Machine Learning for Statistical NLP: Advanced

Year: 2025/2026

Final project report

Hateful Meme classification

5th November 2025

Andrea De Biasi

Index

1 Introduction

Disclaimer: Most images classified as "Hateful" in the dataset are offensive and racist, I tried to put in this report the least Hateful I could find.

The proliferation of multimodal content, specifically **memes (Image + Text)**, on social media platforms presents a significant challenge for automated content moderation. Hate speech detection in this context is inherently difficult because the malignant intent is often **implicit** and relies on the semantic interaction between the visual and linguistic components, frequently involving sarcasm, irony, or visual misdirection. This complexity necessitates the use of robust **multimodal models** capable of fusing information from both domains, as unimodal approaches fail the task.

1.1 The Facebook Hateful Memes Challenge (2020)

The idea for this project came after seeing online the official **Facebook Hateful Memes Challenge** (2020) [1], which established the idea of creating a multimodal classification solution.

- ◇ **Goal and Metric:** The challenge required participants to perform **binary classification** (Hateful vs. Non-Hateful). The official evaluation metric was the **Area Under the ROC Curve (AUROC)**, which is robust against the natural class imbalance present in such datasets.
- ◇ **Stakes:** The competition featured a substantial prize pool, totalling $100,000, underscoring the real-world value of a successful solution. The first-place team was awarded $50,000.
- ◇ **State-of-the-Art (SOTA):** Winning solutions relied on complex methodologies, primarily involving large **Ensembles** of specialized Vision-and-Language (V+L) models, such as ERNIE-ViL, VisualBERT, UNITER, and OSCAR. These approaches demanded intensive pre-processing, feature extraction, and other advanced techniques.

2 Dataset Overview

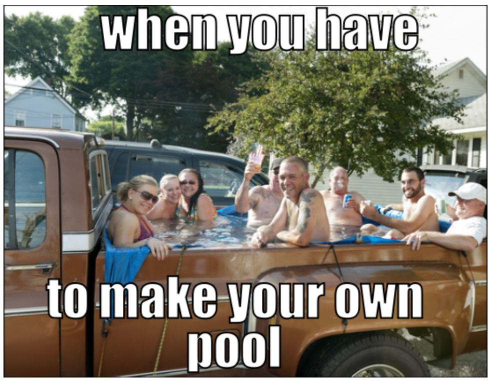
The dataset, sourced from the Facebook Hateful Memes Challenge, consists of **10,000 multimodal samples**. Each entry includes a meme image and its corresponding text, curated to test subtle forms of hate speech.

Table 1: Hateful Memes Dataset Split

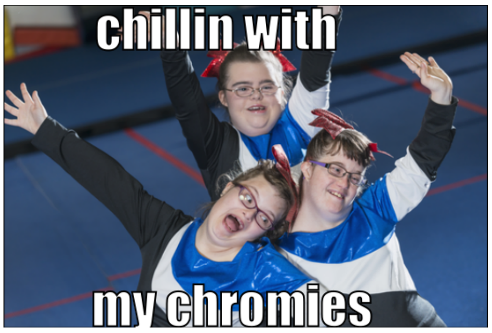
Split	**Purpose**	**Size (Approx.)**
Train	Model Training	8,500
Dev (Validation)	Hyperparameter Tuning / Loss calculation	500
Test	Final Performance Evaluation (Unseen)	1,000

3 Methodology: CLIP-based Classification

The primary goal of this project is to achieve a performance level comparable to these complex SOTA ensembles, but using a simpler, more resource-efficient **Contrastive Learning** approach based on **CLIP** (Contrastive Language-Image Pre-training)[3].



(a) Example of **NON HATEFUL** meme.



(b) Example of **HATEFUL** meme.

Figure 1: Examples of memes from the dataset

3.1 CLIP Architecture

CLIP is fundamentally a dual-encoder model designed to learn which image and text pairs match in a large dataset of 400 million image-text pairs.

- ◇ **Components:** It consists of two independent encoders: a **Text Encoder** (Transformer-based) and an **Image Encoder** (Vision Transformer, ViT).
- ◇ **Cross-Modal Alignment:** Crucially, both encoders project their respective inputs into a **shared, low-dimensional embedding space**. This pre-alignment allows us to calculate the similarity between an image and a text string using simple metrics like cosine similarity, which is the core principle used for zero-shot classification.

