COE 379L

Fine-tuning CLIP Model for Multimodal Meme Analysis

Jason Kim jk46965

Braulio Lopez bl27466

Introduction and Project Statement

Social media has become a big platform for everyone. Memes are commonly posted and often convey nuanced messages through a combination of images and texts. These messages may include offensive content that is highly inappropriate. Thus, understanding the sentiment and content of these memes is essential for allowing platforms to monitor posts and flag them accordingly.

To address this need, we can train and utilize ML models to effectively identify these offensive memes and prevent negativity across social media platforms. Thus, for this project, we aim to fine-tune a pretrained CLIP model by training it with a multimodal dataset that contains meme images and their associated text - this will be a binary classification model, identifying whether a meme and its associated text are either ‘offensive’ or ‘non-offensive’.

We chose to work with the CLIP model, developed by OpenAI, due to its robust features, zero-shot capabilities, and efficiency/practicality. Furthermore, since our goal was to work with a multimodal dataset/model, the CLIP model proved to be well-suited for our specific project, as it is inherently multimodal.

Data Sources and Technologies Used

Images

Some sources that we use to work with this project is an extensive directory of images containing non-offensive and offensive political memes; these are in the form of png or jpg, depending on the image. It is important to note that the images are in different sizes and resizing will play a big part in the pre-processing of our data. To address this issue, we implemented the following libraries:

- OS: this is used to interact with the operating system
- Image: Imported from PIL this class is useful in accessing and manipulating image files
- resize : Imported from torchvision.transforms.functional
- transforms: This torchvision module includes typical picture transformations, which are utilized here to create a transformation sequence.

After correctly importing all the necessary libraries, we designed a function that resizes the images, thereby obtaining a 224x224 picture that ensures uniformity across all neural network inputs. It is important to note that when resizing, we could lose important information as some of the images could have contextual information about their original size.

CSV Files

Another essential data source for the project is "Testing_meme_dataset.csv," "Training_meme_dataset.csv," and "Validation_meme_dataset.csv." These are csv files containing labels and appropriate splits, so we don't have to manually split the dataset. The csv files contain 3 columns, in which the first one, "image_name," identifies the image name from the images directory, "sentence," which includes associated text from the images, and lastly, "label," which identifies if the image is offensive or non-offensive. It is important to note that "sentence" contains noise, this noise includes punctuation marks, or special characters that need to be tokenized, which will make the text more uniform for analysis. To address this issue, we implemented the following:

- **Loading data**: We started by importing Pandas to load the training, testing, and validation csv files located in their specific directory. By using this library, we successfully load them into a Python data frame that we can inspect for cleaning,filtering,g and transformation.
- **Missing values check:** After successfully loading the dataframes, we needed to ensure the integrity of the data by checking for missing values. After a check, we were able to determine that there were no missing values in all the data frames, indicating a clean and prepared set for the next steps.
- **Label Distribution Check:** Checking the distribution of offensive and non-offensive was a crucial part of the project, as we were able to identify that in the datasets there are more non-offensive labels, which is better because we wouldn't want our trained model to mispredict an actual non-offensive image as offensive.
- **Text Preprocessing:** To prepare the textual content, we implemented a preprocessing function using NLTK. This function helps us convert text to lowercase, tokenize the text, lemmatize each token, and delete stopwords and punctuation. This process was important for reducing the complexity of the text and enhancing meaningful words, which aid in providing accuracy and having a more uniform analysis. It is important to note that while we tokenize the text, we may lose contextual information, as punctuation and words carry important information when written properly.

Flask

Some technologies used for the project include a Flask based API to provide a machine learning model capable of evaluating memes using both textual and visual input.

- **Prediction Endpoint (/predict):** we implemented a /predict endpoint that accepts POST requests. This endpoint expects an image and a text from the request. The CLIP processor receives the preprocessed data and prepares the model's inputs. The model does inference

without updating weights (torch.no_grad()), and the output probabilities are utilized to make the final prediction (offensive or not).
- **Server Execution:** The Flask server is configured to operate at host='0.0.0.0' and port=5000, making it available via the local network.

Docker

In our project, Docker plays a big role in that our flask based meme analysis application is deployable and runs in different environments; hence, we introduce a Dockerfile and a docker-compose.yml.

- Dockerfile: This gives an introduction to Docker on how to set up and configure the environment inside the container.
- Docker-compose.yml: It lets us specify how our application's containers are constructed, operated, and interconnected.

Methods Employed

Fine-tuning CLIP

Since CLIP is inherently designed for broad multi-class applications, it presented a challenge when applied to our concentrated binary classification task—determining whether a meme is 'offensive' or 'non-offensive'.

After a seemingly “successful” training process for our model, we then needed to test it against our test dataset. When trying to test, however, we encountered an error, saying that the model’s predictions spanned up to 15 different classes as per its multi-class training background, when instead we were striving to predict binary classification. This indicated a misalignment between CLIP’s capabilities and our project requirements, so we had to fine-tune CLIP’s powerful capabilities within the constraints of binary classification.

To address this, we developed two key components in the form of Python classes:

1. **BinaryClassifier:** A neural network layer designed to parse CLIP's rich feature representations and return a single probability score. This score represents the likelihood that the meme is 'offensive', reducing the multidimensional output to a binary format.

2. **CLIPBinaryModel:** An integration framework that blends the basic CLIP model and our newly developed BinaryClassifier. This adaptation ensures that, while the model continues to use CLIP's advanced, pre-trained embeddings, the final output is customized to our binary classification requirements.

