CNN + RNN Model with Attention for Image Captioning

# Overview

This project aims to generate captions for images using a hybrid deep learning model that combines Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), enhanced with Attention Mechanism. The model utilizes a pre-trained DenseNet201 for feature extraction from images and a LSTM network to generate captions with attention applied on the LSTM's output.

# Project Structure

- Image Feature Extraction: DenseNet201 (pre-trained on ImageNet) is used for extracting image features.  
- Text Processing: Captions are preprocessed and tokenized.  
- Model Architecture: The architecture consists of an encoder-decoder structure, where the encoder processes image features, and the decoder generates captions, incorporating attention on the LSTM's output.  
- Attention Mechanism: A simple attention layer is applied to the LSTM output to highlight important parts of the caption generation sequence.  
- Data: The Flickr8k dataset is used, consisting of images and their corresponding captions.

# Setup and Installation

1. Mount Google Drive:  
```python  
from google.colab import drive  
drive.mount('/content/drive')  
```  
2. Install Kaggle and download the Flickr8k dataset:  
```python  
!pip install kaggle  
!mkdir -p ~/.kaggle  
!cp /content/drive/MyDrive/credit\_project/kaggle.json ~/.kaggle/  
!chmod 600 ~/.kaggle/kaggle.json  
!kaggle datasets download -d adityajn105/flickr8k  
```  
3. Extract the dataset:  
```python  
import zipfile  
with zipfile.ZipFile('/content/flickr8k.zip', 'r') as zip\_ref:  
 zip\_ref.extractall('/content')  
```

# Data Loading and Preprocessing

The captions are preprocessed and tokenized. A clean\_caption function is defined to lower case the captions, remove punctuation, and add startseq and endseq tokens to the captions.  
```python  
def clean\_caption(caption):  
 caption = caption.lower()  
 caption = re.sub(r'[^\w\s]', '', caption)  
 caption = caption.strip()  
 caption = 'startseq ' + caption + ' endseq'  
 return caption  
```

# Tokenizer and Vocabulary

The captions are tokenized using Keras Tokenizer, and the vocabulary size is calculated.  
```python  
tokenizer = Tokenizer()  
tokenizer.fit\_on\_texts(captions['caption'])  
```  
The tokenizer is saved for later use in inference.

# Feature Extraction (DenseNet201)

DenseNet201 is used to extract feature vectors from each image. The images are resized to (224, 224), and processed to match the input shape expected by DenseNet201.  
```python  
def extract\_features(image\_path):  
 if not hasattr(extract\_features, 'model'):  
 extract\_features.model = DenseNet201(include\_top=False, weights='imagenet', pooling='avg')  
 img = load\_img(image\_path, target\_size=(224, 224))  
 img = img\_to\_array(img)  
 img = np.expand\_dims(img, axis=0)  
 img = tf.keras.applications.densenet.preprocess\_input(img)  
 features = extract\_features.model.predict(img, verbose=0)  
 return features.flatten()  
```

# Model Architecture

The model consists of an encoder (DenseNet201) and a decoder (LSTM with attention). The architecture is as follows:  
- Encoder: DenseNet201 processes the image and outputs an embedding.  
- Decoder: LSTM network processes tokenized captions, with attention applied on the LSTM output.  
```python  
def create\_model(vocab\_size, max\_length, embedding\_dim=128, units=256):  
 image\_input = layers.Input(shape=(1920,))  
 image\_features = layers.Dense(embedding\_dim, activation='relu')(image\_input)  
  
 text\_input = layers.Input(shape=(max\_length,))  
 embedding = layers.Embedding(vocab\_size, embedding\_dim)(text\_input)  
 lstm\_output, state\_h, state\_c = layers.LSTM(units, return\_sequences=True, return\_state=True)(embedding)  
  
 attention\_dense = layers.Dense(1, activation='tanh')(lstm\_output)  
 attention\_weights = layers.Softmax(axis=1)(attention\_dense)  
 context\_vector = layers.Multiply()([lstm\_output, attention\_weights])  
 context\_vector = layers.Lambda(lambda x: tf.reduce\_sum(x, axis=1))(context\_vector)  
  
 decoder\_combined = layers.Concatenate()([context\_vector, image\_features])  
 output = layers.Dense(units, activation='relu')(decoder\_combined)  
 output = layers.Dropout(0.5)(output)  
 output = layers.Dense(vocab\_size, activation='softmax')(output)  
  
 model = Model(inputs=[image\_input, text\_input], outputs=output)  
 model.compile(loss='categorical\_crossentropy', optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001), metrics=['accuracy'])  
 return model  
```

# Training the Model

The data is split into training and validation sets. The model is trained using categorical cross-entropy loss and Adam optimizer.  
Callbacks such as early stopping and model checkpoint are used for efficient training.  
```python  
history = model.fit(  
 train\_generator,  
 validation\_data=val\_generator,  
 epochs=10,  
 callbacks=[  
 tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True),  
 tf.keras.callbacks.ModelCheckpoint('best\_model\_attention1.h5', monitor='val\_loss', save\_best\_only=True)  
 ]  
)

# Saving the Model

The trained model is saved for later use in inference.  
```python  
model.save('image\_caption\_model\_attention1.h5')  
with open('tokenizer.pickle', 'wb') as handle:  
 pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST\_PROTOCOL)  
```

# Running Inference

Once the model is trained, it can be used to generate captions for unseen images. The process involves:  
1. Extracting features from the image using DenseNet201.  
2. Using the tokenizer to predict the caption word by word.  
3. Applying the attention mechanism during decoding.

# Conclusion

This project demonstrates how to use a CNN + RNN with Attention model to generate captions for images. The attention mechanism helps improve the focus on relevant parts of the sequence, enhancing the quality of the generated captions.