Final Project Write-Up

Ryan Lucero

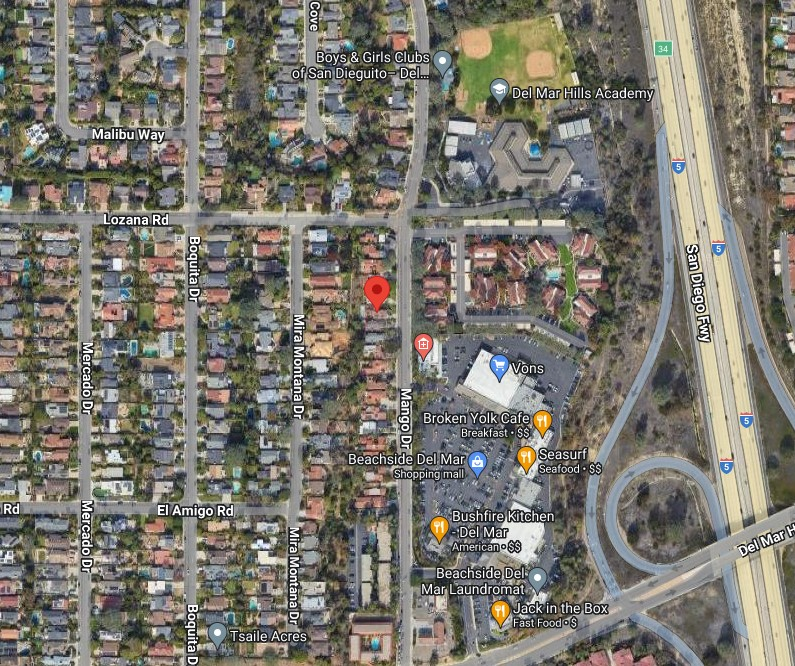
**Using Neural Networks to Identify Properties with Negative External Influences**

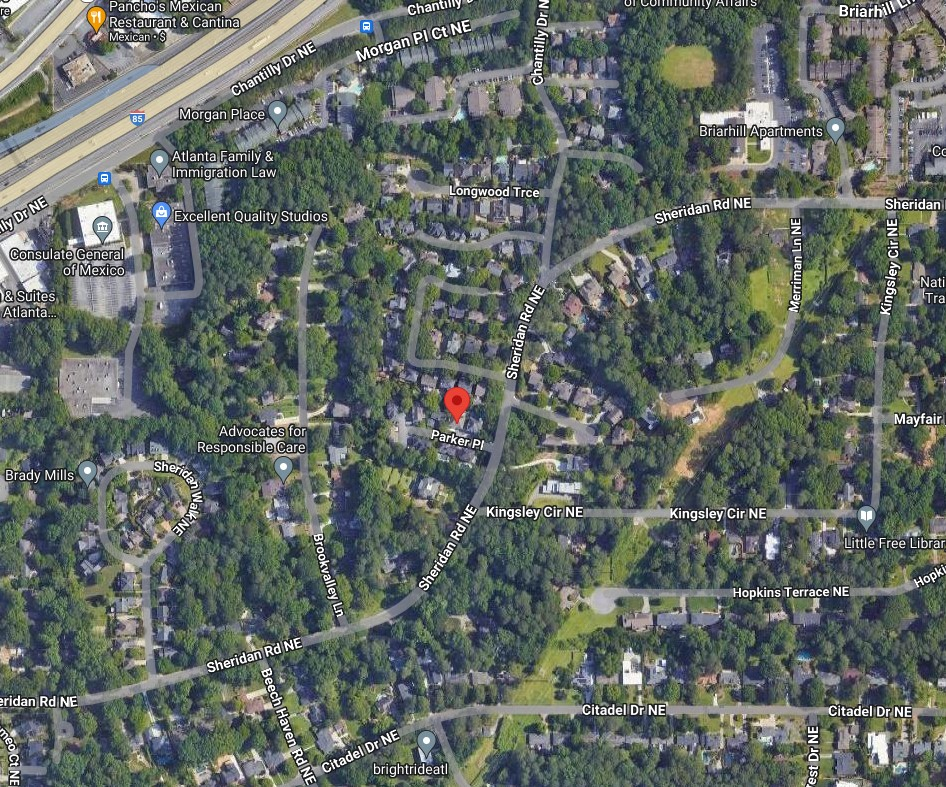
Working for an appraisal management company, one of the problems we most frequently run into is properties whose values are negatively affected by their surroundings. This is a called negative external influence. Negative influence typically occurs when a property is near a busy road/highway, commercial property, train tracks, etc. and adversely impacts the value of a property. This can also affect what kind of appraisal the property is eligible for depending on the lender’s requirements.

