Multimodal Hate Speech Detection in Memes

Degree's Thesis
Audiovisual Systems Engineering

Author: Benet Oriol Sàbat
Advisors: Xavier Giró-i-Nieto
Cristian Canton

Universitat Politècnica de Catalunya (UPC)
2019

Abstract

This thesis explores a multimodal approach to Hate Speech detection, involving vision and language (text). More specifically, we deal with the context of *memes*, a form of internet humour which will present additional challenges.

We first gather meme data from different sources. This way, we create a hate memes dataset for this task. Then, we use this data for the training and evaluation of statistical models, which are based on state-of-the art neural networks.

We study different ways to fine-tune pretrained descriptors for our specific task. We propose a way to add expert knowledge into the system and orient it into a real world issue-solving system. We also discuss ways to deal with the issue of reduced amount of data, experimenting with a self-supervised learning approach for pre-training.

We also compare the effect or contribution of each modality in the overall performance of the model.

Resum

Aquesta tesi exposa un enfocament multimodal a la detecció del Discurs d'Odi, fent servir imatge i llenguatge (text). Concretament, tractem amb el context dels *mems*, una forma d'humor present a internet, que presentarà reptes adicionals.

Primerament, recolectem dades de diferents fonts. D'aquesta manera, generem una base de dades de *mems d'odi* per aquesta tasca. Aleshores. fem servir aquestes dades per entrenar i evaluar models estadístics, que estaran basats en xarxes neuronals de l'estat de l'art.

Estudiem diferents formes de adaptar els descriptors d'imatge i text per a la nostra tasca específica. Proposem una manera d'afegir coneixement d'expert al sistema i orentar-ho cap a un sistema per a resoldre un problema real. També parlem de maneres de tractar amb el problema de disposar de poques dades i experimentem amb enfocaments d'aprenentatge auto-supevisat per un pre-entrenament.

També comparem l'efecte o contribució de cada modalitat sobre el rendiment del model.

Resumen

Esta tesis explora un enfoque multimodal para la deección automática del Discurso de Odio, usando vision y lenguage (texto). Concretamente, tratamos con el contexto de los *memes*, una forma de humor presente en Internet, que presenta desafios adicionales.

Primero, recojemos datos de distintas fuentes. De esta forma, vamos a generar una base de datos de *memes de odio* para esta tarea. Entonces, vamos a usar estos datos para entrenar y evaluar modelos estadísticos, que van a estar basados en redes neuronales del estado del arte.

Estudiamos distintas formas de adaptar descriptores de imagen y texto para nuestra tarea. Proponemos una forma de añadir conocimiento de experto al sistema y orientar-lo hacia resolver un problema real. También hablamos de maneras de tratar con el problema de disponer de pocos datos y esperimentamos con enfoques de aprenentage auto-supervisado para un pre-entreno de la red.

También comparamos la aportación o efecto de cada modalidad sobre el rendimento del modelo.

Acknowledgements

First of all, I want to thank my supervisor, Xavier Giro-i-Nieto, for guiding the project and advising me on what steps to take and helping me to tackle this unexplored topic.

I want to thank Cristian Canton for proposing the great idea of exploring this domain and giving helpful advice on the approach we took.

Thanks to Andrea Calafell for the template of the Document.

Revision history and approval record

Revision	Date	Purpose
0	15/06/2019	Document creation
1	21/06/2019	Document revision
2	23/06/2019	Document revision
3	25/06/2019	Document approval

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Benet Oriol Sàbat	benet.oriol.sabat@gmail.com
Xavier Giró i Nieto	xavier.giro@upc.edu
Cristian Canton	ccanton@fb.com

Written by:		Reviewed and approved by:	
Date	21/06/2019	**Date**	24/06/2019
Name	Benet Oriol	**Name**	Xavier Giró i Nieto
Position	Project Author	**Position**	Project Supervisor

Contents

Chapter 1

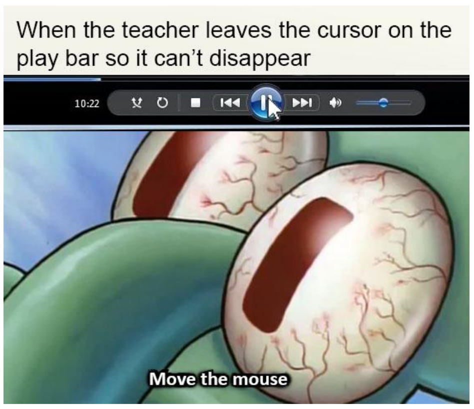
Introduction

1.1 Statement of purpose

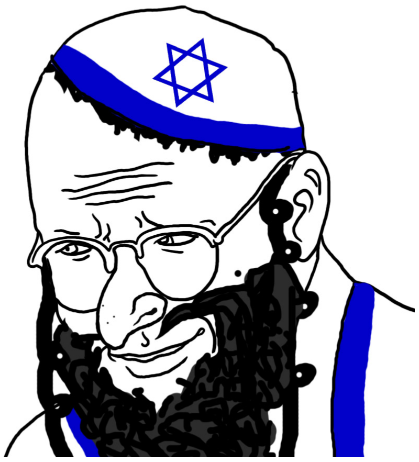
The quickly growing communication flux in blogs, social networks, messaging platforms and internet communication has lead to an also increasing amount of hate speech in different contexts and forms. On the other hand, this has collided with another phenomena: memes.

Hate speech does not have a formal definition, but it is often defined as speech that targets disadvantaged social groups in a manner that is potentially harmful to them (Jacob and Potters 2000 [18]).

Memes are a form of humorist content which is normally based on an image, and most of the time some sort of text such as a caption. They have gained a lot of popularity in the last few years and have been used in many different contexts and specially by young people. However, this format has also been used to produce hate speech in the form of dark humour. In figure 1.1 we can see two memes. (a) is a regular meme containing no harm and (b) is a clearly targeted humor against an ethnicity, in this case, the Jewish community.


When the teacher leaves the cursor on the play bar so it can't disappear
10:22
Move the mouse

(a)



(b)

Figure 1.1: In this figure we see in (a) a harmless meme and (b) a *hate meme*.

Social media companies are struggling to filter hate speech or objectionable content in general in their platforms. Their usual approach is using automatic or semi-automatic methods, so they can scale to the huge amount of content that is published every day, hour or second. For instance, automatic detection of nudity or sexual content is one of the usual tasks. Hate speech of text has been widely tackled too (we will talk about it in future sections). However, the multimodal (combined) analysis of image and text for hate speech has been a limitedly explored task, and can be used to tackle the hate memes problem.

In this context, our purpose is to create a tool to automatically predict hate speech on memes, using both image and text modalities. Also we will collect data and set a baseline for this task for further research.

Given the complexity of the problem, the main contributions of this thesis are:

- Training and evaluating a multimodal (image and text) statistical model for hate speech detection. We will collect hate memes data from different sources and tag them in a automatic way.
- Comparing the contribution of text and image in the classification of hate.
- Exploring ways to train a good model when there is a limited amount of labelled data. We will use classic regularization techniques such as data augmentation or dropout and we will experiment with self-supervised pre-training.

1.2 Requirements and specifications

Since it is the first time the task of multimodal hate speech is approached, we do not have any baseline to compare with.

That's why our requirements will be:

- Studying and measuring whether if multimodal hate speech in memes is possible and if it is, how good it works.
- Providing an accurate description of our system in case it has to to be reproduced and/or improved by other people researching or fighting multimodal hate speech. We will provide not only the models and implementation, but also the technique used to retrieve the data, and the data itself.

The specifications are the following

- Working with the Python Language.
- Working on Linux machines. Working with bash language.
- Using the *Pytorch* [28] API as the basic deep learning framework for development.

1.3 Methods and procedures

We have developed a system for multimodal hate speech detection. It uses pretrained embeddings for text and image, and we train a classifier with these embeddings as input features. We use an automatic system to get labelled data, based on searching images with certain keywords. We use these data to train and evaluate the model.

All the code has been written in Python, and has used Pytorch as main framework to implement neural networks. There are some utility scripts in bash. The implementation has been written from scratch except for the implementations of BERT and VGG-16 features [1].

1.4 Work Plan

We have used a work plan in order to keep track of the tasks and make sure we are having a correct distribution of time on the different tasks.

