Codec at SemEval-2022 Task 5: Multi-Modal Multi-Transformer Misogynous Meme Classification Framework

Ahmed Mahran
Codec AI
Alexandria, Egypt
ahmed@codec.ai

Carlo Alessandro Borella
Codec AI
WC1N 2EB, London, UK
carlo@codec.ai

Konstantinos Perifanos
Codec AI
WC1N 2EB, London, UK
kostas@codec.ai

Abstract

In this paper we describe our work towards building a generic framework for both multi-modal embedding and multi-label binary classification tasks, while participating in task 5 (Multimedia Automatic Misogyny Identification) of SemEval 2022 competition.

Since pretraining deep models from scratch is a resource and data hungry task, our approach is based on three main strategies. We combine different state-of-the-art architectures to capture a wide spectrum of semantic signals from the multi-modal input. We employ a multi-task learning scheme to be able to use multiple datasets from the same knowledge domain to help increase the model's performance. We also use multiple objectives to regularize and fine tune different system components.

1 Introduction

In this paper, we present the system that we have built to participate in SemEval 2022 task 5 (Fersini et al., 2022), Multimedia Automatic Misogyny Identification (MAMI) challenge. The task is targeted at identification of misogynous memes by basically using the meme's image and pre-extracted English text content as input sources. The task is divided into two main sub-tasks: Sub-task A is a binary classification task where a meme should be categorized either as misogynous or not misogynous, Sub-task B is a multi-label binary classification task, where the type of misogyny should be recognized among the potential overlapping categories: stereotype, shaming, objectification and violence. Generally, meme classification is a challenging task as memes are multi-modal, rely heavily on implicit knowledge, and are subject to human misinterpretation especially among different backgrounds and cultures.

We have used transformer (Vaswani et al., 2017) based architectures and took a transformer based approach to combine them. Transformer based architectures are achieving state-of-the-art performance for Natural Language Processing and Computer Vision related tasks. There are architectures for language, vision, and language and vision combined. There are also architectures for other modalities however in MAMI's scope, we are interested in images and text only.

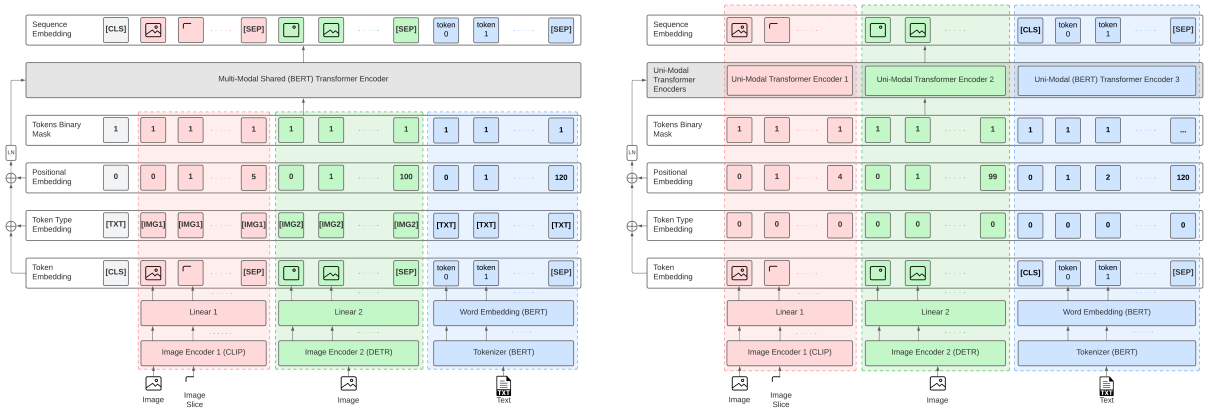
Pretraining a deep neural network and a transformer based architecture from scratch is a data hungry and computation resources demanding task. Especially in a multi-modal domain where input can have multiple image and text modalities. The literature is rich in a variety of architectures which achieve competitive performance in different tasks for different modalities. In our work, we took an approach towards building a framework that would allow us to combine different pretrained architectures. With relatively few epochs and using relatively less compute resources, our goal was to build and train a classifier framework that could harness the power of pretrained architectures as backbones, using relatively limited resources: no multiple GPUs for training and constrain the cost to be relatively small, in the range of hundred dollars in total.

We have assumed that using as many different backbone architectures which are trained on different tasks for different modalities can allow us to capture a wide spectrum of semantic signals from the input modalities. Then we just need to build a classifier to learn the relationship between these signals and the target classes.

