Detailed Report for NDSC 2019

Team Name: Psychic Learners
Team Members: Sun Yitao, Love Johan Nordling and Gordon Foo

Abstract
In this report, we present a three-layer stacked ensemble for the Product Category Classification Challenge which won 2nd place with a private leaderboard score of 80.653% without any additional data other than the sources provided. The models in the first layer takes inputs ranging from titles, images, text extracted from images and item IDs of the product listings. The output probabilities of the first layer are fed to the second layer. Finally, we do a weighted average of the 2nd layer's output probabilities, with the weights optimised by differential evolution.

1. Introduction

Correct classification of product categories in an e-commerce setting allows consumers to find what they are searching for faster as well as paving the way for doing data analysis and developing recommendation systems. The dataset provided consists of 839017 examples with item ID, product image and title, of which 20.5% are unlabelled test examples. The dataset also provides a superclass of the 58 categories in the image file paths, which we will refer to as the large categories: beauty, fashion and mobile. From our exploratory data analysis, we found that the dataset is highly imbalanced with some categories having tens of thousands of examples while others only having hundreds (Fig. 1). While we made initial attempts to do some dataset balancing by under-sampling and oversampling, it was found that such methods had a negative impact on accuracy which is our only evaluation metric so we did not adopt them in the end.

From the plot of item IDs against categories (Fig. 2), it seemed very likely that item ID may be naturally generated from attributes such as the date of upload since a certain range of item IDs are missing from certain mobile categories. We hypnotise that this is due to certain mobile items not being available before or after a certain date.

We also tried to translate the non-English unique words to English with Google Translate API. We decided to translate only the unique vocabulary as translating each of the 839017 sentences would take too long and be prohibitively expensive with an estimated S$1200 cost at S$15 per million characters. From the detected languages, it can be seen that many of the words are not in English (Fig. 3). However translation of a single word without surrounding context is not very accurate and sometimes brand names and even English words can become mistakenly detected as a non-English language. For future work, it may be better to use n-grams for translation as they may provide a better context for more accurate translations.

Also noteworthy is the 'Others' category in each of the 3 large categories. This category provides significant confusion for the classifiers as some of the examples in the category may actually also belong to other categories.

Fig1. Number of examples in each category

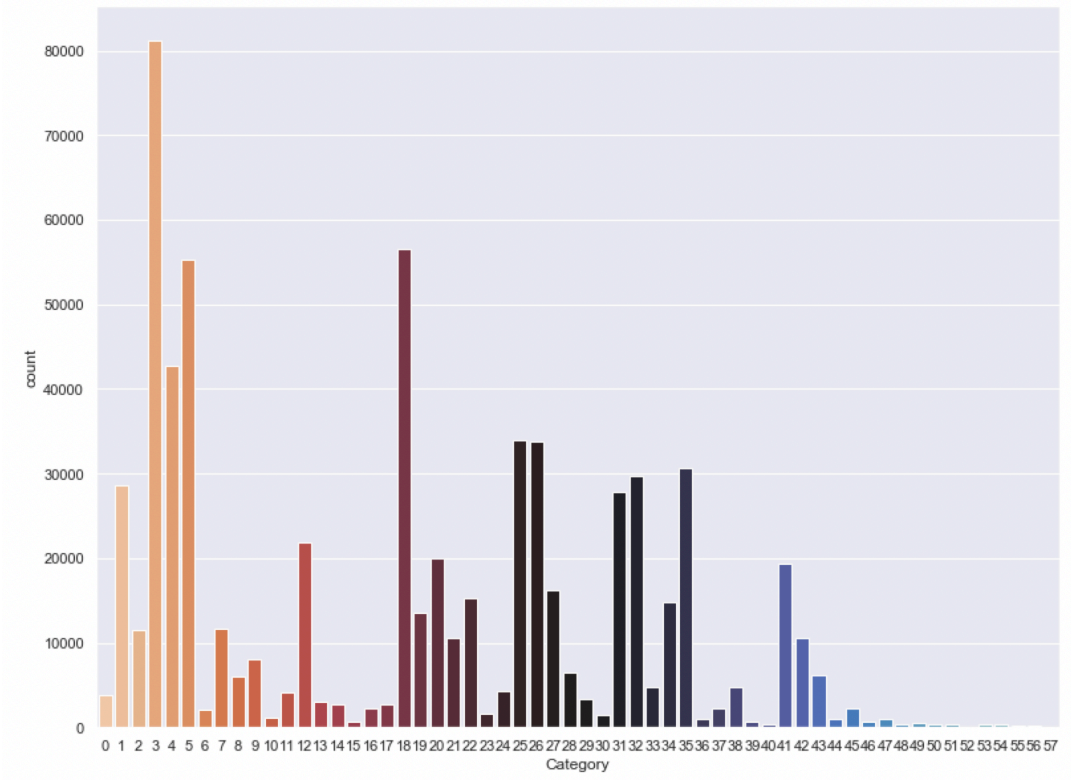


Fig. 2. Item ID values against Category

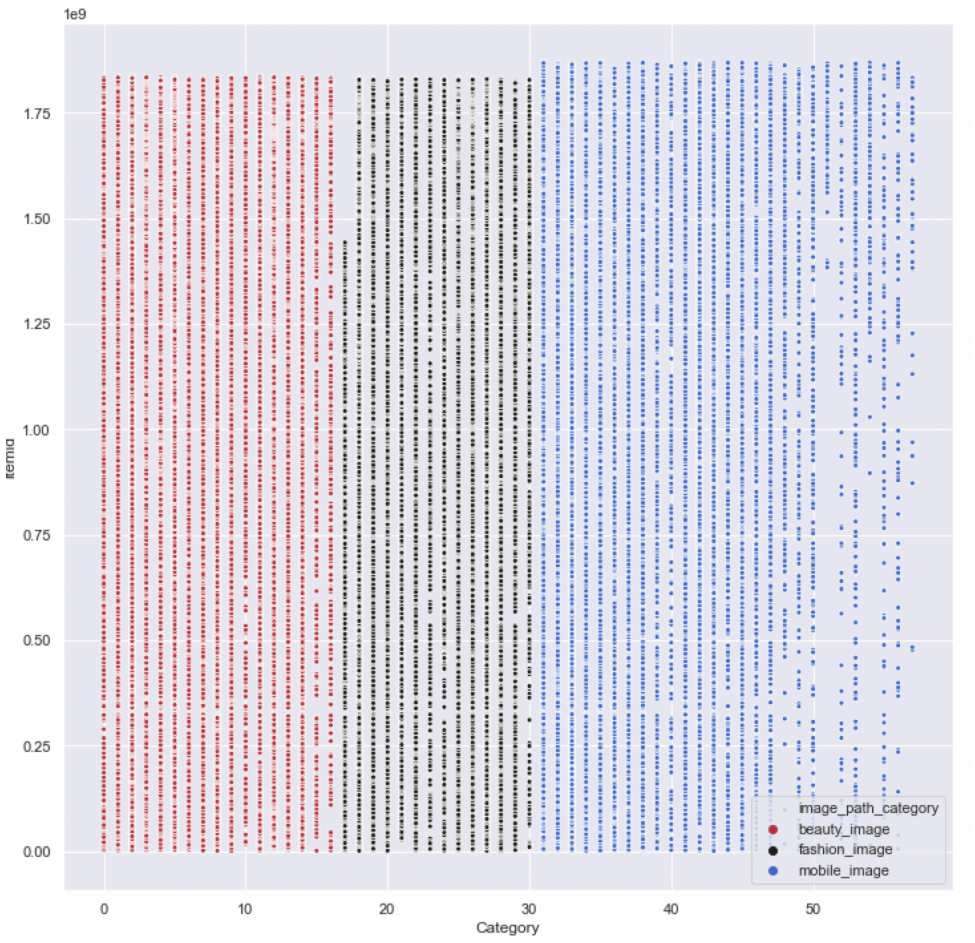


Fig. 3. Number of unique words for the 5 most common detected languages

Language	Unique Word Count
Indonesian	5746
French	1153
Malay	1061
Hindi	942
Italian	934

2. Data Preparation

We used 80% of the labelled data as the training set for the first layer of algorithms, obtained using stratified random split. The first layer is validated against 20% of the remaining labelled data. For the second layer, we did a 4-fold stratified cross-validation on the 20% remaining labelled data to do parameter tuning. Once the parameters are decided, we used 3 of the folds to train the final model and the remaining fold is used to train the final layer. Each split of the data is further split into one of the 3 large categories and a separate model is trained for each large category.

We also used Google's open source Tesseract Optical Character Recognition Engine [1] to extract text from the product images. This text may be words found on the packaging or sentences added by the seller to the original image. As the extractions are very noisy, we only accepted words that are alphanumerical and have a length of more than 2 letters. 59.2% of the train and test examples did not have any extracted text.

3. First Layer Algorithms

Our algorithm consists of a 3 layer stacked ensemble. In the first layer, we implemented 18 models for fashion categories and 19 models for mobile and beauty categories. For model selection, we went for diversity of algorithms rather than simply standalone performance in order to generate predictions with errors that are as uncorrelated as possible. With uncorrelated errors, a meta estimator can learn to overcome each model's weaknesses while with correlated errors, there is little for the ensemble to learn from [2]. While it can be more optimal to tune the parameters of the base models to optimise the accuracy of the entire ensemble [3], due to time constraints we only optimised the parameters for the individual models' accuracies. We made use of a few open source implementations of the following models:

Product title classification using Tobias Lee's Text Classification Repository [4] :

1) Adversarial Training Methods for Semi-Supervised Text Classification [5]
2) Attention-Based Bidirectional Long Short-Term Memory Networks [6]
3) Independently Recurrent Neural Network [7]
4) Multihead Attention Module [8]

Product title classification using Dongjun Lee's Tensorflow implementations of Text Classification Models Repository [9] :

5) Word-level CNN [10]
6) Character-level CNN [11]
7) Word-level Bi-Recurrent Neural Network [12]
8) Recurrent and Convolutional Neural Network [13]

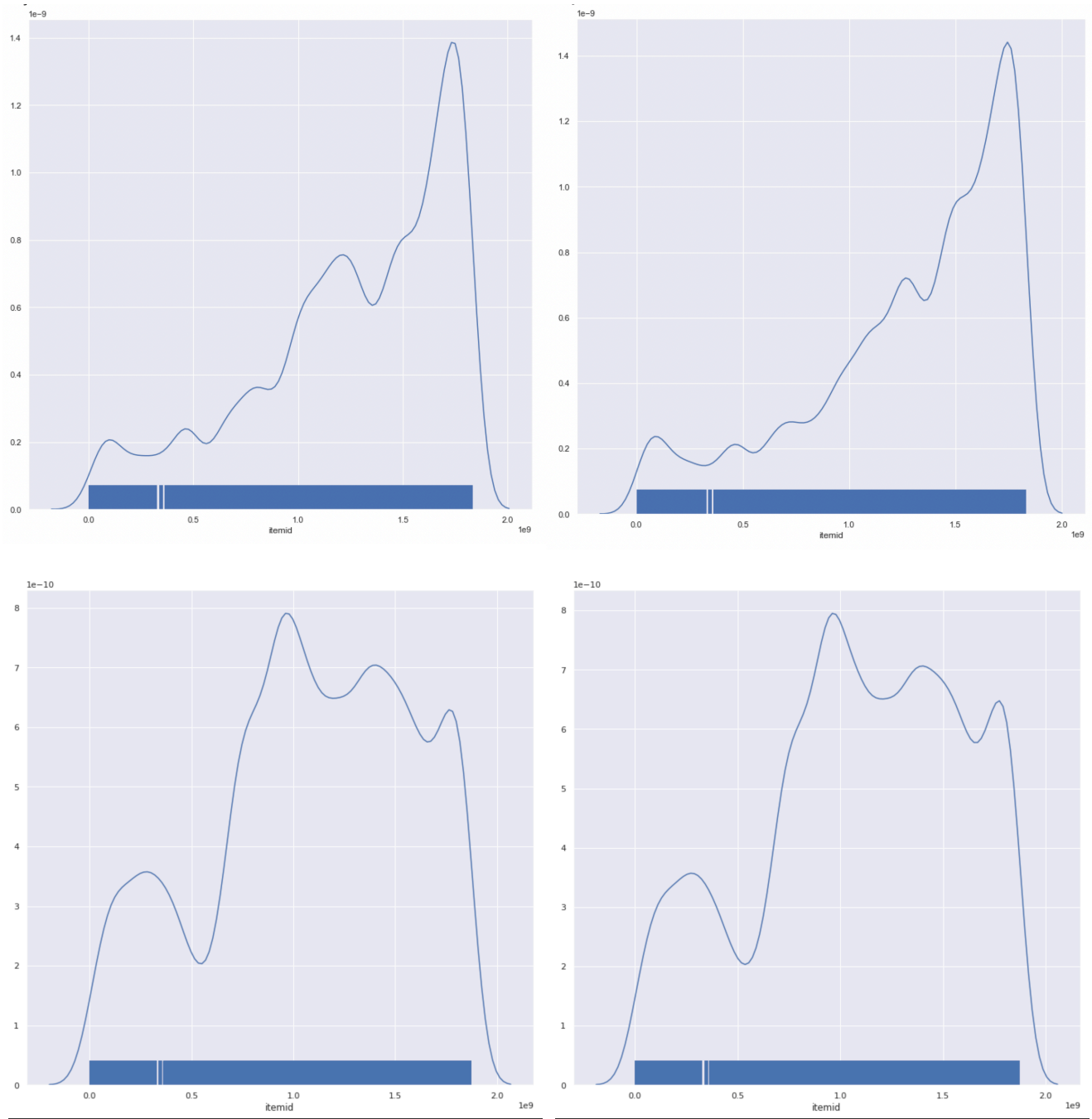
Product title and item ID classification with sci-kit learn library:

9) K-nearest neighbours classification on item IDs with K set as 21 and 4 for beauty and mobile respectively. For fashion products, we found that the distributions of item IDs within 750000000 and 1500000000 does not correlate as much between the test and train sets (Fig. 4) so we used k=400 for values within this range and k=50 for values outside this range.
10) Random Forest on item IDs for beauty and mobile. We did not use this for fashion as we were running out of submissions and did not want to risk wasting a submission for a 0.1% improvement.
11) K-nearest neighbours with K=5 on term frequency-inverse document frequency (TFIDF) word vectors
12) Same as model 11 with K=10

13) Same as model 11 with K=40
14) Logistic Regression on TFIDF word vectors with n-gram range 1 to 10
15) Multinomial Naive Bayes on count word vectors with n-gram range 1 to 10

Others:
16) Fasttext on product titles [14].
17) Fasttext on extracted text from images with pytesseract [15]
18) BERT base multi-language uncased fine-tuning on product titles [16], [17]
19) Image classification: We used a Squeeze-Excitation ResNext50 [18], [19] for fashion images and fine-tuned a pre-trained Inception ResNet V2 [20] for mobile and beauty images, with random erasing augmentation [21-22]. We did not have time to do significant parameter tuning due to the large quantity of data so we just went with what seemed to perform well in the first few epochs. We also resized images to 240x240 to speed up training and used class weights proportional to the inverse of frequency. Training took roughly 50 hours on a P6000 and 2 Tesla T4 GPUs. Image classification had a roughly 10% lower accuracy compared to text classification across the 3 large categories.

Fig. 4. Distribution of item IDs on fashion validation set (top left), fashion test set (top right), mobile validation set (bottom left), mobile test set (bottom right)



4. Second Layer Algorithms

The algorithms on this layer were trained on the output probabilities of the first layer algorithms. Parameter tuning was done with a 4-fold stratified cross-validation. Finally, we trained the second layer models on 3 of the 4 validation folds and saved their output probabilities on the fourth fold. The algorithms used are:

20) XGBoost, we did a Bayesian parameter search which helped improve internal cross-validation scores by 0.11 percent over an initial grid search. This outperforms neural network by about a 0.5% margin.
21) Neural Network, implemented in Keras with 200 first layer units and 3 layers of 48 units with parametric rectified linear units [23] as activation and 0.2 dropout.
22) AdaBoost with Extra Trees as the base estimator. Our initial idea was to build an ensemble with a 2 layer stack with only one meta-algorithm, but during the last week of the competition, we came across a Kaggle post on the winner's solution of the Otto Group Product Classification Challenge [24] which heavily inspired our final approach.

5. Third Layer Weighted Average Ensemble

We opted to do a simple weighted average ensemble of the second layer's output probabilities as we only had one of the 4 validation folds and the public leaderboard score left to train and validate. The weights were obtained using SciPy's differential evolution function to maximise the accuracy on the single validation fold. The index of the weights array corresponds to neural network, XGBoost and AdaBoost respectively.

Beauty Equal Weights Accuracy: 0.823
Optimised Beauty Weights:
[0.22207012, 0.772792, 0.00513788]
Optimised Beauty Weights Accuracy: **0.8272**

Fashion Equal Weights Accuracy: 0.734
Optimised Fashion Weights:
[0.24182756, 0.75672089, 0.00145155]
Optimised fashion Weights Accuracy: **0.7414**

Mobile Equal Weights Accuracy: 0.878
Optimised Mobile Weights:
[0.29369731, 0.58890224, 0.11740045]
Optimised Mobile Weights Accuracy: **0.8799**

Fig. 5a. Validation accuracies

Model No.	Beauty	Fashion	Mobile
1	0.70954	0.56193	0.76918
2	0.72331	0.58544	0.76008
3	0.71638	0.56553	0.75845
4	0.72181	0.57306	0.75571
5	0.78048	0.62308	**0.82398**
6	0.74717	0.57802	0.77364
7	0.74989	0.61059	0.78771
8	0.77109	0.62360	0.79713
9	0.38252	0.38585	0.39940
10	0.38768	NA	0.38400
11	0.73625	0.55319	0.76154

Fig. 5b. Validation accuracies (cont.)

Model No.	Beauty	Fashion	Mobile
12	0.73934	0.56184	0.76634
13	0.71750	0.56138	0.76694
14	0.64757	0.44958	0.64743
15	0.77162	0.61031	0.80311
16	**0.78261**	**0.64343**	0.82335
17	0.39855	0.29389	0.26861
18	0.76706	0.61031	0.80848
19	0.66988	0.52618	0.69649
20	**0.82581**	**0.73966**	**0.87676**
21	0.81737	0.72328	0.87077
22	0.80795	0.70517	0.86154

6. Other algorithms and approaches that we tried but did not add into the final stack due to lack of performance improvement or time:

1) BERT Large English. We only had time to train for 30 epochs, so it may be possible to get better performance with more time or GPUs.
2) Capsule Network [25], [26]. A decent individual performer but failed to add any value to the ensemble.
3) K-nearest neighbours with K set as 5, 10, 20, 80, 160, with TFIDF and count word vectors.
4) Logistic regression on count word vectors.
5) Multinomial Naive Bayes on text extracted from images.
6) Random Forest on count and TFIDF word vectors.
7) XGBoost on count and TFIDF word vectors.
8) Translating all words to English with a dictionary obtained by translating each unique word with Google Translate API.
9) Stemming and lemmatisation of product titles.
10) Using PySpellChecker [27] to correct spelling mistakes.
11) Extracting word embeddings with Embeddings from Language Models (ELMo) [28], [29] feeding it to the second layer algorithms so they can differentiate which first layer algorithms are better at which sentences. However, the training was extremely slow and we had to cancel it.

7. Downloads and Links

1) https://drive.google.com/drive/folders/1ZT3ptVDoqgHGDcWPe-3FA0L64UhJqosh?usp=sharing Fasttext models
2) https://drive.google.com/drive/folders/1ZT3ptVDoqgHGDcWPe-3FA0L64UhJqosh?usp=sharing Our CSVs with extracted text from images
3) https://drive.google.com/open?id=1IDXhF4YwbDK99a5LRkHWzx0ZngPkab96 Checkpoints for our models
4) https://drive.google.com/open?id=1gPG6_6qL5fRxO_s4I0rv111vP3wRoHBY Saved probabilities in .npy format
5) https://github.com/sun-yitao/PsychicLearners Github for this project
6) https://www.kaggle.com/c/ndsc-beginner/discussion/85396 Kaggle post on the solution

References

[1] R. Smith, “An Overview of the Tesseract OCR Engine,” in Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2, Curitiba, Parana, Brazil, 2007, pp. 629–633

[2] L. I. Kuncheva, “Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy,” p. 27.

[3] A. Toscher, M. Jahrer, and R. M. Bell, “The BigChaos Solution to the Netflix Grand Prize,” p. 52.

[4] T. Lee, Implementation of papers for text classification task on DBpedia: TobiasLee/Text-Classification. 2019.

[5] T. Miyato, A. M. Dai, and I. Goodfellow, “Adversarial Training Methods for Semi-Supervised Text Classification,” May 2016.

[6] P. Zhou et al., “Attention-Based Bidirectional Long Short-Term Memory Networks for Relation Classification,” in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Berlin, Germany, 2016, pp. 207–212.

[7] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, “Independently Recurrent Neural Network (IndRNN): Building A Longer and Deeper RNN,” *arXiv:1803.04831 [cs]*, Mar. 2018.

[8] A. Vaswani *et al.*, “Attention Is All You Need,” *arXiv:1706.03762 [cs]*, Jun. 2017.

[9] D. LEE, Tensorflow implementations of Text Classification Models.: dongjun-Lee/text-classification-models-tf. 2019.

[10] Y. Kim, “Convolutional Neural Networks for Sentence Classification,” *arXiv:1408.5882 [cs]*, Aug. 2014.
[11] X. Zhang, J. Zhao, and Y. LeCun, “Character-level Convolutional Networks for Text Classification,” *arXiv: 1509.01626 [cs]*, Sep. 2015.
[12] Z. Yang, G. Xu, and B. Shi, “Bidirectional LSTM-RNN with Bi-attention for reading comprehension” 2017.
[13] S. Lai, L. Xu, K. Liu, and J. Zhao, “Recurrent Convolutional Neural Networks for Text Classification,” p. 7.
[14] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, “Bag of Tricks for Efficient Text Classification,” *arXiv: 1607.01759 [cs]*, Jul. 2016.
[15] “madmaze/pytesseract: A Python wrapper for Google Tesseract.” [Online]. Available: https://github.com/madmaze/pytesseract.
[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *arXiv:1810.04805 [cs]*, Oct. 2018.
[17] TensorFlow code and pre-trained models for BERT. Contribute to google-research/bert development by creating an account on GitHub. Google AI Research, 2019.
[18] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks,” *arXiv:1709.01507 [cs]*, Sep. 2017.
[19] “titu1994/keras-squeeze-excite-network: Implementation of Squeeze and Excitation Networks in Keras.” [Online]. Available: https://github.com/titu1994/keras-squeeze-excite-network.
[20] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” *arXiv:1602.07261 [cs]*, Feb. 2016.
[21] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, “Random Erasing Data Augmentation,” *arXiv: 1708.04896 [cs]*, Aug. 2017.
[22] Y. Uchida, Cutout / Random Erasing implementation, especially for ImageDataGenerator in Keras: yu4u/cutout-random-erasing. 2019.
[23] K. He, X. Zhang, S. Ren, and J. Sun, “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification,” *arXiv:1502.01852 [cs]*, Feb. 2015.
[24] G. Titericz & S. Semenov “Otto Group Product Classification Challenge.” [Online]. Available: https://kaggle.com/c/otto-group-product-classification-challenge..
[25] “leftthomas/CCCapsNet: A PyTorch implementation of Compositional coding Capsule Network based on the paper ‘Compositional coding capsule network with k-means routing for text classification.’” [Online]. Available: https://github.com/leftthomas/CCCapsNet.
[26] H. Ren and H. Lu, “Compositional coding capsule network with k-means routing for text classification,” *arXiv:1810.09177 [cs, stat]*, Oct. 2018.
[27] T. Barrus, Pure Python Spell Checking http://pyspellchecker.readthedocs.io/en/latest/: barrust/pyspellchecker. 2019.
[28] Pre-trained ELMo Representations for Many Languages: HIT-SCIR/ELMoForManyLangs. 哈工大社会计算与信息检索研究中心, 2019.
[29] M. E. Peters *et al.*, “Deep contextualized word representations,” *arXiv:1802.05365 [cs]*, Feb. 2018.