

This is Going Swimmingly!

Catherine Yu (cyu83), Isabelle Shapiro (iashapir), Aaron Martin (amart172), Jake Regenwetter (jregenwe)

Introduction

Our project is based off of a paper implementing a relational context learning and multiplex fusion network to address the task of multimodal sarcasm detection, in other words, classifying image/text pairs as sarcastic or not. This classification task is important because it can help filter misinformation online, and has also proven helpful for other machine learning tasks such as sentiment analysis. Our chosen paper focuses on implementing sarcasm detection without the use of graph structures, which are used by many existing sarcasm detection models. Using graph structures presents several limitations, the most important of which is that it is extremely computationally expensive to create graph networks modeling images and text. Instead, the paper's model aims to understand the dynamic relationships between image and text pairs to capture the context needed to classify something as sarcastic or not. We chose to reimplement this paper because it addresses a problem that we are all interested in and it builds off of concepts we've discussed previously in class such as various types of attention and classification problems.

Methodology

Data

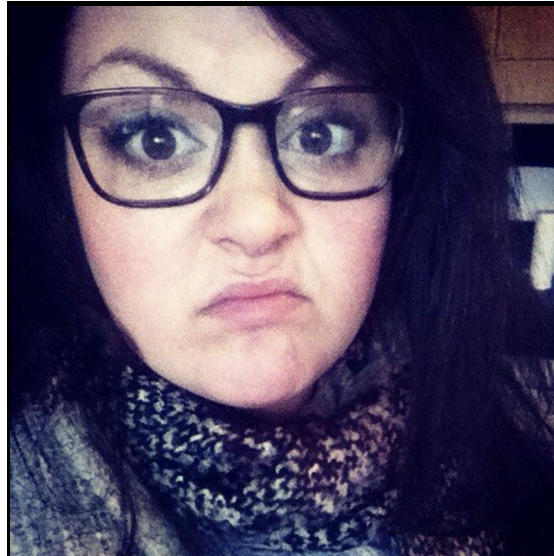
In the original RCLMuFN paper, the model was trained on the MMSD and MMSD 2.0 datasets. The MMSD dataset contains pairs of images and related text sourced from social media platforms, some of which are sarcastic in nature, while others are not. The paper's original model was also trained on the MMSD dataset, which contains emojis and hashtags, in addition to MMSD 2.0. That model mainly focused on the presence of tags like “#sarcasm” to make predictions instead of the actual substance of the posts. We chose to instead use MMSD 2.0, which removes those cues.

We trained and tested our model on the MMSD 2.0 dataset, which contains a total of 24,635 samples, divided into 19,557 training, 2,387 validation, and 2,373 test samples. The text is broken down using the BERT tokenizer, which splits sentences into subword units. All tokenized sequences were padded or clipped to a maximum length of 77 tokens. Text features were encoded to 768-dimensional vectors using BERT, and images were resized to 224x224 pixels, and encoded using ResNet-50 to produce 768-dimensional embeddings for CLIP compatibility.

We also tested the model on a few other datasets. We created a custom testing dataset by combining the MORE dataset with the Flickr 8k captioned image dataset. MORE, created by researchers at Cornell for their own Multimodal Sarcasm Explanation model, contains images with only sarcastic descriptions, while none of the Flickr8k data are sarcastic. We took the training split from MORE (2983 samples), 2993 random samples from Flickr8k, and shuffled them together to create our custom testing set. Additionally, we tested the model on SarcNet,

