Faculty of Computers and Information

EELU

Department of Computer Science

Project Title: Image Caption Generator

CNN & LSTM Image Captioning Code

This documentation provides an overview of the provided Python code, which implements an image captioning model using a Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) network. The code processes images from the Flickr8k and COCO datasets, extracts features using VGG16, and generates captions. Below, the code is divided into key sections with explanations.

1. Loading and Preprocessing VGG16 Model

The code begins by loading the VGG16 model, a pre-trained CNN, to extract image features.

- **Purpose**: Extracts high-level features from images to be used as input for the captioning model.
- **Key Components**:
  - Loads VGG16 with custom weights from a specified path.
  - Creates a new model (model_vgg) that outputs features from the second-to-last layer (fully connected layer) of VGG16.
  - Input images are resized to 224x224 pixels and preprocessed using preprocess_input.

```
[  base_model = VGG16(weights=None)

weights_path =
'/kaggle/input/backup/keras/default/6/vgg16_weights_tf_dim_ordering_tf_kernels.h5'

base_model.load_weights(weights_path)   ]
```

2. Loading and Mapping Dataset Captions

The code processes captions from Flickr8k and COCO datasets, creating mappings between image IDs and their captions.

- **Purpose**: Organizes image captions for training and evaluation.
- **Key Components**:
    - **Flickr8k Captions**: Reads captions from a text file, splits each line into image name and caption, and maps image IDs to lists of captions.
    - **COCO Captions**: Loads captions from a JSON file, extracts image IDs and captions, and creates a similar mapping.
    - **Combined Mapping**: Merges both datasets into a unified dictionary (combined_mapping) with unique image IDs prefixed by "flickr_" or "coco_". Saves the mapping to a pickle file.
    - **Image File Mapping**: Creates dictionaries (combined_image_id_to_filename and filename_to_image_id) to map image IDs to file paths and filenames.

```
   [# Example for Flickr8k caption loading
def load_flickr8k_captions(file_path):
  mapping = {}
  with open(file_path, 'r') as f:
    for line in f:
      tokens = line.strip().split()
      if len(tokens) < 2:
        continue
      image_name, caption = tokens[0], ' '.join(tokens[1:])
      image_id = image_name.split('.')[0]
      if image_id not in mapping:
        mapping[image_id] = []
      mapping[image_id].append(caption) , return mapping ]
```

3. Feature Extraction

The code extracts features from images using the VGG16 model.

- **Purpose**: Converts images into feature vectors for the captioning model.
- **Key Components**:
    - Iterates through images in combined_image_id_to_filename.
    - Loads each image, resizes it to 224x224, converts it to an array, and preprocesses it.
    - Uses model_vgg to predict features (4096-dimensional vectors) and stores them in a dictionary (features).
    - Saves features to a pickle file (combined_features.pkl).

```
[def extract_features(combined_image_id_to_filename):
    features = {}
    for image_id, (image_dir, filename) in tqdm(combined_image_id_to_filename.items()):
        img_path = os.path.join(image_dir, filename)
        image = load_img(img_path, target_size=(224, 224))
        image = img_to_array(image)
        image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
        image = preprocess_input(image)
        feature = model_vgg.predict(image, verbose=0)
        features[image_id] = feature
    return features
]
```

4. Text Preprocessing and Tokenization

The code prepares captions for the LSTM model by tokenizing and padding them.

- **Purpose**: Converts captions into numerical sequences for training.
- **Key Components**:
  - Collects all captions, adds startseq and endseq tokens, and tokenizes them using Tokenizer.
  - Saves the tokenizer to a pickle file.
  - Calculates vocabulary size (vocab_size) and maximum caption length (maxLength).
  - Splits data into training (80%) and validation (20%) sets.

```
        [all_captions = [f"startseq {caption} endseq" for caption in all_captions]

tokenizer = Tokenizer()

tokenizer.fit_on_texts(all_captions)

vocab_size = len(tokenizer.word_index) + 1

maxLength = max(len(caption.split()) for caption in all_captions)

]
```

5. Model Architecture and Training

The code defines and trains a CNN-LSTM model for caption generation.

