Soumil Gupta
2022102035
soumil.gupta@students.iiit.ac.in

Precog Recruitment Task
Task 2: Analyzing Hateful Memes

2024-02-21

Before we begin, I would like to inform a few important things with regards to this submission:

- I have attempted all the compulsory sub-tasks associated with the task. I have also tried my best to complete the bonus tasks as well.
- The dataset to be used for this classification task was to be downloaded from this link. But, in order to overcome time and compute restrictions, I have used a subset of this dataset. The subset contains a train-set and a validation-set. Each of the two sets contains the meme images in a classified fashion i.e., divided into *Hateful Memes* and *Not Hateful Memes* folder.
- In particular, the train-set contains 4800 images (2400 hateful and 2400 not-hateful) and the validation-set contains 1200 images (600 hateful and 600 not-hateful). This amounts to a total of 6000 sample images in the dataset, which is around 50% of the original dataset.

1 Object Detection

The task has been solved in a Python Notebook and can be found in the GitHub repository associated with this task. The notebook employs a structured approach to apply computer vision techniques for object detection in meme images.

1.1 Methodology

For the application of object detection model to identify elements in memes, the process unfolds as follows:

1. **Importing Necessary Libraries:** The analysis begins with loading specific Python libraries essential for image processing and object detection tasks
2. **Model Loading:** It utilizes the **facebook/detr-resnet-50 model**, a pre-trained deep learning model developed by Facebook AI, designed for efficient and accurate object detection.
3. **Dataset Preparation:** The notebook sets up paths and categorizes for classification, to better organize the task of handling the dataset. It specifies the base folder path and defines categories in order to manage the two types of images, namely **Meme and Not Meme** systematically.
4. **Object Detection Execution:** The pre-trained model is applied to the dataset to perform object detection. This step involves processing images, detecting objects within them, and analyzing the detection output.

For the task of cataloging detected objects and analyzing their frequency and distribution, I maintained two separate text files, one of them was the frequency counter for when the model is run over hateful-memes and the other is the frequency counter for when the mode is run over non-hateful memes. If an object is present in a hateful meme, its frequency increases by 1 in the corresponding text file. Similarly, if its present in a non-hateful meme, its frequency increases by 1 in the corresponding text file.

1.2 Results & Observations

As mentioned above, the notebook employs the use of the **Facebook Detr ResNet-50** deep learning model. The model is designed for object detection and recognition tasks. It is based on the ResNet-50 architecture, a widely used convolutional neural network.

Just like in our case, when used on an image, the model processes the image through its layers to extract features and then employs a transformer-based architecture to perform object detection. The output of the model includes bounding boxes around detected objects along with their corresponding class labels (in our case, the labels are: hateful memes and not-hateful memes) and confidence scores. This output provides information about the location and identity of objects present in the input image.

The results have been recorded in a text file. The file contains the Image ID followed by the objects detected by the model along with their locations and confidence scores. An example is shown below

```
Image - C:\Users\soumi\Desktop\precog\img\validation_data\hateful_memes\80156.png :
Detected person with confidence 1.0 at location [0.35, 42.49, 533.22, 789.95]
```

Note: In my code, I am only printing the results of objects detected with a confidence score of more than 0.9 to ensure reliable results.

The results for the frequency distribution task was surprising. Here are the first few entries for the hateful-memes dataset frequency analysis:

```
person: 1104
tie: 92
car: 51
```

Now, here are the first few entries for the not-hateful memes dataset frequency analysis:

```
person: 1109
tie: 133
car: 50
```

As you can observe, the results are pretty similar. This is probably because the object detection model that we have chosen is a very general one, meaning that it has very generic class labels like ”person”, ”weaponëtc-, which is because it has not been particularly trained on a meme dataset specifically. The implication of this is that it gives little aid in determining if a meme can be ”hatefulör not because it is oblivious to the specificity's of the image. The model will label a image of a ”normal personänd the image of ”Hitler”both to a ”person. But we intuitively know that ”Hitlerïs more likely to be associated with a hateful-meme than a picture of a ”normal person”.