We have also found and discuss below that there are available text and image datasets for hate speech, sexism and hateful memes which sound related to this task and could be used to augment MAMI's dataset. Eventually, our goal was to combine different datasets in a multi-task classifier.

In order to guide the model during training towards the main objective, which is to minimize

Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022), pages 679 - 688



(a) Multi-modal shared transformer encoder

(b) Uni-modal multi transformer encoders

Figure 1: Sequence embedding

the classification loss, we have employed different auxiliary objectives. Breaking down the whole architecture into components, each component has a sub-target. We have assumed that if we could assure that each component performs well on the sub-target, then the whole system would perform better on the main target. A component producing a clear signal can facilitate the learning of the downstream dependent components. This could increase model performance in terms of either accuracy or convergence speed. So, if we could formulate an objective function for each component, we can linearly combine them with the main objective function. This helps fine tuning and regularizing the components of the system.

We have built a generic classification framework and applied it to both sub-tasks A and B. During the evaluation phase of the competition, we have achieved, for sub-task A, a macro-average F1-Measure of 0.715, and for sub-task B, a weighted-average F1-Measure of 0.698. During post-evaluation phase we have achieved higher score for sub-task A of 0.761. Our code is available at `https://github.com/ahmed-mahran/MAMI2022`.

2 Background

Datasets: Besides MAMI dataset, there are many datasets that could be used to train our model. The input could be uni-modal as text only or image only, or bi-modal as pairs of image and text. (Vidgen and Derczynski, 2020) reviews 63 publicly available training datasets and they have published a dataset catalogue on a dedicated website [1]. We have used the hateful meme dataset created by Facebook AI (Kiela et al., 2020) which consists of $10K$ memes labeled hateful or not.

Multi-modal frameworks: MMBT (Kiela et al., 2019) concatenates, into a single sequence, linear projection of ResNet (He et al., 2016) output for image pooled to N different vectors, with BERT (Devlin et al., 2018) tokens embeddings for text. The sequence is fed into a transformer encoder, they call it a bi-transformer, after adding positional embeddings and segment embeddings to distinguish which part is image and which part is text. The architecture is generic enough to use different image encoders. As a variant of how we combine signals from image and text, we extend the MMBT architecture to combine more than two encoders however that wasn't our top performing variant. MMCA (Wei et al., 2020) combines Faster R-CNN (Ren et al., 2015) with BERT to compute two embedding types for each modality: self-attention embedding to capture intra-modality interactions, and cross-attention embedding to capture both intra- and inter-modality interactions.

3 System overview

3.1 Tokens embedding

At this stage, we encode each input modality into a sequence of vectors in a unified dimension space. We also generate a binary mask vector with length equal to the sequence's length to indicate which part of the sequence the model should consider. This produces the output at "Token Embedding" and "Tokens Binary Mask" layers illustrated in figure 1. We can use different encoders per modality and generate different sequence types to capture as

[1] hatespeechdata.com

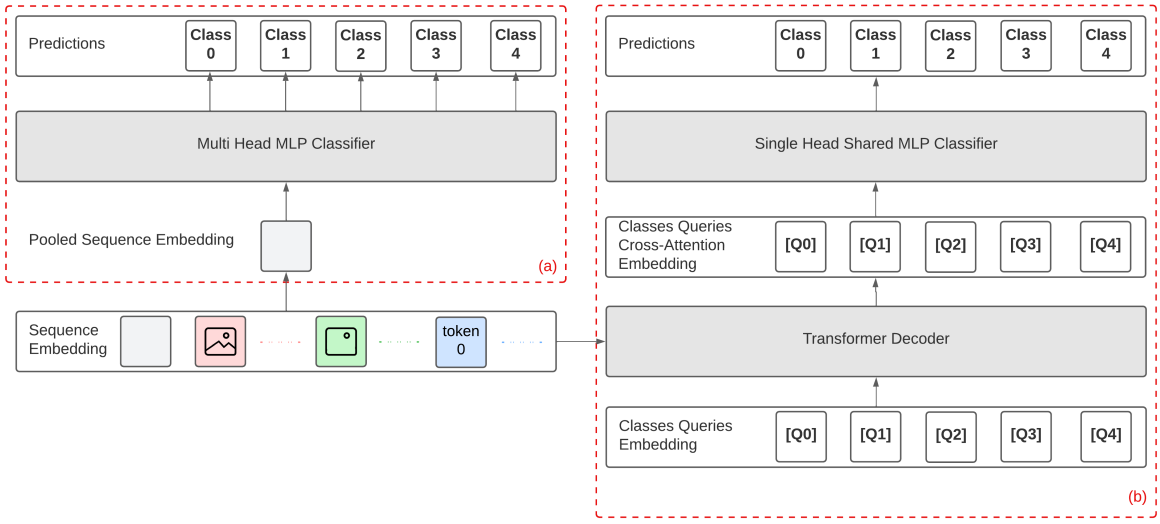


Figure 2: Classifiers: (a) using pooled sequence embedding, (b) using the whole non-pooled sequence embedding.

many semantic signals as possible. For our setup, we have tested CLIP (Radford et al., 2021) and DETR (Carion et al., 2020) encoders for image modality and BERT (Devlin et al., 2018) encoder for text modality.

