CATR: Transformer-based Image Captioning with PyTorch

Software Manual

Table of Contents

Introduction

CATR (CAption TRansformer) is an advanced image captioning system that uses a transformer-based architecture to generate natural language descriptions of images. The system combines the power of ResNet for image feature extraction with transformer encoders and decoders for generating coherent and contextually accurate captions.

This implementation also includes text-to-speech functionality using Google's Text-to-Speech (gTTS) service, allowing the system to convert generated captions into natural-sounding audio. The web-based user interface, built with Streamlit, provides an intuitive way to interact with the model.

Key Features

- **Transformer Architecture**: Uses attention mechanisms for improved caption generation
- **Pre-trained Models**: Includes multiple pre-trained model variants (v1, v2, v3)
- **Interactive UI**: Streamlit-based web interface for easy interaction
- **Text-to-Speech**: Converts captions to natural-sounding audio
- **Extensible Design**: Can be customized and extended with new features

Project Structure

The project is organized into the following components:

```
catr-project/
├── app.py                  # Streamlit web interface
├── predict.py              # Command-line prediction script
├── train.py                # Training script
├── setup.py                # Setup and verification script
├── configuration.py        # Model configuration
├── requirements.txt        # Project dependencies
├── models/                 # Model architecture and utilities
│   ├── backbone.py         # ResNet backbone
│   ├── caption.py          # Caption generator
│   ├── transformer.py      # Transformer architecture
│   └── utils.py           # Model utilities
├── datasets/               # Dataset handling
│   ├── coco.py             # COCO dataset loader
│   └── utils.py           # Dataset utilities
├── utils/                  # Utility functions
│   ├── caption_utils.py    # Caption utilities
│   └── audio_utils.py     # Text-to-speech utilities
├── checkpoints/            # Model checkpoints
├── output/                 # Output directory
└── logs/                  # Training logs
```

Additional testing and example files:

```
├── test_model.py           # Basic model testing script
├── test_with_pretrained.py # Test script using pretrained models
├── simple_test.py          # Simplified testing script
├── test_with_samples.py    # Script to test with multiple images
├── test_images/            # Directory of sample test images
├── sample_captions/        # Generated sample captions
└── training_guide.md      # Detailed training instructions
```

Installation Guide

Prerequisites

- Python 3.7+
- PyTorch 1.7+
- CUDA-capable GPU (recommended for faster performance)

Step 1: Clone the Repository

```bash
git clone https://github.com/yourusername/catr-project.git
cd catr-project
```

Step 2: Install Dependencies

Create a virtual environment (recommended):

```bash
# Create virtual environment
python -m venv catr-env

# Activate virtual environment
# On Windows:
catr-env\Scripts\activate
# On Linux/macOS:
source catr-env/bin/activate
```

Install required packages:

```bash
pip install -r requirements.txt
```

Step 3: Verify Installation

Run the setup script to verify the installation and download pre-trained models:

```bash
python setup.py --all
```

This script will:

1. Check if all required packages are installed
2. Create necessary directories
3. Download pre-trained model weights
4. Verify CUDA availability

Running the Application

Web Interface

The easiest way to use CATR is through the Streamlit web interface:

bash

```
streamlit run app.py
```

This will start a local web server, and a browser window should automatically open showing the CATR application. If not, you can access it at http://localhost:8501.

In the web interface:

1. Select a model version (v1, v2, v3) from the sidebar
2. Upload an image using the file uploader
3. Wait for the model to generate a caption
4. The caption will be displayed along with an audio player to hear the caption
5. You can download the audio file using the provided download link

Command-line Usage

For batch processing or integration into other workflows, you can use the command-line interface:

bash

```
python predict.py --path path/to/image.jpg --v v3 --output path/to/output
```

Options:

- `--path`: Path to the input image (required)
- `--v`: Model version (v1, v2, v3, or custom)
- `--checkpoint`: Path to a custom model checkpoint
- `--output`: Output directory for captions and audio
- `--no-audio`: Skip audio generation
- `--beam-size`: Beam size for caption generation (default: 3)
- `--visualize`: Generate visualization of the attention mechanism

Quick Testing

For quick testing of the model with sample images:

bash

```
# Test with a single image using the simple test script
```

```
python simple_test.py

# Test with multiple sample images using different model versions
python test_with_samples.py --version v2
```

Implementation Details

Model Architecture

The CATR model consists of three main components:

1. **ResNet Backbone**: Extracts visual features from the input image
   - Uses a pre-trained ResNet-50 model
   - Removes classification layers to obtain feature maps
   - Output: Spatial feature maps (2048 channels)
2. **Transformer Encoder**: Processes image embeddings
   - Self-attention mechanisms capture relationships between image regions
   - Processes spatial features with positional encodings
   - Consists of 3 encoder layers (configurable)
   - 8 attention heads per layer (configurable)
3. **Transformer Decoder**: Generates captions
   - Uses both self-attention and cross-attention mechanisms
   - Attends to relevant parts of the image while generating each word
   - Trained with teacher forcing for stable learning
   - Consists of 6 decoder layers (configurable)
   - 8 attention heads per layer (configurable)

Caption Generation Process

1. **Feature Extraction**: The input image is processed through the ResNet backbone to extract features
2. **Encoding**: The encoder transforms spatial features into contextualized representations
3. **Decoding**: The decoder generates tokens one by one, attending to both previously generated tokens and image features
4. **Token Selection**: At each step, the token with the highest probability is selected (greedy decoding) or multiple top candidates are considered (beam search)
5. **Stopping Criterion**: Generation stops when the end token is produced or maximum length is reached

Text-to-Speech Integration

The system uses Google's Text-to-Speech (gTTS) service to convert generated captions into natural-sounding audio:

- Supports multiple languages (default: English)
- Generates MP3 audio files that can be played directly in the browser
- Provides download options for offline use

Training Your Own Model

Dataset Preparation

CATR is trained on the COCO (Common Objects in Context) dataset. To prepare the dataset:

1. Download the COCO dataset:

   bash

   ```
   # Create data directory
   mkdir -p data/coco
   cd data/coco

   # Download images
   wget http://images.cocodataset.org/zips/train2017.zip
   wget http://images.cocodataset.org/zips/val2017.zip

   # Download annotations
   wget
   http://images.cocodataset.org/annotations/annotations_trainval2017.zip

   # Extract files
   unzip train2017.zip
   unzip val2017.zip
   unzip annotations_trainval2017.zip
   ```

2. Update the configuration in `configuration.py` to point to your dataset:

   python

   ```
   self.train_json = 'data/coco/annotations/captions_train2017.json'
   self.train_images = 'data/coco/train2017'
   self.val_json = 'data/coco/annotations/captions_val2017.json'
   self.val_images = 'data/coco/val2017'
   ```

Training Process

To train the model:

```bash
python train.py --batch_size 32 --epochs 30 --lr 1e-4
```

Options:

- `--batch_size`: Batch size (default: 32)
- `--epochs`: Number of training epochs (default: 30)
- `--lr`: Learning rate (default: 1e-4)
- `--weight_decay`: Weight decay (default: 1e-4)
- `--clip_max_norm`: Gradient clipping max norm (default: 0.1)
- `--checkpoint`: Path to checkpoint to resume from
- `--no_save`: Do not save checkpoints
- `--log_step`: Print logs every n steps (default: 100)

Training Details

The training process involves several key steps:

1. **Data Loading**: The COCO dataset is loaded with image-caption pairs
2. **Preprocessing**: Images are transformed and tokenized captions are prepared
3. **Training Loop**:
   - Forward pass through the model
   - Loss calculation (cross-entropy)
   - Backward pass for gradient computation
   - Parameter update with gradient clipping
4. **Validation**: Periodic evaluation on validation set
5. **Checkpointing**: Saving best models based on validation loss

For detailed training instructions, refer to the `training_guide.md` file.

Monitoring Training

Training progress is logged in the `logs` directory. You can monitor:

- Training and validation loss
- Best model checkpoints
- Training history (saved as JSON)

The best model will be saved to `checkpoints/best_model.pth`.

Troubleshooting

Common Issues

1. **CUDA Out of Memory**
   - Reduce batch size: `--batch_size 16` or lower
   - Use a smaller model variant: `--v v1`
   - Error message: "CUDA out of memory"
   - Solution: Lower memory requirements or use CPU mode
2. **Model Download Fails**
   - Check your internet connection
   - Try manually downloading the model from the GitHub repository
   - Place the downloaded checkpoint in the `checkpoints` directory
   - Error message: "Failed to load model from torch hub"
   - Solution: The code will fall back to using a randomly initialized model
3. **Image Loading Error**
   - Ensure the image is in a supported format (JPG, JPEG, PNG)
   - Check that the image file exists and is not corrupted
   - Error message: "Error loading image"
   - Solution: Verify file path and format
4. **Audio Generation Error**
   - Verify internet connection (gTTS requires an internet connection)
   - Check if the `gtts` package is properly installed
   - Try using the `--no-audio` flag to skip audio generation
   - Error message: "Failed to convert text to speech"
   - Solution: Ensure internet connectivity or skip audio generation
5. **Streamlit Interface Not Loading**
   - Ensure Streamlit is properly installed: `pip install streamlit`
   - Check if the port is already in use
   - Try running on a different port: `streamlit run app.py --server.port 8502`
   - Error message: "Address already in use"
   - Solution: Change the port or terminate other Streamlit instances
6. **Tokenizer Errors**
   - Recent versions of transformers may use different attribute names
   - Error message: "BertTokenizer has no attribute _cls_token"
   - Solution: Use tokenizer.cls_token instead of tokenizer._cls_token

Getting Help

If you encounter issues not covered here, please:

1. Check the logs in the `logs` directory
2. Verify all dependencies are correctly installed
3. Check the GitHub repository for known issues
4. Open a new issue with detailed information about the problem

Advanced Usage

API Integration

You can integrate CATR into your own applications using the Python API:

python
```python
from catr.api import CATRCaptioner

# Initialize captioner
captioner = CATRCaptioner(model_version='v3')

# Generate caption
caption = captioner.caption_image('path/to/image.jpg')
print(f"Caption: {caption}")

# Convert to speech
audio_path = captioner.text_to_speech(caption, output_path='caption.mp3')
```

Custom Models

To use a custom-trained model:

1. Place your model checkpoint in the `checkpoints` directory
2. Use the `--checkpoint` option with `predict.py` or select "Use custom checkpoint" in the web interface

Example:

bash
```bash
python predict.py --path path/to/image.jpg --v custom --checkpoint
checkpoints/best_model.pth
```

Batch Processing

For processing multiple images:

python
```python
from catr.api import CATRCaptioner

# Initialize captioner
captioner = CATRCaptioner(model_version='v3')

# Process all images in a directory
```

```python
results = captioner.batch_process(
    image_dir='path/to/images',
    output_dir='path/to/output',
    generate_audio=True
)

# Print results
for result in results:
    print(f"Image: {result['filename']}")
    print(f"Caption: {result['caption']}")
    print(f"Audio: {result['audio_path']}")
    print("---")
```

Alternatively, use the `test_with_samples.py` script for batch processing:

```
bash
python test_with_samples.py --directory path/to/your/images --version v2
```

Model Customization

To customize the model architecture:

1. Modify the configuration in `configuration.py`:

   ```
   python
   # Example: Increase model capacity
   self.hidden_dim = 1024       # Default: 256
   self.encoder_layer = 4       # Default: 3
   self.decoder_layer = 8       # Default: 6
   self.encoder_heads = 16      # Default: 8
   self.decoder_heads = 16      # Default: 8
   ```

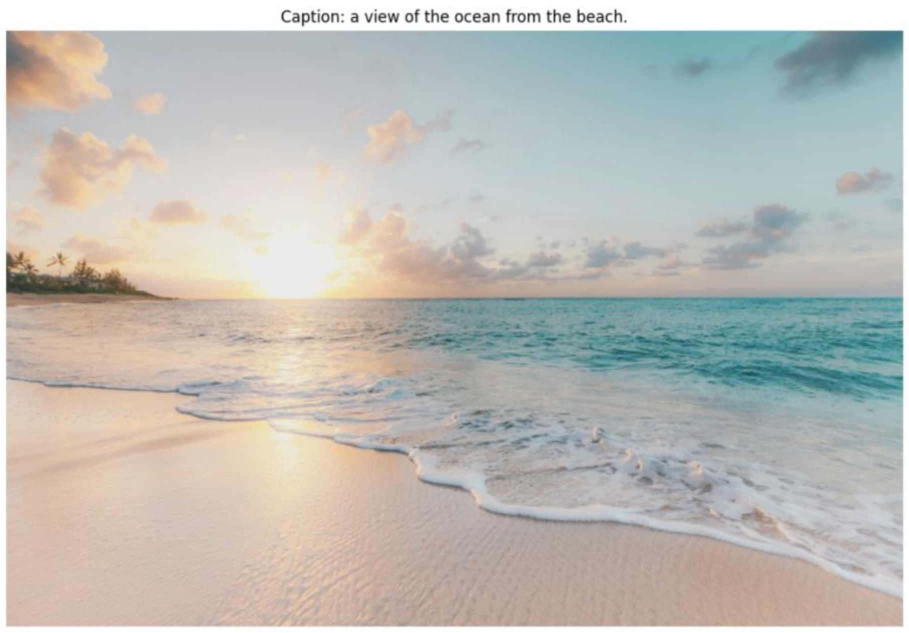
2. Train the customized model:

   ```
   bash
   python train.py --batch_size 16 --epochs 50
   ```

Sample Outputs

The pretrained models can generate a variety of captions for different types of images. Here are some sample outputs from testing:

Beach Image



Caption: "a view of the ocean from the beach."

