Investigating a New Approach to Classifying Geospatial Polygons

Group 1: Dan Vitale, Limin Tan, Nathan Justice

Begin Forward – clarifications by Nathan Justice

What we aimed to do with this project was to train a 2D convolutional neural network that classifies spatial polygons based solely on their geometry, without incorporating any information related to their source, geographic position, or descriptive attributes. Based on our research, there is no indication that this impossible. The paper we modeled (van't Veer 2019) showed using experimental procedures, a neural network can perform competitively with traditional machine learning approaches trained on Fourier descriptors extracted from spatial polygons. Regardless of the accuracy, which is dependent on the data, the question both we and the authors of the paper attempted to answer is whether or not deep learning approaches are viable candidates for solving this problem.

As mentioned above, it makes sense the accuracy will be dependent on the training data and difficult to use in a generalized setting. When we started this project one of the first things we did was train a 2D CNN to classify species taxonomic class using the shapes of polygons representing geographic ranges across the United States. We achieved an accuracy of 55% in one of our implementations. We didn't know if this accuracy was good or bad because we had no basis to make a comparison, which is exactly why we used the paper as a point of reference for whether or not 2D CNN's make sense as a modeling approach compared to other commonly used models for this type of problem.

I know from experience that extremely high accuracy can be achieved classifying spatial polygons without any attribute information. I did this exact project for my Machine Learning I final. I scraped a bunch of random polygon shapefiles from Montana's GIS web portal and

added Montana land owned by the Bureau of Land Management as my target label – a binary classification. I trained a bunch of machine learning models using feature extractions that I came up with. I ended up with a 94% prediction accuracy using random forest, 93% with decision tree, and 92% with k-nearest neighbors. I impressed James very much with this project, so much so that when I went into my Capstone last week James introduced me to the other judges by saying I had a very impressive Machine Learning I final. He asked if I continued working on it at all, I said yes I'm using it for my Machine Learning II project because I want to be able to use images as inputs.

I know this idea has applicability because it was proposed to me by experts. Before leaving my hometown to enroll in this program I met with the Director of Engineering at onX. The flagship product at onX is a highly praised hunting app specializing in the delivery of offline land ownership maps to mobile devices. During our meeting the director went over a handful of obstacles the company is facing with further innovating their product, problems they suspect a data scientist could be well equipped to handle. The most interesting challenge to me was the issue of determining land ownership based on spatial polygon geometry. OnX makes a GPS mapping tool for hunters and outdoorsman. One of the many features available is administrative GIS layering, where the user can identify what type of land they are in based on their location. According to the folks at onX, two of their biggest challenges with this feature relate to data integration and data resolution. In order to build these layers, the onX team has to aggregate GIS data from an extraordinary number of different sources, ranging from federal to municipal management, across the entire country. Often two given datasets are too disparate to be easily merged. A possible solution to their data integration problem is to build a classification model to predict the type of land (in this case the level of land ownership) based on the shape of the administrative boundary defining it. For example, when doing my Machine Learning I final I

hypothesized that private and block management land typically have a more box-like shape because they're defined by arbitrary decision-making. Whereas the shape of other ownership types like National Forests (belonging to the Federal Government) are defined by aspects of the landscape itself and therefore have generally more ambiguous shapes. The folks at onX loved my first attempt at solving this problem.

I have scoured the literature for answers to this problem for a year now, since I started Machine Learning I. It is possible that we are working on something that has not been fully investigated. Based on the comprehensive literature review of the one paper we did find, which again was published just earlier this year, we may be doing something close to genuinely new. Indeed, it is possible that this is an unachievable task, but we wouldn't know unless we try.

Reflecting more on our post-presentation conversation, I'm still not entirely sure what the majority of the confusion was about. Hopefully the contents of this report and the code will help clarify it. However, I do think I understand better your question about the projection of spatial polygons in images being unknown. I don't think I thought about this quite carefully enough, and yes I do understand the implication. This is an issue. That being said, there is absolutely a solution – train a model using data projected in every coordinate reference system (and ensemble?). It's an excessive solution, yes, but it is completely viable. Or in other situations, like I described with onX, if for whatever reason they need to classify a screenshot of an image because they lost the underlying data and need to relocate it, the polygon image would be in a known projection because of their internal warehousing system. Another point of confusion potentially was if we have the raw geometry data, why not just train on that? We absolutely can, and probably should, as it seems that is a better performing approach based on our results. We didn't, however, because the whole point was an investigation into whether or not a 2D CNN

with images, not an array of coordinates or a single stream of data, can be used and yield positive results.

