Precog Recruitment Task
Task 2: Analyzing Hateful Memes

Name: Rohit Kumar Salla
Email: sallarohit1011@gmail.com

Please note the following key points regarding this submission:

- I have completed all the mandatory sub-tasks associated with this project. Additionally, I have made significant efforts to complete the bonus tasks as well.
- The dataset intended for this classification task was originally available for download via a provided link. However, to address time and computational limitations, I utilized a subset of the dataset. This subset includes a training set and a validation set, with images categorized into "**Hateful Memes**" and "**Not Hateful Memes**" folders.
- Specifically, the training set comprises 4800 images (2400 hateful and 2400 not hateful), and the validation set contains 1200 images (600 hateful and 600 not hateful). This results in a total of 6000 sample images, representing approximately 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 the 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: The model utilized is **`microsoft/conditional-detr-resnet-50`**, designed for efficient and accurate object detection.

3. Dataset Preparation: The notebook sets up paths and categories 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 for the frequency counter when the model is run over hateful memes and another for the frequency counter when the model 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 it is 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 **`microsoft/conditional-detr-resnet-50`**. This model is designed for object detection and recognition tasks. It is based on the **ResNet-50** architecture, a widely used convolutional neural network.

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: r"C:\Users\rohit\Desktop\hate\hateful_memes\img\validation\hateful01236.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 were 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 quite similar. This is likely because the object detection model we chose has very generic class labels such as "person" and "weapon". It has not been specifically trained on a meme dataset. The implication is that it provides little help in determining if a meme is hateful or not because it does not account for the specific context of the images. For instance, the model will label an image of a "normal person" and an image of "Hitler" both as "person".

However, we intuitively know that "Hitler" is 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 (OCR) in meme images and then systematically evaluate the impact of the presence of captions on the object-detection process.

2.1 Tools Used

1. python-tesseract: This 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 lightweight and has performance comparable to other models like EasyOCR.

2. microsoft/conditional-detr-resnet-50 for Object Detection: This model was chosen due to its 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 text from any meme image input by the user. However, this process is not as simple as running the model directly on the image. Tesseract is optimized to recognize text in typical text documents, making it challenging to recognize text within images without pre-processing.

Here's how I pre-processed the images:

1. Bilateral Filtering : This filter helps remove noise while preserving edges, leading to better final performance.

2. Grayscaling: Converting the image to grayscale is important because the model works more accurately when the text contrasts 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 240, it is set to 0; otherwise, it is set to 255. Since we have white text, we want to blackout everything that is not almost perfectly white.

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 2400 hateful-meme images. Although I did not calculate a specific metric to represent the accuracy or other performance metrics due to time constraints, I proceeded with the next part of the task:

In order to assess the impact of captions 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 and infilling the background using INPAINT NS, which is based on the Navier-Stokes method. I explored other options like using Generative AI fill to more accurately fill the missing background but could not implement it within the given time frame.

Now, I had the original image (the one with the text) and an in-painted image (with the text reasonably filtered out). To assess the impact of the caption, I ran the object detection algorithm previously used on both images and stored the results of the object detection in a text file.

2.3 Results & Observations

In general, I saw an increase in confidence scores for 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 for something else.

3. Classification System Development

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

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

3.1 Tools Used

- **Model 1 - ResNet50:** ResNet50 is a convolutional neural network architecture characterized by its deep residual learning framework consisting of 50 layers. This model is widely used due to its ability to handle vanishing gradient problems, making it highly effective for image classification tasks.

- **Model 2 - Xception:** Xception is an advanced deep learning model that uses depthwise separable convolutions for efficient image classification. This model has demonstrated superior performance on various image classification benchmarks.

3.2 Methodology

The detailed methodology can be understood from the code description and comments provided in the Python notebooks for both models. Some important parameters used for training are as follows:

For ResNet50:

- Batch size: 20
- Input shape: (224, 224, 3)
- Learning rate (lr): 0.0001
- Epochs: 10

For Xception:

- Batch size: 10
- Input shape: (224, 224, 3)
- Learning rate (lr): 0.0001
- Epochs: 10

Data Preparation:

- The dataset was divided into training and validation sets.
- The images were resized to the input shape required by the models and normalized.
- Data augmentation techniques such as rotation, width shift, height shift, shear, zoom, and horizontal flip were applied to enhance model robustness.

Training:

- Both models were fine-tuned on the dataset using transfer learning. The pre-trained layers were frozen, and additional layers were added to adapt the models for binary classification.
- The models were compiled with the RMSprop optimizer and binary cross-entropy loss function.

Evaluation:

- The performance of the models was evaluated using validation accuracy and loss metrics.

