Detecting Gerrymandering with Machine Learning Models

A Dissertation

Submitted to

The University of Liverpool

By

Andrew Frisby

ID: 201525902

Advisor: Aris Filos-Ratsikas

Second Marker: Alkmini Sgouritsa

September 23, 2022

Abstract

Detecting Gerrymandering with Machine Learning Models

By

Andrew Frisby

Gerrymandering in the U.S. is an issue that remains difficult to define and categorize due to its subjectivity, which has been highlighted in recent Supreme Court cases and redistrictings. This dissertation provides a proof of concept for machine learning models that detect and predict gerrymandering in a U.S. congressional district. Combining generated image data of districts with their numerical, demographic data mixed models with convolutional, dense, and concatenation layers are produced. Data is then labeled using a wide range of metrics and supplied to the models for training. The predictive output is binary for most models tested; however, a multiclass model is explored as well. Many parameters, metrics, and research questions were assessed and compared to achieve the best test accuracy. The highest accuracy achieved was 86%, which was an encouraging result for a foundational research project and proved that a machine learning model could be leveraged to detect, analyze, and quantify gerrymandering.

Student Declaration

I hereby certify that this dissertation constitutes my own product, that where the language of others is set forth, quotation marks so indicate, and that appropriate credit is given where I have used the language, ideas, expressions or writings of another.

I declare that the dissertation describes original work that has not previously been presented for the award of any other degree of any institution.

Signed,

Andrew Frisby

Acknowledgements

I would like to start by thanking my advisor, Aris Filos-Ratsikas, for his support, encouragement, and expertise. Without his probing questions and insightful comments this dissertation would have been impossible. Thank you for challenging me and always being available to help.

Next, I must thank my family, especially, Jill Rogers, Dan Frisby, and Samantha Riestenberg. When I was thousands of miles away, their love, care, and empathy got me through the hardest parts of this program and dissertation. I will never be able to thank them enough for what they have done, but I am so grateful for each of them. I love you all!

Table of Contents

List of Tables

List of Figures

Introduction

Gerrymandering, or the geographic manipulation of voters to sway election results, in modern democracies is a challenging issue to address and quantify. It occurs when policymakers draw voting districts, within a larger state or country, in such a way that a political party wins more proportional representation in the legislative body than their popular vote share indicates. For example, in a two-party system, a gerrymandered election would look similar to the results in Table 1. Party A received a minority of the popular vote, but still has majority control in the legislature. Party A achieves this gerrymandered result by a combination of "packing" party B's supporters into fewer districts and "cracking" concentrated areas of party B supporters across multiple districts, thus diluting their vote (Chandrasekaran, 2018). The practice is illegal and/or mitigated in most democracies via non-partisan drawing commissions.

Table 1

	Party A	**Party B**
Popular Vote Share	49%	51%
Legislative Share	55%	45%

Table 1: Example of Gerrymandered Election Results

However, if a country is unable to curb the effects of gerrymandering, the consequences are severe and threaten the foundations of democracy. As Samuel Issacharoff states in his Harvard Law Review paper, "Gerrymandering and Political Cartels," gerrymandering causes three central damages: harm to democracy through loss of democratic accountability, harm to the individual through weakening of an individual right (voting), and harm to specific minority groups via group-based discrimination (Issacharoff, 2002). As citizens notice and feel these effects of gerrymandering, they begin to lose faith in their election systems. Consequently, if this cynicism and these infringements become severe enough, democracy itself begins to erode and a country may face public unrest or even revolution.

Other authors, such as Nicholas Stephanopoulos, concur on the dangers of gerrymandering. After conducting a quantitative analysis on the impacts of gerrymandering, Stephanopoulos finds that "gerrymandering is both a deliberate and a democratically debilitating phenomenon" and it causes this debilitation more than any other variable considered (Stephanopoulos, 2018). Its threats are well documented, but curbing gerrymandering is still a difficult challenge for a democracy as policymakers are incentivized to gerrymander and it is often subtle.

Although technically illegal, gerrymandering still persists in the United States because the current methods of detecting and proving gerrymandering are undefined and difficult. The most common methods are statistical – developing metrics and indices to find outliers in a set of districting plans (Bangia *et al.*, 2017). These techniques have been presented in U.S. Supreme Court cases (Smith, 2019), but they were determined to be too subjective[1], and they stop short of sophisticated machine learning (ML) models. It was this dissertation's intention to take that step and develop a ML model to detect gerrymandering and lay the foundation for an adequate standard of evaluation. The dissertation was a research-based, proof of concept project that used both images

[1] See *Rucho v. Common Cause*, 2019. Also, more discussion on this ruling in Background section.

and numerical data to make the models. Given the mixed data, the models combined both convolutional neural networks (CNNs) and normal neural networks (NNs) to illustrate the feasibility of such models to be used in the detection and evaluation of gerrymandering.

The dissertation focused on data and examples from the United States of America, where gerrymandering is still prevalent and problematic. It specifically analyzed data associated with the drawing of federal congressional districts within individual states.

Aims and Objectives

The aims and objectives of this dissertation are listed below.

- (Achieved) Develop ML model to detect gerrymandering in a district and/or districting plan
  - (Achieved) Generate realistic districting plans and accompanying demographic data
  - (Achieved) Score and label generated plans
  - (Achieved) Implement effective method to combine numerical demographic data and district images in single ML model
  - (Achieved) Tune, train, and test model on labeled data
- (Partially Achieved) Discover most appropriate metrics and metric composites to label data used in model
  - (Achieved) Incorporate and develop metrics to label data
  - (Achieved) Compare results of models using different labeling metrics
- (Achieved) Offer a standard for evaluating gerrymandering with ML
- (Not Achieved) Illustrate power of gerrymandered districts on elections
  - (Not Achieved) Visualize and report election results of most gerrymandered district plans and compare them to non-gerrymandered plans

Background

There is a substantial amount of research on quantifying and analyzing gerrymandering. The most notable example being the work led by Duke University professors Jonathan Mattingly and Gregory Herschlag, “Redistricting: Drawing the Line,” which was used as a central source of evidence in a recent US Supreme Court case. In the paper, the authors use a Monte Carlo Markov Chain (MCMC) method to generate thousands of realistically drawn districts for North Carolina. They then implement several metrics and statistical tests to quantify the effect of gerrymandering across these district plans. In the end, they compare the results of the tests in simple box plots and violin charts to the actual districting plans used in the 2012 and 2016 elections (Bangia *et al.*, 2017). The results indicated that the 2012 and 2016 districting plans were gerrymandered, however, when these results were presented in the court case, *Rucho v. Common Cause*, the U.S. Supreme Court ruled that the issue of partisan gerrymandering was not justiciable by the federal courts due to a lack of a “’limited and precise standard’ for evaluating partisan gerrymandering” (*Rucho v. Common Cause*, 2019).

The paper also asserts that the value of constraint parameters (compactness, population deviation, etc.) and the choice of the initial plan do not impact end results after many iterations of the MCMC algorithm. This was tested in the dissertation as well.

The details and mathematics of the MCMC method are also discussed in the paper. The Markov chain uses the Metropolis-Hastings algorithm to "define a random walk" on the set of all possible redistrictings "which has [a probability distribution] as its unique, attracting stationary measure" (Bangia *et al.*, 2017). More information about the engine of the chain, the Metropolis-Hastings algorithm, can be found in C.P. Robert's paper, "Metropolis-Hastings Algorithm" (Robert, 2016). This MCMC method is utilized to generate the data for this dissertation and is discussed further in the Design section.

A similar technique, approach, and set of metrics were used to analyze the Wisconsin state assembly districts in the paper, "Evaluating Partisan Gerrymandering in Wisconsin" (Herschlag, Ravier and Mattingly, 2017). The authors find that the 2010 state districts are highly gerrymandered and skew quite favorably to the Republicans. The process to come to these results is virtually the exact same as in the Bangia *et al.* 2017 report, and one can find similar discussions in both. Although parallel, it is still important and useful to have another case study and proof of concept for the MCMC data generation.

The metrics used to define gerrymandering were of particular interest for this project, and Bangia *et al.* 2017 and Herschlag, Ravier and Mattingly, develop two different indices that they use to quantify the phenomenon. The papers use a "Gerrymandering Index" and a "Representative Index." In general terms, the Gerrymandering Index "measures the extent to which a particular redistricting has districts whose vote margins for each election deviate from what is expected," and the Representative Index "measures how anomalous the partisan composition of a redistricting is" (Herschlag, Ravier and Mattingly, 2017). The Gerrymandering Index is rather straightforward mathematically, it is "[t]he squareroot of the sum of the square deviations" from the expected vote totals (Bangia *et al.*, 2017). The Representative Index is slightly more conceptual, but still simple mathematically. It creates a proportion between the two least polarized districts that each party won (i.e., the districts with the smallest margins of victory for each party) to determine where these vote counts surpass the 50% threshold. It then adds that fraction to the total number of seats won by the party of interest. The fraction itself is a measure of how safe a party's victory was, and by adding this value to the number of seats won, the metric also includes information on the total number of seats won in the state (Bangia *et al.*, 2017).

The Bangia *et al.* 2017 paper also considers the efficiency gap score (EGS) as a potential third metric. The efficiency gap metric is just a proportion of how many wasted votes (i.e., votes acquired over the 50% winning threshold) each party has across the districts to the total votes, with larger values indicating more wasted votes. The authors found that this metric had high correlation (approximately, 0.75) to the Representative Index, and, therefore, concluded that it was redundant to include. Given this correlation and the simplicity of the efficiency gap metric, this dissertation utilized it extensively in the data labeling phase.

Another contributor to the intersection between mathematics and gerrymandering is Omer Lev. He has authored several papers on the topic and focuses primarily on how the parameters of a district or state (e.g., population density, district size, number of total districts) impact, nullify, and/or amplify gerrymandering. One of the more relevant papers he authored was "Big City vs. the Great Outdoors: Voter Distribution and How it Affects Gerrymandering." This paper analyzed the effect of the urban vs. rural population dynamic in gerrymandering and asserted that the preferred party of the rural voters has more gerrymandering power (Borodin *et al.*, 2018). The authors, in fact, developed a Mixed Integer Linear Program (MILP) to generate gerrymandered districts. The algorithm tries to maximize a party's representation above their proportional share. This is the only metric they use to detect gerrymandering. While interesting, and would have perhaps been useful for my purposes, they leverage IBM CPLEX, a proprietary software that would require too much time to become familiar with, to solve the MILP.

An additional paper that Omer Lev authored related to this subject is "Divide and Conquer: Using Geographic Manipulation to Win District-Based Elections." The authors, again, use a generated grid-state, but this time analyze just how effective gerrymandering can be for parties and the parameters that impact it. They confirm why gerrymandering is such a concern for democracies by providing two real world examples (2015 Israeli and British elections) and illustrate how different district parameters and compositions can lead to any of the major parties involved winning pluralities (Lewenberg, Lev and Rosenschein, 2017).

These papers were where I first encountered generated grid-states for testing purposes. These synthetic states, which, along with other assumptions, simplified the task and allowed the authors to make more broad conclusions about gerrymandering. If grid states were used with my models, they would generate a more generalized set of models that could be applied to other grids and/or states. Although the prospects of a generalized model were exciting, I wanted to remain as grounded in reality as I could for this dissertation. There are many assumptions that these authors make in their papers that ignore important US federal laws on drawing districts (e.g., district populations must be within a certain percentage of each other), which could lead to a loss of legitimacy and/or applicability.

