Final Report: Multimodal Classification To Detect Hate Speech

Kartikay Kaushik
Dept. of ECE
NYU Tandon
kk4332@nyu.edu

Navya S. Jammalamadaka
Dept. of ECE
NYU Tandon
nsj9072@nyu.edu

Aadrit Ghimire
Dept. of ECE
NYU Tandon
ag8520@nyu.edu

Abstract

As part of the project, we have designed, developed and trained a multimodal classification model on the Hateful Memes dataset. The challenge was initially organized by Facebook AI to detect hate speech in multimodal memes. The winner of the challenge had an accuracy of 84.50%. The state-of-the-art methods to perform the classification are quite imprecise. The original objective of the project was to maximize the classification accuracy by modifying/fine-tuning the base model provided as part of the Hateful Memes Challenge. But, following the suggestions in the project review, in addition to the above, we created our own multimodal models with different text and image classifiers, and extended our dataset to improve the accuracy of classification of memes into hateful or non-hateful. The model is then extended as a service where users can upload a meme and the application would inform the user if the meme is hateful or not.

1 Introduction

With the proliferation of hate speech on internet, it is important to tackle and mitigate the societal problems associated with it. In many countries, three quarters or more of the victims of online hate speech are members of minority groups[1][2][3]. Hate speech covers advocate, incite, promote or justify hatred, violence and discrimination against a person or group of persons.[15] The prevalence of hate speech in online social media is a great societal responsibility for many social media companies.[21] The scale of social media makes it difficult to curb hate speech by human efforts alone. Therefore, it is important for AI and Machine Learning to address this challenge.

Inspired by this scenario, we have created a multimodal model that challenges (to some extent) the state-of-the-art methods to perform the classification of text-image data into hateful or non-hateful. We use memes as inputs to train our model to understand the co-relation between images and texts and the effect of their combination.[8][9] The model involves a mid-level concat fusion type of architecture, where the input data would be split into text and image. The image data would pass through an image model, the text data would pass through a text/language model. Then, these feature representations are concatenated to create a new feature vector. Finally, this feature vector is sent through a fully connected layer for classification.[4] We have also experimented with concatenating existing dataset with memotion datatset to improve the accuracy. We have used PyTorch Lightning to train the model. The models were tested on Jupyter Notebook on NYU HPC and Google Colab. We were able to achieve a maximum accuracy of 63.45%[10-14]

2 Methodology

We started the project by understanding the problem statement, and the available resources. After comprehending our dataset (Section 2.1) we then, resolved the installation issues (Section 2.2), and created a custom multimodal model (Section 2.4). Then, we proceeded to test the pre-trained models with tweaked parameters. Then, we proceeded to develop our own multimodal model with different text and image classifiers. Finally, we experimented by concatenating Memotion Dataset 7k with Hateful Memes Dataset and finetuning pretrained models with the new dataset (Section 2.5).

2.1 Understanding the hateful memes dataset

We intend to use the Hateful Memes dataset provided by Facebook AI[5]. The dataset contains 10,000 memes with following categories [7] -

- Multimodal hate - 40% (Benign confounders were found for both text and image)
- Unimodal hate - 10% (Either text or image or both were already hateful on their own)
- Benign image - 20% (Image is non-hateful)
- Benign text - 20% (Text is non-hateful)
- Random not-hateful - 10%

A benign confounder is a minimum replacement image or replacement text that flips the label for a given multimodal meme from hateful to non-hateful.[7]

Dev data = 5%, Test data = 10% and Fine-tuning = 85%

Output classes = 2 (0 - non-hateful, 1 - hateful)

2.2 Challenges with setup

We proceeded to install the required prerequisites with the necessary configuration [16]. But, we observed that the installation kept failing due to several factors -

