Predicting US Elections with Social Media and Neural Networks

Ellison Yin Nang Chan

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science at

Concordia University

Montreal, Quebec, Canada ´

February 2019

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Ellison Yin Nang Chan

Entitled: Predicting US Elections with Social Media and Neural Networks

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair

Dr. T.-H. Chen

_____ Examiner

Dr. T. Glatard

_____ Examiner

Dr. C. Poullis

_____ Co-supervisor

Dr. Ching Y. Suen

_____ Co-Supervisor

Dr. Adam Krzyzak

Approved by _____ Date: _____

Abstract

Increasingly, politicians and political parties are engaging their electors using social fpmedia. In the US Federal Election of 2016, candidates from both parties made heavy use of Social Media, particularly Twitter. It is then reasonable to attempt to find a correlation between popularity on Twitter, and eventual popular vote in the election. In this thesis, we will focus on using the subscriber 'location' field in the profile of each candidate to estimate support in each state.

A major challenge is that the Twitter location field in a user profile is not constrained, requiring the application of machine learning techniques to cluster users according to state.

In this thesis, we will train a Deep Convolutional Neural Network (CNN) to classify place names by state. Then we will apply the model to the Twitter Subscriber 'location' field of Twitter subscribers collected from each of the two candidates, Hillary Clinton (D), and Donald Trump (R). Finally, we will compare predicted popular votes in each state, to the actual results from the 2016 Presidential Election.

The hypothesis is that a city name has a strong correlation to the people who founded it and then incorporated it. Further, it's hypothesized that the original settlers were mostly homogeneous, relative to the country of origin and shared a common language, thus resulting in place names using the language of their origin.

In addition to learning the pattern related to the State Names, this additional information may help a machine learning model learn to classify locations by state.

The results from our experiments are very promising. Using a dataset containing 695,389 cities, correctly labelled with their state, we partitioned the cities into a training dataset containing 556,311 cities, a validation dataset containing 111,262, and a test dataset containing 27,816. After the trained model was applied to the test dataset. We achieved a Correct Prediction rate of 84.4365%, a False Negative rate of 1.6106%, and a False Positive rate of 1.0697%.

Applying the trained model on Twitter Location data of subscribers of the two candidates, the model achieved an accuracy of 90%. The trained model was able to correctly pick the winner, by popular vote, in 45 out of the 50 states. With another US and Canadian election coming up in 2019, and 2020, it would be interesting to test the model on those as well.

Acknowledgments

I express my gratitude to Dr. Ching Y. Suen, and Dr. Adam Krzyzak for their great patience and guidance, in refining the ideas in my research. I thank Dr. Suen for his friendship and welcoming me into the CENPARMI family. I thank Dr. Krzyzak for an excellent Machine Learning course, which opened my eyes to the possibilities. In addition, I thank Dr. Leila Kosseim and her insightful course, introducing me to the intricacies Artificial Intelligence and Dr. Sudhir Mudur, for an excellent Advanced Graphics Course which introduced me to Nvidia GPUs, which was a major component of this thesis.

Finally, I would like to dedicate this work to both my parents Wai Hong Chan and Shun Har Chan, who unfortunately are no longer with me, at the end of my academic journey. I'm sure they both are smiling from heaven.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Research Topic

Increasingly, politicians and political parties are engaging their electors using social media. In the US Federal Election of 2016, candidates from both parties made heavy use of Social Media, particularly Twitter. It is then reasonable to research a correlation between popularity on Twitter, and eventual popular vote in the election. In this thesis, we will focus on using the subscriber location information of each candidate to estimate support in each state.

For this thesis, over 7.8 million subscriber profiles were gathered from Twitter for the two candidates, Hillary Clinton and Donald Trump, between 2015 and 2016.

The goal is to use the locations of subscribers to each Candidate to determine who will win the most popular votes by state.

In fact, cities and towns in the US have been named by the people who colonized the area. For instance, in the Eastern US states, the northern states have primarily been colonized by people of French descent, and many place names are French, whereas place names get increasingly English as you move south. Same patterns are also seen. as you head west, with the place names adopting more Native American languages, and progressively more Spanish as you head towards the west and south. With this technique, as well as other features, we will classify city and town names into states by using a suitably trained CNN classifier. It is felt from after having studied the analysis done by LeCun [1] that for its accuracy and shorter convergence time, especially when

using GPU acceleration that CNN is an ideal algorithm for classifying the state names when representing them as sparse images.

Furthermore, the CNN should also be able to recognize letter patterns, such as the short state names along with full state names. (i.e. CA/California, AK/Alaska, ME/Maine …)

1.2 Challenges

A major challenge is that the Twitter “location” field in a user profile is not constrained, requiring the application of AI and Machine Learning algorithms to create clusters corresponding to state names.

The task at hand is to turn the chaos into order. The Twitter "location" field in a Twitter subscriber profile provides information about where the user resides. However, the text is unconstrained. To obtain information from this Twitter field requires filtering out text that has a low likelihood of being a location, then grouping the remaining locations into US States. Additionally, we need to remove locations with a low likelihood of being a US State.
Since we have over 7.8 million locations to classify, we need an algorithm suitable for such a large dataset.

One possibility is an unsupervised clustering algorithm, such as K-Means, which has an algorithmic complexity of $O(n^2)$. The challenge of using this algorithm is the time required to achieve the clustering on such a large dataset and the second challenge is that we need to apply the long clustering process to every new dataset. Research has been done to speed up this algorithm with GPU acceleration, with good success. [2] However, this does not achieve our goal of having an algorithm that can easily be applied to input data.

Using a supervised algorithm, such as Convolutional Neural Network (CNN), will achieve the goal of training once, and applying to future data. Additionally, many modern Deep Learning frameworks such as Tensorflow, and Keras can use GPU acceleration to speed up training.

1.3 Thesis Approach

In recent years there has been a big resurgence in Neural Networks, primarily due to an increase in computing power, which enables training dense Deep Learning Neural Networks quickly.

The company Google has created a computational framework called Tensorflow, which enables GPU accelerated high-performance computations. This is especially suitable for creating and training Deep Learning Neural Networks. Weight calculations are matrix multiplications, which are well suited for parallel execution on massively parallel systems, such as the GPU graphics platform.

In this thesis, our primary tool for learning the similarity function for state classification will be done using a Convolution Neural Network, with the Keras library running in the Tensorflow framework. This enables high-speed training, and inferencing.

After training a CNN to recognize an input dataset of cities labelled with their corresponding states, we will apply this CNN to predict the US state of a Twitter Subscriber, based on what they input into the 'location' field in their profile. The hypothesis is that subscribers, on twitter, to a candidate is an expression of support, and will translate to a vote for the candidate, in the election. We will then compare the predicted result to the actual election results for the 2016 US Federal Election.

We will encode the cities as bigrams, with a vocabulary of 26 uppercase and 26 lowercase letters, including 3 non-alphabet characters, giving us a total vocabulary size of 55 characters. We will then convert this bigram matrix into 55x55 RGB images. This will enable a CNN to train on the city names. In fact, once encoded as images, the images resemble what's in the MNIST database, which is a standard dataset used to train models to recognize handwritten numbers. A sample from the MNIST Database is in Figure 1. A sample of our encoding of the city name bigrams can be found in Figure 8. They are similar in size and sparseness of the image.

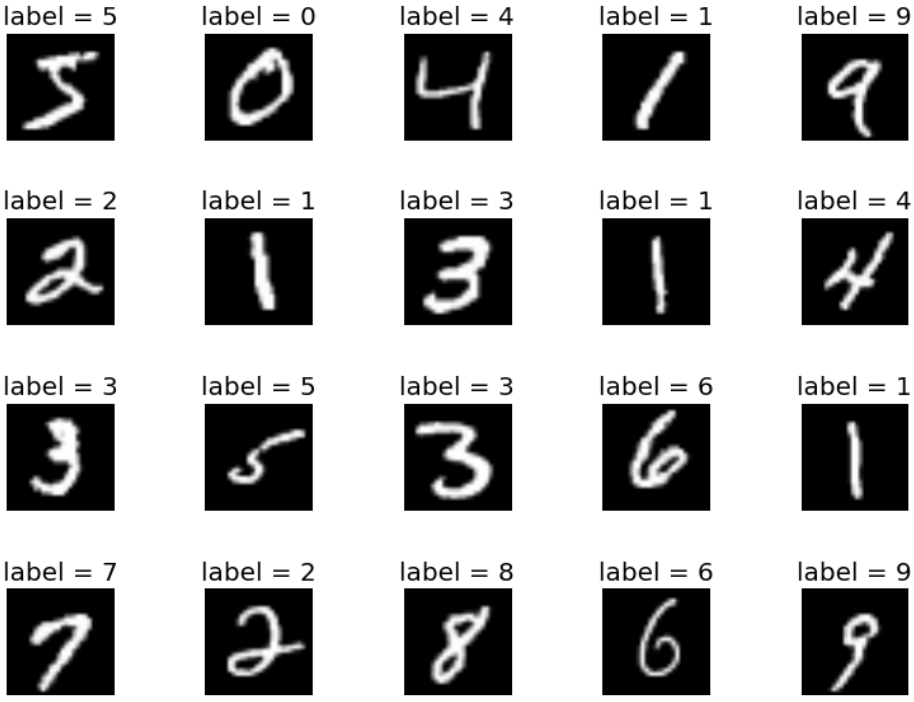


Figure 1 - MNIST Database Sample[1]

[1] C. Cortes, C. J. Burges and Y. LeCun, "MNIST database of handwritten digits - Yann LeCun," [Online]. Available: http://yann.lecun.com/exdb/mnist/

1.4 Related Work

Use of CNN for NLP is not new. Typically, sentences are converted into vectors using word embedding, using an algorithm such as Word2Vec [3], from Google. Recently, Google has patented this algorithm, which might have an impact on the research community.

In related work at CENPARMI, Ebrahimi, Mohammad Reza; Suen, Ching Y.; Ormandjieva, Olga [4], has used a Deep Learning CNN and Word2Vec with GPU acceleration, in order to detect predatory conversations, using social media. In the paper, "CNN-Webshell: Malicious Web Shell Detection with Convolutional Neural Network" [5], a CNN and Word2Vec are used to detect malicious Web Shell patterns inside the URLs of HTTP requests.

1.5 Novelty of Thesis Approach

As mentioned in the previous section, CNNs have been used in combination with a Word2Vec embedding, to perform NLP on sentences. In this thesis, we are not interested in parsing sentences and classifying them into categories. Instead, we are interested in building a classifier for state names. This prevents us from using Word2Vec type of embedding, the letters in themselves do not have associated meanings. We, therefore, need to apply context to the letters in relationship to specific state names. The feature we choose to use, is the frequency of occurrence of a bigram in city names, in the state. Each state will have a different frequency which could uniquely identify the states. An example of plots of bigrams frequency is seen in Figure 2, which is the side view and in Figure 3, which is the top view.

The novelty of this thesis approach is that rather than relying on Google's word embedding, we train a CNN to learn the bigram embedding in relationship to each state. We do this by training a CNN to recognize the affinity that a state has to a set of letter bigrams.

After searching through prior work, it is believed that this a new approach that has not been tried yet.

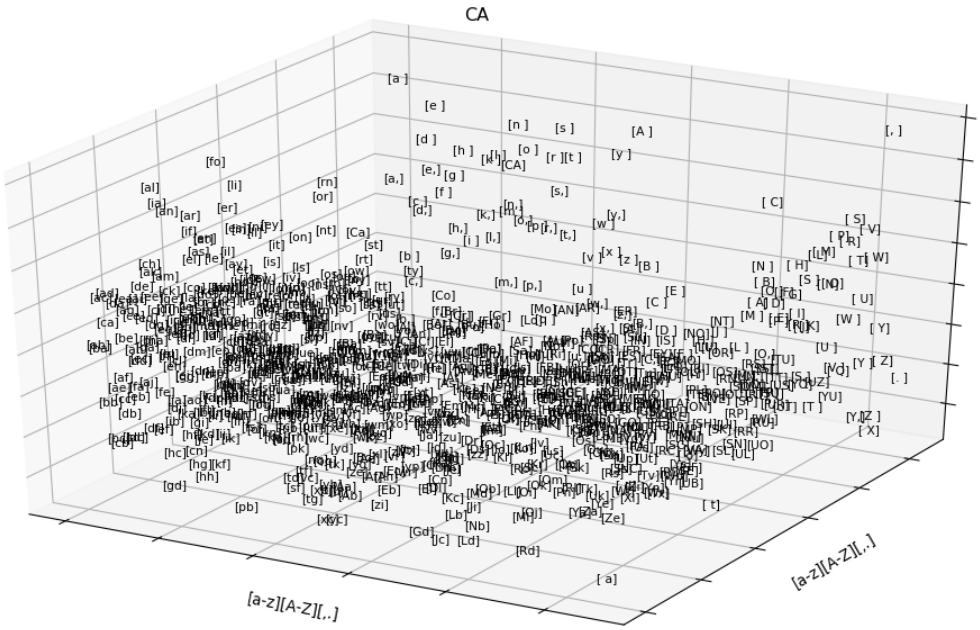


Figure 2- California Bigram Frequency Plot (side view)

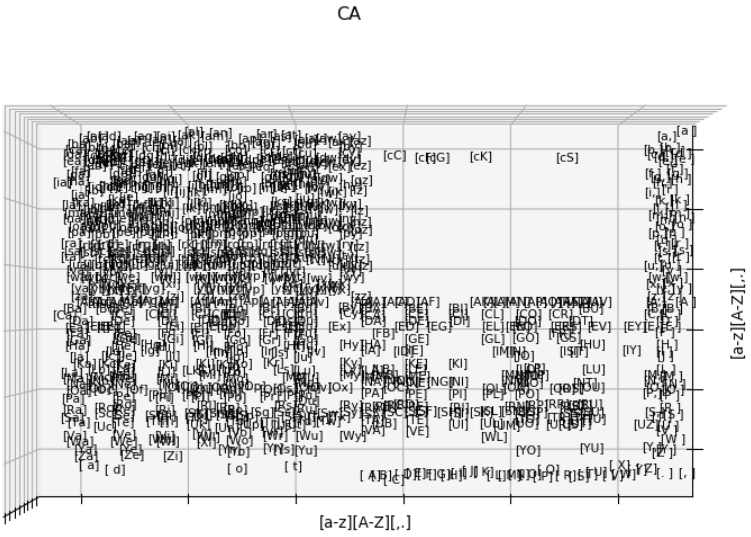


Figure 3 - California Bigram Frequency Plot (top view)

2 Performance Analysis for MNIST of Classification Algorithms

The approach chosen is to convert state names into letter bigrams, and subsequently encode the bigrams as 2-dimensional arrays which can be easily represented as 2D images. Using our dataset of correctly labelled states, we will train a model to classify location names into their respective states.

The MNIST database is the standard when it comes to image recognition algorithm performance analysis. In LeCun's website [1], he compares various machine learning algorithmic performances on classification of the MNIST handwritten digit database. The following tables are extracted from the LeCun publication [1], tabulating the Test Error rate for each type of classifier. This data clearly indicates that the best performing algorithm for small images with sparse features, such as what is contained in the MNIST Database are well suited for the Convolution Neural Network based algorithms. The Error Rate for CNN, according to this survey is below 2% with rates as low as 0.23%. (Table 4) On the other hand, other algorithms, such as K-Nearest-Neighbor (KNN) have significantly higher Error Rates, which can be as high as 5.0%. (Table 2)

2.1 Linear Classifiers

In Table 1 LeCun [1] has tabulated some results of linear classifier performance on the MNIST dataset. The error rates are as high as 12%.

Table 1 - Linear Classifiers [1]

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Linear Classifiers			
linear classifier (1-layer NN)	none	12.0	LeCun et al. 1998
linear classifier (1-layer NN)	deskewing	8.4	LeCun et al. 1998
pairwise linear classifier	deskewing	7.6	LeCun et al. 1998

2.2 K-Nearest Neighbors

In Table 2 LeCun [1] examines the KNN algorithm on MNIST, and the results are much better going down to the single digit test error rates for most of the tested algorithms. Although performance is higher than Linear Classifiers, we do not achieve the goal of training a general classifier that can be applied to new datasets. We must apply the KNN algorithm each time we want to classify a new set of data.

Table 2 - K-Nearest Neighbors [1]

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
K-Nearest Neighbors			
K-nearest-neighbors, Euclidean (L2)	none	5.0	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	none	3.09	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	none	2.83	Kenneth Wilder, U. Chicago
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4	LeCun et al. 1998
K-nearest-neighbors, Euclidean (L2)	deskewing, noise removal, blurring	1.80	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring	1.73	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 1-pixel shift	1.33	Kenneth Wilder, U. Chicago
K-nearest-neighbors, L3	deskewing, noise removal, blurring, 2-pixel shift	1.22	Kenneth Wilder, U. Chicago
K-NN with non-linear deformation (IDM)	shiftable edges	0.54	Keysers et al. IEEE PAMI 2007
K-NN with non-linear deformation (P2DHMDM)	shiftable edges	0.52	Keysers et al. IEEE PAMI 2007
K-NN, Tangent Distance	subsampling to 16x16 pixels	1.1	LeCun et al. 1998
K-NN, shape context matching	shape context feature extraction	0.63	Belongie et al. IEEE PAMI 2002
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

2.3 Neural Networks

In Table 3, we have LeCun's, analysis of neural networks of varying size and deepness. Performance varies depending on how the training data is prepared, the size of the hidden layers, and the number of hidden layers. We can see the deeper the network, the better the performance we get.

Table 3 - Neural Networks [1]

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Neural Nets			
2-layer NN, 300 hidden units, mean square error	none	4.7	LeCun et al. 1998
2-layer NN, 300 HU, MSE, [distortions]	none	3.6	LeCun et al. 1998
2-layer NN, 300 HU	deskewing	1.6	LeCun et al. 1998
2-layer NN, 1000 hidden units	none	4.5	LeCun et al. 1998
2-layer NN, 1000 HU, [distortions]	none	3.8	LeCun et al. 1998
3-layer NN, 300+100 hidden units	none	3.05	LeCun et al. 1998
3-layer NN, 300+100 HU [distortions]	none	2.5	LeCun et al. 1998
3-layer NN, 500+150 hidden units	none	2.95	LeCun et al. 1998
3-layer NN, 500+150 HU [distortions]	none	2.45	LeCun et al. 1998
3-layer NN, 500+300 HU, softmax, cross entropy, weight decay	none	1.53	Hinton, unpublished, 2005
2-layer NN, 800 HU, Cross-Entropy Loss	none	1.6	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [affine distortions]	none	1.1	Simard et al., ICDAR 2003
2-layer NN, 800 HU, MSE [elastic distortions]	none	0.9	Simard et al., ICDAR 2003
2-layer NN, 800 HU, cross-entropy [elastic distortions]	none	0.7	Simard et al., ICDAR 2003
NN, 784-500-500-2000-30 + nearest neighbor, RBM + NCA training [no distortions]	none	1.0	Salakhutdinov and Hinton, AI-Stats 2007
6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions]	none	0.35	Ciresan et al. Neural Computation 10, 2010 and arXiv 1003.0358, 2010
committee of 25 NN 784-800-10 [elastic distortions]	width normalization, deslanting	0.39	Meier et al. ICDAR 2011
deep convex net, unsup pre-training [no distortions]	none	0.83	Deng et al. Interspeech 2010

2.4 Convolutional Neural Networks

In Table 4, LeCun [1] examines the Convolutional Neural Network (CNN) performance on the MNIST dataset. On sparse images such as MNIST, the CNN a better performer. This is due to the pooling of features between convolution networks. Since the bigram image encoding scheme, employed in this thesis creates images that are similar in sparseness, we feel that CNN is the algorithm to choose. We will elaborate on this in the next chapter.

Table 4 - Convolutional Neural Networks (CNN) [1]

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)	Reference
Convolutional nets			
Convolutional net LeNet-1	subsampling to 16x16 pixels	1.7	LeCun et al. 1998
Convolutional net LeNet-4	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with K-NN instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-4 with local learning instead of last layer	none	1.1	LeCun et al. 1998
Convolutional net LeNet-5, [no distortions]	none	0.95	LeCun et al. 1998
Convolutional net LeNet-5, [huge distortions]	none	0.85	LeCun et al. 1998
Convolutional net LeNet-5, [distortions]	none	0.8	LeCun et al. 1998
Convolutional net Boosted LeNet-4, [distortions]	none	0.7	LeCun et al. 1998
Trainable feature extractor + SVMs [no distortions]	none	0.83	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [elastic distortions]	none	0.56	Lauer et al., Pattern Recognition 40-6, 2007
Trainable feature extractor + SVMs [affine distortions]	none	0.54	Lauer et al., Pattern Recognition 40-6, 2007
unsupervised sparse features + SVM, [no distortions]	none	0.59	Labusch et al., IEEE TNN 2008
Convolutional net, cross-entropy [affine distortions]	none	0.6	Simard et al., ICDAR 2003
Convolutional net, cross-entropy [elastic distortions]	none	0.4	Simard et al., ICDAR 2003
large conv. net, random features [no distortions]	none	0.89	Ranzato et al., CVPR 2007
large conv. net, unsup features [no distortions]	none	0.62	Ranzato et al., CVPR 2007
large conv. net, unsup pretraining [no distortions]	none	0.60	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [elastic distortions]	none	0.39	Ranzato et al., NIPS 2006
large conv. net, unsup pretraining [no distortions]	none	0.53	Jarrett et al., ICCV 2009
large/deep conv. net, 1-20-40-60-80-100-120-120-10 [elastic distortions]	none	0.35	Ciresan et al. IJCAI 2011
committee of 7 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.27 +- 0.02	Ciresan et al. ICDAR 2011
committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions]	width normalization	0.23	Ciresan et al. CVPR 2012

2.5 Simple Pattern Matching

Creating regular expressions to match location names. It's a manual process requiring lots of work to capture a pattern or set of patterns for all 50 states, and various permutations. Old school method, being replaced by Machine Learning algorithms.

City and state were obtained from compiled list of cities, and properly labeled states. In total 51 states labels were trained. (See section 9.1.4) Using reverse indexed array and pattern matching would be possible but would not provide the flexibility of SoftMax classification with a neural network. Pattern matching is a all or nothing method, and require multiple iterations to develop a generalized matcher. Even after such a pattern matcher is developed, we could still lose data due to previously unseen data falling out of the pattern matched. Additionally, using a SoftMax activation for the classifier in a neural network provides us a probability for each class(state) and allows us to adjust our acceptance cutoff.

It's not our opinion that a regular expression will work, however it will not be easy or possible to define one that can match all cases of misspelling of a state or city name, for instance. Also defining a regular expression is a manual process whereas training a neural network to recognized misspelled place names can be automated.

3 Convolutional Neural Network Basics

After studying LeCun's research, CNN was chosen for it's quick convergence time, and high accuracy. In addition, we are looking for a supervised algorithm that would enable us to train a model on a dataset of cities labeled by state, apply the model to Twitter subscriber location field. According to LeCun's study, a Convolutional Neural Network (CNN) is used for image recognition, to classify entire images or recognition of objects inside an image. A CNN will typically have a test error rate of under 1%. With today's hardware advancements, and software advancements such as GPU acceleration, and neural network enabling frameworks like Keras, and Tensorflow, training large Deep Learning Neural Networks has been greatly improved. It makes sense to try and reframe the classical text classification problem as an image classification problem.

It's felt that the spatial relationship is present in place names and state labels. In fact what the CNN appear to have learned is the special relationship between the bigrams of a place name, relative to language of the place name, and presence of either the short state name or long state name bigrams.

A typical CNN will consist of alternating Convolution Layers, Pooling Layers, and finally a Fully Connected (FC) layer. Each layer is fed into a subsequent layer in a sequential fashion. The convolution layers serve the purpose of learning features in the input image, eventually feeding into the FC layer for classification. The following is a detailed explanation of the function of each layer.

3.1 Convolution Layers

The simple concept of a Convolution Layer is that it's a series of dot products. The goal is to reduce the size of the original image into a smaller set of features. In effect, the image is filtered each time it passes through a convolution Layer.

The following figures help to illustrate how a CNN network functions and are extracted from Michael Nielsen's online book[2].

