Local App Classification using Deep Neural Network based on Mobile App Market Data

Keiichi Ochiai
NTT DOCOMO, INC
Tokyo, Japan
ochiaike@nttdocomo.com

Fatina Putri
NTT DOCOMO, INC
Tokyo, Japan
fatina.putri.fw@nttdocomo.com

Yusuke Fukazawa
NTT DOCOMO, INC
Tokyo, Japan
fukazawayu@nttdocomo.com

***Abstract*—Due to the spread of smartphones, mobile applications (app) have been widely used in daily life. Several apps which provide real-world related information (called "local apps") are useful for not only tourist but also residents. There is a category that seems to contain local apps in app market such as "Travel & Local" in Google Play, but many local apps are categorized into other categories. Thus, we present a method to classify local apps based on app market data using deep neural network (DNN). We leverage the fact that each app is manually labeled by developer to pre-train the DNN. In addition, we create features from an app market data because app markets involve multi-modal data such as app name, category and number of installs. We conducted an experiment on a real-world dataset crawled from Google Play to validate the effectiveness of the proposed method. Our evaluation shows that the proposed method outperforms the baseline method by 5.5% regarding F1 score.**

***Index Terms*—Local Mobile App, Deep Neural Network, Location based services**

I. Introduction

With the rapid adoption of modern smartphones, mobile applications (app) are being used increasingly in daily life, and there is a large number of apps in various app markets, such as Google Play and Apple's App Store. In an app market, each app has multi-modal data such as a text description, screen-shots, manually labeled category[1] (e.g. Travel & Local, Shopping and Education) and rating [1]. Several apps support the users' real world activities by providing local, real-world related information, and such apps are useful for both tourists and local residents. For example, there are apps for a specific area (e.g. KYOTO Trip+[2] is an app for tourists and residents in Kyoto), specific spots (e.g. YOKOHAMA WORLD PORTERS[3] is the official app of a shopping mall) and specific real-world activities (e.g. Tide Chart FREE[4] is fishing app). We refer to such real-world related apps as "local apps". By extracting the location name from the app name or description, a local app can be geolocated to a real-world geographic location (i.e. latitude and longitude). One of an application of geolocated local app is recommending local apps based on the user's current location. To realize this recommendation, local apps must be filtered by the "Travel & Local" category (in the Google Play case) and geographic range. However, we have discovered that many local apps are categorized into other categories. For example, the YOKOHAMA WORLD PORTERS and Tide Chart apps are categorized as "Shopping" and "Weather", respectively.

Classifying local apps is important for several mobile app market stakeholders. For users, local apps support their real-world activities by providing event information, coupon, etc. For app developers, an appropriate app category can be recommended by classifying their app prior to registering the app to an app market, and an appropriate category help developers increase the number of users who view their apps.

Classifying local apps is a non-trivial and difficult problem. One naive method is simply extracting the location name from the app name or description. For example, if we search apps whose app name contains "Tokyo", the search result contains many non-local apps, such as games and Internet shopping apps, that do not provide real-world related information. Thus, this naive method would fail. Another naive method is to calculate the geographic bias of app usage using app usage log and location data. However, a cold start problem would occur when calculating geographic bias because this method requires a sufficient amount of app usage logs that contain location data. Therefore, it is important to classify local apps using only app market data.

In previous studies [2], [3], classifying mobile app was considered a text classification problem because short text data, such as an app name and web search snippets, were used to classify apps. Generally, short text classification faces the data sparseness problem [2], and a semantic feature (i.e. topic distribution) and bag-of-words features are used to address this problem [2], [3]. However, the effectiveness of such methods is limited.