1.4.1 Work Packages

- WP 1: Initial Research.
- WP 2: Early Experiments
- WP 3: Data Collection
- WP 4: Improving the models
- WP 5: Documentation

1.4.2 Gantt Diagram

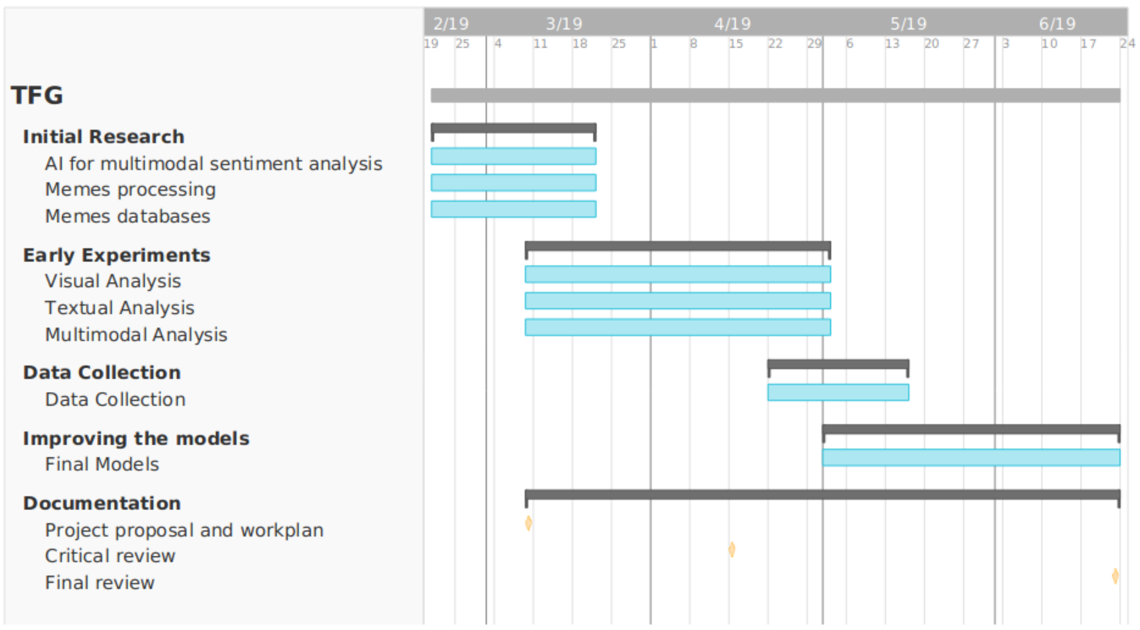


Figure 1.2: Gantt Diagram with the work packages and subtasks

[1]Further referenced

1.5 Incidents and Modification

Overall, we could be able to follow the planning we had in mind when we wrote the critical review. One of the major incidents has been the unexpected initial bad results of the text modality. We had to spend more time than expected trying to solve this issue. This is not a specific task in the work plan but it internally delayed the Work Package of improving the models

Chapter 2

State of the art

2.1 Hate speech Detection

Hate speech is a quite widely studied topic in the context of social science. This phenomena has been monitored, tracked, measured or quantified in a number of occasions ([36], [37], [26]). It appears in media such as newspapers or TV news, but one of the main focus of hate speech with very diverse targets has appeared in social networks (Silva, 2016 [27]). Twitter has been one of the popular study domains (Kwok, 2013 [20]; Malmasi 2017 [25])

However, we want to focus on automatically detecting this kind of content, rather that just monitoring it. Almost all the work in detection of hate speech has been made for text. The usual approach is to generate an embedding of some kind, using bag of words (Kwok, 2013 [20]) or N-gram features (Davidson 2017 [12]) and many times using expert knowledge for keywords. After that, the embedding will be fed in a Machine Learning system which is usually a SVM to predict whether it contains hate speech or not.

A very popular trend is to use an end-to-end Deep Learning approach in many different tasks of signal processing, and Hate Speech Detection is not an exception. We can see the use of LSTMs in *Deep learning for hate speech detection in tweets* (Badjatiya 2017[5]), CNNs in *Using Convolutional Neural Networks to Classify Hate-Speech* (Gambak 2017 [14]) on even a mix of both Convolutional and Recurrent networks (Zhang 2018 [38]). Most of these systems use some sort of word embedding such as the Neural Network-based word2vec [31]. Then they are fed into the corresponding architecture and classified by the model. Normally, these systems differ in small architecture and training hyperparameters, sometimes have different number of output classes and work on different domains (such as news or Twitter, for example).

As we said before, there is no work on detecting hate speech for image and text multimodal data. However, we think we should mention the work by Blandfort[8] in which they predicted gang violence using a multimodal approach of image and text. Their task was a bit different, but worked on a similar domain and approach. They aimed at detecting certain violent behaviour such as threads, insults or online harassment, as well as detection of potentially dangerous content such as drugs or guns in pictures. We took their approach as inspiration to start our own architecture. They extract features from both modalities using pretrained embeddings for text and image(for example word2vec or Fast R-CNN features), they merge both vectors and feed the multimodal features into a classifier.

2.2 Vision and Language Fusion

The world that surround us and the information that we receive has multiple modalities. We see images, hear audio, read text, ... and many times, we want to find relations between these data, use information from more than one at the time or maybe "translatate" from one modality to another, among other possibilities. Actually, *Multimodal Machine Learning:A Survey and Taxonomy* [6] talks about 5 kind of multimodal tasks or concepts which are:

- Representation. Making a single representation out of data from different modalities, including relevant information from both of them, which can be used for further tasks. Representing multiple modalities poses many difficulties: how to combine the data from heterogeneous sources; how to deal with different levels of noise; and how to handle missing data. The ability to represent data in a meaningful way is crucial to multimodal problems, and forms the backbone of any model.

  A very usual way to train a neural network for feature extraction is to train it for a certain task and then using intermediate hidden layers as the features for other tasks. This way, we can make the hypothesis that the network will have learned valuable representations of the data (Bengio 2013 [7]). For a multimodal representation, we can feed these vectors into another neural network that maps both into a joined representation or use it directly for prediction.

  One of the classic and more usual ways to perform an early multimodal descriptor out of different single-mode descriptors is by concatentation (D'mello 2015 [13]). However other ways are being explored, such as addition or multiplication.

- Translation. Mapping one kind of of data into the other kind. For example image description (text) translated into image (pixels) (Reed 2016 [30]). This can also be applied in the apposite direction and is called image captioning. Other popular tasks are video captioning or automatic speech recognition.

- Alignment. For example aligning the steps in a recipe with the frames a the video with the recipe.

- Fusion. Joining information from different modalities to perform a prediction. In this project, we are going to focus in this task, which is complementary with Representation. One of the possible multimodal combinations is video and its corresponding audio. That's because those two have an obvious correlation and contain complementary information that can be very useful for different tasks such as speech recognition (Rao 2013 [29]) or video classification (Lin 2002 [23]). Another important task is Visual Question Answering [2]. This one deals with vision and language modalities, and different archutectures have been proposed for this task(Hu 2017 [17]).

- Co-learning. Tranfering knowledge between modalities, representation and predictive models. For example, training a model for one modality and being able to adapt it for another modality.

Chapter 3

Methodology

3.1 Baseline

One of the goals of this project was to set a baseline for the task of hate speech detection. That's why we don't have a baseline specific for our work.

3.2 Dataset collection

3.2.1 Dataset for hate prediction

There is not a dataset for this task either, so we had to collect our own. We had a total of 5020 memes (images). They were automatically labelled assigning it a class depending on its source or how we retrieved the image.

The classes and subclasses are:

- Hate Memes. All the memes were retrieved from Google Images. We used the Google download tool[1] to automatically download hundreds of images. We used certatin keywords to search for them, and got a total of 1695 *hate memes*. The keywords we used and number of memes per class were:

  racist meme. 643 Memes.

  jew meme. 551 Memes.

  muslim meme. 501 Memes.

- Non-hate Memes. Gotten from the Reddit Memes Dataset [2]. We considered them non-hate memes, since we made the assumption that average Reddit memes do not or should not contain language that targets and harms specific minorities. This class contains 3325 memes.

As we can see, the two classes are balanced with a relation of 66% - 34%. In a binary classification problem, the minimum accuary will be the percentage of examples of the class with more examples. That's why we will refer to this 66% as the dataset's accuracy lower bound.

We also observed that this method of collecting data might result in a noisy dataset. However, we decided to stay with this methodology, since we don't have access to manually labelled data.

[1]https://github.com/hardikvasa/google-images-download
[2]https://www.kaggle.com/sayangoswami/reddit-memes-dataset

3.2.2 Unlabelled Dataset

One of the issues of Deep Learning is the need of big amounts of data. Ideally, we would have a huge dataset of labelled memes and maybe we could just train in on them and get great results. However, this is not the case, and we need to think of alternatives to train a good model with small amount of data. This is also a motivation for the regularization techniques we used. See Section 4.3.

A popular trend is using non-labeled data to train models. This has been used in different domains such as translation (Artetxe 2017 [4]), or image generation (Goodfellow 2015 [15]).