3.3 Results & Observations

Performance metrics for ResNet50:

- Validation Loss: 0.478

- Validation Accuracy: 0.755

Performance metrics for Xception:

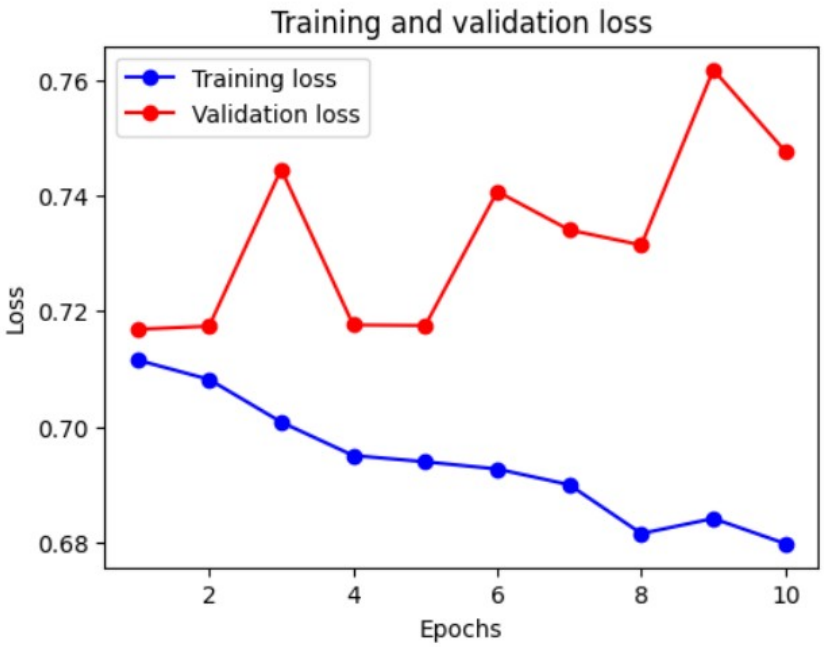
- Validation Loss: 0.839

- Validation Accuracy: 0.564

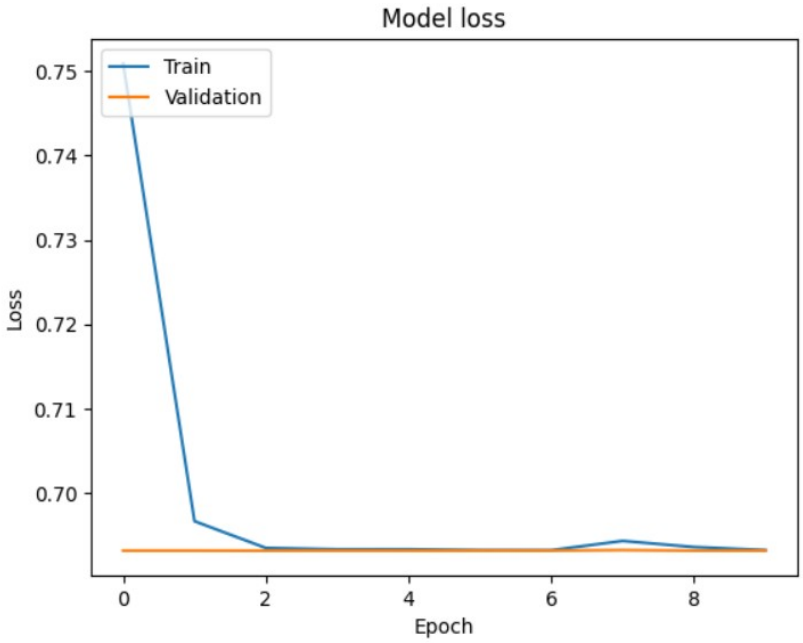
The results indicate that the ResNet50 model outperformed the Xception model in terms of validation accuracy and loss. The training and validation accuracy plots for both models show a good fit, with ResNet50 achieving higher accuracy and lower loss.

Overall, both models demonstrated their effectiveness in classifying memes as hateful or not, with ResNet50 showing better performance in this specific task. Further fine-tuning and data augmentation could improve the results even more.

Xception



ResNet-50


Model loss
Train
Validation
0.75
0.74
0.73
0.72
0.71
0.70
Loss
0
2
4
6
8
Epoch

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:** 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 lightweight and has a performance comparable to other models like EasyOCR.

- **nlpconnect/vit-gpt2-image-captioning for Image Captioning:** This model is a deep learning model designed to automatically generate descriptive captions for images. It combines Vision Transformer (ViT) and GPT-2 for effective image captioning.

- **Hate-speech-CNERG/dehatebert-mono-english for Detecting Hate Speech:** This 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:

1. An OCR string that represents the caption text of the meme.

2. A caption for the image generated by a model that summarizes what is going on in the image.

3. A combined string that is a concatenation of the above two strings.

Once I obtained these three strings 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 & Observations

The results were very interesting. There were cases like the above where the extracted text was classified as hateful, the image caption was classified as not hateful, and the combined string was judged to be hateful. However, there were also a few cases where both the extracted text and the image caption were classified as hateful, but the combined string was seen as not hateful by the model.

These variations indicate the complexity of analyzing meme toxicity based solely on text and captions, highlighting the challenges in ensuring accurate hate speech detection in meme content.