2019 20th IEEE International Conference on Mobile Data Management (MDM)

Deep Learning-Based Spatial Analytics for Disaster-Related Tweets: An Experimental Study

Shayan Shams, Sayan Goswami, and Kisung Lee
Division of Computer Science and Engineering
Louisiana State University (LSU)
Baton Rouge, LA 70803, USA
sshams2@cct.lsu.edu, sgoswami@cct.lsu.edu, klee76@lsu.edu

***Abstract*—Online social networks are being widely used during unexpected large-scale disasters not only for sharing latest news but also requesting emergency rescues. Particularly, social network posts with their location information are becoming more important because they can be utilized for emergency management, urban planning, and various studies to understand effects of the disasters. Despite their importance, the percentage of such posts is generally tiny. In this paper, to address the location sparsity problem on Twitter in the event of disasters, we propose a deep learning-based framework to spatially analyze the disaster-related tweets by focusing on classifying tweets from affected areas of disasters. We also study effects of different deep learning architectures and input embedding techniques for this classification task. Our experimental results demonstrate that our ConvNet model with the Word2vec word embedding has the highest classification accuracy.**

I. Introduction

We are witnessing widespread use of online social networks (e.g., Twitter, Facebook, and Weibo) for not only everyday lives but also major issues such as sports events, elections, earthquakes, and terrors. Particularly, during unexpected large-scale disasters, online social networks can serve as an important medium of emergency communication because traditional emergency systems (e.g., 911 calls in US) can be extremely slow (or unreachable at all). For example, when Hurricane Harvey hit the Houston metropolitan area with massive flooding in 2017, several people used social networks to request emergency rescues in life-threatening situations because they could not get through to a 911 emergency call center [1].

Social network posts with their location information can be effectively utilized before/during/after disasters for not only emergency management and rescues but also various social, political, environmental, and urban planning studies to understand effects of the disasters. For example, researchers use geotagged tweets to understand social-geographical disparities of Twitter use during Hurricane Harvey of 2017 [2]. However, despite the invaluable importance of the location information on online social networks, most social network users are reluctant to share their specific location because of various reasons including privacy concerns. This location sparseness problem limits opportunities for geographical studies in various disciplines.

To tackle the location sparseness problem on online social networks, researchers have explored various directions for extracting location-specific hints and predicting the location of each user or even each post. Even though existing approaches have shown some promising results for extracting location-related information, most of them depend on a gazetteer and a Part-Of-Speech (POS) tagger that are not optimized for social network posts including a lot of typos, grammatical errors, abbreviations, and hashtags. To address such limitations without heavy feature engineering and manual tagging, we develop a deep learning-based framework to spatially analyze the disaster-related tweets. Particularly, in this paper, we focus on classifying tweets from affected areas of two hurricanes (Hurricane Harvey of 2017 and Hurricane Sandy of 2012) only based on the text of tweets. This binary classification can be useful during disasters to quickly find those in affected areas and after disasters to conduct studies on the effects of the disasters. It can be also used as a building block for finer-grained classification tasks such as point of interest (POI) prediction and trajectory analysis. We also study effects of various deep learning architectures and input embedding techniques for this classification task. Our experimental results demonstrate that our ConvNet model with the Word2vec word embedding has the highest classification accuracy.

The rest of the paper is organized as follows. In Section II, we introduce the background of text embedding and deep learning followed by related work. We describe the design of our framework in Section III. We show our evaluation results in Section IV and conclude this paper in Section V.

II. Background

In this section, we introduce the background and related work of this paper.

A. Input Embeddings for Text

1) Bag-of-Words Models: A bag-of-words (BoW) model is a method by which we can extract features from textual data so that we can use the features in machine learning algorithms. A bag-of-words of a text indicates the frequencies of all words in the text regardless of their locations or structures. The model checks the occurrences of known words in the document. As the vocabulary size (i.e., the number of words) increases, so does the size of the vector representing the input document. Despite its simplicity, this model has several limitations. First

2375-0324/19/$31.00 ©2019 IEEE
DOI 10.1109/MDM.2019.00-40

of all, its wide and sparse representations make it hard to handle large-scale data for running machine learning algorithms because of the high computational requirements. In addition, it is hard to capture the semantics of words in text documents because this model disregards grammar and word order.