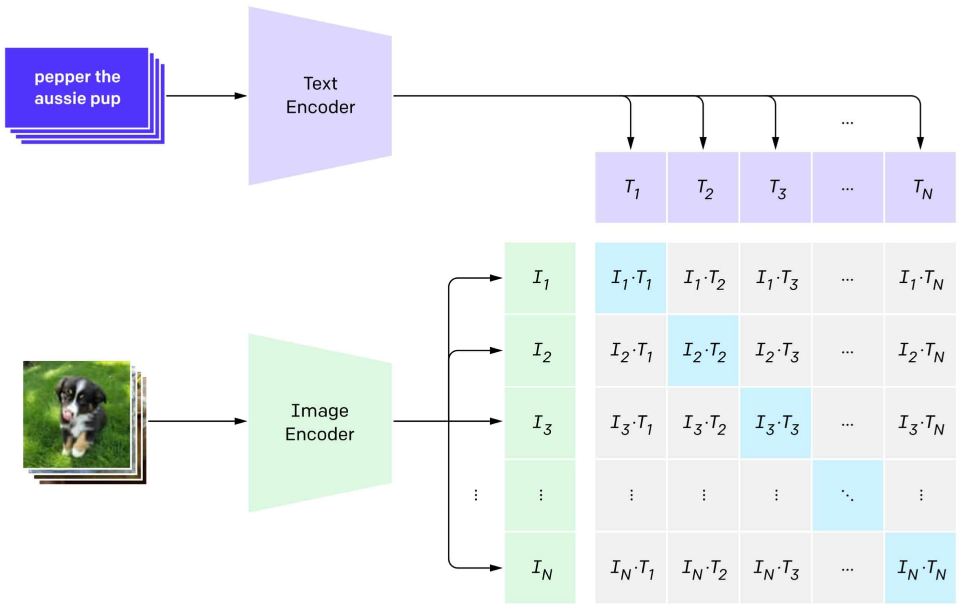

pepper the aussie pup
Text Encoder
Image Encoder
T1 T2 T3 ... TN
I1 I2 I3 ⋮ IN
I1·T1 I1·T2 I1·T3 ... I1·TN
I2·T1 I2·T2 I2·T3 ... I2·TN
I3·T1 I3·T2 I3·T3 ... I3·TN
IN·T1 IN·T2 IN·T3 ... IN·TN

Figure 2: Schematic of the CLIP dual-encoder architecture, which projects image and text into a joint embedding space. (Adopted from CLIP original paper)

3.2 Late Fusion and Classification Head

We implemented a **Late Fusion** strategy by extending the pre-trained CLIP model (ViT-B/32) with a small, trainable classification layer, as defined in the custom `CLIPClassifier` module.

- **Feature Extraction:** The image and text embeddings are generated by the two frozen CLIP encoders. The features are then **L2-normalized**.
- **Fusion:** The resulting 512-dimensional image feature and 512-dimensional text feature are **concatenated** to form a 1024-dimensional fused vector. This concatenation is defined as Late Fusion, as the fusion happens **after** the initial feature extraction steps.
- **Classification Head:** A simple Multi-Layer Perceptron (MLP) is placed on top of the fused vector to perform the final binary classification:

$$\text{Classifier}(\mathbf{f}_{\text{fused}}) = \text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Dropout} \rightarrow \text{Linear}_{\text{output}}$$

3.3 Training Strategy

To prevent catastrophic forgetting and maintain computational efficiency, a transfer learning approach was adopted.