Thus, we tackle this problem by exploiting word embeddings and a deep Convolutional Neural Network (CNN) which have achieved high text classification accuracy [4]–[6]. However, deep learning generally requires a significant amount of training data. In fields in which deep learning models have shown success, such as image recognition, there is a large number of publicly available annotated data such as ImageNet [7]. On the other hand, there is no available public

[1]https://support.google.com/googleplay/android-developer/answer/113475?hl=en

[2]https://play.google.com/store/apps/details?id=jp.kyoto.pref.visitkyoto&hl=en

[3]https://play.google.com/store/apps/details?id=com.ywpapp&hl=en

[4]https://play.google.com/store/apps/details?id=jp.gr.java_conf.hanitaro.tide&hl=en

978-1-5386-9148-9/19/$31.00 ©2019 IEEE

data suitable for deep learning relative to the app classification problem. Therefore, we focus on the fact that all mobile apps are manually categorized into pre-defined categories by developers upon release. We exploit this as manually annotated data to pre-train a deep learning model and efficiently fine-tune the model using a small amount of annotated data for local app classification. In addition, app markets have multi-modal data; thus the propose method extracts features from text descriptions, number of installs, and app category, etc. to classify local apps. Finally, deep learning is suitable for integrating multi-modal data for classification [8]. Therefore, we integrate features from text using CNN and other multi-modal data. In this study, we conducted an experiment on a real-world dataset crawled from Google Play to validate the effectiveness of the proposed method.

The contributions of this study are summarized as follows.

- We propose a learning method for local app classification using app category classification as pre-training. The proposed method uses word embeddings and a deep CNN for local app classification. (Section. III-C)
- By integrating features extracted from the text description by the CNN and multi-modal features, we build a deep learning-based classifier for local mobile apps. (Section. III-B, III-D)
- We conduct an experiment on a real-world dataset collected from Google Play to validate the effectiveness of the proposed method. The experimental results demonstrate that the proposed method outperforms a state-of-the-art method [3]. The F1 score of the proposed method is higher than the baseline by 5.5%. (Section. IV)

This paper is organized as follows. In section 2, we overview the related work for mobile app analysis and deep learning for text classification. We explain the proposed method in section III. Section IV describes the evaluation and application for industry. Finally, we conclude this paper in Section V.

II. Related Work

A. Mobile App Market Analysis and Classification

Several studies have examined mobile app classification. For example, Zhu et al. classified mobile apps based on app name and web search snippets queried by app name [2]. They considered an app name and search snippet as a document. As mentioned in Section I, app names and search snippets are short text; thus, the data sparseness problem can arise when using such short text data. Thus, Zhu et al. used both term frequency (bag-of-words) and topic distribution based on Latent Dirichlet Allocation [9] as a textual features. In addition to textual features, they used real-world context log data, such as time or location information. Finally, they classified mobile apps using Maximum Entropy Model with textual features (term frequency and topic distribution) and real-world context features. Li et al. [3] classified mobile apps using a Naive Bayes Classifier and the same features used by Zhu et al.

Chen et al. proposed a tagging method for mobile app based on app market data [10]. In their method, ten kinds of modalities from an app market, such as app name, app description, ratings and screenshot were used as features for tagging, and tagging was performed by calculating the similarity between a tagged mobile app and the target mobile app.

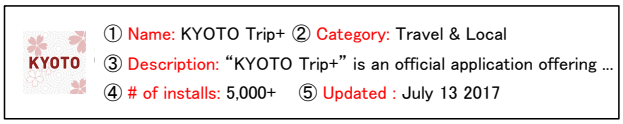


Fig. 1. Example of app market data

B. Deep Learning for Text Classification

In the past few years, CNNs have demonstrated state-of-the-art results with natural language processing tasks, including text classification [4], [6] and sentiment analysis [5]. In these studies, word embedding is first applied, and then an embedding vector is used for the CNN. Word embedding frameworks such as word2vec [11], Glove [12] and fastText [13] are typically pre-trained using public data (e.g. Wikipedia) or a task-specific corpus [6]. In a study by Lee et al. [6], one of the main research contribution was a comparison of the performance of different datasets for pre-training relative to medical documents to detect adverse drug events in tweets. The results demonstrated that task-specific pre-training was superior to fastText trained on Twitter. Thus, data selection for pre-training is an important consideration for text classification using a CNN.