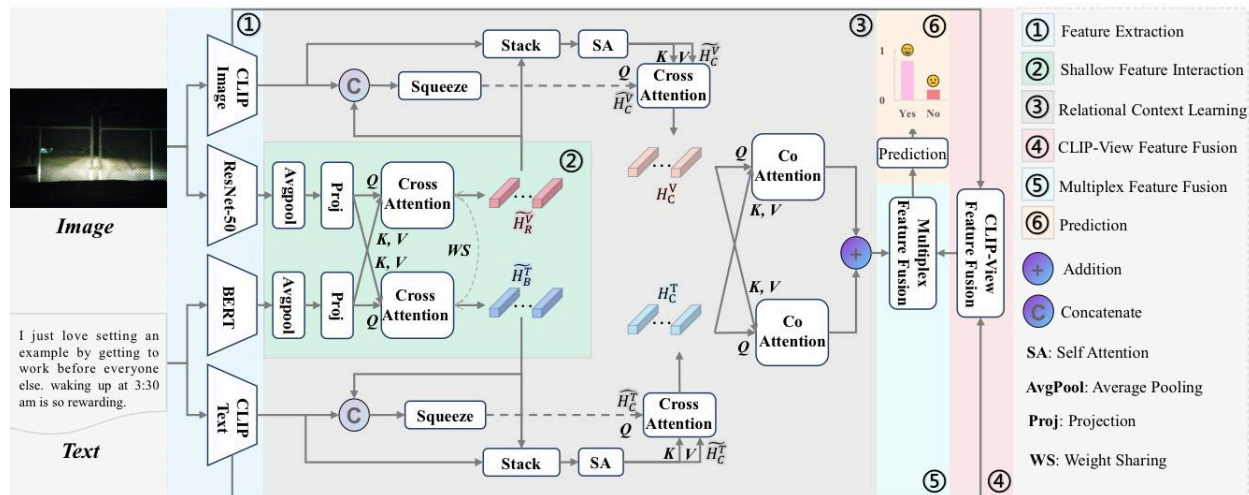
which contains both sarcastic and non-sarcastic image-text pairs with both English and Chinese captions. For fun, we took some of our own photos (pictured in the “Results” section) and gave them sarcastic captions to see whether our model would correctly classify them.

Sample input from the MMSD 2.0 dataset:



text: got a nice cold for the rest of winter #lovebeingill #off
label: 1

Model Architecture



The model contains six main components:

Feature Extraction (1)

We used pretrained models CLIP Image and Text encoder, ResNet-50 and BERT to extract features from the original inputs. CLIP-Image and ResNet-50 are used for image feature extraction, while CLIP-Text and BERT are used for text-feature extraction.

Shallow Feature Interaction Module (2)

This module uses the features from ResNet-50 (images) and BERT (text), runs pooling and scaling operations to standardize them, and performs cross attention between the two sets of features to learn relational context. First, it runs Adaptive Average Pooling on the initial output features from the pretrained models, which helps generalize the model's dependencies on specific features of the input. Next, we map the pooled image and text features to be the same dimensions to use for cross attention. For cross-attention, we use the same transformer layer for two operations: first with the image features as queries and second with the text features as queries. We created a custom attention class named `TransformerCrossLayer` to do this, which contains linear layers, dropout, and normalization. The result of these operations is a set of image features that pays attention to text features and a set of text features that pays attention to image features.

Relational Context Learning Module (3)

The goal of this module is to first learn the context of the image and text features separately, then fuse these features. First, the outputs from step (2) are concatenated with the CLIP feature extraction results from step (1). Next, we perform self attention on the text and image features respectively using `TransformerEncoderLayers`, another custom attention class. This enhances our model's understanding of what the text and image mean separately. Finally, we perform cross-attention on the self-attention outputs (once we ensure that dimensions match up). We use two more `TransformerCrossLayers` to do this, with the post-self-attention image features as queries and the post-self-attention text features as keys and values and vice-versa. Finally, we perform a weighted sum of the outputs of these two cross-attention layers (0.6 for images and 0.4 for text were the weights that the RCLMuFN researchers determined were optimal).