- **Freezing:** The entire **CLIP backbone** (Image and Text encoders) is initially **frozen** (param.requires_grad = False).
- **Training:** Only the parameters of the lightweight **Classification Head** are optimized using the AdamW optimizer.
- **Partial Unfreezing:** In a subsequent refinement step, the last transformer block of the Vision Transformer (model.clip.visual.transformer.resblocks[-1]) is **unfrozen** to allow some task-specific adaptation of the image encoder.

```python
class CLIPClassifier(nn.Module):
    def __init__(self, clip_model, num_classes):
        super().__init__()
        self.clip = clip_model
        # ... (Freeze parameters) ...
        embedding_dim = self.clip.token_embedding.weight.shape[-1]
        fusion_dim = embedding_dim * 2

        self.classifier = nn.Sequential(
            nn.Linear(fusion_dim, fusion_dim // 2),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(fusion_dim // 2, num_classes) # Output: 2 classes (0 or 1)
        )
    # ... (omissis) ...
    def forward(self, images, text_tokens):
        with torch.no_grad():
            image_features = self.clip.encode_image(images)
            text_features = self.clip.encode_text(text_tokens)

        # 2. Normalize and Concatenate (Late Fusion)
        # ... (omissis) ...
        fused_features = torch.cat((image_features, text_features), dim=1)

        logits = self.classifier(fused_features.float())
        return logits
```

Listing 1: Implementation of the `CLIPClassifier` with Late Fusion

3.4 Training and Evaluation Procedures

The model is trained using the standard **Cross-Entropy Loss** (criterion = nn.CrossEntropyLoss()). Given the official metric of the challenge, evaluation and model selection are based on the Area Under the ROC Curve (AUROC) on the development (validation) set.

3.4.1 AUROC Calculation

The evaluation is performed by the `evaluate_model` function, which explicitly calculates the AUROC. This requires collecting the model's output probabilities for the positive class (Hateful, label 1) and the true labels across the entire validation set. The `roc_auc_score` function from the `sklearn.metrics` [2] library is used for the final calculation, as shown below:

```python
# ... (inside evaluate_model function) ...
with torch.no_grad():
    for images, text_tokens, labels, has_label in loader:
        # Filter valid samples and get model outputs
        # ...

        # 2. Transformation of logits into probabilities
        probabilities = torch.softmax(outputs, dim=1)

        # Probability of the positive class (HATE)
        all_probabilities.extend(probabilities[:, 1].cpu().numpy())
        all_labels.extend(labels_valid.cpu().numpy())

# 3. AUROC calculation (Scikit-learn)
if total_samples > 0 and len(set(all_labels)) > 1:
    auroc = roc_auc_score(all_labels, all_probabilities)
# ...
return avg_loss, accuracy, auroc
```

Listing 2: Implementation detail of the `evaluate_model` function focusing on AUROC.

3.4.2 Model Selection

The training loop (`train_model`) employs an early stopping mechanism by conditionally saving the model based on the validation performance. Specifically, the Classification Head's weights are saved only when the current epoch's **Dev AUROC** exceeds the previously recorded best AUROC score. This ensures the final model is optimized according to the official competition metric.

4 Results and Discussion

The project aimed to establish a strong baseline using the resource-efficient CLIP Late Fusion model, demonstrating that pre-aligned multimodal embeddings can lead to competitive results without the need for complex V+L ensembles.

4.1 Quantitative Performance

The model was trained for 15 epochs. The performance was tracked against the random baseline (0.5000 AUROC).

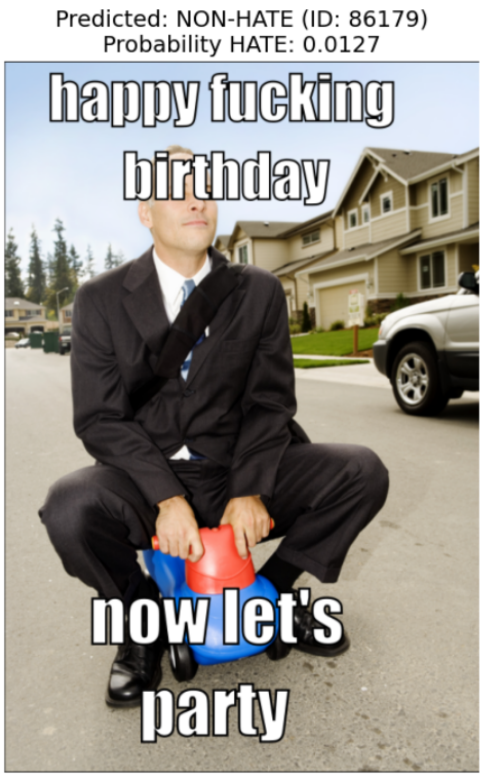
- **Baseline (Random):** 0.5000 AUROC

- ⋄ **SOTA Ensembles Target:** $\approx 0.87 - 0.89$ AUROC
- ⋄ **Our CLIP Model (Best Dev Score): 0.7227** AUROC (Achieved in Epoch 8/15)

