III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally?

   If you are on a team, please complete this block for each member of the team.

   I'm an undergrad at PKU studying Comp. Science & Business Management. I'm really humbled (& fascinated) that today we can create something more intelligent than ourselves (even if only in narrow areas), so I devote most of my time to studying AI. Especially, transformers, graph neural networks, reinforcement learning & putting the software to work in robotics are my areas of interest!

2. What motivated you to compete in this challenge?

   I really want to intern at Facebook AI Research, so I thought this would be a great opportunity to learn more about the work they do. I think we learn best if we're in an environment surrounded by people smarter than us, so it was pretty tough participating in this competition alone – I hope I'll be able join such an environment at a place like FAIR one day! 😊

3. High level summary of your approach: what did you do and why?

   Current state of the art (2020) in NLP tend to be transformer models. Since the problem at hand requires combining image & text input, it lends itself to the opportunity of applying the same NLP transformers to images. Yet since images contain far more information than words, it makes sense to pre-extract the most important content first via CNN's such as FasterRCNN due to hardware limits. The transformer models I have implemented then take two different approaches of dealing with the extracted images and the text:

   a) Run them through separate transformers and then combine them at the very end; A two-stream model
   b) Run them in parallel through the same, big, transformer; A one-stream model

   The second approach is superior as it allows the model to early-on understand the connections between the image and the text, yet I still include both types of models and average all of their predictions to obtain more diverse & stable results.

   I think the approach could be improved by feeding in the original images into transformers / graph-like networks without extraction, however current hardware limits make this unfeasible and hence we make use of locally limited convolution operations in the extraction.

4. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

a) *Applying SWA*

```
                if (args.swa) and (ups > self.swa_start):
                    self.swa_model.update_parameters(self.model)
                    self.swa_scheduler.step()
```

Stochastic Weight Averaging is great when you want to train models “blind”. There was not a lot of data accessible so I decided to train models on all the data possible without validation, but use SWA to obtain stable results at the end. I still use models trained with validation, however to get score estimates and weights for the ensemble.

b) *Gather-Indexing img & text*

```
gather_index = gather_index.unsqueeze(-1).expand(
            -1, -1, self.config.hidden_size)
        embedding_output = torch.gather(torch.cat([txt_emb, img_emb], dim=1),
                                        dim=1, index=gather_index)
```

The above is part of the UNITER model (all credit goes to the original UNITER-team), but it such a beautiful & compact way of dealing with the two modalities. BERT/NLP-Models are normally always used with a certain sequence length and the unused fields are just padded e.g. with 0’s and ignored by the model. Hence many models have opted for an approach of padding both text & image and having separate sequence lengths, i.e. just duplicating the NLP approach for images. The gather index above, however, takes the image sequence and sticks it back on to the text and hence gives us the most compact input for every text-image sequence of variable length possible. I am a big fan of the notion that compression = intelligence, so such simplifications are really fascinating to me!

c) *Using Multi-Sample dropout*

```
            output = torch.mean(
                torch.stack(
                    [self.classifier(self.high_dropout(seq_out)) for _ in
range(5)],
                    dim=0,
                ),
                dim=0,
            )
```

Multi-sample dropout is an idea that originated in the following paper: https://arxiv.org/abs/1905.09788. In essence, we take a transformers hidden output and feed 5 copies of it through a dropout and subsequent classifier. For each of the 5, this drops random values out of it and then creates a classification output. We then take the mean of those 5 classification outputs to be our final output. Intuitively, it is like ensembling the output of a model with random copies of itself.

5. Please provide the machine specs and time you used to run your model.

- *CPU (model): Intel(R) Xeon(R) CPU @ 2.20GHz*
- *GPU (model or N/A): NVIDIA P-100*
- *Memory (GB): 25GB*
- *OS: Linux*
- *Train duration: 25h (Train + Inference)*
- *Inference duration: 5h (Inference only)*

6. Please list any external data or pre-trained models you used to develop your winning submission..

   I have included links to all the pre-trained models in the SCORE_REPRO.md & the README.md. I only use pre-trained models and the original data.