- **Purpose**: Combines image features and partial captions to predict the next word in a sequence.
- **Key Components**:
    - **Model Architecture**:
        - **Image Input**: Takes 4096-dimensional VGG16 features, applies dropout (0.3), and reduces to 256 dimensions via a dense layer.
        - **Text Input**: Takes padded sequences, embeds them into 256-dimensional vectors, applies dropout (0.3), and processes them with an LSTM (256 units).
        - **Decoder**: Merges image and text features, passes them through a dense layer (256 units, ReLU), and outputs probabilities over the vocabulary.
    - **Training**:
        - Uses a data generator to yield batches of image features, input sequences, and one-hot-encoded output words.
        - Compiles with Adam optimizer (learning rate 0.0003) and categorical crossentropy loss.
        - Includes early stopping (patience=3) and model checkpointing to save the best model.
        - Trains for up to 40 epochs with batch size 8.
    - Plots training and validation loss curves.

```
[input1 = Input(shape=(4096,))

fe1 = Dropout(0.3)(input1)

fe2 = Dense(256, activation='relu')(fe1)

input2 = Input(shape=(maxLength,))

se1 = Embedding(vocab_size, 256, mask_zero=True)(input2)

se2 = Dropout(0.3)(se1)
se3 = LSTM(256)(se2)

decoder1 = Add()([fe2, se3])

decoder2 = Dense(256, activation='relu')(decoder1)

outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[input1, input2], outputs=outputs)

]
```

6. Evaluation and Caption Generation

The code evaluates the model and generates captions for test images.

- **Purpose**: Assesses model performance and generates captions for new images.
- **Key Components**:
  - **Caption Prediction**: Uses predictCaption to generate captions by iteratively predicting words until endseq or maxLength is reached.
  - **BLEU Scores**: Computes BLEU-1 and BLEU-2 scores to evaluate generated captions against ground truth.
  - **Visualization**: Displays images with actual and predicted captions using generateCaption.
  - Handles errors for missing images, features, or model components.

```
[def predictCaption(model, image_feature, tokenizer, maxLength):

  in_text = 'startseq'

  for _ in range(maxLength):

    sequence = tokenizer.texts_to_sequences([in_text])[0]

    sequence = pad_sequences([sequence], maxlen=maxLength, padding='post')

    yhat = model.predict([image_feature, sequence], verbose=0)
```

```
        yhat = np.argmax(yhat)
        word = idxToWord(yhat, tokenizer)
        if word is None or word == 'endseq':
            break
        in_text += ' ' + word
    return in_text.replace('startseq ', '').replace(' endseq', '')]
```

Summary

The code implements an image captioning system using VGG16 for feature extraction and an LSTM-based model for caption generation. It processes Flickr8k and COCO datasets, handles data preprocessing, trains the model with a custom data generator, and evaluates performance using BLEU scores. The generateCaption function allows testing on individual images, displaying both actual and predicted captions.

CNN & LSTM with Attention Image Captioning Code

This documentation provides an overview of the Python code implementing an image captioning model using EfficientNetB0 for feature extraction, LSTM for sequence modeling, and a custom Bahdanau Attention mechanism. The code processes the Flickr8k dataset, extracts image features, and generates captions with beam search. Below, the code is divided into key sections with explanations.

1. Feature Extraction with EfficientNetB0

The code uses EfficientNetB0 to extract image features, incorporating data augmentation to enhance robustness.

- **Purpose**: Extracts compact feature representations (1280-dimensional vectors) from images for the captioning model.
- **Key Components**:
    - Loads EfficientNetB0 with ImageNet weights, excluding the top classification layer, and uses global average pooling.
    - Applies data augmentation (rotation, flipping, brightness adjustment) using ImageDataGenerator.
    - Processes images by resizing to 224x224, applying augmentation, and preprocessing with preprocess_input.
    - Saves extracted features to a pickle file (efficientnetb0_features.pickle) to avoid redundant computation.
    - Checks for existing features to skip extraction if already done.

```
[ base_model = EfficientNetB0(weights='imagenet', include_top=False,
pooling='avg')

model = Model(inputs=base_model.input,
outputs=base_model.output)

datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2]
)

def extract_features(image_path, model, datagen):
    img = load_img(image_path, target_size=(224, 224))
    img = img_to_array(img)
    img = datagen.random_transform(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    features = model.predict(img, verbose=0)
    return features]
```

2. Caption Loading and Preprocessing

The code loads and processes captions from the Flickr8k dataset, preparing them for training.