To the best of our knowledge, our study differs from previously proposed methods as follows.

- Existing studies can be divided as follows: (1) classifying mobile apps by short text classification based on app name, description or app usage log, and (2) tagging mobile apps based on muti-modal features. Thus, this study is the first to classify local mobile apps.
- No existing study considered text classification using a CNN that exploits Google Play data for pre-training.

III. Proposed Method

In this section, we describe the proposed feature generation and classifier training methods. We use the following types of features in the proposed method.

1) Features based on various app market modalities.
2) Textual features extracted from description using a CNN.

Note that we evaluated the proposed method using data crawled from Google Play; thus, we explain the feature generation method in the context of Google Play.

A. App Market Data

Figure 1 shows an example of app market data. Here, the app market provides app name, description, number of installs, etc. The names of modalities are shown in red. Note that we generate features based on the app name, category, description, number of installs, and updated modalities.

TABLE I
LIST OF HAND-CRAFTED MULTI-MODAL FEATURES

Source	Feature
App name	# of occurrences of prefecture names
App name	# of occurrences of location names
Description	# of occurrences of prefecture names
Description	# of occurrences of location names
Category	Category
# of installs	# of installs (categorical)
Updated date	Updated year

B. Multi-modal features

Table I shows the features used in the proposed method. We extract the location name from the app name and description, and we use the number of location names as a feature based on the hypothesis that many place names appear in the names and descriptions of local apps. Although there is no explicit local app category, various categories, such as travel, can be considered similar to a local app; therefore we use app category as a feature. We adopt the number of installs under the hypothesis that the number of users of apps used in a specific area is less than that used nationwide. In Google Play, the number of install is represented as a range, such as "1,000-5,000" or "10,000-50,000". Thus, we treat the number of install as categorical feature.

C. Feature extraction using CNN

We use a CNN to extract effective textual features (CNN-feature) from app description. Generally, collecting training data for deep learning is very expensive because deep learning, including CNNs, requires vast amounts of training data. Therefore, semi-supervised or transfer learning methods are often used. To construct a high-performance classifier with such methods, the model is pre-trained using publicly available data and fine-tuned using a small amount of training data. For example, for image recognition, the ImageNet [7] database is used to pre-train various tasks [14]. However, there is no publicly available dataset for training the app classification task. Therefore, we focus on the mobile app registration procedure. When a developer registers their app in an app market, they must manually select a label from pre-defined categories. Thus, we can collect large-scale manually annotated data from an app market. In this study, pre-training is performed by estimating the app category from the app description, and fine-tuning is performed by classifying local apps based on the app descriptions. Our intuition for using app category which is manually labeled by app developer is that because several categories such as "Maps & Navigation" and "Travel & Local" are strongly related to local app, we can efficiently learn the deep learning model by classifying these app categories as pre-training.

The CNN architecture used in the proposed method is shown in Figure 2(a). In the first embedding layer, each word in the app description is converted into a distribution representation. Then, three convolution layers and three max pooling layers are added. Finally, category classification is performed by a fully-connected layer via a flatten layer. Dropout [15] is used for generalization in the fully-connected layer. The embedding layer can be initialized by the distributed representation using, for example, word2vec trained on a public dataset. A Rectified Linear Unit (ReLU) [16] is used as an activation function in the convolutional layers and a softmax function is used in the fully-connected layer. By using softmax function, the output of each unit can be treated as the probability of each label. Here, N is the number of units of the output layer (i.e. the total number of categories), x is input and x_i is the output of unit i. Then, the output p_i of unit i is defined as follows.