Also, you can use the unsupervised approach, for some kind of pretraining for your task, and then fine-tune it with your labelled data. An option for unsupervised mulimodal pretraining is the following:

First you need to think up a task that: a) Can be solved using your architecture and inputs. b) There has to be a way to automatically generate labels for this task. In our case, we thought of the task of binary classifying whether the input text and the input image correspond to the same meme or not. This has been proved to get good quality multimodal representations, but in the context of video and its corresponding audio (Arandjelovic 2017 [3]).

In other, words, we have a bunch of unlabelled memes, and feed to the network images and texts and the model has to guess whether they are from the same original meme or not. The idea behind this is that the network is going to have to learn some sort of feature about the text and image and their relation, which should be useful for the other task too. Or at least, it should start getting some features that will lead into a faster convergence of the algorithm when we fine-tune it.

We will introduce the *unlabelled dataset*. This means we downloaded data from a source where we don't know whether it contains hate speech or not (actually, this aspect is just like in the Reddit Memes Dataset). We downloaded two meme packs from [3] and [4]. This way we got a total of 1586 memes. Maybe this amount of memes doesn't fully exploit the advantatges of unsupervised training, which is the huge amount of unlabelled memes available. However, we decided it was good enough to make a few tests and see whether this could be helpful. We will detail the use of this dataset in section 4.6.1.

3.3 System architecture

The overall system consists of an image or meme as input and a hate score output. The first step of the process is extracting the text of the image with Optical Character Recognition (OCR). After some preprocessing, we will encode both image and text into independent feature vectors. Then we conatenate both vectors into a single one and feed it into a fully connected feed-forward neural network that will predict the hate score. In the remain of the section we will explain in detail all the different modules of the system.

[3]https://archive.org/details/mega-dank_memes
[4]https://archive.org/details/HugeMemePack

3.3.1 OCR - Pytesseract

The first step is to extract the text from the image. This task is called Optical Character Recognition (OCR) and there are several implented systems that detect text in an image and extract it. One of the most popular and usable is Tesseract[5]. We will treat this as a text detector black box, but we want to mention that this library actually uses state-of-the art Deep Learning algorithms to extract the text. We used the pytesseract Python wrapper[6].

3.3.2 Text embedding - BERT

The text detected by the OCR is encoded in a BERT (Bevlin 2018 [10]) embedding, a state of the art model that has been proved to perform very well in different tasks. This model turns a (sub)word sequence into a 768-dimentional feature vector. It is basically a transformer architecture and is pre-trained in a self-supervised way, which is a kind of encoder-decoder architecture.

Encoder-decoder architectures have been used as the top performing models for sequence to sequence tasks such as machine translation.

The encoder maps an input sequence of symbol representations $(x_1, ..., x_n)$ to a sequence of continuous representations $z = (z_1, ..., z_n)$. Given z, the decoder then generates an outputsequence $(y_1, ..., y_m)$ of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.

LSTMs (Schmidhuber 2005 [16]) have been used for a long time to model both encoder and decoder. However, the state-of-the art is currently the Transformer model (Vaswani, 2017 [35]). " The Transformer follows this [encoder-decoder] architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of [Figure 3.1], respectively. "

We used a pytorch implementation available at the repo below [7]. This implementations includes multiple pretrained versions and we chose the one called bert-base-multilingual-cased. This version has 12 layers, 768 hidden dimensions, 12 attention heads with a total of 110M parameters and is trained on 104 languages.

3.3.3 Image embedding - VGG16

VGG are (Simonyan 2014 [32]) are a set of image classification architectures proposed for the first time by the Visual Geometry Group in the university of Oxford. The architectures in the VGG-N models differ in the number on layers - which is N - but have a very similar architecture. All of them have convolutional layers at the begginning and end up with a fully connected network for classification.

Deep convolutional neural networks (CNNs) are a set of function models based on an alternate stacking of convolutions and non-linear activations.

[5]https://github.com/tesseract-ocr/tesseract
[6]https://pypi.org/project/pytesseract/
[7]https://github.com/huggingface/pytorch-pretrained-BERT

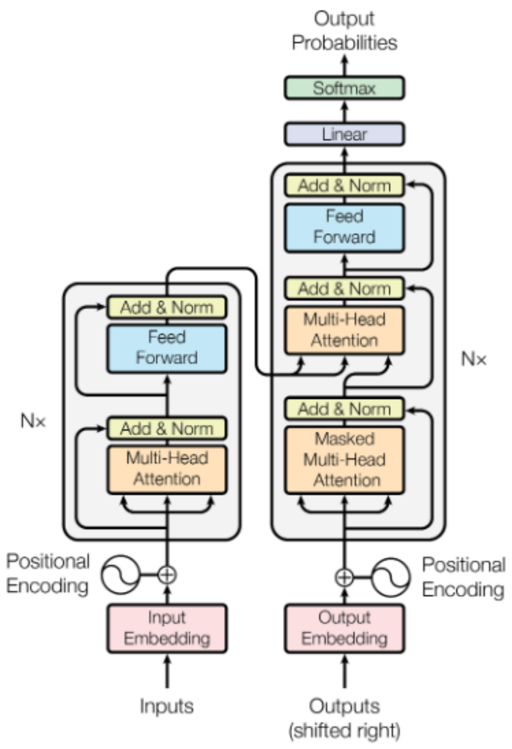


Figure 3.1: Transformer architecture.

In other words, they are a type of Deep Neural Networks which use the convolution as the linear operation. They often also use additional operations such as max-pooling to reduce the size of the activations along layers. Normally, after some convolutional layers, the activation is flattened into a one-dimensional layer, which is considered a feature vector, and used for further tasks such as classification using any sort of model. We can see an example of a CNN architecture in Figure 3.2.

These convolutions often operate with small kernels and therefore have the advantage of having many less parameters than a regular fully-connected network. They are specially good for images or any kind of data with exploitable spatial information. They are used in many different tasks such as image classification (Hinton, 2012 [19]), Face Recognition (Lawrence, 1997 [21]) or even audio synthesis (Vinyals, 2016 [34]).

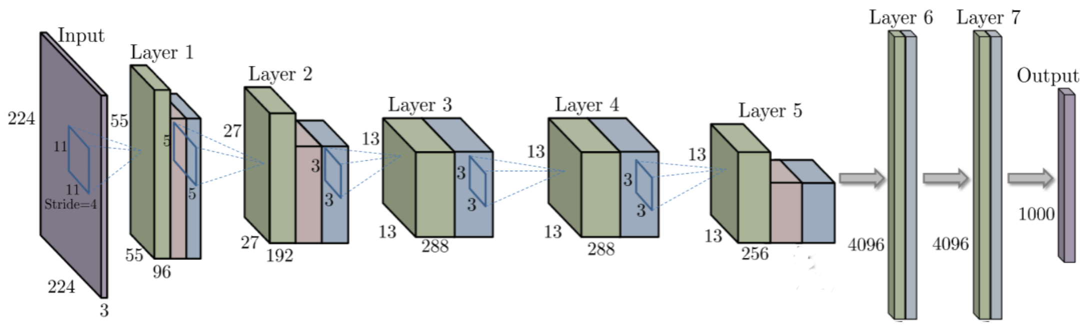


Figure 3.2: Convolutional Neural Network architecture. Purple represents the input 3 channel matrix and the output vector. Each block is a layer, consisting of the vectors after convolution, max pooling and non-liner activation in green, red and blue, respectively.

Once we defined an architecture, we still have to find the all model's parameters that actually model or fit the function of, for example, classification. The way this is achieved is defining an objective loss function that depends on the prediction and the ground truth target output and minimizing it (with respect to the parameters) through some gradient descent algorithm, therefore optimizing the parameters in the model. The way we compute gradients with respect to the loss function is through backpropagation [22], which is basically the chain rule. We do this because each layer of the network is mathematically a function composition (the input is the output of the previous layer).

These models are often computationally expensive to train and require a lot of data. They existed for decades, but we couldn't easily use them until a few years ago, when we had more powerful GPUs, more efficient algorithms and huge amounts of data available to actually be able to train them.

We used the VGG-16 architecture, pre-trained on the ImageNet classification dataset (Deng 2009 [9]). We obtained the pretrained model from the Torchvision module in Pytorch, which makes it really easy to use these pretrained models. Then we used the activations of a hidden layer as feature vectors for the image, which has been proved to provide relevant information of the image, not only for ImageNet classification (for which it was trained), but it can also be used for other tasks such as visual tracking (Ma, 2015 [24]). Specifically, we used the penultimate layer (last hidden), which has 4096 dimensions.