- **Purpose**: Organizes captions and converts them into tokenized sequences.
- **Key Components**:

    - Reads captions from captions.txt using pandas and groups them by image name.
    - Cleans captions by converting to lowercase, removing special characters, and adding startseq and endseq tokens.
    - Creates a tokenizer to convert captions to numerical sequences, saving it to tokenizer.pickle.
    - Computes vocabulary size (vocab_size) and maximum caption length (max_length).
    - Splits data into training (6000 images) and testing sets.

```
[def clean_caption(caption):
   caption = caption.lower()
   caption = ''.join(c for c in caption if c.isalnum() or c.isspace())
   caption = ' '.join(caption.split())
   caption = 'startseq ' + caption + ' endseq'
   return caption
tokenizer = Tokenizer(oov_token='<OOV>')
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(caption.split()) for caption in all_captions)]
```

3. Model Architecture with Attention

The code defines a CNN-LSTM model with a custom Bahdanau Attention layer for caption generation.

- **Purpose**: Combines image features and partial captions to generate the next word, focusing on relevant image regions via attention.
- **Key Components**:

- **Image Input**: Takes 1280-dimensional EfficientNetB0 features, applies dropout (0.5), and projects to 512 dimensions.
- **Text Input**: Embeds tokenized sequences into 512-dimensional vectors, processes them with two LSTM layers (512 units each, return sequences for the first).
- **Attention**: Implements BahdanauAttention to compute a context vector by weighting image features based on the LSTM hidden state.
- **Decoder**: Concatenates the context vector and LSTM output, passes through a dense layer (512 units, ReLU), and outputs probabilities over the vocabulary.
- Uses L2 regularization and an exponential decay learning rate schedule (initial rate 0.0005).

```
[class BahdanauAttention(Layer):

  def __init__(self, units):

    super(BahdanauAttention, self).__init__()

    self.W1 = Dense(units, kernel_regularizer=l2(0.01))

    self.W2 = Dense(units, kernel_regularizer=l2(0.01))

    self.V = Dense(1)

  def call(self,


 features, hidden):

    hidden_with_time_axis = Reshape((1, -1))(hidden)

    score = self.V(tf.nn.tanh(self.W1(features) +
self.W2(hidden_with_time_axis)))

    attention_weights = tf.nn.softmax(score, axis=1)

    context_vector = attention_weights * features

    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector
inputs1 = Input(shape=(1280,))

fe1 = Dropout(0.5)(inputs1)

fe2 = Dense(512, activation='relu', kernel_regularizer=l2(0.01))(fe1)

fe3 = Reshape((1, 512))(fe2)

inputs2 = Input(shape=(max_length,))

se1 = Embedding(vocab_size, 512, mask_zero=True)(inputs2)

se2 = Dropout(0.5)(se1)

se3 = LSTM(512, return_sequences=True, use_cudnn=False)(se2)

se4 = LSTM(512, return_sequences=False, use_cudnn=False)(se3)

attention = BahdanauAttention(512)

context_vector = attention(fe3, se4)

concat = Concatenate()([context_vector, se4])

decoder1 = Dense(512, activation='relu',
kernel_regularizer=l2(0.01))(concat)

outputs = Dense(vocab_size, activation='softmax')(decoder1)]
```

4. Training and Data Pipeline

The code sets up a data pipeline and trains the model using a custom dataset.

- **Purpose**: Efficiently feeds data to the model and optimizes performance.
- **Key Components**:
    - **Data Generator**: Yields batches of image features, input sequences, and one-hot-encoded output words, shuffling data for training.
    - **tf.data.Dataset**: Wraps the generator in a TensorFlow dataset, enabling caching and prefetching for performance.
    - Trains for 80 epochs with batch size 64, using early stopping (patience=20) based on validation loss.

