

IMAGE CAPTION GENERATION

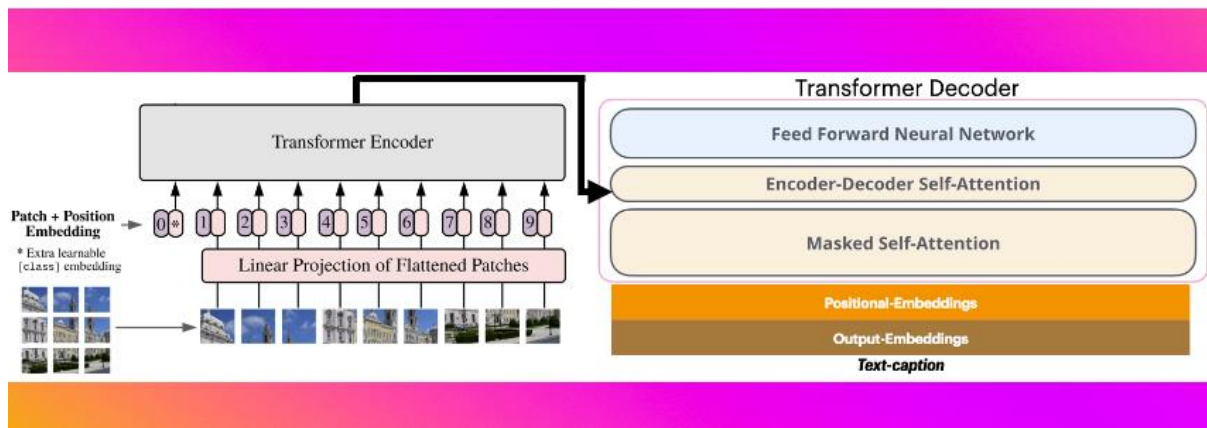
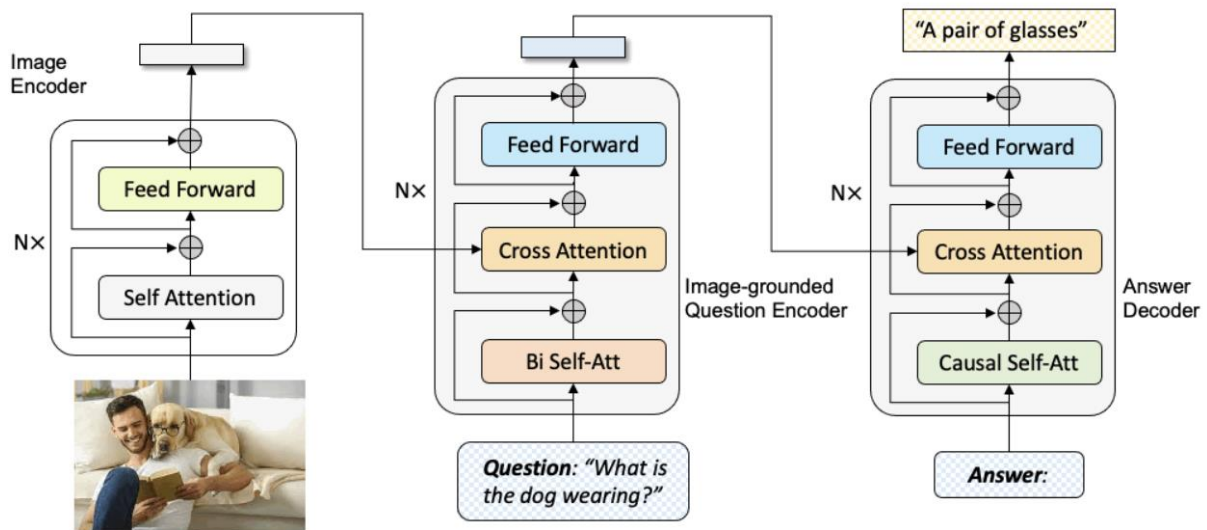
INTRODUCTION:

Image caption generation is a compelling intersection of computer vision and natural language processing, challenging the boundaries of artificial intelligence. The task involves developing algorithms that can autonomously generate coherent textual descriptions for images. Unlike humans who effortlessly interpret visual content and articulate meaningful descriptions, computers face formidable challenges in understanding and translating visual information into human-like language.

