1. Details on How does the system take inputs and outputs.

Input Processing:

1. Image Data Handling:

Image Reading: Images are read from a `.parquet` file containing image data in byte format. The pandas library is used to load this data into a DataFrame.

Image Preprocessing: Each image byte data is converted into an actual image, resized to `224x224` pixels, and preprocessed to suit the ResNet50 model requirements. This preprocessing includes RGB conversion and normalization specific to how the ResNet50 model was trained.

2. Text Data Handling:

Text Extraction: Text descriptions associated with each image are extracted from the DataFrame.

Text Preprocessing: Text data is converted to lowercase and framed with special tokens (startofseq and endofseq) to indicate the beginning and end of a text sequence, which helps the model learn the sequence context better.

Output Handling:

1. Feature Extraction:

ResNet50 Model: This pre-trained model is used to extract features from the preprocessed images. Only the features from the penultimate layer of the model are used, which provides a rich representation of the image content without the final classification layer specific to the original ResNet50 training.

2. Text Encoding:

Vocabulary Creation: A vocabulary dictionary is created where each unique word in the text corpus is assigned a unique integer index.

Text Vectorization: Texts are converted into sequences of integers based on this vocabulary. This numerical representation of text is necessary for training the LSTM model.

Training Data Preparation:

Generator Function: A custom generator function is employed to handle large datasets efficiently by loading data in batches. This generator prepares batches of image features and corresponding text sequences in a format suitable for training:

Image Features: Features extracted from the images.

Input Text Sequences: Partial text sequences that the model uses as input (e.g., the first few words of a caption).

Output Text Sequences: The next word in the sequence that the model needs to predict, often one-hot encoded to fit the classification framework of the output layer.

Model Architecture:

Combined Model: The model architecture integrates image feature extraction and text processing. It uses concatenated outputs from a Dense layer applied on image features and an LSTM layer processing the text. This concatenated output is then passed through additional LSTM layers to predict the next word in the caption sequence.

Model Training and Output Generation:

Training: The combined model is trained using the batches of input image features and text sequences generated by the custom data generator. Model performance is monitored using callbacks like ModelCheckpoint and EarlyStopping.

Prediction: For generating captions, the model uses extracted image features and starts with the startofseq token to generate one word at a time until it predicts the endofseq token or reaches the maximum caption length.

2. Number of layers applied to the system

1. ResNet50 Model: This is a pre-built model from TensorFlow's Keras applications, which is primarily used for image feature extraction. The original ResNet50 model consists of 50 layers, including convolutional layers, activation layers, batch normalization layers, and a fully connected layer at the end.

2. Image Processing Model:

1. Dense Layer: Converts the 2048 extracted features into an embedding of size 128.

2. RepeatVector Layer: Repeats the feature vector to match the sequence length expected by the LSTM layers.

3. Caption Generating Model:

Embedding Layer: Maps each integer-encoded vocabulary word to a vector space.

LSTM Layer: First LSTM layer with 256 units, returns sequences to feed into the next LSTM layer.

TimeDistributed Layer: Applies a Dense layer to each item in the sequence.

4. Final Model Combination:

Concatenate Layer: Merges the outputs of the image processing model and the caption generating model.

LSTM Layer: A 128-unit LSTM that processes the concatenated output and returns sequences.

LSTM Layer: A 512-unit LSTM that processes the output of the previous LSTM layer.

Dense Layer: The final output layer that predicts the next word in the caption.

The models are built sequentially and then combined into a final model that integrates both the image features and text processing capabilities. The final combined model is used for generating the product descriptions based on the image features.

In total, aside from ResNet50 we have:

- 3 layers in the image model.

- 3 layers in the caption model.

- 4 layers in the final combined model.

This makes a total of 10 custom layers added to the pre-existing layers of the ResNet50 model.

3. ⁠an overview of the system

The system is designed to generate textual descriptions for fashion products based on their images. This process involves several stages of development, including image preprocessing, feature extraction, text preprocessing, model building, and training.

Image Preprocessing and Feature Extraction

The system begins by loading images from a dataset, which are stored in a format that requires decoding (bytes format in this case). These images are preprocessed to a standard size and color format suitable for the ResNet50 model, a pre-trained deep neural network known for its efficacy in extracting robust features from images. The ResNet50 model processes the images to produce a set of features for each image, simplifying the complex visual information into a form that can be efficiently used for further processing.

Text Preprocessing

Parallel to image processing, the system handles textual data associated with each image. Text descriptions (captions) are preprocessed by converting them into a lowercased format, appending special tokens to denote the start and end of a sequence. This standardization helps in maintaining consistency in data handling and prepares the text for encoding and decoding during model training.