The Python library utilized for the data generation in this project is GerryChain created by the Metric Geometry and Gerrymandering Group (MGGG). Their GitHub is full of resources and documentation for the library and other related libraries (*GerryChain*, 2022). Additionally, one of its creators, Daryl DeFord, has developed a user guide for the library (DeFord, 2019). This combined with the official documentation were crucial to the data generation and metric development stages of this project.

Additionally, through researching one of the included metrics, the Polsby-Popper metric, I discovered an entire set of metrics created around the compactness of a district. These metrics relate to the area and perimeter of a district and more information was found in Fisher, 2017. The specific metrics used in the dissertation are below:

- Schwartzberg
- Reock Score
- Polsby-Popper

A more detailed discussion on the metrics used in the dissertation can be found in the Design section.

Exploration into ML models that can handle mixed data found that it was more straightforward than originally thought. Adrian Rosebrock details just how to develop such models in his article, "Keras: Multiple Inputs and Mixed Data." The key to generating the model is to keep the mixed data inputs in separate models until they both have similar fully connected layers, then concatenate them, and move on to the output layers (Rosebrock, 2019). Figure 1 illustrates the general architecture of such models.

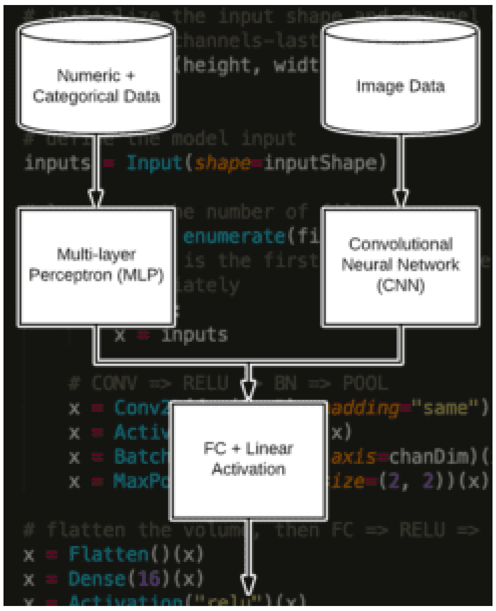

Numeric + Categorical Data
Image Data
Multi-layer Perceptron (MLP)
Convolutional Neural Network (CNN)
FC + Linear Activation

Figure 1: Mixed Data Model Architecture (Rosebrock, 2019)

This general architecture was utilized in the creation of the combined models for this dissertation. Further details on the model architecture can be found in the Design section of this paper.

Data Sources

The data utilized in this project was originally collected from the U.S. federal government, specifically the Census Bureau. It contains publicly available shapefiles and census data for each state and territory in the U.S. (*TIGER/Line Shapefiles*). The shapefiles are unique, proprietary file types that contain several files that have geospatial IDs and geodata that allow maps to be created. The census data contains demographic data on the population of interest, as well as geospatial IDs so one can identify where citizens live within a state.

However, to use this data with the data generation library, GerryChain, one needs to prorate the census and voting data. Fortunately, the developers of GerryChain, MGGG, have done this data preparation for all fifty states and the files can be found on their GitHub page. During this proration, the developers also add election results data from over ten (depending on the state) different statewide elections from 2010 to 2016 and several baseline district plans to use as the initial plans in the MCMC generation. Using these prorated data files, I then generated thousands of realistic district plans for the state of Pennsylvania using the MCMC process.

Design and Implementation

The design of this dissertation followed the original proposed cycle closely. This cycle can be seen in Figure 2 below. Each phase of the cycle incorporated several steps and, depending on the input images and metrics, had other unique design choices. The coding pipeline mirrored this development cycle as well. Each block had their own Python notebook(s) and generated the required outputs for the next stage.

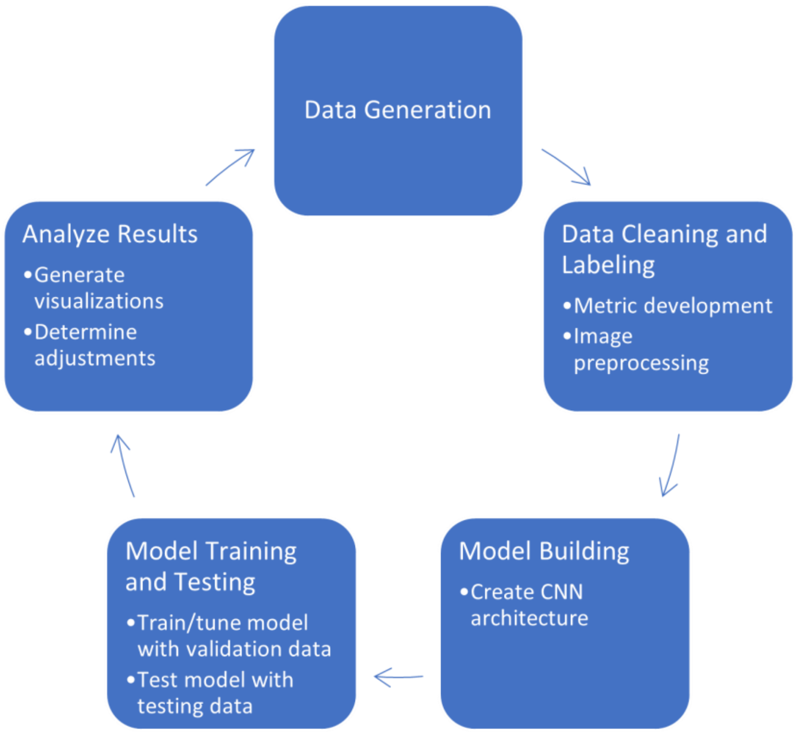


Figure 2: Dissertation Development Cycle

The dissertation examined both statewide districting plans and individual districts as input images. This required some separation in the coding pipeline(s) for the two input images. However, they were, overall, quite similar. The differences will be noted in the subsections below where relevant.

Data Generation

The data generation stage relied heavily on the GerryChain library to create, track, and save the realistic districts and accompanying demographic data. As described in the Background section, the library relies on a Monte Carlo Markov Chain (MCMC) to algorithmically flip groups of border voting districts to create new congressional districts. This process is visualized in Figure 3.

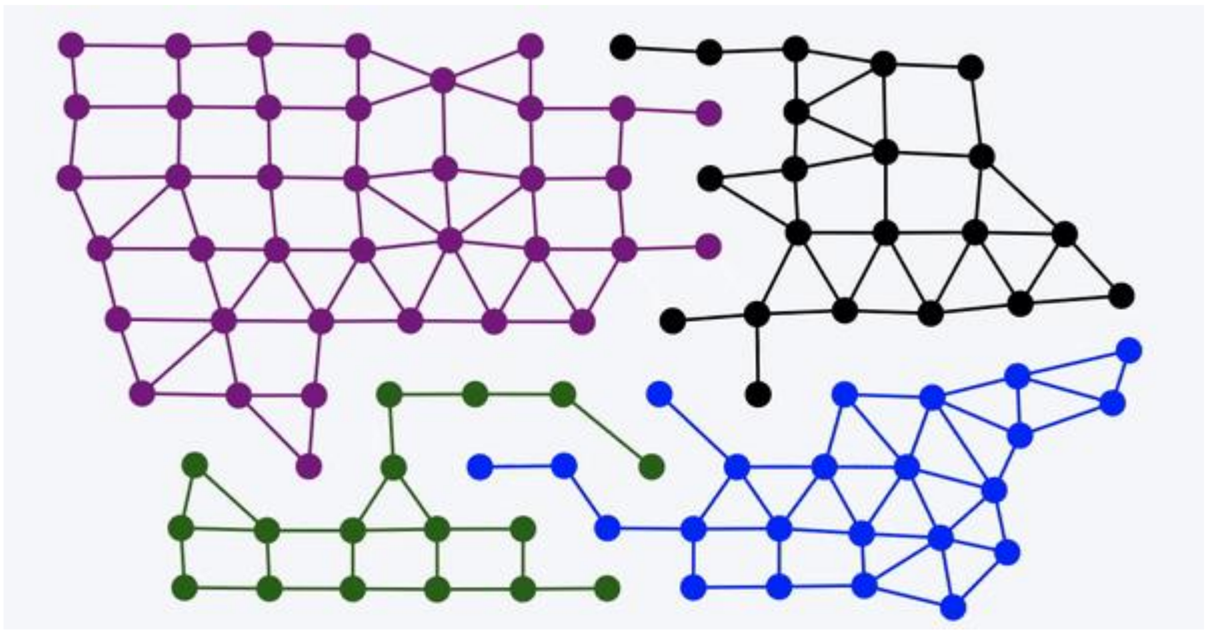
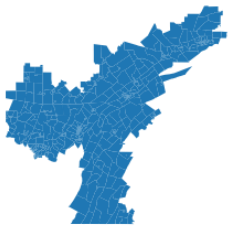


Figure 3: Animation of the MCMC Process (Bangia et al., 2015)

The nodes in the graphic represent individual voting districts and the colors represent the overall congressional districts within the state. Given a starting map, the algorithm will select a group of border nodes and flip them to the adjacent congressional district. It then verifies if the newly generated congressional districts are within the provided constraints that one sets when starting the MCMC. These constraints include population deviations, noncontiguous congressional districts, and district compactness.

Using this technique and library, I generated demographic data and images of both entire districting plans and individual districts for the state of Pennsylvania[2], examples of which can be found in Figure 4. Pennsylvania was selected because it has a relatively diverse and politically balanced electorate in that it has two large urban areas on either side of the state (Philadelphia in the East and Pittsburgh in the West) and large rural areas in between the cities. Additionally, the state is landlocked geographically which makes the code for the MCMC image generation simpler as one does not have to deal with islands and how to connect them to the network.

[2] The total number of observations varied depending on what I was testing and computational limits of my computer. For the district generation this typically fell between 5,000-13,5000 observations and for the statewide districting plans totals only reached 250-500 observations. This limitation is discussed further in Results and Evaluation section.



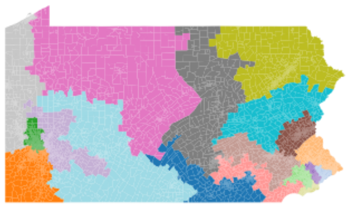


Figure 4: Generated District Image (left) and Statewide Image (right)

The MCMC function also has the ability to track important demographics, election results, and metrics throughout the Markov chain via, what it calls, updaters. The specific constraints and updaters that I used to generate a majority of my data can be found in Table 2 and Figure 5 below.

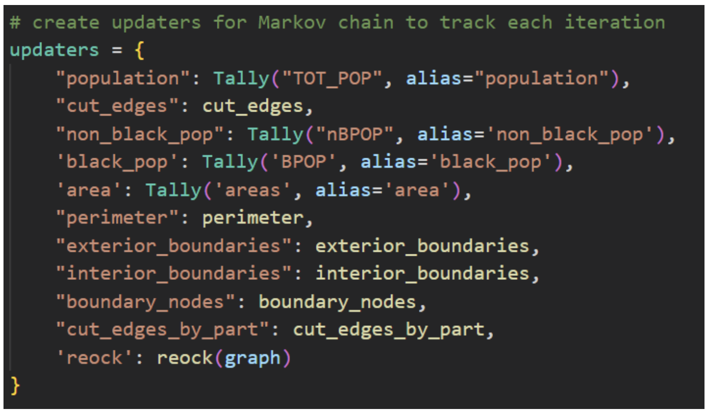
Table 2		
Type	**Name**	**Value(s)**
Constraint	Population	Within 2% of 704,000 (initial average)
Constraint	Compactness Upper Bound	4 times the number of total cut edges[3] as original plan
Initial Plan	Initial Districting Plan	2011 Congressional Map, 2018 Congressional Map, 538 Non-Partisan Map

Table 2: Data Generation Constraints

For the population constraint, federal congressional districts based on the 2010 census averaged around 700,000 people per district across the country. This constraint was placed to ensure that average was maintained when new districts were generated in the Markov chain. The other compactness constraint was used to safeguard the generated districts from being unrealistically "snake-like," i.e., thin and uncondensed. As for the initial districting plan, several were used to test the impact it had on the data generated. The first two were actual congressional maps used in Pennsylvania from 2011-2017 and 2018-2022, and the third was a proposed non-partisan map by statistics website FiveThirtyEight.[4]

[3] Cut edges are edges between two nodes in different areas of the district.

