PRECOG TASK

Analyzing hateful memes
Ashray Gupta

TASK A

I had used Yolov5 for Object detection with 25% of images from hateful meme dataset

YOLOv5 model was made based on its popularity and effectiveness in real-time object detection tasks. YOLOv5 (You Only Look Once version 5) is known for its balance between accuracy and speed, making it suitable for various applications, including image and video analysis.

Challenges while doing this task-

Previously I had attempted to do this task with Yolov8 and Yolov6 but there were several problems in implementing them-

1. All the libraries I had installed on my local system were not compatible with it.
2. On Kaggle there were some implicit issues while using them. Specifically Qulerror and unresolved dependencies with scipy library while using Kaggle prompted me to go ahead with yolov5
3. Yolov8 and Yolov6 had a much less community support, resources available online for taking references.

Result:

The detailed results can be found in the result csv files.

1. Category Count results- The count of categories in each image as detected.
2. Object detection results- Lists the categories found in each image along with the confidence score.
3. Object Frequency distribution- Types of objects detected and their frequency across the entire dataset.
4. Used image dataset- Documented all the images used (the 25%) for this task.

The overall frequency distribution was as follows:

Object_Class	Frequency
person	3443
tie	263
sheep	104
backpack	6
frisbee	12
kite	6
banana	3
orange	6
clock	12
dog	74
chair	42
boat	5
traffic light	7
truck	9
cow	29
car	77
microwave	6
oven	5

cell phone	47
cake	19
cat	33
mouse	5
skateboard	3
surfboard	4
horse	28
bottle	40
cup	71
wine glass	9
bowl	31
potted plant	22
book	7
suitcase	2
vase	12
donut	21
laptop	14
bed	5
sink	3
teddy bear	4
train	3
hot dog	25
toilet	2
airplane	17
remote	4

bird	37
elephant	6
sports ball	13
bus	1
bear	10
umbrella	9
motorcycle	4
couch	1
handbag	8
bicycle	6
toothbrush	1
keyboard	2
baseball glove	6
apple	4
tv	5
stop sign	5
baseball bat	3
fork	3
bench	2
zebra	2
refrigerator	2
dining table	2
sandwich	2
pizza	4
spoon	2

carrot	5
skis	1
knife	1
tennis racket	2

Conclusion:

I had manually checked 25 images of the output to cross check the results. The results were right for 20/25 images.

TASK B:

Script 1:

In this code I had ran two scripts on the same set of used images in TASK A

Here I used the Yolov5 object detection model to analyze meme images, specifically investigating the influence of text captions on object detection accuracy. The script iterates through a list of used images (which I had obtained from the previous step) , initially detecting objects in the original images. Subsequently, the script utilizes Keras OCR for text recognition and generates a mask to erase captions from the images. The processed images without captions are then subjected to YOLOv5 object detection again. The script records metrics for each image, such as the number of detected objects in both the original and caption-removed versions. The script ultimately saves the processed images without captions and the comparison results, including metrics and image filenames, to respective output files. The aim is to assess the impact of removing captions on object detection outcomes and provide a comprehensive analysis in CSV format for further evaluation.

Here the major output file tells me the number of objects detected before caption removal and number of objects detected after caption removal.

Script 2:

This script performs a comprehensive image processing pipeline that combines object detection and text removal on a set of images. It utilizes the YOLOv5 model for object detection and Keras OCR for text detection. The script iterates through a list of images, loads the YOLOv5 model with specified weights, detects objects in the images, uses Keras OCR to identify and draw bounding boxes around text regions, and subsequently removes the detected text from the images. The cleaned images are saved, and information about the detected objects, such as categories and their frequencies, is recorded in various CSV files. The script also generates CSV files containing overall object distribution, details about object detection after caption removal, and category counts after caption removal.

Here the outputs are more comprehensive and detailed so the comparison with TASK A becomes easier.