Traditionally, classical methods used handcrafted features and rule-based systems for image captioning. However, the advent of deep learning, particularly Convolutional Neural Networks (CNNs) for image analysis and Recurrent Neural Networks (RNNs), such as Long Short-Term Memory (LSTM) networks for sequential data, has reshaped this landscape. Deep neural networks have proven instrumental in achieving state-of-the-art results in image captioning.

This Python-based project explores the synergy of CNNs and LSTMs to automatically generate descriptive captions for images. CNNs excel at extracting hierarchical features from images, while LSTMs are proficient in understanding sequential data. By combining these architectures, the project aims to replicate the human ability to perceive visual content and articulate it in natural language. The significance of extensive datasets and robust computing power is emphasized, highlighting their role in training sophisticated models capable of generating captions for a diverse range of images. The objective is to delve into the complexities of image caption generation, showcasing the transformative potential of deep learning in bridging the gap between visual understanding and linguistic expression.

TECHNICAL ARCHITECTURE:



PRE-REQUISITES :

To embark on the journey of Image Caption Generation using PyTorch in Google Colab, you need to set up your environment with the necessary tools and libraries. Here are the pre-requisites:

Google Colab:

Google Colab is a free, cloud-based platform that provides access to a GPU, which is essential for training deep learning models efficiently. You can access Colab through your Google account.

Google Drive:

Google Drive will be used for storing datasets, model checkpoints, and other project-related files. Connect your Google Drive to Colab for seamless data access.

PyTorch:

PyTorch is an open-source machine learning library developed by Facebook. It provides a dynamic computational graph and is widely used for deep learning tasks. Install PyTorch using the following command:

```
!pip install torch torchvision
```

Transformers Library:

The Transformers library by Hugging Face provides pre-trained models for natural language processing tasks, including image captioning. Install it with:

```
!pip install transformers
```

Other Python Libraries:

Install additional libraries required for data manipulation, plotting, and miscellaneous tasks:

```
import torch  
from PIL import Image
```

Colab Notebook:

Create a new Colab notebook where you will write and execute your code. You can create a new notebook by going to Google Drive, right-clicking, and selecting "Google Colaboratory."

Project Objectives:

By the end of this project, you will:

Understand the fundamentals of image caption generation using deep learning.

Gain practical experience in setting up a project in Google Colab for image captioning.

Learn how to leverage PyTorch and pre-trained models for image captioning tasks.

Build an end-to-end image caption generation system.

Data Collection:

Gather or download a dataset of images and their corresponding captions.

Ensure that the dataset is structured appropriately.

Data Pre-processing:

Pre-process the images and captions to make them suitable for training. This involves resizing images, tokenizing captions, and handling any other necessary pre-processing steps.

Model Building:

Utilize pre-trained models from the Transformers library or build a custom model using PyTorch. This step involves defining the model architecture for image captioning.

Training the Model:

Train the model on your dataset, adjusting hyperparameters as needed.

