Task 1: Object Detection

Task 1: Approach 1 - Object Detection Using DETR

Objective:

Perform object detection on a dataset of meme images using the DETR (DEtection TRansformer) model.

Approach:

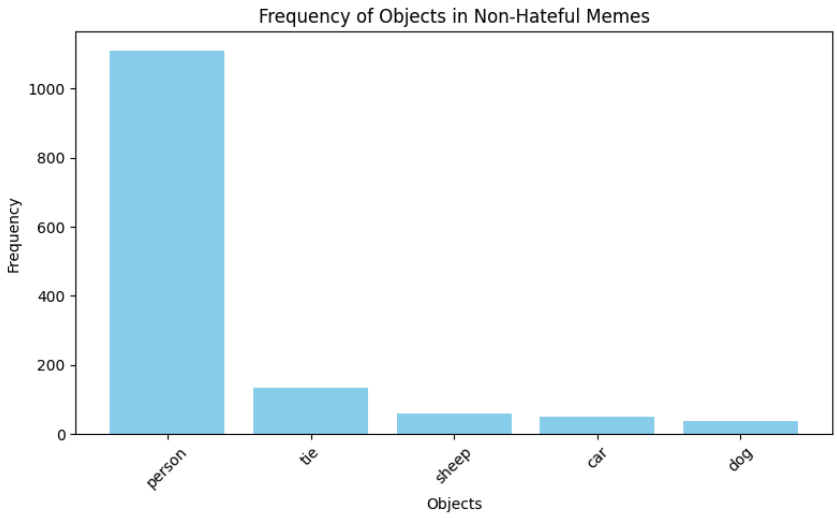
- The task involves performing object detection on a dataset of meme images.
- DETR is a powerful and state-of-the-art object detection model that has shown impressive performance in various computer vision tasks.
- DETR is based on the transformer architecture, which has proven to be effective in various natural language processing tasks and has also been successfully applied to computer vision problems.
- The pre-trained DETR model (facebook/detr-resnet-50) from the Hugging Face Transformers library is utilized, which provides a good starting point and can generalize well to diverse object categories.

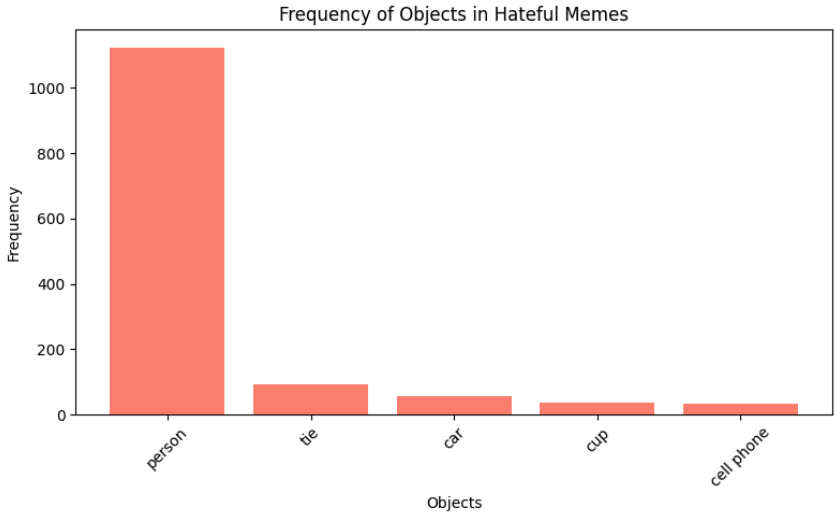
Methodology:

1. **Model Initialization**
   - Initialize the DETR object detection model (`facebook/detr-resnet-50`) from the Hugging Face Transformers library.
   - Initialize the `DetrImageProcessor` for preprocessing input images.
2. **Image Processing**
   - The `process_images` function processes images within specified categories.
   - For each category, iterate through images, open them, convert to RGB format, and preprocess using `DetrImageProcessor`.
3. **Object Detection**
   - Open and preprocess images.
   - Pass images to the DETR model for inference.
   - Post-process outputs to the COCO API format.
   - Apply a confidence threshold of 0.9 to filter low-confidence detections.
4. **Result Logging**
   - Log detection results for each image in text files named `output_results_{category}.txt`, including image path, detected object labels, confidence scores, and bounding box coordinates.
5. **Object Frequency Analysis**
   - Track object frequency across all images in each category using `object_counter`.
   - Write object frequencies to separate text files named `object_frequency_{category}.txt`.
6. **Output Files**
   - Return a list of output file paths containing detection result files and object frequency analysis files for each processed category.