   - *OSCAR Large Pre-Trained*
   - *UNITER Large Pre-Trained*
   - *VisualBERT Re-pretrained by MMF*
   - *ERNIE-VIL Small Pre-Trained & VCR Fine-tuned*
   - *ERNIE-VIL Large Pre-Trained & VCR Fine-tuned*

7. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

Not a quick overview but here's my list:

HM - Directions tried & scraped:

- *Data Augmentation*
  - *Images via swaps etc / NLP via swaps, insertions - Theory: Images are often made up of two images and may even follow an order with the text, hence flips may hinder learning; NLP swaps showed some promise, but not stable -- I assume there is already enough noise in meme text*
- *Roc-Star Loss*
  - *A custom loss to optimize roc auc; Did help sometimes but not reliably, I decided to stick with CrossEntropy (NLLLoss + Logsoftmax)*
- *Hyperparameter Tuning*
  - *Ended up not tuning a lot, as changes in architecture / data have a far bigger impact than minor HP changes (I just sticked with standards used in previous similar problems)*
- *Correcting words*
  - *Set up a dict to replace e.g. nigga, niqqa etc on all occassions with nigger or maga with make america great again.. etc - Did not help - Theory: Too few such occassions; There are differences which the NN learns, when sb uses the qq vs the gg version (To find such, I tried to use sound / character distances)*
- *Adding subwords*
  - *Often esp. with hashtags words are concatted, but at the same time many words in BERT Vocab are only known as non-subwords, hence the idea was*

        to reinclude all words as subwords (i.e. increasing BERT vocab by about 20K) -- Did not find a good way to do this, even asked on Huggingface forums

- *Pretrain BERT on jigsaw hate speech, text only*
    - *Did show an improvement, but not over pretrained v+l models --- pretraining a new v+l model with a jigsaw pre-trained bert could be show further improvements*
- *Not relying on pre-trained models*
    - *I implemented lots of newer language transformers within some of the VL models, such as Roberta or Albert for LXMERT. They did perform well, especially when also task specific pre-trained, but still came about absolute 1-3% short of the pretrained model. I think pretraining them from scratch (i.e. on COCO etc) could give a solid performance boost*
- *Adding Generated Captions to the text via [SEP]*
    - *Worked in the beginning, but only because my models lacked the ability to do good with the features*
- *Adding predicted objects & attributes to the text via [SEP] similar to OSCAR*
    - *Oddly did not help at all not even for OSCAR*
- *Using features with different min_boxes & max_boxes with padding*
    - *Did not improve anything; Probably the model learns to focus on the first few boxes only anyways as they are sorted by confidence*
- *Label Smoothing*
- *Ensemble of ensembles*
- *Reiniting final layers & pooler*
    - *Did help initially, but not when pretraining - Could be used for the models we are not pretraining*
- *Adding new words*
        - *Against Jacob's recomm. I tried & learned. (My intuition was that BERT will never have seen words such as handjob in the given context, but probably finetuning is solving such issues already even though embeddings do not change during finetuning - Worsened rcac by 3% on a Bert-only)*
        - *https://github.com/google-research/bert/issues/9*
    - *Flagging profanity words with a tag*
        - *https://github.com/RobertJGabriel/Google-profanity-words*

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

    No.

9. How did you evaluate performance of the model other than the provided metric, if at all?

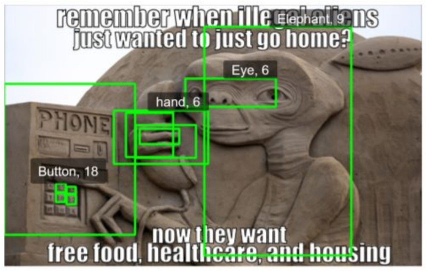
    Roc-Auc & Accuracy.

10. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

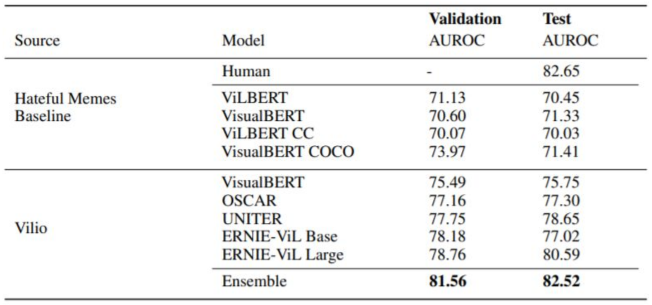
    The features tend to be big in size, hence copying them into both the data & ernie-vil/datafolder requires a lot of memory. In practice due to my hardware limits, I always

trained all models in isolation, and then just combined all the csv files exactly how it is outlined in the reproduction md file.
Apart from that, two models are implemented with PaddlePaddle, which has a very different structure than PyTorch, yet those are the two models that perform the best.