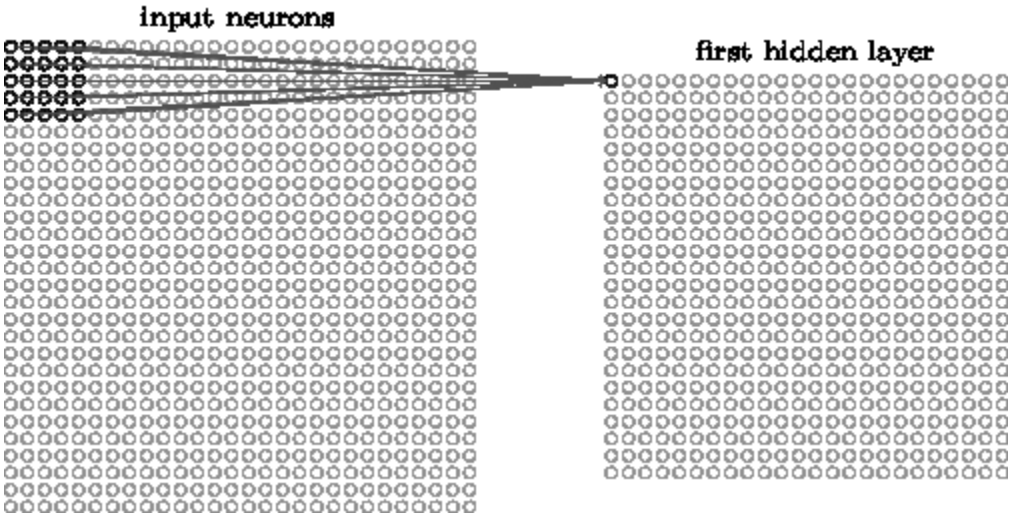


Figure 4- Convolution Layer First Pass

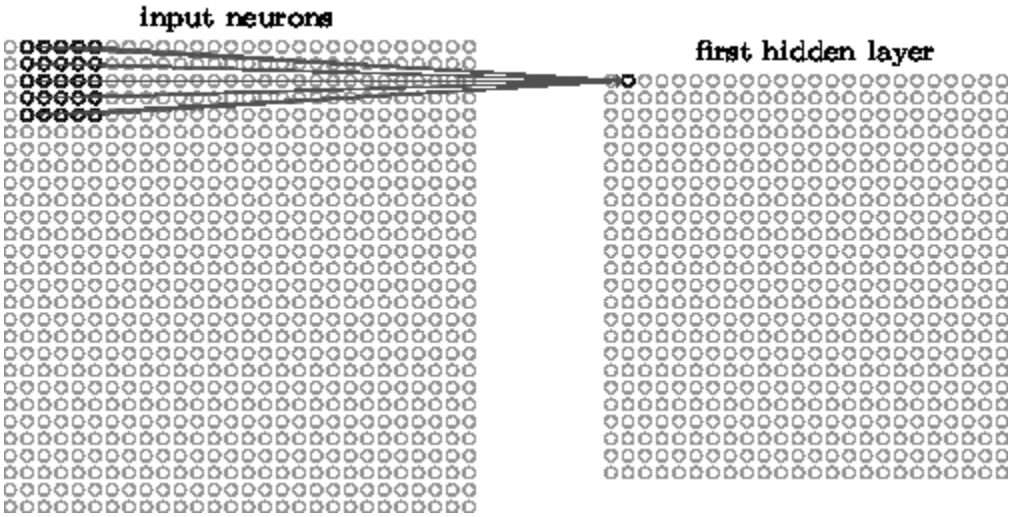


Figure 5- Convolution Layer Second

[2] http://neuralnetworksanddeeplearning.com/chap6.html

3.2 Pooling Layers

The output of a convolution layer often results in a sparse matrix. To further reduce the output of a convolution layer, while keeping the learned features intact, a pooling layer is applied after each convolution. As an example, a 2x2 max pooling layer would take the maximum value of a group of 4 neurons on the output of a convolution layer and reduce that to a single neuron. In this paper, the authors discuss the use of Stochastic Pooling for regularization of a CNN. [6]

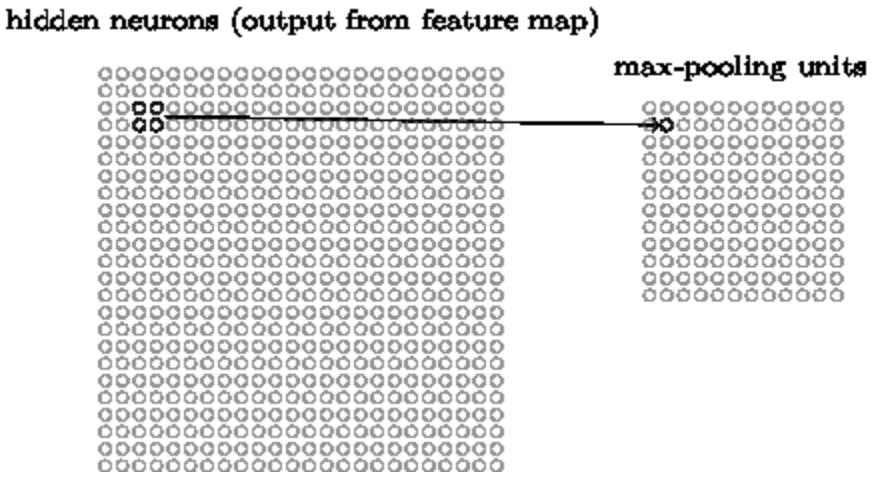


Figure 6- 2x2 Max Pooling

3.3 Fully Connected Layer

After the application of one or more convolution and pooling layers, the outputs are then passed into a fully connected layer with an activation function for classification. Typically, this will be a two-layer neural network classifier, where the inputs represent the features found in the convolution layers, and the output being the classes under which the inputs fall.

Typical choices for activation are Sigmoid for multiclass, and SoftMax for single class.

3.4 Dropout

To avoid overfitting, a technique often used is to randomly reset a certain percentage of each convolution pass. This is achieved using a “dropout” layer after each pooling layer. At each dropout lay weights are randomly selected and removed. Because in applications where images are considered sparse, we can remove weights which have become zero.

An analysis is made by Park S., and Kwak N. [7] analysis the dropout effect on a CNN.

4 Data Description and Model Training

The method we decided to use to build our model with is a Convolutional Neural Network for classifying our city names. With current advances in hardware (GPU training), and software (Tensorflow), we can train on GPU hardware and Deep Learning Neural Networks.

We will be converting location names to bigram frequency, and them mapping them on two axes and using a Convolution Neural Network to classify into states by image recognition.

Data representation is the key to a successful algorithm. In our algorithm, we choose to represent place names as bigrams and then convert bigrams into a two-dimensional matrix with each point representing the frequency of occurrence of a bigram in the place name.

4.1 Data Sources

The cities and labels were obtained from two sources.

Purchased Dataset:

https://www.uscitieslist.org/cart/packages/

Open Source Dataset;

https://github.com/grammakov/USA-cities-and-states

4.2 Data Description

The data comes in two categories. The first is a labelled set of city names and labelled with each state that the city belongs to. These are all correctly spelled and correctly labelled. This is the basis of the training, validation and test sets, from which the model is trained, validated and tested. A sample of this is found in Table 5. We will place it under the umbrella of Model Data.

Table 5 - US Cities Dataset (Excerpt)

1,Adak,Aleutians West Census Area,AK,Alaska,City,51.88,-176.65806
2,Akhiok,Kodiak Island Borough,AK,Alaska,City,56.94556,-154.17028
3,Akiachak,Bethel Census Area,AK,Alaska,CDP,60.90944,-161.43139
4,Akiak,Bethel Census Area,AK,Alaska,City,60.91222,-161.21389
5,Akutan,Aleutians East Borough,AK,Alaska,City,54.13556,-165.77306
6,Alakanuk,Kusilvak Census Area,AK,Alaska,City,62.68889,-164.61528
7,Alatna,Yukon-Koyukuk Census Area,AK,Alaska,CDP,66.56393,-152.838

The second category of input data is the Twitter subscriber profile 'location' field. This is the data which we want to apply the model against to help us classify Twitter subscribers into states for each of the election candidates. A sample of these data can be found in Table 6.

To improve data processing time, the data has been converted to JSON format with only the essential information that we anticipate will be needed for training and prediction. Examples of the converted JSON are in Table 7. We will extract the location of each subscriber and convert the location field into bigram. A sample of this converted location name is in Table 8.

Several possibilities exist for those who are subscribing to both candidates. Those could include journalists, or possibility undecided voters. From a statistics point of view, and in relation

to building our model, these subscribers would cancel out each other for the state they reside and will not skew or bias the results.

Table 6 – Sample Twitter Profile Locations


```
[{"logged_at": "2016-10-23T10:30:06",
 "created_at": "2011-12-09T18:30:19",
 "id_str": "433007427",
 "last_tweet": "788554771250581505",
 "favourites_count": 324,
 "followers count": 56,
 "friends count": 184,
 "listed count": 3,
 "statuses count": 294,
 "following_id_str": "HillaryClinton",
 "is_translator": 0,
 "geo_enabled": 0,
 "location": "Arlington, Texas",
 "verified": 0,
 "screen_name": "<removed>",
 "lang": "es",
 "utc_offset": -1,
 "time_zone": "", "name":
 "????", "description":
 "<removed>",
 "argmax": 0.0,
 "orig": null,
 "idx": null,
 "prob": null,
 "correct": null}]
```


Table 7 – US Cities Dataset Converted o JSON


```
{"state_short": "FL", "letters": ["FL", " W", "at", "er", "ga", "te"], "location":
"FL Watergate"}
{"state_short": "TX", "letters": ["Cr", "ec", "y,", " T", "X "], "location": "Crecy,
TX"}
{"state_short": "TN", "letters": ["Da", "nd", "ri", "dg", "e ", "TN"], "location":
"Dandridge TN"}
{"state_short": "NC", "letters": ["NC", " L", "um", "be", "rt", "on"], "location":
"NC Lumberton"}
```


Table 8 – Sample Twitter Profile Locations Converted to JSON

```
{"letters": ["so", "me", "wh", "er", "e ", "be", "in", "g ", "pe", "tt", "y "],
"following_id_str": "HillaryClinton", "logged_at": "2016-10-23T10:30:06", "id_str":
"35399372", "created_at": "2009-04-25T22:11:25", "verified": 0}
{"letters": ["Ar", "li", "ng", "to", "n,", " T", "ex", "as"], "following_id_str":
"HillaryClinton", "logged_at": "2016-10-23T10:30:06", "id_str": "859131001",
"created_at": "2012-10-02T21:35:56", "verified": 0}
{"letters": ["DC"], "following_id_str": "HillaryClinton", "logged_at": "2016-10-
23T10:30:06", "id_str": "2333943649", "created_at": "2014-02-08T12:02:58",
"verified": 0}
```


4.3 Model Input

The Model Data will be pre-processed before inputting into the CNN. This includes a permutation of place names to augment the input data, and remove any duplicate city names, the total number of cities with corresponding state names is 695389 entries. The premise is that a CNN can learn to recognize a pattern in the place names when properly trained.

We will calculate a letter bigram for each city, and then count the frequency of occurrence in each city name. This will be fed into the CNN as training data. This has been successfully used to classify languages based on small corpora. It's a hypothesis that this can work in location classification as well.

4.4 Duplicate Data

It is difficult to deal with collisions of city names. However, in most cases, city names are entered in conjunction with state names. Additionally, the model does not learn only by matching 112 city names as a pattern, but rather the frequency of bigrams in all city in a state. This is one advantage over a simple pattern matcher.

4.5 Algorithm

The bigrams for each place name will be converted into an image representation to be fed into a CNN for training. In image processing, the HOG algorithm is quite successful. By representing our bigrams as a histogram of occurrences in the location names we can build a matrix to represent each location and convert that matrix into an image for our CNN model.

4.5.1 Vocabulary of Characters

. We want to limit our vocabulary size for our input data so that each city name will not map into too large a matrix for the efficient training and running of our model. Characters chosen to be part of our vocabulary are listed in Table 9. They include all letters of the alphabet, distinguishing between upper and lower case. We chose to do this city because we are classifying place names where capitalization is an important factor. In addition, we will include two punctuations, comma and period, which are also important in place names. Space is included in vocabulary, but we will trim the left and right of the place name and remove punctuations and spaces. Only internal punctuation and spaces are kept. With the x-axis representing bigram

combinations of our vocabulary and. y-axis representing bigram frequency shows histogram plots of bigram frequencies for some sample states. The states California, Arkansas, Massachusetts, Pennsylvania, and New York are representative of densely populated states. The histograms are similar in profile but there are enough dissimilarities to consider each state's histogram to be unique. It's a reasonable assumption, then, to expect that a neural network will be capable of learning this pattern. The question this thesis tries to answer is whether it can be generalized to the recognition of city names, which are not a part of the training data.

Table 9- Characters in the Vocabulary

Lower Case Letters	abcdefghijklmnopqrstuvwxyz
Upper Case Letters	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Punctuation (internal only)	,.
Space (internal only)	

4.5.2 Histogram Representation

With a vocabulary size of 55 characters, our bigrams can be represented by a 55x55 matrix of bigram frequencies, giving us a mapping space of 3025 bigrams. Our algorithm is a histogram method, such as HOG which converts an image into histograms, to recognize features for classification. The HOG algorithm has been successfully applied in previous work to recognize sparse featured images such as handwriting. An example of this is this paper where the authors used it for Arabic handwriting. [8] For further details, please see Appendix A

.

However, in our method, we will not directly calculate the Sobel operator, but rather we will train a Convolutional Neural Network to learn the features of an image built from letter bigram frequencies found in cities, of each US state. The histograms can be found in Figure 7. These histograms would represent all the cities for the state converted into bigram frequencies composed into one graph for each state. The individual city histograms would be much sparser. For instance, in Figure 8, and Figure 1, we present a 3D scatter plot of South Bradenton, FL, with bigram frequency on the z-axis. The X-Y coordinates are the characters in our vocabulary.

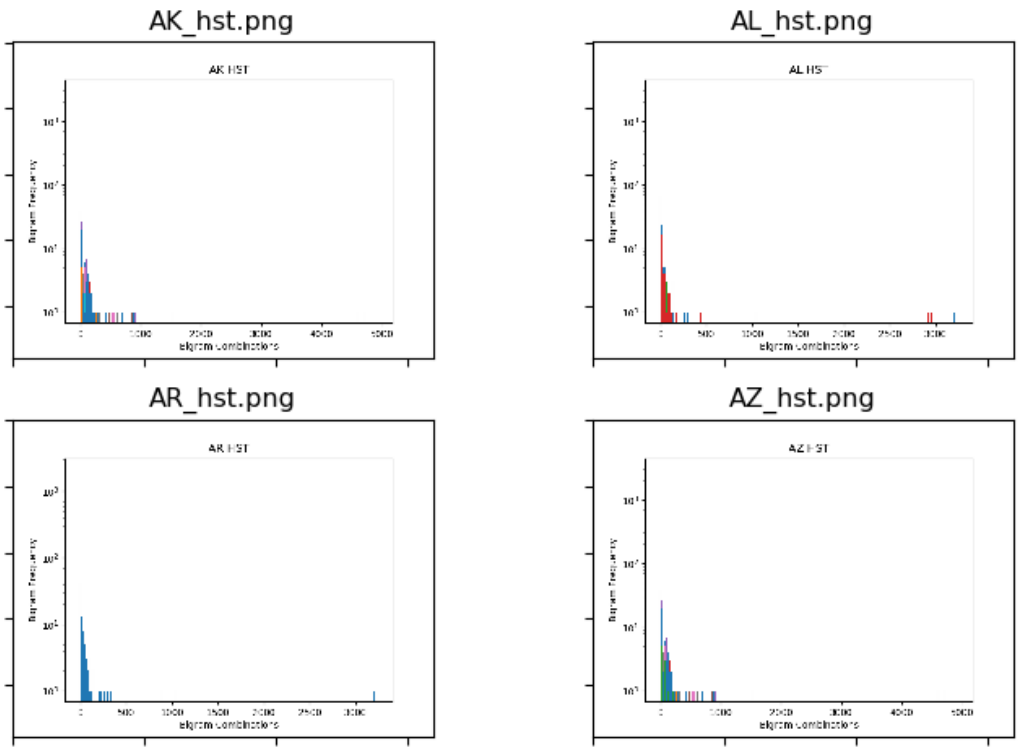


Figure 7 - Composite State Histograms for Some States

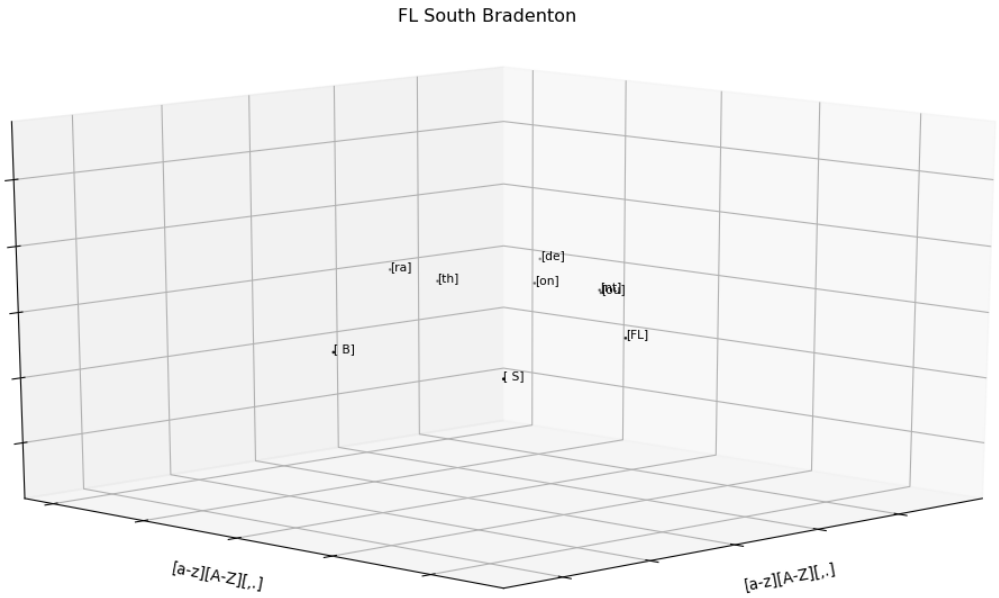


Figure 8- Bigram Frequency of South Bradenton, FL(Side)

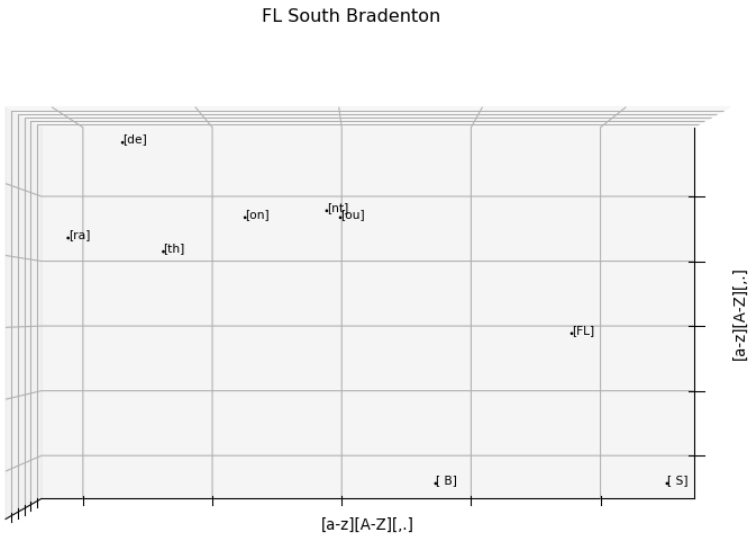


Figure 9 - Bigram Frequency of South Bradenton, FL(top)

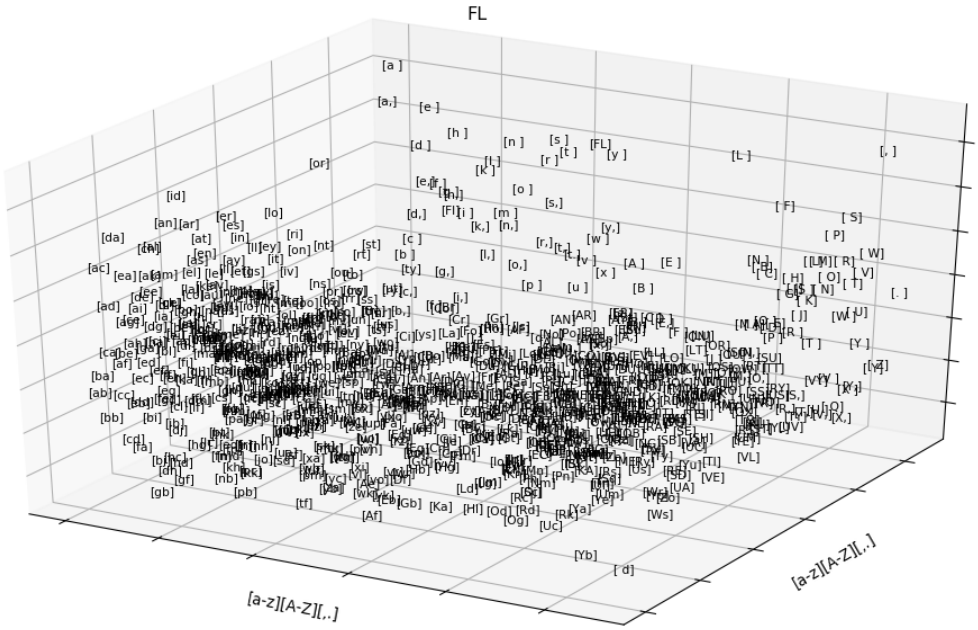


Figure 10 - Bigram Frequency for Florida (Side)

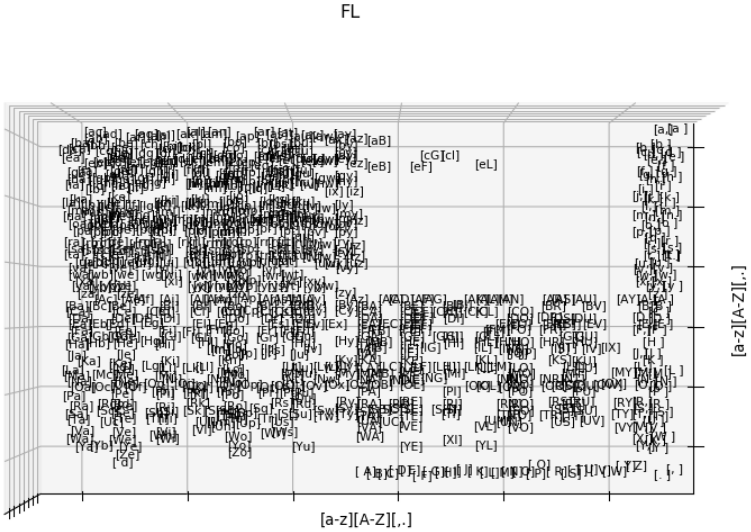


Figure 11 - Bigram Frequency for Florida (Top)

4.5.3 Data Encoding

As previously stated, we want to reframe the place name classification problem into an image processing problem, and to apply state-of-the-art hardware and software, actively being used in image classification research. Our data encoding is simple. The x and y-axis of our image will be one character in the bigram and each pixel in our images will represent the 24-bit frequency for that bigram. By convention, we will encode the first letter of the bigram in the x-axis and the second character in the y-axis.

Steps are:

1 - Remove leading and training spaces.

2 – Create bigram by selecting two-character pairs in succession.

3- Remove any character not falling in the vocabulary list, which are upper- and lower-case letters, and special characters, space, comma, period.

4.5.4 Converting to Image Representation

To enable a CNN to train on our histograms, we will convert them into a two-dimensional image. The x and y-axis are indexes to each letter of our vocabulary, and the RGB values represent a 24-bit sized frequency of occurrence for the bigram. More details of the encoding process are found in section Data Encoding. These are composite images of all the bigrams of all cities in that state.

Sample images built in this manner are shown in Figure 13. These images represent the superposition of all bigram frequencies for all cities in the respective state. The images help to illustrate that some bigrams combinations occur more frequently with some states. Together this represents a “fingerprint” of sorts that we hope a neural network can learn the pattern to and thus

be able to classify cities by their bigrams. As in fingerprint recognition, each fingerprint is composed of recognizable features. The CNN will be trained to recognize the features that each state fingerprint is composed of. Using our method, the image in Figure 12 represents bigram frequencies of all cities in the US.