Model Architecture

The core of the system is a composite neural network model that integrates image features and textual data. This model has two main components:

1. An image processing model that uses a dense layer to refine the image features into a compact form.

2. A caption generating model that includes an embedding layer for textual input, followed by LSTM layers which are effective in handling sequence data, helping the model to learn the context and sequence of words in captions.

These two components are merged using a concatenation layer, allowing the model to simultaneously consider both image features and partial captions to predict the next word in a caption. This setup uses additional LSTM layers post-merge to enhance learning from the combined data.

Training and Output

The training process involves feeding the model with batches of image features and corresponding text sequences, where the model learns to predict captions word by word. The output of the model is a sequence of words that forms a coherent caption, describing the image in a context similar to the training data.

This system, built using TensorFlow and Keras, represents a comprehensive approach to automated text generation based on visual input, highlighting the intersection of computer vision and natural language processing technologies.

4. Details on how the ResNet50 works with the system

The ResNet50 model plays a crucial role in extracting meaningful features from images, which are then used to generate textual descriptions. ResNet50, a variant of the Residual Network architecture developed by Microsoft, is specifically designed to handle very deep networks through the use of skip connections or shortcut connections.

Integration of ResNet50 in the System

The system utilizes ResNet50 primarily for its capability to process and transform images into a high-level, compressed representation. In the context of this system, the images are first preprocessed to align with the input requirements of ResNet50, which involves resizing the images to 224x224 pixels and converting them to RGB color space. This standardization is crucial as it ensures the images are in a form that the pre-trained network can effectively analyze.

Once the images are prepared, they are passed through the ResNet50 model. However, in this specific application, the top layer of ResNet50, which is usually used for classifying images into 1000 different categories, is not used. Instead, the output just before the final classification layer is extracted. This output serves as a dense representation of the image's features, capturing various aspects of the image that are significant for understanding its content without being tied to specific class labels.

Feature Extraction

The features extracted from ResNet50 are a vector of 2048 elements for each image. These elements represent a distilled version of the image's visual content, encoded in a form that a neural network can process. By utilizing a pre-trained model like ResNet50, the system leverages learned patterns applicable across a wide range of visual contexts, which is beneficial given the diverse nature of fashion product images.

Role in the Captioning Model

These extracted features become the foundational input for the next stage of the system—the caption generation model. The dense feature vector from ResNet50 is first processed through a dense layer to transform these features into a more suitable format for sequential processing, aligning the dimensions for integration with text data. This transformation is critical as it bridges the gap between raw image features and the textual data that the model will generate.

7. Accuracy of the system

0.99213

8. ⁠How the model was trained

Preparation of Training Data

The initial step involves preparing the training data, which includes both the image features extracted using the ResNet50 model and the corresponding text descriptions. The text data is processed to include special tokens that mark the beginning and end of a sequence, which helps the model understand sentence boundaries. This preprocessing converts the text into a sequence of integers, where each integer represents a word in a vocabulary created from the text data.

Model Architecture

The architecture of the model consists of two main parts: an image feature processing component and a text generation component. The image features, a vector of 2048 elements per image, are first processed through a dense layer to convert them into a smaller, more manageable size that matches the expected input size of the text generation model. This component uses a repeat vector to ensure the image features are appropriately structured to combine with the sequential text data.

The text generation model includes embedding layers, LSTM layers, and dense layers. The embedding layer converts integer-encoded words into dense vectors, capturing semantic meanings. LSTM layers are utilized for their ability to handle sequences, making them ideal for text generation where the context is vital. The model processes the combined image and text data to predict the next word in a sequence, learning the relationships between the visual content and the textual descriptions.

Training Process

The model is trained using a generator function that continuously feeds data into the model in batches. This is crucial for handling large datasets that might not fit into memory all at once. The generator fetches batches of image features and corresponding sequences of text, which are used to train the model. The output of the model at each step is a predicted word, and the training process involves adjusting the model weights to minimize the difference between the predicted words and the actual words in the text sequences.

Optimization and Regularization

To optimize the training, the model uses the RMSprop optimizer, a variant of stochastic gradient descent that maintains a moving average of the squares of gradients and divides the gradient by the root of this average. This method is known for its effectiveness in deep learning models, especially RNNs like LSTMs.

ModelCheckpoint and EarlyStopping are two key callbacks used during training. ModelCheckpoint saves the best model based on a monitored metric (accuracy in this case), ensuring that the best performing model is retained even if the model's performance degrades in subsequent epochs. EarlyStopping is used to prevent overtraining by stopping the training process if the model's performance does not improve for a defined number of epochs.

9. Reason on Why stream lit was used

Streamlit was chosen for its user-friendly interface, its ability to rapidly prototype and iterate, and its strong integration with the Python ecosystem. These features make it an ideal choice for demonstrating the capabilities of a machine learning model in a web environment, facilitating easy testing, interaction, and presentation of the model's functionality.