2) Vector Space Models: A vector space model (VSM) is another approach to extract information from text documents for running machine learning models. This approach alleviates the data sparsity and lack of semantics problems in the bag-of-words models. In VSMs, words are embedded in a continuous vector space such that those similar to each other are mapped closer to each other in the space. An instance of VSMs is the Word2vec [3] model, which can be either a continuous bag-of-words (CBOW) architecture that predicts target words from source context or a skip-gram architecture that does it the other way round.

3) Social Network-Specific Models: Processing text (e.g., tweets) from online social networks has a few challenges that are difficult to address with traditional natural language processing techniques. For example, tweets often contain informal words, spelling and grammatical errors, abbreviations, emoticons, and special characters, which make the vocabulary size too large for word-level models. Some of these problems are addressed in the Tweet2vec [4] model, which embeds an entire tweet into a vector by extracting complex dependencies in text. GloVe [5] also provides pre-trained word vectors for Twitter, trained using aggregated global word-word co-occurrence statistics.

B. Deep Learning Architectures

Recent advances in deep learning have garnered a lot of attention from various domains such as image and video recognition, speech recognition, and natural language processing. Complemented by the advancements in accelerator technologies [6], [7], deep learning techniques have been widely used not only in computer science but also in medicine, chemistry, and transportation, to name a few. The field is still constantly evolving with newer models, and many novel applications have reported remarkable success using the deep learning techniques [8]–[11].

1) Convolutional Neural Networks: One of the driving forces behind the advancement of deep learning is the development of convolutional neural networks (CNNs or ConvNets). These networks have proved to be extremely good in image processing such as classification, object recognition, and diagnosis from medical imagery. Moreover, CNNs are still under active development, and researchers frequently report variants with better accuracy results [12]–[14]. Since CNNs are designed to work on images, their structural characteristics significantly differ from those of traditional neural networks. Neurons in neighboring layers exhibit a local connectivity pattern, ensuring that filters learn to detect patterns based on spatial locality. Each filter scans the entire image for finding a certain pattern, and several filters can be stacked, producing a 3D volume of neurons that is capable of detecting many different patterns. The network consists of a hierarchy of spatial patterns where smaller receptive fields create larger ones. Based on weight sharing, a filter learns a certain pattern across the whole image.

2) Recurrent Neural Networks: CNNs are generally not optimized when the input data are interdependent as a temporal pattern. Since CNNs do not preserve any sort of correlation between the previous input and the next input, all the outputs are independent. On the other hand, recurrent neural networks (RNNs) consist of states (i.e., memory) that are utilized to process sequences. This makes them suitable for processing various temporal data such as language modeling, text generation, machine translation, speech recognition, signal processing, generating image descriptions, and video tagging, in which the output depends on a sequence of words or pictures that are temporally correlated. An RNN stores all the previous step input and merges that information with the current step input. Therefore, it can also capture some information regarding the correlation between the current step and the previous steps. In other words, the decision at time $t-1$ affects the decision taken at time t. Long Short-Term Memory (LSTM), a specific type of RNNs, is particularly well-suited for time-series data [15].

C. Related Work

There are many existing techniques to predict the location of each Twitter user, and we can broadly categorize them into two classes: a) a content-based approach that tries to infer the user's location from all tweets posted by a user and b) a network-based approach that extracts the location information using other users in a person's network [16]. The content-based methods find location-specific words from each user's tweets to predict the location of the user [17]–[19]. A variation of the content-based methods uses TF (Term Frequency) or TF-IDF (Inverse Document Frequency) to score words in tweets, label the top words with locations, and use them in a supervised machine learning or deep learning models [20]–[23]. In addition, there are several existing techniques to predict the next location of a social network user [24] or future visitors at given locations [25] using location-based social networks (LBSN).