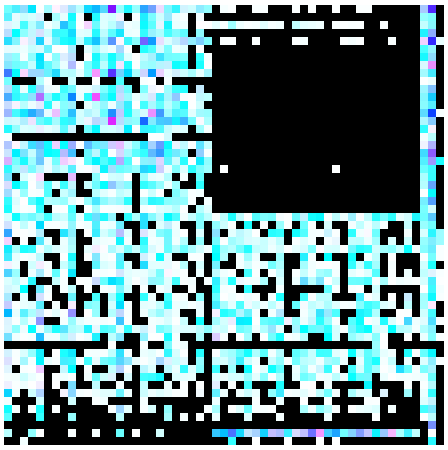


Figure 12 - Composite Picture of Bigrams in All Cities and All States

In Figure 13, are the images of bigram frequencies for some sample states individually. Even with a visual inspection, we can see that each state is quite unique. The full set of images for each state can be found in Appendix 7.2.

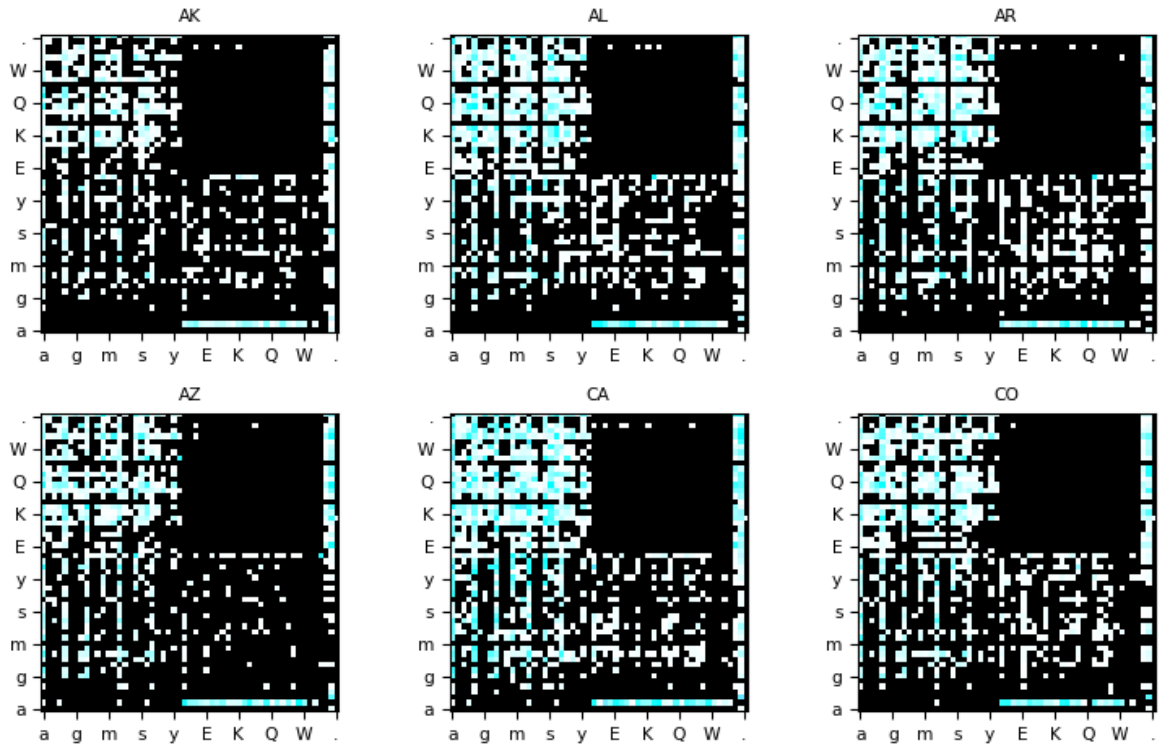


Figure 13 - Image Representation of Bigram Frequencies (Sample)

4.5.5 Splitting Dataset

Starting from the full list of cities and labels, we will create the Training, Validation, and Test Datasets. The CNN will be trained using the Training Set. The Validation Set will be kept separate from the Training set and used during training to calculate the loss and accuracy after every epoch. The test set is used to test the model accuracy at the end of training. It will be data that was not used as part of the training process.

It should be noted that we are only training on the 51 states that can vote in the US Presidential election. Territories such as Puerto Rico, Guam, etc. are not included in any of the training or validation datasets. It is felt that including non-voting states could skew the training.

4.5.6 Augmenting the Dataset

To augment the basic list of city names and states, the operations in Table 10 are applied to the original list. This will help to capture the variations that people will use in the location field of their Twitter profile. It's interesting to note that these permutations do not contain the city name alone with no state name. This is done intentionally to avoid confusing the neural network. Many states carry common city names and adding this permutation would result in adjusting weights towards one state, and then when the same city came in for another state it would undo that training resulting in the neural network oscillating back and forth, until the last entry of that city.

Table 10 - Permutation Operation on State Names and Cities

Operation	Description
[Long State Name]	Long state name alone
[City], [Long State Name]	City and Long State separated by a comma
[City] [Long State Name]	City and Long State Name separated by space
[Long State Name], [City]	Long State Name and City separated by a comma
[Long State Name] [City]	Long State Name and City separated by a space
[City], [Short State Name]	City and Short State Name separated by a comma
[City] [Short State Nam]	City and Short State name separated by a space
[Short State Name], [City]	Short State Name and City separated by a comma
[Short State Name] [City]	Short State Name and City separated by a space

4.5.7 Shuffling Data

An initial shuffling of the data is necessary, for the next step of splitting between the training, validation, and test sets. This will ensure an even distribution for each of the datasets after splitting. To achieve this, the numpy "shuffle" function is used. Numpy is a common python module used in data science for numerical and data analysis. In Figure 14, we have graphed the frequency with which a state appears in the dataset. The first graph represents the full dataset.

This matches very closely with the same graphs for the Training, Validation, and Test data splits. Therefore, we are confident that each split represents the total population of the full dataset.

4.5.8 Splitting Data into Training, Validation, and Test Sets

The final step is to split the data into the separate components. We will keep 80% of the original data for training while splitting the remaining 20% into validation (16%), and test (4%). The details on actual sizes of each and descriptive statistics are listed in Table 11.

4.5.8.1 Training Data

This is the main data set used to train the CNN.

4.5.8.2 Validation Data

During training we are using the validation dataset to compare training accuracy relative to accuracy for validation data. The difference between the two accuracies is calculated. Since we are using "early stop" in our training, if the difference between epochs is too low or unchanged, we will stop training the model.

4.5.8.3 Test Data

Test data is set aside and never seen during training. We will use this data to objectively evaluation our model, at the end of training.

4.5.9 Addressing Data Imbalance

We can see from Figure 14 that some states have significantly more cities than others. Data imbalance is a significant issue, when it comes to two class, as this could lead to imbalance in

predictions. However, in our case we have 51 classes, and it's felt that we had enough labeled cities in each state to avoid the issue. We also did not observe any bias when applying the trained model to the full dataset, and the subscriber locations. Furthermore, with Data Augmentation, as described in Table 10, we have boosted training samples for the lower city count states.

In future work, it would be useful to do a study on data imbalance.

Figure 14- Frequency Distribution Histogram for Datasets (All, Train, Validation, Test)

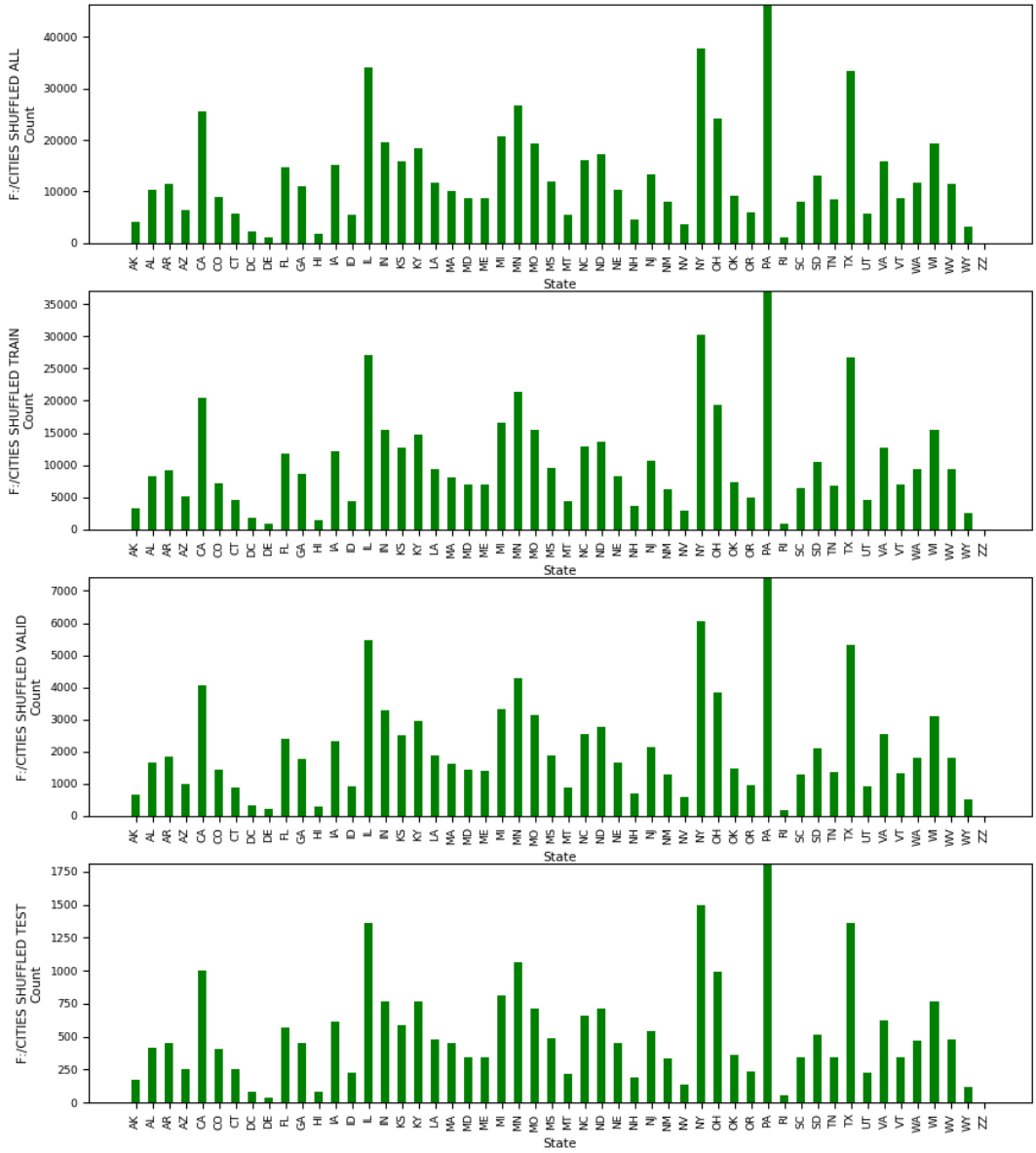


Table 11 - Training, Validation, Test Data

	City Entries	File Name	Description Statistics Numpy.describe()
Full Data (100%)	695389	cities_shuffled_all.json	nobs=52, minmax=(0, 46271), mean=12929.98076923077, variance=94827083.3525641, skewness=1.3722954619398018, kurtosis=1.9301177227759876
Training (80%)	556311	cities_shuffled_train.json	nobs=52, minmax=(0, 37041), mean=10342.307692307691, variance=60726949.62895928, skewness=1.3738867581234702, kurtosis=1.9387584597763698
Validation (16%)	111262	cities_shuffled_valid.json	(nobs=52, minmax=(0, 7420), mean=2069.6923076923076, variance=2435641.3152337857, skewness=1.3721646821298885, kurtosis=1.915920922857353
Test (4%)	27816	cities_shuffled_test.json	nobs=52, minmax=(0, 1810), mean=517.9807692307693, variance=148579.43099547512, skewness=1.3356616905991185, kurtosis=1.7888861304572803

4.6 Model Preparation

4.6.1 Model Definition

We are using five 2D convolution layers with 2x2 max pooling layers between each convolution layer. The Keras Neural Network Framework is being used to define our Convolutional Neural Network. Keras is a popular framework for defining, training, and generating predictions, and provides GPU acceleration using a Tensorflow Backend. The programming language of choice is Python.

The model is summarized in Table 12. We will provide the details in the next sections.

4.6.2 Convolution Layers Description

In out CNN model, we use 5 convolution layers. The first convolution layer is an input layer and designed for our 55x55, 3 colour image. The first layer will pick out the smaller features of our image, and pass it on to subsequent layers, after passing the features into a pooling layer.

4.6.3 Pooling Layers Description

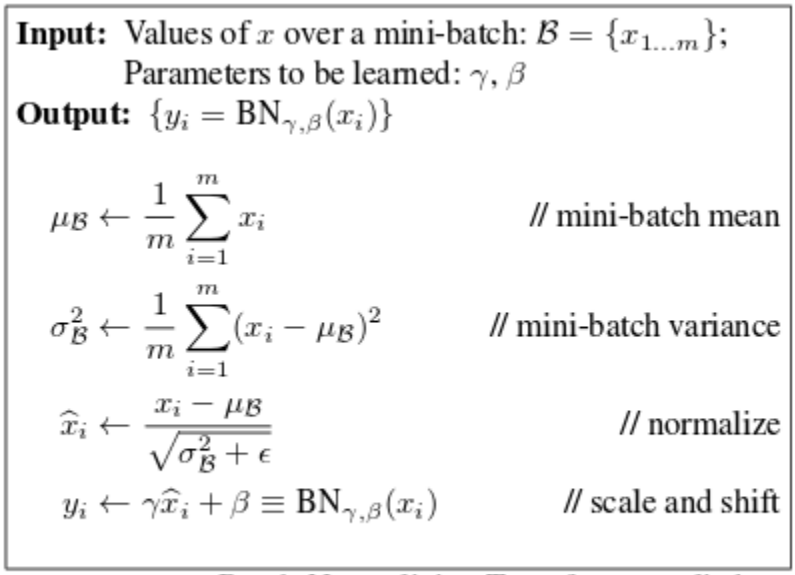
We are using a 2x2 pooling layer in between each convolution. Since each image represents the whole bigram space of 55x55 characters, resulting in 3025 bigrams, the resulting matrix of frequencies is considered sparse. This is a similar recognition problem of that of MNIST, and handwritten digit recognition. Like CNN models built to recognize MNIST characters, we will place pooling layers after each convolution layer. The pooling function we are using is Maxpool2d. This is a function that will condense every recognized feature from a 2x2 grid of neurons into one neuron. This is illustrated in Figure 6. This will help us, in reducing the size of subsequent convolution layers, by reducing the number of values that are zero and reducing the number of similar features.

4.6.4 Fully Connected Layers Description

The last two layers of our model are fully connected classification layers. The first one is a dense input layer which takes the input of the final pooling layer as input and feeds into a dense hidden layer that will classify the features into the final 51 classes. In our CNN, we have 51 classes, one for each state. We are using the Sigmoid activation function which is more suited for multiclass.

4.6.5 Batch Normalization

As the last step of each convolution layer, we will apply a batch normalization layer. According to S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [9], it will help to increase the accuracy or the model, even with the use of higher learning rates. This is achieved by normalizing the weight at the end of each convolution. Although some think that Batch Normalization has little effect on convergence and accuracy, such as discussed in the paper by these authors of the paper, “Batch Normalization: Is Learning An Adaptive Gain and Bias Necessary?” [10]. A deeper discussion is beyond this thesis, so please refer to the paper written by Ioffe & Szegedy for more details. In Figure 15, we present the formulae used in batch normalization.



Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Figure 15- Batch Normalization Formulae [9]

Table 12- Keras Model Definition

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 55, 55, 32)        896
_________________________________________________________________
activation_1 (Activation)    (None, 55, 55, 32)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 55, 55, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 18, 18, 64)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 18, 18, 64)        256
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 64)        36928
_________________________________________________________________
activation_3 (Activation)    (None, 18, 18, 64)        0
_________________________________________________________________
batch_normalization_3 (Batch (None, 18, 18, 64)        256
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
activation_4 (Activation)    (None, 9, 9, 128)         0
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 9, 128)         512
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 9, 9, 128)         147584
_________________________________________________________________
activation_5 (Activation)    (None, 9, 9, 128)         0
_________________________________________________________________
batch_normalization_5 (Batch (None, 9, 9, 128)         512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 4, 4, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 1024)              2098176
_________________________________________________________________
activation_6 (Activation)    (None, 1024)              0
_________________________________________________________________
batch_normalization_6 (Batch (None, 1024)              4096
_________________________________________________________________
dropout_4 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_2 (Dense)              (None, 52)                53300
_________________________________________________________________
activation_7 (Activation)    (None, 52)                0
=================================================================
```

4.7 Model Training

We trained our CNN model on 556311 city names from our training dataset. In addition to the model definition in Table 12, when training there are hyperparameters that need to be set, which can affect the outcome of the trained model. These are listed in Table 13. We will discuss these parameters in greater detail and provide some discussion on how training and model accuracies are affected in the next sections.

Table 13 - Training Hyperparameters

Parameter	Value	Description
Batch Size	32	Number of samples between gradient calculation.
Optimizer	Adam	Adam optimizer, with an initial learning rate of 0.001, and decay rate of 0.001.
Loss Function	Binary Cross Entropy	Calculates the loss between each gradient calculation.
Maximum Epochs	<= 75	Rather than using a fixed number of epochs, we are using early stop, if the validation loss does not change for 4 epochs.

4.7.1 Optimizer

In neural networks, the "learning rate" determines how fast it converges or whether it will converge at all. For our training, we are using the Adam optimizer to find the optimal learning rate. It has been shown to converge quickly for CNN [11] [12] [13] [14]. In Figure 16 and Figure 17, we have plotted the training and validation accuracies for each epoch. We can tell by this that the model has converged by epoch 4 with slight oscillation. After epoch 4 the training and validation accuracies are similar along with the training and validation losses.

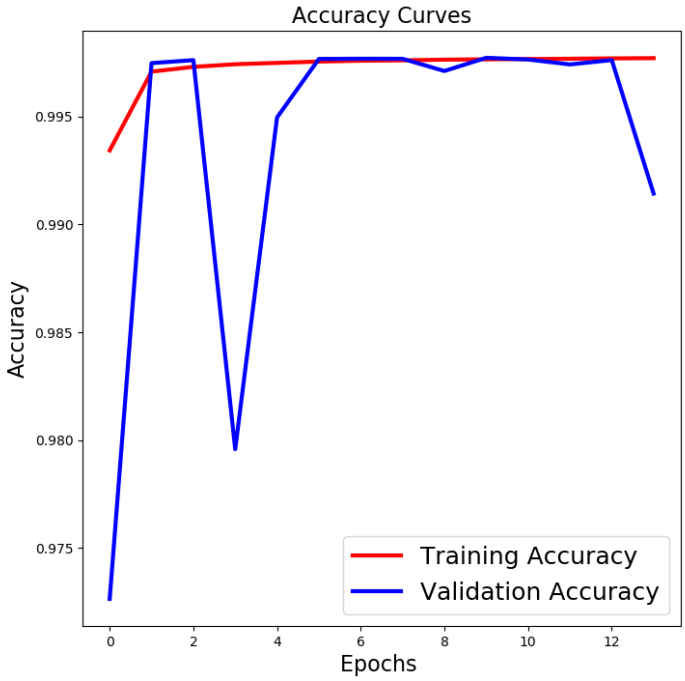


Figure 16-Training and Validation Accuracy Curves

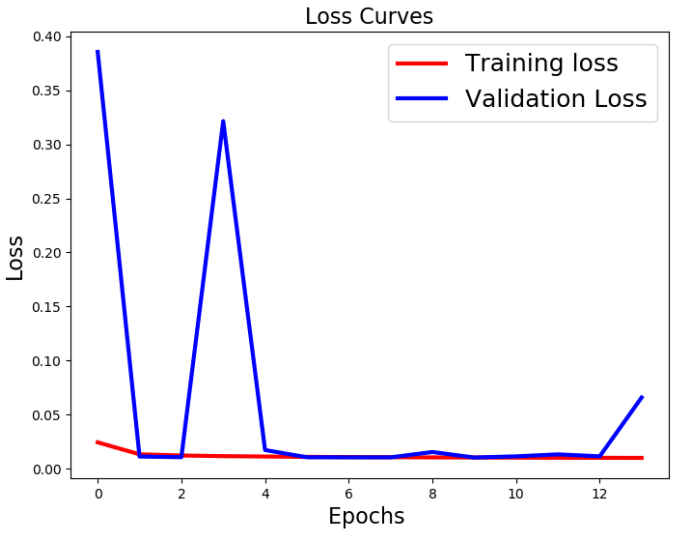
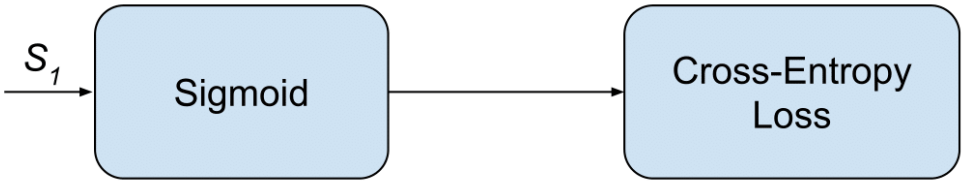


Figure 17- Training and Validation Loss Curves

4.7.2 Loss Function

For our loss function, it is standard to use the Binary Cross Entropy function. This uses a sigmoid function binary classification on each category. That is the decision will be to determine the probability of the input is in the state or not. Then we will calculate loss scores for each class independently of the other classes. The classes, in our case, are the 51 states. Figure 18 shows the formula and flow diagram for this the Binary Cross Entropy function. A good discussion on cross-entropy can be found in this paper. [15]



$$f(s_i) = \frac{1}{1 + e^{s_i}} \qquad CE = -t_1 log(f(s_1)) + (1 - t_1) log(1 - f(s_1))$$

Figure 18 - Binary Cross Entropy

4.7.3 Epochs and Batch Size Effect on Optimization

The other hyperparameters Batch Size and Maximum Epochs control the mechanics of the training. Batch Size determines how many samples from the training set are passed into the CNN before the optimizer and loss calculations are done. The larger the batch size the more data is input into the neural network between gradient and loss calculation. The goal of a gradient descent optimization is to find the lowest minima of all weights in the neural network. As the optimization and loss calculations are relatively expensive operations using a large batch size means we need to calculate these functions less frequency resulting in faster training. However, this can also lead to missing some local minima, and overall lead to a less optimal model. Due to our large data set

which consists of training set of 556311 samples, would require higher training time with a smaller batch size. Because we are using GPU hardware accelerated libraries for our CNN model training, this reduces out training time. The training accuracy and loss curves, in Figure 16 and Figure 17 shows slight fluctuations after convergence after the 4th epoch.

4.7.4 GPU Acceleration

The strength of GPU acceleration is that we can calculate the optimization and losses faster. These are basically matrix operations, which can be done in parallel. We can leverage the power of GPU acceleration and will be able to choose a smaller batch size, thus leading to a potentially more optimized model. Because of this, we chose a batch size of 32. Because of this batch size, for our training data size of 556311 entries, we will calculate the loss and optimization of a total of 17384 times for all weights in the neural network. Each epoch in the training takes approximately 40 minutes per epoch, running on a Nvidia GTX-1070 video card. (Table 14). Without GPU acceleration, training with such a small block size would take approximately 75 minutes per epoch running on an Intel I7 processor with 16GB of system memory. (Table 15)

Table 14 - Epoch #1 of CNN Training (GPU, Nvidia GTX 1070)