Challenges:

1. It was difficult to devise a method to analyze the impact of caption removal without any ground truth.
2. If there were a ground truth, the accuracy and metrics would have been explicit in telling the impact.
3. I had attempted to apply pytesseract for caption removal, but while having higher computation time the results were not good. Hence, I dropped it altogether.
4. RGB channel issue: I was previously converting to have all the image be in selected channels of RGB but some images had single channels, some had 3 and some had 4 therefore a single strategy wasn't working and was causing tensor mismatch issue with the model again and again
5. Models like Yolov8 and Yolov6 were very cost and time intensive. So I wasted a lot of time in trying to implement those.

Outputs:

Script 1:

1. Comparison Results: Tells the number of objects detected before caption removal and after caption removal across the categories
2. Used_images list: Documents all the images used in the script

Script 2:

1. Overall object distribution: Tells the distribution of the detected object classes and their frequency across the dataset
2. Object detection (with text removal): Lists the image name, the detected categories and its cleaned path
3. Category Count: Tells us the number of objects of each category as detected in each image
4. Used_image: Documents all the images used in the script

Observations:

1. The number of images detected after caption removal are much higher as compared to before caption removal.
2. After manual screening of 25 images, it was observed that for 25/25 images the caption removal worked better for object detection.

TASK C:

Data Preparation:
Sampling Images: The code samples a specified percentage (20%) of images randomly from original meme and non-meme datasets. This is done to create a smaller dataset for training and testing.

Splitting and Labeling Datasets: The sampled images are then split into training (70%), validation (15%), and test (15%) sets. Each set is labeled according to its class (meme or non-meme).

Custom Dataset Class:
A custom dataset class (CustomDataset) is defined to load and preprocess the images. It inherits from torch.utils.data.Dataset and implements the __len__ and __getitem__ methods. The class loads images, applies transformations, and assigns labels based on the directory structure.

Data Loaders:
Data loaders are created for the training, validation, and test sets using the DataLoader class from PyTorch. These loaders are responsible for efficiently loading batches of data during training and evaluation.

CNN Model:
A simple CNN model (SimpleCNN) is defined using PyTorch's nn.Module. It consists of two convolutional layers followed by max-pooling, and two fully connected layers. The output layer uses sigmoid activation to predict binary outcomes (meme or non-meme).

Model Training:
The CNN model is trained using the BCEWithLogitsLoss loss function and the Adam optimizer. The training loop runs for a specified number of epochs (10), and tqdm is used to display a progress bar.

Model Evaluation:
The trained model is evaluated on the test set. Predictions are compared to true labels, and accuracy is calculated. The code also prints some predicted labels and true labels for further investigation.

Hyperparameters:
Image Dimensions: img_width and img_height are set to 150x150 pixels.

Batch Size: batch_size is set to 32.
Number of Epochs: Training is done for 10 epochs.
Learning Rate: The Adam optimizer is used with a learning rate of 0.001.
Techniques and Considerations:
Data Augmentation: No explicit data augmentation is applied in the code, but you could extend the transforms in the CustomDataset class to include augmentations.
Normalization: The images are normalized using transforms.ToTensor().
Activation Function: ReLU is used for hidden layers, and sigmoid is used for the output layer.
Loss Function: Binary Cross Entropy with Logits (BCEWithLogitsLoss) is used for binary classification.

Optimizer: Adam optimizer is used for model optimization.

Challenges:

1. Despite changing and tweaking hyperparameters I am getting a very high accuracy in the end. I observed an inconsistent data labeling and was able to get it down from 3151 to 2699

2. Was difficult to build the model from scratch.

3. Was able to achieve 99% accuracy with TensorFlow

Results:

Test Accuracy: 2699.53%

Comments:

Something is definitely going wrong with the model introducing such a large accuracy which after several attempts I wasn't able to resolve.

Potential reasons:

1. Data leakage

2. Lack of Regularization etc

3.Inaccurate and inconsistent labeling.