3.3.4 Fusion and the classifier

The text and image encodings must be combined to infer classification or make a prediction of hate content. The fusion method we will use is concatenation of vectors.

This way we end up with a feature vector of 4894 dimensions. This feature vector is fed as input into a fully connected feed-forward network with a one-dimensional prediction output. In the next figure we can see an overall scheme of the system.

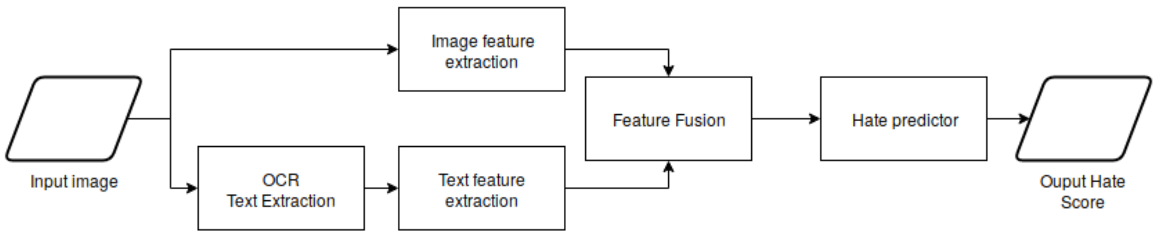


Figure 3.3: Overall scheme of the system

We tested different architectures for this module and finally opted for 2 hidden layers of size 50. For the activation function, we used ReLU in all the layers except for the last one, which has no activation function. We also set dropout on the first hidden layer with a probability of 0.2. The architectures we considered differed in size and dropout. We will justify these justify these hyperparameters in chapter 4.3.

Chapter 4

Experiments and Results

This chapter presents the different experiments we made, which will be changes in the architecture and hyperparameters, and other changes or ideas in the training algorithm.

4.1 Experimental Setup

4.1.1 Implementation - Setup and practices

The implementation of the system was developed in Python due to it's huge popularity in Deep Learning systems. It is also a very user-friendly and high-level language that's very easy to understand and is flexible for many different domains.

For the implementation of deep neural networks, we used Pytorch as the main framework. Pytorch is a library that implements automatic differentiation and this allows us to compute the gradient very easily for optimization. It also includes many utilities for the training process.

The code was developed using the PyCharm IDE and deployed on the GPI [1] computational cluster. The exeperiments were performed on different GPU Models, but no experiment has exceeded a GPU Memory usage of 8GB. All the code is available at a gihub repository [2]. We will also publish installation instructions in this repo, as well as scripts to download the data.

The code has been written using good Python practices an keeping in mind our will to make it easy to replicate the experiments, or even further developing the system.

Most of the training curves have been drawn using Tensorboard [3] and TensorboardX [4], a Tensordboard wrapper for Pytorch (it is originally for Tensorflow[5]). Note that this library draws the real curve in a soft color and a smoothed version in a stronger color. This way we can get rid of some of the variable's variance for the sake of visualization. For some curves also we used Matplotlib [6].

4.1.2 Hate class encoding

For all the experiments, the output was general hate prediction. In other words, we do not try to classify the images in the different hate subclasses (jew, muslim, racist) we described in the Section 3.2, but we just binary classify the presence of hate speech in the dataset. We did

[1]Grup de Processament d'Imatge, ETSETB, UPC
[2]https://github.com/benoriol/memes_processing
[3]https://github.com/tensorflow/tensorboard
[4]https://github.com/lanpa/tensorboardX
[5]https://github.com/tensorflow/tensorflow
[6]https://github.com/matplotlib/matplotlib

this to keep the task simple and because at this point we do not need to be able to distinguish between different hate speech targets. We encoded the hate label as a 1.0 for hate and 0.0 for non-hate memes.

4.1.3 Preprocessing

All images and text are preprocessed in a certain way for the model. The VGG-16 takes as inputs images of 224*224 RGB pixels (total of 224*224*3). Every input image was resized by using cubic interpolation.

We previously extract and store the text for all the images in the database because it has a high computational cost (0.5s per image in average) and would drastically slow down the training process (+1700%).

BERT takes as input a BERT Token sequence. The pytorch implementation of BERT we used also provides the tokenizers. Every token represents a sub-word that is normally made of a few letters. So we have a dictionary of N tokens and split the input text into units of tokens. Then, we limit the maximum length to 50 subwords or pad shorter sequences to this length. The reason we do this is because every sequence inside a batch needs to have the same lenght (even though not between batches). Finally we feed this sequence of BERT tokens into the BERT model in order to get the fixed-length embedding.

4.1.4 Dataset Split

In machine learning, it is an obligatory practice to split the dataset into at least two subsets. One for training and the other to test how good your model does on data that has never seen label or how good it generalizes. Sometimes three subdatasets are used: train, validation and test. In summary, *train* is used to optimize the model's parameters, *validation* to optimize the hyperparameters of the system (often by exhaustive search) and *test*, to actually assess performance on completely unseen data.

In our case, we did not use three subsets, but two. That is because we had a very small amount of labelled examples, and we thought it was better to just use as much as we could to train and get a good validation set to get trustworthy metrics.

4.1.5 Evaluation metric

The evaluation metric is the binary accuracy:

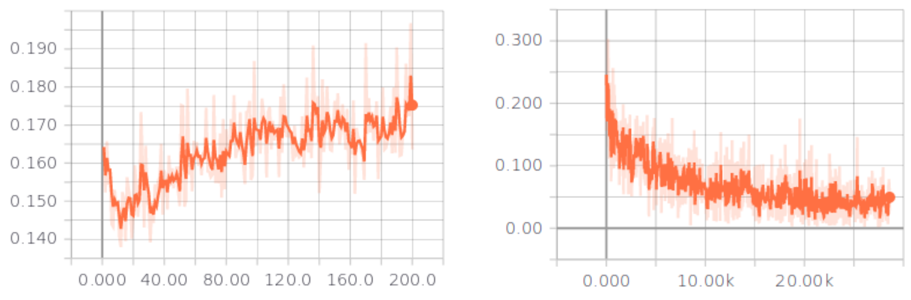
$$acc(y, \hat{y}) = \begin{cases} 1, & \text{if } |y - \hat{y}| < 0.5 \\ 0, & \text{otherwise} \end{cases}$$

The overall accuracy of the validation would be the average of all the examples in this set. Note that it is performed only on the validation set and is never used as a loss function to train the model. It is only a validation metric and, moreover, it is not differenciable, so we would not be able to backpropagate the gradient. Sometimes the loss function (MSE) will be also used as a metric to understand certain experiments.

4.2 Early Experiments - Baseline

We started with a first set of experiments with the following specifications:

- Text Descriptors. VGG16 and BERT. Frozen, meaning their parameters are not updated or optimized at training time, we keep the ones that came from the pretrained models we downloaded. Total of 4864 dimensions.
- Classifier: A Multi-Layer Perceptron (MLP) with two Hidden Layers, Hidden size = 100. As we mentioned, hidden layers are the layers in a network that aren't input or output. In other words, our classifier will have four layers. Input (size 4864), 2 hidden (size 100) and an output layer(size 1).
- Optimizer: SGD with momentum. Learining rate = 0.01, momentum = 0.9.
- Batch size = 30. This is the number of examples of the dataset that we will use at each timestep of the SGD - Momentum algorithm. We set this parameter as high as we could, with the restriction that we have limited GPU memory to load all the models, inputs, gradients and activations at the same time.
- Loss function: Mean Squared Error (MSE).

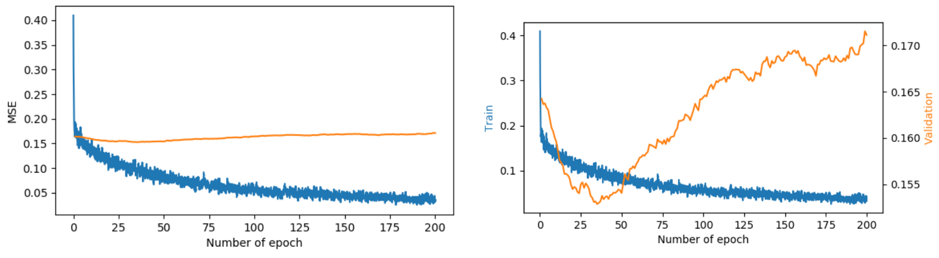


(a) (b)

Figure 4.1: In this figure we observe in (a) the validation Accuracy and in (b) the train loss. Validation variables are with respect to the epoch number and train loss is with respect to the iteration or batch number

In figure 4.1 we can see the evolution of the train and validation metrics. This achieved a maximum accuracy of 82.6% in validation at epoch #30. We can compare this to the dataset balance of 66% - 34% which is the baseline accuracy. However, we noticed that there was a tendency of overfitting from this point onwards. Overfitting means the model has optimized the parameters to perform very well on the training data, but in a way that does not generalize to useen (validation) data. It is often compared to "memorizing the training data" in contrast of "understanding the training data". We can observe this by taking a look at the validation loss curve in figure 4.2, and comparing it to the train loss curve. We see that both losses are decreasing, down to a point where the train MSE keeps decreasing and the validation MSE starts growing back. That's a clear synthom for overfitting.