[4] The original 2011 map, based on the 2010 census, was challenged in the Pennsylvania Supreme Court and, ultimately, deemed gerrymandered. In response, the state Supreme Court redrew the districts in 2018 (Finnerty, 2018). These were in place until the new districting plan, based on the 2020 census, took effect this year.



```python
# create updaters for Markov chain to track each iteration
updaters = {
    "population": Tally("TOT_POP", alias="population"),
    "cut_edges": cut_edges,
    "non_black_pop": Tally("nBPOP", alias='non_black_pop'),
    'black_pop': Tally('BPOP', alias='black_pop'),
    'area': Tally('areas', alias='area'),
    "perimeter": perimeter,
    "exterior_boundaries": exterior_boundaries,
    "interior_boundaries": interior_boundaries,
    "boundary_nodes": boundary_nodes,
    "cut_edges_by_part": cut_edges_by_part,
    'reock': reock(graph)
}
```

Figure 5: GerryChain Updaters

The most important updaters to note were total population, total non-black population, total black population, and area. These four made up the features of the demographic data used in the final models. What I hoped to capture by including these features in the model are the racial undertones of gerrymandering and the urban vs. rural impact on gerrymandering. Race is one of the main demographics used to gerrymander as policymakers use it as a proxy for party preference and to disenfranchise African American voters, so it is important to capture that via features in my model. The urban vs. rural impact has been well documented in Borodin *et al.*, and area of a district will be used as a proxy for this phenomenon. The guiding intuition being that given similar total district populations (as required by U.S. federal law), larger districts with more area have a less dense population and are, therefore, more rural, and smaller districts are denser and more urban. Including these features in the models captured these effects.

Most of the other updaters were metrics or were used in metric calculations. The metrics utilized to eventually label the observations for the model belonged to two different categories – compactness and partisan metrics. The compactness metrics measure how compact a district is on a scale from $[0, 1]$, with more compact districts (values closer to 1) considered less gerrymandered and less compact districts (values closer to 0) considered more gerrymandered. These are metrics based on the geometry of the district and only apply to the district-based models that were created. The three compactness metrics that were used in this dissertation and a brief description can be found below. Their mathematical equations can be found in Appendix A.

- Polsby-Popper – "the ratio of the area of the district to the area of a circle whose circumference is equal to the perimeter of the district" (Fisher, 2017)
- Schwartzberg – "the ratio of the perimeter of the district to the circumference of a circle whose area is equal to the area of the district" (Fisher, 2017)
- Reock – "the ratio of the area of the district to the area of a minimum bounding circle that encloses the district's geometry" (Fisher, 2017)

The other category of metrics was partisan metrics. These metrics used actual statewide election vote counts to calculate their different scores. The reliance on actual statewide election results in most of these metrics meant that they were only viable for the statewide models that were created. However, the EGS was able to be applied on a district-level and used for both types of models. Additionally, most of these metrics have different scoring scales making any sort of average or combination of them unfeasible. The four partisan metrics used in this dissertation and a brief description can be found in the bullets below. Their mathematical equations can be found in Appendix B.

- Partisan Bias – "the number of districts with above-mean vote share [for a party] divided by the total number of districts, minus [0.5]" (*GerryChain*, 2022)
- EGS – "the difference between the parties' respective wasted votes, divided by the total number of votes cast" (Stephanopoulos and McGhee, 2014)
- Seats Secured – the total number of congressional seats won by a party (Democrat in my data) across the state
- Mean-Median Score (MMS) – simply the difference between the median and the mean vote share for a party across all congressional districts

As mentioned above, the partisan metrics rely on results from actual statewide elections. The nine elections that were used to calculate these metrics are listed below. The election data came included in the GerryChain data for the state of Pennsylvania, as described in the Data Sources section.

- 2010 Governor
- 2010 Senate
- 2012 Senate
- 2012 Attorney General
- 2012 President
- 2014 Governor
- 2016 Senate
- 2016 President
- 2016 Attorney General

Both sets of metrics were tracked, calculated, and output in this data generation phase.

While in each step of the MCMC, I was also able to alter the color of the generated districts. This was of important because I planned to test the impact of the color of the districts on the models I was building. I generated districts in blue, red, and green to assess this effect.

Data Cleaning and Labeling

Once the images, features, and metrics were generated and saved, I moved to the next stage of the development cycle – Data Cleaning and Labeling. In this stage I prepared the raw output to use in the ML models. To accomplish this, I loaded, converted, and merged the features and metrics into dataframes where each row corresponded to an individual observation. I also generated an image filename column in the new dataframes to act as a unique ID so that each row can be matched with its corresponding image. The resulting dataframes now had the four feature columns (area, total population, black population, and non-black population), and the respective metric columns. At this point in the Data Cleaning and Labeling phase, the process for the district-based data separated from the statewide-based data as they had different sets of metrics and required slightly different techniques. They will be discussed separately in the sections below.

District Model

Now that the dataframes were complete and in the correct shape, I began the process of examining the metrics, setting the thresholds, and labeling each observation. For the district-based models, I used the EGS metric and the three compactness metrics mentioned in the previous subsection. In order to set the labeling thresholds for both binary and multiclass classifications, I generated histograms and data summaries for the metrics. Examples of the histograms can be found in Figure 6. It is important to note that every set of generated data produced similar distributions of metrics, i.e., every histogram for these metrics had these shapes.

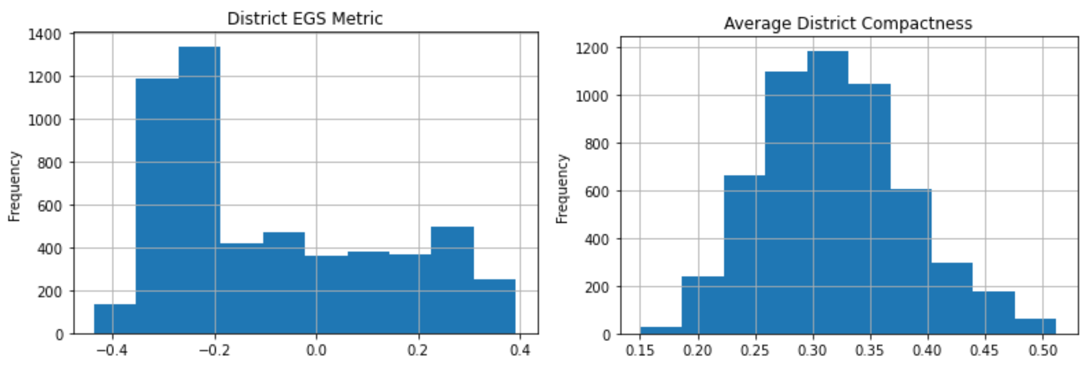


Figure 6: District Metric Histograms

The EGS metric has a range from $[-1, 1]$ with positive values indicating a district more gerrymandered for Democrats and negative values indicating a district more gerrymandered for Republicans. The mean for this metric typically fell near -0.08 and its standard deviation, around 0.2. Determining the binary True-False threshold to set for this two-sided metric was difficult, but ultimately, I decided to set it at one standard deviation away from the mean in both directions. For example, using the example values mentioned above, any districts with an EGS less than -0.28 or greater than 0.12 were labeled as gerrymandered. These thresholds were selected to ensure consistency across the different data generated and typically resulted in a 40/60 True-False split for the binary classification. The one standard deviation from the mean threshold also is consistent with EGS legal experts who assert that 1.5 standard deviations from the mean constitutes a district plan that "is gerrymandered to an unusual extent and…has an unusually large impact on the makeup of the House" (Stephanopoulos and McGhee, 2014). Considering that 1.5 standard deviations were extreme cases, I set the standard slightly lower to capture more subtle instances of gerrymandering and preserve class balance as well.

Similar thresholds were set for the multiclass classification categories. The three classes generated for these models were low, medium, and high levels of gerrymandering. The low class had a range from $[-0.5, 0.5]$, the medium class from $[\mu - 1\ std,\ -0.5] \cup [0.5,\ \mu + 1\ std]$, and the high class from $[-1,\ \mu - 1\ std] \cup [\mu + 1\ std,\ 1]$. Similar intuition as the binary thresholds was used to set these values.

Unfortunately, such standards were not as defined for the compactness metrics. As evidenced in Figure 6, the compactness metrics for all districts were less than 0.5. For a one-sided metric like the three compactness scores, this made setting such thresholds more difficult, subjective, and arbitrary. The low compactness scores for all districts indicated that every district in the data tended to be less compact and more gerrymandered. Several thresholds were tested, but ultimately, the threshold used was the mean of the averaged metrics. Any values below the mean were labeled true and any value above the mean were labeled false. Multiclass models were not generated using these metrics.

Once the observations were labeled with two different categories of metrics, I then train-test-split the data. The train-test-split was set at 80%-20% and a further 10% of the training data was set aside for the validation data. After the splits, the images were moved into the corresponding train/test/validation folders using the unique ID image filename column created earlier.

Statewide Model

The statewide data required a few alterations to the feature columns before moving to the labeling phase. Given that these were statewide districting plans, comprised of individual congressional districts, the four demographic features (area, total population, black population, and non-black population) could not be used in their current format. At a statewide level these features never changed – every districting plan had the exact same total area and population. To address this issue, I made two alterations: first, I calculated the maximum and minimum feature values for the individual districts within a statewide plan and replaced the original feature columns with the difference between the maximum and minimum. Second, I replaced the total population feature with district perimeter (and did the same transformation to max-min difference).

The rationale for using the max-min difference arose out of necessity as without it, the demographic data was unusable. The features in the statewide model now captured something slightly different – they represented the how polarized the features were in a districting plan. For example, if a black population max-min difference value was large for a given statewide plan, it could indicate that the plan packed minority voters into fewer districts, a telltale sign of gerrymandering. The same reasoning can be applied to both the area and perimeter features but regarding the urban vs. rural divide.

Once these features had been converted, I then moved on to the metrics. As described in the Data Generation subsection, I utilized the four partisan metrics to create the labels for the statewide observations (partisan bias, EGS, seats secured, and MMS). Similar to the district-based data, I generated histograms and summary statistics to get a sense for each metric and the distribution in the data. Figure 7 shows a histogram of the statewide EGS metric.