2 Caption Impact Assessment

The task has been solved in a Python Notebook and can be found in the GitHub repository associated with this task. The notebook employs a structured approach to apply computer vision techniques for Optical Character Recognition in the meme image and then systematically evaluate the impact of the presence of caption in the object-detection process.

2.1 Tools Used

1. **python-tesseract**: It is an optical character recognition (OCR) tool for python. Python-tesseract is a wrapper for Google's Tesseract-OCR Engine. it can read all image types including jpeg, png, gif, bmp, tiff, and others. Another benefit of using tesseract is that it is very lightweight and has a comparable performance to other models like EasyOCR etc-.
2. **facebook/detr-resnet-50 for Object Detection**: The reason why I chose this model was because it uses a transformer architecture, which is inherently parallelizable, allowing DETR to scale efficiently to larger datasets and higher resolutions without sacrificing performance. This was important for me because I chose to run all of my codes locally on my computer and not on platforms like Google Colab Notebook.

2.2 Methodology

Firstly, I developed a code to use python-tesseract to extract the text from any meme-image that has been inputted by the user. But, this is not as simple as simply running the model directly over the image. The problem is that tesseract is optimized to recognize text in typical text documents, so it can be difficult to recognize text within the image without pre-processing it.

Here's how I pre-processed the image:

1. **Bilateral Filtering:** This filter helps to remove the noise, but, in contrast with other filters, preserves edges instead of blurring them. Visually, it might not make a lot of difference but it leads to a better final performance.
2. **Grayscaling:** Grayscaling the image is important because the model works much more accurately when the text is in a contrast with its background, i.e, dark text on a light background.

3. **Binarization:** For every pixel, the same threshold value is applied: If the pixel value is smaller than the 240, it is set to 0, otherwise, it is set to 255. Since we have white text, we want to blackout everything is not almost perfectly white (roughly).
4. **Colour-Inversion:** Since tesseract is trained to recognize black text, we also need to invert the colors before finally sending it for the actual OCR operation.

I used this code and ran it over a dataset of the 2400 hateful-meme images. I didn't calculate a metric as such to represent the accuracy or any other performance metric due to time constraints. But here is the next part: In order to access the impact of caption on the object detection results, I did the following:

- Use OCR to extract the text and recognize the location (box) of the text container.
- Removed the text by creating a mask around the text container infilling the background using **INPAINT_NS** which is based on Navier-Stokes method. I explored other options like using Generative AI fill to more accurately fill the missing background but I was not able to implement it in the given time frame.

So now I have the original image (the one with the text) and a in-painted image (with the text reasonably filtered out). Now, in order to access the impact of the caption, I ran the object detection algorithm previously used on both of the images and stored the results of the object detection in a text file.

2.3 Results & Observation

In general, I saw an increase in confidence score of some of the objects recognized, this might be due to the text interfering with that object in the original image leading to a lower confidence score which improved after the application of in-painting.

There were also incidences of false objects being reported by the model, suggesting that the imperfect in-painting might have led to the model mistaking the image with something else.

3 Classification System Development

The task has been solved in a Python Notebook and can be found in the GitHub repository associated with this task. The notebook employs a structured approach to apply computer vision techniques for classification of meme images into **hateful or not-hateful** categories.

In my approach, I have separately used two pre-trained models, originally built on generic image classification and modified, tuned and trained it on my dataset to output a binary classification: whether the meme is **hateful or not.**

3.1 Tools Used

- **Model 1 - VGG16:** VGG16 is a convolutional neural network architecture characterized by its deep architecture consisting of 16 layers. VGG16 has a straightforward architecture with small convolutional filters, making it easy to understand and implement. Despite its simplicity, VGG16 achieves competitive performance on various image classification tasks due to its deep architecture.
- **Model 2 - MobileNetV2 from keras:** Keras includes a number of pre-trained models, one of these is MobileNetV2, which has been trained to classify images. By using the **imagenet** weights, we invoke the pre-trained model that we can use in the code for image classification. Similar to VGG16, we tune and train the model to work as a binary image classifier.