CLIP-View Feature Fusion Module (4)

Since the CLIP pretrained module is so good at feature extraction, a lot of weight is put on its features. This module fuses the raw image and text features from the pretrained modules to ultimately create features that we weight with the output of step (3) to assist in prediction. For the feature-fusion part, we perform cross attention using the exact same cross attention layer from the previous module in a similar way: first with images as queries, second with text as queries. We then take another weighted sum, this time weighting the text features heavier than the image features.

Multiplex Feature Fusion Module (5)

Our final step before predicting the class of our text/image pair is to fuse the two streams of cross-attention that we've created from the previous two modules. We first concatenate the weighted outputs of steps (4) and (3) and pass these features through a small MLP consisting of a linear layer, ReLU, and Sigmoid. We separately pass the same concatenated features through a similar MLP that uses Layer Norm instead of Sigmoid. The outputs of these two MLPs are multiplied, then weighted with HCLIP, the output of step (4) (illustrating the importance of the CLIP feature extractor in our overall model). The RCLMuFN model uses a 50/50 weighting on these two outputs.

Prediction Module (6)

Finally, we pass the output of step (5) into a simple linear layer and apply softmax to get the output probabilities of each class (sarcastic or not).

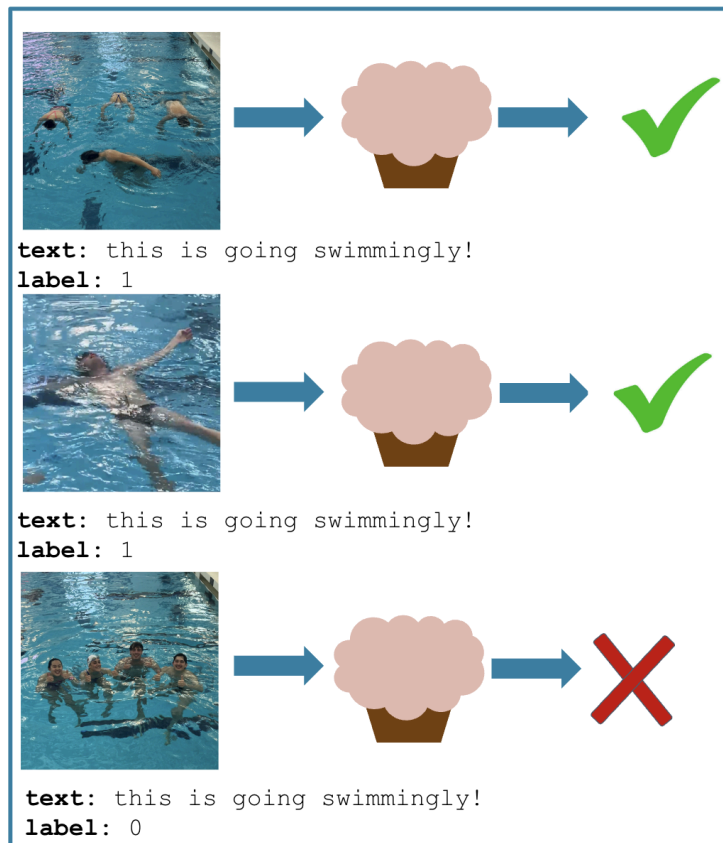
Results

The chart below depicts an abridged version of the results table obtained from testing different hyperparameters of our three main datasets: MMSD 2.0, our custom Muse/Flickr combined dataset, and SarcNet. Each line depicts the highest-accuracy result obtained on each dataset, along with the relevant model hyperparameters used to achieve such a result, and the accuracy, precision, recall, and overall F1 scores respectively. For the entire table of results, see Appendix 1. For additional features that were not reflected in the experimental results, such as the positional encoding of features, see Appendix 2.