That is the baseline we started from and the following experiments aim at improving these.



(a) (b)

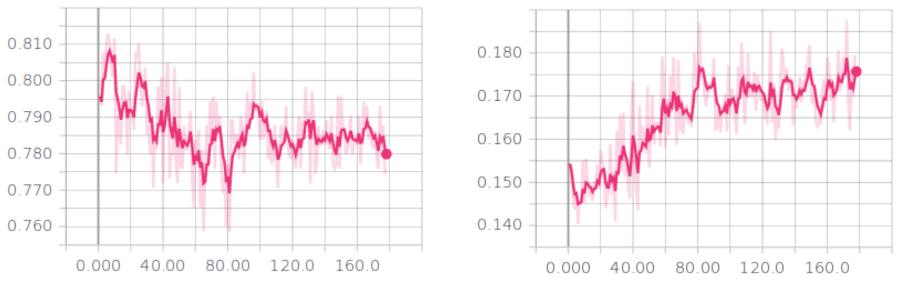
Figure 4.2: We can see both train (blue) and validation (orange) MSE curves at training time. The only difference in both figures is the scale of the validation MSE, which we changed to be able to visually recognize overfitting

4.3 Regularizarized Fusion MLP

Regularizarion techniques refer to a set of practices or "tricks" in order to reduce overfitting and make the model better at generalizing. We introduce some of the most usual techniques:

- Data augmentation. In our context, data augmentation means automatically generating (more) labelled data from other labelled data. For a better explanation, let's use our case as an example, were we used the cropping function to perform data augmentation. The assumption is that if we crop a relatively big part of the image, the cropped part is going to be an image of the same class. Then we use the cropped image as if it was another image labelled as the same class and use it to train the model. This way, out of a single picture, we could generate as many labelled images as possible croppings of a relatively big size. Even though the cropped image will look very similar to the original one, this method is still good to give some variety to the dataset instead of doing multiple epochs with the exact same data.

  The way we applied this is: Instead of resizing it directly to 224x224 as we mentioned in Section 4.1.3, we will resize it to 256x256 and then randomly cropping it into a 224x224 image. We only used augementation at training time. We used different random cropping for every image in every epoch. The results were the ones we see in figure 4.3. There, we see validation accuray and loss. We can see we did not get any improvements using this technique, so we did not continue using it from now on.



(a) Accuracy (b) MSE

Figure 4.3: In (a) we see the validation accuracy and in (b) the validation MSE

- Reducing the capacity. The capacity of the architecture is the variety or complexity of the functions it can fit or model. This is often a cause for overfitting. We reduced the hidden size of the classifier in order to reduce the capacity and we obtained the results in figure 4.4.

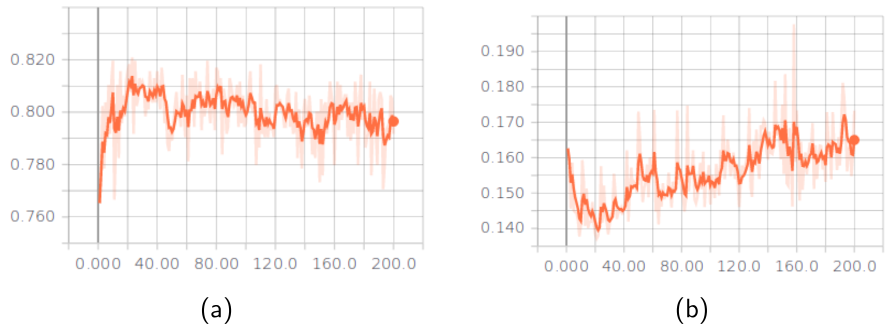


Figure 4.4: In (a) we see the validation accuracy and in (b) the validation MSE

As we can see, we improved a little bit the maximum validation accuracy, up to 82.0% at epoch 24. However, if we take a look at the MSE evolution, we see that we still had clear overfitting.

- Dropout. Dropout means randomly dropping to 0 the activation of certain neurons. This forces the model to learn less complex mappings and can be seen as a very efficient way of averaging models (Hinton 2015 [33]). We only used drop out at training time. At the beggining, we set dropout to the whole classifier's hidden neurons with a probability of 0.5. We obtained the esults depiced in Figure 4.5

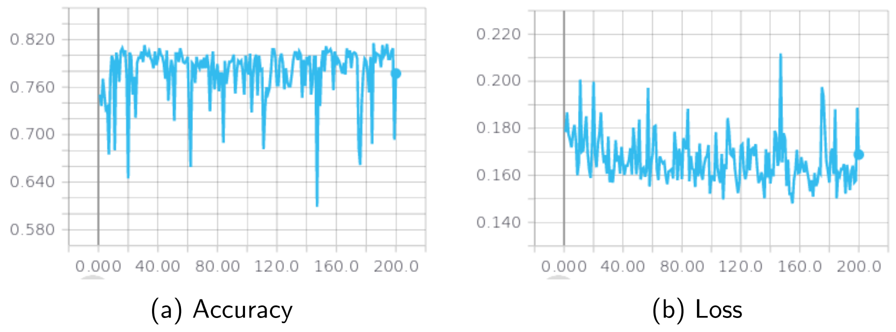
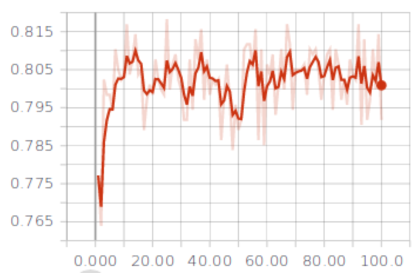


Figure 4.5: In (a) we see the validation accuracy and in (b) the validation MSE

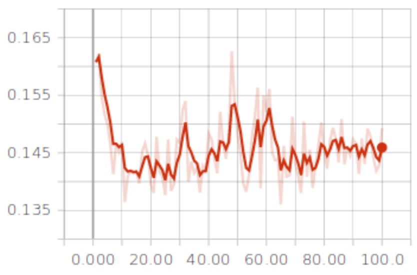
As we can see, there is strange phenomena appearing. Periodically, the accuracy drops approximately to its lower bound. Our hypothesis about this observation was that the amount of dropout we were introducing was so high that sometimes we could get really noisy gradients that can bring parameters to non non-optimal ones and mess up the whole training of the model.

The way we addressed this was reducing the amount of dropout. Also, since we had the previous result of improving the accuracy by reducing the capacity we also started using hidden layer of size 50 instead of 100 from this point onward. After testing a few parameters of where should we put dropout and in what amount, we ended up using Dropout on the first layer of the classifier with a probability of 0.2. The result was the one in Figure 4.6:

As we can see, the validation MSE curve 4.5 (b) doesn't start going up after a few epochs as before. On the other hand, validation accuracy goes down back to 81%. We want to



(a)



(b)

Figure 4.6: In (a) we see the validation accuracy and in (b) the validation MSE

Modalities	Validation Accuracy
Text only	66.0%
Image only	82.0%
Multimodal	82.0%

Table 4.1: Results of multimodal and monomondal experiments

point out that our priority will be to get a stable model that doesn't overfit fast, even if we lose a bit of overall accuracy.

From now on all the experiments will be done with a dropout probability of 0.2 on the first layer of the classifier and a classifier with 2 hidden layers of size 50.

4.4 Multimodal fusion

One of the interesting aspect of the multimodal analysis is also being able to measure and analyze how much does each modality contribute to the overall of solving the task. We trained a model using text only and a model using image only, and the results are the ones in Table 4.1.

This was an unexpected result. First of all, we did not expect that the image-only model would work so well. We were surprised by the correlation the classifier found between the VGG-16 visual features and the hate class. Also we didn't expect the text-model to work so poorly, and our next steps were taken towards improving the text-only model and the text contribution to the multimodal model. The image model performed up to 82.0% and text had an accuracy of 66.0%

Another interesting aspect is comparing how the MSE in training evolves. While multimodal and image approaches are able to converge, we see that the text gets stuck at a certain loss value, it just does not learn. This is a symptom of the model not being able to learn anything. Two other symptoms that we are not getting any useful information from text are:

- The image-only accuracy is very similar or equal to the multimodal accuracy. Hence the text is information is not adding valuable information.
- The text-only has an accuracy approximate to the lower bound, 66.0%. Note that a *dumb system* predicting always non-hate (most present class in the dataset), would get the same accuracy, that's why we call it the lower bound.