CLIP image embedding: We split the image into 4 equal size patches (2×2) then we use CLIP (Radford et al., 2021) to encode the whole image along with its 4 slices into a sequence of 5 vectors. Then we project each vector into the model hidden dimension space. In our experiments, we have used CLIP model named "RN50x4" (which uses as backbone ResNet-50 scaled up 4x using the EfficientNet scaling rule (Tan and Le, 2019))[2].

DETR image objects embedding: We use DETR (Carion et al., 2020) [3] to encode the image into a sequence of 100 image objects representations. DETR's transformer decoder produces a sequence of 100 possible object boxes representations that we use as objects embeddings. However, not all of the objects are real objects as DETR can produce a no-object prediction. So, we use DETR's classifier which is trained on the object detection task to generate masks for no-objects. For each box from the 100, the classifier produces 92 logits which correspond to 92 possible object labels. We take the softmax of the 92 logits and mask out the corresponding object box if the label with the highest softmax probability is the no-object label. We fallback to another masking strategy if all the 100 boxes are masked out; we ignore the logit of the no-object label and then take the softmax of the rest labels to select 4 out of 100 boxes with highest softmax probabilities. Similarly with CLIP output, we project each vector into the model hidden dimension space.

BERT text embedding: We use BERT (Devlin et al., 2018) [4] pre-trained on hate speech (Mathew et al., 2020) to tokenize input text and generate tokens embeddings (for a maximum of 120 tokens). We don't project BERT's embeddings as we use its output space as the model's hidden dimension space (which has length of 768 dimensions).

3.2 Sequence embedding

At this stage, the model generates one combined sequence of vectors using tokens embedding from the tokens' embedding stage. This is the output in the "Sequence Embedding" layer illustrated in figure 1. For the token embedding output, we add token type embedding and positional embeddings that encode the position of each token in the corresponding input sequence per type then we apply a layer normalization (Ba et al., 2016). Along with input masks, the layer normalized sum of embeddings is fed to a multi-layer transformer encoder stage to produce the final sequence embedding at the "Sequence Embedding" layer in figure 1. The output sequence has the same number and dimen-

[2]We have used OpenAI implemantion on https://github.com/openai/CLIP

[3]We have used Huggingface Transformers implementation of DETR

[4]We have used Huggingface Transformers implementation of BERT with weights from "Hate-speech-CNERG/bert-base-uncased-hatexplain"

sionality of the input tokens. We have two variants for the transformer encoder stage:

Shared transformer encoder: The general architecture of the shared transformer encoder variant is illustrated in Figure 1a. What distinguishes this variant is that we use a shared multi-layer transformer encoder to capture the intra- and inter-modality interactions. Because of this, for each input embedding type, we add token type embedding which is the same for each input encoder to distinguish which tokens are from which input encoder. In our setup, we have three token types; that is one for each of: CLIP, DETR and BERT encoders. We also append a special [SEP] token at the end of each sequence type and we prepend to the whole combined sequence a global and special [CLS] token. We use the pre-trained BERT transformer encoder as the shared transformer encoder.

Multiple transformer encoders: The general architecture of this variant is illustrated in Figure 1b. The difference here is that instead of using one shared transformer encoder, we use a multi-layer transformer encoder per token type. We still add a token type embedding such that each transformer encoder learns its own token type parameters. We think we can remove this step however we have not tested this. Also, in this variant we don't need to add the extra [SEP] token per type and the global [CLS] token however we add a local [CLS] and [SEP] tokens for BERT only. For CLIP and DETR, we use PyTorch's implementation of the transformer encoder which is described in (Vaswani et al., 2017) with 8 heads and 6 layers but for BERT we keep it as in (Devlin et al., 2018). Then the final output sequence is just the concatenation of all sequences from each transformer encoder.

3.3 Classification

We have two modes of classification depending on the sequence length of the output of the transformer encoder stage. As shown in figure 2, we either use the whole sequence of vectors or pool it to one vector.