3.2 Methodology:

The detailed methodology can be understood from the code description and comments provided in the python notebook itself for both of the models. Some important parameters that I used for training were:

For VGG16:

- batch_size = 10
- input_shape = (224, 224, 3)

- learning_rate (lr) = 0.0001
- epochs = 10

For MobileNetV2:

- batch_size = 1
- input_shape = (224, 224, 3)
- learning_rate (lr) = 0.001
- epochs = 10

3.3 Results & Observation

Performance metrics for VGG16:

```
Validation Loss: 0.6894544363021851
Validation Accuracy: 0.5575000047683716
```

Performance metrics for MobileNetV2:

```
Validation Loss: 0.8392002582550049
Validation Accuracy: 0.5641666650772095
```

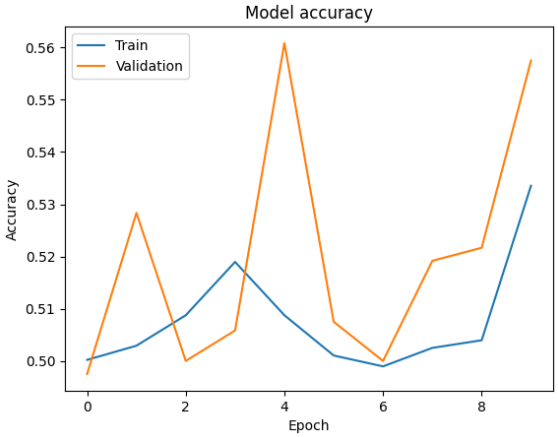

Abbildung 1: Accuracy Plot for VGG16

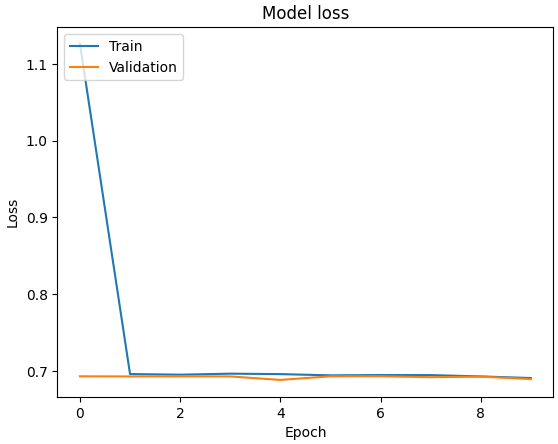

Abbildung 2: Loss Plot for VGG16

4 Bonus Task

The task has been solved in a Python Notebook and can be found in the GitHub repository associated with this task. The notebook employs a structured approach to apply computer vision techniques and natural language processing techniques to predict meme toxicity based on image text.

4.1 Tools Used

- **python-tesseract**: It is an optical character recognition (OCR) tool for python. Python-tesseract is a wrapper for Google's Tesseract-OCR Engine. it can read all image types including jpeg, png, gif, bmp, tiff, and others. Another benefit of using tesseract is that it is very lightweight and has a comparable performance to other models like EasyOCR etc-.
- **bipin/image-caption-generator for Image Captioning:** The bipin/image-caption-generator model is a deep learning model designed for automatically generates descriptive captions for images.
- **facebook/roberta-hate-speech-dynabench-r4-target for Detecting Hate Speech:** The model is a RoBERTa variant fine-tuned for hate speech detection on the Dynabench dataset, offering robust identification of hate speech in text for online safety and community moderation.

4.2 Methodology:

Here's my approach towards solving the problem: Given a meme image, I extract three strings describing the meme:

- An OCR string that represents the caption text of the meme
- A caption for the image generated by a model that summarizes what is going on in the image.
- The third string is a concatenation of the above two strings.

Once I gained these three string for an image, I put them individually through a Hate Speech Classifier which gives a binary classification on whether the string is hateful or not. I ran this over a total of 600 hateful-memes taken from the validation-set. The result was saved in a .txt file. An example of the result is:

```
Image File Name: 79846.png
Extracted_text - Classification: hateful, Confidence: 0.7749
Image_caption - Classification: not hateful, Confidence: 0.9994
Combined_text - Classification: hateful, Confidence: 0.7213
```

4.3 Results & Observation

The results were very interesting. There were cases like the above when the extracted_text was hateful and the image_caption was not hateful and the combined string was judged to be hateful. But also I saw a few cases where both extracted_text and image_caption were hateful but the combined_string was seen as not hateful by the model.