Here's a hypothetical application of where a model like this could be really cool. Imagine you have a predictive classification model trained on the geometry of archaeological sites. A group of researchers come across a new finding, and the first thing they do is map out topology of the areas of interest, possibly even just a few square meters large for each artifact. It would be a huge asset to have a tool they could use to predicatively classify the type of archeological discovery they are standing on before excavating anything. Armed with this knowledge ahead of time, the team could make better informed decisions about how to allocate their resources and which experts they should recruit before continuing with the dig.

End Forward – clarifications by Nathan Justice

Introduction

Term	GIS Definition
Geometry	spatial representation of an object comprised of one or more points
Vector	geometry defined by vertices and edges
Feature	geospatial object
Shape	geospatial object geometry
Polygon	sequence of three or more connected lines
Multi-Polygon	feature instance with two or more polygons

Table 1. Interpretation of select terms when dealing with geospatial data and analyses

The goal of our project is to develop a deep learning framework for classifying spatial polygons, based solely on their geometry, that is more flexible, light-weight, and accurate than existing frameworks. In the paper "Deep Learning for Classification Tasks on Geospatial Vector

Polygons.”, Veer, Rein van 't et al. seek to examine the ability for deep learning methods to accurately classify geospatial vector polygons and compares this method to other supervised and unsupervised techniques. This method requires specific geospatial vector polygon data which is a type of data describing polygons in GIS software. The drawback to utilizing this type of data is that it is not always accessible or available for all types of geospatial data. Overall, when someone is looking at geospatial data, they are looking at a map and the vector polygon is often not included. To this end, we are seeking to examine a method to use convolutional neural networks to predict class labels for geospatial images for which there is no accompanying vector polygon data.

Data

It was logically evident from the beginning that given the nature of our project goal, to classify polygons based solely on their geometry, our models and achievable accuracies were going to be extremely sensitive to the type of spatial data used during training. In the early stage of the project, a full week was dedicated to “shopping” for an appropriate dataset for us to use. Although the notion of picking and choosing data for specific results runs contrary to much of the intuition of a machine learning engineer, it was an entirely appropriate endeavor for us to due to the novelty of our modeling approach. We sought to establish a proof of concept, and in order to do this we needed a curated training set of spatial data that was fully interpretable to each of us.

We began by using the Protected Areas Database of the United States (PAD-US), compiled by the U.S. Geological Survey (U.S. Geological Survey (USGS) Gap Analysis Project (GAP) 2018). Despite the clear interpretability of this dataset and the plethora of observations and available target labels, we ultimately decided to abandon it. We found mounting evidence

that due to the way PAD-US is aggregated from multiple different state agencies, erroneous data artifacts such as misaligning slivers and overlapping shapes were abundant (GreenInfo Network 2019). Since we were dealing with a sparsely documented modeling approach, we thought it best to focus on datasets with less documented noise.

Next we moved onto using CONUS_2001, the U.S. Geological Survey Gap Analysis Project Species Range Map((U.S. Geological Survey (USGS) Gap Analysis Project (GAP) 2018). There were no striking quality issues with this data. We also made an effort to incorporate global species data using the IUCN Red List species range dataset (IUCN 2019). The IUCN dataset includes everything in the CONUS dataset, so we fully transitioned to using that. Our biggest concern with regard to establishing a proof of concept was finding a dataset with enough inter-class dissimilarity between the target label, something we were worried about given the ecological and geographical properties associated with using species range data.

It was at this time that we discovered the van't Veer paper and realized we could use the same datasets, available on their GitHub repository for the paper (https://github.com/SPINLab/geometry-learning), and compare results to use as a clear metric for assessing whether or not or modeling approach was progressing in the right direction. We tried using the data the van't Veer team used from their original source, however we were unable to complete this task due to preventative paywalls and an inability to understand the Dutch language. We settled on using the WKT (Well-Known Text representation of geometry) storage for the neighborhood inhabitants and building type available on their GitHub repository. We were unable to use the archaeological dataset used in their original research because we couldn't figure out how to translate the target class labels from Dutch words into English. There was initially the same issue with the building type dataset, but fortunately there was a

conversion map from Dutch to English for all of the building types commented in one of their scripts.

All of the Python code used to handle shapefiles, such as those described above, were removed because of obsolesce. However, the source code is still retrievable in the commit logs for the project repository (https://github.com/nthnjustice/Final-Project-Group1). Please note, although we described in the presentation that the model can take input as an image, we used only the tabular WKT data for training and prediction. We were not interested in training on the coordinates in an array form, or as a single stream of data, because our investigation was into whether or not 2D CNN trained on image representations of geospatial polygons can be used as a basis for classification.