Multi-head MLP classifier: We use the pooled sequence embedding as classifier input. In our setup, we have used the first [CLS] token in the shared transformer encoder variant. The classifier is a two feed forward linear layers with a GELU activation in between and a hidden size of 768. The final layer produces number of logits equals to number of classes and we apply a sigmoid activation to compute each binary label probability.

Transformer decoder with single-head shared MLP classifier: Here the whole sequence embedding along with the binary mask is fed into a multi-head transformer decoder as a source sequence. Then the target sequence is formed by a learnable target class query embedding for each class in the target classes. The transformer decoder learns how each class interacts with each input modality signal through the cross-attention mechanism between the source and target sequences. Moreover, the decoder learns the dependency among the target classes through the self-attention mechanism for the target sequence. The decoder output is then fed into a single-head MLP classifier that shares parameters for all classes such that $MLP(q_i)$ is the logit of label i using the corresponding class query embedding, q_i, from the decoder output. The MLP classifier has the same architecture as the previous one in terms of number of layers, type of activation and hidden size, and similarly as well we compute the binary label probability for each class. We use PyTorch's implementation of the transformer decoder which is described in (Vaswani et al., 2017) with 8 heads and 6 layers.

3.4 Multi-task learning

In order to use more training data from other but similar datasets, we have followed a multi-task learning approach. For each dataset $\mathcal{D}$, there is a set of target labels $\mathcal{L}$, we can define as many tasks as the sets of labels in the power set $\mathcal{P}^+(\mathcal{L})$ (excluding the empty set) such that it is possible to use the same label in more than one task. Each task has a separate MLP classifier while all tasks across the datasets share the rest of the parameters including the learnable classes queries. During training, each mini-batch contains data from only one dataset and we compute the targets per task for all the tasks of the dataset.

3.5 Multi-objective

For a training instance i, we use x_i to refer to the input regardless from its actual representation, $y_{i,c} \in \{0, 1\}$ is the value of the target binary label c, and θ is the set of learnable parameters.

Main objective: We use a binary cross entropy to minimize the loss per label.

$$\begin{aligned}\mathcal{L}_0(i, c) = \; & y_{i,c}.\log p(c|x_i, \theta) \\ & +(1 - y_{i,c}).\log(1 - p(c|x_i, \theta))\end{aligned} \tag{1}$$

Algorithm 1 Multi-task learning

```
datasets ← {D_1, D_2, ...}
tasksPerD ← {(D_1, {T_1, T_2, ...}), ...}
labelsPerT ← {(T_1, {l_1, l_2, ...})}
for epoch ∈ epochs do
    for b ∈ mini-batches do
        dataset ← sample 1 from datasets
        tasks ← tasksPerD[dataset]
        for task ∈ tasks do
            learn b, task, labelsPerT[task]
        end for
    end for
end for
```

Token encoding projection alignment: We linearly project modal encoding into the hidden model space (as described in section 3.1). In order to preserve a similar structure of data points across spaces, we impose a cosine similarity constraint such that for any two input instances, x_i and x_j, the similarity, $s(.,.)$, between their encoding, $f(.)$, is the same as the similarity between the projection, $g(.)$, of their encoding. We apply this to all pairs in each batch.

$$\begin{aligned}\mathcal{L}_1(i,j) = \ & |s(f(x_i), f(x_j)) \\ & -s(g(f(x_i)), g(f(x_j)))|\end{aligned} \tag{2}$$

Contrastive embedding loss: This is intended at regularizing the embedding space of the MLP classifier to make instances of dissimilar labels more separable. For any two input instances, x_i and x_j, we apply the embedding loss per class c on the input, h^{l-1}, of each layer l of the MLP classifier.

$$\begin{aligned}&\mathcal{L}_2(i,j,c) = \\ &\frac{1}{2}\sum_l 1 - s_c(y_i, y_j) s_h(h^{l-1}_{i,c}, h^{l-1}_{j,c})\end{aligned} \tag{3}$$

$$s_c(y_i, y_j) = (2y_{i,c} - 1)(2y_{j,c} - 1) \tag{4}$$

$s_c(.,.) \in \{-1, 1\}$ is the labels similarity for class c of two instances; -1 indicates dissimilar labels while 1 indicates similar labels. $s_h(.,.) \in [-1, 1]$ is the cosine similarity of layer input when comparing two instances. It is worth noting that in case of the multi-head MLP classifier, $h^{l-1}_{i,c}$ is the same for all classes. We also apply this loss to all pairs in each batch. This loss encourages the transformer encoder in case of the multi-head MLP classifier, the decoder in case of the shared single head MLP classifier, as well as the hidden layers of the MLP classifier to produce embeddings with structures that capture labels similarity such that instances with the same label value get closer embeddings than instances with different label value.