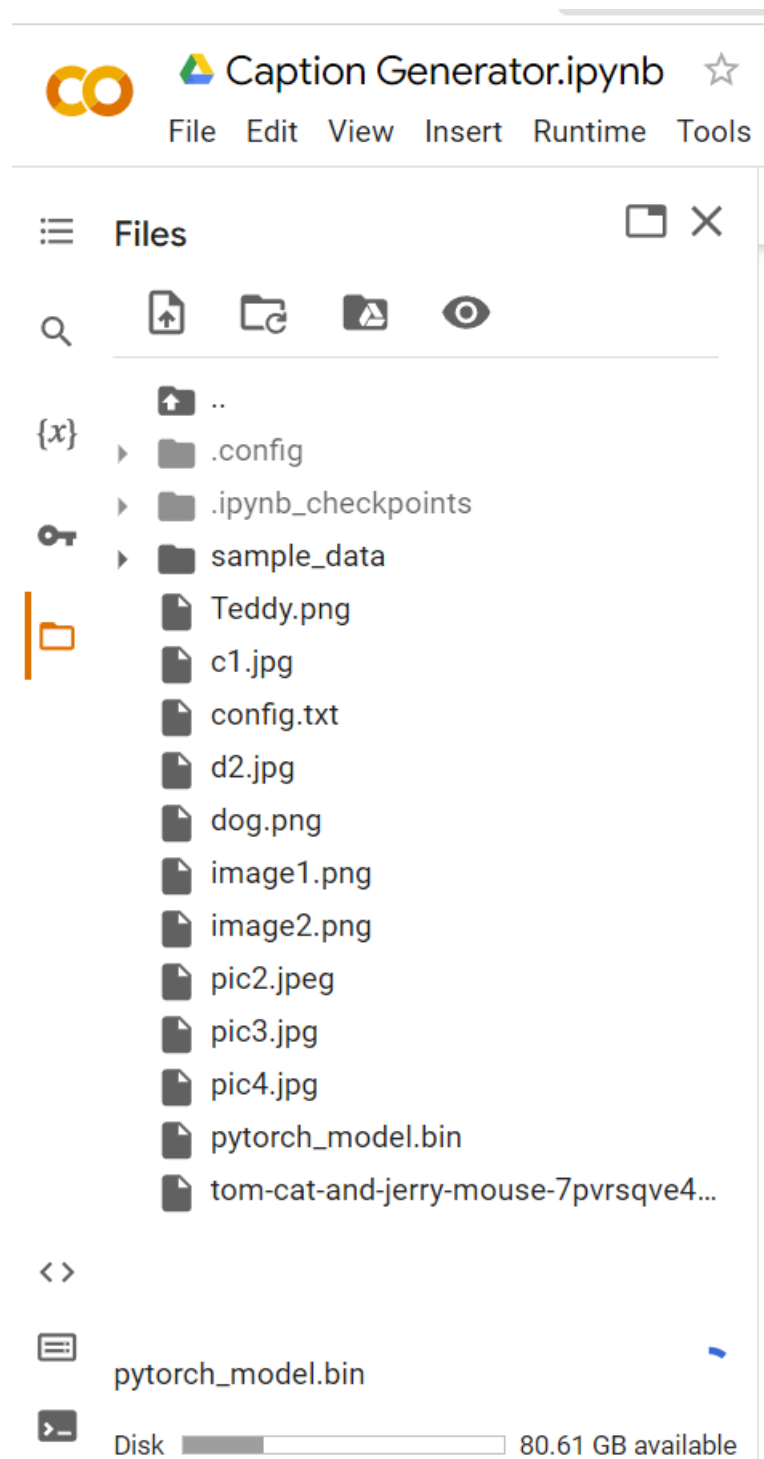
Monitor the training process for convergence and model performance.

Generating Captions:

Once the model is trained, use it to generate captions for new images. Evaluate the model's performance on a test set.

PROJECT STRUCTURE:

Create a Project folder which contains files as shown below



Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
!pip install transformers

Collecting transformers
  Downloading transformers-4.35.0-py3-none-any.whl (7.9 MB)
    7.9/7.9 MB 40.9 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
    302.0/302.0 kB 18.6 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
    3.8/3.8 MB 62.3 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
    1.3/1.3 MB 59.1 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.12.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (4.5.0)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
    295.0/295.0 kB 23.4 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.1)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers, transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1 transformers-4.35.0

from transformers import VisionEncoderDecoderModel, ViTFeatureExtractor, AutoTokenizer
import torch
from PIL import Image
```

Activity 2: Exploratory Data Analysis