We can see it depicted in Figure 4.7

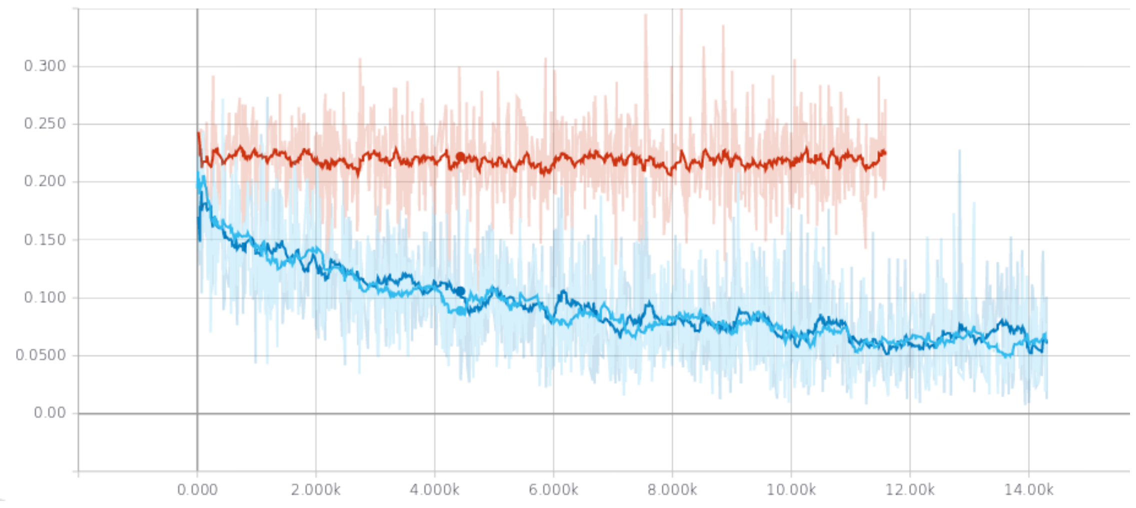


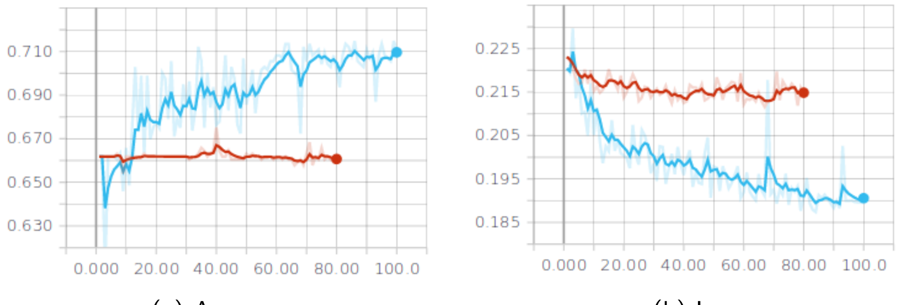
Figure 4.7: MSE of the training dataset for each batch iteration. In dark blue, the multimodal analysis; in light blue, the image-only model; In garnet the text-only model

4.5 Fine-tuning BERT and VGG-16

Not being able to reduce the training loss is a clear symptom that you are trying to optimize a model that will never be able of solving the task. That can be for different reasons:

- A first option is that there is not enough correlation between the text in the input and the hate score. We will discard that by now.
- The second one is that the model doesn't have enough capacity to fit the mapping. What this means is that the function that maps text features into hate score is too complex to be modeled by our architecture. We will discard that too, because with the small amount of data we have, and the capacity that DNNs have, we think the model should be able to, at least, memorize or overfit into our data. And this is not happenning, since we see that the train MSE doesn't go down.
- Right now, we are training a hate speech classifier using BERT as a task-agnostic text descriptor. This version of BERT was pretrained for general domain use. However, it has been proved that BERT without fine-tuning for your task has much worse representations than with fine-tuning (Devlin 2018 [11]). Given that our domain is very specific and text has been extracted with OCR (duvious quality), we think that BERT might be giving representations that are uncorrelated with Hate Speech. We want to aim at improving this part by fine-tuning BERT for our task, unfreezing its weights. Unfreeze means computing gradients and updating the weights at train time, therefore finding optimal values for the text embedders, specific to our task, not only the weights of the classifier.

  From this point onwards, we will be using Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. This is the recommended algorithm to fine-tune Adam [11]. In figure 4.8 we present the experiments of fine-tuning BERT that show how not only we were able to reduce the Loss MSE for the text only model (not present in figure 4.8) but we were able to reduce significantly the validation loss too, and we can see the validation accuracy is at around 71%, better than the old 66%, which is close to the dataset's lower bound.



(a) Accuracy (b) Loss

Figure 4.8: In blue, the evolution with BERT unfrozen and in garnet BERT is frozen. In (a) we see the validation accuracy and in (b) the validation MSE

Following the same approach, we decided to apply fine-tuning to image and text combined, and for both descriptors. In the case of VGG-16, we decided to unfreeze only the fully connected classifer [7]. It is a common practice to freeze the convolutional layers (for example Kaoutar 2018[1], since they are consiered the lower-level features and more common to different domains, therefore it's not so important to update them. The result is in figure 4.8:

As we can see, we are improving the original model. We get a top accuracy of 83% in contrast to the 82% we had before.

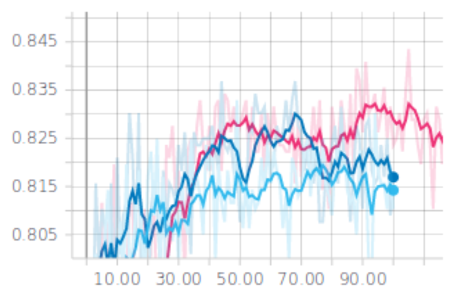
4.5.1 Progressive Fine-Tuning

Now we present another idea to further experiment with this concept. VGG and BERT were pretrained and now have optimized weights. However, when we fine-tune them for our task, we start with a very bad (randomly initialized) classifier that can get us very large loss functions, therefore big and noisy gradients, that can make BERT and VGG lose quality of the pretrained representations. In order to solve this, we wait a few epochs before unfreezing BERT and VGG. This way we will be getting gradients from more high-quality predictions and can fine-tune our descriptors better.

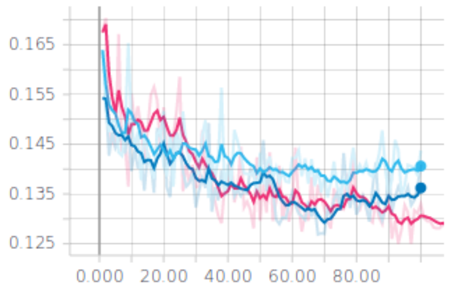
In figure 4.9 we can see the results from unfreezing at epoch 0 (the previous experiment), and doing so at epoch 10 or 50. Unfreezing at epoch 10, we got an accuracy up to 83.7% The idea behind giving it 10 epoch was to not start giving it completely random gradients that would damage the pretrained embeddings. However, we see that if we let the model converge further (at epoch 50) before unfreezing the descriptors, the results are even better, reaching an accuracy up to 84.3%, which is the best one so far.

Table 4.2 summarizes the results of multimodal experiments.

[7]Here, we are not referring to the hate classifer in our system, but to the fully connected top layers in the VGG-16 architecture, which were used to solve classification for ImageNet during the pretraining



(a)



(b)

Figure 4.9: In blue, the evolution with BERT and VGG classifier unfrozen is frozen. In light blue, we unfreeze the models in epoch 10 and in li In (a) we see the validation accuracy and in (b) the validation MSE

Hidden size	Dropout	Data augmentation	Unfreeze encoders (at epoch)	Accuracy
100	-	No	-	82.6%
100	-	Yes	-	81.2%
50	-	No	-	82.0%
50	0.5 - All HL	No	-	81.0%
50	0.2 - First HL	No	-	81.7%
50	0.2 - First HL	No	0	83%
50	0.2 - First HL	No	10	83.7%
50	0.2 - First HL	No	50	**84.3%**

Table 4.2: Summary of multimodal experiments. HL stands for the MLP's Hidden Layers

4.6 Failed experiments

4.6.1 Self-supervised pre-training

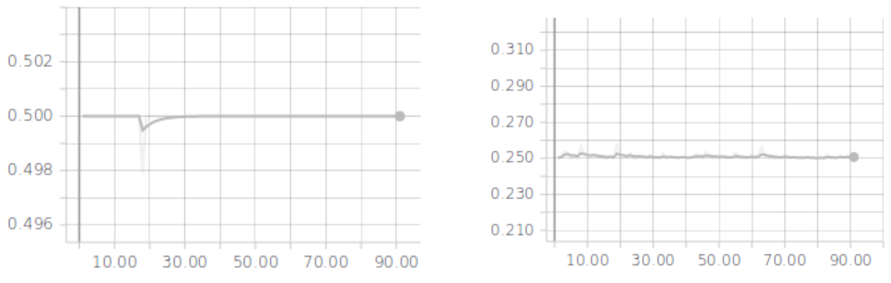
In section 3.2.2 we described the self-supervised task of image-text matching. We made an experiment of this idea. BERT and the top layers of VGG-16 were unfrozen during pretraining. We can see the training metrics in figure 4.10:

As we can see, the results for training of this task are very bad. We have a validation accuracy of 50% (lower bound in binary classification) and a MSE of 0.25 in both train and validation (upper bound). So, in other words, the model wasn't able to learn anything, therefore our pretraining is going to be useless.