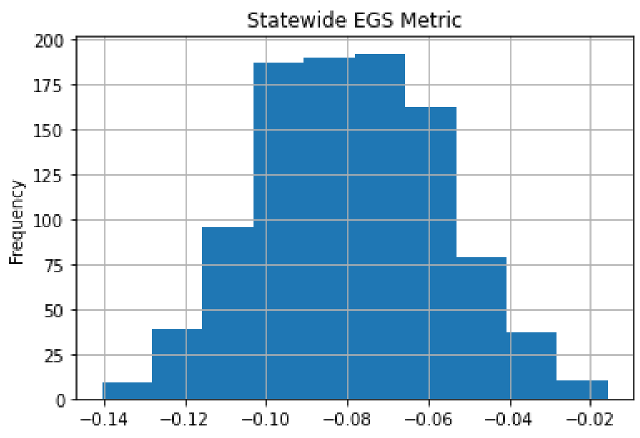


Figure 7: Statewide EGS Histogram

This EGS histogram is similar to the compactness metric histogram in Figure 6 and has the same problem. All its values are within the bounds that are considered gerrymandered – there is no distinct variation. Facing the same problem, I addressed it in a similar way, but slightly altered. I set the threshold at one standard deviation from the mean instead of at the mean. Values less than one standard deviation from the mean were considered gerrymandered and values greater than were labeled False. The histogram in Figure 7 depicts the same problems that were present in the other metrics as well. I applied the same thresholds for those metrics as well. I was well aware of this predicament and the problems it could cause during modeling, and my thoughts on its impact are discussed in the Results and Evaluation section of this paper.

Once I generated labels from each of the four partisan metrics, I developed a composite metric to establish an overall label. It simply took the four labels from the partisan metrics and if an observation had two or more True labels, then its overall label was True, otherwise, it was an instance of no gerrymandering (False class). With the final labels sorted, I did the same 80%-20%

train-test-split with the 10% validation as the district data and moved the corresponding images into their proper folders.

I also should note that, due to a larger class imbalance in the statewide data, I oversampled the statewide data to match the two classes. Specifically, I used a random oversampler to generate additional observations of the True class. This process was separated from the main statewide cleaning and labeling process and the oversampled data was train-test-split and saved separately in order to analyze the impact it had on the model.

Image Processing

To process the images two functions from the Keras library were used, ImageDataGenerator and FlowFromDirectory. Using these functions, I was able to zoom, set dimensions, normalize, and test other image modifications quickly and easily. The final image modifications used in the modeling stage, as well as the other parameters tested, can be seen in Table 3. The final images were zoomed in 30%, set to a sample mean of zero, and then normalized with the input's standard deviation. Many other parameters and combinations of parameters were tested, but ultimately, these were the best performing.

Table 3		
Parameter	**Testing Range**	**Final Value**
Zoom_range	[0, 1]	0.3
Samplewise_center	True, False	True
Samplewise_std_normalization	True, False	True
Width_shift_range	[0, 1]	N/A
Shear_range	[0, 1]	N/A
Brightness_range	[0, 2]	N/A

Table 3: Image Preprocessing Parameters

Once the images were preprocessed, I then saved the new images and generated a dataframe that matched the original image filename with the new, preprocessed image filename. This allowed me to merge these preprocessed image filenames onto the previously generated feature dataframes via the original filename column. Also, it is important to note that, although done in separate notebooks, the image processing was the same for both the statewide and district images.

Model Building

All models that were built had a similar architecture that emulated the Rosebrock combined model depicted in Figure 1. The specific architecture that was used can be found in Figure 8 below.[5]

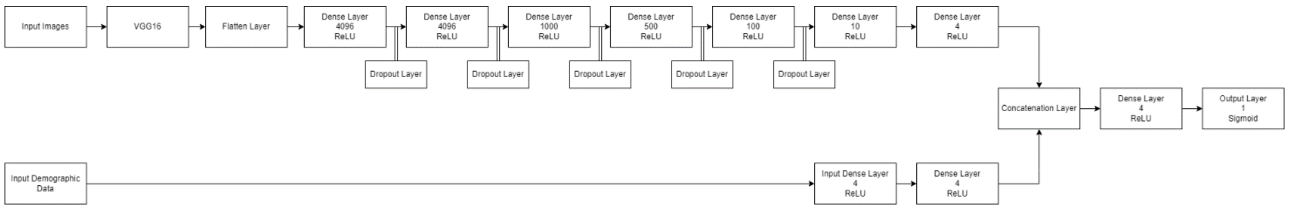


Figure 8: General Model Architecture

In the image/CNN pipeline across the top of the architecture, the images are first input into the VGG16 CNN. This network is an award-winning CNN that is popular in data science communities

[5] A larger, more readable version of the model architecture can be found in Appendix G. Additionally, the Python code to generate the models can be found in Appendix H.

for use in image recognition.[6] Its sophistication, accuracy, and overall simplicity make it an ideal network to use for transfer learning. I used it as such in my models. After going through the VGG16 network, the images are then flattened and moved through the fully-connected, dense layers with optional dropout layers in between. While the number of nodes, activation functions, and total layers may change from model to model, the general structure remains the same across the models. The image pipeline then moves its output to the concatenation layer where it meets the output of the basic neural network for the demographic data which only contained two dense layers. From there the network has one more combined, dense layer and then a final output layer. All models were compiled with the Adam optimizer, which is widely accepted and known for its efficiency, low memory requirements, and versatility (Kingma and Ba, 2017).

Given that I was creating, training, and testing many models I built Python functions to quickly generate these objects. This made the Model Training and Testing simpler and more streamlined. I also performed parameter searching and tuning in the next phase using the Keras-Tuner library. Functions to facilitate that process were built in this stage as well.

Model Training and Testing

This stage of the development cycle began with the importation of the generated dataframes and some final merging and data preparation. These final data cleaning steps included min-max normalizing the demographic data and converting the images to arrays.[7] From there, I was ready to initialize and compile the models using the functions built in the model building Python notebook. At first, I used an arbitrarily selected learning rate and epsilon of 0.001 and 0.000005 to get a quick sense of the models' accuracies. These were then tuned later if necessary. Before fitting the models, I created a vital early stopping callback with a patience of three that tracked the validation accuracy of the epochs. If the training process dropped off or stopped improving, this callback truncated the run early.

Once the callback was in place, the fitting and training of the models could begin. Input data was organized into batches of 32, which is the default size, and the models were trained on the training and validation data for a maximum of 20 epochs. After the model was fit, it was then evaluated on the test data. The accuracy on the test data was the central metric used to compare models with one another.

After getting a baseline accuracy score for the models, I ran parameter searches using the Keras-Tuner functions I built previously for the most promising models. The functions used the Hyperband search class to explore the parameter ranges and combinations.[8] Once the best parameters were found in the search, the training/fitting and testing process was repeated with those parameters. The parameters searched and their value ranges can be found in Appendix D.

Analyze Results

After the model had completed its evaluation on the test data, the results and model information were recorded in an Excel spreadsheet. This made for quick and easy comparisons between the models. I also created a Python function to generate a confusion matrix and

[6] For more details on the network see Appendix C or the seminal paper by Simonyan and Zisserman in 2015, "Very Deep Convolutional Networks for Large Scale Image Recognition."
[7] An additional adjustment for the multiclass data was done here as well. I had to label encode the three different classes (low, medium, high) to integers (0, 1, 2) for model compatibility.
[8] More information on the Hyperband search algorithm can be found in the paper "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," by Li *et al.*, 2018.

classification report for a given model. The classification report contained the precision, recall, and F1 scores. These generated reports are used in the Results and Evaluation section of this paper.

Once these reports were generated, overall analysis of the model was done. Comparing it to its baseline, previous iterations, and other similar models, led to tweaks and changes to future models. Once these future adjustments were decided upon, the development cycle would restart from the beginning.

Results and Evaluation

Given the myriad of models that were generated, trained, and tested,[9] I will focus on a select few of the most notable models that addressed key research questions in this section. Following the development plan for this project, the first iterations of the binary models focused on the district data and saw test accuracies in the mid-sixties, with the initial model reaching a 65.8% accuracy. This model had similar structure as the one in Figure 8, but there were no dropout layers and there were only four dense layers in the image pipeline. The model improved upon the baseline data accuracy of 58% and acted as a foundation to build on in future models. In the early stages of the modeling process, this was the target to beat.

The district data for these early models was comprised of 1,800 blue images and their accompanying demographic features. The sample size was intentionally kept small in these early stages while the code pipeline was being finalized and debugged. Further, the district data was labeled using the EGS metric, with the thresholds described in the Design and Implementation section. Compactness metrics were also used to label the same set of data, but accuracy for the models only reached 48%. Future compactness models never indicated any signs of promise and were, ultimately, shelved in favor of the better performing models. Further discussion on those models and their results can be found in Appendix E.

The early-stage statewide models also struggled for accuracy, only reaching 54% with 250-500 total observations. The data was labeled using the process described in the Design and Implementation section. Oversampling the data to balance the classes proved to be ineffective as the accuracy was also 54%. I attempted many alterations to both the data and model architecture from this stage for the statewide models, however, only minor improvements to accuracy were seen.[10] In response, I also shelved these models to focus on the district-based models. The details of those models and their results can be found in Appendix F.

The overall results of the early-stage models have been compiled in Table 4.

Table 4

Model	Labeling Metric	Accuracy
District	EGS	68%
District	Compactness	48%
Statewide	Composite	54%
Statewide Oversampled	Composite	54%

[9] Well over 40 models were tested during this process.

[10] The best statewide model I was able to produce achieved a test accuracy of 68%. Its results can be found in Appendix F.

Table 4: Initial Model Results

With a solid foundation and baseline accuracy, more data was generated and testing on the model architecture and image preprocessing began. The new data that was generated included batches of 4,500 blue districts, 9,000 green districts, and 13,500 red districts.[11] The image preprocessing adjustments and final values are described in the Design and Implementation section and in Table 3.

With more data and the images processed, the primary focus from this point was tuning the model architecture to reach the best accuracy for the model. This phase saw the introduction of additional dense layers in the CNN top layers and the dropout layers once overfitting was observed. The model began to take the shape of the one depicted in Figure 8. During this time, accuracies began to push into the eighties, culminating at 86% test accuracy.

This best performing model was a district-based model using the 13,500 red districts as the input data. Its confusion matrix and classification report can be seen in Figure 9 and Figure 10. The model was able to reach this performance before using Keras-Tuner to search and tune its parameters and architecture. However, it was discovered that the Keras-Tuner searches had extremely long run times and only resulted in as good or slightly worse accuracies. This was a surprising result, and it was determined in the late stages that any benefit that would be gained by allowing the searches to continue to examine the parameter space did not justify their long run times.

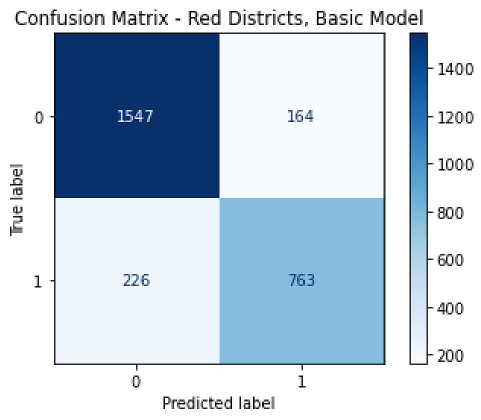


Figure 9: Best Model Confusion Matrix

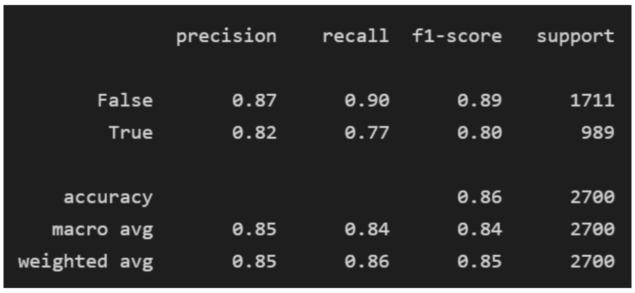

```
              precision    recall  f1-score   support

       False       0.87      0.90      0.89      1711
        True       0.82      0.77      0.80       989

    accuracy                           0.86      2700
   macro avg       0.85      0.84      0.84      2700
weighted avg       0.85      0.86      0.85      2700
```

Figure 10: Best Model Classification Report

[11] While generating these larger data sets, I discovered a memory leak in the GerryChain library that resulted in my computer freezing after a certain number of iterations of the Markov chain. This limited the total number of observations I could generate which, undoubtedly, hurt the statewide models disproportionately, and contributed to their reduction in priority.