In a perfect world, appraisers would research the property and the lender’s appraisal requirements prior to completing the appraisal report to verify property eligibility, but appraisers frequently fail to do so, resulting in appraisal reports that are failed by the lender’s underwriters.

The goal for this project is to train a neural network to determine whether individual residential properties suffer from negative external influence. The model will take in satellite images of individual properties and output “Adverse” if the property's surroundings adversely affect its value, or “Neutral” if the location has a neutral or positive affect on value.

**Examples**

 The image above (14004 Mango Dr, Del Mar, CA 92014) is an example of a property, denoted by the red marker, that was determined to suffer from negative external influences due to its location on an arterial road and its proximity to commercial properties. The freeway might also be close enough to adversely affect the value of the subject.

 This next image (1110 Parker Pl Atlanta GA 30324) is an example of a property with a neutral location. In the photo, we can see commercial properties and a highway, but they are too far away to impact the marketability of the subject.

**Data Collection**

For this project, I manually collected 500 satellite images from Google Earth and sorted them 50/50 into Neutral and Adverse. Each photo has a red marker denoting the exact location of the property, and was collected using the same scale settings to ensure the images would be as uniform as possible. I was already tracking appraisal orders that had external issues for my company, so I searched those order numbers one at a time in our system and pulled the addresses for the satellite images. I then did the same for orders that were completed with no locational issues. The addresses are randomly distributed nationwide and contain examples of properties from a myriad of different surroundings (suburban, rural, urban, etc.). If the network is working properly, it will learn to recognize when a residence is located too close to a negative influence and what negative influences look like.

**Convolutional Neural Networks – Implementation and Refining**

The images were processed using ResNet preprocessing in batches of five and resized to 255x255. They were then split into training and validation batches using an 80/20 split. I created a basic convolutional neural network with a stack of alternated Conv2D (with ReLU activation) and MaxPooling2D layers, a Flatten layer, and an output layer with Sigmoid activation. The basic model quickly overfit to the training data and failed to produce better than 50/50 guesses on the validation set.

The second model I created was an enhancement of the base model. I added several additional convolutional and pooling layers with the hopes that the model would be able to learn more nuances in the data, and I implemented dropout to help with overfitting. The enhanced model also made more use of architectural best practices like modularity, hierarchy and reuse. There’s a block of repeated layers that progress in the number of convolution filters used (32, 64, 128, 256, 256) creating a more ideal pyramid-like structure. There was an improvement over the base model, peaking as high as 64% validation accuracy during my runs, and the model overfit slower than before.

Because both models overfit, I decided to add a data augment stage and reprocess the images. I re-ran the enhanced model on the augmented data and got very different results. The model no longer overfit, but the validation accuracy regressed to 50% again.

The next two models I tried were pre-trained Keras applications. Since I had a very small dataset, it made sense to use the pre-trained models that had already “seen” thousands of images. Additionally, the pre-trained models are very deep, and already structured using ideal architectural practices featuring several blocks. I also wanted to implement depthwise separable convolution layers as they require fewer parameters and generally help models converge faster and overfit less. This is important for me since the dataset is small.