The overall loss per batch and task given a dataset d:

$$\begin{aligned}\mathcal{L}^d_t = & \frac{1}{N_i N^t_c}\sum_{i,c}\mathcal{L}_0(i,c) \\ & + \frac{1}{N_i(N_i-1)}\sum_{i\neq j}\mathcal{L}_1(i,j) \\ & + \frac{1}{N_i(N_i-1)N^t_c}\sum_{i\neq j,c}\mathcal{L}_2(i,j,c)\end{aligned} \tag{5}$$

The overall loss per batch for all tasks of the dataset is the average task loss:

$$\mathcal{L}^d = \frac{1}{N^d_t}\sum_t \mathcal{L}^d_t \tag{6}$$

N_i is the batch size, N^d_t is the number of tasks for dataset d, and N^t_c is the number of classes for task t.

4 Experimental setup

In addition to MAMI's dataset, we have used Facebook's hateful memes dataset. We have set the tasks configurations as shown in table 1.

Dataset	Task	Labels
MAMI	MAMI	{misogynous, shaming , stereotype , objectification, violence}
	Task_A	{misogynous}
	Task_B	{shaming, stereotype , objectification, violence}
FBHM	Hateful	{hateful}

Table 1: Tasks labels configurations per dataset. FBHM is short for Facebook Hateful Meme.

We have added the redundant task MAMI for the MAMI dataset to make sure that the decoder learns the dependency among all labels as Task_B's labels depend on Task_A's label we wanted to make sure that the model captures this dependency.

We have split both datasets into 80% train and 20% dev sets using stratified sampling. We have used the data provided at post-evaluation period of the competition as the test set. We have used a batch size of 16 and number of epochs as 15. We have used MADGRAD (Defazio and Jelassi, 2021) for optimization. Learning rate

was set to 2×10^{-4} and we used a learning rate linear scheduler with a warmup period [5]. Gradients are accumulated every 20 batches so there was a total gradient accumulation steps of: total number of batches$/20 \times$ number of epochs. We have set the warmup period to: total gradient accumulation steps$/10$. For all parameters except biases and layer normalization weights, we have used a weight decay of 5×10^{-4}. We clip gradients to overall norm of 0.5. Our implementation is PyTorch based and we have used HuggingFace Transfomers implementation for both BERT and DETR. We have run our experiments on Google Colab Pro plus using one Tesla P100 GPU. We didn't perform any special data preprocessing, just the requirements for each backbone. Also, to make experiments quicker, we didn't perform any fine tuning for any of the backbones and we pre-generated and stored each backbone output instead of re-evaluating the same data across epochs and experiments.

We used the official accuracy measures to score how the model performs on each task. For the single class tasks, we have used macro-average F1-Measure and we called it scoreA. In particular, for each class label (i.e. true and false) the corresponding F1-Measure will be computed, and the final score will be estimated as the arithmetic mean of the two F1-Measures. For the multi class tasks, we have used weighted-average F1-Measure and we called it scoreB. In particular, the F1-Measure will be computed for each label and then their average will be weighted by support, i.e. the number of true instances for each label.

5 Results

5.1 Ablations

We have the following system configurations [6] which would result in different architecture variations.

Transformer encoder (Xformer Enc) whether to use shared transformer (**Shared**) or multi transformers (**Multi**).

Encoder output pooling (Pooling) whether the whole sequence is pooled using [CLS] embedding (**[CLS]**), this means we use the multi-head MLP classifier and no decoder), or no pooling (**No**) or only text tokens are pooled using text's [CLS] token (**txt [CLS]**), this means we use the decoder with the shared single head MLP classifier.

Token encoding projection alignment (Proj Align) whether to enable it (**Yes**) or not (**No**).

Contrastive embedding loss (Contrastive) whether to enable it (**Yes**) or not (**No**).

Multi-task learning (Multi-task) whether to use Facebook's hateful meme dataset (**Yes**) or not (**No**).

Image encoders (Backbones) whether to use CLIP only (**CLIP**), DETR only (**DETR**), or both together (**CLIP and DETR**).

We plan experiments as tournament of rounds such that in each round we test subset of the configurations fixing the rest. Then we use the winning configuration values for subsequent rounds. For each experiment, we report the max score for Task_A and Task_B on all splits. Appendix A contains more details on scores distributions.

5.1.1 Round 1

At this round we compare transformer encoder and encoder output pooling methods. We perform four experiments with configurations and results summaized in table 2. The winner of this round is the multi-transformers encoders without output pooling architecture variant.