Similar results were observed for the models trained with the other colored districts and it was determined that district color had no impact on the models. This proved that the CNN branch of the model was relying on the shape of the district more than the color – an indication that it was learning to detect the oddly-shaped trait that gerrymandered districts often have. Confusion matrices for models with the blue and green district images can be found below in Figure 11. Their test accuracies reached 80% and 82% respectively. It should be noted that the green district model performed best when tuned with Keras-Tuner, so its model architecture is slightly different.

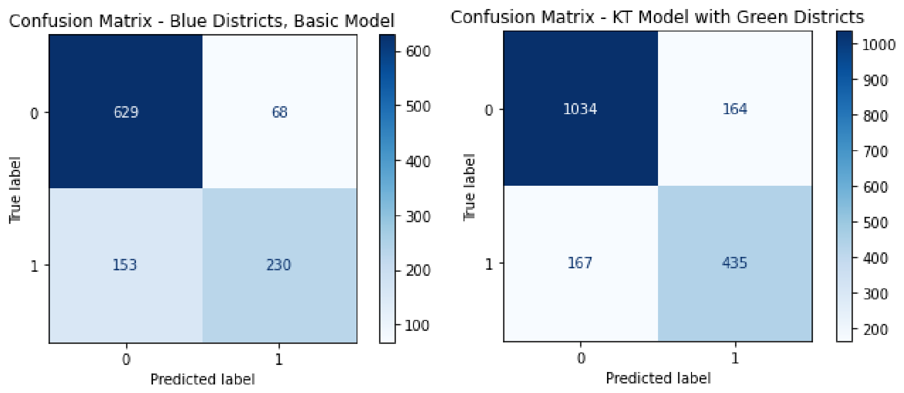
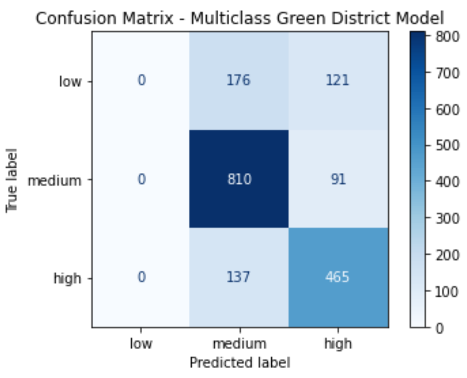


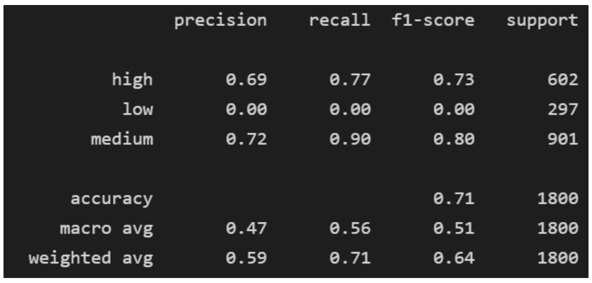
Figure 11: Confusion Matrices for Other Colored District Models

Overall, reaching accuracies from 80-86% with the models offered a compelling proof of concept for the use of ML models in gerrymandering detection and evaluation.

Another research question that was explored when generating additional datasets was the impact the starting districting plan had on the MCMC algorithm. As described in the Design and Implementation section and Table 2, three initial districting plans were used to start the Markov chains. According to the research on the MCMC algorithm noted in the Background section, the starting plan should have no impact on the final data generation results after many iterations (Bangia *et al.*, 2017). This proved to be true in my case as well – the data not only had similar metric histograms once labeled, but also resulted in comparable final testing accuracies.

I was also able to explore a multiclass model using the thresholds described in the Design and Implementation section. The three output classes were low, medium, and high levels of gerrymandering, and the model did not perform nearly as well as the binary classification models that were previously generated. It only achieved 71% testing accuracy, which is not awful, but when one examines the confusion matrix and classification report, glaring issues are present. Figure 12 depicts these visualizations for the multiclass model, and one can see that the model essentially behaved like a binary classification model. It only predicted two of the three classes and ignored the low class. This indicated to me that the data generated lends itself to a binary classification, not a multiclass one.





```
              precision    recall  f1-score   support

        high       0.69      0.77      0.73       602
         low       0.00      0.00      0.00       297
      medium       0.72      0.90      0.80       901

    accuracy                           0.71      1800
   macro avg       0.47      0.56      0.51      1800
weighted avg       0.59      0.71      0.64      1800
```

Figure 12: Confusion Matrix and Classification Report for Multiclass Model

Project Reflection

In this section I will offer a critical evaluation of the dissertation, including the strengths and weaknesses, and share some of the lessons learned throughout the process. The strengths and weaknesses are organized into the bullet points below.

Strengths

- Overall model performance was higher than expected and, ultimately, a successful proof of concept that laid a potential foundation for models such as these being used as the standard for detecting gerrymandering
- A wide range of research questions were explored and addressed
- The coding pipeline that was developed proved essential to the development cycle and streamlined the entire process
- The proposed development cycle and timeline were accurate and followed closely[12]
    - These kept the project organized and efficient
- The cloud storage that was used provided a crucial safety net with backups of data and Python notebooks, but also allowed for multitasking in the final stages of the modeling and testing phases once a second computer was secured

Weaknesses

- Data generation was hampered by a memory leak in the GerryChain library
    - Limited the total number of observations for both the district based and statewide models, resulting in the statewide models being disproportionately impacted
    - Not enough time to code a separate data generation solution
- Metrics were more difficult to handle than anticipated
    - Lack of variance in some metrics resulted in arbitrary thresholds and/or poor model performance when using those metrics
- Room for improvement with the demographic data
    - While the models performed well, additional demographic data could have been incorporated[13]
- Clearer file and save management from the onset would have saved time later in the project during image processing and large data generation runs

[12] The timeline of the project can be found in Appendix I.
[13] More thoughts on additional features discussed in Conclusion section.

- Run times of data generation, model training, and parameter searching were long
    - Data generation: 4-7 hours
    - Model training: 1.5-3 hours
    - Parameter searching: 17+ hours, rarely finished a complete search
    - Resulted in strict prioritization of models and aims/objectives

Lessons Learned

There were many learning points over the course of this dissertation. One of the most essential was learning to prioritize and work within a tight timeline. I was ambitious with my aims, objectives, and research questions from the onset, and during this process I was forced to sideline some areas of exploration that I was interested in. Deciding what to shelve and what to keep in focus, was difficult, but an excellent learning experience in prioritization in a large-scale project. This lesson was reinforced when deciding what information to highlight in the final presentation. With only 15 minutes to present and months of work to summarize, I had to rely on these prioritization skills once again to determine what to convey in the brief amount of time I had.

There were also several more technical lessons learned during the dissertation. One crucial learning point was the creation of a coding pipeline of separate Python notebooks that fed into one another. Separating each major development step into one or multiple notebooks allowed for compartmentalization of code and quick testing of edits without having to run entire files with unrelated code. If a function was needed in a separate notebook, only a simple import was required. Given the long run times of some processes, this organization and efficiency proved vital.

Related to the long run times, it was discovered that working with image data is computationally expensive. They require more resources at every stage in the modeling process than numerical data and this was reflected in the extensive run times I encountered. This was an important lesson to learn due to the rise in popularity of image classification tasks in data science. In future work with image data, I will ensure that additional development time is secured for the longer processing times.

Furthermore, my experience with and knowledge of ML models, specifically Keras models, has improved through this dissertation. The extensive work I did to combine CNNs and neural networks will undoubtedly prove beneficial in the future, and my familiarity with such models gives me great confidence to utilize them in related projects.

The final lesson learned that I will highlight is related to the difficulty I had labeling my generated data with the gerrymandering metrics. The many metrics around gerrymandering are relatively subjective and the thresholds within them even more so. While I have always been careful to note and be aware of bias in any data science work, this was to a greater extent and a focal point of the work, which made it especially challenging. Although the subjectivity was frustrating at times, my experience working with partisan data was, ultimately, helpful and insightful. The care required to ensure your work is as credible and reliable as possible is an important lesson that I took away from this dissertation.

Professional Issues

The British Computer Society's (BCS) code of conduct has four central pillars: public interest, professional competence and integrity, duty to relevant authority, and duty to the profession. My dissertation itself and the work I did to complete it are all in line with those four pillars. Regarding the public interest pillar, I initially took up this project because gerrymandering actively harms the public's interest by disenfranchising voters and weakening democracy. My work intends to detect and, ultimately, prevent gerrymandering from occurring, which is in the public's interest. Related to the second pillar, I have been clear in this dissertation what areas my analyses explore and how I conducted the work. I did not stray into areas outside of my expertise. As for the final two pillars, I always maintained a strict sense of professionalism with my advisor and second reader during this process and take full responsibility for the work I have completed.

Conclusion

Future Work

I was ambitious with what I wanted to examine in this dissertation, and that resulted in some research questions being sidelined when prioritization was required. Future work could explore such areas, which include adding more demographic data to the NN-branch of the model – especially voter registration data. When a U.S. citizen registers to vote in most states, they often register with a party as well (Republican, Democrat, etc.), this data would be useful in my model as another feature or two with the total number of registered Republicans/Democrats in the district. Additionally, there may be other demographics out there that could be incorporated into the model and examining feature importance metrics within the model (e.g., Shapley values[14]) may inform where to look for those additional features.

It was discovered early in the dissertation that most of the bias in the project would lie within the gerrymandering metrics and how the data was labeled. While extensive research and time was spent with these metrics, I believe there is an entire separate dissertation's worth of work that could be done with these metrics and how to best use them. This could be another area of future work to benefit the precision of evaluating and quantifying gerrymandering. Another option regarding the labeling of the data is to explore possibilities for unsupervised learning. If one could just eliminate a step that has bias altogether and have a ML algorithm find its own labels in the data, that may be an effective course of action.

Final Thoughts

Overall, reaching 86% accuracy for the model was an exciting result as I was unsure what to expect at the onset of this project. I believe that with an accuracy that high I was able to prove that the concept of a predictive, ML model can detect gerrymandering in individual districts. The models are capable of learning the differences in districts, but they are dependent on the labeling metrics – that is the crux of the issue and where the bias lies. However, if a state's policymakers were to at least come together and agree upon a metric or set of metrics to quantify gerrymandering, and they ensured that their selection was based in statistics and in line with the established literature, then a model such as this one could be used as a standard for both detecting and measuring gerrymandering.

A reasonable question to ask in response to this conclusion is if a state agrees upon a metric that defines gerrymandering, why take it a step further and build a model? Why not just use the metric threshold(s) that were agreed upon to determine if a district is gerrymandered? However,

[14] Shapley values are based in game theory and are used to generate feature importance scores for models. More information about them and the Python library to generate them can be found in Lundberg and Lee, 2017.

that is what was done in the past and it was ultimately, dismissed by the U.S. Supreme Court as too imprecise. A model, such as the one I built, that incorporates those metrics, district images, and district demographics makes a more compelling standard for gerrymandering detection and classification. If continued to be built upon, these models can prevent the dangers of gerrymandering in the U.S. and provide an unbiased, precise tool for policymakers to use.

Bibliography

Bangia, S. *et al.* (2015) *Quantifying Gerrymandering : Data+ 2015 @ Duke*. Available at: https://services.math.duke.edu/projects/gerrymandering/index-dataPlus.html (Accessed: 8 July 2022).