Another approach tries to find locations mentioned in the tweet text using Named Entity Recognition (NER) tools [26], another location-based social network [27], [28], or deep learning models [29]. These methods are designed to work on individual tweets as opposed to a user's tweet history mentioned previously. In this paper, we focus on classifying tweets from affected areas of disasters only based on the text of tweets, which can be useful for emergency response organizations, urban planners, and social/environmental scientists. Unlike existing approaches that typically require heavy feature engineering or labor-intensive manual tagging for generating ground truth, we develop a deep learning-based framework without such burdens and also study the effects of different deep learning and input embedding techniques for this classification.

TABLE I
NUMBER OF TWEETS CONTAINING LOCATION INFORMATION

Hurricane	With *geo*	With *location*	Affected	Not affected
Harvey	13,814	175,424	49,285	96,874
Sandy	131,540	162,032	67,749	91,935

III. FRAMEWORK DESIGN

In this section, we describe the detailed pipeline in our framework for this study.

A. Data Preprocessing

For this study, we have acquired tweets related to Hurricane Harvey of 2017 using hashtags "harvey" and "hurricaneharvey" and Hurricane Sandy of 2012 using hashtags "sandy" and "hurricanesandy" (all case insensitive) via Gnip. The total numbers of tweets gathered after filtering out non-English and non-US tweets are 21 million for Harvey and 15.4 million for Sandy. For Hurricane Sandy, about 162 thousand tweets are associated with a *location* field and denoted by a polygon. The field indicates a semantic place, but its resolution widely varies from points of interest and neighborhoods to cities, admin regions (e.g., US states), and even countries. The *geo* field stores the fine-grained location (i.e., latitude and longitude) of a tweet where the tweet was posted. Out of the 162 thousand tweets for Hurricane Sandy, about 132 thousand tweets have their geo-coordinate. For Hurricane Harvey, about 175 thousand tweets have their location information while about 14 thousand tweets among them have their geo-coordinate. It is worth noting that the percentage of geo-tagged tweets for Hurricane Harvey is significantly lower than previously reported percentages in other research publications (around 1%) [27]. Even though we could not figure out its exact causes, one possible reason would be improved awareness about social network privacy in recent years.

For each hurricane, we obtain a list of affected zip codes (i.e., postal codes in US) and convert them into latitude-longitude pairs. We use the pairs to create a polygon of affected areas. Among the tweets associated with location, we discard those with state- or country-level resolutions and consider only neighborhoods, points of interest, and cities. After applying this filter, we have about 146 thousand and 160 thousand tweets for Hurricanes Harvey and Sandy respectively.

We compare the location of each filtered tweet with the polygon of affected areas and mark the tweet as "from affected areas" if there is more than 50% overlap. In other words, we divide the tweets into two classes: 1) those originating from areas affected by the hurricane and 2) those originating from the rest of the US. Details of the datasets are summarized in Table I.

B. Word Embedding

Since the size of the filtered tweets varies from 1 to 34 words, we only keep the tweets whose length is between the 25th percentile (9 words) and 75th percentile (19 words) in this study, giving us about 176 thousand filtered tweets in total. We perform this additional filtering for two reasons. First, it is unlikely that tweets with only a few words have enough information for this classification task. Second, for designing LSTM models based on the embedding techniques, we need to add many zeros to properly represent the tweets with a large number of words, causing the data sparsity problem and introducing extra training time without meaningful benefit in the trained models. We leave efficient processing of the excluded tweets as our future work. Figure 1 illustrates the histogram for the word lengths of the tweets after our preprocessing described in Section III-A.