Problem	Solution
Change in the dataset location. The dataset can no longer be accessed via a link directly and has to be downloaded after filling a form.	Downloaded the dataset after filling form, uploaded on Google Drive (File was too large for GitHub) and used gdown to extract it in the current directory
Default paths were invalid for current dataset in the mmf (Multi Modular Framework) configuration	Corrected the paths in code and uploaded the zip file on GitHub. The framework installation now occurs from personal GitHub repo and not Facebook AI's GitHub repo
Missing pandas-path and gdown were not installed as prerequisites	Added the new requirements in requirements.txt of mmf-client
Version compatibility between torch, torchvision and mmf. mmf could not be installed with required versions of torch and torchvision	First installed mmf and then installed the required versions of torch and torchvision rather than including the required versions as part of requirements.txt
Different default paths for files picked in case of pre-trained and self-trained models setup due to permission issues on NYU HPC for /root folder.	Created different mmf zip files with different default paths. The pre-trained models run on Google Colab and self-trained models can run on both HPC and Google Colab

2.3 Running a pretrained model

In order to verify the setup of our base code and understand the impact of multimodal models, we first ran pre-trained models. The pretrained models also helped us in identifying the models which

provided better accuracy. We intend to study the models with higher accuracies to pick components for our customized models.

1. Downloaded the dataset zip file (hateful_memes.zip), extracted it and was converted into mmf (Multi Modular Framework) format.
2. Built the dataset using mmf libraries
3. Imported the pretrained model from MMBT using the model key and baseline config
4. Test the model with a custom image URL and text.

Table 1: Accuracies of pre-trained multimodal models

Model	Accuracy
MMBT-Region	59.45%
MMBT-Grid	64.32%
ViLBERT	62.14%
Visual BERT	64.86%

We also ran our dataset on unimodals - Image-Grid for Image classification and Text Bert for textual modality (Both models were pre-trained). We observed that Text Bert performed better than Image-Grid and the accuracy was closer to that of multimodal models. This can be used to infer that text component of the memes have a higher impact on classification of than the image component.

Table 2: Accuracies of pre-trained unimodal models

Model	Accuracy
Image-Grid	50.34%
Text Bert	57.56%

2.4 Creating a custom model

Following the comparative study conducted on the pre-trained models, we created a custom model using the sample base architecture provided by Facebook AI [4] on the Hateful Memes dataset. The code was designed to implement this architecture for different image and text classifiers. As opposed to our earlier assumption, the code could not be directly adopted from [4] due to differences in versions of pytorch-lightning. Due to break in backward compatibility, the parameters and functions were different and had to be modified to ensure that the code is compatible.

2.4.1 Pre-processing the data - Class HatefulMemesDataset

Jsonl data files were used to contain the multimodal data (image files and the corresponding textual captions) as dictionaries for training, testing, and validation datasets.

The HatefulMemesDataset class is a subclass derived from the *torch.utils.data.Dataset* available in the pytorch library, that is used to create and load a custom dataset for machine learning models. The constructor for this class (__init__) loads the dataset from the jsonl file into a pandas dataframe.The __getitem__ method takes an index as an argument and locates the indexed and converts it to a Tensor. The data transforms for both the text and image inputs are conducted here, and the corresponding tensors are returned as a dictionary. This method is especially useful in the creation of Data-Loaders. The transformed dataset was then loaded into a Dataloader with batch-size 4 for the purpose of training and testing.

We observed that the sizes of the images were different. This issue was resolved by resizing the images using torchvision.transforms.Compose function. The images in the dataset were resized to dimensions of 3 X 224 X 224 (3 X 299 X 299 for inceptionV3) and normalized to the mean and standard deviation of the ImageNet dataset. Also, since our architecture uses text and image classifiers, we must split our input data accordingly into text and image sections. Additionally, we observed that the training data was imbalanced - Output 0 (Non-hateful) = 5481 and Output

1 (Hateful) = 3019. This could lead to an issue in training due to a compromised prior that the input would more likely be non-hateful. We used class-balancing to prevent this issue. We used PyTorch's Dataset class to resolve the cases.

Our input was thus converted into a dictionary with the following keys - "id" (image id), "image" (tensor of the image), "text" (tensor of the text), "label" (true value).