We split our data into a validation set with 20% of the data and a test set with 10% of the data regardless of the data source being used.

The process for converting geospatial polygons into 100x100 image representations of the spatial geometry went as follows:

1. Loop through the list of unique target labels
2. Subset the data by the target label in focus
3. Extract the feature geometries of the data subset in focus
4. Loop through the feature geometries
5. Get all x and y coordinates of the geometry in focus
6. Get the minimum and maximum values for both the x and y coordinates, these are relative to the polygon geometry in focus
7. Compute the geometric distance in both the x and y axes by subtracting the corresponding max coordinate value from the cooresponding minimum value

8. Calculate the x and y axes ratios, this is what's used to help scale the pixel values so they can be drawn in the correct proportion on the final image, by diividing the output image dimension (100px) by the x and y distances
9. Initiate a new Image canvas
10. Loop through each coordinate pair of feature geometries in focus
11. Compute the x- and y-axis pixel values using the scaling ratio multiplied by the corresponding maximum value for that axis minus the value of the corresponding coordinate in focus
12. Draw the new points on the image canvas and repeat
13. Look at the outputs in the data directory to verify
14. See spatial_data_setup/shp_2_png.py for implementation

Experimental Setup

"Neighborhoods"- Baseline Architecture

We began by utilizing the architecture that was used for the "Neighborhoods" data in the paper, only substituting 1D Convolutional Layers for 2D Convolutional Layers to train on our 100x100 images. This initial architecture was as follows:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 100, 100, 32)	832
activation_1 (Activation)	(None, 100, 100, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 34, 34, 32)	0
conv2d_2 (Conv2D)	(None, 34, 34, 65)	52065
activation_2 (Activation)	(None, 34, 34, 65)	0
global_average_pooling2d_1 (	(None, 65)	0
dense_1 (Dense)	(None, 32)	2112
activation_3 (Activation)	(None, 32)	0
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 2)	66
activation_4 (Activation)	(None, 2)	0

Total params: 55,075
Trainable params: 55,075
Non-trainable params: 0

Layer 1:

Conv2D layer with 5x5 kernel and depth of 32, 3x3 max pooling, ReLU activation

Layer 2:

Conv2D layer with 5x5 kernel and depth of 65, ReLU activation, and Global Average Pooling

Layer 3:

Fully-Connected Dense layer with depth of 32, ReLU activation

Layer 4:

Fully-connected Dense layer with 2 output classes and Softmax activation

This network was trained with Adam optimizer with a learning rate of 0.0001 with categorical crossentropy loss function.

"Neighborhoods"- More Complex Model

Next, we moved from the baseline architecture to build a more complicated model by adding a third 2D convolutional layer as well as spatial dropout to each convolutional layer to drop a portion of feature maps between each layer.

The architecture is as follows:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 96, 96, 32)	832
batch_normalization_1 (Batch	(None, 96, 96, 32)	128
activation_1 (Activation)	(None, 96, 96, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 32)	0
spatial_dropout2d_1 (Spatial	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 28, 28, 64)	51264
batch_normalization_2 (Batch	(None, 28, 28, 64)	256
activation_2 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 10, 10, 64)	0
spatial_dropout2d_2 (Spatial	(None, 10, 10, 64)	0
conv2d_3 (Conv2D)	(None, 6, 6, 128)	204928
batch_normalization_3 (Batch	(None, 6, 6, 128)	512
activation_3 (Activation)	(None, 6, 6, 128)	0
spatial_dropout2d_3 (Spatial	(None, 6, 6, 128)	0
global_average_pooling2d_1 (	(None, 128)	0
dense_1 (Dense)	(None, 700)	90300
activation_4 (Activation)	(None, 700)	0
dropout_1 (Dropout)	(None, 700)	0
dense_2 (Dense)	(None, 2)	1402
activation_5 (Activation)	(None, 2)	0

Total params: 349,622
Trainable params: 349,174
Non-trainable params: 448

Layer 1:

Conv2D layer with 3x3 kernel and depth of 32, batch normalization, ReLU activation, 3x3 max pooling, and 0.2 spatial dropout

Layer 2:

Conv2D layer with 5x5 kernel and depth of 64, ReLU activation, 3x3 max pooling, and 0.2 spatial dropout

Layer 3:

Conv2D layer with 3x3 kernel and depth of 128, ReLU activation, 0.2 spatial dropout, and Global Average Pooling

Layer 4:

Fully-Connected Dense layer with depth of 700, ReLU activation

Layer 5:

Fully-connected Dense layer with 2 output classes and Softmax activation

This model was trained with SGD optimizer with learning rate of 0.0001 and momentum of 0.9 and categorical crossentropy loss function.