```
Epoch 1/75

    1/17384 [..............................] - ETA: 15:55:46 - loss: 0.6931 - acc: 0.9808
    2/17384 [..............................] - ETA: 8:12:57 - loss: 0.8755 - acc: 0.7338
    5/17384 [..............................] - ETA: 3:20:05 - loss: 0.9696 - acc: 0.5987
    9/17384 [..............................] - ETA: 2:04:00 - loss: 0.9757 - acc: 0.5569
…
  222/17384 [..............................] - ETA: 38:52 - loss: 0.3538 - acc: 0.8466
   223/17384 [..............................] - ETA: 38:50 - loss: 0.3526 - acc: 0.8472
  224/17384 [..............................] - ETA: 38:48 - loss: 0.3514 - acc: 0.8478
  225/17384 [..............................] - ETA: 38:46 - loss: 0.3503 - acc: 0.8484
  226/17384 [..............................] - ETA: 38:44 - loss: 0.3491 - acc: 0.8490
  227/17384 [..............................] - ETA: 38:41 - loss: 0.3480 - acc: 0.8495
  228/17384 [..............................] - ETA: 38:43 - loss: 0.3468 - acc: 0.8501
       …
```

Table 15 - Epoch #1 of CNN Training (CPU, Intel I7, 16GB)