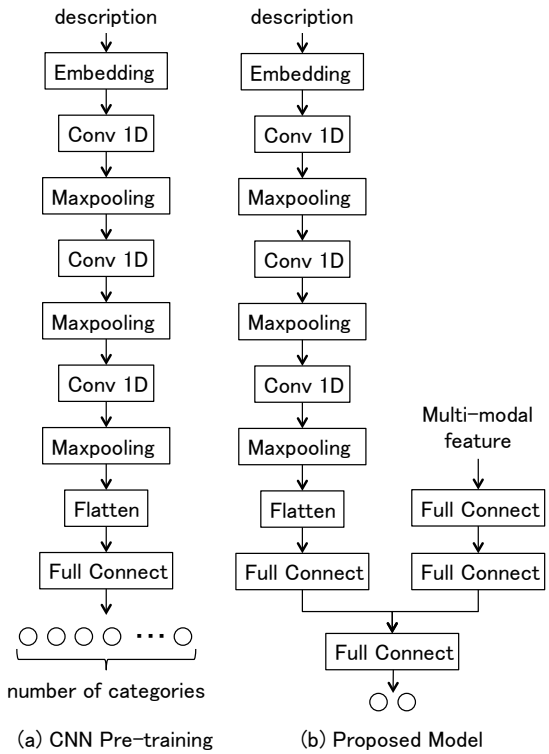


Fig. 2. Structure of deep neural network

$$p_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{1}$$

In the pre-training phase, we train the model to predict the category of each app. Cross-entropy is used as the loss function of classifier L_{CLF} as follows.

$$L_{CLF}(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij} log(p_{ij}) \tag{2}$$

where n is sample size, m is the number of classes, p_{ij} is the output of the classifier of class j of the ith sample and y_{ij} is an annotated label of class j of the ith sample. The weight of neural network $\boldsymbol{w}$ is trained by minimizing Eq.(2) using Stochastic Gradient Descent (SGD) [17].

D. Building Classifier (fine-tuning)

Here, we describe building the classifier for local apps using multi-modal features and CNN features. The structure of the proposed deep neural network is shown in Figure 2 (b). The pre-trained weight is used in the CNN component (left), and multi-modal features are input to the fully-connected layer (right). Then, the multi-modal features and CNN features are concatenated and fed to fully-connected layer. Finally, the output is a binary classification, i.e. the app is local or not local. The parameters of the proposed method except for fully-connected layers are fixed, and the parameters of fully-connected layers (a total of three layers: one layer from CNN part, and two layers for multi-modal features) are fine-tuned. The ReLU is used as an activation function for all layers except the final fully-connected layer, which uses softmax function as an activation function. Similar to the pre-training process, the weight of the neural network $\boldsymbol{w}$ is learned using SGD by setting cross-entropy as the loss function. The details of parameters for the proposed model is described in the next section.

In our proposed model, we use both hand-crafted feature and automatically extracted feature based on CNN because hand-crafted features and CNN-features have different characteristics. Domain knowledge is required hand-crafted features but can be trained by small amount of training data. On the other hand, abstract feature can be learnt by CNN model but large amount of training data is needed for model training. Several existing studies [18], [19] exploit these different characteristics to achieve higher accuracy by integrating these features in deep learning model. In our study, we adopt same strategy for achieving higher accuracy.

IV. Evaluation

A. Experimental Setup and Data

In our evaluation, we used the data of 400,000 mobile apps used in Japan. These data, which we refer to as GP data, were crawled from Google Play between 6/28/2016 and 3/11/2017. We used Mecab[5] as Japanese morphological analyzer to separate sentences into a set of words. Approximately 100,000 apps in the top 20 ranked categories regarding number of registered apps in the GP data were used for pre-training. In other words, pre-training is performed by 20 classes classification using 100,000 apps. We call this dataset as "pre-training data." Table II shows the category and number of apps for each category regarding pre-training data. We annotated 1,518 apps as training data (for fine-tuning), including 759 apps for each positive/negative instance (called "training data"), and 500 apps for hold out test data (called "test data"), including 84 positive instances (local apps) and 416 negative instances (non-local apps). Table III shows the category and number of apps for each category regarding fine-tuning data. The number of positive and negative samples in training data is equal because imbalanced data is problematic in training phase. On the other hand, the test data were imbalanced because, although we cannot know the exact ratio, the number of non-local app is greater than that of local apps in a practical setting.