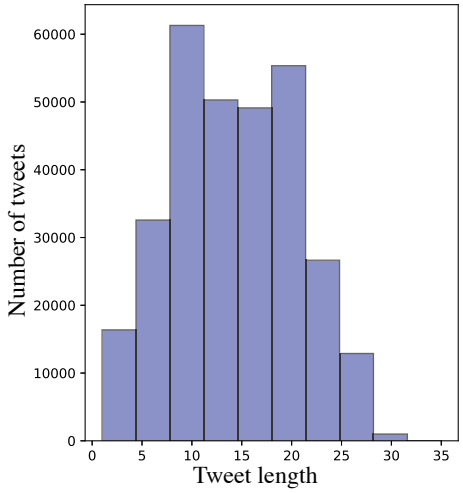

Fig. 1. Histogram of the number of words in tweets after preprocessing

1) Word2vec: To train a Word2vec model, we first make a dataset of words in context from the filtered tweets. We define the context of a word w to be a window spanning from words to the left of w to those at the right of w. In this study, we use the skip-gram model for mapping each word to a vector. Next, we make a dictionary of contexts and targets and convert the dataset into a set of inputs and outputs, which can be used for training the model. For feature learning in Word2vec, we train our skip-gram architecture to classify words into either real targets or noise based on their probability values. In each step, we choose one noisy example from the unigram distribution and calculate the loss for a target-noise pair. We update the parameters in a way that the model assigns a higher probability to the observed word than the noisy word. We use stochastic gradient descent (SGD) with a mini-batch size of 128 for maximizing the objective over the whole dataset.

We obtain 1,639,779 total words and 141,185 distinct words (i.e., vocabulary size) using both the Harvey and Sandy datasets. In our setting, we convert each word into a vector of size 64 and 500 for LSTM and ConvNet respectively. The window size (i.e., the number of left and right words) is one for each direction, and we use 64 as the number of negative examples to sample.

2) Tweet2vec: We use the open source code[1] of Tweet2vec [4] to embed each tweet to a vector of size 500. In the Tweet2vec model, each tweet is divided into characters, and each character is converted to a one-hot vector with one bit set to one representing the presence of the character. The one-hot vectors are multiplied by the matrix $M_C \in \mathbb{R}^{|C| \times d_c}$, where d_c is the size of the vector for the character space and $|C|$ is the number of characters, to be projected into the character space. The character vectors are fed to a bi-directional Gated Recurrent Unit (GRU) and a dense layer to be processed for obtaining a final embedding vector for the tweet. The last layer uses the softmax function with the output size equal to the number of hashtags in the dataset to compute the probability for each hashtag. In our setting, we convert each tweet into a vector of size 500.

3) GloVe: Stanford University's NLP group has processed massive datasets from Twitter containing 2 billion tweets with a vocabulary of about 1.2 million words and uploaded the pre-trained embedding vectors and corresponding words on its Github page[2]. The Github page provides embedding vectors of sizes 25, 50, 100, and 200. In this study, we use the pre-trained embedding vectors of size 200 to convert each word in tweets into a vector of size 200.

C. Model Training

1) Convolutional Neural Networks: While convolutional neural networks (CNNs) are widely used for analyzing data with spatial correlation such as images, recent works show that CNNs can have excellent success in analyzing data with temporal correlation such as speech recognition, text, and DNA sequences analysis [11], [30], [31]. Therefore, we use a ConvNet (CNN) architecture as one of our deep learning models to classify if a tweet was originated from affected areas of a hurricane. We implement a ConvNet model with five 1-D convolutional layers followed by two fully connected layers. The first three layers are followed by max-pooling layers, and the last two layers are followed by drop-out layers and two fully connected layers. Activation functions are ReLU for all layers except the last layer's activation, which is the softmax function to calculate the probability of the tweet being sent from the affected areas of a hurricane. We use the standard binary cross-entropy loss with SGD to train the model.

2) Long Short-Term Memory: Long Short-Term Memory (LSTM), a specific type of RNNs, has been successfully applied to various temporal data such as text and speech by preserving the temporal correlations in a sequence of inputs. Therefore, in this study, we use a single-layer LSTM model and a stacked version with three LSTM layers to compare their performance with that of the ConvNet model. Our single-layer LSTM model comprises of one LSTM with 100 hidden units followed by a fully connected layer with the softmax activation function. Our stacked LSTM model includes three LSTM layers with 100 hidden units followed by a fully connected layer with the softmax activation function. For both the LSTM models, the number of time steps is 19. We use the standard binary cross-entropy loss with SGD to train the models.