- Saves the trained model to image_captioning_model.h5.
- Plots training and validation loss/accuracy curves.

```
[def create_dataset(descriptions, photos, tokenizer, max_length, vocab_size, batch_size):
    dataset = tf.data.Dataset.from_generator(
        lambda: data_generator(descriptions, photos, tokenizer, max_length, vocab_size, batch_size),
        output_signature=(
            (
                tf.TensorSpec(shape=(None, 1280), dtype=tf.float32),
                tf.TensorSpec(shape=(None, max_length), dtype=tf.int32)
            ),
            tf.TensorSpec(shape=(None, vocab_size), dtype=tf.float32)
        )
    )
    return dataset.cache().prefetch(tf.data.AUTOTUNE)
train_dataset = create_dataset(train_descriptions, features, tokenizer, max_length, vocab_size, batch_size)
history = model.fit(
    train_dataset,
    epochs=80,
    steps_per_epoch=steps_per_epoch,
    validation_data=test_dataset,
    validation_steps=validation_steps,
    callbacks=[early_stopping]
)]
```

5. Caption Generation and Evaluation

The code generates captions using beam search and evaluates the model on test images.

- **Purpose**: Produces human-readable captions and visualizes results.
- **Key Components**:
  - **Beam Search**: Generates captions by maintaining the top beam_width (5) sequences, scoring them based on log probabilities.
  - **Testing Function**: Displays up to 5 test images with generated and actual captions, using matplotlib for visualization.
  - Handles edge cases (e.g., missing tokens) and removes startseq/endseq tokens from output.
  - Saves training history plot to training_history.png.

```
[def generate_caption(model, tokenizer, photo, max_length, beam_width=5):

  photo = np.squeeze(photo)

  photo = np.expand_dims(photo, axis=0)

  start = [tokenizer.word_index['startseq']]

  sequences = [[start, 0.0]]

  for _ in range(max_length):

    all_candidates = []

    for seq, score in sequences:

      padded_seq = pad_sequences([seq], maxlen=max_length)

      yhat = model.predict([photo, padded_seq], verbose=0)

      top_words = np.argsort(yhat[0])[-beam_width:]

      for word_idx in top_words:

        new_seq = seq + [word_idx]

        new_score = score - np.log(yhat[0, word_idx] + 1e-8)
```

```
            all_candidates.append([new_seq, new_score])
        sequences = sorted(all_candidates, key=lambda x: x[1])[:beam_width]
        if sequences[0][0][-1] == tokenizer.word_index.get('endseq', 0):
            break
    best_seq = sequences[0][0]
    caption = ' '.join([tokenizer.index_word.get(idx, '') for idx in best_seq if idx
in tokenizer.index_word])
    return caption.replace('startseq', '').replace('endseq', '').strip()]
```

Summary

The code implements an advanced image captioning system using EfficientNetB0 for feature extraction, LSTM with Bahdanau Attention for sequence modeling, and beam search for caption generation. It processes the Flickr8k dataset, applies data augmentation, and uses a robust data pipeline with tf.data.Dataset. The model is trained with regularization and early stopping, and results are visualized with generated captions compared to ground truth.

CNN & LSTM with Beam Search Image Captioning Code

This documentation outlines a Python script that implements an image captioning model using VGG16 for feature extraction, LSTM for sequence modeling, and beam search for caption generation. The code processes the Flickr8k dataset to generate descriptive captions for images. Below, the code is divided into key sections with explanations.

1. Caption Loading and Tokenization

The code loads and preprocesses captions from the Flickr8k dataset, preparing them for training.

- **Purpose: Organizes captions and converts them into tokenized sequences for the LSTM model.**
- **Key Components:**
    - **Reads captions from captions.txt using pandas, adding startseq and endseq tokens to each caption.**
    - **Uses Tokenizer to convert captions into numerical sequences, computing the vocabulary size (vocab_size) and maximum caption length (max_length).**
    - **Saves the tokenizer to tokenizer_full.pkl for later use.**

```
[captions = pd.read_csv('/kaggle/input/flickr8k/captions.txt')

captions.coluxmns = ['image', 'caption']

captions['caption'] = captions['caption'].apply(lambda x: 'startseq ' + x +
' endseq')
tokenizer = Tokenizer()

tokenizer.fit_on_texts(captions['caption'])

vocab_size = len(tokenizer.word_index) + 1

max_length = max(len(c.split()) for c in captions['caption'])]
```

2. Image Feature Extraction with VGG16

The code extracts image features using the VGG16 model.