Dataset tested	Relevant model hyperparameters	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
MMSD 2.0 <i>(Left figure)</i>	10% dropout before last MLP layer; BERT/Resnet/CLIP frozen; shuffling over entire training dataset; 2 epochs	79.99	72.43	86.40	78.80
Muse/Flickr Custom Dataset <i>(Middle figure*)</i>	BERT/Resnet/CLIP frozen; custom frozen batch 2d norm ; shuffling over entire training dataset; 2 epochs	72.42	67.83	85.08	75.48
SarcNet <i>(Right figure)</i>	10% dropout before last MLP layer; BERT/Resnet/CLIP frozen; shuffling over entire training dataset; 2 epochs	60.81	62.16	83.20	71.16

The following graphic depicts a small sample of results obtained based on image and text pairs that we manually photographed, captioned, and labeled as either sarcastic or not sarcastic. Passing our “dataset” into our model (the muffin, after “RCLMuFN” or “rackle-muffin”) yielded correct predictions of sarcastic on the image-text pairings which were intended to be sarcastic (one depicted all four swimmers apparently drowning in a pool, while the other depicted a single group member floating face-up, seemingly unconscious). However, for the image-text pairing we labeled as non-sarcastic (an image of all four group members giving a thumbs-up in the pool), the output of the model was still that this multimodal datum was predicted to be sarcastic. For a number of variations of caption, predictions of sarcastic were always returned, reflecting a prior issue encountered with the larger datasets — that for datasets on which our model had not

trained, we observe high rates of recall and a large proportion of false predictions being false positives (i.e. the model predicts the datum is sarcastic when in fact it is not).



Challenges

One problem we ran into was not being able to unfreeze the CLIP and BERT parameters of our model. When running on Oscar, we consistently ran into out-of-memory errors, and our requests for more than 3 GPUs never made it out of the queue. Another major issue we ran into with our model was that, for a while, it trained to classify image-text pairs only as sarcastic. We realized that this was due to the training data being shuffled within each batch (of 32 data points) only, instead of shuffling over the entire dataset. The first few batches of data only consisted of sarcastic inputs, causing the model to output sarcastic predictions only. After fixing this by shuffling over the entire dataset, we were able to achieve higher accuracy with our model and get predicted outputs of both 0's and 1's; however, our model is still slightly skewed towards predicting things as sarcastic, especially when we apply it to non MMSD datasets in different formats. This makes sense, as MMSD consists largely of Twitter data, in which sarcasm is likely very common.

Reflection

Overall, we are content with the final results of our project. Though we were unable to achieve as high of an accuracy as the paper did, we think we did reasonably well given our computational limitations. Additionally, sarcasm detection is a hard task, sometimes even for humans.

Our base, target, and stretch goals were as follows:

- **Base:** Model successfully labels test split with either sarcastic/not sarcastic. Model is implemented with Keras MultiHeadedAttention (no multiscale deformable attention). Goal accuracy of 60%
- **Target:** Accuracy of 78% on MMSD 2.0
- **Stretch:** Model is implemented with multiscale deformable attention

Overall, we were able to meet our target goal. Our model successfully labeled the test split, using multi-headed attention, with an accuracy of over 78% based on proper choices of hyperparameters (79.99%). Additional goals which we made after check-in 2 included testing on other multimodal datasets. We were able to do this as well, testing on our custom Muse/Flickr dataset and the SarcNet dataset in addition to the test split of MMSD 2.0. This indicates that we were able to achieve our target goals, and even slightly exceed them given our ability to test on not only 1 but 2 additional datasets.

Ultimately, certain aspects of the model turned out to work out very differently than expected. The more we delved into the codebase and implementation of our chosen paper, the more incongruencies we seemed to find: several classes they wrote were not actually used in their model (despite being cited in their paper, such as multiscale deformable attention). Additionally, the model's performance seemed to be overly reliant on inputs to the model such as CLIP based on their ablation study, the multiplex fusion network essentially just consisted of an MLP, and so on. Then, while testing our model, we found unexpected results relating to the choice of whether or not to integrate some of the self-designed model components in [backbone.py](#) into the overall model: integrating the custom frozen batch 2D normalization implemented in [backbone.py](#) had a negative effect on the accuracy for most, but not all cases. The notable exception to this trend was the best result for testing on the Muse/Flickr custom dataset, as can be observed in Appendix 1.