Bangia, S. *et al.* (2017) ‘Redistricting: Drawing the Line’. arXiv. Available at: http://arxiv.org/abs/1704.03360 (Accessed: 29 June 2022).

Borodin, A. *et al.* (2018) ‘Big City vs. the Great Outdoors: Voter Distribution and How It Affects Gerrymandering’, in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*. *Twenty-Seventh International Joint Conference on Artificial Intelligence {IJCAI-18}*, Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, pp. 98–104. Available at: https://doi.org/10.24963/ijcai.2018/14.

DeFord, D. (2019) ‘How to build districting ensembles: A guide to GerryChain’.

Finnerty, J. (2018) *Supreme Court will get to pick new map for state’s congressional districts*, *Meadville Tribune*. Available at: https://www.meadvilletribune.com/news/local_news/supreme-court-will-get-to-pick-new-map-for-states-congressional-districts/article_ce46acd2-1edc-5987-86c4-ffc1f1c097aa.html (Accessed: 7 September 2022).

Fisher, Z. (2017) *Measuring Compactness*. Available at: https://fisherzachary.github.io/public/r-output.html (Accessed: 29 June 2022).

‘GerryChain’ (2022). Metric Geometry and Gerrymandering Group. Available at: https://github.com/mggg/GerryChain (Accessed: 29 June 2022).

ul Hassan, M. (2018) ‘VGG16 - Convolutional Network for Classification and Detection’, 20 November. Available at: https://neurohive.io/en/popular-networks/vgg16/ (Accessed: 30 June 2022).

Herschlag, G., Ravier, R. and Mattingly, J.C. (2017) ‘Evaluating Partisan Gerrymandering in Wisconsin’. arXiv. Available at: http://arxiv.org/abs/1709.01596 (Accessed: 29 June 2022).

Issacharoff, S. (2002) ‘Gerrymandering and Political Cartels’, *Harvard Law Review*, 116(2), p. 593. Available at: https://doi.org/10.2307/1342611.

Kingma, D.P. and Ba, J. (2017) ‘Adam: A Method for Stochastic Optimization’. arXiv. Available at: http://arxiv.org/abs/1412.6980 (Accessed: 14 September 2022).

Lewenberg, Y., Lev, O. and Rosenschein, J.S. (2017) ‘Divide and Conquer: Using Geographic Manipulation to Win District-Based Elections’.

Li, L. *et al.* (2018) ‘Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization’.

Lundberg, S.M. and Lee, S.-I. (2017) ‘A Unified Approach to Interpreting Model Predictions’, in *Advances in Neural Information Processing Systems*. Curran Associates, Inc. Available at: https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html (Accessed: 14 September 2022).

Metric Geometry and Gerrymandering Group (no date). Available at: https://github.com/mggg (Accessed: 29 June 2022).

MGGG States (no date) *GitHub*. Available at: https://github.com/mggg-states (Accessed: 29 June 2022).

Redistricting Report Card Methodology (2021). Available at: https://gerrymander.princeton.edu/redistricting-report-card-methodology (Accessed: 29 June 2022).

Robert, C.P. (2016) ‘The Metropolis-Hastings algorithm’. arXiv. Available at: http://arxiv.org/abs/1504.01896 (Accessed: 29 June 2022).

Rosebrock, A. (2019) ‘Keras: Multiple Inputs and Mixed Data’, *PyImageSearch*, 4 February. Available at: https://www.pyimagesearch.com/2019/02/04/keras-multiple-inputs-and-mixed-data/ (Accessed: 18 July 2022).

Rucho v. Common Cause (2019) *Oyez*. Available at: https://www.oyez.org/cases/2018/18-422 (Accessed: 6 July 2022).

Simonyan, K. and Zisserman, A. (2015) ‘Very Deep Convolutional Networks for Large-Scale Image Recognition’. arXiv. Available at: http://arxiv.org/abs/1409.1556 (Accessed: 30 June 2022).

sklearn.model_selection.GridSearchCV (no date) *scikit-learn*. Available at: https://scikit-learn/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (Accessed: 1 July 2022).

sklearn.model_selection.RandomizedSearchCV (no date) *scikit-learn*. Available at: https://scikit-learn/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html (Accessed: 1 July 2022).

Smith, R. (2019) ‘Duke Mathematics Has Its Day in Court’, 25 March. Available at: https://today.duke.edu/2019/03/duke-mathematics-has-its-day-court (Accessed: 6 July 2022).

Stephanopoulos, N.O. (2018) ‘THE CAUSES AND CONSEQUENCES OF GERRYMANDERING’, *William and Mary Law Review*, 59(5), pp. 2115–2159.

Stephanopoulos, N.O. and McGhee, E.M. (2014) ‘Partisan Gerrymandering and the Efficiency Gap’, *The University of Chicago Law Review*.

TIGER/Line Shapefiles (no date) *Census.gov*. Available at: https://www.census.gov/geographies/mapping-files/time-series/geo/tiger-line-file.html (Accessed: 6 June 2022).

Ye, A. (2020) *Turning Non-Image Data into Images for Classification is Surprisingly Effective*, *Medium*. Available at: https://towardsdatascience.com/turning-non-image-data-into-images-for-classification-is-surprisingly-effective-70ce82cfee27 (Accessed: 29 June 2022).

Appendices

Appendix A

Polsby-Popper mathematical equation:

$$PP = 4\pi \times \frac{Area}{Perimeter^2}$$

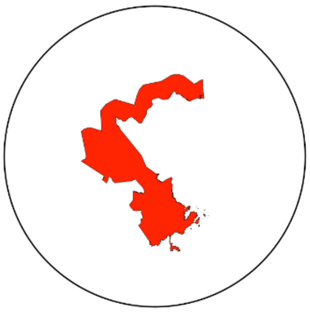


Figure 13: Visualization of the Polsby-Popper Metric (Fisher, 2017)

Schwartzberg mathematical equation:

$$S = \frac{1}{Perimeter/Circumference}$$

$$Circumference = 2\pi\sqrt{Area/\pi}$$

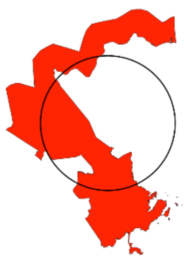


Figure 14: Visualization of the Schwartzberg Metric (Fisher, 2017)

Reock mathematical equation:

$$R = \frac{Area_{District}}{Area_{Minimum\ Bounding\ Circle}}$$

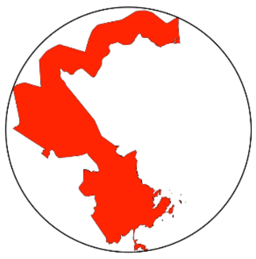


Figure 15: Visualization of the Reock Metric (Fisher, 2017)

Appendix B

Partisan Bias mathematical equation:

$$\frac{Number\ of\ districts\ with\ above\ mean\ vote\ share}{Total\ Districts} - 0.5$$

Efficiency Gap Score mathematical equation:

$$\frac{Wasted\ Votes\ for\ Party\ 2 - Wasted\ Votes\ for\ Party\ 1}{Total\ Votes}$$

Seats Secured mathematical equation:

$$\frac{Total\ Congressional\ Seats\ Won\ Across\ State}{Total\ Congressional\ Seats\ Available}$$

Mean-Median mathematical equation:

$$Median_{Vote\ \%\ across\ all\ districts} - Mean_{Vote\ \%\ across\ all\ districts}$$

Appendix C

The architecture of the VGG16 network can be found in Figure 16 below.

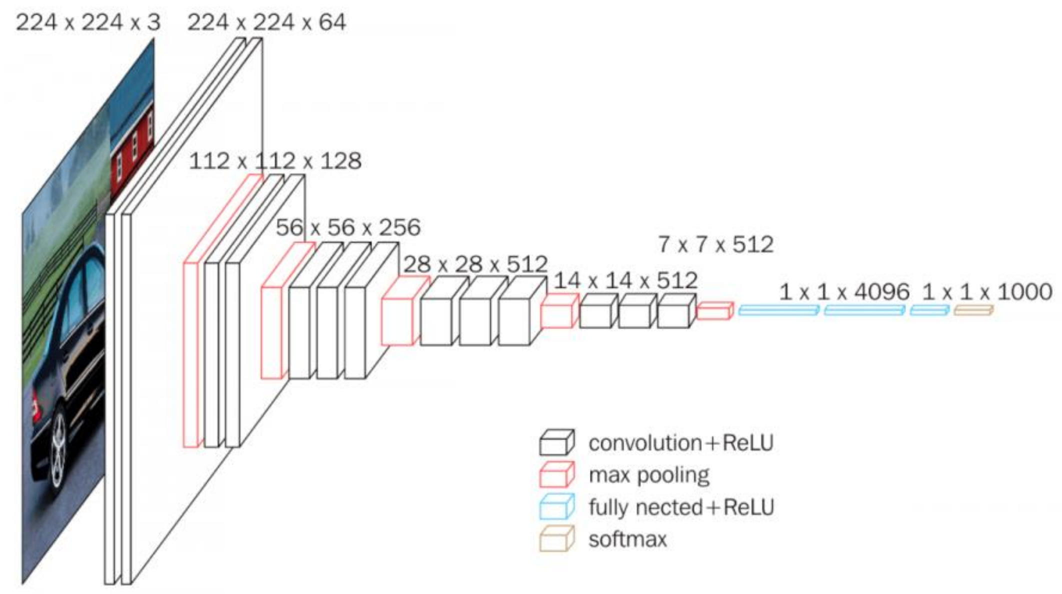


Figure 16: General Architecture of the VGG16 Network (ul hassan, 2018)

Appendix D

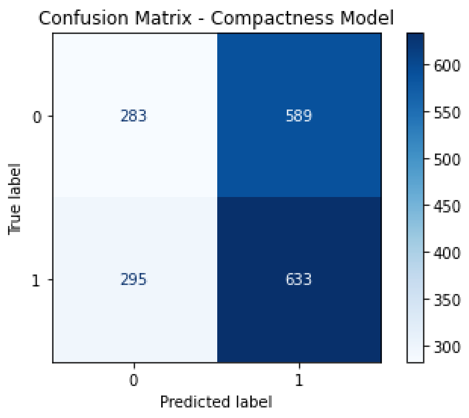
The parameters searched and their value ranges can be found in the table below. The CNN branch of the model had two sets of layers – high-node layers and low-node layers. These were separated at first to ensure there was a gradual reduction in nodes as they reached the final output layers. In future iterations of the parameter tuning searches, these node selections were combined with smaller value ranges.

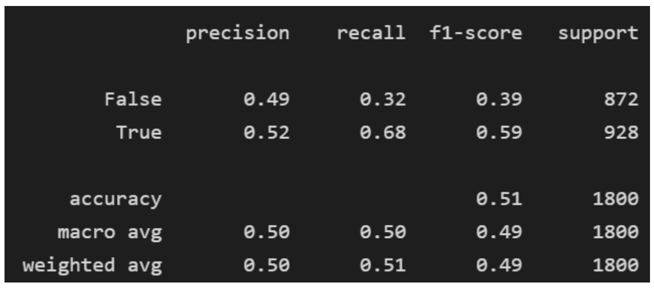
Table 5	
Parameter	**Search Values/Range**
Neural Network (NN) Input Activation Function	[ReLU, linear, ELU, GELU, SELU]
NN Hidden Layers	1-5
NN Hidden Layer Activation Functions	[ReLU, linear, ELU, GELU, SELU]
NN Hidden Layer Nodes	3-36
CNN High-Node Layers	1-3
CNN High-Node Layer Activation Functions	[ReLU, linear, ELU, GELU, SELU]
CNN High-Node Layer Nodes	1000-5000
CNN Low-Node Layers	1-3
CNN Low-Node Layer Activation Functions	[ReLU, linear, ELU, GELU, SELU]
CNN Low-Node Layer Nodes	10-500
Learning Rate	0.00001-0.1, log sampling
Epsilon	0.000000001-0.00001, log sampling

Table 5: Parameters Searched with Keras-Tuner and Their Value Ranges

Appendix E

As described in the Results and Evaluation section, the compactness models never reached notable levels of accuracy, with the highest being 51%. This is most likely due to the difficulties encountered with the metrics' lack of variance in the data that resulted in an arbitrary threshold for the labels, which was described in the Design and Implementation section. The confusion matrix and classification report for the best performing compactness model can be found in Figure 17.