2.4.2 Architecture - Class LanguageAndVisionConcat

1. We pass our pre-processed image data through a pretrained model selected from the torchvision library (DenseNet, ResNet, InceptionV3, etc). The output of the same was finetuned to fit our dataset by tweaking the parameters of the final layer of the network.
2. Similarly, word embedding using the various text embedding models (fasttext, distillBERT, BERT) were carried out on the text captions in order to map the data to an output feature set vector of predetermined dimensions corresponding to each individual data. Word embedding gives the learned visualization that distributes the output attributes with respect to the similarity in meanings of the input textual data.
3. The output of this text transform is then passed through an additional trainable layer that serves the purpose of fine tuning.
4. The feature set of both the transformed image and text inputs were individually passed through ReLU non linearities.
5. In order to get a feature vector of predetermined dimensionality as output from the vision model, a Fully connected Linear layer was implemented on the output of the network.
6. The outputs if the two non-linearities were then concatenated into a single vector and processed through a linear layer giving us an intermediate vector of predefined dimensions.
7. This vector was then passed through a ReLU non-linearity
8. Dropouts was applied to improve the learning.
9. This was followed by a final linear layer that took as input the intermediate vector
10. A softmax activation was used to obtain the classification metric in terms of probability and output the binary classification labels of the data as either hateful or not hateful.

We had tried the architecture with Word2Vec model for text model too but inorder to use sentence vectors, we would need to either take the averages of words or use Doc2Vec.[17]

2.4.3 Vision Models used

The following vision models were used from the Torchvision library and feature extracted upon to fit our classification task. Pretrained models were initialized and only the final layer of the models were modified to extract the number of required features and update the weights in the training step. The following models have been pretrained on the ImageNet dataset, thus have 1000 output classes, which was reduced to 300 classes for our task. Note that the torchvision models require an input image of dimensions (224 X 224), except the GoogleNet inception_v3 (299 X 299).

1. ResNet152
2. DenseNet201
3. GoogLeNet InceptionV3

2.4.4 Text Models Used

We used the following text models -

1. FastText
2. BERT

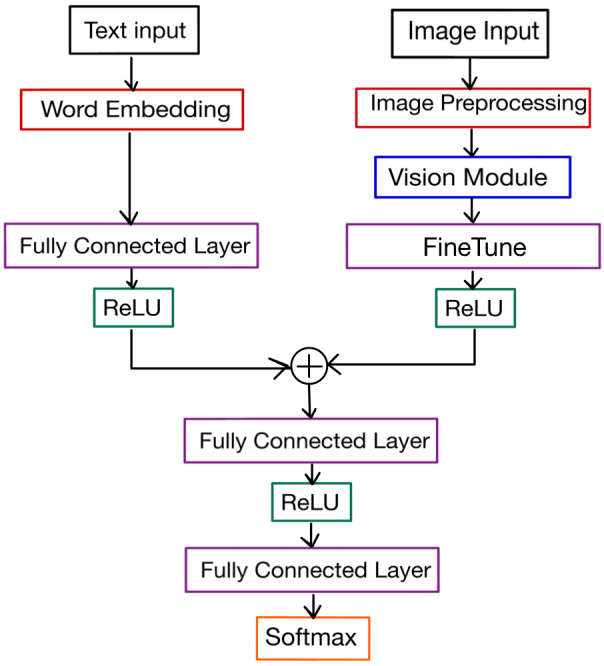

Figure 1: Model Architecture

BERT:

Neural Network models in Deep Learning expect inputs in the form of real-vectors. In order to leverage these models for NLP applications, word embeddings provide the means to convert textual data into real-valued vectors

BERT language encoding model was taken from the transformers library in HuggingFace.The following steps were taken to conduct the text embedding.

1. Pre-trained DistilBERT model was loaded.
2. The input text data was tokenized in order to assign each word in a unique token index
3. To feed into BERT, we need to have token sequences of consistent length.
4. Padding was done to ensure the same, and a mask was created to ignore the padded values of 0.
5. The tokens were then passed through the pretrained BERT model to generate the feature embeddings.
6. Being a classification task, we ignore the tokens associated with the individual words and take only the CLS token, that represents the entire sentence and maps it contextually.
7. This feature was then passed into a trainable linear fully connected layer that served as the language model.