[1]https://github.com/bdhingra/tweet2vec

[2]https://github.com/stanfordnlp/GloVe

IV. Experiments

In this section, we present the classification results for three different deep learning models (ConvNet, single-layer LSTM, and stacked LSTM) and one baseline model using linear logistic regression (LLR). Furthermore, we show the effects of different embedding techniques on prediction accuracy. In this paper, we define the accuracy as the proportion of correctly classified tweets among all evaluated tweets. Table II summarizes our experimental results.

A. Input Vectors for ConvNet

We evaluate the ConvNet model using three different embedding techniques (Tweet2vec, Word2vec, and GloVe). For Tweet2vec, we convert each tweet to a vector of size (1×500). For Word2vec, we first find the learned embedding vector for each word in a tweet, and so each word is converted to a vector of size 500. Next, we stack all the corresponding vectors vertically and calculate the average of the stacked vector for each feature in the learned vector space, as described in [32]. For example, if a tweet has 15 words, the stacked vector's size is (15×500), and the input vector's size is (1×500). For GloVe, we convert each word to a vector of size 200 using pre-trained embedding vectors. Similar to the processing for Word2vec, we stack all the corresponding vectors vertically and calculate the average for each feature. The input vector's size is (1×200).

B. Input Vectors for LSTM

We evaluate two LSTM models (single-layer and stacked models). As inputs for the LSTM models, we convert each tweet to a vector using the learned embedding from the Word2vec model or pre-trained GloVe embedding vectors. For Word2vec, we first find the learned embedding vector for each word in a tweet, and so each word is converted to a vector of size 64. Then, we concatenate all word vectors of a tweet to form a tweet vector. Finally, since 19 is the maximum number of words for the tweets in our evaluation as explained in Section III-A, we pad the obtained tweet vector to reach the length of 19 in the first axis. For example, if a tweet has 15 words, its tweet vector's size is (15×64), and the final vector's size is (19×64) with four zero vectors for 16th, 17th, 18th, and 19th words (i.e., a zero matrix of size (4×64)). We follow the same process for the GloVe embedding vectors except the size (1×200) of each word vector, and so the final input vector's size is (19×200).

Since we convert each input tweet to a vector of size $(19 \times$ *the size of the learned vectors*$)$, we consider 19 time steps for our LSTM models. To eliminate the effects of zero padding, we make a mask matrix and multiply it to the loss matrix. For example, for a tweet with 12 words, the mask matrix has 1s for the first 12 rows and 0s for the last 7 rows. We exclude Tweet2vec for evaluating our LSTM models since Tweet2vec

TABLE II
CLASSIFICATION ACCURACY FOR DIFFERENT DEEP LEARNING MODELS

Model	Input	Accuracy	#Param	Tr-Time
LLR	Word2vec	53.60%	1,002	166 µs
Single LSTM	Word2vec	57.32%	66,202	280 µs
	GloVe	56.50%	120,602	350 µs
Stacked LSTM	Word2vec	60.00%	227,002	763 µs
	GloVe	58.21%	281,402	830 µs
ConvNet	Word2vec	**68.44**%	109,632,130	428 µs
	GloVe	60.03%	40,426,114	280 µs
	Tweet2vec	64.52%	109,632,130	430 µs

converts each tweet into a vector, and so meaningful time steps are not defined for LSTM.

C. Training Time and Parameters

We also report the number of trainable parameters and training time (Tr-Time) for one training iteration with a batch size of 256 for each model in Table II. We measure the training time for one iteration by calculating the average of 1000 iterations after the 5th epoch because the first several epochs' training time is usually longer than the rest of training. We use a NVIDIA Tesla V100 PCIe GPU and TensorFlow version 1.11 [33] for measuring the training time. Note that the difference in terms of the number of parameters for the same model with different embedding techniques is because of the different input vector size, not model configurations. For instance, the input size for ConvNet with Word2vec is (1×500) while the size with GloVe is (1×200), and so ConvNet with GloVe has less parameters.