How did your approach change over time? What kind of pivots did you make, if any? Would you have done differently if you could do your project over again?

Overall, the implementation of the project followed a fairly intuitive trajectory. Initially, our model's accuracy was extremely low (around 43%, which is abysmal considering random guessing would be 50%). Thus, our initial approach focused largely on targeting the issues with the model and figuring out how to improve it. We fixed our issues with data shuffling, dropout, and training vs. testing configuration, which led to a far better-performing model. After this, we focused more on making fine-grained adjustments.

Some of the pivots we made included adjusting our expectations for training on Oscar given GPU limitations and adjusting model hyperparameters accordingly, changing what other datasets we used for testing and in what ways we tested on other datasets, and changing our priorities for implementation to-do's upon approaching and achieving our target progress checkpoint. Specifically regarding the use of other datasets for testing, we originally tested on Muse (all sarcastic captions) to find the essentially trivial result that our model would successfully classify (virtually) every datapoint as sarcastic. To remedy this, the solution was to integrate the all-sarcastic data with a non-sarcastic multimodal dataset, which we found in the Flickr 8k dataset. Other pivots can be found in the Challenges section of the report. Should we do the project over again, incorporating the testing of other datasets might be a slightly earlier priority than it was, and finding solutions for the lack of GPU space on Oscar might be achieved, through either fixing some hypothetical (currently unbeknownst to us) memory leakage or through having more compute allocated to us.

What do you think you can further improve on if you had more time?

The most available examples of further improvement include the implementation of multiscale deformable attention and integration with the rest of the model. Besides this, a more long-term improvement would be to train on multilingual corpuses, which, although potentially doable in the short-term with SarcNet, should ideally be accomplished across a wider range of languages than English and Chinese. Additionally, while the model in the paper didn't end up using positional encoding, it makes sense that this should have a positive effect on the model's accuracy, so we would ideally incorporate it.

What are your biggest takeaways from this project/what did you learn?

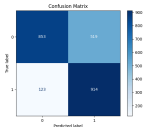
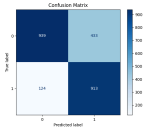
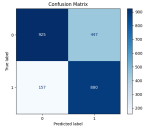
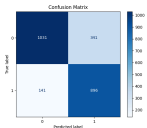
Our general takeaways from this project included a much better insight into the current state of multimodal sarcasm detection as a deep-learning problem. This theoretical understanding was also accompanied by procedural understanding of the workflow of collaborative code implementation in the context of this multimodal sarcasm detection problem, particularly through utilizing version control software in collaborative coding such as Git and Github and gaining an appreciation for the interconnectedness of the different aspects of a deep learning model, from data processing to model implementation to training and testing. More facetiously, the high rate of sarcasm detection in our final model might indicate to us another lesson: that we're more sarcastic than we think!

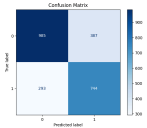
Appendix 1: Full experimental results table

Models we tried training but they didn't work:

- RESNET/CLIP frozen; **BERT unfrozen**; shuffling over entire training dataset; 2 epochs – ran out of GPU memory at 424/620 batches
- RESNET/BERT frozen; **CLIP unfrozen**; shuffling over entire training dataset; 2 epochs – ran out of GPU memory at 603/620 batches :(

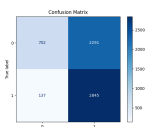
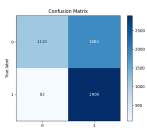
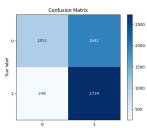
Testing Dataset: MMSD 2.0