“Buildings” Basic Model

Next, we switched to the “Buildings” dataset, comprised of the same type of shape data with different classes. The architecture was as follows:

Layer 1:

2D Convolutional layer with batch normalization, ReLU Activation, 3x3 max pooling, and 0.2 spatial dropout

Layer 2:

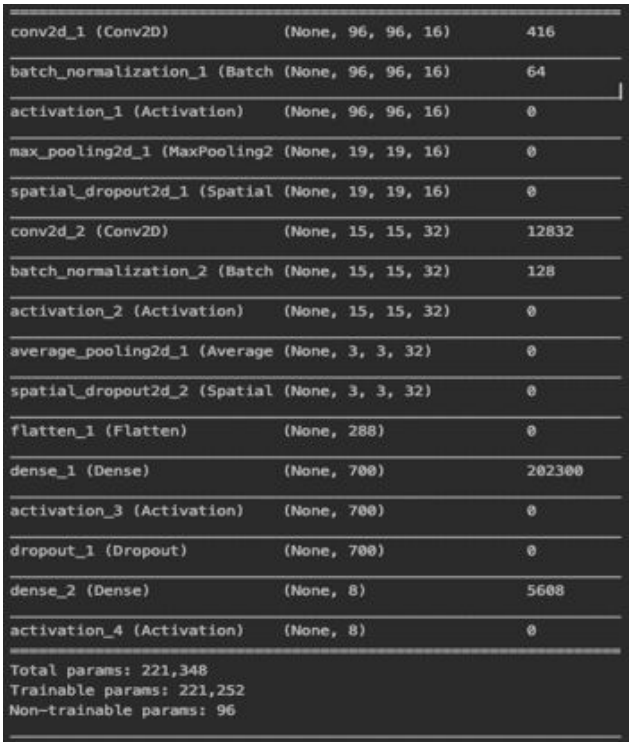
2D Convolutional layer with batch normalization, ReLU Activation, Average pooling, and 0.2 spatial dropout.

Layer 3:

Fully-connected Dense layer with depth of 700, ReLU activation, 0.5 dropout

Layer 4:

Fully-connected Dense layer with 8 output classes and ReLU activation, and Softmax activation.



conv2d_1 (Conv2D)	(None, 96, 96, 16)	416
batch_normalization_1 (Batch	(None, 96, 96, 16)	64
activation_1 (Activation)	(None, 96, 96, 16)	0
max_pooling2d_1 (MaxPooling2	(None, 19, 19, 16)	0
spatial_dropout2d_1 (Spatial	(None, 19, 19, 16)	0
conv2d_2 (Conv2D)	(None, 15, 15, 32)	12832
batch_normalization_2 (Batch	(None, 15, 15, 32)	128
activation_2 (Activation)	(None, 15, 15, 32)	0
average_pooling2d_1 (Average	(None, 3, 3, 32)	0
spatial_dropout2d_2 (Spatial	(None, 3, 3, 32)	0
flatten_1 (Flatten)	(None, 288)	0
dense_1 (Dense)	(None, 700)	202300
activation_3 (Activation)	(None, 700)	0
dropout_1 (Dropout)	(None, 700)	0
dense_2 (Dense)	(None, 8)	5608
activation_4 (Activation)	(None, 8)	0

Total params: 221,348
Trainable params: 221,252
Non-trainable params: 96

This model was trained with Adam optimizer with learning rate of 0.001 and categorical crossentropy loss function.

“Buildings” Final Model

Based on our results from the previous model, we sought to resolve overfitting, as indicated by fluctuating validation loss by adjusting the architecture. We ended up adding another convolutional layer in the following architecture:

Layer 1:

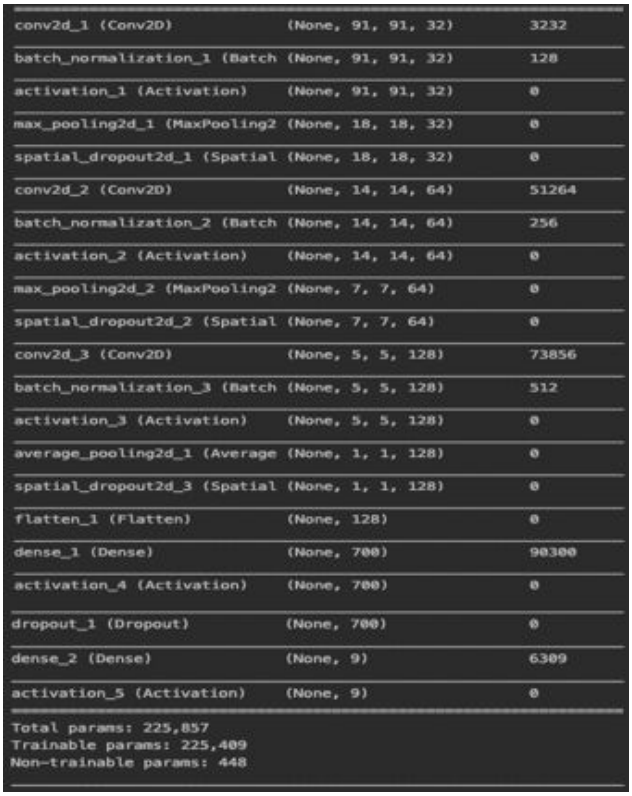
2D Convolutional layer with batch normalization, ReLU Activation, 3x3 max pooling, and 0.2 spatial dropout

Layer 2:

2D Convolutional layer with batch normalization, ReLU Activation, 3x3 max pooling, and 0.2 spatial dropout.

Layer 3:

2D Convolutional layer with batch normalization, ReLU Activation, Average Pooling, and 0.2 spatial dropout.



conv2d_1 (Conv2D)	(None, 91, 91, 32)	3232
batch_normalization_1 (Batch	(None, 91, 91, 32)	128
activation_1 (Activation)	(None, 91, 91, 32)	0
max_pooling2d_1 (MaxPooling2	(None, 18, 18, 32)	0
spatial_dropout2d_1 (Spatial	(None, 18, 18, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	51264
batch_normalization_2 (Batch	(None, 14, 14, 64)	256
activation_2 (Activation)	(None, 14, 14, 64)	0
max_pooling2d_2 (MaxPooling2	(None, 7, 7, 64)	0
spatial_dropout2d_2 (Spatial	(None, 7, 7, 64)	0
conv2d_3 (Conv2D)	(None, 5, 5, 128)	73856
batch_normalization_3 (Batch	(None, 5, 5, 128)	512
activation_3 (Activation)	(None, 5, 5, 128)	0
average_pooling2d_1 (Average	(None, 1, 1, 128)	0
spatial_dropout2d_3 (Spatial	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 700)	90300
activation_4 (Activation)	(None, 700)	0
dropout_1 (Dropout)	(None, 700)	0
dense_2 (Dense)	(None, 9)	6309
activation_5 (Activation)	(None, 9)	0

Total params: 225,857
Trainable params: 225,409
Non-trainable params: 448

Layer 4:

Fully-connected Dense layer with depth of 700, ReLU activation, 0.5 dropout

Layer 5:

Fully-connected Dense layer with 8 output classes and ReLU activation, and Softmax activation.

This model was trained with Adam optimizer with learning rate of 0.001 and categorical crossentropy loss function.

“Buildings” Pretrained Model

By applying the pretrain model, we can find the features in the picture. We can achieve the purpose of classifying pictures by adding a fully connected layer and an activation function to the final output layer of the model. To avoid taking up all the memory during the training of the model, we can know how many pictures can be accepted each time by estimating the size of each picture and multiplying by the batch size. For other training parameters, my processing method is to use the most basic model to run a small part of the data and adjust the learning rate by observing the accuracy and loss of each epoch. When I ran the most basic model for the first time, I found that the train loss continued to decrease, but the decrease in the validation loss was not so large, so I adjusted the model dropout rate, and later found that the gradient descent was faster and not smooth so I reduced learning rate.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 100, 100, 3)	0
block1_conv1 (Conv2D)	(None, 100, 100, 64)	1792
block1_conv2 (Conv2D)	(None, 100, 100, 64)	36928
block1_pool (MaxPooling2D)	(None, 50, 50, 64)	0
block2_conv1 (Conv2D)	(None, 50, 50, 128)	73856
block2_conv2 (Conv2D)	(None, 50, 50, 128)	147584
block2_pool (MaxPooling2D)	(None, 25, 25, 128)	0
block3_conv1 (Conv2D)	(None, 25, 25, 256)	295168
block3_conv2 (Conv2D)	(None, 25, 25, 256)	590080
block3_conv3 (Conv2D)	(None, 25, 25, 256)	590080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0
block4_conv1 (Conv2D)	(None, 12, 12, 512)	1180160
block4_conv2 (Conv2D)	(None, 12, 12, 512)	2359808
block4_conv3 (Conv2D)	(None, 12, 12, 512)	2359808
block4_pool (MaxPooling2D)	(None, 6, 6, 512)	0
block5_conv1 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block5_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block5_pool (MaxPooling2D)	(None, 3, 3, 512)	0