11. Do you have any useful charts, graphs, or visualizations from the process?

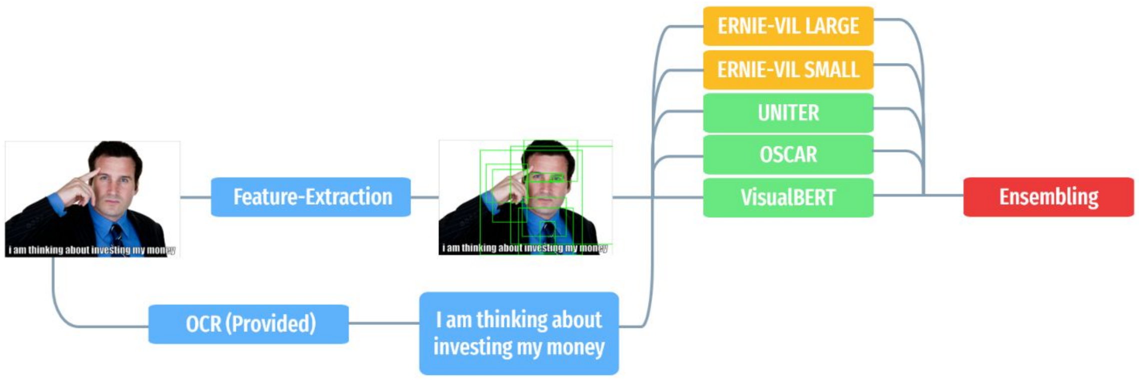

remember when illegal aliens just wanted to just go home?
Elephant, 9
Eye, 6
hand, 6
Button, 18
PHONE
now they want free food, healthcare, and housing
(a) Hateful meme

(b) Non-hateful meme



Source	Model	Validation AUROC	Test AUROC
	Human	-	82.65
Hateful Memes Baseline	ViLBERT	71.13	70.45
	VisualBERT	70.60	71.33
	ViLBERT CC	70.07	70.03
	VisualBERT COCO	73.97	71.41
Vilio	VisualBERT	75.49	75.75
	OSCAR	77.16	77.30
	UNITER	77.75	78.65
	ERNIE-ViL Base	78.18	77.02
	ERNIE-ViL Large	78.76	80.59
	Ensemble	**81.56**	**82.52**

Table 1: Model performance.



12. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

Here are five ideas:

- *Re-pretraining with superior Bert models*
  - *Using Roberta, Albert with e.g. Uniter; Oscar & replicate should give a boost as HM is heavily text-dependent; I have all the code necessary for that already implemented I just did not have the resources to re-pretrain on e.g. COCO*
  - *model could then look at different parts of data.*
- *Grid features instead of boxes*
  - [*https://drive.google.com/file/d/1j9QE6xBq7Al_92ylmQEO4Ufq4f5n3Awa/view*](https://drive.google.com/file/d/1j9QE6xBq7Al_92ylmQEO4Ufq4f5n3Awa/view)
- *Using Transformers/Graph-like models directly without extraction*
  - *I think this has the most potential to improve existing approaches – However current hardware limits make the extraction more reasonable. I would bet though that in the future we will just feed in everything in one big graph!*
- *Integrate the OCR pipeline into the trainable model*
  - *In the Hateful Memes dataset meme captions were standardized, but in reality they may vary in font and size.This is useful information that may help the model determine hatefulness.*
- *Create an ERNIE-VisualBERT Model*
  - *Single-stream encoders, such as VisualBERT, seem to outperform dual-stream encoders, such as VilBERT,with the exception of ERNIE-ViL. ERNIE-ViL, which copies VilBERT's encoders, differentiates itself with changes like its Scene Graph Parser. Could an ERNIE-VisualBERT model with a single-stream encoder outperform the dual-stream ERNIE-ViL?*

13. Provide a link to the GitHub repository for your model and source code.

    [*https://github.com/Muennighoff/vilio*](https://github.com/Muennighoff/vilio)

14. Provide a link to your published academic explanation of your solution.

    [*https://arxiv.org/abs/2012.07788*](https://arxiv.org/abs/2012.07788)