- **Purpose**: Converts images into feature vectors for input to the captioning model.
- **Key Components**:
  - Loads VGG16 with ImageNet weights, excluding the top layers, and uses global average pooling to produce 4096-dimensional feature vectors.
  - Processes each unique image by resizing to 224x224, applying preprocess_input, and extracting features.
  - Stores features in a dictionary (image_features) and saves them to image_features_full.npy.

```
[vgg_model = VGG16(weights='imagenet', include_top=False, pooling='avg')

for img_name in tqdm(captions['image'].unique(), desc="Extracting image features"):

    img_path = os.path.join(image_dir, img_name)

    if os.path.exists(img_path):

        img = load_img(img_path, target_size=(224, 224))
        img = img_to_array(img)

        img = preprocess_input(img)

        img = np.expand_dims(img, axis=0)

        feature = vgg_model.predict(img, verbose=0)

        image_features[img_name] = feature.flatten()

np.save("image_features_full.npy", image_features)]
```

3. Model Architecture and Training

The code defines and trains a CNN-LSTM model for caption generation.

- **Purpose**: Combines image features and partial captions to predict the next word in a sequence.
- **Key Components**:
  - **Model Architecture**:
    - **Image Input**: Takes 4096-dimensional VGG16 features, applies dropout (0.5), and projects to 256 dimensions via a dense layer.
    - **Text Input**: Embeds tokenized sequences into 256-dimensional vectors, applies dropout (0.5), and processes them with an LSTM (256 units).
    - **Decoder**: Merges image and text features using add, passes through a dense layer (256 units, ReLU), and outputs probabilities over the vocabulary.
  - **Training Data Preparation**:

        - Converts captions to sequences, generating input-output pairs for each prefix of a caption.
        - Uses pad_sequences to ensure consistent input length and prepares image features, input sequences, and target words.
    - **Training**: Compiles with Adam optimizer and sparse categorical crossentropy loss, training for 10 epochs with a batch size of 256.
    - Saves the model to caption_model_full.h5.

```
[inputs1 = Input(shape=(4096,))

fe1 = Dropout(0.5)(inputs1)

fe2 = Dense(256, activation='relu')(fe1)

inputs2 = Input(shape=(max_length,))

se1 = Embedding(vocab_size, embedding_dim,
mask_zero=True)(inputs2)

se2 = Dropout(0.5)(se1)

se3 = LSTM(256)(se2)

decoder1 = add([fe2, se3])

decoder2 = Dense(256, activation='relu')(decoder1)

outputs = Dense(vocab_size, activation='softmax')(decoder2)

caption_model = Model(inputs=[inputs1, inputs2],
outputs=outputs)

caption_model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam')
caption_model.fit([X1, X2], y, epochs=10, batch_size=256,
verbose=1)]
```

4. Caption Generation with Beam Search

The code implements beam search to generate captions for new images.

- **Purpose**: Produces high-quality captions by exploring multiple sequence possibilities.
- **Key Components**:
    - Uses generate_desc_beam to generate captions with a beam width of 3, keeping the top 3 sequences at each step.
    - Predicts word probabilities, updates sequence scores with log probabilities, and selects the best sequence.
    - Removes startseq and endseq tokens from the final caption and joins words into a readable string.

```
 [def generate_desc_beam(model, tokenizer, photo, max_length,
beam_index=3):

  start = [tokenizer.word_index['startseq']]

  start_word = [[start, 0.0]]

  while len(start_word[0][0]) < max_length:

    temp = []

    for s in start_word:

      sequence = pad_sequences([s[0]], maxlen=max_length)

      preds = model.predict([photo, sequence], verbose=0)
```

```
            word_preds = np.argsort(preds[0])[-beam_index:]
            for w in word_preds:
                next_seq = s[0] + [w]
                prob = s[1] + np.log(preds[0][w] + 1e-10)
                temp.append([next_seq, prob])
        start_word = sorted(temp, reverse=False, key=lambda l: l[1])
        start_word = start_word[-beam_index:]
    final_seq = start_word[-1][0]
    final_caption = [tokenizer.index_word[i] for i in final_seq if i in
tokenizer.index_word]
    final_caption = final_caption[1:-1]  # remove startseq and
endseq
    return ' '.join(final_caption)]
```

Summary

The code implements a straightforward image captioning system using VGG16 for feature extraction and an LSTM-based model for generating captions. It processes the Flickr8k dataset, extracts image features, trains a model with a simple CNN-LSTM architecture, and uses beam search for caption generation. The model is trained efficiently, and the tokenizer and model are saved for future use.