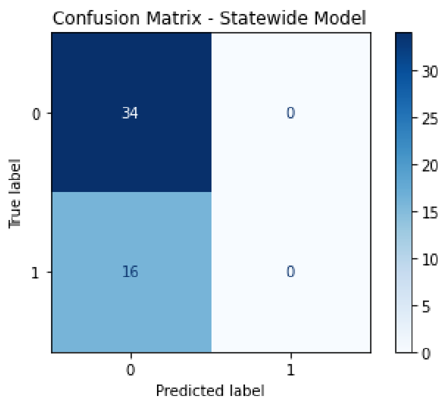
```
              precision    recall  f1-score   support

       False       0.49      0.32      0.39       872
        True       0.52      0.68      0.59       928

    accuracy                           0.51      1800
   macro avg       0.50      0.50      0.49      1800
weighted avg       0.50      0.51      0.49      1800
```

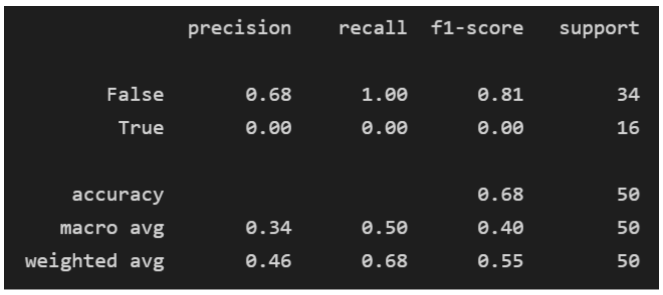
Figure 17: Confusion Matrix and Classification Report for Compactness Model

The disappointing accuracy led me to believe that perhaps the metrics, since they measure the geometry of a district, are better suited as demographic features in the NN-branch of the model. More research would be needed on the compactness metrics in general to determine if this would be an acceptable course of action.

Appendix F

As noted in the Results and Evaluation section, the statewide models were eventually shelved to focus on the more promising district-based models. The highest accuracy they achieved was 68%, but this is misleading. The confusion matrix and classification report for that model can be found in Figure 18.





```
              precision    recall  f1-score   support

       False       0.68      1.00      0.81        34
        True       0.00      0.00      0.00        16

    accuracy                           0.68        50
   macro avg       0.34      0.50      0.40        50
weighted avg       0.46      0.68      0.55        50
```

Figure 18: Confusion Matrix and Classification Report for Statewide Model

It is apparent that the model was not making any predictions, but just selecting the False label for each observation. The 68% accuracy was misleading because it was just the baseline of the data. This illustrates the challenges the statewide models had in both their labeling and their lack of data. The labeling troubles were described in the Design and Implementation section but were similar to the compactness metrics' issues. The lack of data stemmed from the memory leak in the GerryChain module that caused my computer to crash after 500-1000 iterations of the Markov Chain. This severely limited the number of statewide observations that could be generated and hampered these models.

Appendix G

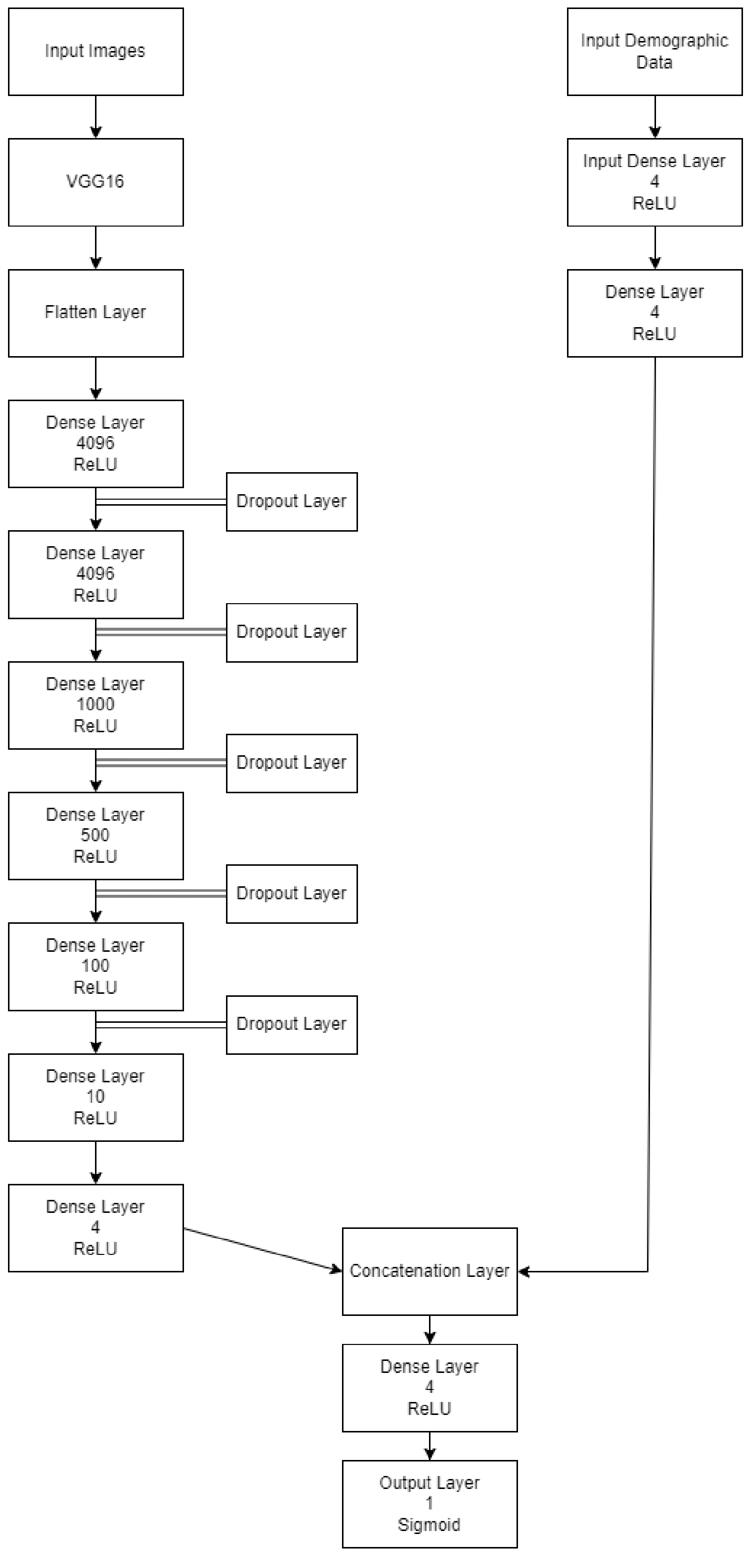
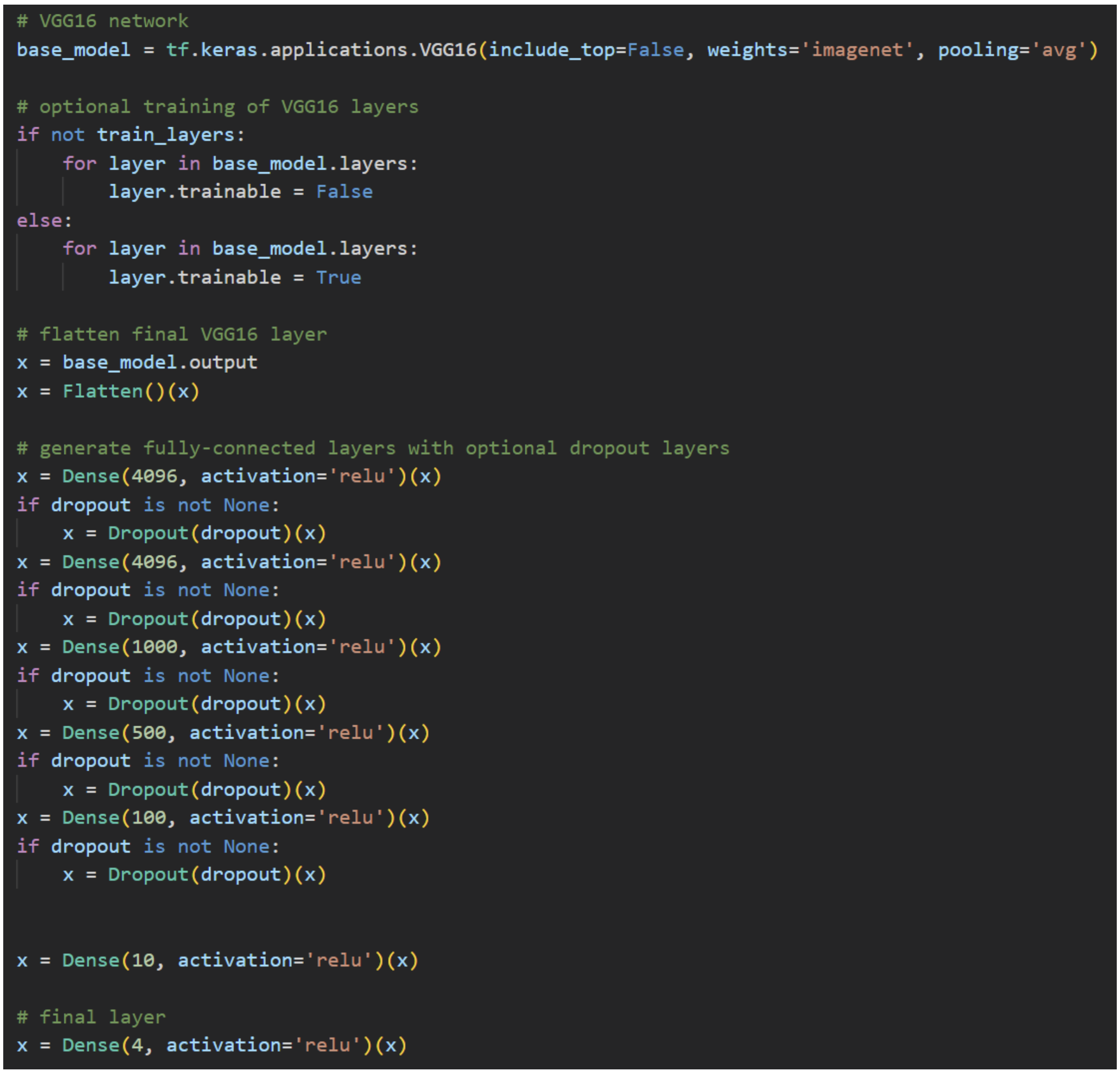