D. Discussion

As Table II shows, our ConvNet model with Word2vec demonstrates the highest accuracy among all evaluated deep learning models and input embedding techniques. We claim that the ConvNet with Word2vec works better because of two main reasons. First, our ConvNet model is much deeper and has much more trainable parameters than the other models, giving the model the significant nonlinear transformation capability to effectively map input data to high-dimensional representations. Second, we use a Word2vec model trained on our Twitter data while we utilize pre-trained embedding vectors for GloVe and Tweet2vec, which might not represent our dataset distribution well.

Even though we show some promising results for this classification task using the ConvNet with Word2vec, we could witness only a marginal improvement using the other deep learning models. For example, the accuracy of the single-layer LSTM model with GloVe is only 3% higher than that of our baseline model (linear logistic regression) even though it has almost 60 times more trainable parameters. These results demonstrate the importance and challenge of choosing a proper deep learning model with adequate input data preparation for classification tasks that are not well studied. They also show that location prediction only based on the text of tweets has inherent difficulties that are hard to tackle even with the power of deep learning and word embedding. For example, our model could correctly classify tweet *"consistently terrible traffic is a problem plaguing houstonians this week hurricaneharvey houstonstrong"* posted from affected areas. On the other hand, it failed to correctly classify tweets *"everyone in texas that will be affected by hurricaneharvey please stay safe and take care of each other texasstrong"* and *"born in houston a few days before hurricaneharvey she is now home in miami fleeing hurricaneirma amazingbaby"* both from non-affected areas. To address such challenges, we plan to extend this work to consider other types of information, such as the location in previous tweets from the same user, the home location of users after some validation, and the follower (or friendship) network of each user. We also believe that, given that the length of each tweet is less than 280 characters (140 characters before November 2017), the limited size of our dataset is another factor of low accuracy results. We plan to continuously collect more disaster-related tweets to further improve our models, and so emergency response organizations and social/environmental scientists can effectively utilize our classification results.

Another interesting observation from our results in Table II is the non-linear relationship between the number of trainable parameters and training time. For instance, the LLR model has 1,002 parameters while the ConvNet model with Word2vec has 109,632,130 parameters, but the training time of the ConvNet model is only 2.5 times longer than that of the LLR model. These results highlight the necessity of having higher-speed connection between GPU and CPU since the results indicate data transfer between CPU and GPU rather than the computation dominated the training time of the LLR model.

V. CONCLUSION

In this paper, we have presented our deep learning-based framework to spatially analyze the disaster-related tweets by focusing on classifying tweets from affected areas of disasters. We have reported our experimental results using different deep learning architectures and input embedding techniques for tweets related to Hurricanes Sandy and Harvey. Our results demonstrate that the ConvNet model with Word2vec shows the highest accuracy, but they also raise several open challenges for deep learning-based spatial analytics for tweets. In this study, we use only the text of tweets for location prediction to understand the capability of word embedding and deep learning techniques for short social network posts. As our future work, we plan to utilize other types of available data (e.g., previous location history, home location, follower network) to further improve the prediction accuracy.

ACKNOWLEDGMENTS

This work was partially funded by NSF grants (IBSS-L-1620451, SCC-1737557, RAPID-1762600) and an LA Board of Regents grant (LEQSF(2016-19)-RD-A-08).

REFERENCES

[1] L. Silverman. (2017) Facebook, Twitter Replace 911 Calls For Stranded In Houston. [Online]. Available: https://n.pr/2vmb8zQ

[2] L. Zou, N. S. Lam, S. Shams, H. Cai, M. A. Meyer, S. Yang, K. Lee, S.-J. Park, and M. A. Reams, "Social and geographical disparities in Twitter use during Hurricane Harvey," *International Journal of Digital Earth*, pp. 1–19, 2018.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] B. Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen, "Tweet2vec: Character-based distributed representations for social media," *arXiv preprint arXiv:1605.03481*, 2016.

[5] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[6] S. Shams, R. Platania, K. Lee, and S.-J. Park, "Evaluation of deep learning frameworks over different HPC architectures," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017, pp. 1389–1396.