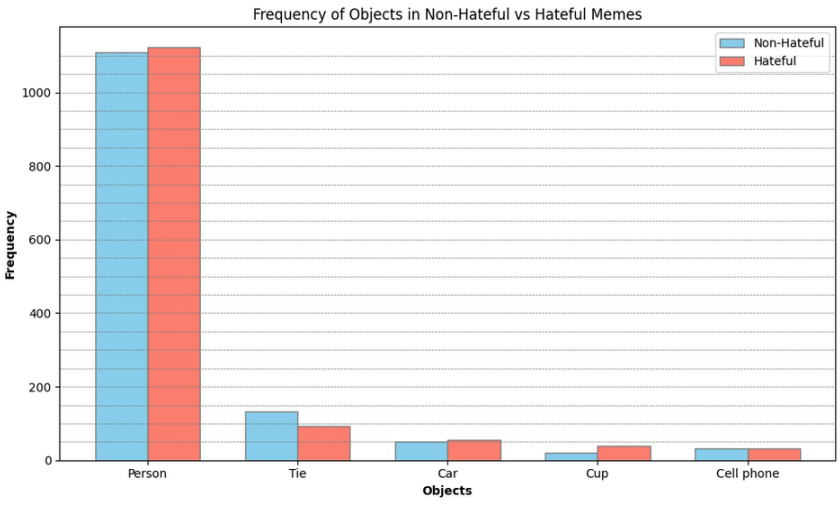
Findings:

The following graphs depict the frequency of the top 5 objects detected within two classes: hateful and non-hateful memes. These counts were derived from a catalog that lists objects along with their respective frequencies.





The following graph depicts a comparison between the frequencies of the top 5 objects.



Positives about this approach:

- DETR is a transformer-based object detection model known for its impressive performance in various computer vision tasks.
- Being pre-trained on a large dataset, DETR can generalize well to diverse object categories.
- It provides bounding box coordinates and confidence scores for detected objects, enabling further analysis and potential applications.

- The approach is well-documented and leverages popular libraries like Transformers and PyTorch, making it easy to understand and implement.

Negatives and potential failure cases:

- Object detection models, including DETR, may struggle with occlusions, unusual object orientations, and unfamiliar objects that are not well-represented in the training data. In this text occlusion is the topic we are going to focus in second task.
- Performance may degrade for low-quality, distorted, or noisy images, which are common in meme datasets.
- Object detection alone may not capture the full context present in meme images, which could be crucial for tasks like toxicity classification or understanding the meme's intended message.
- Transformer-based models like DETR can be computationally expensive, especially for high-resolution images, which may impact performance or require specialized hardware.

Task 1: Approach 2 - Object Detection Using LLaVa

Objective:
Perform object detection on meme images using the LLaVa (Large Language Vision Assistant) model.

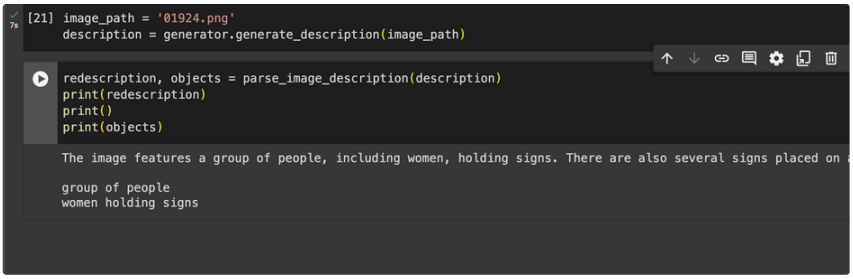
Approach:

- The task involves performing object detection on a dataset of meme images.
- LLaVa is a powerful multimodal model that can process both text and visual information, making it suitable for tasks like object detection, image captioning, and visual question answering.
- By leveraging LLaVa's multimodal capabilities, the approach aims to generate human-readable descriptions of the objects present in the meme images and explicitly list the most highlighted or prominent objects.
- The approach utilizes LLaVa's language understanding capabilities to provide flexible prompting, allowing customization of the output format or obtaining specific insights from the visual information.