Figure 19: Combined Model Architecture Displayed Vertically

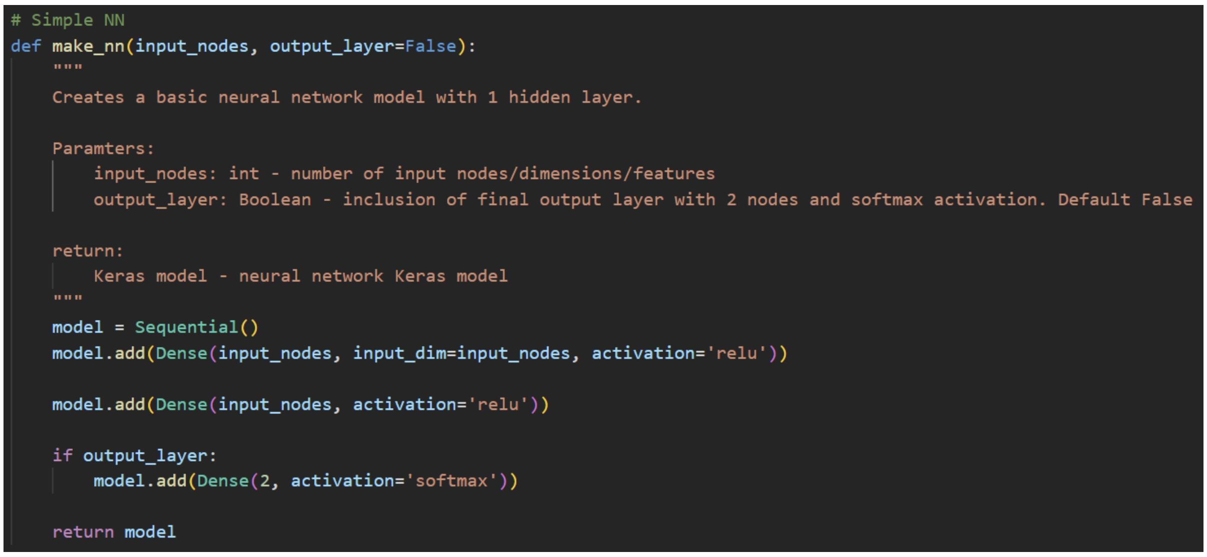
Appendix H

The Python code to generate the models can be found in the figures below. Figure 20 depicts the code to generate the CNN branch of the model. Figure 21 generates the basic NN for the demographic data, and Figure 22 shows the code to combine the two branches.



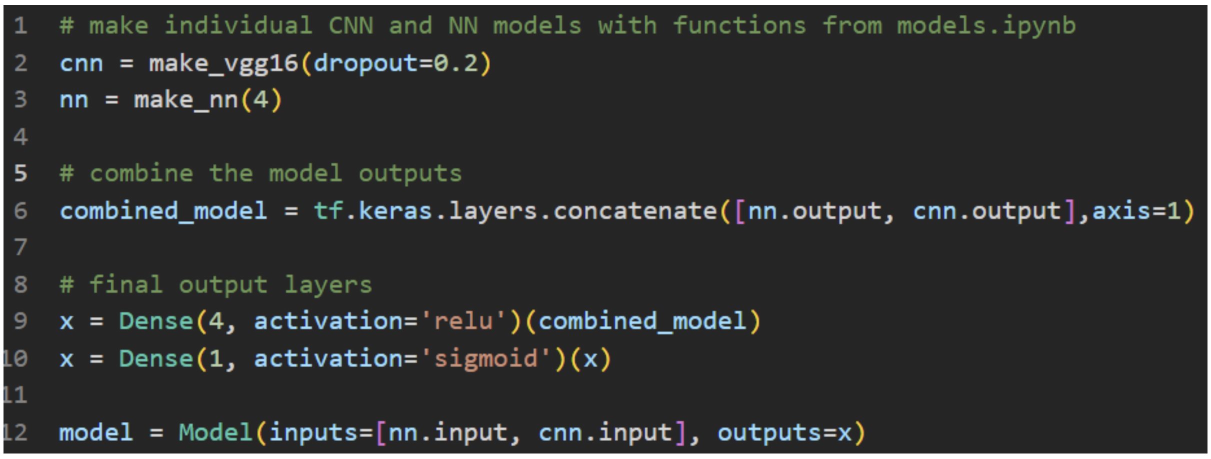
```python
# VGG16 network
base_model = tf.keras.applications.VGG16(include_top=False, weights='imagenet', pooling='avg')

# optional training of VGG16 layers
if not train_layers:
    for layer in base_model.layers:
        layer.trainable = False
else:
    for layer in base_model.layers:
        layer.trainable = True

# flatten final VGG16 layer
x = base_model.output
x = Flatten()(x)

# generate fully-connected layers with optional dropout layers
x = Dense(4096, activation='relu')(x)
if dropout is not None:
    x = Dropout(dropout)(x)
x = Dense(4096, activation='relu')(x)
if dropout is not None:
    x = Dropout(dropout)(x)
x = Dense(1000, activation='relu')(x)
if dropout is not None:
    x = Dropout(dropout)(x)
x = Dense(500, activation='relu')(x)
if dropout is not None:
    x = Dropout(dropout)(x)
x = Dense(100, activation='relu')(x)
if dropout is not None:
    x = Dropout(dropout)(x)


x = Dense(10, activation='relu')(x)

# final layer
x = Dense(4, activation='relu')(x)
```

Figure 20: Python Code to Generate CNN



```python
# Simple NN
def make_nn(input_nodes, output_layer=False):
    """
    Creates a basic neural network model with 1 hidden layer.

    Paramters:
        input_nodes: int - number of input nodes/dimensions/features
        output_layer: Boolean - inclusion of final output layer with 2 nodes and softmax activation. Default False

    return:
        Keras model - neural network Keras model
    """
    model = Sequential()
    model.add(Dense(input_nodes, input_dim=input_nodes, activation='relu'))

    model.add(Dense(input_nodes, activation='relu'))

    if output_layer:
        model.add(Dense(2, activation='softmax'))

    return model
```

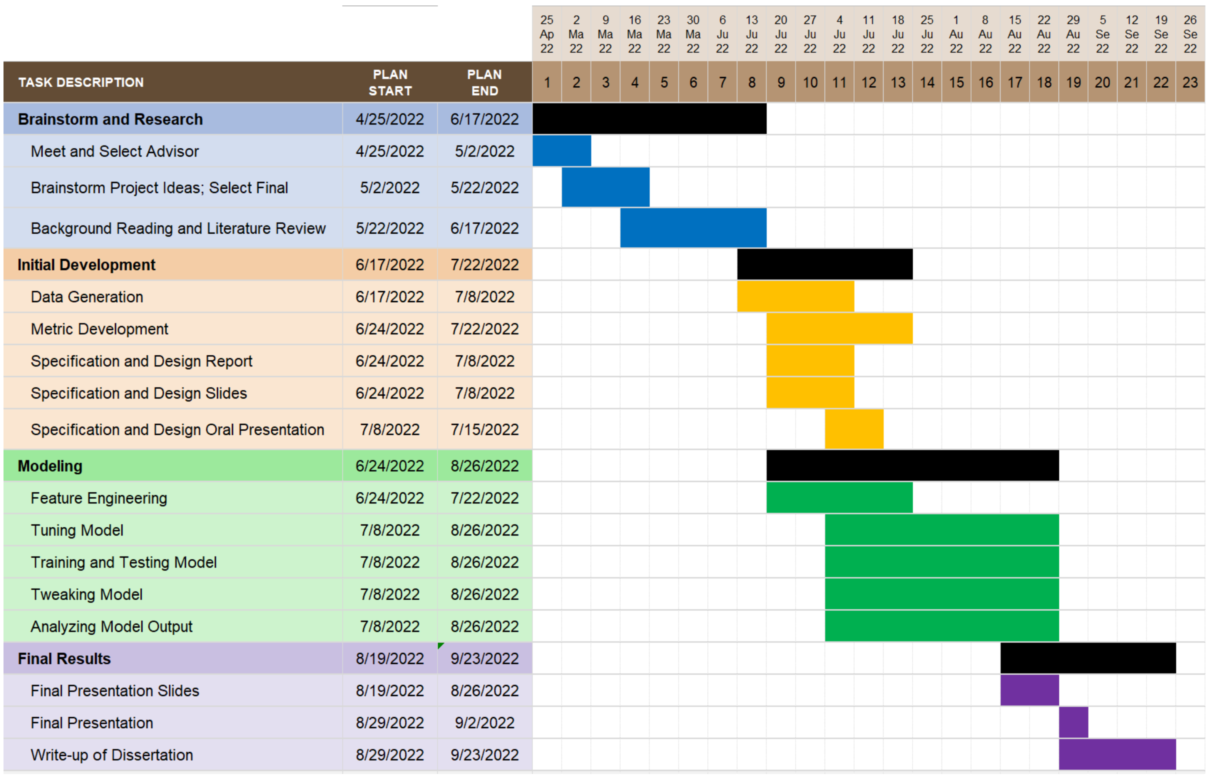
Figure 21: Python Code to Generate Basic NN



```python
# make individual CNN and NN models with functions from models.ipynb
cnn = make_vgg16(dropout=0.2)
nn = make_nn(4)

# combine the model outputs
combined_model = tf.keras.layers.concatenate([nn.output, cnn.output],axis=1)

# final output layers
x = Dense(4, activation='relu')(combined_model)
x = Dense(1, activation='sigmoid')(x)

model = Model(inputs=[nn.input, cnn.input], outputs=x)
```

Figure 22: Python Code to Generate Combined Model (CNN + NN)

Appendix I



			25 Ap 22	2 Ma 22	9 Ma 22	16 Ma 22	23 Ma 22	30 Ma 22	6 Ju 22	13 Ju 22	20 Ju 22	27 Ju 22	4 Ju 22	11 Ju 22	18 Ju 22	25 Ju 22	1 Au 22	8 Au 22	15 Au 22	22 Au 22	29 Au 22	5 Se 22	12 Se 22	19 Se 22	26 Se 22
TASK DESCRIPTION	**PLAN START**	**PLAN END**	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Brainstorm and Research	4/25/2022	6/17/2022	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛															
Meet and Select Advisor	4/25/2022	5/2/2022	🟦	🟦																					
Brainstorm Project Ideas; Select Final	5/2/2022	5/22/2022		🟦	🟦	🟦																			
Background Reading and Literature Review	5/22/2022	6/17/2022				🟦	🟦	🟦	🟦	🟦															
Initial Development	6/17/2022	7/22/2022								⬛	⬛	⬛	⬛	⬛	⬛										
Data Generation	6/17/2022	7/8/2022								🟨	🟨	🟨	🟨												
Metric Development	6/24/2022	7/22/2022									🟨	🟨	🟨	🟨	🟨										
Specification and Design Report	6/24/2022	7/8/2022									🟨	🟨	🟨												
Specification and Design Slides	6/24/2022	7/8/2022									🟨	🟨	🟨												
Specification and Design Oral Presentation	7/8/2022	7/15/2022											🟨	🟨											
Modeling	6/24/2022	8/26/2022									⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛	⬛					
Feature Engineering	6/24/2022	7/22/2022									🟩	🟩	🟩	🟩	🟩										
Tuning Model	7/8/2022	8/26/2022											🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩					
Training and Testing Model	7/8/2022	8/26/2022											🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩					
Tweaking Model	7/8/2022	8/26/2022											🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩					
Analyzing Model Output	7/8/2022	8/26/2022											🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩					
Final Results	8/19/2022	9/23/2022																	⬛	⬛	⬛	⬛	⬛	⬛	
Final Presentation Slides	8/19/2022	8/26/2022																	🟪	🟪					
Final Presentation	8/29/2022	9/2/2022																			🟪				
Write-up of Dissertation	8/29/2022	9/23/2022																			🟪	🟪	🟪	🟪	

Figure 23: Timeline of Dissertation