Meme Captioning Model Report

Rishi Roy

2K21/CO/384, Delhi Technological University
rishiroy_co21a6_45@dtu.ac.in

Priyanshu Gupta

2K21/CO/352, Delhi Technological University
priyanshugupta_co21a6_16@dtu.ac.in

Abstract

The project aims to develop a model that can generate captions for memes by understanding both the visual metaphors and the textual context within the memes. This submission template allows authors to submit their papers for review to an ACM Conference or Journal without any output design specifications incorporated at this point in the process.

Memes are a widely popular tool for web users to express their thoughts using visual metaphors. Understanding memes requires recognizing and interpreting visual metaphors with respect to the text inside or around the meme, often while employing background knowledge and reasoning abilities. We present a model of meme captioning which has worked with a dataset called MEMECAP. Our model works on a dataset containing 6.3K memes along with the title of the post containing the meme, the meme captions, the literal image caption, and the visual metaphors.

CCS CONCEPTS

- Machine learning → Computer vision
- Natural language processing → Text generation
- Deep Learning →Fine Tuning

Keywords

Meme captioning, multimodal model, computer vision, natural language processing, data augmentation

1. Introduction

The integration of computer vision and natural language processing (NLP) presents significant challenges, especially when interpreting visual metaphors in memes and generating coherent captions. This project develops a model to bridge this gap by leveraging background knowledge and domain-specific datasets. The model aims to predict an appropriate meme caption that encapsulates the visual metaphor and contextual information from the image, incorporating relevant semantic information from the provided image captions. It aims at creating a model which can generate meme captions for a given meme if the metaphors associated with the meme are available.

2. Dataset Description

2.1 Training Data

The training data consists of columns for meme category, image descriptions, ground truth meme captions, title, URL, image filename, metaphors, and post ID.

2.2 Test Data

The test data has a similar structure but lacks the meme captions, which need to be predicted by the model.

3. Methodology

3.1 Data Loading and Preprocessing

```
import json
import pandas as pd

# Load training data from JSON
with open('train.json', 'r') as file:
    train_data = json.load(file)

# Convert to Pandas DataFrame
train_df = pd.DataFrame(train_data)

# Select relevant columns
train_df = train_df[['meme_category', 'image_descriptions',
'meme_captions', 'title', 'url', 'image_filename', 'metaphors',
'post_id']]
```

- **Loading Data**: The training data is loaded from a JSON file and converted to a Pandas DataFrame.
- **Selecting Columns**: Columns of interest are extracted for further processing.

3.2 Custom Dataset Class

```
from torch.utils.data import Dataset
from PIL import Image

class MemeDataset(Dataset):
    def __init__(self, dataframe, transform=None):
        self.dataframe = dataframe
        self.transform = transform

    def __len__(self):
        return len(self.dataframe)

    def __getitem__(self, idx):
        img_name = self.dataframe.iloc[idx, 4]
        image = Image.open(img_name)
        caption = self.dataframe.iloc[idx, 2]

        if self.transform:
            image = self.transform(image)

        return image, caption
```

- **Initialization**: The dataset and processor are initialized.
- **Length**: Returns the number of items in the dataset.
- **Get Item**: Reads and processes the image and associated captions, encoding them for model input.

3.3 Model and Processor Initialization

```
from transformers import BlipProcessor, BlipForConditionalGeneration

# Initialize model and processor
processor =
BlipProcessor.from_pretrained('Salesforce/blip-image-captioning-base')
```

```
model =
BlipForConditionalGeneration.from_pretrained('Salesforce/blip-image-ca
ptioning-base')
```

- **Loading Pre-trained Model**: The BLIP model and its processor are loaded from the Hugging Face library.

3.4 Optimizer and Device Setup

```
import torch
from transformers import AdamW

# Set up optimizer
optimizer = AdamW(model.parameters(), lr=5e-5)

# Set device to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

- **Checking GPU Availability**: The code checks if a GPU is available and clears the GPU cache.
- **Optimizer**: AdamW optimizer is configured.
- **Device Setup**: The model is moved to the appropriate device (GPU or CPU).

3.5 Training Loop

```
from torch.utils.data import DataLoader

def train(model, dataloader, optimizer, device, num_epochs=5):
    model.train()
    for epoch in range(num_epochs):
        for images, captions in dataloader:
            images = images.to(device)
            captions = captions.to(device)

            # Forward pass
            outputs = model(images, captions=captions,
return_dict=True)
            loss = outputs.loss

            # Backward pass and optimization
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # Save model weights periodically
            if epoch % 5 == 0:
                torch.save(model.state_dict(),
f'model_epoch_{epoch}.pt')

            # Early stopping based on loss threshold
            if loss < 0.1:
                break

# Example DataLoader
dataloader = DataLoader(MemeDataset(train_df), batch_size=16,
shuffle=True)
train(model, dataloader, optimizer, device)
```

- **Training**: The model is set to training mode.
- **Epoch Loop**: Runs for a specified number of epochs (1 in this case).
- **Batch Processing**: For each batch, the input IDs and pixel values are extracted, and the model outputs are computed.
- **Loss Calculation**: The loss is printed and back propagated.
- **Optimizer Step**: The optimizer updates the model weights.
- **Early Stopping**: If the loss falls below a certain threshold (0.1), the loop breaks early.

3.6 Test Data Processing and Caption Generation

```
# Load test data
with open('test.json', 'r') as file:
    test_data = json.load(file)

test_df = pd.DataFrame(test_data)

# Generate captions for test images
results = []
for idx in range(len(test_df)):
    img_name = test_df.iloc[idx, 4]
    image = Image.open(img_name)

    inputs = processor(images=image, return_tensors="pt").to(device)
    outputs = model.generate(**inputs)
    caption = processor.decode(outputs[0], skip_special_tokens=True)

    results.append({
        "post_id": test_df.iloc[idx, 7],
        "generated_caption": caption
    })
```

- **Loading Test Data**: The test data is loaded from a JSON file and converted to a DataFrame.
- **Caption Generation**: For each image in the test set, the image is processed, and the model generates a caption, which is stored in the `outputs` list.
- **Progress Tracking**: Every 10 images, the index is printed.

```
# Save results to CSV
results_df = pd.DataFrame(results)
results_df.to_csv('generated_captions.csv', index=False)
```

- **Adding Captions**: The generated captions are added to the test DataFrame.
- **Saving to CSV**: The DataFrame is saved to a CSV file named `submission.csv`.

4. Results and Discussion

4.1 Training and Validation Performance

Discuss the model's performance on the training and validation datasets, including metrics such as loss and accuracy.

4.2 Caption Generation Quality

Evaluate the quality of generated captions using metrics like BLEU, ROUGE, and METEOR.

5. Recommendations for Improvement

5.1 Enhanced Data Augmentation

Implement techniques such as random cropping, flipping, rotation, and color jittering to increase model robustness.

5.2 Hyperparameter Tuning

Experiment with different learning rates, batch sizes, and the number of epochs to find the optimal training configuration.

5.3 Model Evaluation

Incorporate evaluation metrics (e.g., BLEU, ROUGE) to quantitatively assess the model's performance on the validation set.

5.4 Error Handling

Improve error handling in the caption generation loop to manage exceptions effectively.

5.5 Visualization

Visualize the generated captions alongside the images for better qualitative analysis.

5.6 Contextual Understanding

Enhance the model's contextual understanding by incorporating external knowledge sources and fine-tuning on domain-specific datasets.