Experiment	00	01	02	03
Xformer Enc	Shared	Shared	Multi	Multi
Pooling	[CLS]	No	No	txt [CLS]
Proj Algn	No			
Contrastive	No			
Multi-task	No			
Backbones	CLIP and DETR			
Score				
Test - Task_A	0.6819	0.7226	**0.7436**	0.7329
Test - Task_B	0.5886	0.6422	**0.6785**	0.6772
Dev - Task_A	0.8395	0.8355	**0.8564**	0.8519
Dev - Task_B	0.6650	0.6933	**0.7420**	0.7310
Train-Task_A	**0.9253**	0.9121	0.9066	0.8998
Train-Task_B	0.7006	0.7242	**0.7782**	0.7647

Table 2: Experiments configurations for round 1 and corresponding results.

5.1.2 Round 2

At this round, we test the significance of the multi-objective approach. The configurations and corresponding results are summarized in table 3. To-

[5] We used get_linear_schedule_with_warmup from Huggingface Transformers.

[6] We give each item a short name in parenthesis to be able to refer to corresponding items in experiments results tables compactly.

ken encoding projection alignment makes improvements on both tasks on the test split when enabled alone. Also, contrastive embedding loss seems to slightly improve Task_B. After performing statistical tests and comparing distributions of the scores, we pick experiment 10 as the winner variant.

Experiment	02	10	12	13
Xformer Enc	Multi			
Pooling	No			
Proj Algn	No	Yes	No	Yes
Contrastive	No	No	Yes	Yes
Multi-task	No			
Backbones	CLIP and DETR			
Score				
Test - Task_A	0.7436	**0.7504**	0.7358	0.7467
Test - Task_B	0.6785	0.6798	**0.6823**	0.6777
Dev - Task_A	**0.8564**	0.8535	0.8515	0.8535
Dev - Task_B	**0.7420**	0.7387	0.7371	0.7313
Train-Task_A	0.9066	0.8983	**0.9090**	0.8988
Train-Task_B	**0.7782**	0.7679	0.7676	0.7573

Table 3: Experiments configurations for round 2 and corresponding results compared to best configurations from round 1.

5.1.3 Round 3

At this round, we test the significance of the image encoders backbones, namely: CLIP and DETR. The configurations and corresponding results are summarized in table 4. It seems that our use of DETR was incompetent to CLIP.

Experiment	10	20	21
Xformer Enc	Multi		
Pooling	No		
Proj Algn	Yes		
Contrastive	No		
Multi-task	No		
Backbones	CLIP and DETR	CLIP	DETR
Score			
Test - Task_A	**0.7504**	0.7426	0.7010
Test - Task_B	0.6798	**0.6865**	0.6306
Dev - Task_A	0.8535	**0.8560**	0.7979
Dev - Task_B	**0.7387**	0.7347	0.6784
Train-Task_A	0.8983	**0.8990**	0.8186
Train-Task_B	**0.7679**	0.7628	0.6704

Table 4: Experiments configurations for round 3 and corresponding results compared to best configurations from round 2.

5.1.4 Round 4

At this round, we test the significance of the additional training data from Facebook's hateful meme dataset. The configurations and corresponding results are summarized in table 5. This time we train for more 15 epochs (i.e. total 30 epochs). We can notice a significant improvement when using more training data from the external dataset.

Experiment	10	30
Xformer Enc	Multi	
Pooling	No	
Proj Algn	Yes	
Contrastive	No	
Multi-task	No	Yes
Backbones	CLIP and DETR	
Score		
Test - Task_A	0.7504	**0.7609**
Test - Task_B	0.6798	**0.6958**
Dev - Task_A	**0.8535**	0.8502
Dev - Task_B	0.7387	**0.7429**
Train-Task_A	0.8983	**0.9127**
Train-Task_B	0.7679	**0.7815**

Table 5: Experiments configurations for round 4 and corresponding results compared to best configurations from round 2.

5.2 Visualizations

In figure 3, we show t-SNE projections of the transformer decoder output per class and the corresponding class learnt input query embedding. We use data from experiment 30 in section 5.1.4. It is interesting that, for all classes, the class learnt query embedding is positioned on the side with denser positive labels. This indicates that the learnt class queries can be thought of as centers of positive labels.

In figure 4, we show average attention weights per transformer decoder layer. Figure 4a illustrates the dependencies between each class and other classes. As shown in the figure, numbers are very close which indicates that each class depends uniformly on other classes. Figure 4b shows the average cross-attention weights between the source and target sequences from the transformer decoder layers and aggregated per input encoder. Clearly, the model pays more attention to CLIP features, and less attention to BERT text features, and the least attention to DETR objects features. This conforms with results from round 3 in section 5.1.3.