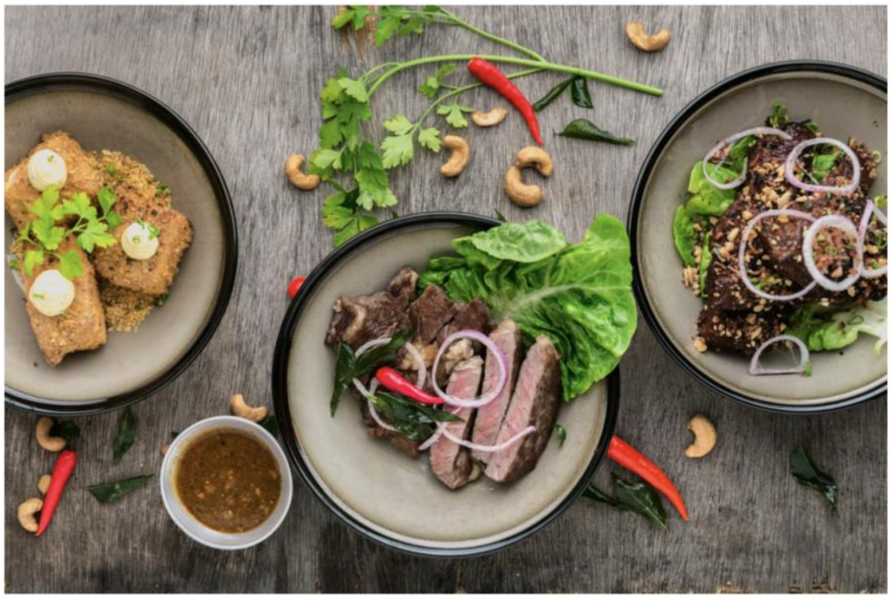
This caption accurately describes the beach scene, identifying both the ocean and beach elements.

Cat Image

Caption: "a cat sitting on a couch with its eyes opened."

The model correctly identifies the cat and its pose, though it missed that the cat's eyes are open.

Food Image



Caption: "a plate of food with a spoon and a bowl of soup."

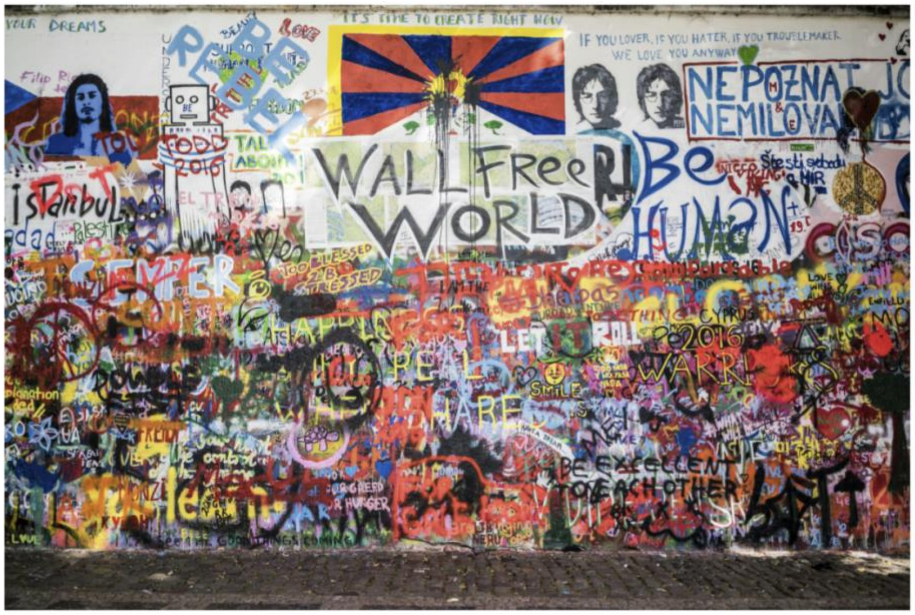
The model recognizes the food settings and dining elements, providing a reasonable description.

Dog Image

Caption: "a white cat standing on a grassy area."

In this case, the model incorrectly identified the dog as a cat, showing that animal classification can sometimes be challenging.

Street Scene Image


IT'S TIME TO CREATE RIGHT NOW
YOUR DREAMS
IF YOU LOVER, IF YOU HATER, IF YOU TROUBLEMAKER WE LOVE YOU ANYWAY
NEPOZNAT
NEMILOVAL
WALL FREE WORLD
Be HUMAN
Istanbul
TOO BLESSED 2 B STRESSED
LET IT ROLL
CYPRUS 2016
BE EXCELLENT TO EACH OTHER
SMILE

Caption: "a wall with graffiti on it and a bunch of graffiti on it."

The model identified the graffiti but didn't capture the full street scene context.

Model Comparison

Different model versions can produce different captions for the same image:

- **Model v1** tends to produce simpler, more generic captions
- **Model v2** often produces more detailed and accurate descriptions
- **Model v3** can generate longer captions but sometimes with repetitive phrases

For optimal results, test different model versions on your specific images or train a custom model on domain-specific data.