```
model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

Downloading (...)ve/main/config.json: 100% 4.61k/4.61k [00:00<00:00, 92.5kB/s]
Downloading pytorch_model.bin: 100% 982M/982M [00:21<00:00, 40.7MB/s]
Downloading (...)rocessor_config.json: 100% 228/228 [00:00<00:00, 9.71kB/s]
/usr/local/lib/python3.10/dist-packages/transformers/models/vit/feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtractor is deprecated
  warnings.warn(
Downloading (...)okenizer_config.json: 100% 241/241 [00:00<00:00, 9.41kB/s]
Downloading (...)olve/main/vocab.json: 100% 798k/798k [00:00<00:00, 6.74MB/s]
Downloading (...)olve/main/merges.txt: 100% 456k/456k [00:00<00:00, 15.5MB/s]
Downloading (...)main/tokenizer.json: 100% 1.36M/1.36M [00:00<00:00, 10.6MB/s]
Downloading (...)cial_tokens_map.json: 100% 120/120 [00:00<00:00, 4.37kB/s]
```

Activity 4: Initializing the model

Keras has 2 ways to define a neural network:

- Sequential
- Function API

The Sequential class is used to define linear initializations of network layers which then,

collectively, constitute a model. In our example below, we will use the Sequential

constructor to create a model, which will then have layers added to it using the add() method.

Activity 5: Configure the Learning Process

- The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
- Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Activity 6: Using a Pre-trained Model

Instead of training the model from scratch, you can leverage a pre-trained model for image captioning. Here's how you can modify the training activity to use a pre-trained model:

Load Pre-trained Model:

Choose a pre-trained model suitable for image captioning. Common choices include models from the Transformers library or pre-trained image captioning models. Load the pre-trained model:

```
from keras.models import load_model
```

```
# Load a pre-trained model
```

```
pre_trained_model = load_model('path/to/pretrained/model.h5')
```

Compile the Model:

Since the model is pre-trained, you don't need to recompile it. The compilation is already done during the model's initial training. You can skip this step or customize it based on your requirements.


```

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

max_length = 16
num_beams = 4
gen_kwargs = {"max_length": max_length, "num_beams": num_beams}
def predict_step(image_paths):
    images = []
    for image_path in image_paths:
        i_image = Image.open(image_path)
        if i_image.mode != "RGB":
            i_image = i_image.convert(mode="RGB")

        images.append(i_image)

    pixel_values = feature_extractor(images=images, return_tensors="pt").pixel_values
    pixel_values = pixel_values.to(device)

    output_ids = model.generate(pixel_values, **gen_kwargs)

    preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)
    preds = [pred.strip() for pred in preds]
    return preds

predict_step(['dog.png'])

```

Activity 7: Save the Model

The model is saved with .h5 extension as follows An H5 file is a data file saved in the

Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

Activity 8: Test The model

Evaluation is a process during the development of the model to check whether the

model is the best fit for the given problem and corresponding data. Load the saved

model.

OUTPUT:


Taking an image as input and checking the results.

+ Code + Text

9s


 We strongly recommend passing in an `attention_mask` since your input_ids may be padded. You may ignore this warning if your `pad_token_id` (50256) is identical to the `bos_tok` ['a dog sitting on top of a lush green field']

12s

 `predict_step(['dog.png'])`


['a dog sitting on top of a lush green field']

dog.png ×




+ Code + Text

9s


 We strongly recommend passing in an `attention_mask` since your input_ids may be padded. You may ignore this warning if your `pad_token_id` (50256) is identical to the `bos_tok` ['a man standing in the middle of a field']

23s

 `predict_step(['pic2.jpeg'])`

['a man standing in the middle of a field']

pic2.jpeg ×



RAM

Disk

Colab paid products - Cancel contracts here

```
predict_step(['pic4.jpg'])
```

['a baseball player running to catch a ball']

[Colab paid products](#) - [Cancel contracts here](#)