Methodology:

1. **Model Initialization**
   - The `ImageDescriptionGenerator` class is responsible for loading the LLaVa model and generating textual descriptions from images.
   - The `model_id` parameter specifies the pre-trained LLaVa model to use.
   - The `BitsAndBytesConfig` is used to enable quantization for efficient model loading and inference.
2. **Model Loading**
   - The `load_model` method initializes the Hugging Face pipeline for the "image-to-text" task, using the specified LLaVa model and quantization configuration.
3. **Image Description Generation**
   - The `generate_description` method takes an image path as input and generates a textual description using the loaded LLaVa model.
   - The image is loaded using the Pillow library, and a prompt is constructed to guide the model's response.
   - The prompt instructs the model to describe objects present while ignoring text elements and provide a list of highlighted objects.
   - The model's output is obtained through the pipeline and returned as the image description.
4. **Description Parsing**
   - The `parse_image_description` function extracts the textual description and list of highlighted objects from the model's output.
   - It searches for the "ASSISTANT:" token, extracts the text after it, splits it into lines, and identifies the line where the object list begins.
   - The function separates the description and object list, removes duplicates from the description, and returns both parts.

Findings:



Positives about this approach:

- LLaVa is a powerful multimodal model that can process both text and visual data, enabling it to understand and describe complex visual scenes.
- The model can generate human-readable descriptions and explicitly list the most highlighted objects, providing interpretable results.
- LLaVa's language understanding capabilities allow for flexible prompting, enabling users to customize the desired output format or obtain various insights from visual information.

Negatives and potential failure cases:

- The LLaVa model is computationally expensive due to its large size and complexity, requiring high computational resources and GPU acceleration for efficient inference.
- The performance and quality of the generated descriptions may vary depending on the complexity of the input images with prompts and the model's understanding of the visual and textual context.

Task 1: Approach 3 - Object Detection Using YOLOv8

Objective:

Perform object detection on meme images using YOLOv8 (You Only Look Once version 8).

Approach:

- The task involves performing object detection on a dataset of meme images.
- YOLOv8 is a highly optimized and efficient object detection model known for its impressive speed and accuracy.
- By leveraging YOLOv8, the approach aims to accurately detect and localize multiple objects in the meme images with real-time performance.
- The approach generates a CSV catalog file containing the detected objects, their confidence scores, and ground truth labels, facilitating further analysis and evaluation.

Methodology:

1. **Model Installation and Import**
   - Install required dependencies, including the `ultralytics` library.
   - Import necessary modules from the `ultralytics` library, including `YOLO`.
2. **Model Loading**
   - Load the pre-trained YOLOv8 model using the `YOLO` class from the `ultralytics` library.
   - Specify the desired model configuration, such as the model version (e.g., 'yolov8n.pt').
3. **Catalog Generation**
   - The `append_to_catalog_from_json` function appends detected objects and their labels to a CSV catalog file.
   - It takes two arguments: `json_file_path` (path to a JSON file containing image paths and labels) and `catalog_path` (path to the CSV catalog file).
   - The function iterates through each entry in the JSON file, retrieving the image path and ground truth label.
   - For each image, it performs object detection using the loaded YOLOv8 model and stores the detected objects, their confidence scores, and ground truth labels in a list.
   - The detected objects, along with their labels and confidence scores, are appended to the specified CSV catalog file.
4. **Result Logging**
   - The function logs a message indicating that the detected objects have been appended to the catalog file.

Findings:

From this process, we detect objects and store their confidence levels, indicating how accurately the object is detected as belonging to a certain class. Additionally, we save the image path, the detected objects, their respective confidence levels, and the ground truth label associated with each detection. This information can be utilized for further processing and analysis.

```
    print(row)

{'Image': '/content/01235.png', 'Object': "['person']", 'Confidence': '[    0.96653]', 'Ground Truth Label': 'hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.75815]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.70568]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['horse']", 'Confidence': '[    0.58712]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.50091]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[    0.48118]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.46822]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[    0.46252]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[    0.36762]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.33579]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[    0.32556]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[     0.2777]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['person']", 'Confidence': '[    0.26241]', 'Ground Truth Label': 'non-hateful'}
{'Image': '/content/01236.png', 'Object': "['sheep']", 'Confidence': '[     0.2592]', 'Ground Truth Label': 'non-hateful'}

Start coding or generate with AI.
```

Positives:

- YOLOv8 is a highly optimized and efficient object detection model, providing real-time performance.
- It provides bounding box coordinates, confidence scores, and class labels for each detected object, enabling further analysis and potential applications.