Total params: 14,714,688

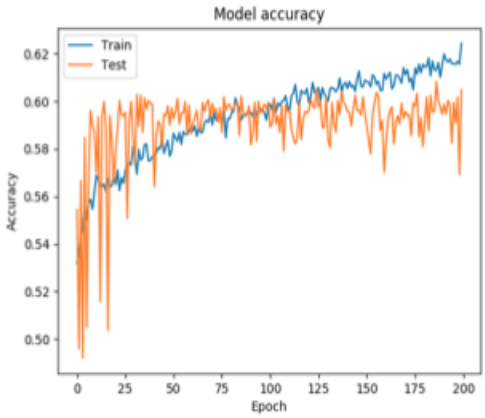
Results

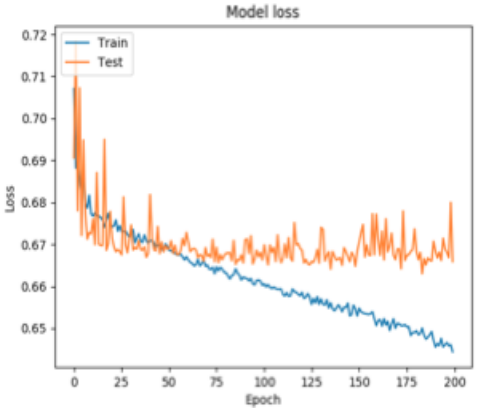
“Neighborhoods”- Baseline Architecture

Using the baseline architecture that was laid out in the paper, we found a relatively unstable model that only did a bit better than random guessing (~50% for random guess). As you can see in the figures to the right, not a lot of learning occurred, and the growing train accuracy and loss indicates overfitting after around epoch 75. Our testing set results show an overall accuracy of

58%, which is quite below the accuracy found by the 1D CNN in the paper (66%). Test results shown below:

Classification Report				
	precision	recall	f1-score	support
0	0.54	0.61	0.57	487
1	0.62	0.55	0.59	570
accuracy			0.58	1057
macro avg	0.58	0.58	0.58	1057
weighted avg	0.58	0.58	0.58	1057

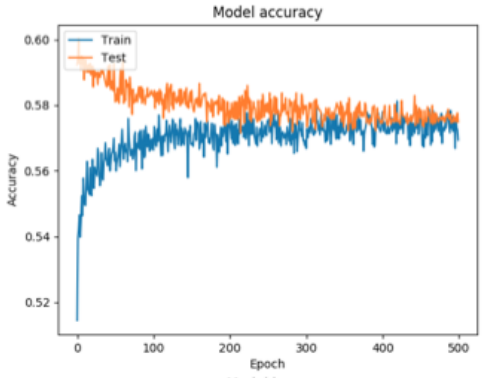


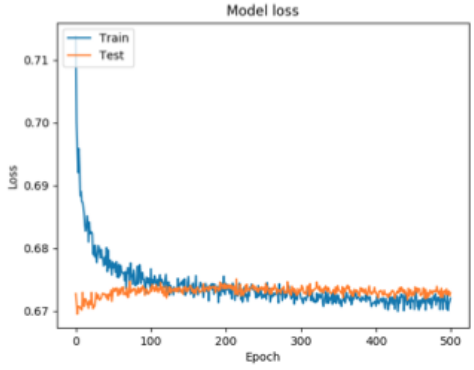


"Neighborhoods" More Complicated Model

Again, we found a model which did not seem to learn and floated just above null accuracy rate during training, only to give the null accuracy for testing as shown below:

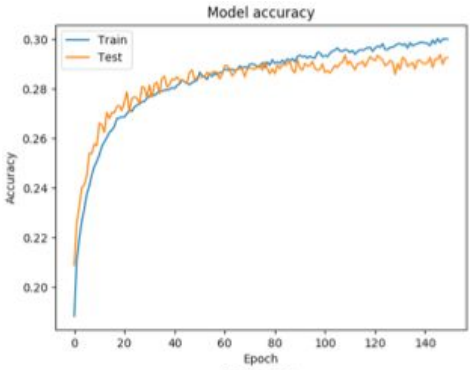
Classification Report				
	precision	recall	f1-score	support
0	0.49	0.91	0.64	487
1	0.71	0.19	0.29	570
accuracy			0.52	1057
macro avg	0.60	0.55	0.47	1057
weighted avg	0.61	0.52	0.45	1057