```
Epoch 1/75

    1/17384 [..............................] - ETA: 10:31:48 - loss: 0.6931 - acc: 0.9808
    2/17384 [..............................] - ETA: 6:07:17 - loss: 0.8753 - acc: 0.7338
    3/17384 [..............................] - ETA: 4:35:44 - loss: 0.9335 - acc: 0.6591
   4/17384 [..............................] - ETA: 3:49:41 - loss: 0.9600 - acc: 0.6194
    5/17384 [..............................] - ETA: 3:22:20 - loss: 0.9681 - acc: 0.5984
    6/17384 [..............................] - ETA: 3:03:12 - loss: 0.9742 - acc: 0.5828
    7/17384 [..............................] - ETA: 2:47:35 - loss: 0.9794 - acc: 0.5708
    8/17384 [..............................] - ETA: 2:35:36 - loss: 0.9780 - acc: 0.5648
…
  222/17384 [..............................] - ETA: 1:15:47 - loss: 0.3534 - acc: 0.8474
  223/17384 [..............................] - ETA: 1:15:45 - loss: 0.3522 - acc: 0.8480
  224/17384 [..............................] - ETA: 1:15:44 - loss: 0.3510 - acc: 0.8485
  225/17384 [..............................] - ETA: 1:15:42 - loss: 0.3499 - acc: 0.8491
  226/17384 [..............................] - ETA: 1:15:41 - loss: 0.3488 - acc: 0.8497
  227/17384 [..............................] - ETA: 1:15:39 - loss: 0.3476 - acc: 0.8503
  228/17384 [..............................] - ETA: 1:15:38 - loss: 0.3465 - acc: 0.8508
…
```

This represents an almost 2:1 ratio. Our CNN model took 12 epochs to converge (See Figure 16, and Figure 17) This translates to 8 hours to train the model on the full training set on GPU versus 15 hours on a CPU. That's a significant reduction in training time. It should be noted that the Intel I7 is considered the top end for CPU regarding performance, whereas the GTX-1070 video card is considered a midrange performing card. On this website[3], a benchmark of the top of the line GTX -1080ti is 53% faster than the GTX-1070. This is due to it having more memory, and processors. However, the price also scales at the same rate. With Nvidia about to release the next generation of RTX-2080ti cards, initial benchmarks show a 94% increase in speed. However, the cost of these cards, at $1000 USD, is a steep price to pay for that performance increase.

[3] http://gpu.userbenchmark.com/Compare/Nvidia-GTX-1080-Ti-vs-Nvidia-GTX-1070/3918vs3609

4.8 Model Cross-Validation

At the end or training the model, we used 10-fold cross validation, with hold out, in order to do a first pass verification of the trained model. In additional to that, we have also applied the model to training, validation, and test datasets, compared false positive, false negatives, correct, and incorrect predictions to further evaluate the model. A visual representation is shown in Figure 19.

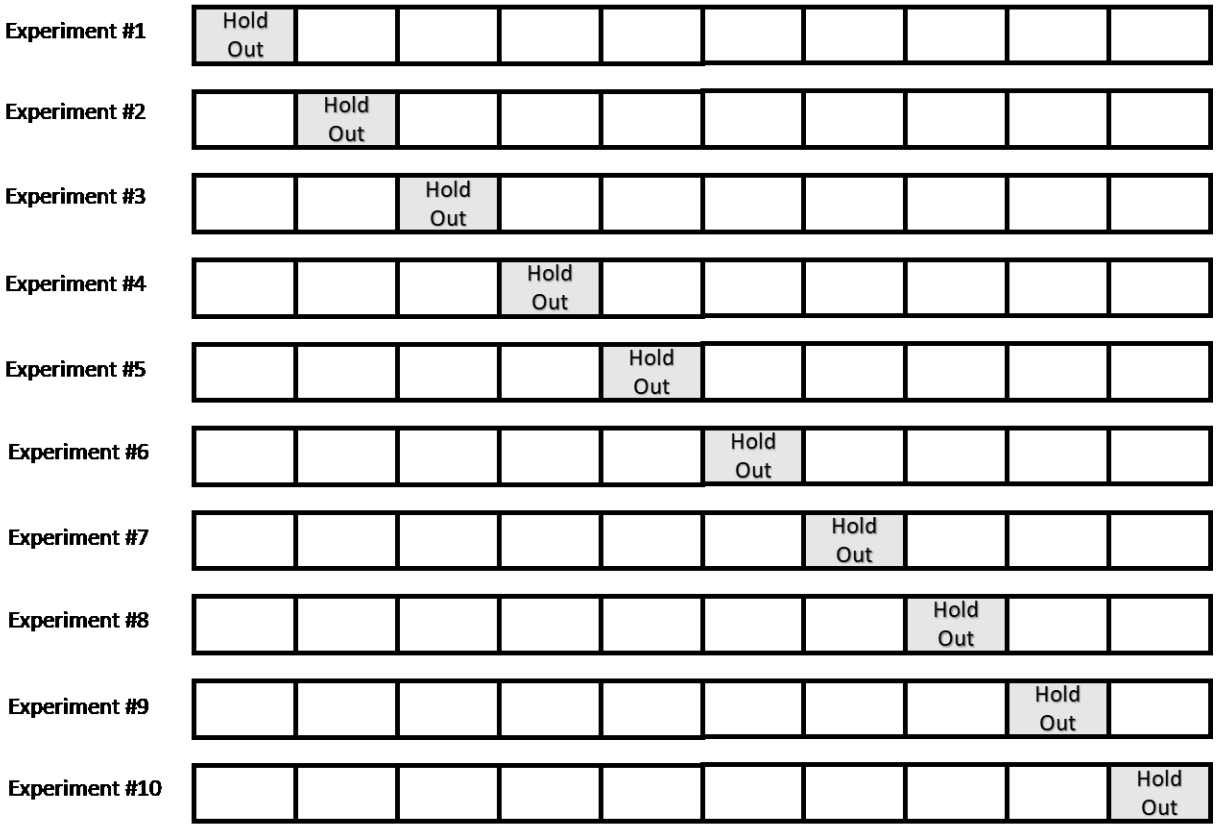


Figure 19 -10-Fold Cross-Validation with Holdout

4.8.1 Randomized Data

To ensure we each fold is a good representation of the total dataset, to avoid any skewing during validation, we have plotted out the histograms of city and state frequencies in the completed dataset, shown in Figure 20. This shows the frequency with which a state occurs in the cities list. In Figure 21 and Figure 22, we show the histogram for each of the holdout folds from fold 1 to fold 10. From these histograms, we can conclude that the folds are representative of the overall dataset.

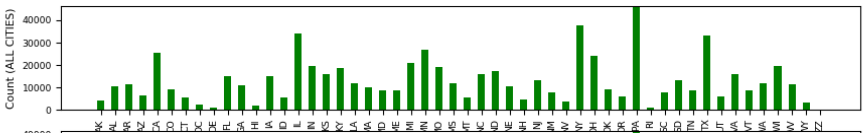


Figure 20 - Bigram Frequency (All Cities, All States)

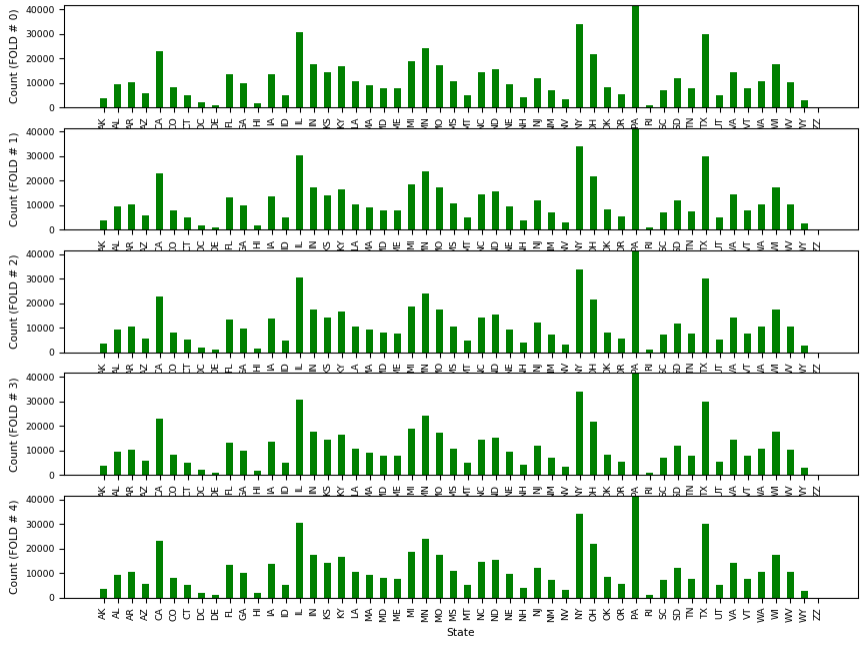


Figure 21- City Frequency Distribution by State (Folds 0 - 4)

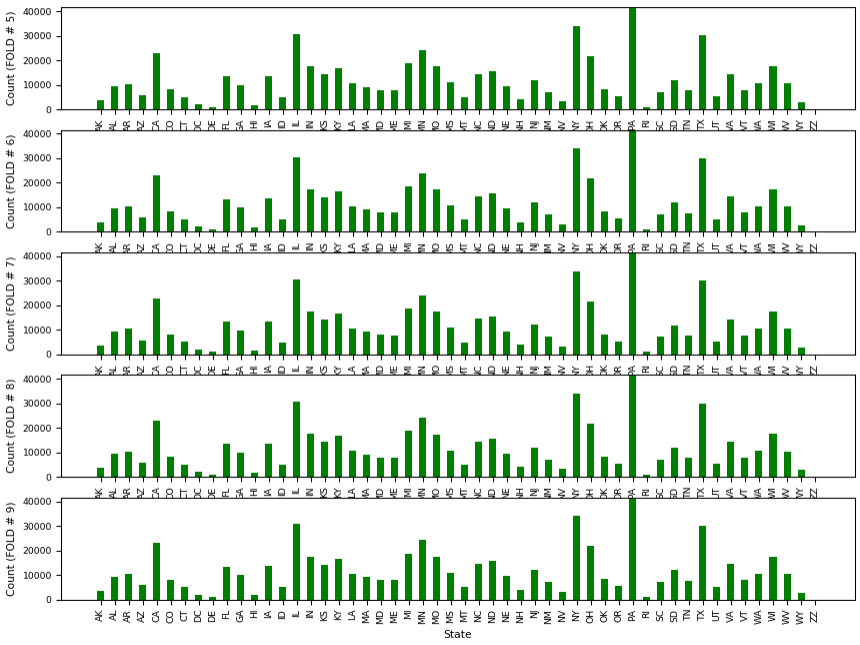


Figure 22- City Frequency Distribution by State (Folds 5 - 9)

4.8.2 Cross-Validation Results

The cross-validation accuracy results are presented in Table 16. The mean Cross-Validation accuracy is 99.74% with a standard deviation of +/- 0.00. This is an indication that the model behaves stably. As with the full training dataset, we are only limiting cross-validation to the 51 states which are eligible to vote in the 2016 Presidential Election.

4.8.3 Discussion on Overfitting

With overfitting we see that accuracy on training set is higher than validation set. In our training and validation accuracy graph we do not see this phenomenon. Additionally, we have employed a dropout layer between every convolution layer in order to mitigate overfitting

Table 16 – Results of 10-Fold Cross-Validation

Fold Number	Accuracy
1	99.74%
2	99.73%
3	99.75%
4	99.74%
5	99.74%
6	99.74%
7	99.74%
8	99.74%
9	99.74%
10	99.74%

5 Discussion

We'll look at the Model applied to each of the Datasets (Training, Validation, and Test) to gauge the model's accuracy before applying it to classify Twitter subscriber "location" field, in order to predict election outcome.

After determining our model to be stable and accurate on the labelled datasets, we apply the model on Twitter location fields. To judge the accuracy of the model on new data, we will perform some standard tests, including a T-Test to gauge the variance between predicted election results versus actual results. Finally, we will look at each state's election results and determine our accuracy in predicting election results for the two candidates.

5.1 Analytical Approach

First, we will run the model on the Test Dataset which we partitioned off from the main data. The test dataset will contain completely unseen data, which was not part of the training data or the validation data and is the first step to evaluating the model before applying it to classify the Twitter locations of subscribers. We will do a quantitative analysis by calculating the prediction rates and plotting the prediction rates for each dataset by state. Then we will do a qualitative analysis of the predictions.

5.2 Quantitative Analysis

We can see from Table 17 that the correct prediction rate on the test dataset is 84.4365%. The rest of the predictions are, including false positives, and false negatives, and wrong predictions. For the test dataset, we have a false positive rate, and a false negative rate of 1.0697% each and wrong predictions is at 12.2632%. This compares favourably to the training and validation dataset accuracy.

Table 17- Prediction Results

Prediction Type	Test	Training	Validation	Description
False Positives	1.0697%	1.5833%	1.7077%	Predictions with a probability greater than and equal to 50%, where the labelled state does not match with the predicted state.
False Negatives	1.6106%	2.0073%	1.1.5513%	Predictions with a probability less than 50%, where the labelled state matches the predicted state.
Wrong Prediction	12.2632%	11.4952%	12.0743%	Predictions with probability less than 50%, where the labelled state and predicted state do not match. (Note that this means the model assigned a state but was able to determine that there was a low probability.)
Correct Prediction	84.4365%	84.9142%	84.6667%	Predictions with probability greater than and equal to 50%, where labelled and predicted states match.

The criteria we use for false positives, false negatives, wrong, and correct predictions are also in Table 17.

The detailed analysis on what the overall prediction rate for the trained model on each of the datasets have been plotted in Figure 23, Figure 24 and Figure 25. Again, the prediction rates are stable between datasets, and not much variance is seen between states and datasets.

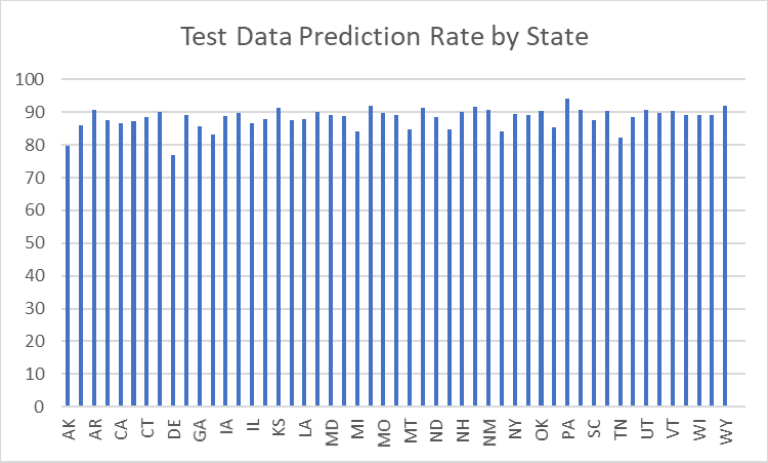


Figure 23 - Prediction Rate for Test Dataset

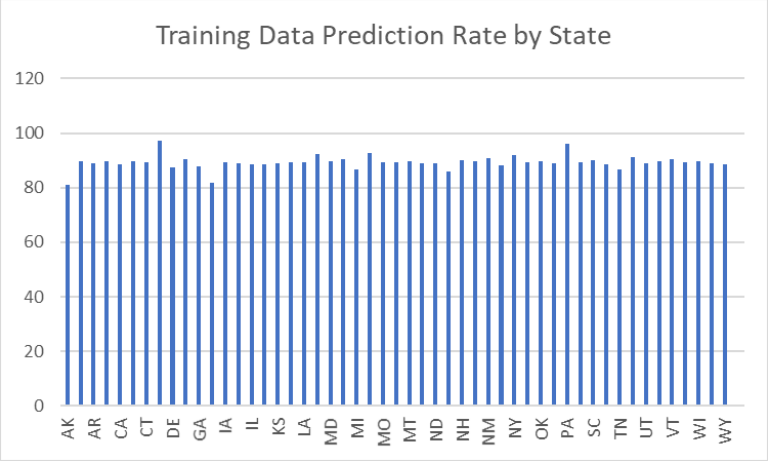


Figure 24 - Prediction Rate for Training Data

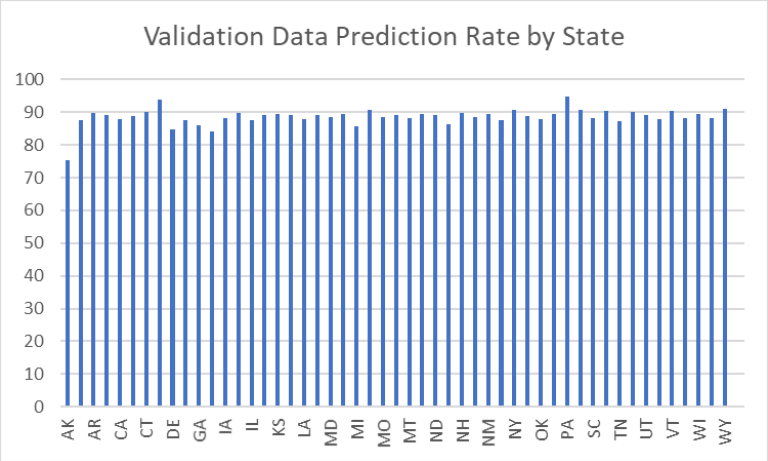


Figure 25 - Prediction Rate for Validation Dataset

5.2.1 False Positives

We have listed a sample of the false positives for the model, in Table 18. The majority of these are due to Puerto Rico appearing in the data to be classified. As stated in our hypothesis, the model may be able to learn a pattern based on a language affinity for cities in states. We can see indeed that the model appears to classify cities in Puerto Rico, most of the time in New Mexico. In Table 17, we see a few examples where Puerto Rico is classified as being in New Mexico. Both were settled originally by the Spanish, and thus have many cities with Spanish names. It does appear to some degree that it has learned that association. It's felt that an overall rate of under 2% for false positives is quite acceptable.

Table 18 - Test Dataset Predictions (False Positives)

	City|Labelled|Predicted|Match|P
False Positives	Hato Rey PR |PR|OR|False|98.76 Illinois Wanlock|IL|AL|False|100.00 Illinois, Coloma|IL|HI|False|100.00 Puerto Rico, Repto Ana Luisa|PR|NM|False|89.46 Puerto Rico Las Piedras |PR|NM|False|76.84 Bo Pueblito Nuevo |PR|NM|False|82.89 Illinois Meacham|IL|AZ|False|100.00 Tennessee, Belvidere|TN|AK|False|100.00 COAMO Puerto Rico |PR|CO|False|100.00 Saline, Michigan|MI|AK|False|100.00 PR URB Sagrado Corazon|PR|CO|False|73.75 Puerto Rico, Parc El Tuque|PR|NM|False|84.25 Pace Florida|FL|AK|False|100.00 Virginia, Bishop|VA|HI|False|100.00 Puerto Rico, URB Arbolada |PR|CO|False|61.53 Volo, Illinois|IL|HI|False|100.00 Oregon|WI|OR|False|96.95 Napier|WV|HI|False|98.85 Ext Bda Monserrate, PR|PR|MT|False|90.46 Keechi TX |TX|IL|False|100.00 Jard De Arroyo Puerto Rico|PR|NM|False|84.77 Puerto Rico Ext Santa Maria |PR|NM|False|88.33 Hide A Way|TX|HI|False|99.76 PR, Repto Daguey|PR|OR|False|78.66

5.2.2 False Negatives

We have listed samples of false negatives, in Table 19. Most cases are because the prediction probability is low relative our 50% cutoff rate. However, since the probability for the

classification is the argmax of all probabilities for all classes, the selected state could still be considered a successful classification. However, to be able to eliminate false positives, for unlabelled data, we decided to use a cutoff probability of 50% or greater to indicate a successful prediction. We need to do additional analysis to determine what an optimal cutoff rate should be without introducing too many false positives.