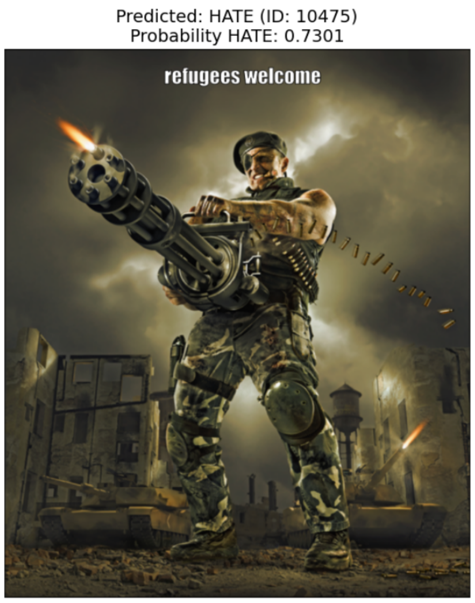
The achieved AUROC of 0.7227 demonstrates that the CLIP-based model successfully generalized the task, classifying the model in the **top 35 places** among more than 3,000 participants in the original competition. This result validates the hypothesis that CLIP's pre-aligned representations are highly effective for multimodal hate speech detection, offering a much simpler and more resource-efficient alternative compared to the complex winning ensembles.

4.2 Qualitative Analysis

Analyzing the model's predictions provides insight into its decision-making. We can see in the figure the score for each prediction. A score above 0.5 gets considered as *HATEFUL*.



(a) Prediction of **NON HATEFUL** meme.



(b) Prediction of **HATEFUL** meme.

Figure 3: Prediction of 2 random memes from the dataset

5 Feature Ablation Analysis

To gain insights into the inner workings of our multimodal classifier, a **Feature Ablation Analysis** is planned. This technique is a crucial interpretability method used to quantify the dependency of a model's prediction on its input components.

5.1 Concept and Methodology

Feature ablation involves **neutralizing one or more input modalities** to measure the resulting drop in model performance. This degradation serves as a proxy for the relative importance or contribution of the ablated feature to the final result.
The analysis is performed on the best-performing **multimodal** model after the classifier (MLP Head) is fully trained and its weights are saved. The procedure consists of evaluating the model in three distinct configurations by manipulating the input provided to the late fusion layer (concatenation):

1. **Baseline (Multimodal)**: The model uses both normalized features, f_{img} and f_{text}, concatenated to form the input $f_{\text{img}}||f_{\text{text}}$.
2. **Image Ablation (Text-Only)**: The visual embedding f_{img} is replaced with a zero vector of equal dimension. The classifier operates only on textual information: $0||f_{\text{text}}$.
3. **Text Ablation (Image-Only)**: The textual embedding f_{text} is replaced with a zero vector of equal dimension. The classifier operates only on visual information: $f_{\text{img}}||0$.

Performance will be primarily measured using the Area Under the Receiver Operating Characteristic curve (**AUROC**). The difference (Δ) in AUROC between the Baseline and the ablated configurations will quantify the loss of predictive power.

5.2 Results and Discussion (To be completed)

Due to ongoing training and time constraints, the full evaluation for the Feature Ablation Analysis has not yet been executed.

Interpretation Plan:
The final analysis will focus on comparing the AUROC Drop (Δ) values. The modality whose removal causes the largest degradation in AUROC performance will be considered the **most influential feature** for the Hateful Memes classification task. This will provide key insights into whether the model relies more heavily on the visual content of the meme or the textual overlay to make its classification decision.

References

[1] Douwe Kiela et al. *The Hateful Memes Challenge: Detecting Hate Speech in Multimodal Memes.* 2021. arXiv: `2005.04790 [cs.AI]`. URL: `https://arxiv.org/abs/2005.04790`.

[2] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[3] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision.* 2021. arXiv: `2103.00020 [cs.CV]`. URL: `https://arxiv.org/abs/2103.00020`.