6 Conclusion

In this paper, we propose a generic framework for both multi-modal embedding and multi-label binary classification tasks. We combine and use as backbones different architectures achieving state-of-the-art in different or similar tasks on different modalities to capture a wide spectrum of semantic signals from the multi-modal input. By employing a multi-task learning scheme, we are able to use

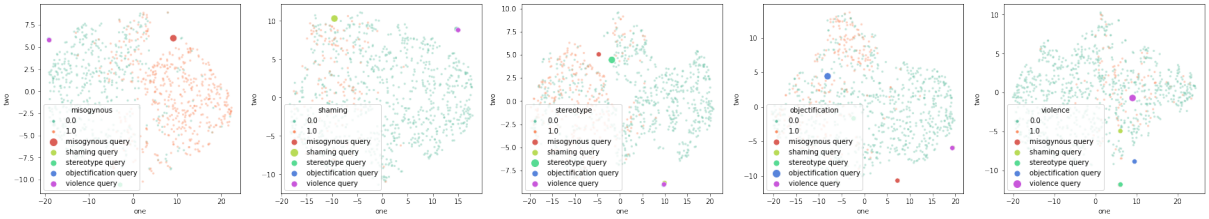
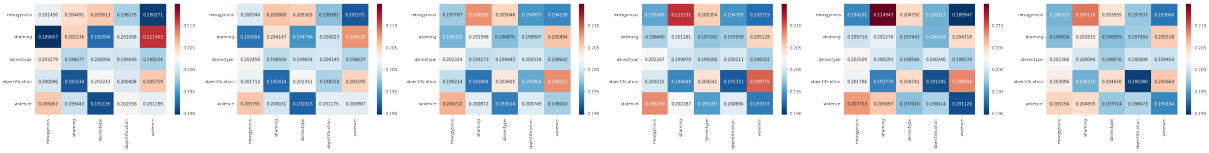
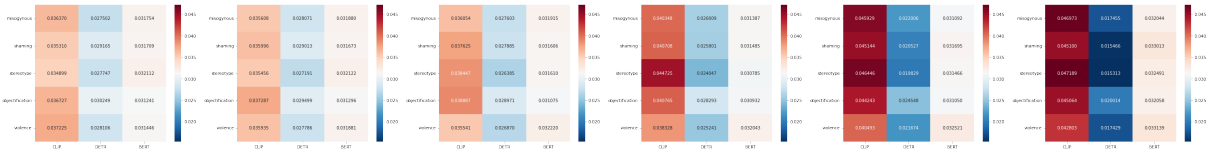


Figure 3: t-SNE projections of transformer decoder output per class (small dots) and the corresponding class query embedding (the biggest dot).



(a) Transformer decoder average self-attention weights



(b) Transformer decoder average source × target cross-attention weights. Source weights are aggregated per input encoder.

Figure 4: Attention weights visualization per transformer decoder layer. First input layer is on the left while last output layer is on the right.

multiple datasets from the same knowledge domain and increase the model's performance. In addition to that, we use multiple objectives to regularize and fine tune the system components. We have carried out experiments to verify our ideas and the results show the significance of some of the ideas. As a future work, we need to do more experiments with different backbone architectures and more datasets. We can also try more objectives and regularizations; for instance, we can use our observation from figure 3 to make the decoder output for a positive instance closer to the corresponding class query embedding.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer.

Aaron Defazio and Samy Jelassi. 2021. Adaptivity without compromise: a momentumized, adaptive, dual averaged gradient method for stochastic optimization. *arXiv preprint arXiv:2101.11075*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Elisabetta Fersini, Francesca Gasparini, Giulia Rizzi, Aurora Saibene, Berta Chulvi, Paolo Rosso, Alyssa Lees, and Jeffrey Sorensen. 2022. SemEval-2022 Task 5: Multimedia automatic misogyny identification. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*. Association for Computational Linguistics.

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*, pages 770–778.

Douwe Kiela, Suvrat Bhooshan, Hamed Firooz, Ethan Perez, and Davide Testuggine. 2019. Supervised multimodal bitransformers for classifying images and text. *arXiv preprint arXiv:1909.02950*.

Douwe Kiela, Hamed Firooz, Aravind Mohan, Vedanuj Goswami, Amanpreet Singh, Pratik Ringshia, and Davide Testuggine. 2020. The hateful memes challenge: Detecting hate speech in multimodal memes. *Advances in Neural Information Processing Systems*, 33:2611–2624.

