Final Assignment Outline FCC module (SE5110)

Table of Contents

Phase 1: Image Captioning

Introduction

In this phase of the assignment, my task is to create an image captioning model from scratch using the Flickr8k dataset. The main objective of this phase is to understand the fundamental concepts involved in building image captioning models and to improve skills in implementing deep learning architectures. By the end of this phase, the model should be capable of taking an image as input and generating a text caption that describes the contents of the image. To achieve this, I have followed a step-by-step process of designing and training my model, and optimizing it to achieve the best possible results.

Data Preparation

First download the zip file of image data set using the wget command and then unzip that file in to a directory called “ML”. Then load zip file of supporting files to the Colab environment.

I have used various functions to preprocess and arrange textual data for image captioning tasks. These functions include loading a text file containing image captions, cleaning the text by removing punctuation and other irrelevant characters, and creating a vocabulary of unique words. Additionally, they combine all of the descriptions into a single file for efficient processing. The process involves dividing the text, converting it to lowercase, removing punctuation, and generating a set of vocabulary. Finally, the clean descriptions are saved in a single file, making it easier to analyze or train models.

After that, I set the paths for the directories of the Flickr8k dataset and load the captions of the images from a text file. Then, I cleand and preprocessed the captions and build a vocabulary from the resulting descriptions. Finally, I saved the processed descriptions into a file. This process prepares the textual data for further use and facilitates tasks such as image captioning or image-text matching.

Model Architecture

I have written a program named "extract_features(directory)" that uses a convolutional neural network (CNN) model. The program takes a directory as input and then iterates through the images in that directory. It preprocesses each image and extracts features using the CNN model. The features are then stored in a dictionary in a flattened form. The CNN model is composed of convolutional layers, followed by max-pooling layers. It is ultimately flattened and connected to dense layers, which ends with an output layer that generates 2048 features. This process converts images into numerical feature representations. The extracted features are returned as a dictionary, where the image filenames are the keys, and their corresponding feature vectors are the values.

Then I have developed a set of functions that work together to prepare data for a machine learning model that generates image captions. The “load_photos” function reads image filenames from a specified file, while “load_clean_descriptions” processes a text file containing image descriptions, filtering and formatting them for model training. To align precomputed image features with the corresponding images, “load_features” is used. Finally, a code snippet is used to assemble training data by loading image filenames, cleaning descriptions, and gathering associated features. This comprehensive approach ensures that the model is fed with organized, relevant data, which is essential for training it to generate accurate image captions.

Then I have moved onto to create the vocabulary for the process. First, I created a function called "dict_to_list" which is responsible for converting a dictionary of descriptions into a list. Then, I created another function called "create_tokenizer" that utilizes Keras Tokenizer to tokenize the list of descriptions. The word-to-index mapping generated by the tokenizer is stored in a "tokenizer.p" file using the pickle module. Finally, I calculated the size of the vocabulary by considering the total count of unique words plus one for padding. This process is extremely important for natural language processing tasks, as it enables the conversion of textual data into a format that can be easily understood by machine learning models.

After that, I have defined a function called "max_length" which calculates the maximum length of text descriptions. The function takes a dictionary of descriptions as input and first converts it into a list. It then iterates through each description in the list to find the length of the longest one by counting the number of words using the "split()" method. Finally, the function returns this maximum length value. The result of calling this function with a "descriptions" parameter, which is assumed to be a dictionary of descriptions, is stored in the "max_length" variable.

For the purpose of creating a captioning model, I utilized TensorFlow's Keras API. The model design comprises two main components:

1. Image Feature Extraction: Initially, I used a Convolutional Neural Network (CNN) model to extract features from images. The process of feature extraction

reduces the images to a 256-node representation. To ensure proper regularization, dropout is applied.

2. Text Sequence Processing: Simultaneously with the image feature extraction, I processed text sequences using an embedding layer. The embedding layer maps each word to a high-dimensional vector representation. This process is followed by dropout for regularization. The processed text sequences are then fed into a Long Short-Term Memory (LSTM) layer with 256 units for sequence processing.

The two pathways are then merged through concatenation, combining the extracted image features and processed text sequences. The merged features undergo further processing through a dense layer. The dense layer consists of neurons that are fully connected to all the neurons in the previous layer. This process enables the model to predict the next word in a caption.

The model is then compiled using categorical cross-entropy loss and the Adam optimizer. Categorical cross-entropy loss is used to measure the dissimilarity between the predicted and actual caption. The Adam optimizer is used to optimize the model's weights to minimize the loss function.