[5]http://taku910.github.io/mecab/

TABLE II
Category and number of apps in pre-training (Top 20 only)

Category	# of apps	Category	# of apps
Entertainment	15016	Social	2275
Personalization	14979	Business	2095
Tools	12638	Sports	1588
Lifestyle	12181	Finance	1503
Education	10405	Communications	1435
Photography	4968	Shopping	1410
Music & Audio	4281	Medical	1408
Travel & Local	3478	Books & Reference	1385
Productivity	3324	News & Magazines	1343
Health & Fitness	2985	Video Players & Editors	1303

TABLE III
Category and number of apps in fine-tuning (Top 20 only)

Category	# of apps	Category	# of apps
Lifestyle	216	Finance	40
Travel & Local	187	News & Magazines	30
Entertainment	132	Action	29
Education	100	Simulation	27
Tools	90	Productivity	27
Maps & Navigation	75	Photography	25
Personalization	74	Casual	24
Sports	72	Music & Audio	24
Puzzle	52	Health & Fitness	19
Business	40	Adventure	17

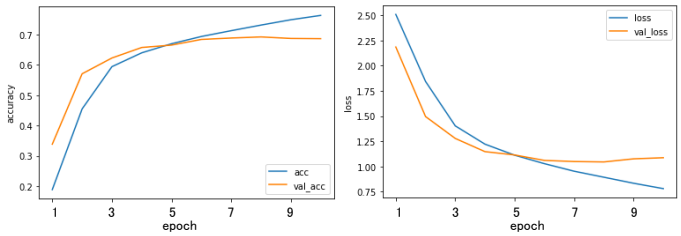
Since a CNN requires fixed length input, we used the first 1,000 words of the app description. The word embedding dimension was 300, the number of feature maps was 128 and the filter size is five. We used fastText trained using Japanese Wikipedia for initialization of embedding layer. The learning rate for SGD is set to 0.01, weight decay parameter is 1e-4, momentum is 0.9 and we used Nesterov momentum [20]. We tuned the model parameters in each model (i.e. baseline and the proposed) and compared the best performance results of each method.

The process of training and test the classification model consists of the following three steps.

- Step1: Pre-training CNN model (described in Fig. 2 (a)) using pre-training data.
- Step2: Fine-tuning the proposed deep model (described in Fig. 2 (b)) using multi-modal features and CNN-features using training data. CNN part of the proposed model is initialized by pre-trained weight.
- Step3: Evaluating the proposed model using test data.

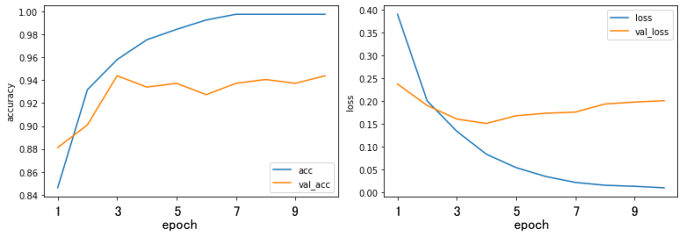
B. Compared Method

We implemented a similar method proposed by Li et al. [3] for comparison. That method uses (1) features based on term frequency, and (2) features based on the document topic. Note that the latter is used to avoid data sparseness. Specifically, the topic probability distribution of each document is used as a feature. In our implementation, we used TF-IDF [21] for term frequency-based features and the topic distribution of each document calculated using LDA for topic based features.