2.4.5 Training - Class HatefulMemesModel

In order to train our model, we used PyTorch lightning. As an input, we have passed a Python dictionary consisting of hyperparameters required to provide custom configuration in the training mechanism.

We override the methods of PyTorch lightning like training_step, training_epoch_end, validation_step, validation_epoch_end, test_step, test_epoch_end, etc according to the format provided in

PyTorch lightning documentation.[18] At every epoch end we ensure to have the progress bar print the training and validation loss.

We configured the optimizers to have the following properties:

- Learning rate = 0.00005
- Optimizer = AdamW

We also configured paths to our training, validation and testing data. The models were fine-tuned to provide the output dimensions of 300[19]

After the epochs were ran, we created a submission frame. The output is generated as a .csv file that contains the following columns with respect to the test data set[6] -

- id - Meme identification number,
- proba - Probability that the meme is hateful,
- label - Binary label that the meme is hateful (1) or non-hateful (0)

Then, we compared the entries in the .csv file with their corresponding entries in the test data jsonl file to calculate the final accuracy.

2.5 Extending the training data with Memotion Dataset 7k

Memotion Dataset 7k is a dataset of 6992 images originally used for sentiment classification of memes (Weirdly focused a lot on Leonardo DiCaprio).[20]

First, we concatenated Memotion Dataset with Hateful Memes Dataset. Then, we fine-tuned previously made models on the concatenated data. We observed that there was no significant improvement in the model classification accuracies. Upon closing inspecting the dataset, we discovered that the dataset is erroneously labeled with respect to classification of memes as hateful and non-hateful.

Hence, we decided to label the dataset ourselves. Due to time constraints, we cherry-picked 328 memes from the Memotion Dataset 7k that would be suitable for the project. We used scripts for labeling the Memotion Dataset and converted them to the same data format as Hateful Memes Dataset.

Then, we extracted features using the modified extract_features_vmb.py script (based on the base mmf package by Facebook AI). Finally, we used the extracted features to fine-tune a pre-trained VisualBERT model as done in Section 2.3.

3 Creating the service

In order to make our application accessible to users, we created a service with AWS as our infrastructural platform. Currently, it is a rudimentary website hosted on an S3 bucket and can be accessed at this LINK.

Input - Image URL and Meme text
Output - Image is hateful or non-hateful and the confidence percentage of the decision.

- First, we created API end points.
- Then, the API end points are connected to User Interface, using the apigclient.
- apigclient sends request to API end point when user interacts.
- This request is then forwarded to the lambda function via REST API call (POST call).
- The request is then forwarded to AWS SageMaker from AWS Lambda function. AWS SageMaker is responsible for creating, training, and deploying machine-learning models on the cloud.
- AWS SageMaker runs our model and gets the output corresponding to our provided input (text and image URL).

- An output is generated corresponding to our inputs. The output contains information if the meme is hateful or non-hateful and the confidence percentage corresponding to the decision.
- This output is then again sent back to Lambda Function via API calls.
- This output is then again forwarded to S3 bucket containing the website hosted. The S3 bucket receives this output and the corresponding output is displayed to the user.

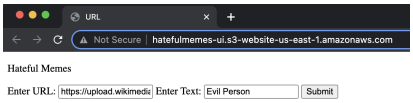

Figure 2: Website Image

4 Results and observations

The code and the corresponding steps to run it is provided on the following GitHub Link. The pretrained models can be run only on Google Colab whereas the self-trained models can be run on NYU HPC or Google Colab.

This section contains the results of the above experimentation. The detailed information about the experiments can be found on this LINK. The Google Sheet contains the tracker maintained to note the results for changes in text models, image models and changes in hyperparameters.