Additionally, a graphical summary of the model is generated and saved as an image file to ensure proper visualization of the model architecture. Finally, the model object is returned for training and evaluation.

Model Training

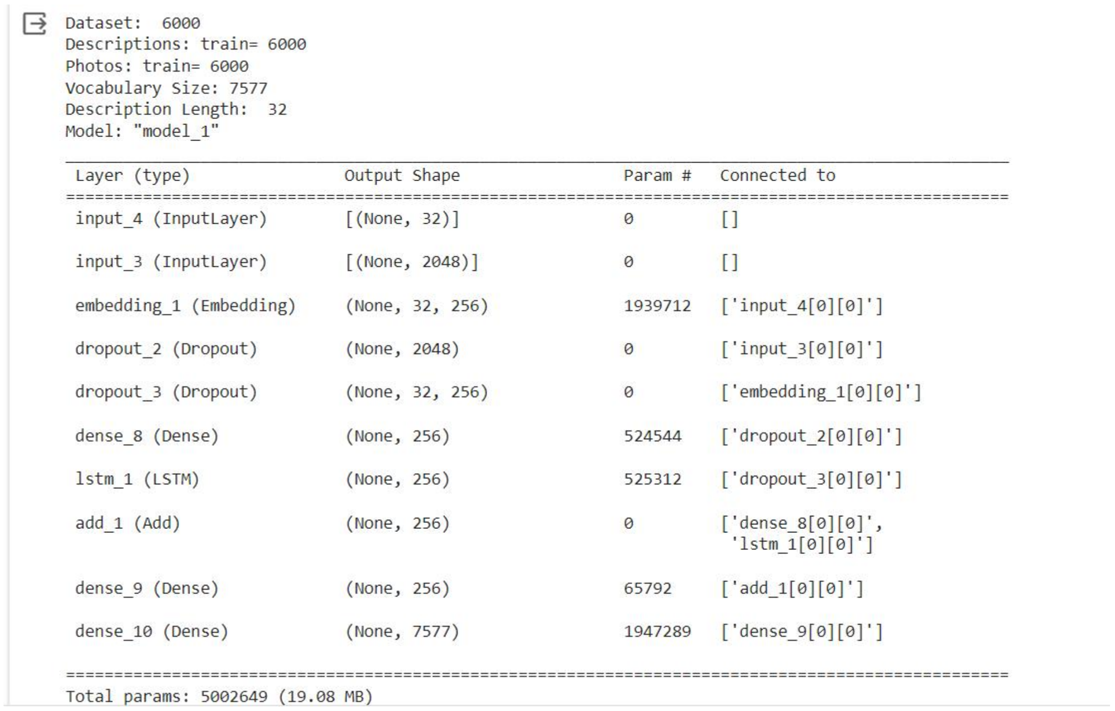
I have created a model for training that can generate images based on textual descriptions. Before we start the training process, the model displays some essential information that would help you understand the training process better. This information includes the size of the training dataset, the number of descriptions, the count of photos, the vocabulary size, and the maximum description length.

After displaying this information, the model sets up a neural network model using the provided vocabulary size and maximum description length. The number of training epochs is set to 10, which means the model will iterate over the dataset ten times during the training process. The number of steps per epoch is calculated based on the length of the training descriptions. This ensures that the model trains efficiently, making the most of the dataset that we have.

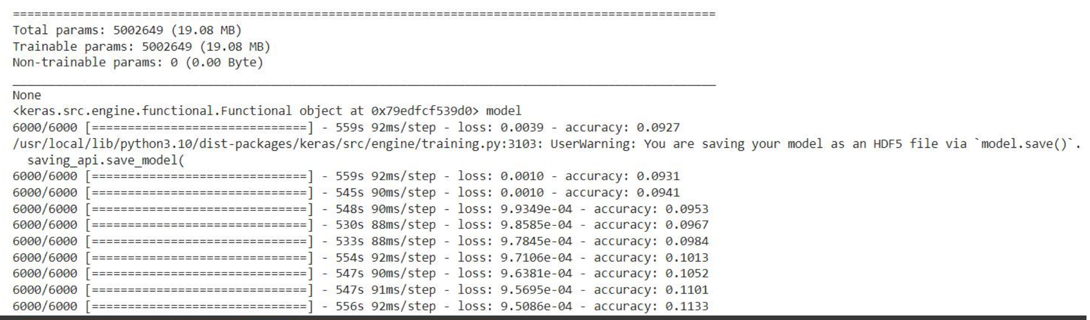
Next, the model checks whether a directory to store the trained models exists or not. If it doesn't exist, the model creates one to store the trained models. This makes it easier for you to access the trained models after the training process is complete.