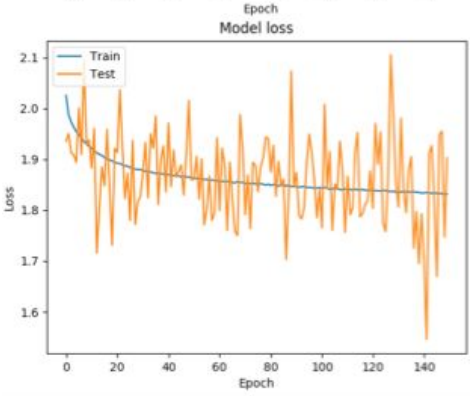




“Buildings” Initial Model

The initial model found our highest validation accuracy but very unstable validation loss. This indicated we had a degree of overfitting. Our testing accuracy made it to 0.29. Final accuracy on testing set was also 0.29 which is a significant step above null accuracy but a bit lower than what was found with 1D convolution which found 0.40 accuracy.

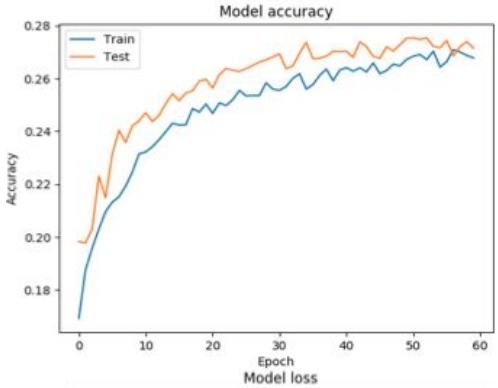


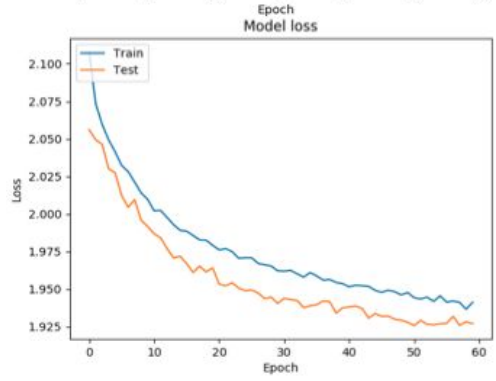


Classification Report	precision	recall	f1-score	support
0	0.32	0.63	0.42	830
1	0.34	0.09	0.14	1789
2	0.34	0.49	0.40	1802
3	0.10	0.13	0.11	637
4	0.25	0.31	0.28	1845
5	0.30	0.47	0.36	1820
6	0.19	0.03	0.06	1698
7	0.31	0.22	0.26	1884
accuracy			0.29	12305
macro avg	0.27	0.30	0.25	12305
weighted avg	0.28	0.29	0.26	12305

“Buildings” Final Model

To mitigate the overfitting in the “Buildings” initial model, we tested the effects of increasing the number of layers in the model architecture. We found a much more stable validation loss as a result, although we sacrificed some accuracy, with max validation accuracy topping out around 0.27. The final testing accuracy, shown below was also 0.27, a bit lower than the 0.40 found on this dataset with 1D CNN in the paper.