The first model was Xception with a base model appended to the end of it. Depthwise separable convolutions are the basis of Xception architecture, which is why I selected it for this task. The base model I added to Xception was a fully connected layer, a dropout layer, a global pooling average layer, and flattening layer, and my output layer. The model yielded the best results with validation accuracy peaking at 70% and the model not overfitting. Wanting to improve the accuracy further, I swapped in the NASNetLarge model using the same structure I used for Xception. I selected the NASNetLarge model because it also utilizes depthwise separable convolutions, and because it’s the largest and deepest of the pre-trained applications. I hoped the additional parameters and layers would help the model learn more complex patterns in the data. The NASNetLarge model ran much slower than Xception due to its added complexity and did not do quite as well on the validation accuracy (65%).

The final model I tried was a fine-tuned Xception model. I loaded the weights and biases from the best run of the original Xception model and froze all but the top two inception blocks. I then reran the model utilizing a smaller learning rate for the optimizer (rmsprop) than before. This run fine-tuned only the unfrozen inception blocks alongside the top Dense layers and resulted in modest improvement and the highest validation accuracy I produced: 74%.

**Observations and Avenues for Further Exploration**

Ultimately, there were problems with the dataset that may have resulted in the models failing to produce better results. The most obvious issue would be the limited size of the dataset. It would be interesting to rerun these models with 2500 images instead of 500. I may have had the order numbers collected to do that, but pulling up the corresponding address and then taking the individual pictures was extremely time consuming.

I’m also concerned that resizing the images resulted in low resolution images that made it harder for the models to pull detailed features from. I did do a run through with larger images (360x360), but the models did not perform any better and ran much slower. Theoretically, you could run the images through at high resolution and possibly produce better results, but that would require much greater computing capacity.

Related to this, I’d be interested in using different zooms in for the satellite images. To speed up the image collection process, the default zoom ¾" = 200’ was used. Zoomed in images would make features larger and easier for the models to pick out.

Additionally, the type of preprocessing used could have made a big difference in the accuracy results. During one run through of my code, I tried preprocessing and augmenting the data using Xception instead of ResNet and my models had wildly different results. The enhanced basic model did not overfit to the training data, failing to go over 60% capacity, while the validation accuracy vacillated between 50-65% nearly every other epoch. My initial Xception model also failed to fit the data, only getting to 60% on the training data and peaking at 56% validation accuracy (compared to 70% with the ResNet preprocessing). I was unable to find specific documentation for the two preprocessors, but it’s possible they use different normalization methods, and that my dataset may align with the ResNet preprocessing method better than the Xception preprocessing.

Another thing I noticed during the data collection (and learned through working directly with the subject matter) is that the difference between a subject labeled with an Adverse location versus a Neutral location is somewhat arbitrary. Sometimes it is obvious. For example, whenever a subject is directly adjacent to commercial property, train tracks or highway, it clearly suffers from negative external influence. However, whether a subject is located too close to a negative external influence, or is located on a busy road, the determination of the subject’s location is up to the individual appraiser’s definition of “close” or “busy”. The appraiser is supposed to make the determination based on market analysis (do comparable properties nearby sell for higher due to having a better location?), but many appraisers look at the satellite image first, and then make the decision. Due to this grey area, some of the properties in the Adverse category have nearly identical location qualities to properties in the Neutral category. A model cannot replicate this logic.

If I were to re-do this project, I would change from using accuracy to using recall as my preferred metric. I would want the model to identify which properties have *potentially* Adverse locations with 100% certainty. This would mean flagging a property for a possible locational issue, instead of sorting each property perfectly into the two categories. This flag would be used to warn an appraiser or underwriter and trigger an appropriate response. This would make the most sense if the model were to ever be utilized in a professional setting.

**References**

Chollet, F. (2017). Deep learning with Python (2nd ed.). Manning Publications Co.