We observed the following accuracies with different combinations of text and language models-

Table 3: Model Accuracies of self trained multimodal models

Language Model	Image Model	Dataset	Accuracy
FastText	DenseNet201	Unbalanced	51.80%
FastText	GoogLeNet InceptionV3	Unbalanced	54.20%
FastText	ResNet152	Unbalanced	55.20%
BERT	ResNet152	Balanced	57.32%
BERT	InceptionV3	Balanced	63.45%

5 Conclusion

Having familiarized with multimodal models by implementing the pre-trained models MMBT-Region, MMBT-Grid, ViLBERT, Visual BERT and fine-tuning the base model, we proceeded to create initial self-trained model with FastText and BERT as text model and ResNet152, DenseNet and InceptionV3 as image models and we observed the highest accuracy in case of BERT with inception_v3, with a balanced dataset with equal number of positive and negative labels. Then, we

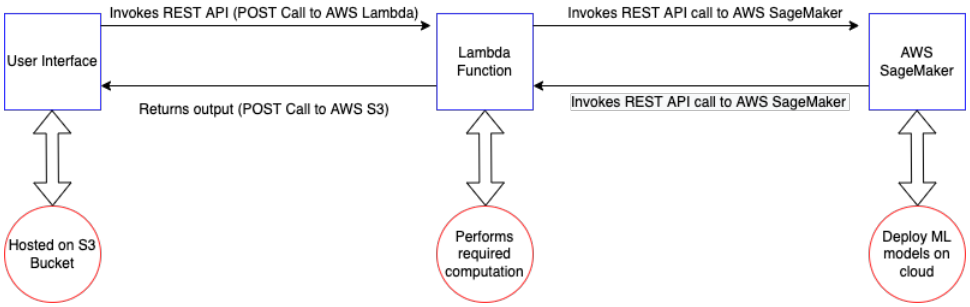

Figure 3: Service Architecture

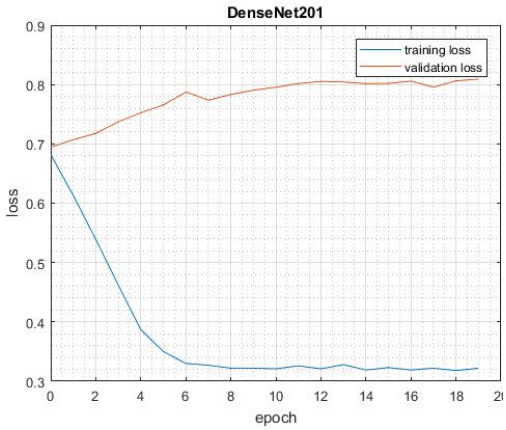

Figure 4: Accuracy observed = 51.80%

Figure 5: Accuracy observed = 54.20%

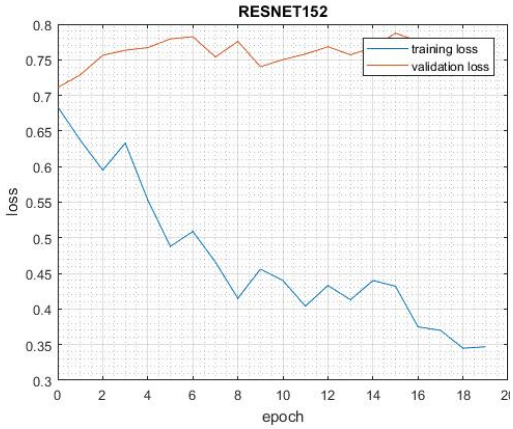

Figure 6: Accuracy observed = 51.80%

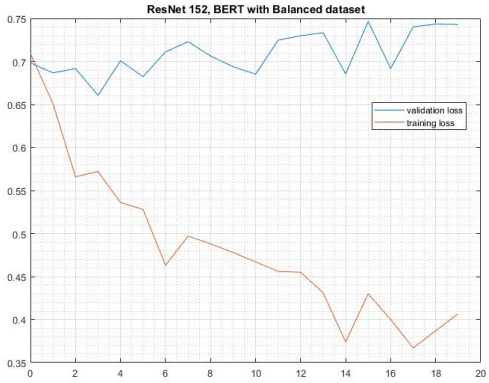

Figure 7: Accuracy observed = 57.32%

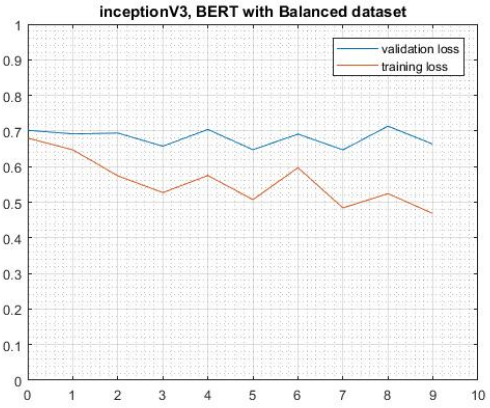

Figure 8: Accuracy observed = 63.45%

experimented with different text encoders as part of our multimodal model. We also provide our deliverable as a service where users can provide an image URL and text and the user would be informed if the meme is hateful or not.

6 References

[1] www.ohchr.org

[2] www.cfr.org

[3] www.gicj.org

[4] hateful-memes-benchmark

[5] hatefulmemeschallenge.com

[6] mmf.sh

[7] Kiela, Douwe & Firooz, Hamed & Mohan, Aravind & Goswami, Vedanuj & Singh, Amanpreet & Ringshia, Pratik & Testuggine, Davide. (2020). The Hateful Memes Challenge: Detecting Hate Speech in Multimodal Memes.

[8] M. Soleymani, D. Garcia, B. Jou, B. Schuller, S. F. Chang, and M. Pantic. A survey of multimodal sentiment analysis. Image and Vision Computing, 65:3–14, 2017.

[9] Fan Yang, Xiaochang Peng, Gargi Ghosh, Reshef Shilon, Hao Ma, Eider Moore, and Goran Predovic. Exploring deep multimodal fusion of text and photo for hate speech classification. In Proceedings of the Third Workshop on Abusive Language Online, pages 11–18, 2019.