Another interesting observation is that most of the false negatives in this table are city names with no state associated with it. All the cities in our training set labelled by state, and a portion of the training data has been prepared using city names with no state. An indication that the model has learned the bigram embedding related to a state is the entry "Monterey Park |CA|CA|True|23.10", in false negatives. The prediction was 23.10% of it being in California, which is high enough to even consider it a correct prediction.

However, it is better to miss some successful classifications due to a high cutoff rate, rather than to allow incorrect classifications. Therefore, we will keep the cutoff rate at 50%. This provided us with a potential 84.4365% prediction rate for the Twitter location data. The overall rate of under 2% for false negatives is acceptable.

Table 19 - Test Dataset Predictions (False Negatives)

	City|Labelled|Predicted|Match|P
False Negatives	Halcott Center|NY|NY|True|4.49 Catharine |PA|PA|True|3.00 Jefferson Valley|NY|NY|True|6.53 Oceola, Ohio|OH|OH|True|34.36 DECATUR IN|IN|IN|True|17.61 Lake of the Woods |VA|VA|True|8.15 Buffalo |MN|MN|True|5.57 MO LAWRENCE |MO|MO|True|43.24 Nebraska, Verdon|NE|NE|True|0.61 Mosherville |PA|PA|True|6.65 Beverly Hts |PA|PA|True|11.48 Newton Lower Falls|MA|MA|True|11.92 Jax Naval Air |FL|FL|True|6.64 Wood Lake (Township)|MN|MN|True|19.25 VA Navy Mutual Aid Assoc|VA|VA|True|34.18 Llewellyn |PA|PA|True|4.88 S Westerlo|NY|NY|True|11.64 Alaska, Kongiganak|AK|AK|True|37.68 Elliott |IL|IL|True|4.22

	Rush Lake |MN|MN|True|10.44 Sunny Isl Bch |FL|FL|True|12.60 JOHNSON, NE |NE|NE|True|48.79 Madison Lake|MN|MN|True|5.86 Elizabethtown |PA|PA|True|5.36 LAFAYETTE, MS |MS|MS|True|27.65 SD POTTER |SD|SD|True|12.81 Mountain (CDP)|WI|WI|True|14.37 Monterey Park |CA|CA|True|23.10 Lower Oxford|PA|PA|True|13.43 Apollo Beach|FL|FL|True|10.33 Hidden Meadows|CA|CA|True|6.94 Terrell Hills |TX|TX|True|12.94 Old Mill Crk|IL|IL|True|5.02 VA, Zacata|VA|VA|True|0.34 Cadams, Nebraska|NE|NE|True|44.02 Indiana, Brunswick|IN|IN|True|48.64

5.2.3 Wrong Predictions

In Table 20, we provide a sample of wrong predictions by the model. Most of these wrong predictions are with low probability, in the single digits. This means our model is sufficiently strong and enables it to easily pick only correct predictions. A few high predictions, that are still below 50% are related to unseen data, such as Puerto Rico and is expected, since training data did not include the US territories. As a refinement to the model, it would be better to train on the non-voting territories, and then eliminate subscribers from the dataset instead. Overall, we feel that a wrong prediction rate of 12% for all three datasets is within an acceptable range, to provide accurate results for predicting the election.

Table 20 - Test Dataset Predictions (Wrong)

	City|Labelled|Predicted|Match|P
Wrong	SWEETWATER|WY|MT|False|5.38 St. Vincent College |PA|AK|False|3.08 Diamond Point |WA|ME|False|5.79 PR, URB Cambalache Ii |PR|CA|False|0.87 Los Padillas|NM|CA|False|13.34 Rouse |SD|PA|False|3.91 North Chester |MA|PA|False|5.83 Pala|CA|MO|False|3.39 Coeur D Alene |ID|NY|False|4.82 PR URB Guanajibo Homes|PR|UT|False|0.49 College Hl|OH|PA|False|5.03 Genoa Bluff |IA|GA|False|17.37 Welcome |NY|TX|False|3.28 Blue Earth City |MN|TX|False|6.55 Hineston|LA|PA|False|4.95 Bda Clausells |PR|PA|False|12.14 Cordova |TN|NE|False|3.64 Midway|TN|PA|False|3.58 Lakehurst Nae |NJ|MN|False|4.23 Natl Institute Stds & Tech|MD|TX|False|45.70 Kelly Ridge |CA|PA|False|5.69 Blue Mound (Township) |KS|IL|False|19.37 Puerto Rico URB Olivia Pk |PR|OH|False|38.90 Mcclure |SD|PA|False|10.33 DIVIDE|ND|WI|False|7.22 Sherman |SD|PA|False|5.39 URB Highland Gdns |PR|GA|False|16.06 Monterey|IN|CA|False|5.04 Portal|GA|PA|False|3.39 Alaska, Kalskag |AK|KS|False|0.00 Puerto Rico, URB Batey|PR|NM|False|27.78 Riverside |MD|PA|False|5.19 Comerica|MI|NY|False|3.70 S Hamilton|MA|NY|False|19.37 URB San Vicente Puerto Rico |PR|OH|False|8.38 Elgin |PA|IL|False|6.13

5.2.4 Correct Predictions

In Table 21, we list examples of correct predictions. The overall rate for correct predictions with the three datasets is 85%. We have successfully trained a machine learning classifier for state names. This is enough to enable us to use this trained model to predict the election.

Table 21 - Test Dataset Predictions (Correct)

	City|Labelled|Predicted|Match|P
Correct	Lakewood Harbor Texas |TX|TX|True|100.00 Broadbent OR|OR|OR|True|100.00 Motley MN |MN|MN|True|99.99 NY Greenwood Lk |NY|NY|True|100.00 Swan Lake Minnesota |MN|MN|True|100.00 Ohio, Wp Air Base |OH|OH|True|99.95 Minnesota, Biwabik|MN|MN|True|100.00 TX, Sanford |TX|TX|True|100.00 GA, Logistics & Distribution Ctr|GA|GA|True|100.00 Findlay IL|IL|IL|True|99.99 Fairview Heights Illinois |IL|IL|True|100.00 Forest Park, LA |LA|LA|True|95.53 TX Lane City|TX|TX|True|100.00 TN, Leach |TN|TN|True|100.00 ME Crouseville|ME|ME|True|100.00 California Twentynine Palms Mcb |CA|CA|True|100.00 SD, Iona|SD|SD|True|100.00 NJ, Boonton Township|NJ|NJ|True|100.00 Texas KLEBERG |TX|TX|True|100.00 Crews, TX |TX|TX|True|100.00 VA Horse Pasture|VA|VA|True|100.00 Social Security Administrat, MD |MD|MD|True|100.00 Defiance, Missouri|MO|MO|True|100.00 Florida, NY |NY|NY|True|99.96 NC, Milton|NC|NC|True|100.00 NY Bath |NY|NY|True|100.00 Leah, GA|GA|GA|True|100.00 AR Booker |AR|AR|True|100.00 Poindexter, KY|KY|KY|True|100.00 East Oakdale, CA|CA|CA|True|100.00 TX Sinton |TX|TX|True|100.00 VT Goose Green|VT|VT|True|100.00 MD Queensland |MD|MD|True|100.00 Twightwee Ohio|OH|OH|True|98.96 Bothell East WA |WA|WA|True|98.99 Missouri Labadie|MO|MO|True|100.00

Predicting 2016 US Election with Trained CNN Model

Now that we have trained a CNN model to classify locations into US states, we can apply the model to predict the election results for the Republican and Democratic candidates in the 2016 US Federal Election. The advantage using a CNN model versus simple pattern matcher, is that the CNN can provide us with a probability of match from location name to US state. With SoftMax, each class is assigned a probability with all classes summing up to 1.0. After the prediction we choose the maximum probability out of each of 51 state classes, and then apply the threshold of

>= 50% to decide on a state match. This is in contrast to using a pattern matcher which is a match or no match decision. In addition, using a CNN model, we can enable automatic model definition, as opposed to building a pattern matcher, which is a manual and time-consuming process.

5.3 Results Analysis

Here we do some basic statistical analysis to determine the accuracy of the CNN predications versus actual election results. We will do a F-Test to check for variances between the predicted and actual results, and then a T-Test to see if the mean difference determines whether the predicted results for each candidate are different from the actual results.

5.3.1 T-Test of Predicted vs Actual Results

One statistical technique we can use to check whether our prediction matches with the actual election results is the T-Test. The test allows us to confirm or reject whether the variance between our predictions is statistically significant to reject the model results.

Comparing both candidates Trump and Clinton predictions to their respective actual results show that the variance between prediction and actual are not equal. (See Figure 26, and Figure 28) Because of this, we will apply the T-Test for unequal variances, using Microsoft Excel, for the two samples.

In candidate Trump's case, the t-stat of 0.46517 is between the range of +/- t-critical value 1.99444. According to this result, we can say that the average difference between predicted results versus actual results is not dissimilar. Within statistical significance, the model has predicted a match with the actual result. Similar results are had for Clinton. The t-stat value -0.46517 falls between t-critical range of -1.99444 and 1.99444.

Both F-tests and T-tests have similar results indicating that the model is stable between the two Candidate datasets.

F-Test Two-Sample for Variances (Trump)		
	trump_actual	*trump_predict*
Mean	52.05640	51.12440
Variance	164.61617	36.09790
Observations	50.00000	50.00000
df	49.00000	49.00000
F	4.56027	
P(F<=f) one-tail	0.00000	
F Critical one-tail	1.60729	

Figure 26 – F-Test Actual vs Predicted (Trump)

t-Test: Two-Sample Assuming Unequal Variances (Trump)		
	trump_actual	*trump_predict*
Mean	52.05640	51.12440
Variance	164.61617	36.09790
Observations	50.00000	50.00000
Hypothesized Mean Difference	0.00000	
df	70.00000	
t Stat	0.46517	
P(T<=t) one-tail	0.32163	
t Critical one-tail	1.66691	
P(T<=t) two-tail	0.64325	
t Critical two-tail	1.99444	

Figure 27 - T-Test Actual vs Predicted (Trump)

F-Test Two-Sample for Variances (Clinton)		
	clinton_actual	*clinton_predict*
Mean	47.94360	48.87560
Variance	164.61617	36.09790
Observations	50.00000	50.00000
df	49.00000	49.00000
F	4.56027	
P(F<=f) one-tail	0.00000	
F Critical one-tail	1.60729	

Figure 28- F-Test Actual vs Predicted (Clinton)

t-Test: Two-Sample Assuming Unequal Variances (Clinton)		
	Predict	*Actual*
Mean	47.94360	48.87560
Variance	164.61617	36.09790
Observations	50.00000	50.00000
Hypothesized Mean Difference	0.00000	
df	70.00000	
t Stat	-0.46517	
P(T<=t) one-tail	0.32163	
t Critical one-tail	1.66691	
P(T<=t) two-tail	0.64325	
t Critical two-tail	1.99444	

Figure 29- T-Test Actual vs Predicted (Clinton)

5.3.2 Election Results Predicted vs Actual