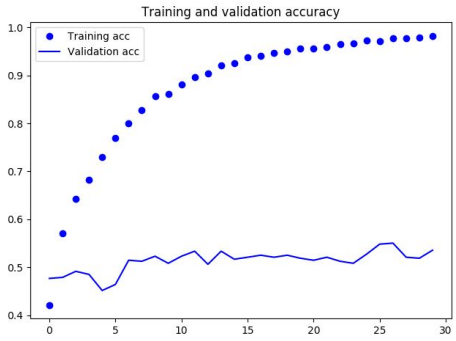


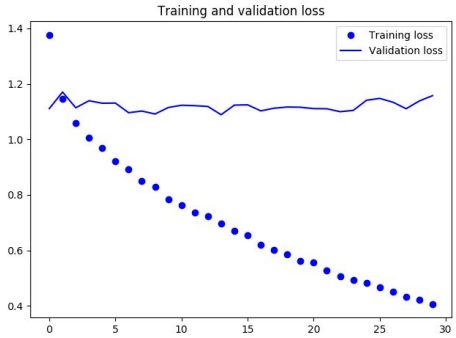


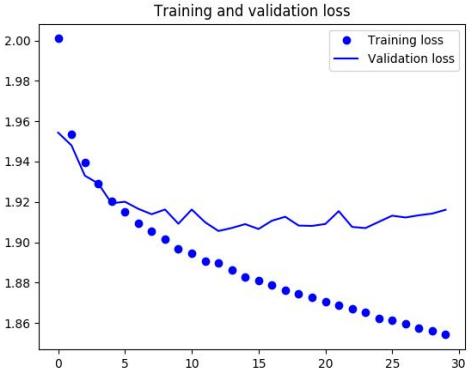
	precision	recall	f1-score	support
0	0.38	0.52	0.44	830
1	0.23	0.19	0.21	1789
2	0.34	0.42	0.37	1802
3	0.73	0.01	0.02	637
4	0.23	0.38	0.29	1845
5	0.27	0.41	0.32	1820
6	0.21	0.06	0.09	1698
7	0.26	0.22	0.24	1884
8	0.00	0.00	0.00	531
accuracy			0.27	12836
macro avg	0.29	0.25	0.22	12836
weighted avg	0.28	0.27	0.25	12836

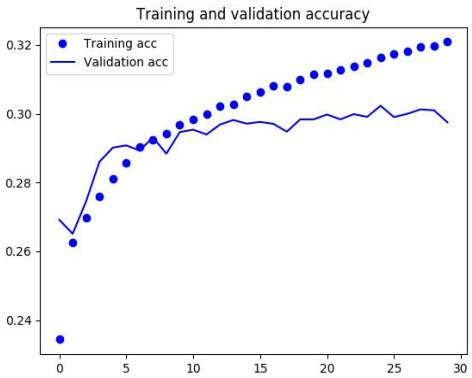
“Buildings” Pretrained Model

For the previous two figures, the results obtained by the most basic model, from the perspective of loss, the training loss is falling faster, but the decrease of the verification loss is not so large, which means that there is a great possibility that our model is overly complex. At the same time, the magnitude of the change in training acc and validation acc also reflects this problem. By preprocessing the data and adjusting the parameters for the entire model, our model ’s overfitting is reduced and the prediction accuracy of the model is hit 30%. Although the result did not reach the expected effect, the possible reason is that the data I got is only the shape of some building outlines. Using these shapes to classify them into specific types of buildings, the features it can learn for neural networks are very less, the second is that there is no obvious feature data between each category or even some overlaps and similarities, which is the reason for the relatively low accuracy rate.









Conclusions

The goal of our project is to develop a deep learning framework for classifying spatial polygons, based solely on their geometry, that is more flexible, light-weight, and accurate than existing frameworks.

Overall, our method proves to be more flexible than methods which require raster or vector data because it has the ability to classify shapes for which there are no raster or vector components and only pure shapes from maps, irrespective of geolocation.

Towards our second endpoint of creating a more light-weight method, it did not entirely succeed. Ultimately, the use of images instead of relative coordinate vectors is likely not lighter-weight as the images require more memory to store.

Towards our final endpoint, we were unsuccessful in training a higher-accuracy model than the 1D CNN. We believe this is largely due to the fact that we introduce noise by putting the shape onto a white background. Overall, this introduction of noise may make it more difficult to extract features. In addition, we spent a lot of time and assessing the viability of the data used in the paper for modeling and did not make it to a point where we could continue to tune the models further for higher accuracy. The final “Buildings” model listed above seemed to have the most promise. We believe that this model could be further tuned by using SGD with decay and running for a large number of epochs to further increase accuracy.

References

Fan H, Zipf A, Fu Q, Neis P (2014) Quality assessment for building footprints data on openstreetmap. International Journal of Geographical Information Science 28(4):700–719

U.S. Geological Survey (USGS) Gap Analysis Project (GAP), 2018, Protected Areas Database of the United States (PAD-US): U.S. Geological Survey data release, https://doi.org/10.5066/P955KPLE.

U.S. Geological Survey – Gap Analysis Project, 2018, U.S.Geological Survey – Gap Analysis Project Species Range Maps CONUS_2001: U.S. Geological Survey data release, https://doi.org/10.5066/F7Q81B3R

IUCN 2019. The IUCN Red List of Threatened Species. Version 2019-2. http://www.iucnredlist.org. Downloaded on 18 July 2019.

GreenInfo Network, 2019, Help System: PAD-US. http://www.protectedlands.net/about-this-help-system/

Keyes L, Winstanley AC (1999) Fourier descriptors as a general classification tool for topographic shapes. IPRCS, pp 193–203, URL http://eprints. maynoothuniversity.ie/66/

Veer, Rein van 't et al. “Deep Learning for Classification Tasks on Geospatial Vector Polygons.” ArXiv abs/1806.03857 (2018): n. pag.