[7] S. Goswami, K. Lee, S. Shams, and S. Park, "GPU-Accelerated Large-Scale Genome Assembly," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, May 2018, pp. 814–824.

[8] S. Shams, R. Platania, J. Zhang, J. Kim, K. Lee, and S.-J. Park, "Deep Generative Breast Cancer Screening and Diagnosis," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2018, pp. 859–867.

[9] N. Ruchansky, S. Seo, and Y. Liu, "Csi: A hybrid deep model for fake news detection," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. ACM, 2017, pp. 797–806.

[10] R. Platania, S. Shams, S. Yang, J. Zhang, K. Lee, and S.-J. Park, "Automated breast cancer diagnosis using deep learning and region of interest detection (bc-droid)," in *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2017, pp. 536–543.

[11] S. Shams, R. Platania, J. Kim, J. Zhang, K. Lee, S. Yang, and S.-J. Park, "A Distributed Semi-Supervised Platform for DNase-Seq Data Analytics using Deep Generative Convolutional Networks," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM, 2018, pp. 244–253.

[12] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[15] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*. IEEE, 2013, pp. 6645–6649.

[16] X. Zheng, J. Han, and A. Sun, "A survey of location prediction on Twitter," *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[17] Y. Yamaguchi, T. Amagasa, H. Kitagawa, and Y. Ikawa, "Online user location inference exploiting spatiotemporal correlations in social streams," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 1139–1148.

[18] B. Han, P. Cook, and T. Baldwin, "Text-based twitter user geolocation prediction," *Journal of Artificial Intelligence Research*, vol. 49, pp. 451–500, 2014.

[19] K. Ryoo and S. Moon, "Inferring twitter user locations with 10 km accuracy," in *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 2014, pp. 643–648.

[20] B. Wing and J. Baldridge, "Hierarchical discriminative classification for text-based geolocation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 336–348.

[21] Y. Miura, M. Taniguchi, T. Taniguchi, and T. Ohkuma, "A simple scalable neural networks based model for geolocation prediction in Twitter," in *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, 2016, pp. 235–239.

[22] A. Rahimi, T. Cohn, and T. Baldwin, "A neural model for user geolocation and lexical dialectology," *arXiv preprint arXiv:1704.04008*, 2017.

[23] Y. Miura, M. Taniguchi, T. Taniguchi, and T. Ohkuma, "Unifying text, metadata, and user network representations with a neural network for geolocation prediction," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2017, pp. 1260–1272.

[24] Y. Su, X. Li, W. Tang, J. Xiang, and Y. He, "Next check-in location prediction via footprints and friendship on location-based social networks," in *2018 19th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2018, pp. 251–256.

[25] M. A. Saleem, F. S. Da Costa, P. Dolog, P. Karras, T. B. Pedersen, and T. Calders, "Predicting visitors using location-based social networks," in *2018 19th IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2018, pp. 245–250.

[26] X. Liu, F. Wei, S. Zhang, and M. Zhou, "Named entity recognition for tweets," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 4, no. 1, p. 3, 2013.

[27] K. Lee, R. K. Ganti, M. Srivatsa, and L. Liu, "When twitter meets foursquare: tweet location prediction using foursquare," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. ICST, 2014, pp. 198–207.

[28] K. Lee, R. Ganti, M. Srivatsa, and P. Mohapatra, "Spatio-temporal provenance: Identifying location information from unstructured text," in *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE, 2013, pp. 499–504.

[29] A. Kumar and J. P. Singh, "Location reference identification from tweets during emergencies: A deep learning approach," *International Journal of Disaster Risk Reduction*, vol. 33, pp. 365–375, 2019.

[30] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Transactions on audio, speech, and language processing*, vol. 22, no. 10, pp. 1533–1545, 2014.

[31] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, "End-to-end text recognition with convolutional neural networks," in *Pattern Recognition (ICPR), 2012 21st International Conference on*. IEEE, 2012, pp. 3304–3308.

[32] T. Kenter and M. De Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM international on conference on information and knowledge management*. ACM, 2015, pp. 1411–1420.

[33] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf