Classification of Multimodal Text and Image Social Media Data

Eyosyas Dagnachew
George Mason University
Fairfax, VA, USA
edagnac@gmu.edu

1 INTRODUCTION

In this paper, we present our implementation of work that has recently been done on classifying text and image multimodal Twitter data on a disaster-related dataset. We conduct six experiments for two classification tasks in which we compare training neural networks on single modality data (text-only and image-only) with training neural networks on multimodal data (both text and image). The multimodal network is trained using a feature level, classification-based fusion approach.[1] Our results closely match the results obtained by the authors of the original paper.

The rest of the paper is organized as follows. In the next section, we provide a full reference of the paper we chose to implement. In section 3, we discuss the source code and other code related details of our implementation. In section 4, we describe the methodology of the authors' implementation, comparing and contrasting with our own methodology. We attempt to closely follow the implementation steps that the authors provided, only deviating from those steps when necessary. Finally un section 5, we discuss the dataset, some experiment details, our results, and some potential improvements that we noticed when implementing the experiments before concluding our report.

2 SOURCE PAPER

The paper we chose to implement is titled "Analysis of Social Media Data using Multimodal Deep Learning for Disaster Response" and is authored by Ferda Ofli, Firoj Alam, and Muhammad Imran [8]. This is a relatively recent paper presented in May 2020 at the 17th International Conference on Information Systems for Crisis Response and Management (ISCRAM). A copy of the paper can be found in the `references` folder that we included with this submission. The following is the Bibtext citation for the paper:

```
@inproceedings{multimodalbaseline2020,
 Author = {Ferda Ofli and Firoj Alam and Muhammad Imran},
 Booktitle = {17th International Conference on
   Information Systems for Crisis Response and Management},
 Keywords = {Multimodal deep learning, Multimedia content,
    Natural disasters, Crisis Computing, Social media},
 Month = {May},
 Organization = {ISCRAM},
 Publisher = {ISCRAM},
 Title = {Analysis of Social Media Data using Multimodal
    Deep Learning for Disaster Response},
 Year = {2020}}
```

[1] Feature level fusion refers to fusion of the features of each modality before a decision for a task is determined (opposite of decision level fusion, which makes decisions using each modality separately and then combines the decisions). Classification-based fusion refers to fusion in which the features of each modality are concatenated before being used as inputs for another model, e.g. a classification model.

3 SOURCE CODE

The language we used was Python (version 3) because it is one of the best languages for machine learning work and has some of the most well documented and utilized machine learning/deep learning libraries. The main libraries we used and for which tasks we used each library are as follows:

- Keras with a TensorFlow (TF) back-end. We used the latest integrated tf.keras version from TF2 for the following:
  - Implementing neural network architectures
  - Training, testing, and saving the models
  - Implementing optimizers, early stopping algorithms, learning rate reducing algorithms, training result visualizations
  - Implementing data generators
- Pandas: used for reading, saving, and manipulating DataFrame objects which contained the text data, paths of the image data, and other metadata
- Numpy: used for manipulating array-like objects
- Pickle: used for saving and loading Python objects (e.g. dictionaries)
- Scikit-Learn: used for creating label encoders and outputting performance metrics
- Gensim: used for building text embeddings (word2vec)
- NLTK: used for preprocessing text data
- other built-in Python libraries for small tasks and any dependency libraries required by the above listed libraries

We used Git for version control, Pip and Conda for package and environment management, and Cookiecutter Datascience for project structure.

There are three main steps and corresponding parts of code that we wrote for each experiment:

(1) Building features: After downloading the dataset (described in Section 5.1), we had to build the features and/or prepare the pipeline that will be used for feeding the data into the models. Initially, we used a Jupyter Notebook to explore the datasets, making sure that the training, validation, and testing data splits were as specified in the paper and that we were able to properly read in the texts, images, and labels. Any code we implemented for building features is found in `src/data/build_[modality]_features.py`.[2] In `build_text_features.py`, the text is preprocessed (details in Section 4.1) and outputted back out with the rest of the original dataset as a .csv file. In `build_image_features.py`, we originally preprocessed the images and outputted them in a .npy file. However, we ran into memory issues during

[2] All paths assume that the current directory is the home directory of the project. Please refer to the README for more details on the project structure.

training caused by trying to read in the .npy files. Therefore, we decided later to perform preprocessing and read in the images on the fly using Keras Sequences[3] as data generators–the code for which can be found in `src/models/custom_dataset.py`. Similarly, we use data generators for training the multimodal model.

(2) Training models: The codes used for training the models (`src/models/train_[modality]_model.py`) are quite straightforward. We first read in the training and validation datasets and/or initialize their data generators. We then initialize the model and any other model related variables (e.g., optimizers, early stopping algorithms, learning rate reducing algorithms, training result visualizers). Finally, we fit the model and save the model with the best weights.

(3) Testing models: The codes used for testing the models (`src/models/predict_[modality]_model.py`) are also simple. We read in the testing dataset and/or initialize its data generator, load in the trained model (i.e., model with the best weights), get predictions of the test data using the model, and output the predictions and the performance metrics.

4 METHODOLOGY

In this section, we describe the methodology used by Ofli et al., pointing out any diversions we made or any other steps we added that were not specified in their paper.

4.1 Preprocessing

For the text data, we preprocess the tweets just as described in the "Data Preprocessing" section of Ofli et al.'s paper. These steps include removing stop words, non-ASCII characters, numbers, URLs, and hashtag signs as well as replacing all punctuation marks with white-spaces. The authors do not specify what list of stop words they remove or what tokenizer they use before removing stop words. We decided to use the English stop words and the TweetTokenizer provided by the standard NLTK library. As described in the "CNN: Text Modality" section of Ofli et al.'s paper, we also convert each tweet into a word-level matrix such that each row of the matrix contains a word-embedding for each word in the cleaned tweet. The word-embeddings, which are of length 300, are generated using a word2vec model pretrained using the Continuous Bag-of-Words approach [7] on a large, 364 million tweet disaster-related dataset [1]. Further details about the word2vec model is provided by Ofli et al. The authors do not specify what to do if a word is not found in the dictionary of the word2vec model, so we generate a random word-embedding of length 300 using a Normal distribution. We then zero-padded the tweets so that there are no more than 25 rows in each word-level matrix (i.e., no more than 25 words per tweet). Ofli et al. perform zero-padding first then convert the tweets into word-level matrices but we reverse the order for ease of coding. Finally, we shuffle the datasets which was not specified by the authors in the paper but is recommended for the training data.

For the image data, we first shuffle the data. After reading in an image, we rescale it to be of size 224x224x3, which the authors do not specify but is required for the next step. The pixels of the image are then scaled to be between 0 and 1 and the channels are normalized with respect to the ImageNet dataset [3].

[3]https://www.tensorflow.org/api_docs/python/tf/keras/utils/Sequence

4.2 Text Model

The text model used by Ofli et al. is similar to the one specified by Yoon Kim in the 2014 paper titled "Convolutional Neural Networks for Sentence Classification" [5]. The input simultaneously goes through three convolutional layers with 100, 150, and 200 filters and corresponding window sizes of 2, 3, and 4, respectively. Each of the convolutional layers are followed by max-pooling layers which have pooling length equal to the corresponding convolutional layer's filter window size. The outputs of the max-pooling layers are concatenated and passed through fully-connected layers. The authors do not specify how many fully-connected layers the resulting feature representation is passed through (they say "one or more"). We pass the concatenated feature representations from the convolutional layers through two fully connected layers with 100 and 50 units and finally an output layer with as many units as there are classes for the task (i.e., two for informative task and 5 for the humanitarian task. Dropout with rate 0.02 is used before the fully-connected layers. The outputs from the convolutional layers, the max-pool layers (after concatenation), and the hidden fully-connected layers are sent through ReLU activation functions. The last layer uses the Softmax activation function. The authors also mention that they perform batch normalization but it is not clear if this was done for the text-only model or for both the text and multimodal models. We chose to only perform batch normalization for the multimodal model.

During training, a learning rate of 0.01 is used with the Adam optimizer [6]. An early-stopping criterion based on the validation accuracy is set with a patience of 10, and the model is trained for 50 epochs. The authors do not specify a batch size, so we used a batch size of 128.

4.3 Image Model

The image model used is VGG-16 [9], which is initialized with weights pre-trained on the ImageNet dataset [3]. The original 1000-unit output layer is changed to have as many units as number of classes in the task (tasks are descibed in Section 5.1) and uses the Softmax activation function. During training, a learning rate of 10^{-6} is used with the Adam optimizer [6]. An early-stopping criterion based on the validation accuracy is set with a patience of 10. The authors write that they reduce the learning rate by a factor of 0.1 when the validation accuracy stalls improving for 100 epochs and train for a maximum of 1,000 epochs. This is excessive and unnecessary for transfer learning an image model, so we trained for 10 epochs with no changes to the learning rate (since it is already low). The authors do not specify a batch size, so we used a batch size of 4 because we were initially having memory issues. However, we could have likely used a higher batch size. The authors also do not specify if they froze the base model and only trained the fully connected layers or if they trained the model end-to-end. We chose to train the model end-to-end.

4.4 Multimodal Model

The multimodal model combines the text and image models mentioned in the previous sections. For the text part of the model, we use the convolutional and max-pool layers as mentioned before. Again, the authors do not specify how many fully connected layers there are (or how many units each layer has) for the text model. This time we used a 2000-unit fully connected layer because the next step requires adding a 1000-unit fully connected layer to both the text model and the image model (replacing the last layer of VGG-16), which the authors suggest is used to create an equal distribution from both modalities. The outputs of the fully-connected layers for both modalities are then passed through ReLU activation functions (which is not mentioned but assumed from previous text). Then we use batch normalization for both modalities (which was hinted at but was not clear). Finally, the outputs from the separate models are concatenated and normalized again. Finally, this concatenated output is ran through dropout with a rate of 0.4, a 500-unit fully-connected layer with the ReLU activation function, dropout with a rate of 0.2, a 100-unit fully-connected layer with the ReLU activation function, dropout with a rate of 0.02, and a fully-connected output layer with the same number of units as classes for the task and Softmax activation function. Again, the exact dropout rates and number of units in fully-connected layers (expect for the output layer) are not specified but implied based on Figure 2 of their paper or chosen by us.

During training, the Adam optimizer is used, but the learning rate is not specified. We used 0.00001 because a higher learning rate exploded the gradients. The authors used a batch size of 32, but we ran into memory issues when training and used a batch size of 16. The authors mention that they do not tune any hyper-parameters, which mean that they could have used the validation data for training. However, we chose to follow their steps here. The authors also mention that they use an early-stopping criterion, but no other information about training is specified. We decided to use an early-stopping criterion based on the validation accuracy with a patience of 10, learning rate reduction by a factor of 0.1 when the validation accuracy stalls improving for 5 epochs and train for a maximum of 50 epochs. The model finished training after 28 epochs for both tasks.

5 EXPERIMENTS

5.1 Dataset

The authors conducted experiments on the CrisisMMD dataset, a real-world, disaster-related Twitter dataset collected during seven major natural disasters [2].[4] The data is labeled for two classification tasks. The first task is to determine if a tweet is informative (IN) or not informative (NI) for humanitarian aid. The second task is to determine if a tweet contains information related to one of five different categories: affected individuals (A), rescue volunteering or donation effort (R), infrastructure and utility damage (I), other relevant information (O), or not-humanitarian (N). There are a total of 11,400 texts (I: 7,631; N:3,768) and 12,708 images (I: 8,431; N:4,277) used for the informative task and a total of 7,216 texts (A: 88; R: 1,037; I: 657; O: 1,666; N: 3,768) and 8,079 images (A: 89; R: 1,187; I: 733; O: 1,753; N: 4,277) used for the humanitarian task.[5] The dataset is broken down into training, validation, and testing sets using a 70%, 15%, and 15% split. The dataset is clearly not balanced, but the authors do not mention any steps taken to address that.

Table 1: Scores for the informative task by Ofli et al.

Modality	Accuracy	Precision	Recall	F1-score
Text only	80.8	81.0	81.0	80.9
Image only	83.3	83.1	83.3	83.2
Multimodal	84.4	84.1	84.0	84.2

Table 2: Scores for our informative task experiment

Modality	Accuracy	Precision	Recall	F1-score
Text only	83.9	84.0	83.9	83.1
Image only	84.0	83.7	84.0	83.7
Multimodal	83.6	84.1	83.6	83.8

Table 3: Scores for the humanitarian task by Ofli et al.

Modality	Accuracy	Precision	Recall	F1-score
Text only	70.4	70.0	70.0	67.7
Image only	76.8	76.4	76.8	76.3
Multimodal	78.4	78.5	78.0	78.3

Table 4: Scores for our humanitarian task experiment

Modality	Accuracy	Precision	Recall	F1-score
Text only	74.6	74.5	74.6	74.2
Image only	78.6	78.0	78.6	77.8
Multimodal	78.3	79.0	78.3	77.3

5.2 Experiment Details and Results

In total, the authors conducted six experiments, all of which we replicated. Three experiments were conducted on the informativeness classification task and three on the humanitarian classification task. The three experiments for each task were classification tasks using only text data, only image data, and multimodal (both text and image) data using the three models specified in the previous section. The authors use F1-score as their main performance metrics. However, they provide the confusion matrices as well as the accuracy, precision, and recall scores for each of the six experiment, which we also output during our experiments. Table 1 and 3 contains the scores provided by the authors and Table 2 and 4 contains the score from our experiments. We decided to remove the confusion matrices from this report to decrease the length of the report; however, the our confusion matrices can be found in the files located in the `reports` folder and Ofli et al.'s matrices can be found on page 7 and 8 of their paper.

[4] https://crisisnlp.qcri.org/crisismmd

[5] Some texts correspond to multiple images.

As shown in the tables, all of our results for the text-only and image-only experiments were better that the results from Ofli et al. The result for the text-only experiment is likely due to the decisions we made for parts of the text model implementation that were not specified by the authors (i.e., the number of fully connected layers and number of units for each layer). The result for the image-only experiment is likely due to our decision to only train the model for 10 epochs, rather than their 1000 epochs which most probably caused overfitting to occur on the training data. Our scores for the multimodal experiments were fairly similar to those from Ofli et al. Overall, all of the scores from our experiments were relatively close to the scores from the authors' experiments.

5.3 Potential Improvements

Although the authors mentioned rooms for improvement when it comes to tuning hyperparameters and model selection, there are several other modifications we would make to the experiments to potentially further improve the results. The following is a list of some of the changes we would make:

- We would conduct more text cleaning steps during preprocessing. For example, we would remove "RT" and the account mentioned next to it for retweets, which would remove noise and potentially improve performance.
- For the text modality, we would use a state-of-the-art NLP model such as a transformer-based model, which better understand context and might improve performance.
- We would tune the hyperparameters during training and experiment with the model selection (e.g., try different sized/number of fully connected layers for the text and multimodal models).
- We would balance the datasets or use some sort of weighting during training to make sure each class receives equal representation.
- We would use a more recent state-of-the-art CV model such as one based on residual networks [4].
- We could attempt using a decision-level fusion approach in which we combine the local decisions from each single modality model via some rule, e.g., pick the decision of the model with higher confidence.

6 CONCLUSION

In this paper, we gave an overview of the results we found after replicating a paper by Ofli et al. in which we classified single modality and multimodal Twitter data for two disaster-related tasks. We provided code and methodology related details of our implementation, and we reported that we successfully replicated the results, pointing out numerous paths for potential improvement.

REFERENCES

[1] Firoj Alam, Shafiq Joty, and Muhammad Imran. 2018. Graph based semi-supervised learning with convolution neural networks to classify crisis related tweets. In *Twelfth International AAAI Conference on Web and Social Media*.

[2] Firoj Alam, Ferda Ofli, and Muhammad Imran. 2018. CrisisMMD: Multimodal Twitter Datasets from Natural Disasters. In *Proceedings of the 12th International AAAI Conference on Web and Social Media (ICWSM)* (USA, 23-28).

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[5] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1746–1751. https://doi.org/10.3115/v1/D14-1181

[6] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[7] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[8] Ferda Ofli, Firoj Alam, and Muhammad Imran. 2020. Analysis of Social Media Data using Multimodal Deep Learning for Disaster Response. In *17th International Conference on Information Systems for Crisis Response and Management*. ISCRAM, ISCRAM.

[9] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).