too, I tried two variations:

* Used the ResNet architecture, as well as pre-trained weights for the encoder, from the ResNet project
* Used only the ResNet architecture initialized with random weights

The pretrained model worked better than the other. I also tried the similar training procedure for a model without the ResNet encoder and the same dataset (which now had around 2,40,000 images, courtesy of the COCO dataset). This model performed better for our problem and so I decided to stick with this model. But the problem of large clubbed buildings was still there. I realized that the buildings in the American dataset were much larger, because of both the resolution difference as well as the architecture of those cities. Thus, I removed the American dataset and used the 40,000 some tiles from the Bangalore dataset. The validation results were much better for the Bangalore tiles.

Tried various combinations of parameters for the UNet architecture (depth, layers, filters, etc.) and settled on the parameters that are in the config in the code. The difference was very little amongst these combinations. Training time, on the other hand, would vary dramatically to these parameter choices. Also coded the polygon generation routine. The GIS team generated more data in the meanwhile which included the follows:

* Small datasets for 8 cities in India (single polygons)
* Later, a large dataset for Hyderabad (single polygons)

For the Hyderabad dataset for the first time, the dataset was processed correctly to get the target output for the models. The masks that were generated from rasterizing the shapefiles with the building polygons, were then overlaid by a mask generated by rasterizing the polylines of the building footprints to a negative value. This ensured that the boundaries between touching buildings was visible on the mask. I used the same preprocessing for the Bangalore data too.

I trained the final version of the model on:

* Only Hyderabad
* Hyderabad and Bangalore mixed

The model performed better on the mixed model, which was the final model used for predictions. At the end, I wrote routines to split a large image into tiles, generate predictions for them, stitch them back together, and then produce a shapefile with the polygons. I was successful in creating this pipeline. And the end, the model was still below satisfactory. The major problems were:

* Irregular boundaries
* Merging of close by buildings
* Tree cover

There are some improvements that could be considered:

* Adding more data is always better (the data should be properly processed)
* Trying more combinations of pre-trained encoders, and other architecture related parameters
* Changing the loss function can heavily affect the training and the success of the model (highly recommend trying new things here)
* Weighted loss function based on the topography, eg. Weigh the boundaries between buildings close by to be very high, so the model will learn to get them right
* Different architecture altogether if necessary
* Improvements in the post-processing pipeline specially to generate polygons from binary masks (simplifying and making regular shaped)