[10] Wei, Bencheng & Li, Jason & Umair, Hafiza & Vovor, Atsu & Durzynski, Natalie. (2021). Offensive Language and Hate Speech Detection with Deep Learning and Transfer Learning.

[11] Das, Abhishek & Wahi, Japsimar & Li, Siyao. (2020). Detecting Hate Speech in Multi-modal Memes.

[12] Gomez, Raul & Gibert, Jaume & Gomez, Lluis & Karatzas, Dimosthenis. (2020). Exploring Hate Speech Detection in Multimodal Publications. 1459-1467. 10.1109/WACV45572.2020.9093414.

[13] I. M. Ahmad Niam, B. Irawan, C. Setianingsih and B. P. Putra, "Hate Speech Detection Using Latent Semantic Analysis (LSA) Method Based On Image," 2018 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), 2018, pp. 166-171, doi: 10.1109/ICCEREC.2018.8712111.

[14] Sabat, Benet & Ferrer, Cristian & Giró-i-Nieto, Xavier. (2019). Hate Speech in Pixels: Detection of Offensive Memes towards Automatic Moderation.

[15] ww.coe.int

[16] Prerequisites Installation

[17] www.stackoverflow.com

[18] PyTorch Lightning Documentation

[19] www.pytorch.org

[20] Memotion Dataset 7k

[21] Sharma, Chhavi & Bhageria, Deepesh & Scott, William & PYKL, Srinivas & Das, Amitava & Chakraborty, Tanmoy & Pulabaigari, Viswanath & Gambäck, Björn. (2020). SemEval-2020 Task 8: Memotion Analysis – The Visuo-Lingual Metaphor!.

BERT:

Neural Network models in Deep Learning expect inputs in the form of real-vectors. In order to leverage these models for NLP applications, word embeddings provide the means to convert textual data into real-valued vectors.

BERT language encoding model was taken from the transformers library in HuggingFace.The following steps were taken to conduct the text embedding. Pre-trained DistilBERT model was loaded. The input text data was tokenized in order to assign each word in a unique token index To feed into BERT, we need to have token sequences of consistent length. Padding was done to ensure the same, and a mask was created to ignore the padded values of 0. The tokens were then passed through the pretrained BERT model to generate the feature embeddings. Being a classification task, we ignore the tokens associated with the individual words and take only the CLS token, that represents the entire sentence and maps it contextually. This feature was then passed into a trainable linear fully connected layer that served as the language model.