(a) Accuracy of Pre-training (b) Learning curve of pre-training

Fig. 3. Accuracy and loss for pre-training



(a) Accuracy of fine-tuning (b) Learning curve of fine-tuning

Fig. 4. Accuracy and loss for fine-tuning

To decide the number of topics for LDA, we evaluated the number of topics for 5, 10, 20, 50. Then, we selected 20 as the best parameter based on perplexity.

C. Evaluation Metrics

We used recall, precision and F1 as quantitative metrics. Recall is the ratio of positives that are correctly classified to actually positive samples. Precision is the ratio of the number of correctly classified positive samples to the number of the samples predicted as positive, and F1 is the harmonic average of precision and recall.

D. Evaluation Results

Figure 3 shows the accuracy and loss for pre-training. From this figure, we can see that the learning converged. Figure 4 shows the accuracy and loss for fine-tuning. As can be seen, accuracy is improved each epoch. Table IV shows the quantitative evaluation results (bold indicates the best result). Here, Multi-modal feature + RF indicates that the classifier is Random Forest and the feature is multi-modal feature, and Multi-modal feature + MLP indicates that the classifier is a Multi Layer Perceptron. CNN model means that the classifier consists of only CNN architecture described in Fig. 2 (a). CNN model (w/o pre-train) is a CNN model trained using only training data, and CNN model (w/ pre-train) is a CNN model with pre-training of classifying Google Play categories fine-tuned by local app classification using pre-training data. Proposed Model is the proposed deep neural network with both multi-modal features and the CNN feature with pre-training of classifying Google Play categories fine-tuned by local app classification (Fig. 2 (b)).

From Table IV, the proposed model achieved the best classification performance among all models. When using the hand-crafted multi-modal features, the proposed model improves each metric by 1-3% over the baseline method. By comparing the CNN feature with and without pre-training, it can be seen that pre-training of Google Play category is effective. Finally, the proposed method outperformed the baseline method by 5.5-6% regarding F1 score and precision.

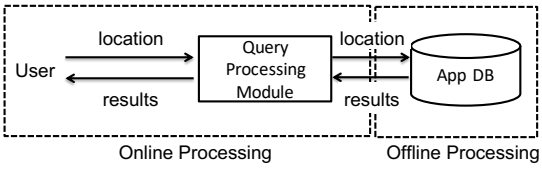


Fig. 5. Architecture of our service

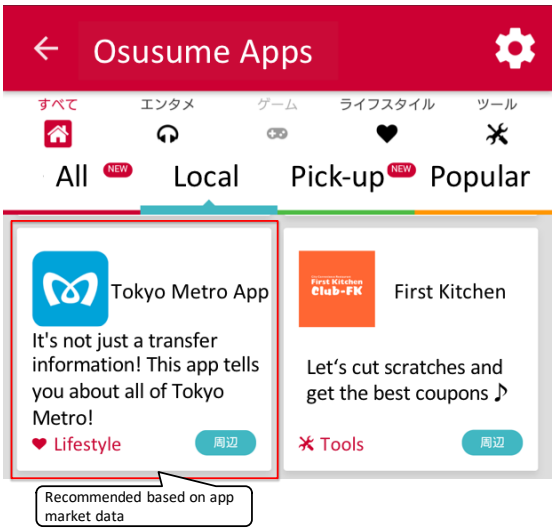


Fig. 6. Screenshot of our app recommendation service

E. Industrial Application

Part of our work has been used as one of a recommendation method in our commercial mobile app recommendation service, called "Osusume Apps (App Recommendation)."[6] Several mobile apps are recommended based on the current location of a user who consent to terms of service. The architecture of the service is described in Fig. 5. Local apps are classified using our method and stored in App DB as offline processing. We used app market data from our app market service, called "d apps & review."[7] When a user receives recommendation from our service, the current location of the user is submitted as query in the online processing. Example of screenshot of our application used in Tokyo area is shown in Fig. 6. "Tokyo Metro App" is recommended based on our method for a user at Tokyo area.