Negatives:

- YOLOv8 is capable of recognizing only the 80 base classes from the COCO dataset on which it was trained, limiting its ability to detect uncommon or specialized objects that may be present in meme images.

Task 2: Caption Impact Assessment

Objective:

Assess the effect of overlaid captions on the accuracy and effectiveness of object detection in meme images through qualitative and quantitative analysis. Explore methods to minimize the impact of captions on the object detection process.

Approach:

1. The approach combines two key components: text detection using EasyOCR and object detection using YOLOv8, which are well-established libraries for their respective tasks.
2. It provides both qualitative and quantitative analysis of the impact of text overlays on object detection.
3. The code includes functions to calculate the intersection between text and object bounding boxes, which can help assess the potential interference caused by text overlays.
4. The approach starts by detecting the text in the image using EasyOCR, which is a popular OCR library known for its accuracy and speed.
5. It then performs object detection using the YOLOv8 model, which is a state-of-the-art real-time object detection system based on deep learning.
6. The code calculates the area covered by the detected text and objects, as well as the intersection area between them.
7. By comparing the object detection results with and without text removal, or by analyzing the intersection between text and object bounding boxes, the impact of text overlays can be assessed.

Methodology:

1. **Assessing Caption Impact**
   - Perform object detection on a subset of meme images with and without overlaid captions using previously employed object detection models (e.g., DETR, YOLOv8, or LLaVa).
   - Analyze detection results qualitatively by comparing detected objects, bounding boxes, and confidence scores for images with and without captions.
   - Develop quantitative metrics to measure the impact of captions on object detection accuracy, such as:
     - Intersection over Union (IoU) between bounding boxes for the same object with and without captions
     - Difference in confidence scores for the same object with and without captions
     - Number of missed or falsely detected objects due to the presence of captions
2. **Minimizing Caption Impact**
   - Explore image processing techniques to filter out or remove text overlays from meme images before performing object detection.
   - Investigate pre-trained models or libraries capable of text detection and removal, such as Google Cloud Vision API, Tesseract OCR, or PyTesseract.
   - Implement a pipeline that combines text detection/removal with the chosen object detection model.
   - Evaluate the effectiveness of the text removal approach by comparing object detection results on meme images with and without text removal.

Findings and Results:

For a sample meme image, and the following results were obtained:

Text Detected:
"hope on a rope its what everyone's been waiting for"

Text Coordinates:
[[60, 0], [429, 0], [429, 59], [60, 59]]
[[345, 361], [359, 361], [359, 387], [345, 387]]
[[33, 345], [479, 345], [479, 455], [33, 455]]
[[77, 979], [406, 979], [406, 1060], [77, 1060]]

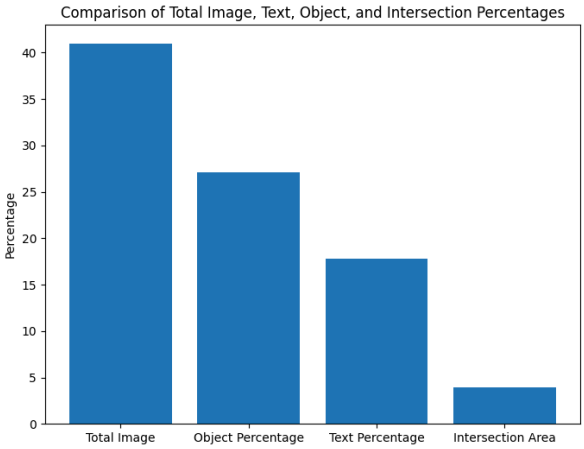
Object Detected:
['person', 'tennis racket']

Object Coordinates:
[[[60, 1], [511, 1], [511, 332], [60, 332]]]

Text Percentage: 17.776889534883722

Object Percentage: 27.122274709302324

Intersection Percentage: 3.966206395348837



The results indicate that the EasyOCR library successfully detected multiple text regions in the meme image, while the YOLOv8 model detected a person and a tennis racket as objects.

The text percentage, which represents the area of the image covered by the detected text, is 17.78%. Similarly, the object percentage, representing the area covered by the detected objects, is 27.12%.

The intersection percentage, which measures the area of overlap between the detected text and objects, is 3.97%. This relatively low value suggests that the text overlays and objects do not significantly interfere with each other in this particular image.