4.6.2 Introducing expert knowledge

One of the concepts behind Machine Learning is that computers can learn how to perform certain algorithms without an expert having to tell them. Very often, this makes the algorithm even better than if it was had-crafted by an expert in the field, and it doesn't require an expert.

On the other hand, if we are aiming at solving a real-world problem, we find ourselves is situations in which, for example, we do not have enough data or the task is too difficult. In



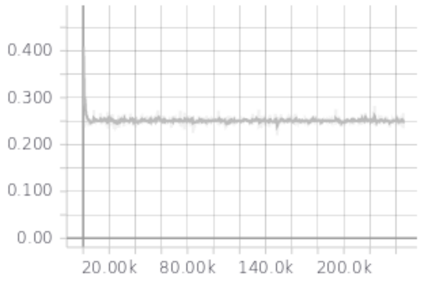


Figure 4.10: Training metrics for the unsupervised pre-training task

our case, we don't think it's a good idea to restrict the system to a pure ML approach. In this section we propose a way of introducing expert knowledge into the model in order to achieve better performance. We set a list of *hate words* that could be potentially correlated with the presence of hate speech. Then we one-hot encode the presence of each word in the text and concatenate the encoded vector along with the VGG-16 and BERT descriptors. Since we are not experts in the field of hate speech, we just gave a list of a few words we thought that could be correlated with hate speech, as a mean of testing this idea. The list of words contained 12 words such as *jew* or *muslim*. This was tested using Hidden size = 50, frozen embeddings and no Dropout.

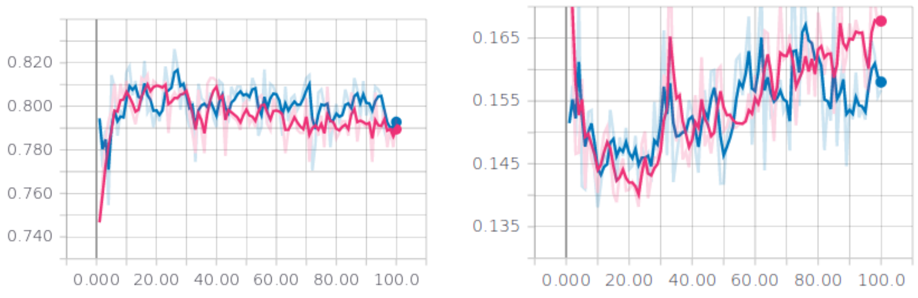


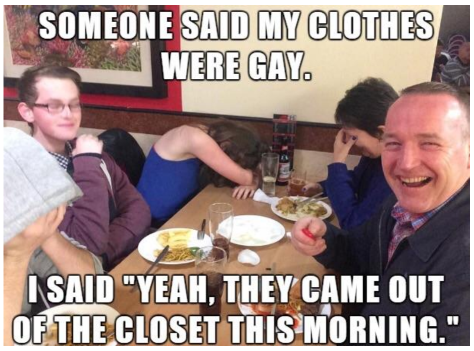
Figure 4.11: In blue the model without the expert knowledge and in pink the one with the expert knowledge coding. In (a) we see the validation accuracy and in (b) the validation MSE

In figure 4.11 we can see the model does not improve with respect to the model with out the expert knowledge. That might be because the hate word list has uncorrelated words, or because the way we introduced the information was bad.

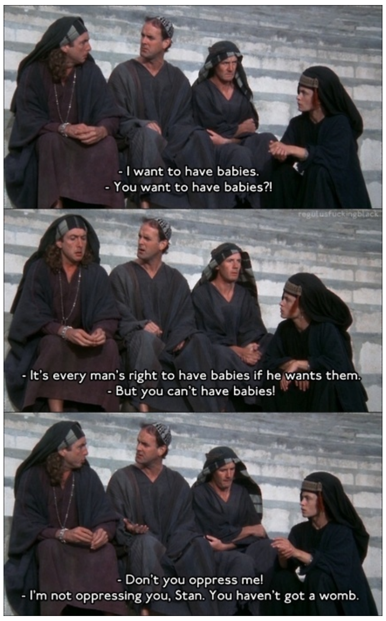
4.7 Qualitative Analysis

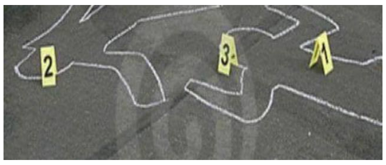
To end up with the results section, we will make an analysis of the predictions made by the model, looking at individual examples. In figure 4.12 we can see the top 4 validation predictions with more loss (best predictions) and in figure 4.13 the top 4 with less loss (worse predictions). We will make hypothesis for why did the model do a correct or incorrect prediction. For the worst predictions in figure 4.12, (b) and (d) are clearly tagged wrong, since they don't contain any hate. We will talk about it in section 6. About (c), maybe because there were people dressed as Jews, the algorithm visually detected the features related to jew memes, this would be a false-positive. We don't have an hypothesis for the prediction (a).

For the best predictions, (a) contain the words Jewish and Hebrew, which might have lead to a hate detection using text. For image (c), we can identify visually a Muslim. For image (b), we can see the word racist twice, and there is a black man (President Obama) in the picture, so maybe a combination of both features lead to a good prediction. (d) has no text or image containing hate speech or related features.


SOMEONE SAID MY CLOTHES WERE GAY.
I SAID "YEAH, THEY CAME OUT OF THE CLOSET THIS MORNING."
(a) Hate speech meme

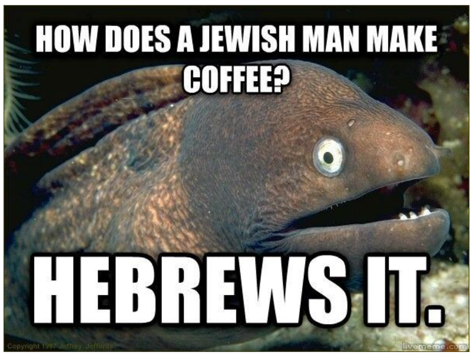
(b) Hate speech meme


- I want to have babies.
- You want to have babies?!
- It's every man's right to have babies if he wants them.
- But you can't have babies!
- Don't you oppress me!
- I'm not oppressing you, Stan. You haven't got a womb.
(c) Neutral meme

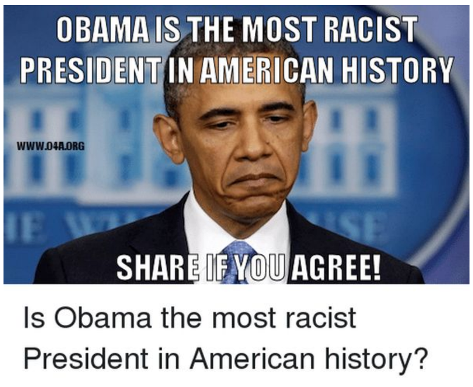


(d) Hate speech meme

Figure 4.12: Top 4 less confident prediction

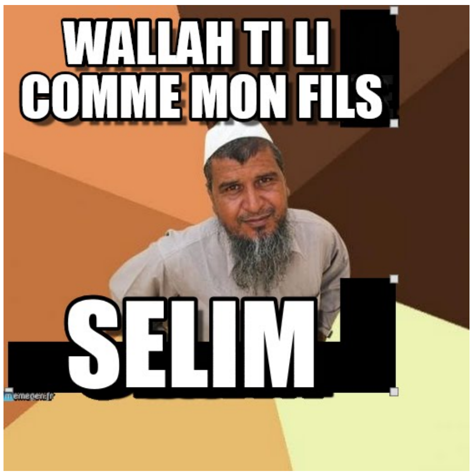


(a) Hate speech meme

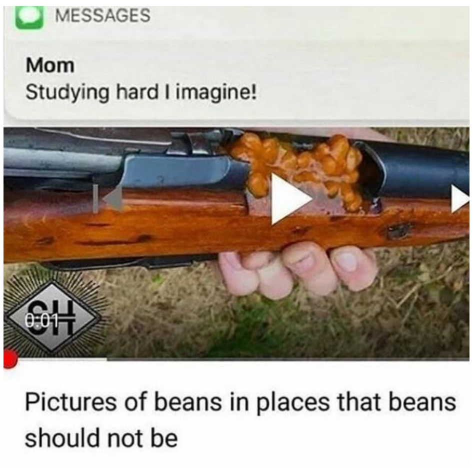


Is Obama the most racist President in American history?