V. Conclusion

In this paper, we have proposed a classification method for local mobile app using multi-modal feature and a CNN feature.

[6]https://www.nttdocomo.co.jp/service/osusume_appli/index.html (Japanese Only)

[7]https://app.dcm-gate.com/ (Japanese Only)

TABLE IV
EVALUATION RESULTS

Method	Precision	Recall	F1
Li et al. (2016) [3]	0.884	0.868	0.874
Multi-modal feature + RF	0.907	0.878	0.887
Multi-modal feature + MLP	0.916	0.902	0.907
CNN model (w/o pre-train)	0.703	0.582	0.629
CNN model (w/ pre-train)	**0.930**	0.892	0.901
Proposed Model	**0.930**	**0.928**	**0.929**

The multi-modal feature was based on the number of location names in the app name and description, and the number of installs, etc. The CNN feature was generated from the app description using a CNN. We focused on the fact that each app is labeled by developer and exploit this information to pre-train the CNN. The evaluation results on a real-world dataset of Google Play data demonstrated the effectiveness of the proposed method. For example, the F1 score of the proposed method is 5.5% greater than that of the baseline. Part of our method is put into practical use of our app recommendation service.

In future, we would like to apply the proposed method to other type of app classification task, such as malware classification.

REFERENCES

[1] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "Simapp: A framework for detecting similar mobile applications by online kernel learning," in *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, ser. WSDM '15. New York, NY, USA: ACM, 2015, pp. 305–314. [Online]. Available: http://doi.acm.org/10.1145/2684822.2685305

[2] H. Zhu, E. Chen, H. Xiong, H. Cao, and J. Tian, "Mobile app classification with enriched contextual information," *IEEE Transactions on Mobile Computing*, vol. 13, no. 7, pp. 1550–1563, 2014.

[3] X. Li, Y.-h. Lian, and H. Yu, "Classification of mobile apps with combined information," in *IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2016, pp. 193–198.

[4] Y. Kim, "Convolutional neural networks for sentence classification," in *Proceedings of EMNLP*. Association for Computational Linguistics, 2014, pp. 1746–1751.

[5] A. Severyn and A. Moschitti, "Twitter sentiment analysis with deep convolutional neural networks," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA: ACM, 2015, pp. 959–962. [Online]. Available: http://doi.acm.org/10.1145/2766462.2767830

[6] K. Lee, A. Qadir, S. A. Hasan, V. Datla, A. Prakash, J. Liu, and O. Farri, "Adverse drug event detection in tweets with semi-supervised convolutional neural networks," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 705–714. [Online]. Available: https://doi.org/10.1145/3038912.3052671

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition, 2009. CVPR 2009.* IEEE, 2009, pp. 248–255.

[8] S. Zhang, L. Yao, and A. Sun, "Deep learning based recommender system: A survey and new perspectives," *arXiv preprint arXiv:1707.07435*, 2017.

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[10] N. Chen, S. C. Hoi, S. Li, and X. Xiao, "Mobile app tagging," in *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, ser. WSDM '16. New York, NY, USA: ACM, 2016, pp. 63–72. [Online]. Available: http://doi.acm.org/10.1145/2835776.2835812

[11] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[12] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of EMNLP '14*, 2014, pp. 1532–1543.

[13] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv:1607.04606*, 2016.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678. [Online]. Available: http://doi.acm.org/10.1145/2647868.2654889

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.

[17] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[18] G. Li and Y. Yu, "Visual saliency detection based on multiscale deep cnn features," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5012–5024, 2016.

[19] S. Zheng, X. Li, A. Men, X. Guo, B. Yang *et al.*, "Integration of deep features and hand-crafted features for person re-identification," in *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*. IEEE, 2017, pp. 674–679.

[20] Y. Nesterov, "Gradient methods for minimizing composite functions," *Mathematical Programming*, vol. 140, no. 1, pp. 125–161, 2013.

[21] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, USA: Cambridge University Press, 2008.