For our Election Results analysis, we are drawing from two sources. The first one, Source #1, contains popular votes for all candidates and parties by state and county. (https://data.opendatasoft.com/explore/dataset/usa-2016-presidential-election-by-county@public/) We will use this source as the basis of our popular vote comparison by state. Of note is that this source was missing vote counts for the state of Alaska, so the analysis did not include this state.

The second one, Source #2, give us a second sanity check and provides the total popular vote by each candidate. An extracted table from this source is located in Figure 38 (https://www.270towin.com/2016_Election/)

We will resample the Twitter subscriber data so that the total Trump subscribers versus total Clinton subscribers will match closely with this ratio. Please see Figure 34 for details. The sample ratio for Trump vs Clinton is 48.9%. Since we had collected more subscriber entries from

Trump, we resampled Trump's data to match with this ratio of Trump popular votes to Clinton popular votes.

In Figure 35 we have tabulated the predicted election results and actual election results for the 2016 US Presidential Election. After summing the total votes received by candidates Clinton and Trump, we calculate the percentage of the total votes each candidate has received in each state. The "match" column indicates whether the predicted result matches with the actual result with '1' indicating a match and '0' indicating mismatch. The match rate is 90%, and the mismatch rate is 10%m with 45 matched and 5 missed. (discounting Alaska) These rates are slightly better than with our model results on the Test dataset, which had a prediction rate of 84% on unseen place names. (Figure 23).

To compare the prediction rates against actual, we have plotted the rates for each candidate in Figure 30, and Figure 31 for Trump and Clinton, respectively. The predicted rates track well, and the minima and maxima are a close match between the predicted curve and actual curve.

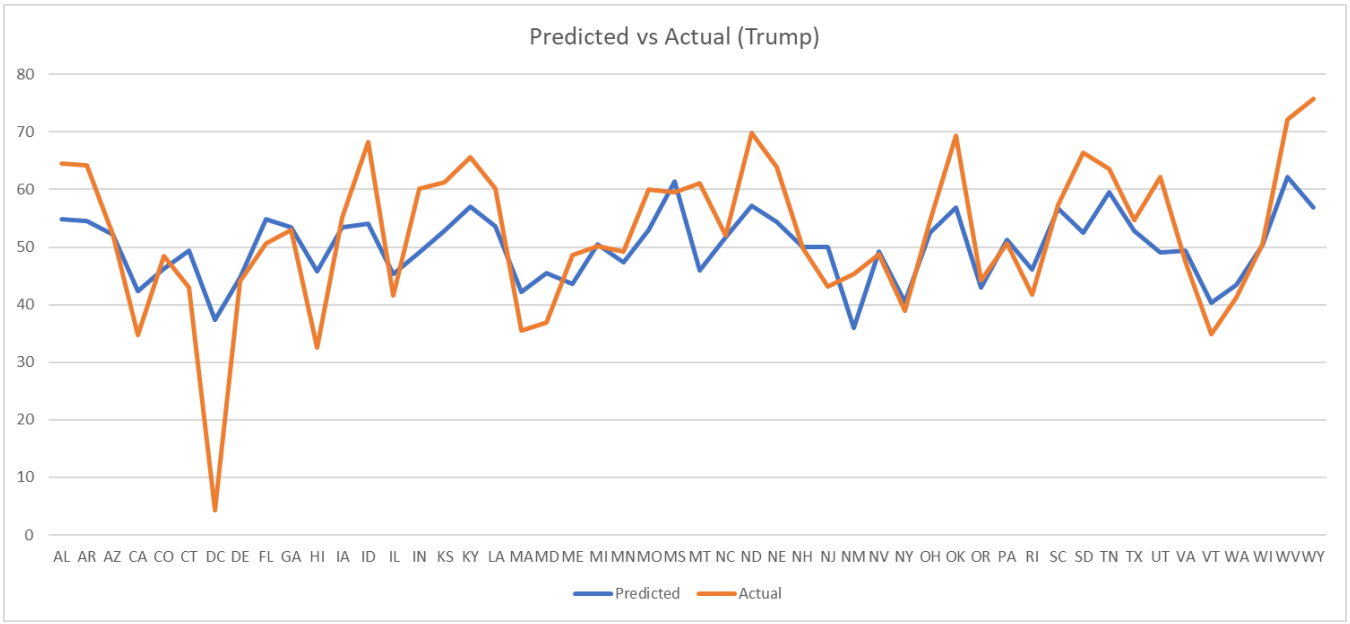


Figure 30- Predict vs Actual (Trump)

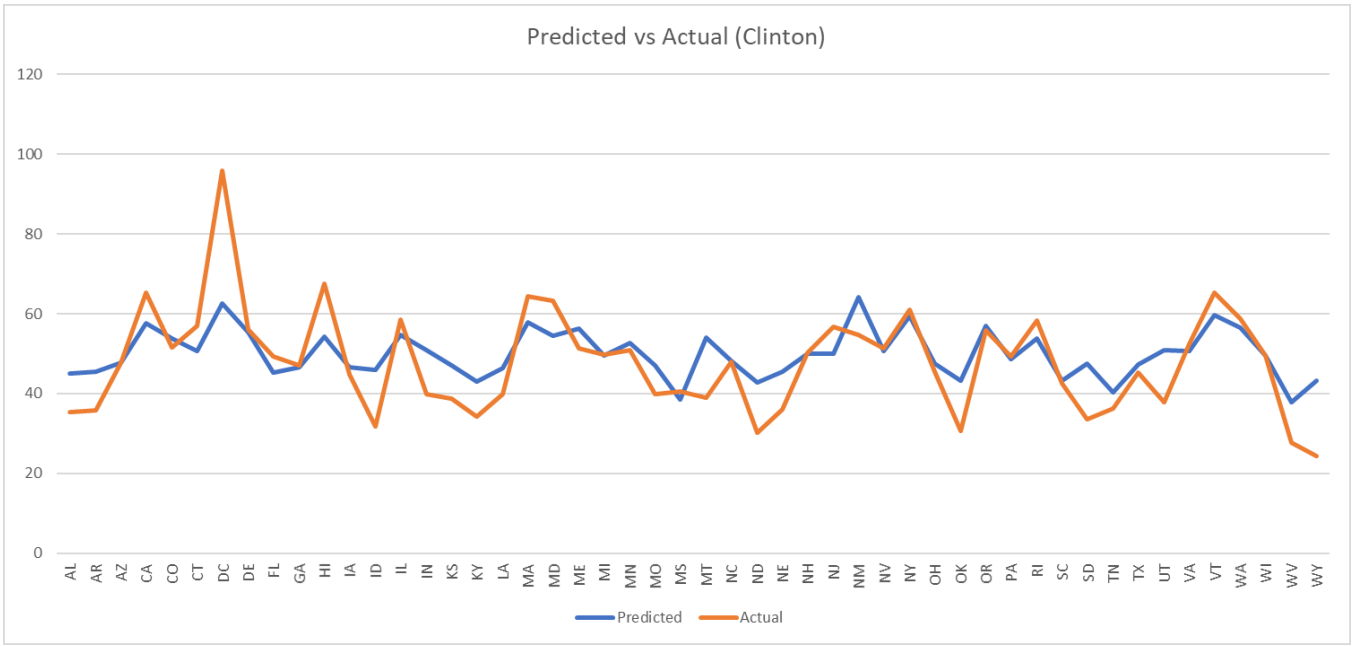


Figure 31 - Predict vs Actual (Clinton)

For further reference, the actual results are plotted separately in Figure 32, and Figure 33.

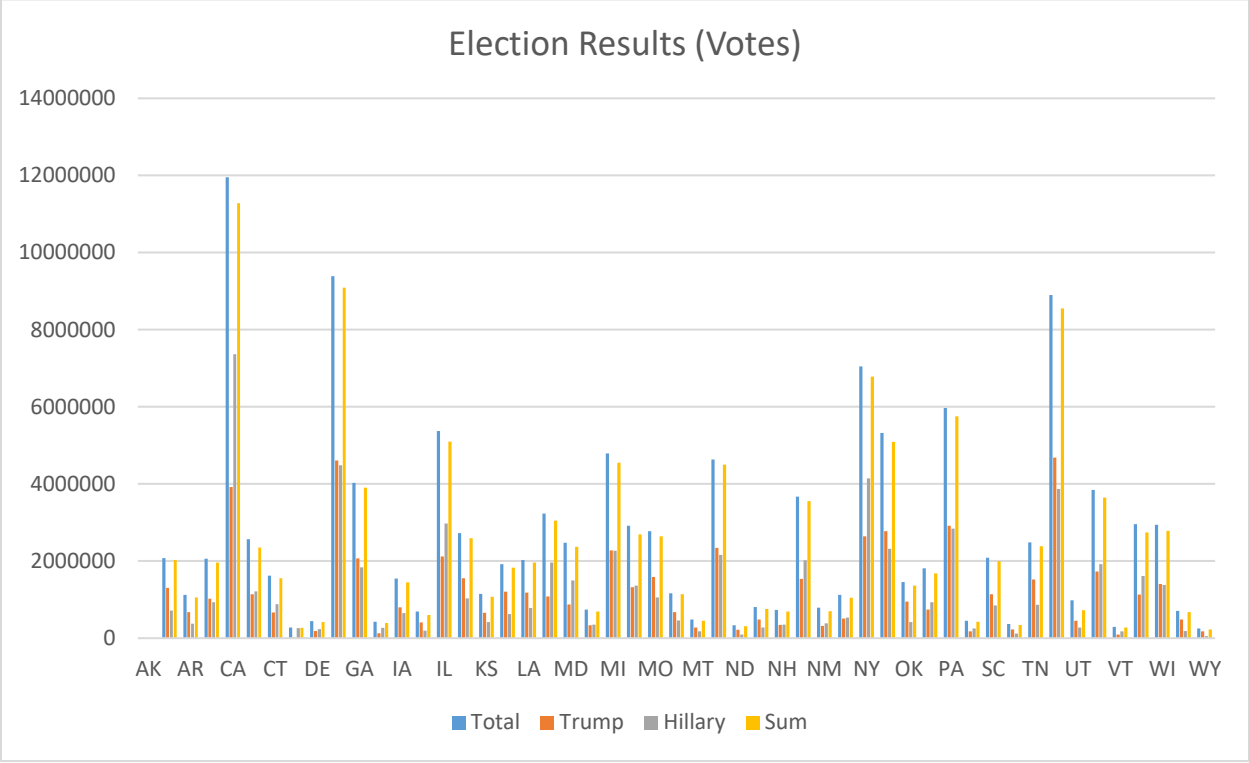


Figure 32- - 2016 Election Results (Votes)

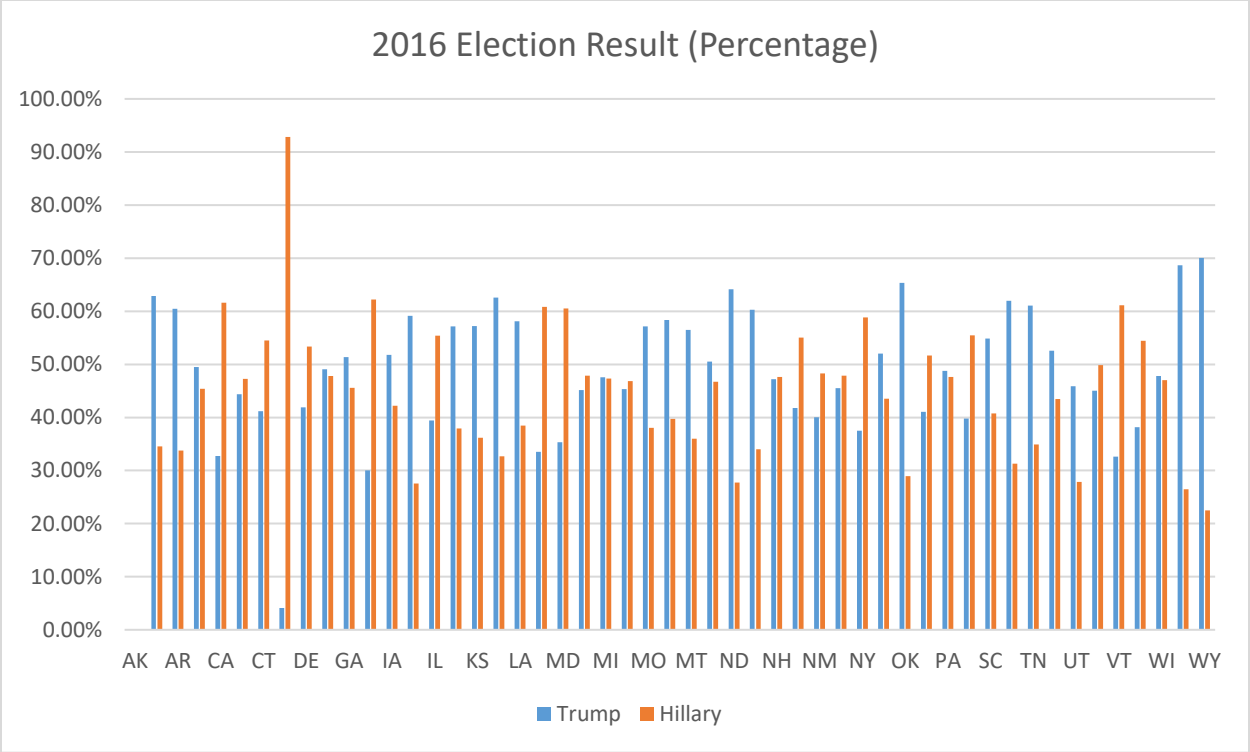


Figure 33 - 2016 Election Results (Percentage)

Finally, the model predicts overall that Trump won 29 states, while Clinton was predicted to win 21. Recall that we did not include Alaska, as there was insufficient data for that state in our source data. This is not too far off the actual results, which is 28 states for Trump and 22 states for Clinton.

	Predict Sample	Actual (Source1)	Actual (Source2)
Trump	387403	61064602	62980160
Clinton	404738	62426228	65845063
Percentage	48.91%	49.45%	49.45%

Figure 34 – Sample Ratio

state	trump_predict	trump_sub	clinton_predict	clinto_sub	winner_pred	trump_actual	votes_actual	clinton_actual	votes_actual2	winner_actu	match
AK	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
AL	54.91	9802	45.09	8050	trump	64.54	1306925	35.46	718084	trump	1
AR	54.5	4278	45.5	3571	trump	64.16	677904	35.84	378729	trump	1
AZ	52.19	9467	47.81	8671	trump	52.17	1021154	47.83	936250	trump	1
CA	42.29	30996	57.71	42306	clinton	34.72	3916209	65.28	7362490	clinton	1
CO	46.29	6420	53.71	7449	clinton	48.41	1137455	51.59	1212209	clinton	1
CT	49.35	3408	50.65	3498	clinton	43.04	668266	56.96	884432	clinton	1
DC	37.36	5799	62.64	9725	clinton	4.25	11553	95.75	260223	clinton	1
DE	44.69	1385	55.31	1714	clinton	44	185103	56	235581	clinton	1
FL	54.84	26000	45.16	21407	trump	50.66	4605515	49.34	4485745	trump	1
GA	53.4	11729	46.6	10235	trump	52.96	2068623	47.04	1837300	trump	1
HI	45.79	26794	54.21	31727	clinton	32.56	128815	67.44	266827	clinton	1
IA	53.44	3563	46.56	3104	trump	55.11	798923	44.89	650790	trump	1
ID	54.01	1327	45.99	1130	trump	68.22	407199	31.78	189677	trump	1
IL	45.35	10225	54.65	12323	clinton	41.57	2118179	58.43	2977498	clinton	1
IN	49.12	14118	50.88	14623	clinton	60.13	1556220	39.87	1031953	trump	0
KS	52.88	3701	47.12	3298	trump	61.26	656009	38.74	414788	trump	1
KY	56.97	5195	43.03	3924	trump	65.67	1202942	34.33	628834	trump	1
LA	53.53	7089	46.47	6155	trump	60.18	1178004	39.82	779535	trump	1
MA	42.17	6625	57.83	9086	clinton	35.54	1083069	64.46	1964768	clinton	1
MD	45.53	4929	54.47	5898	clinton	36.84	873646	63.16	1497951	clinton	1
ME	43.61	1774	56.39	2294	clinton	48.55	334838	51.45	354873	clinton	1
MI	50.43	8779	49.57	8628	trump	50.13	2279805	49.87	2268193	trump	1
MN	47.32	5273	52.68	5870	clinton	49.19	1322891	50.81	1366676	clinton	1
MO	53.01	6454	46.99	5722	trump	60.05	1585753	39.95	1054889	trump	1
MS	61.45	4003	38.55	2511	trump	59.49	678457	40.51	462001	trump	1
MT	45.99	2054	54.01	2412	clinton	61.1	274120	38.9	174521	trump	0
NC	51.77	13859	48.23	12913	trump	51.97	2339603	48.03	2162074	trump	1
ND	57.11	956	42.89	718	trump	69.8	216133	30.2	93526	trump	1
NE	54.44	1976	45.56	1654	trump	63.95	485819	36.05	273858	trump	1
NH	50.06	1306	49.94	1303	trump	49.8	345789	50.2	348521	clinton	0
NJ	50.01	8761	49.99	8758	trump	43.17	1535513	56.83	2021756	clinton	0
NM	35.9	2072	64.1	3699	clinton	45.35	315875	54.65	380724	clinton	1
NV	49.23	2955	50.77	3048	clinton	48.74	511319	51.26	537753	clinton	1
NY	40.52	23793	59.48	34927	clinton	38.92	2640570	61.08	4143874	clinton	1
OH	52.52	15158	47.48	13704	trump	54.47	2771984	45.53	2317001	trump	1
OK	56.8	5441	43.2	4139	trump	69.31	947934	30.69	419788	trump	1
OR	42.95	3820	57.05	5074	clinton	44.27	742506	55.73	934631	clinton	1
PA	51.29	12624	48.71	11989	trump	50.59	2912941	49.41	2844705	trump	1
RI	46.11	1267	53.89	1481	clinton	41.79	179421	58.21	249902	clinton	1
SC	56.7	5681	43.3	4339	trump	57.38	1143611	42.62	849469	trump	1
SD	52.53	1016	47.47	918	trump	66.43	227460	33.57	114938	trump	1
TN	59.58	9009	40.42	6113	trump	63.64	1517402	36.36	867110	trump	1
TX	52.8	31157	47.2	27852	trump	54.76	4681590	45.24	3867816	trump	1
UT	49.07	2857	50.93	2965	clinton	62.25	452086	37.75	274188	trump	0
VA	49.41	8374	50.59	8573	clinton	47.45	1731156	52.55	1916845	clinton	1
VT	40.32	577	59.68	854	clinton	34.79	95053	65.21	178179	clinton	1
WA	43.4	6260	56.6	8165	clinton	41.21	1129120	58.79	1610524	clinton	1
WI	50.38	4490	49.62	4422	trump	50.41	1403694	49.59	1380823	trump	1
WV	62.12	2222	37.88	1355	trump	72.17	486198	27.83	187457	trump	1
WY	56.85	585	43.15	444	trump	75.7	174248	24.3	55949	trump	1

Figure 35 - Election Results Predicted vs Actual

Column1	# States (Pred)	# States (Act)
Trump Wins	28	29
Clinton Wins	22	21

Figure 36 - Overall Winner

Matched	45
Mismatched	5
% Matched	90%

Figure 37 - Prediction Rate for Election 2016

	Candidate	Party	Electoral Votes	Popular Votes
	Donald J. Trump	Republican	304	62,980,160
	Hillary R. Clinton	Democratic	227	65,845,063
	Gary Johnson	Libertarian	0	4,488,931
	Jill Stein	Green	0	1,457,050
	Evan McMullin	Independent	0	728,830

Figure 38- Actual Total Votes by Candidate[4]

[4] https://www.270towin.com/2016_Election

6 Conclusion

We feel this project has been successful. The major finding is that we can reframe a traditional text pattern matching task into an image recognition task. We have also successfully applied a trained machine learning model to help predict locations of Twitter subscribers of the two major candidates in the 2016 US general election, and have our predicted results match actual results. However, the model needs to be refined further to prevent breaking down, in cases that location names do not carry enough clues for classification.

6.1 Finding #1 – Reframed Text Classification into Image Recognition

Our primary finding from this experiment is that it is, in fact, possible to reframe a traditional text classification problem into an image classification one for a CNN to train and build a model for multiclass classification. In our experiment, we collected the location field from Twitter subscriber accounts and converted them to bigrams. Then we encoded them into images for our CNN model. This gives us the advantage of training with GPU hardware acceleration for higher speed, by allowing us to use smaller training batch sizes, and slower learning rates for a more optimized model at the end of training.

In keeping with CNN performance in general (Table 4), our model accuracy is very high. Generating predictions on unseen data, such as the test dataset. When not excluding the non-voting US territories, we get an accuracy rate of 84% (Table 17). Since we did not train on Puerto Rico, which is a territory not eligible for voting in the election, this lower rate is to be expected. Our

cross-validation accuracy on the training dataset, which does exclude non-voting territories, show a stable model with a mean accuracy of 99.74%. (Table 16)

In Figure 36, we can see the model has predicted a Trump win of 29 states to Clinton's 21. Close match to the actual results where Trump wins with 28r and Clinton garners 22. We have removed Alaska due to insufficient data.

It can be concluded that the model is stable and can, in fact, learn to classify Twitter subscriber locations into US states using a Convolutional Neural Network.

6.2 Finding #2 – Model Successfully Applied to Election Prediction

After satisfying ourselves on the accuracy of the model, we applied our model to classifying Twitter subscriber locations by state with the goal of predicting the 2016 Presidential Election outcome. Our analysis shows a 90% correct prediction rate when compared against actual election results, by state. All the large population states such as California, New York and several swing states such as Pennsylvania, Wisconsin, and Florida were also predicted correctly. It can be concluded that, in fact, the trained CNN model was able to predict the US 2016 Presidential Election, with regards to the popular votes.

It should be noted that prediction by popular votes is a simplification of the US election process. In fact, the US uses an Electoral College system whereby members of each state casts a vote to decide the winner. Whichever candidate wins the most Electoral College votes is declared the winner. The votes of the electoral college will typically be correlated with popular votes cast in their respective states. Since our model predicts the winner by state, it should correlate closely to the Electoral College votes.

6.3 Finding #3 – Model Breaks Down when Cues are Insufficient

Where the model breaks down, is when a location does not contain any cues regarding the state that it is in. However, not in all cases. With a 90% correct prediction rate on unlabelled data collected from the candidates' Twitter subscribers, we feel the model is sufficiently accurate for an election prediction. Even humans would not be able to select a state by city name alone, without additional context.

6.4 Future Work

Although the current model is behaving stably, and with high accuracy, there are places where it will break down. In Finding #3, when place names contain no clue for the state, the model accuracy drops. In the future, it might help to boost model performance, and stability, by adding more information from the Twitter Subscriber Profile, such as the name of the subscriber, and the description field.

With the 2020 election one year away, it would be interesting to apply the model to a new set of subscribers before the election and use the model to predict election results before the 2020 election. At this point we still don't know the Candidates on both sides, so we will have to wait. From the model design aspect, we can continue to tune the layer configuration and hyperparameters to increase the model accuracy and reduce the training time.

6.4.1 Application of Model for Campaign Intelligence

By gathering the subscriber profiles of a candidate, and applying the trained model, a party can determine where their supporters located, and where they their support is light. Such an analysis can also be applied to their opponents to determine where extra campaigning needs to be done.

6.4.2 Application for NLP

In this work, we have enabled a CNN to learn the bigram embedding relative to city and state names. However, it's felt that we can tokenize words from sentences and use a CNN to learn the embedding between individual words as well. This would be like Word2Vec from Google.

7 Appendixes

7.1 Appendix A

Image Processing using Histogram of Gradients method (HOG)

A common method used as the first step in image processing is to plot a histogram of pixel intensities as a histogram to identify similarities and differences between images. To go one step further a common technique in image recognition is called Histogram of Oriented Gradients (HOG), which calculates for each X,Y coordinates, within a sliding window of cells and blocks covering the entire image. Then a 1-D Sobel operator is applied to the values in the cells.

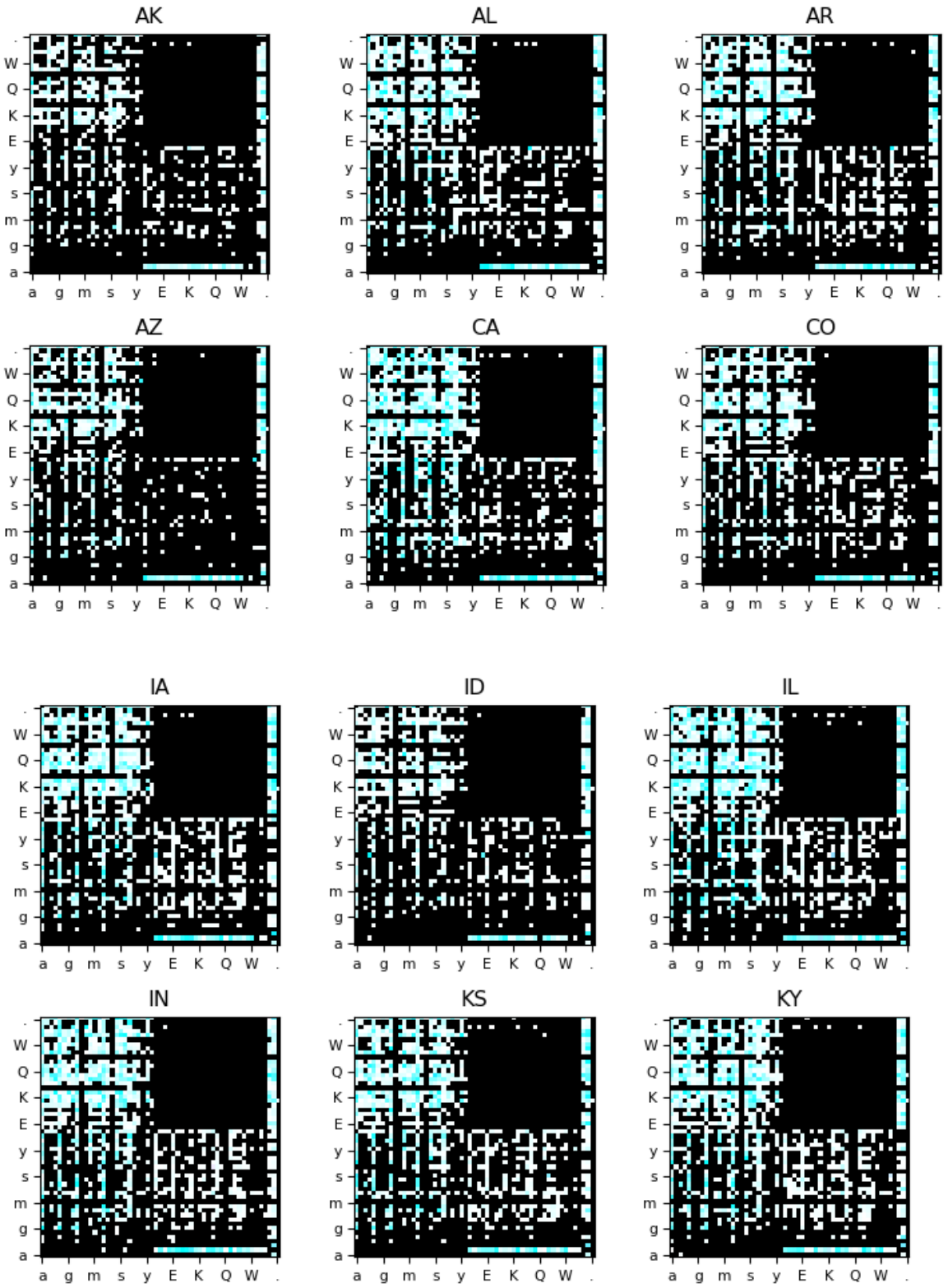
Equation 1 - HOG Operator

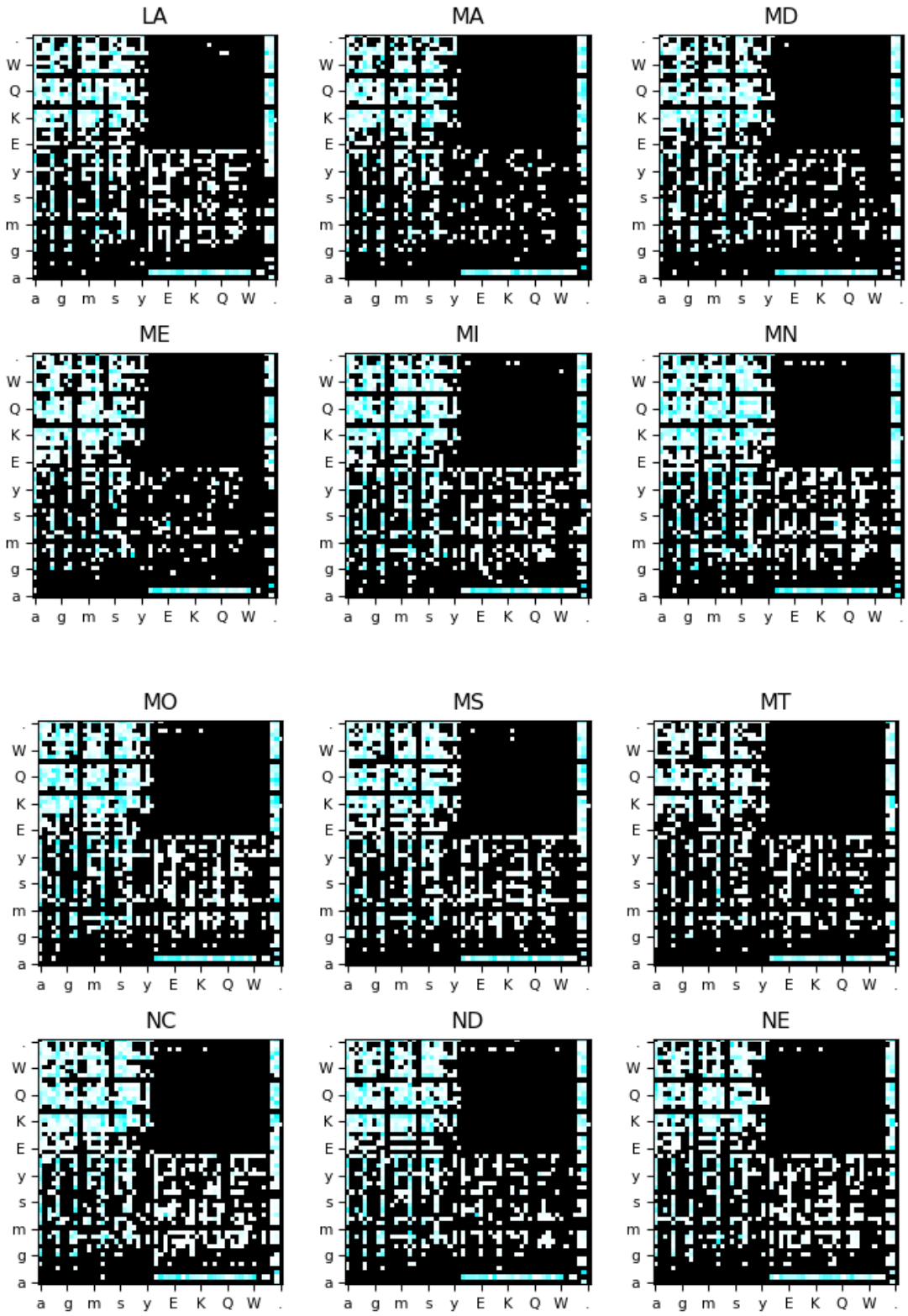
$$G(y,x) = \sqrt{G_x(y,x)^2 + G_y(y,x)^2}, \theta(y,x) = \arctan\left(\frac{G_y(y,x)}{G_x(y,x)}\right)$$

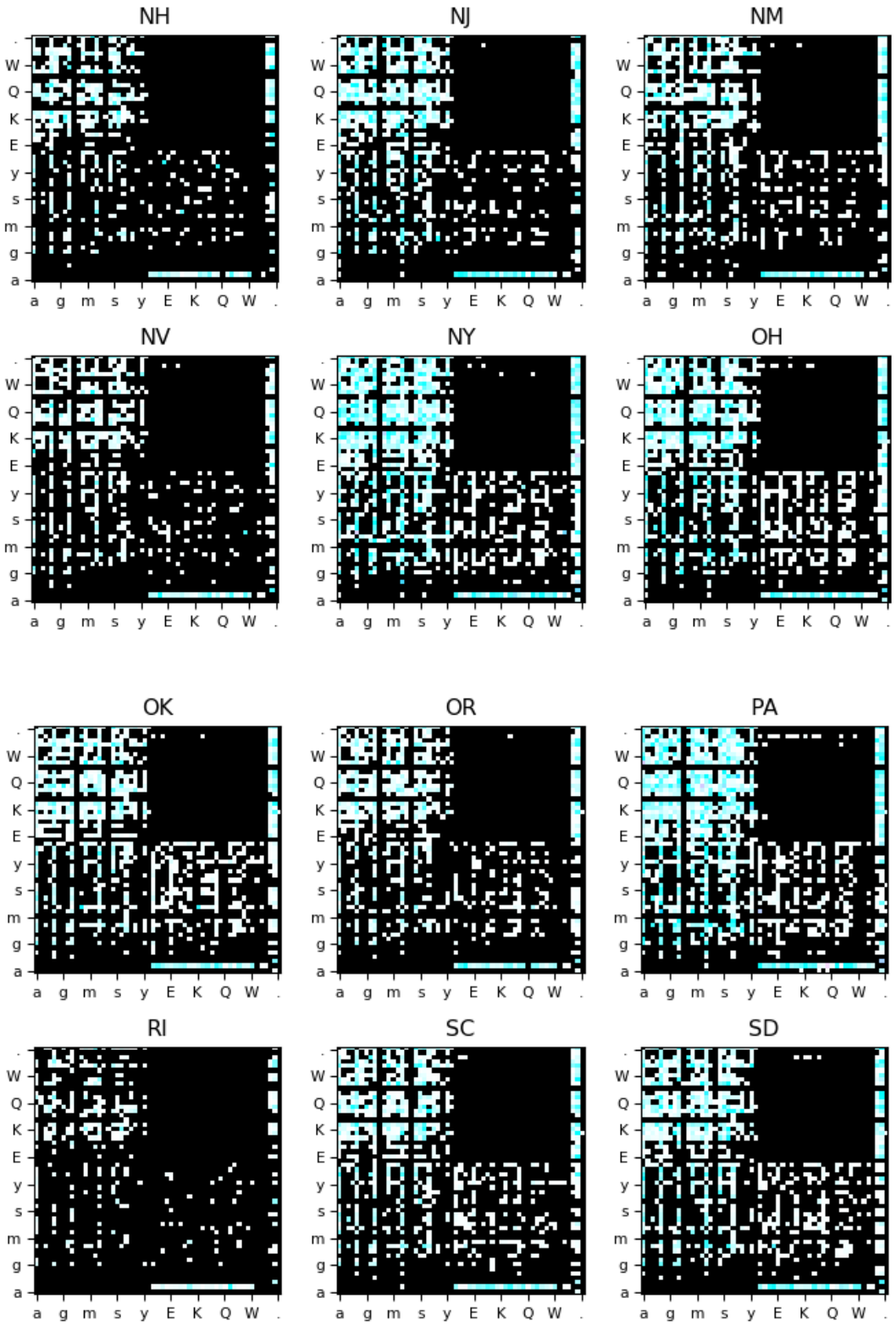
7.2 Appendix B

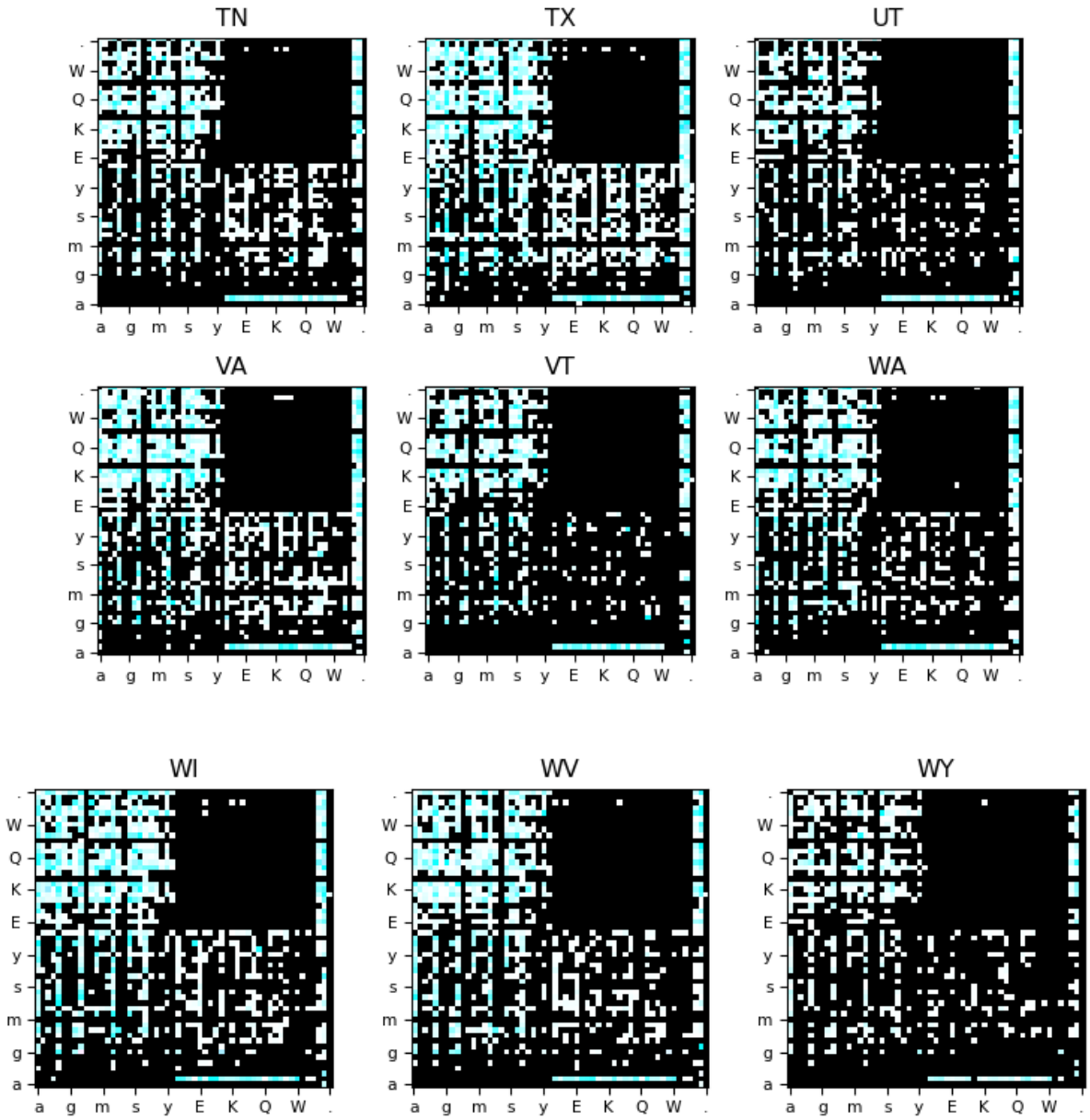
Image Representation of US States

These are the bigram frequencies of each state represented as images. These images are a composite of all bigrams of all cities in each respective state. The individual cities are fed into the CNN and are all correctly labelled for the state from which they belong. The x and y-axes are the two-letter components of the bigram. Each point represents the frequency of each bigram with colour intensity representing the frequency of the bigram.


AK
AL
AR
AZ
CA
CO
IA
ID
IL
IN
KS
KY
. W Q K E y s m g a
a g m s y E K Q W .


LA
MA
MD
ME
MI
MN
MO
MS
MT
NC
ND
NE
. W Q K E y s m g a
a g m s y E K Q W .

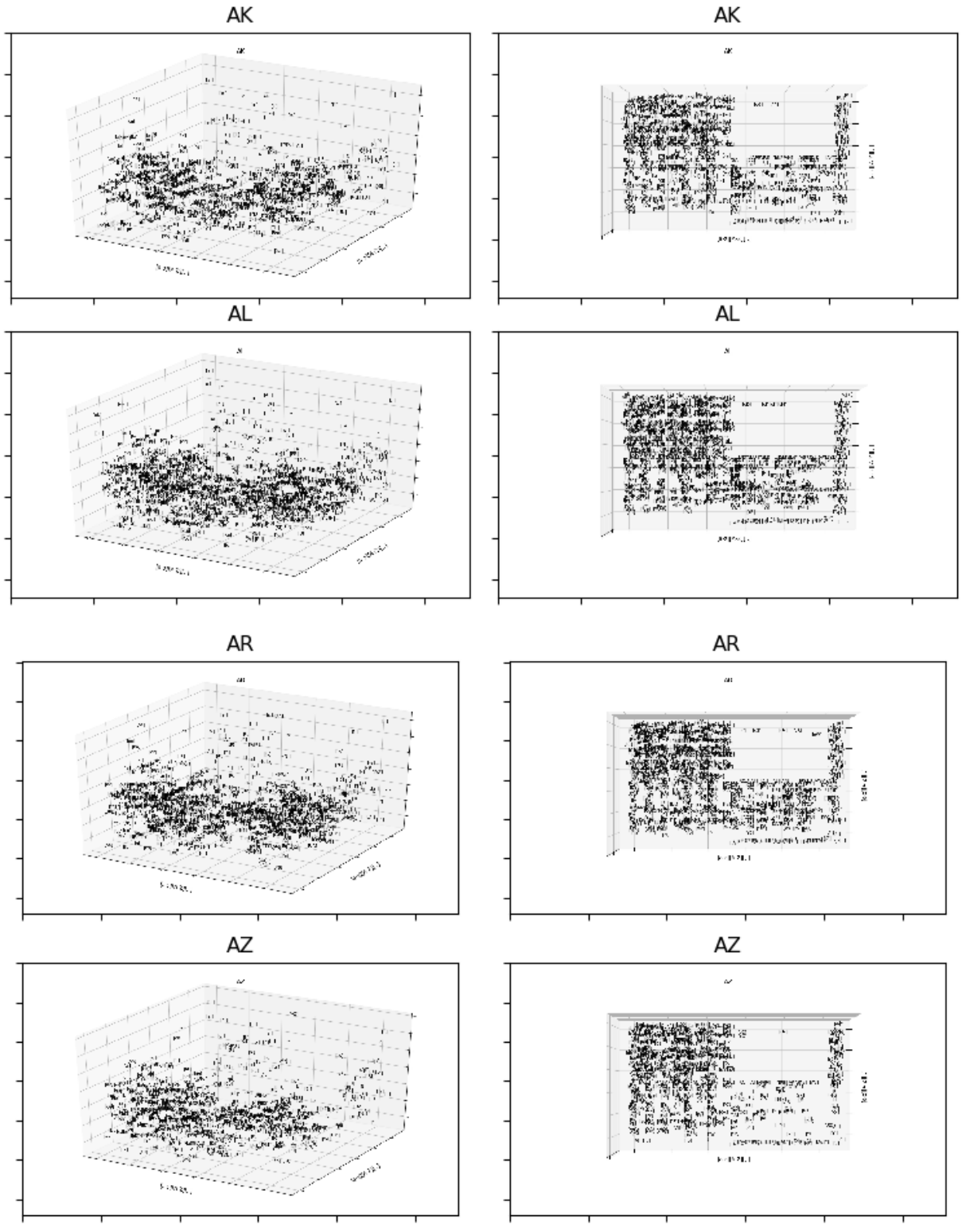

NH
NJ
NM
NV
NY
OH
OK
OR
PA
RI
SC
SD
. W Q K E y s m g a
a g m s y E K Q W .

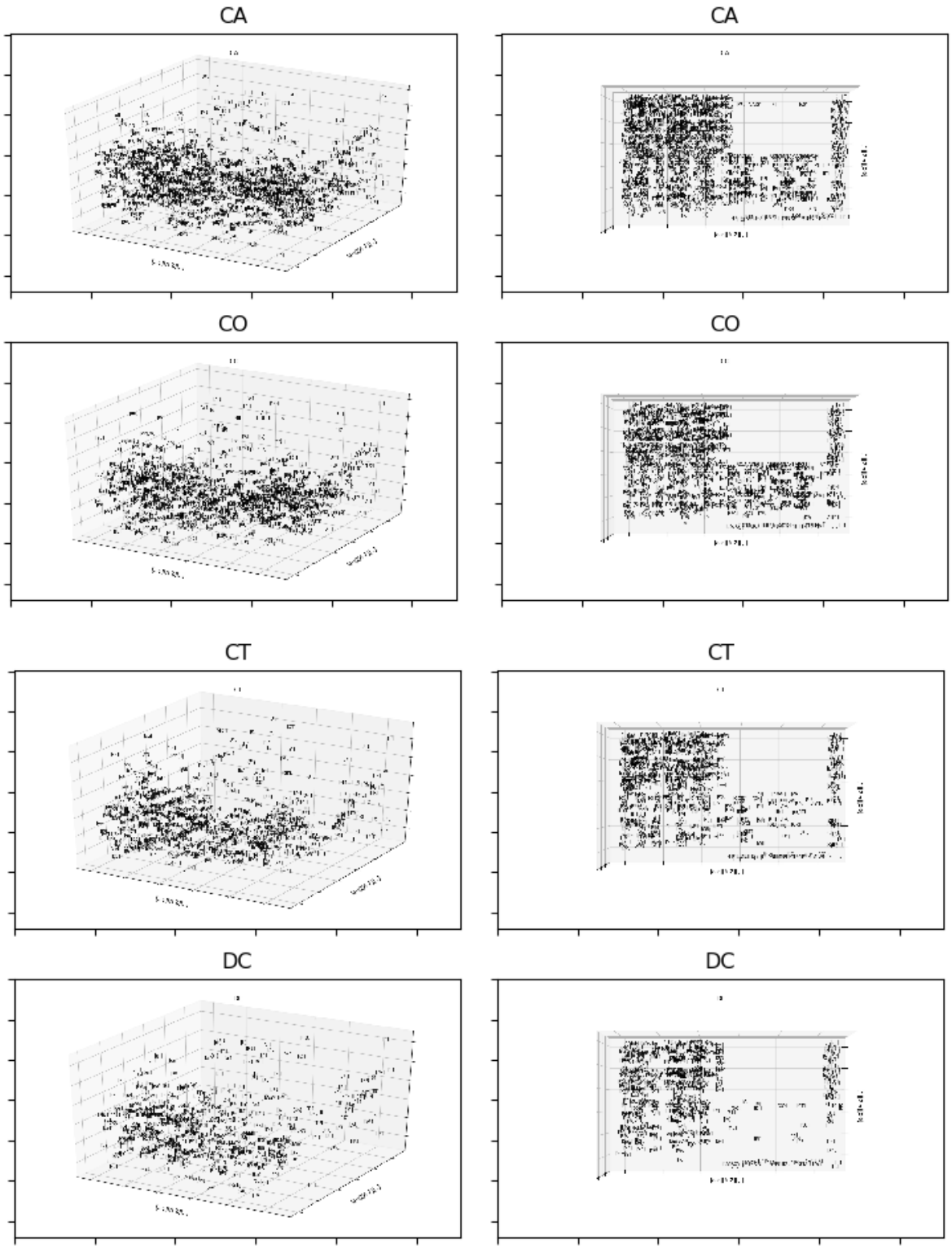

TN
TX
UT
VA
VT
WA
WI
WV
WY
. W Q K E y s m g a
a g m s y E K Q W .

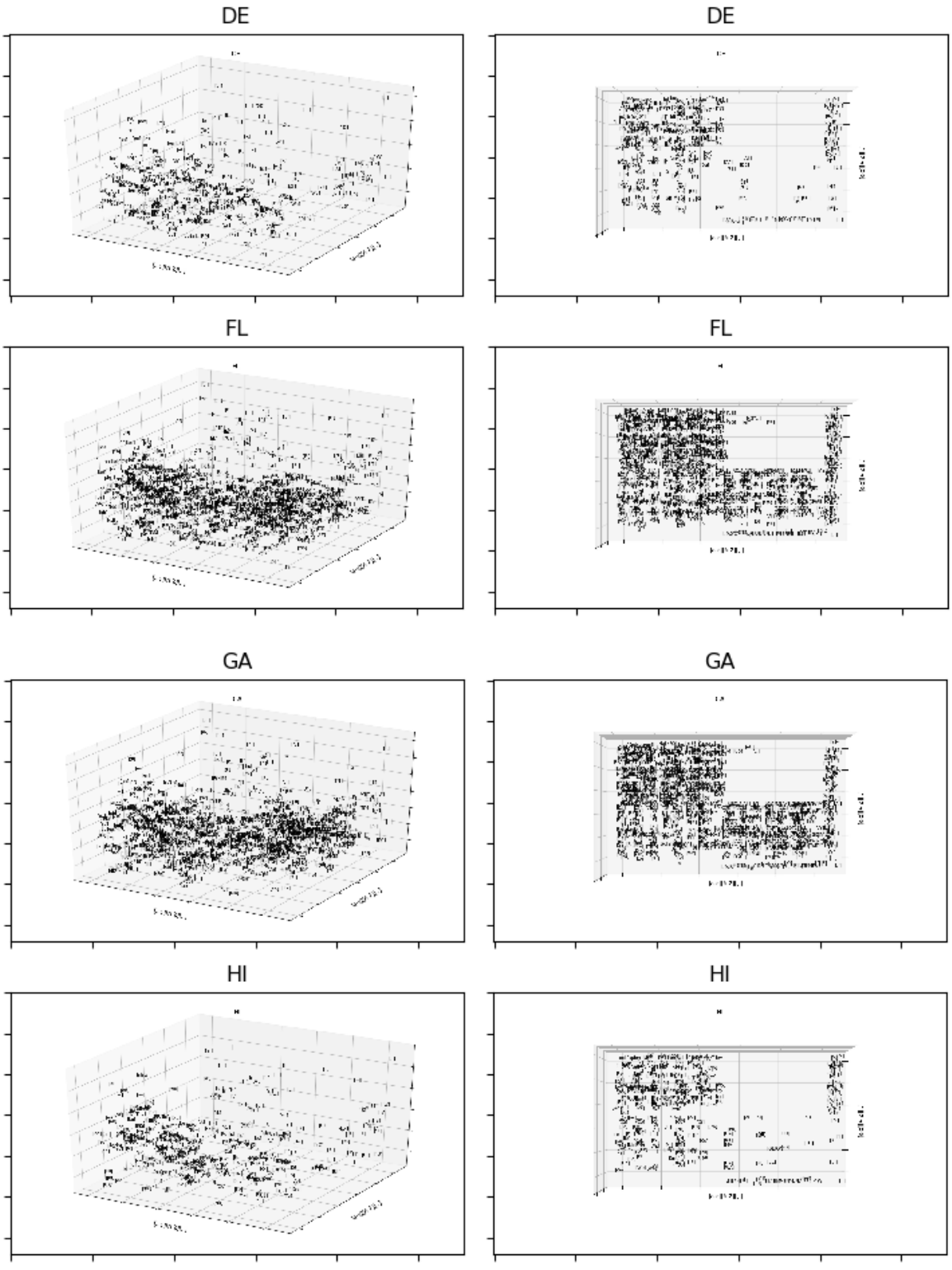
7.3 Appendix C

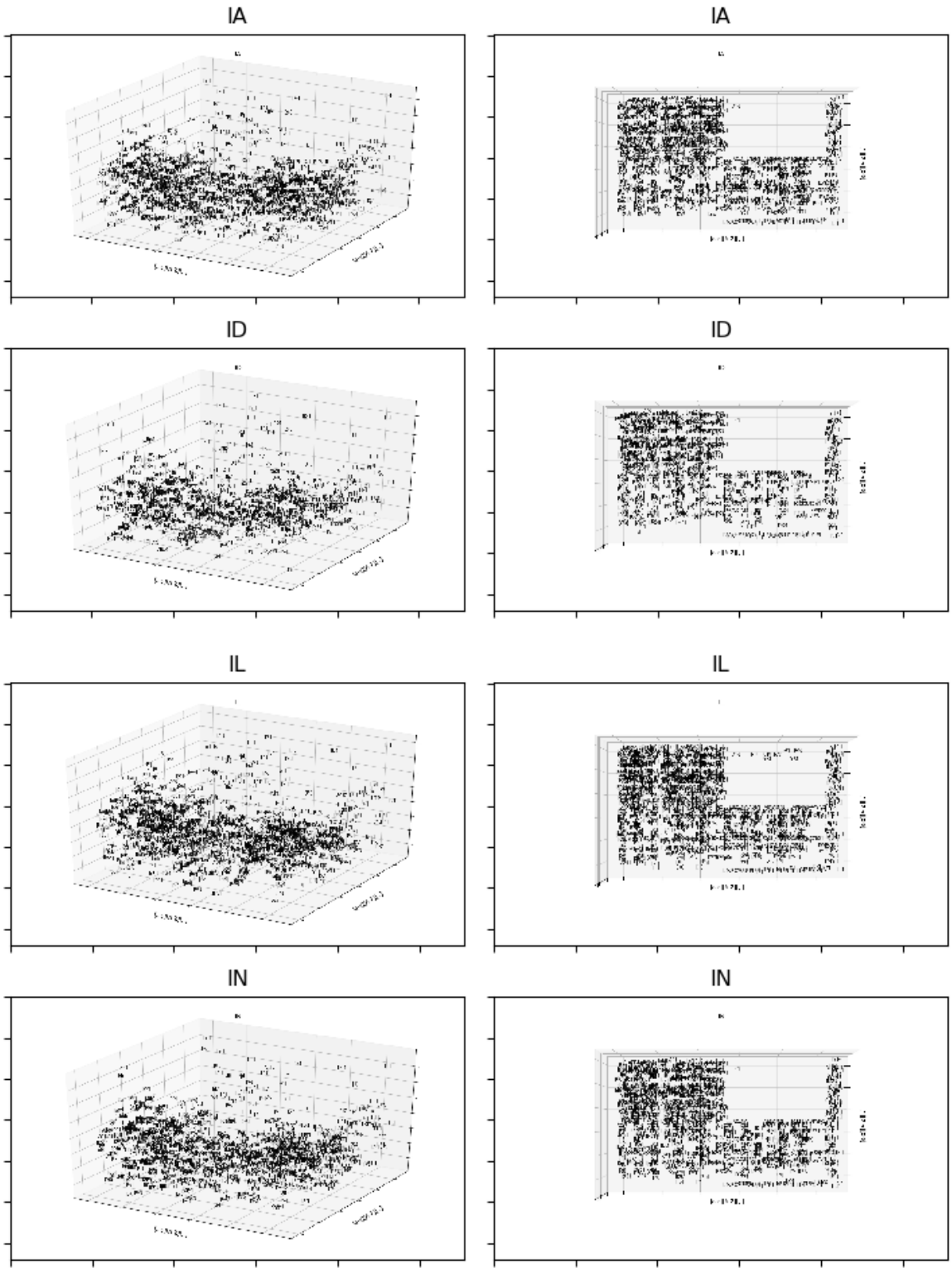
Sample 3D Plots of US States

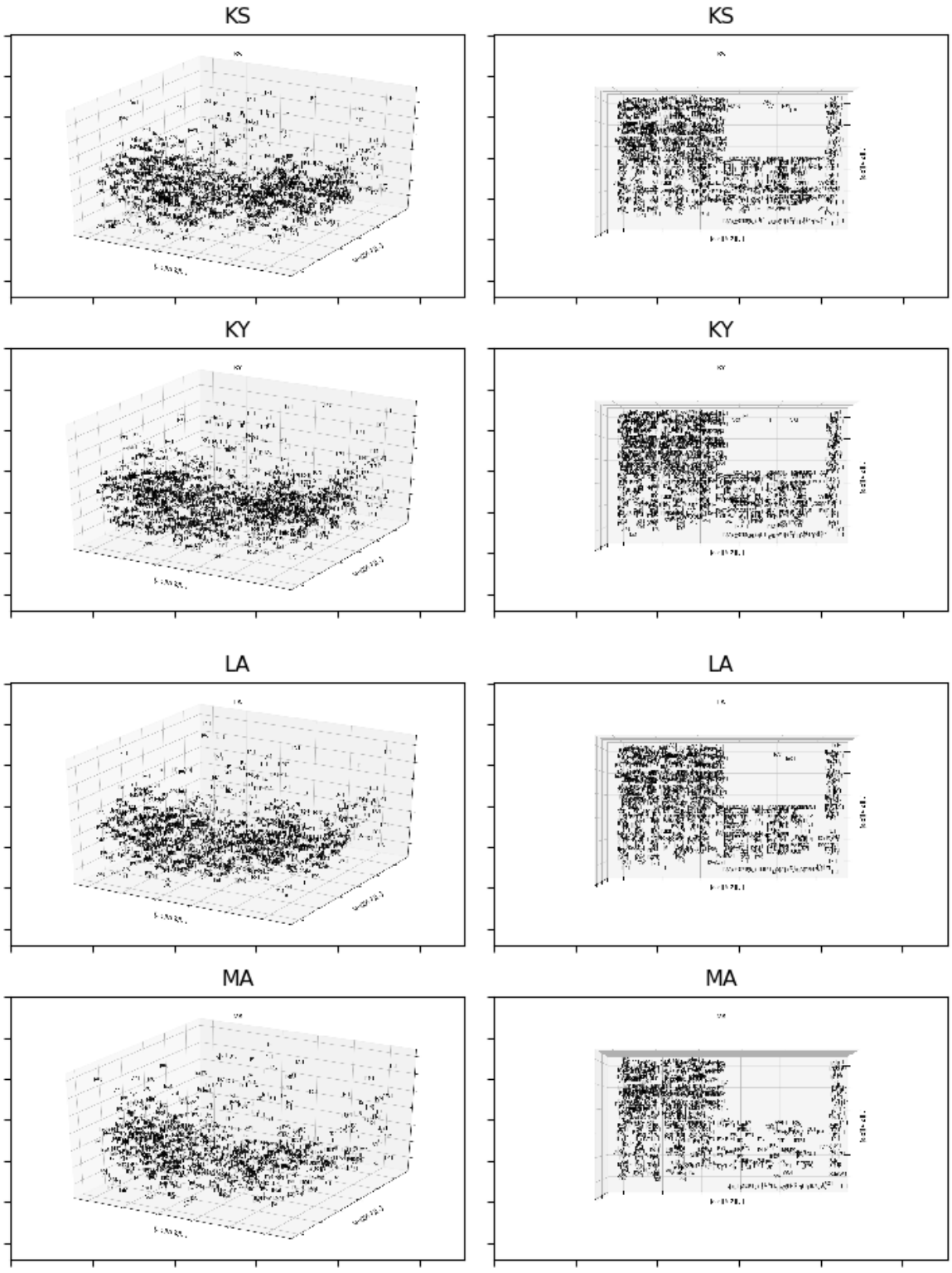
These are the 3-D representations of the bigrams, with the axes again representing the letter component of each bigram with the z-axis representing the bigram frequency.

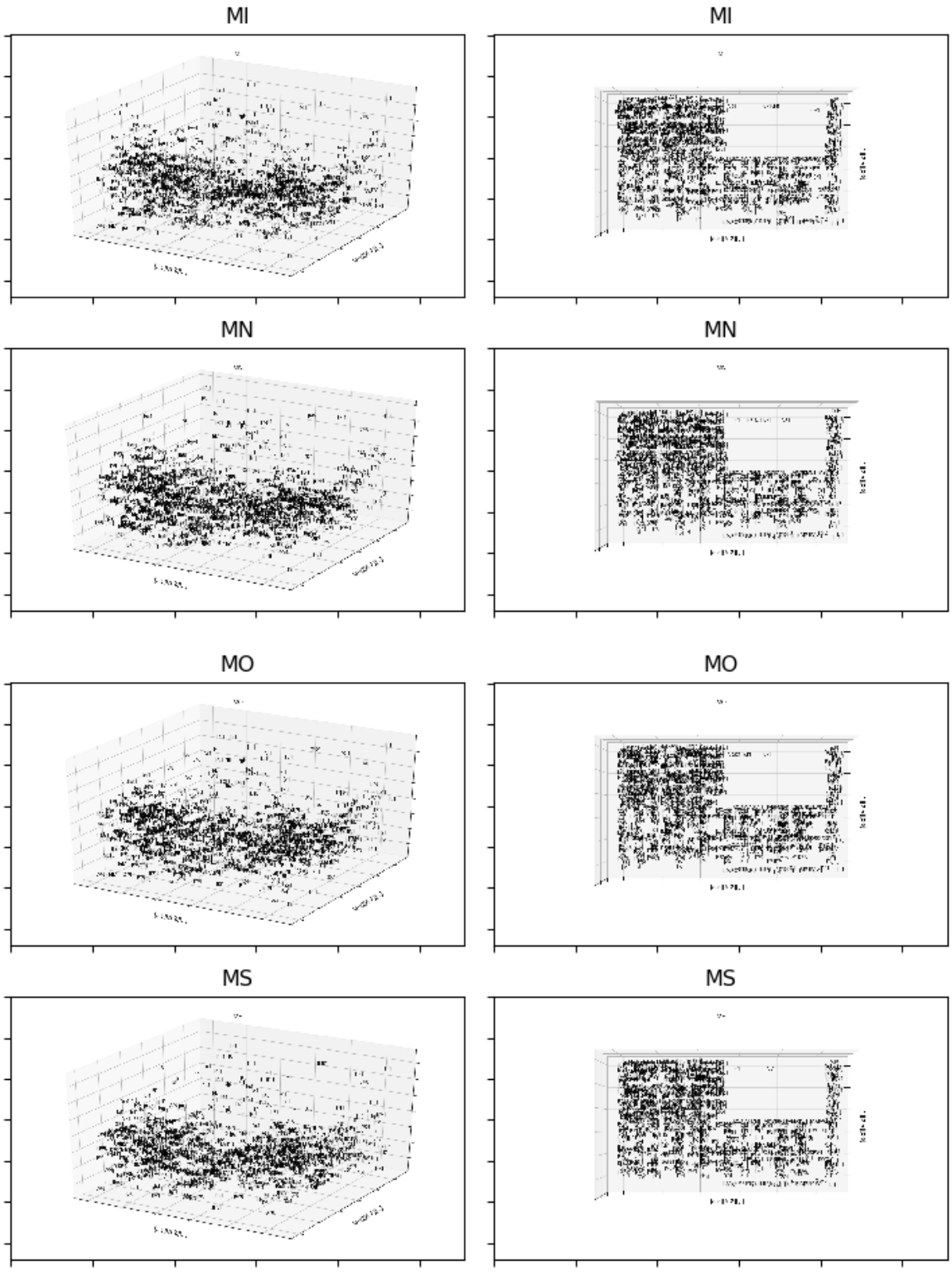

AK
AK
AL
AL
AR
AR
AZ
AZ

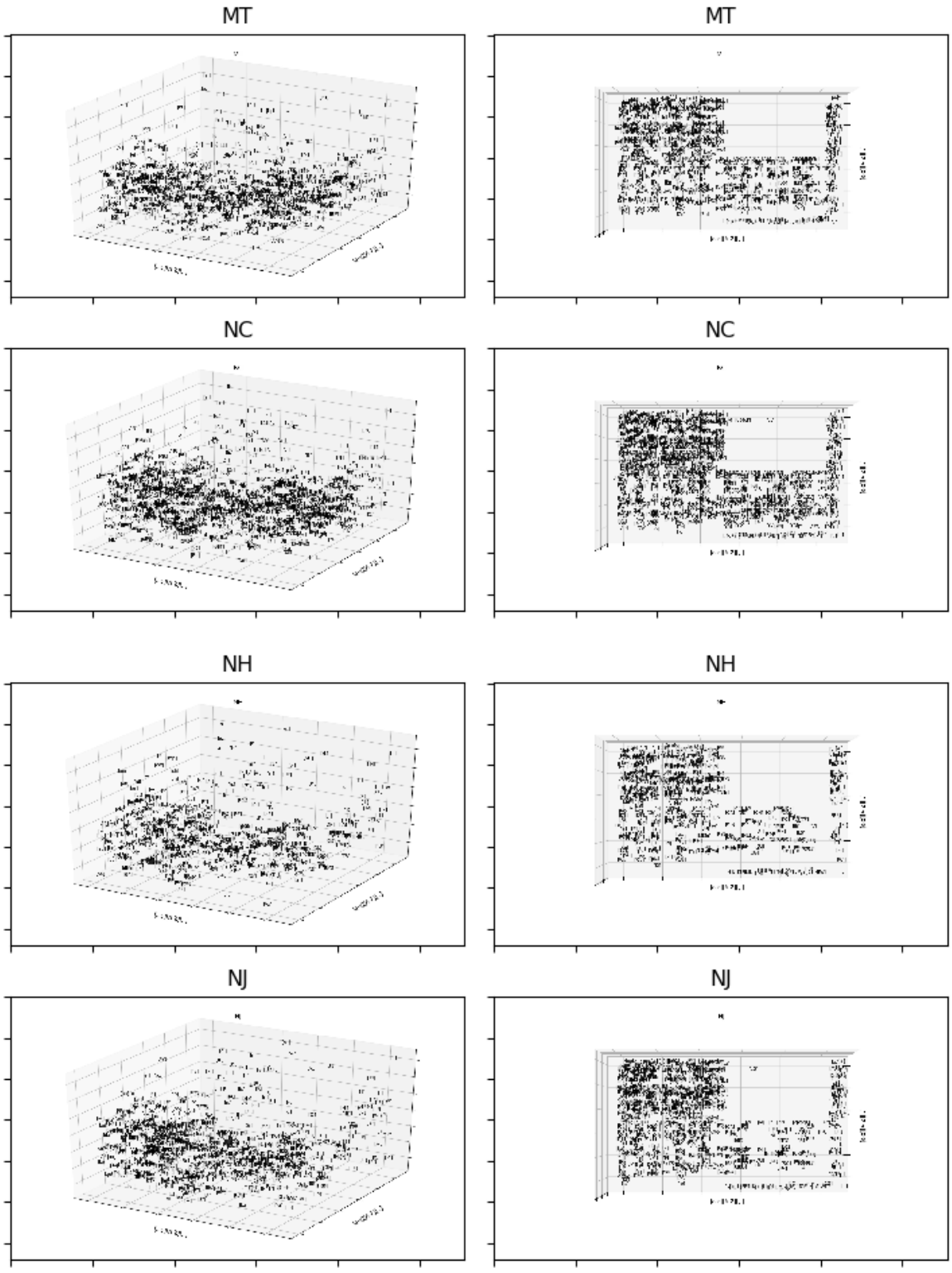

CA
CA
CO
CO
CT
CT
DC
DC

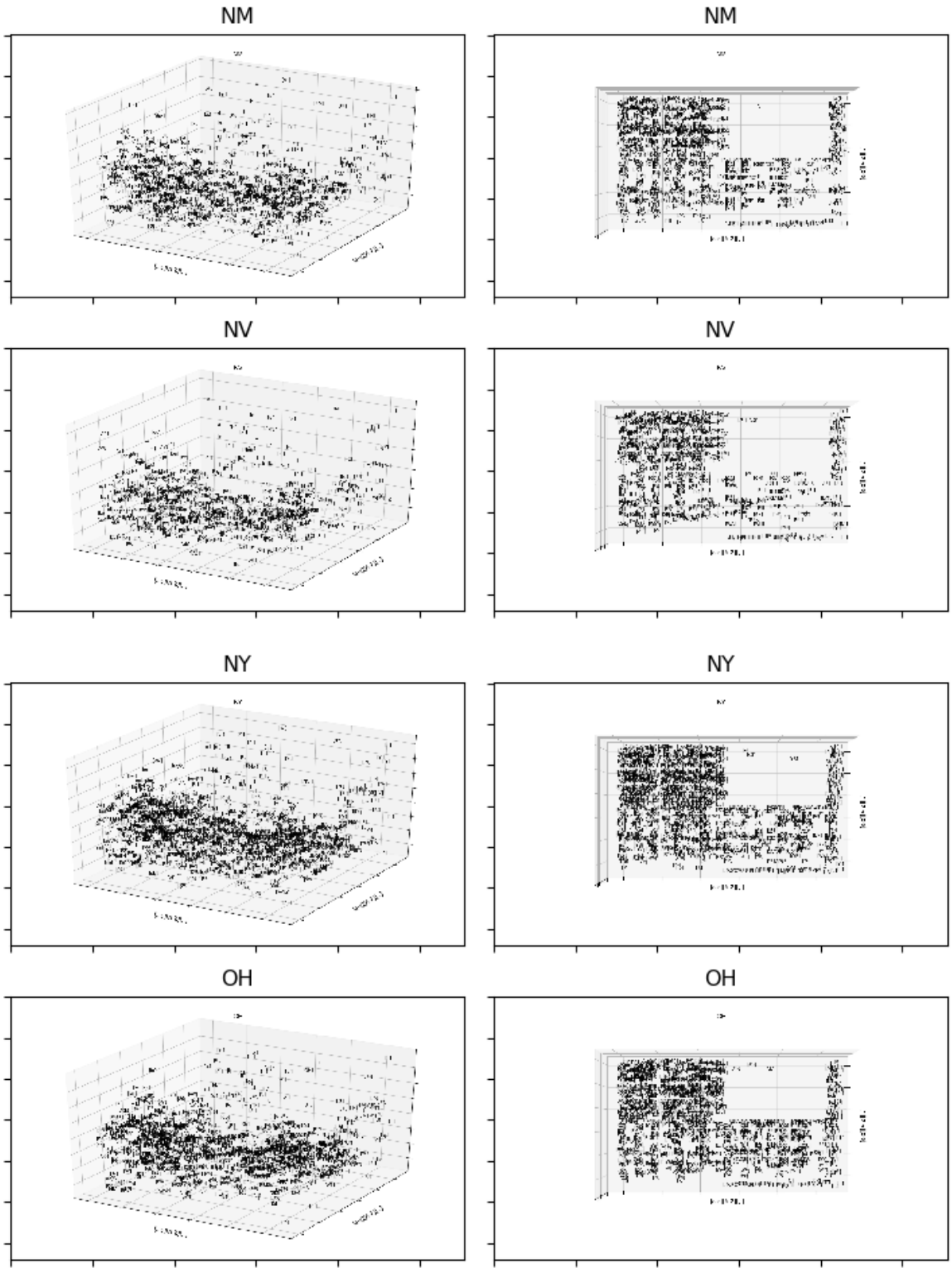

DE
DE
FL
FL
GA
GA
HI
HI

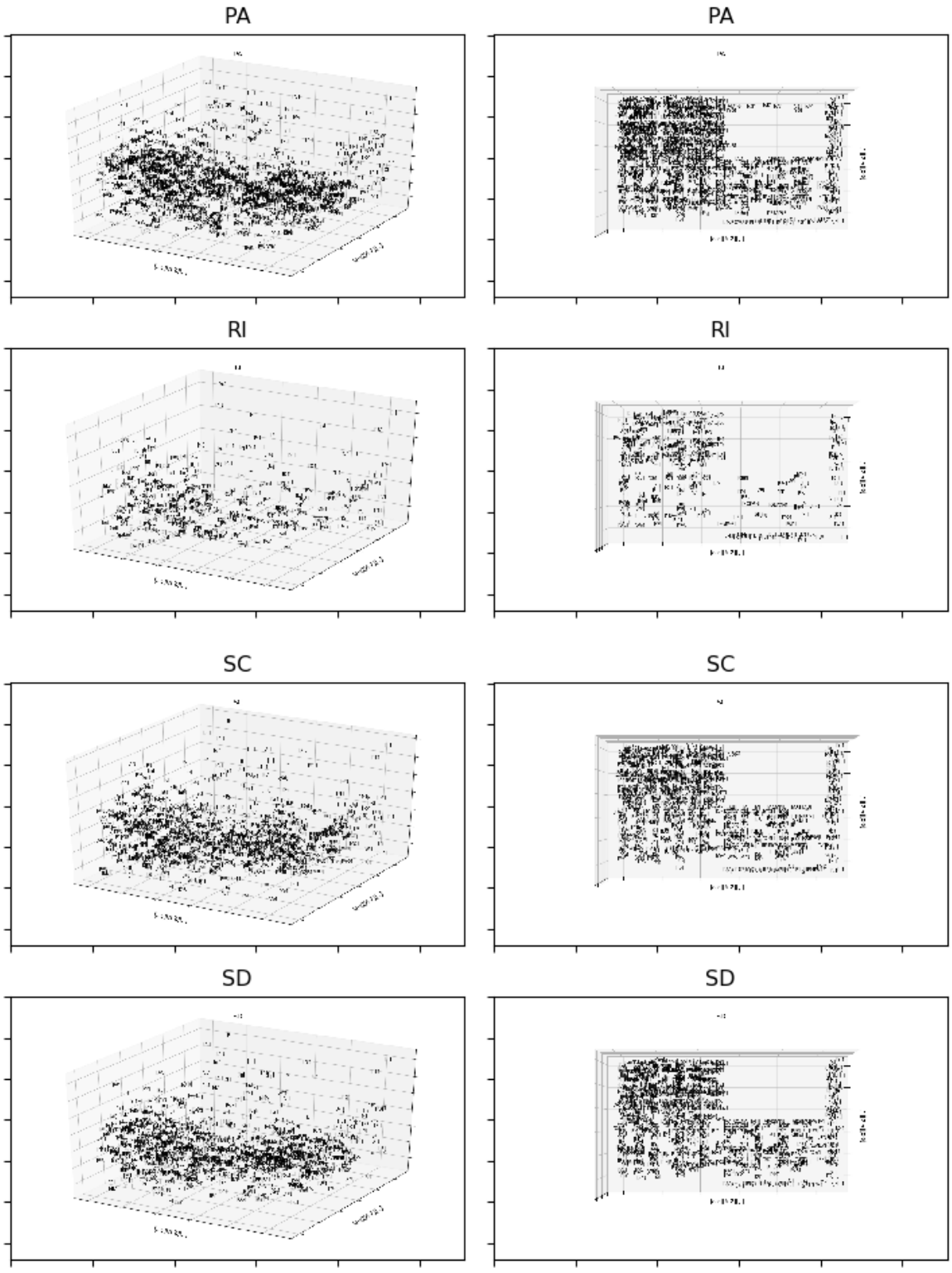

IA
IA
ID
ID
IL
IL
IN
IN

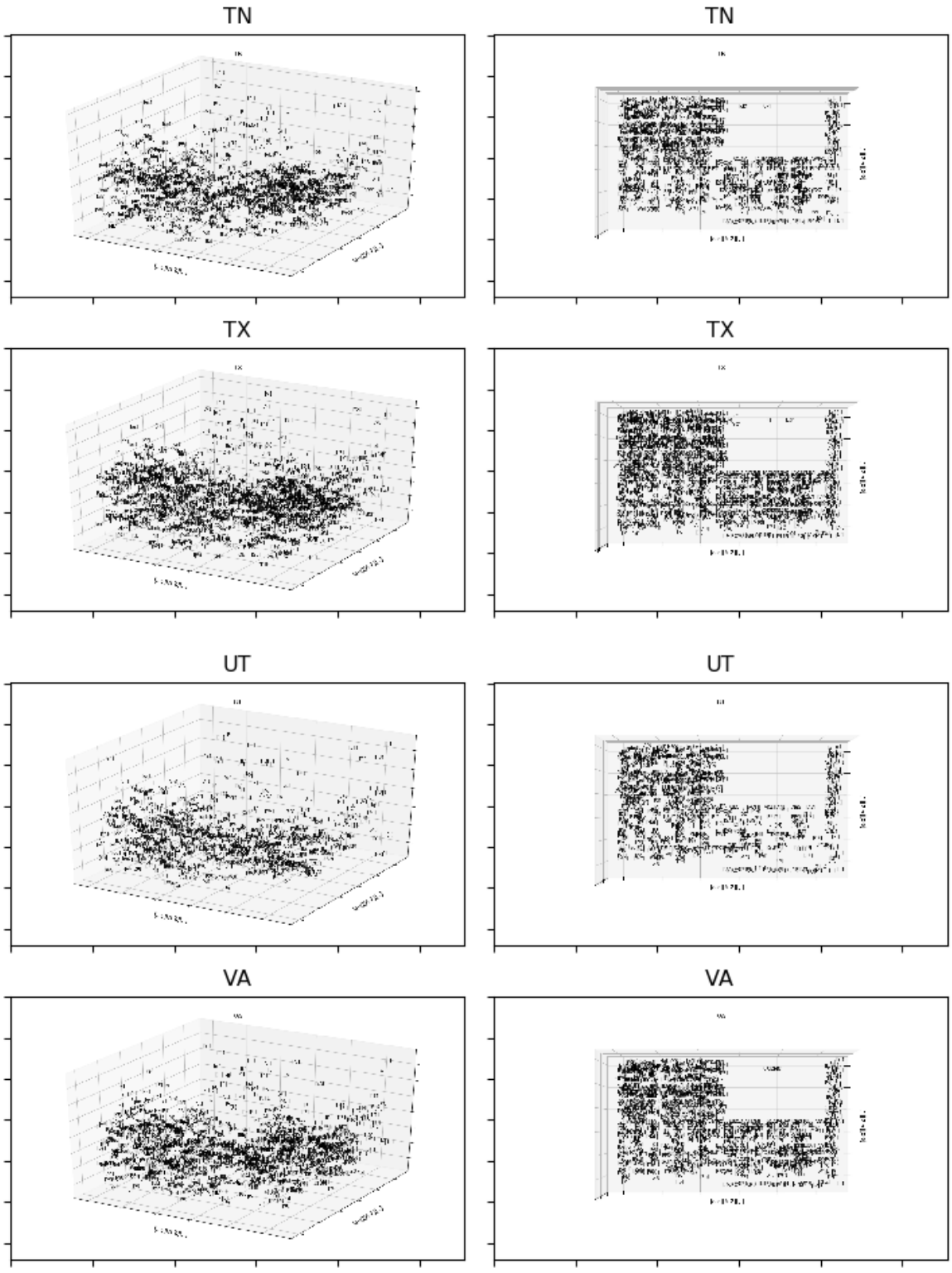

KS
KS
KY
KY
LA
LA
MA
MA

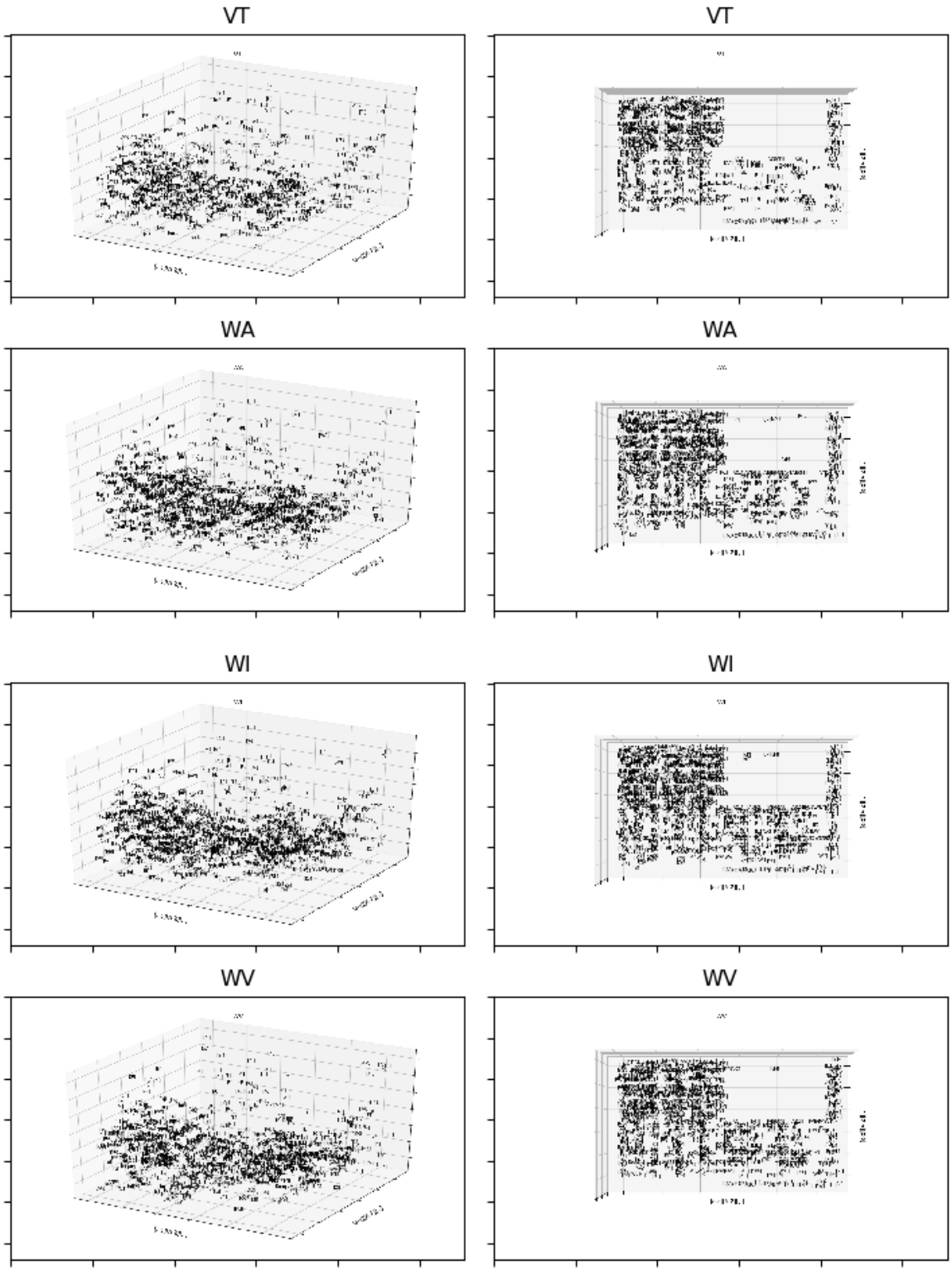

MI
MI
MN
MN
MO
MO
MS
MS

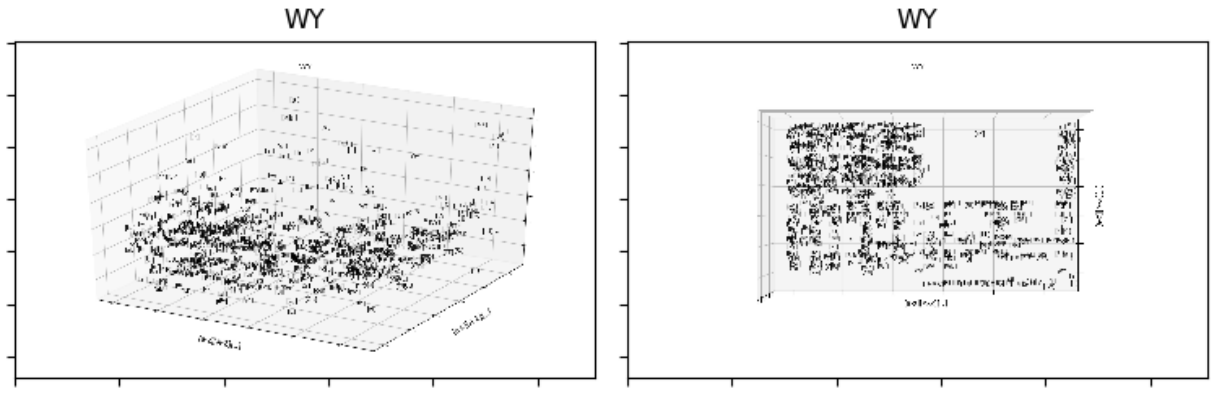

MT
MT
NC
NC
NH
NH
NJ
NJ


NM
NM
NV
NV
NY
NY
OH
OH


PA
PA
RI
RI
SC
SC
SD
SD


TN
TN
TX
TX
UT
UT
VA
VA


VT
VT
WA
WA
WI
WI
WV
WV


WY
WY

8 Glossary

Term	*Definition*
Deskewing	Compute the principal axis of the shape that is closest to the vertical, and shifting the lines to make it vertical (wiki)
Image Blurring	In image processing, a Gaussian blur (also known as Gaussian smoothing) is the result of blurring an image by a Gaussian function (named after mathematician and scientist Carl Friedrich Gauss). It is a widely used effect in graphics software, typically to reduce image noise and reduce detail. (wiki)
Pixel Shift	Pixel shifting is a term used both for a method to prevent "burn-in" of static images on displays and as a method to increase resolution in digital imaging devices and projectors. (wiki)

9 Bibliography

[1] C. Cortes, C. J. Burges and Y. LeCun, "MNIST database of handwritten digits - Yann LeCun," [Online]. Available: http://yann.lecun.com/exdb/mnist/.

[2] Y. Li, K. Zhao, X. Chu and J. Liu, "Speeding up k-Means algorithm by GPUs," *Journal of Computer and System Sciences,* vol. 79, no. 2, pp. 216-229, 2013.

[3] Q. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, Beijing, China, 2014.

[4] M. R. Ebrahimi, C. Y. Suen and O. Ormandjieva, "Detecting predatory conversations in social media by deep Convolutional Neural Networks," *Digital Investigation,* vol. 18, no. , pp. 33-49, 2016.

[5] Y. Tian, J. Wang, Z. Zhou and S. Zhou, "CNN-Webshell: Malicious Web Shell Detection with Convolutional Neural Network," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, Kunming, China, 2017.

[6] H. Fang, J. Fei, Q. Yin, C. Yang and D. Wang, "Restricted Stochastic Pooling for Convolutional Neural Network," pp. 1-4, 2018.

[7] S. Park and N. Kwak, "Analysis on the Dropout Effect in Convolutional Neural Networks," in *Computer Vision -- ACCV 2016*, Taipei, Taiwan, 2016.

[8] N. A. Jebril, H. R. Al-Zoubi and Q. A. Al-Haija, "Recognition of Handwritten Arabic Characters using Histograms of Oriented Gradient (HOG)," *Pattern Recognition and Image Analysis,* vol. 28, no. 2, pp. 321-345, 2018.

[9] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning*, Lille, France, 2015.

[10] Y. Wang, X. Wu, Y. Chang, S. Zhang, Q. Zhou and J. Yan, "Batch Normalization: Is Learning An Adaptive Gain and Bias Necessary?," in *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, Macau, China, 2018.

[11] K. Gopalakrishnan, S. K. Khaitan, A. N. Choudhary and A. Agrawal, "Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection," *Construction and Building Materials,* vol. 157, no. , pp. 322-330, 2017.

[12] F. Xiong, Y. Xiao, Z. Cao, K. Gong, Z. Fang and J. T. Zhou, "Towards Good Practices on Building Effective CNN Baseline Model for Person Re-identification.," *Computer Vision and Pattern Recognition ,* vol. abs/1807.11042, 2018.

[13] R. Ha, P. Chang, J. Karcich, S. Mutasa, R. Fardanesh, R. Wynn, M. Z. Liu and S. Jambawalikar, "Axillary Lymph Node Evaluation Utilizing Convolutional Neural Networks Using MRI Dataset," *Journal of Digital Imaging,* vol. 31, no. 6, p. 851–856, 2018.

[14] H. Li, X. Gong, H. Yu and C. Zhou, "Deep Neural Network Based Predictions of Protein Interactions Using Primary Sequences," *Molecules,* vol. 23, no. 8, p. 1923, 2018.

[15] S. Mannor, D. Peleg and R. Y. Rubinstein, "The cross entropy method for classification," in *Proceedings of the 22Nd International Conference on Machine Learning*, Bonn, Germany, 2005.

[16] S. Hallmark, R. R. Souleyrette and S. Lamptey, "Use of n-Fold Cross-Validation to Evaluate Three Methods to Calculate Heavy Truck Annual Average Daily Traffic and Vehicle Miles Traveled," *Journal of The Air & Waste Management Association,* vol. 57, no. 1, pp. 4-13, 2007.

[17] P. Klęsk, "Probabilities of discrepancy between minima of cross-validation, Vapnik bounds and true risks," *International Journal of Applied Mathematics and Computer Science,* vol. 20, no. 3, pp. 525-544, 2010.

[18] T. He, W. Huang, Y. Qiao and J. Yao, "Text-Attentional Convolutional Neural Network for Scene Text Detection," *IEEE Transactions on Image Processing,* vol. 25, no. 6, pp. 2529-2541, 2016.

[19] Y. LeCun, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, p. 2278–2324, 1998.

[20] B. Wang, L. Najjar, N. N. Xiong and R. C. Chen, "Stochastic Optimization: Theory and Applications," *Journal of Applied Mathematics,* vol. 2013, no. , pp. 1-2, 2013.

[21] J. Ding, X.-H. Hu and V. N. Gudivada, "A Machine Learning Based Framework for Verification and Validation of Massive Scale Image Data," *IEEE Transactions on Big Data,* Vols. PP, no. 99, pp. 1-1, 2018.

[22] N. Gajhede, O. Beck and H. Purwins, "Convolutional Neural Networks with Batch Normalization for Classifying Hi-hat, Snare, and Bass Percussion Sound Samples," in *Proceedings of the Audio Mostly 2016*, New York, NY, USA, 2016.