Binny Mathew, Punyajoy Saha, Seid Muhie Yimam, Chris Biemann, Pawan Goyal, and Animesh Mukherjee. 2020. Hatexplain: A benchmark dataset for

explainable hate speech detection. *arXiv preprint arXiv:2012.10289*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

Mingxing Tan and Quoc Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Bertie Vidgen and Leon Derczynski. 2020. Directions in abusive language training data, a systematic review: Garbage in, garbage out. *Plos one*, 15(12):e0243300.

Xi Wei, Tianzhu Zhang, Yan Li, Yongdong Zhang, and Feng Wu. 2020. Multi-modality cross attention network for image and sentence matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10941–10950.

A Experiments scores distributions

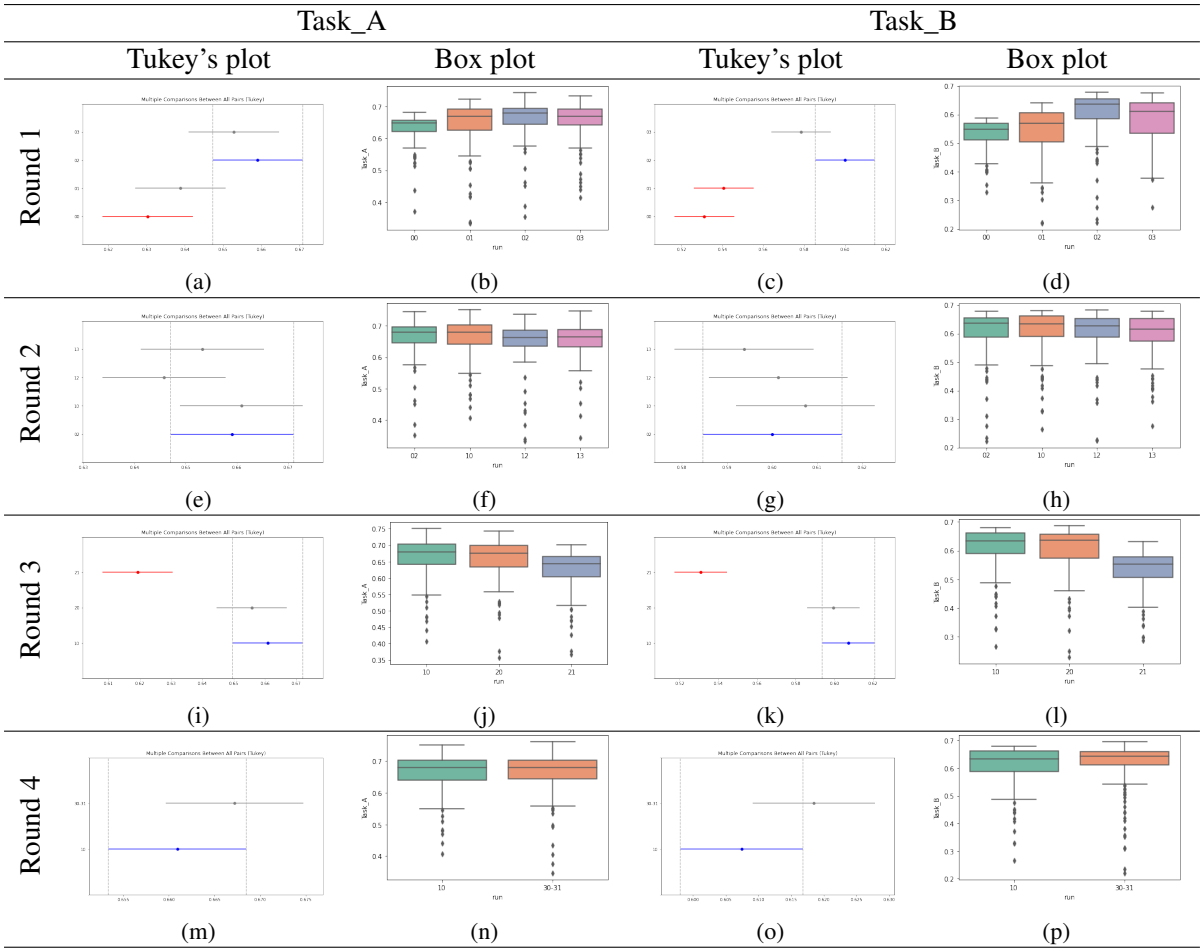


Figure 5: Performance comparison on test split for Task_A and Task_B collected from evaluation phases during model training epochs for different experiments described in 5.1. Tukey's plots visualize a universal confidence interval of scores mean on x-axis for each run on y-axis, any two runs can be compared for significance by looking for overlap. Box plots summarize the distribution of scores on y-axis for each run on x-axis.