Moving forward, we fine-tuned by integrating the above two components:

- **Adjusting Model Architecture:** By adding the BinaryClassifier to CLIP, we were able to restructure the model's output strategy to correspond with our binary classification objectives.

- **Retraining with Specific Data:** We fine-tuned our integrated model using our dataset of memes and accompanying text, adjusting the 'BinaryClassifier' weights to appropriately reflect the binary labels.

Thus, in implementing a methodological approach stated above, we were able to successfully fine-tune the CLIP model to predict binary-wise.

Comparing Optimizers and Learning Rate Hyperparameters

In training our fine-tuned CLIP model, we wanted to try different configurations of optimizers, batch sizes, and learning rates to identify the one with the best improvements in training and validation loss. Here are the different configurations we tested:

```
configurations = [
    {'optimizer': 'Adam', 'lr': 5e-6, 'batch_size': 16},
    {'optimizer': 'Adam', 'lr': 1e-5, 'batch_size': 16},
    {'optimizer': 'AdamW', 'lr': 1e-5, 'batch_size': 16},
    {'optimizer': 'SGD', 'lr': 1e-5, 'batch_size': 16, 'momentum': 0.9},
    {'optimizer': 'Adam', 'lr': 5e-6, 'batch_size': 8},
    {'optimizer': 'Adam', 'lr': 5e-6, 'batch_size': 32}
]
```

We used 10 epochs and saw that the top two performing configurations were the ones highlighted in green above. Please refer to the Python notebook to see the numerical improvement in the loss.

Results and Conclusion

After identifying the top two performing configurations, we then tested the models and computed the following metrics. Here are the results:

Testing configuration: {'optimizer': 'AdamW', 'lr': 1e-05, 'batch_size': 16}

- Accuracy: 0.6040
- Precision: 0.6081
- Recall: 0.9890
- F1 Score: 0.7531

Testing configuration: {'optimizer': 'Adam', 'lr': 1e-05, 'batch_size': 16}

- Accuracy: 0.6040

- Precision: 0.6212
- Recall: 0.9011
- F1 Score: 0.7354

We can see that both configurations achieved an accuracy of 0.6040 and similar F1 scores. Notably, the recall score for both is quite high, especially for the AdamW optimizer. This implies that our model is effective in identifying 'non-offensive' memes, capturing nearly all positive instances in the test set.

When testing our fine-tuned CLIP model against an image with its associated text, the probability score calculated from the 'predict' function represents how 'non-offensive' an image is (class label 1).

This essentially means that our model likely outputs a higher probability score when it predicts an image to be 'non-offensive'. Notably, the sigmoid output value tends to center around ~0.62, suggesting that our model is moderately biased towards predicting a meme to be 'non-offensive'. This tendency is most likely attributed to the nature of the label distributions across our training dataset - there's a larger count of 'non-offensive' examples in the dataset.

This imbalance between 'non-offensive' and 'offensive' labels was out of our control, so we couldn't do anything about it. However, from our observations, we think that the original provider for this data source likely did this to avoid false positives (incorrectly labeling a 'non-offensive' meme as 'offensive').

However, the reader might be thinking, isn't the *goal* of this project to be able to successfully identify *offensive* memes and flag them accordingly? So why are they focusing so much on avoiding false positives?

If we had our model more likely to label a meme as 'offensive', avid meme creators who do *not* post offensive content on social media may be falsely flagged, which would be unfair and problematic. Thus, it's important to strike a balance between our *goal* and the repercussions of doing so.

Nonetheless, we still believe there is a lot of room for improvement, especially due to the nature of our model, which gives a probability of ~0.62 for almost every passed in meme with its associated text. We could improve this by increasing and diversifying our training dataset to avoid overfitting with dominant features.

Model Deployment and Inference

Assuming the user has forked or pulled the latest changes from our repository, in order to deploy our model, they must first ensure they are in the Project3/ root directory where the docker-compose.yml and Dockerfile are located and run the following command(s):

```
$ docker-compose up
You may add '-d' to put it in daemon mode.
```

```
$ docker ps -a
To ensure that the container is up and running.
```

In doing so, the model should be successfully deployed and running inside a docker container.

Here are the possible endpoints to hit from the Flask app.py:

POST '/predict'

expects a full path to the image.png and associated text.

returns results of the inference in JSON.

From the command line, please run the following commands, respectively, to call endpoints:

```
$ curl -X POST -F "image=@/full/path/to/image.png" -F "text=meme's
associated sentence/text here" localhost:5000/predict
```

For the POST '/predict' that predicts a single sample image, the user should expect an output something like:

{"prediction":"Offensive","probability":0.6265420317649841}

The probability score calculated from the 'predict' function represents how 'non-offensive' an image is (class label 1).

Refer to the GitHub repo to see comprehensive examples of the curl commands and expected output.

ChatGPT

Due to the inexperience and complexity of fine-tuning the CLIP model, we heavily relied on chatGPT when it came to debugging issues we were facing. Without GPT's help, we would not have made the amount of progress we did. However, we made sure to carefully interact with chatGPT and ensure that we were learning along the way and not merely relying on it.

References

Suryawanshi, S., Chakravarthi, B. R., Arcan, M., & Buitelaar, P. (2020, May). Multimodal Meme Dataset (MultiOFF) for Identifying Offensive Content in Image and Text. *In Proceedings of the Second Workshop on Trolling, Aggression and Cyberbullying* (TRAC-2020). Association for Computational Linguistics.

ChatGPT