To draw more comprehensive conclusions about the impact of text overlays on object detection, further analysis should be conducted on a larger dataset of meme images, considering various factors such as text size, font, color, and positioning relative to the objects of interest.

Assumptions:

1. The approach assumes that the text overlays are horizontal and can be detected.

Positives about this approach:

1. Utilizes well-established and reliable libraries for text detection (EasyOCR) and object detection (YOLOv8).
2. Provides both qualitative (visual inspection) and quantitative (metrics like IoU, confidence scores, and missed/false detections) analysis of the impact of text overlays.
3. Includes functionality to calculate the intersection area between text and object bounding boxes, which can help identify potential interference.
4. Modular design with separate functions for different tasks, making the code easy to understand and maintain.

Potential failures:

1. The accuracy of the text detection and object detection models may vary depending on the quality and complexity of the meme images.
2. The approach assumes that the text overlays are horizontal. It may struggle with non-horizontal or heavily stylised text.
3. The approach does not explicitly handle cases where the text and objects overlap significantly, making it difficult to separate their contributions to the object detection task.

Task: Minimizing the Impact of Captions in Meme Images

Objective:

Develop and implement methods to minimize the impact of captions in meme images, particularly for object detection tasks.

Proposed Approaches:

1. **Text Detection and Removal Pipeline:**
   - Explore various image processing techniques to filter out or remove text overlays from meme images while preserving important visual features.
   - **Utilize Pre-trained Models or Libraries:** Consider using pre-trained models or libraries capable of text detection and removal. Options include Google Cloud Vision API, Tesseract OCR, or PyTesseract.
2. **Easy OCR with Image Denoising:**
   - **Utilize Easy OCR:** Apply Easy OCR to detect text regions within meme images.
   - **Blur Text Regions:** Use image processing techniques to blur the regions containing detected text, effectively obscuring the text while retaining image context.
   - **Denoise Image:** Employ a diffusion model or similar image denoising technique to remove artifacts caused by blurring, ensuring that the image remains visually coherent.

Task 3: Meme Image Classification System

Objective:

Develop a classification system to distinguish meme images from non-meme images, using the provided meme dataset as the positive class and sourcing non-meme images as the negative class.

Approach:

The approach leverages transfer learning by using a pre-trained convolutional neural network (CNN) model, VGG16, as the base model. This is a common and effective technique in computer vision tasks, especially when dealing with limited training data. By utilizing the feature extraction capabilities of a pre-trained model, the system can take advantage of the knowledge learned from a large-scale dataset, enabling better generalization and potentially improved performance.

1. The code follows a typical workflow for image classification tasks, starting with data loading and preprocessing.
2. The approach utilizes data augmentation techniques (rotation, shifting, shearing, zooming, flipping) on the training data to increase data diversity and improve model robustness.
3. The pre-trained VGG16 model is used as the base model, with its layers frozen to preserve the learned features.
4. Custom layers (flatten, dense, dropout) are added on top of the pre-trained model to adapt it to the specific task of meme image classification.
5. The model is trained using the provided training and validation data generators, with the option to visualize the training history (accuracy and loss curves).
6. The trained model is evaluated on the validation data, and performance metrics (loss and accuracy) are reported.
7. The model is saved for future use, and a function is provided to make predictions on new images.

Methodology:

1. **Data Collection and Preprocessing**
   - Collect a non-meme image dataset from reliable sources, ensuring a diverse range of image types.
   - Preprocess the datasets (meme and non-meme) by resizing, normalizing, or applying necessary transformations for consistency.
2. **Feature Extraction**
   - Explore different feature extraction techniques, such as pre-trained CNNs (VGG, ResNet, EfficientNet), to extract relevant visual features from images.
   - Consider using self-supervised or transfer learning approaches to leverage pre-trained models on large-scale datasets.
3. **Model Selection and Training**
   - Choose an appropriate classification model architecture (CNN, Transformer-based, or a combination).
   - Split the datasets into train, validation, and test sets.
   - Train the chosen model on the training set, employing techniques like data augmentation, early stopping, and learning rate scheduling.
4. **Model Evaluation**
   - Evaluate the trained model's performance on the test set using appropriate metrics (accuracy, precision, recall, F1-score, confusion matrix).
   - Analyze the model's performance separately on meme and non-meme classes to identify potential biases or areas for improvement.
5. **Interpretation and Visualization**
   - Explore techniques like saliency maps, activation maps, or attention visualization to understand the model's decision-making process.
   - Visualize and interpret the model's predictions on successful and challenging examples.
6. **Hyperparameter Tuning and Ensemble Methods**

- Experiment with hyperparameter tuning techniques (grid search, random search, Bayesian optimization) to improve model performance.
- Consider ensemble methods (bagging, boosting) to combine multiple models and potentially enhance classification accuracy.