During the training process, the model loops over the specified number of epochs, generating data batches using a data generator function. The model compiles with the RMSprop optimizer and binary cross-entropy loss, which is a standard loss function used in machine learning. Finally, the model trains for one epoch using the generated data batches and saves the trained model after each epoch with a unique identifier. This ensures that the trained models are saved and can be accessed later for further use.

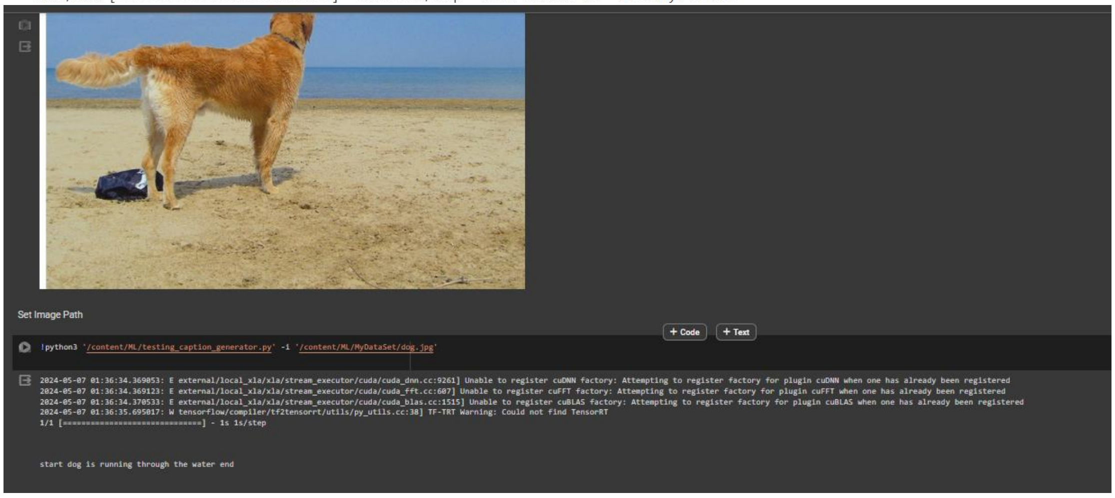
Results



```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model_1"
______________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==============================================================================================
 input_4 (InputLayer)        [(None, 32)]                 0         []

 input_3 (InputLayer)        [(None, 2048)]               0         []

 embedding_1 (Embedding)     (None, 32, 256)              1939712   ['input_4[0][0]']

 dropout_2 (Dropout)         (None, 2048)                 0         ['input_3[0][0]']

 dropout_3 (Dropout)         (None, 32, 256)              0         ['embedding_1[0][0]']

 dense_8 (Dense)             (None, 256)                  524544    ['dropout_2[0][0]']

 lstm_1 (LSTM)               (None, 256)                  525312    ['dropout_3[0][0]']

 add_1 (Add)                 (None, 256)                  0         ['dense_8[0][0]',
                                                                     'lstm_1[0][0]']

 dense_9 (Dense)             (None, 256)                  65792     ['add_1[0][0]']

 dense_10 (Dense)            (None, 7577)                 1947289   ['dense_9[0][0]']

==============================================================================================
Total params: 5002649 (19.08 MB)
```



```
==============================================================================================
Total params: 5002649 (19.08 MB)
Trainable params: 5002649 (19.08 MB)
Non-trainable params: 0 (0.00 Byte)
______________________________________________________________________________________________
None
<keras.src.engine.functional.Functional object at 0x79edfcf539d0> model
6000/6000 [==============================] - 559s 92ms/step - loss: 0.0039 - accuracy: 0.0927
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`.
  saving_api.save_model(
6000/6000 [==============================] - 559s 92ms/step - loss: 0.0010 - accuracy: 0.0931
6000/6000 [==============================] - 545s 90ms/step - loss: 0.0010 - accuracy: 0.0941
6000/6000 [==============================] - 548s 90ms/step - loss: 9.9349e-04 - accuracy: 0.0953
6000/6000 [==============================] - 530s 88ms/step - loss: 9.8585e-04 - accuracy: 0.0967
6000/6000 [==============================] - 533s 88ms/step - loss: 9.7845e-04 - accuracy: 0.0984
6000/6000 [==============================] - 554s 92ms/step - loss: 9.7106e-04 - accuracy: 0.1013
6000/6000 [==============================] - 547s 90ms/step - loss: 9.6381e-04 - accuracy: 0.1052
6000/6000 [==============================] - 547s 91ms/step - loss: 9.5695e-04 - accuracy: 0.1101
6000/6000 [==============================] - 556s 92ms/step - loss: 9.5086e-04 - accuracy: 0.1133
```