(b) Hate speech meme



(c) Hate speech meme



Mom
Studying hard I imagine!

Pictures of beans in places that beans should not be

(d) Neutral meme

Figure 4.13: Top 4 more confident predictions

Chapter 5

Budget

The costs will be calculated based on the salary of the people involved in research. We will assume a salary of 8€/h for the student, since it is the standard internship salary and we will estimate a salary of 20€/h for the supervisors, with a salary of experienced engineer.

We will not take into account the cost of the salaries of the System Administrators in GPI. The cost of computation has been estimated with respect to the Google Cloud service pricings **??**, with the GPU model NVIDIA Tesla T4 **??**

	Amount	**Cost/hour**	**Time**	**Total**
Junior engineer	1	8,00 €/h	25 h/week	5,000 €
Senior engineer	2	20,00 €/h	2 h/week	4,000 €
Google Cloud computation	1	0.95 €/h	150 h total	142.5 €
			Total	5,942.5 €

Table 5.1: Budget of the project

Chapter 6

Future work

There are pros and cons of researching unexplored areas. On the positive side, we have a lot to explore and can easily work on something scientifically valuable. On the other hand, we have spent a lot of our time to establish a baseline and did not have enough time to make all the improvements we wanted. In this section we will talk about some ideas to improve the system.

6.1 Dataset

Atfer a few observations, we have seen that the dataset has poor annotations. There was not any manually labelled image and the method to collect data did not turn out so good as we expected. We did not find almost any hate meme in the Reddit Memes Dataset, but we actually downloaded many non-hate memes or images that were not even memes from the Google image search queries, which we labelled as hate memes. The quality of the annotations is important, but even more if the amount of data is so small. We should clean the dataset before proceeding with further research.

Also, from the result that the image-only model worked so well, we have the suspicion that there is some sort of image bias (uncorrelated to hate) that has been used by the classifier to predict the class. However, we did not visually find any obvious bias in the dataset.

6.2 Descriptors

In terms of the descriptors, we can explore other fusion methods like late fusion. The idea behind this is to fuse the embedding from both modalities after a few transformation (or layers) rather than directly after the pretrained embedding.

Another idea was to train at the same time multimodal and unimodal classifiers. This way we would be forcing the model to learn how to predict the class from both modalities, thus creating quality representations, and at the same time creating a multimodal representation from the unimode ones. Also, the one-hot encoding of the presence of certain keywords should work better, and might be because we are not fusing it the right way with the rest of the data.

Also, we should investigate more on different ways to code the text. It would be obvious that text should be one of the main components of hate speech and we are not getting a good classification out of it. We could experiment with the new XLNet model [1] , released June 2019, which has been proved to outperform BERT.

Also, we should test the quality of the OCR extractions, since it might be really difficult to extract the text from such noisy images.

[1]https://arxiv.org/abs/1906.08237

Chapter 7

Conclusions

In this project we have proposed a baseline approach for multimodal hate speech detection in memes. We used pretrained image and text embeddings and trained a hate predictor using those features.

We proved that it is possible to get an accuracy of up to 84% for this task, even though it is the domain of a noisy dataset. We observed the importance of the visual features in the detection and were surprised by the bad results with text features.

We studied different ways to provide quality features and train a network when a large amount of data. These techniques include regularization techniques such as dropout or data augmentation. In order to deal with this issue, we expored the unsupervised pretraining using the text-image mathcing task. We saw this task was too difficult for the model, and we were not able to pre-train the model for it. We also studied how to fine-tune BERT and VGG for domain-specific feature extraction.

As we commented on Chapter 6, the quality of the annotations is very bad, and this has probably been one of the main limitation of the project and reasons to get much worse training and evaluation of the model.

We also saw that the expert knowledge we introduced didn't help for the prediction. However, we think there should be ways to improve the system using expert knowledge, specially if this idea has to be put to real life use.

7.1 Challenges

We have seen that humour is not an easy domain to work for computational analysis. It in one of the hardest and most unexplored parts of audiovisual analysis and it requires a very high level understanding of the image and text, and sometimes even understanding the social or cultural context. Also, memes come in a large variety of forms and can be very noisy data.

Also, a challenging part of the project was implementing the whole system from scratch and collecting the data. Having no starting point has made it very time consuming to build a whole pipeline, and we have had to deal with many bugs at the implementation of the system

Bibliography

[1] Kaoutar Ben Ahmed and Ahmad Babaeian Jelodar. Fine-tuning VGG neural network for fine-grained state recognition of food images. *CoRR*, abs/1809.09529, 2018.

[2] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[3] Relja Arandjelovic and Andrew Zisserman. Look, listen and learn. *CoRR*, abs/1705.08168, 2017.

[4] Mikel Artetxe, Gorka Labaka, Eneko Agirre, and Kyunghyun Cho. Unsupervised neural machine translation. *CoRR*, abs/1710.11041, 2017.

[5] Pinkesh Badjatiya, Shashank Gupta, Manish Gupta, and Vasudeva Varma. Deep learning for hate speech detection in tweets. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pages 759–760. International World Wide Web Conferences Steering Committee, 2017.

[6] T. Baltrušaitis, C. Ahuja, and L. Morency. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):423–443, Feb 2019.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[8] Philipp Blandfort, Desmond U Patton, William R Frey, Svebor Karaman, Surabhi Bhargava, Fei-Tzin Lee, Siddharth Varia, Chris Kedzie, Michael B Gaskell, Rossano Schifanella, et al. Multimodal social media analysis for gang violence prevention. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 13, pages 114–124, 2019.

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[12] Nemanja Djuric, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavljevic, and Narayan Bhamidipati. Hate speech detection with comment embeddings. In *Proceedings of the 24th international conference on world wide web*, pages 29–30. ACM, 2015.

[13] Sidney K D'mello and Jacqueline Kory. A review and meta-analysis of multimodal affect detection systems. *ACM Computing Surveys (CSUR)*, 47(3):43, 2015.

[14] Björn Gambäck and Utpal Kumar Sikdar. Using convolutional neural networks to classify hate-speech. In *Proceedings of the First Workshop on Abusive Language Online*, pages 85–90, Vancouver, BC, Canada, August 2017. Association for Computational Linguistics.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[16] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[17] Ronghang Hu, Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Kate Saenko. Learning to reason: End-to-end module networks for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 804–813, 2017.

[18] James B. Jacobs and Kimberly Potter. *Hate Crimes: Criminal Law and Identity Politics*. Oxford University Press USA, 1998.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[20] Irene Kwok and Yuzhou Wang. Locate the hate: Detecting tweets against blacks. In *Twenty-seventh AAAI conference on artificial intelligence*, 2013.

[21] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

[22] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[23] Wei-Hao Lin and Alexander Hauptmann. News video classification using svm-based multimodal classifiers and combination strategies. In *Proceedings of the tenth ACM international conference on Multimedia*, pages 323–326. ACM, 2002.

[24] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 3074–3082, 2015.

[25] Shervin Malmasi and Marcos Zampieri. Detecting hate speech in social media. *CoRR*, abs/1712.06427, 2017.

[26] Toni M Massaro. Equality and freedom of expression: The hate speech dilemma. *Wm. & Mary L. Rev.*, 32:211, 1990.

[27] Mainack Mondal, Leandro Araújo Silva, and Fabrício Benevenuto. A measurement study of hate speech in social media. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*, pages 85–94. ACM, 2017.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[29] Ashwin P Rao. Multimodal speech recognition system, January 15 2013. US Patent 8,355,915.

[30] Scott E. Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *CoRR*, abs/1605.05396, 2016.

[31] Xin Rong. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*, 2014.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[34] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[36] Jeremy Waldron. *The harm in hate speech*. Harvard University Press, 2012.

[37] Samuel Walker. *Hate speech: The history of an American controversy*. U of Nebraska Press, 1994.

[38] Ziqi Zhang, David Robinson, and Jonathan Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *European Semantic Web Conference*, pages 745–760. Springer, 2018.