Findings and Results:

Data Exploration Results: (For a Sample of 6000 images Data)

```
print( total validation not hateful images: ')
```

```
total training hateful images: 2400
total training not hateful images: 2400
total validation hateful images: 610
total validation not hateful images: 600
```

```
        shuffle=False)
```

```
Found 4800 images belonging to 2 classes.
Found 1210 images belonging to 2 classes.
```

```
from tensorflow.keras.applications.vgg16 impo
```

Defining Model Architecture: (Before Training)

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 512)               12845568

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 1)                 513

=================================================================
Total params: 27,560,769
Trainable params: 12,846,081
Non-trainable params: 14,714,688
_________________________________________________________________
```

Training Logs: (For 10 Epochs) (For a Sample of 6000 images Data)

```
Epoch 1/10
480/480 [==============================] - 1661s 3s/step - loss: 1.2256 - ac
c: 0.4910 - val_loss: 0.6925 - val_acc: 0.5300
Epoch 2/10
480/480 [==============================] - 1527s 3s/step - loss: 0.6974 - ac
c: 0.5190 - val_loss: 0.6933 - val_acc: 0.4908
Epoch 3/10
480/480 [==============================] - 813s 2s/step - loss: 0.6960 - acc:
0.5054 - val_loss: 0.6933 - val_acc: 0.4925
Epoch 4/10
480/480 [==============================] - 737s 2s/step - loss: 0.6958 - acc:
0.5108 - val_loss: 0.6935 - val_acc: 0.4908
Epoch 5/10
480/480 [==============================] - 2495s 5s/step - loss: 0.6945 - ac
c: 0.4956 - val_loss: 0.6932 - val_acc: 0.5083
Epoch 6/10
480/480 [==============================] - 813s 2s/step - loss: 0.6935 - acc:
0.4960 - val_loss: 0.6931 - val_acc: 0.5083
Epoch 7/10
480/480 [==============================] - 836s 2s/step - loss: 0.6943 - acc:
0.5013 - val_loss: 0.6933 - val_acc: 0.4958
Epoch 8/10
480/480 [==============================] - 831s 2s/step - loss: 0.6964 - acc:
0.4985 - val_loss: 0.6929 - val_acc: 0.5083
Epoch 9/10
480/480 [==============================] - 772s 2s/step - loss: 0.6950 - acc:
0.5015 - val_loss: 0.6931 - val_acc: 0.5092
Epoch 10/10
480/480 [==============================] - 718s 1s/step - loss: 0.6948 - acc:
0.4990 - val_loss: 0.6931 - val_acc: 0.5092
```

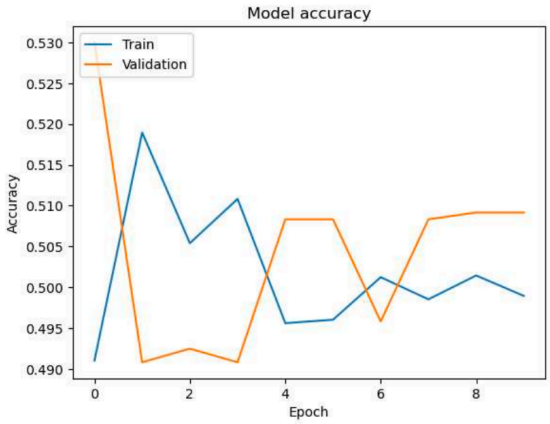
Testing with Validation Data:

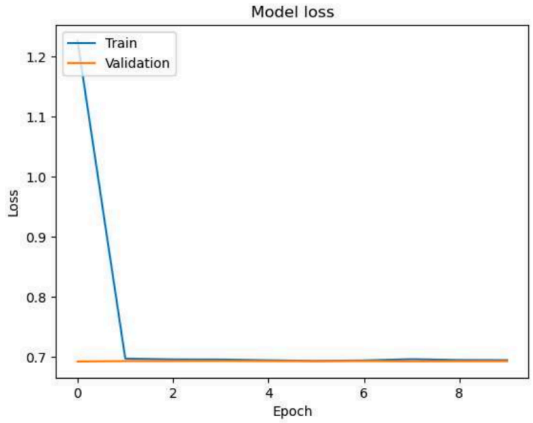
```
121/121 [==============================] - 130s 1s/step - loss: 0.6931 - acc:
0.5050
Validation Loss: 0.6931177973747253
Validation Accuracy: 0.5049586892127991
```

Inferring the trained model using the data:

```
1/1 [==============================] - 0s 125ms/step
01235.png is a Hateful Meme
1/1 [==============================] - 0s 117ms/step
01236.png is a Hateful Meme
1/1 [==============================] - 0s 134ms/step
01247.png is a Hateful Meme
1/1 [==============================] - 0s 126ms/step
01256.png is a Not Hateful Meme
1/1 [==============================] - 0s 129ms/step
01258.png is a Hateful Meme
1/1 [==============================] - 0s 122ms/step
01269.png is a Hateful Meme
1/1 [==============================] - 0s 131ms/step
01274.png is a Hateful Meme
1/1 [==============================] - 0s 138ms/step
01295.png is a Hateful Meme
1/1 [==============================] - 0s 110ms/step
01327.png is a Not Hateful Meme
1/1 [==============================] - 0s 118ms/step
```

Visualization of Performance of Model (Accuracy & Loss)





Where it can fail:

1. While transfer learning can be effective, the performance of the system may still be limited by the chosen base model (VGG16) and the amount of available training data.
2. If the provided meme image dataset or the collected non-meme image dataset contains biases or lacks diversity, the trained model may exhibit biased behavior or fail to generalize well to unseen data.
3. Training deep learning models, especially on large datasets, can be computationally intensive and may require significant computational resources (e.g., GPU acceleration).

These potential issues can be mitigated by:

1. Exploring different base models or architectures (e.g., ResNet, EfficientNet, Vision Transformers) and comparing their performance.
2. Collecting and curating a larger and more diverse dataset to improve the model's generalization capabilities.
3. Implementing more advanced data augmentation techniques or incorporating techniques like self-supervised learning or semi-supervised learning to leverage unlabeled data.
4. Considering ensemble methods or other techniques to combine multiple models and potentially improve overall performance.

Bonus Task: Predicting whether or not a meme is toxic

Building upon the existing LLaVa model implementation as shown in approach 2 of task 1, we can extend it to tackle the problem of toxicity classification based on meme text. The relevant information from the previous documentation has been included, and new components specific to the toxicity classification task have been added.

Toxicity Classification using LLaVa

1. **Model Initialization**
2. **Textual Data Processing**
3. **Toxicity Classification Model**
   - Implement or utilize a separate toxicity classification model trained specifically on meme text and corresponding toxicity labels.
   - This model should be capable of predicting the toxicity level of the generated meme text descriptions.
   - Potential model architectures include transformer-based models like BERT, RoBERTa, or XLNet, fine-tuned on meme text toxicity datasets.
4. **Toxicity Prediction**
   - Pass the textual descriptions generated by LLaVa to the toxicity classification model.
   - Obtain the predicted toxicity level or score for each meme text.
5. **Implementation Details**
   - Extend the existing `ImageDescriptionGenerator` class with a new method to handle toxicity classification.
   - This new method should take the image path as input, generate the textual description using LLaVa, and pass it through the toxicity classification model.
   - The method should return the predicted toxicity level or score.

```
class ToxicityClassifier:
    def __init__(self, model_id, toxicity_model_path):
        self.image_description_generator = ImageDescriptionGenerator(model_id)
        self.toxicity_model = load_toxicity_model(toxicity_model_path)

    def classify_toxicity(self, image_path):
        description = self.image_description_generator.generate_description(image_path)
        toxicity_score = self.toxicity_model.predict(description)
        return toxicity_score
```

The `ToxicityClassifier` class encapsulates both the LLaVa model for generating textual descriptions and the toxicity classification model. The `classify_toxicity` method orchestrates the entire process, taking an image path as input and returning the predicted toxicity score.

Potential Enhancements and Future Work

- This section remains the same as described in the previous documentation.

By integrating the toxicity classification model with the existing LLaVa implementation, we can leverage the multimodal capabilities of LLaVa to generate context-aware textual descriptions, which can then be fed into the toxicity classification model for accurate and efficient toxicity prediction based on meme text.