Set Image Path

```
!python3 '/content/ML/testing_caption_generator.py' -i '/content/ML/MyDataSet/dog.jpg'
```

```
2024-05-07 01:36:34.369053: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
2024-05-07 01:36:34.369123: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2024-05-07 01:36:34.370533: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered
2024-05-07 01:36:35.695017: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
1/1 [==============================] - 1s 1s/step


start dog is running through the water end
```

Final test accuracy : 0.1133
Final loss: 9.5086e-04

Model Interpretation

Conclusion

The current model is yielding outputs with low accuracy, indicating a need for improvement. To enhance the performance of the CNN captioning model, we can explore the option of implementing a suitable activation function. By doing so, we can potentially increase the accuracy of the model and achieve more accurate outputs.

Phase 2: Image Generation

Introduction

The goal in phase 2 is to create a model that can turn written words into images. To make this happen, I am using natural language processing and computer vision techniques to connect language and visuals. The primary objective is to create a framework that can generate detailed and precise visual outputs from descriptive texts effortlessly. This framework could be utilized for various purposes, including content creation and making technology more accessible.

Data Preparation

Please refer the data preparation in part 1.

Model Architecture

In Phase 2 of the process, text-to-image generation is performed using a pre-trained deep learning model that was created in Phase 1. Initially, this phase imports necessary libraries such as TensorFlow, Numpy, and PIL. The pre-trained generator model and tokenizer are then loaded from specified file paths.

The function 'generate_image_from_caption' takes a textual caption as input and converts it into a sequence of tokens using the loaded tokenizer. The sequence is then padded to a fixed length and fed to the generator model for image generation.

The generated image is post-processed to ensure pixel values are within the valid range and is converted to a PIL image format. Finally, an example caption is provided, and the generated image corresponding to this caption is displayed.

This process showcases how the model can translate textual descriptions into visually interpretable images, offering potential applications in areas such as creative content generation and computer vision tasks.

Model Training

Results

Model Interpretation

Conclusion

The Image Generation model has several potential areas for improvement and avenues for future exploration. These areas can help enhance the model's performance and provide better results for various applications.

One of the most important areas to focus on is Model Evaluation. The quality of the generated images must be assessed accurately. Metrics such as Inception Score or Frechet Inception Distance (FID) can be used to provide quantitative measures of image quality and diversity. These metrics can help evaluate the effectiveness of the model and identify areas that need improvement.

Another potential area for improvement is Model Training. Fine-tuning the generator model on a larger and more diverse dataset could enhance its ability to generate high-quality images across various input captions. This can lead to better results and more diverse outputs.

Hyperparameter Tuning is another area that should be explored. Optimizing hyperparameters such as the sequence length, batch size, and learning rate can potentially improve the model's performance and convergence speed. This can help improve the model's effectiveness and speed up the training process.

Handling Out-of-Vocabulary (OOV) Words is also important. The tokenizer may encounter words it hasn't seen during training, which can affect the model's performance. Implementing strategies to handle OOV words, such as replacing them with similar known words or using character-level tokenization, could improve the model's robustness and accuracy.

Image Post-processing is another potential area for improvement. Experimenting with different post-processing techniques such as adjusting brightness, contrast, or applying style transfer could further enhance the visual quality of generated images. This can lead to more visually appealing and realistic image outputs.

User Interface is another essential area to consider. Building a user-friendly interface to interact with the model, allowing users to input captions and visualize the corresponding generated images easily, would enhance usability. This can help make the model more accessible and user-friendly.

Memory Efficiency is another critical area to focus on. For larger models or datasets, optimizing memory usage during inference can be crucial, especially when deploying the model in resource-constrained environments. This can help improve the model's performance and speed up the inference process.

Finally, exploring Conditional Generation is another potential area for improvement. Investigating methods for conditional image generation, where additional factors such as image style or scene composition are taken into account, can lead to more contextually relevant and diverse image outputs. This can help the model generate more diverse and realistic images for various applications.

In conclusion, addressing these areas can contribute to the refinement and advancement of text-to-image generation systems, improving their utility and effectiveness in various applications.

Video Link

https://youtu.be/mOQrkmN_uW0

Github Link

https://github.com/Shanika-Wickramathunga/FCC-Phase1.git