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	Confusion Matrix
BERT/Resnet /CLIP frozen; shuffling data within each batch (of 32); 2 epochs	44	44	100 * model was classifying everything as sarcastic	61	—
BERT/Resnet /CLIP frozen; shuffling over entire training dataset; 5 epochs	73.35	63.78	88.14	74.01	
BERT/Resnet /CLIP frozen; shuffling over entire training dataset; 2 epochs	76.84	67.83	88.04	76.63	
BERT//CLIP frozen; Resnet unfrozen ; shuffling over entire training dataset; 2 epochs	74.93	66.31	84.86	74.45	
Add dropout (10%) before last MLP layer; BERT/Resnet /CLIP frozen; shuffling over entire training	79.99	72.43	86.40	78.80	

dataset; 2 epochs					
BERT/Resnet /CLIP frozen; replace normalization in Resnet with custom frozen batch 2d norm ; shuffling over entire training dataset; 2 epochs	71.77	65.78	71.74	68.63	

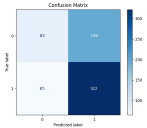
Testing Dataset: Muse/Flickr Custom Dataset

Dataset formation process: We took the training data split from Muse (2983 samples) and 2993 random samples from Flickr, and shuffled them together to create a custom dataset to test on.

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	Confusion Matrix
BERT/Resnet /CLIP frozen; shuffling over entire training dataset; 5 epochs	59.36	55.39	95.41	70.09	
BERT/Resnet /CLIP frozen; shuffling over entire training dataset; 2 epochs	67.11	60.63	97.25	74.69	
Add dropout (10%) before last MLP layer ; BERT/Resnet /CLIP frozen; shuffling over	63.35	58.47	91.68	71.40	

entire training dataset; 2 epochs					
BERT/Resnet /CLIP frozen; replace normalization in Resnet with custom frozen batch 2d norm ; shuffling over entire training dataset; 2 epochs	72.42	67.83	85.08	75.48	—

Test Dataset: Sarcnet

Model	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)	Confusion Matrix
Add dropout (10%) before last MLP layer; BERT/Resnet /CLIP frozen; shuffling over entire training dataset; 2 epochs	60.81	62.16	83.20	71.16	
BERT/Resnet /CLIP frozen; replace normalization in Resnet with custom frozen batch 2d norm ; shuffling over entire training	59.31	62.29	75.97	68.45	-

dataset; 2 epochs					
-------------------	--	--	--	--	--

Presentation-ready Experimental Results document

Dataset tested	Relevant model hyperparameters	Accuracy (%)	Precision (%)	Recall (%)	F1 Score (%)
MMSD 2.0	10% dropout before last MLP layer; BERT/Resnet/CLIP frozen; shuffling over entire training dataset; 2 epochs	79.99	72.43	86.40	78.80
Muse/Flickr Custom Dataset	BERT/Resnet/CLIP frozen; custom frozen batch 2d norm ; shuffling over entire training dataset; 2 epochs	72.42	67.83	85.08	75.48
SarcNet	10% dropout before last MLP layer; BERT/Resnet/CLIP frozen; shuffling over entire training dataset; 2 epochs	60.81	62.16	83.20	71.16

Appendix 2: Additional Features

- Position_Encoding: A standard positional encoding class ran in conjunction with backbone, its output was never used in the model. We re-implemented the method, and also never ended up using it.
- FrozenBatchNorm2d: A custom frozen batch normalization class was also implemented in the paper. We did so as well, and found (see above) that using it instead of Resnet normalization only improved our performance on the Muse/Flickr customized dataset. This was probably due to the dataset's small batch size, as frozen batch normalization helps avoid issues stemming from excessive random noise in small batches.
- Multi-scale Deformable Attention: This method was implemented in the paper, it combines information from multiple feature map scales by sampling at learned offsets and weighting the results. This helps the model focus on the most useful areas in the image. We found that the paper didn't actually use the method in their model, so we also decided not to use it.

Appendix 3: Previous Checkin Documents

[Checkin 2](#)

[Checkin 3](#)