Module Code & Module Title

CU6051NI Artificial Intelligence

Assessment Weightage & Type

25% Individual

Semester

2024/25 Autumn

Project Name: Nepali Image Captioning

Student Name: Miraj Deep Bhandari

London Met ID: 22067814

College ID: np01cp4a220197

Group :L3C8

Assignment Due Date: 22 January 2025

Assignment Submission Date: 21 January 2025

22067814 Miraj Deep Bhandari.docx

Islington College,Nepal

Document Details

Submission ID

trn:oid:::3618:79759032

Submission Date

Jan 21, 2025, 8:26 PM GMT+5:45

Download Date

Jan 21, 2025, 8:28 PM GMT+5:45

File Name

22067814 Miraj Deep Bhandari.docx

File Size

30.8 KB

37 Pages

4,520 Words

26,630 Characters

9% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

Match Groups

40 Not Cited or Quoted 9%
Matches with neither in-text citation nor quotation marks

1 Missing Quotations 0%
Matches that are still very similar to source material

0 Missing Citation 0%
Matches that have quotation marks, but no in-text citation

0 Cited and Quoted 0%
Matches with in-text citation present, but no quotation marks

Top Sources

6%	Internet sources
1%	Publications
5%	Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.

Match Groups

- **40** Not Cited or Quoted 9%
  Matches with neither in-text citation nor quotation marks
- **1** Missing Quotations 0%
  Matches that are still very similar to source material
- **0** Missing Citation 0%
  Matches that have quotation marks, but no in-text citation
- **0** Cited and Quoted 0%
  Matches with in-text citation present, but no quotation marks

Top Sources

6%	Internet sources
1%	Publications
5%	Submitted works (Student Papers)

Top Sources

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Internet	conference.ioe.edu.np	3%
2	Internet	www.scribd.com	<1%
3	Submitted works	University of Greenwich on 2023-01-17	<1%
4	Submitted works	Great Lakes Institute of Management on 2024-05-04	<1%
5	Internet	dokumen.pub	<1%
6	Submitted works	Bournemouth University on 2023-01-06	<1%
7	Submitted works	Liverpool John Moores University on 2021-01-31	<1%
8	Submitted works	City University on 2023-10-01	<1%
9	Submitted works	Universidade de Aveiro on 2024-07-31	<1%
10	Submitted works	Birla Institute of Technology and Science Pilani on 2023-05-22	<1%

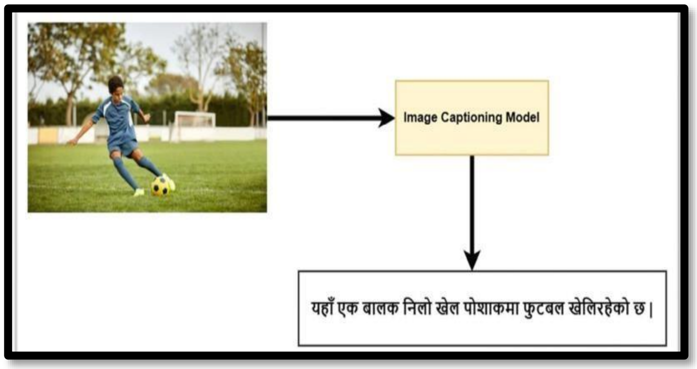
Table of Contents:

Table of Figures:

Table of Tables:

1. Introduction:

Image captioning is an advanced research and application domain that involves describing an image with textual information, generally using the realms of computer vision and NLP. It combines the powers of deep learning, AI, and neural networks in order to bridge the gap between visual perception and comprehension of human language. This has opened up ways for a range of applications involving access to visual information to visually impaired persons, automatic content creation, and better human-computer interaction.



I am working on Nepali Image Captioning in Deep Learning for my project. Fundamentally, Nepali image captioning involves the semantic content of an image and transforms this knowledge into grammatically correct and contextually meaningful text in Nepali. The system has to extract features from the images, spot objects, relationships, and activities within the image, and generate captions that reflect this insight.

Two important technologies followed here are **computer vision and natural language processing.** Computer vision is a subfield of AI dealing with providing computers with the ability to interpret visual data from the environment such as images and videos. It includes object detection, image classification, segmentation, and presentation, among others, which are important for meaningful content to be extracted from the images for image captioning (Computer Vision for Visual Computing: Techniques and Applications, 1998). On its part, NLP deals with the development of the ability of machines to understand and generate human language. The techniques of NLP apply in giving images captions that can be understood by humans, which require syntax, background knowledge, and semantics understanding for coherent and meaningful descriptions to be generated (Rao and McMahan, 2019).

1.1 Problem Domain & Its Solutions:

The problem domain of image captioning encompasses several longstanding challenges that this technology seeks to address:

1. **Accessibility**:
   The first and foremost challenge is that millions of visually impaired people cannot understand visual content. Image captioning tools allow them to learn about images through descriptive captions, thus being inclusive and opening up previously inaccessible information. In this way, image captioning fills the gap and achieves equality in access to digital and physical materials.

2. **Information Overload:**
   The growth of visual data on social networks, news websites, and online marketplaces demands an effective method of management. Humans cannot categorize, tag, or describe the vast number of images, a process that can be time-consuming and unrealistic. Image captioning, therefore, gives automation. This is because the caption needs to be short yet accurate, which helps in visual data searching, retrieval, and usage.

3. **Social Media:**
   On platforms like Instagram and Twitter, the engagement is totally dominated by visual content. Many experiences are cut off from visually impaired users due to a lack of proper description. Automatic image descriptions can facilitate higher engagement since the content is all-inclusive, thus fostering inclusivity and interaction in social communities.

4. **E-commerce:**
   The only factor behind the survival of online retail is the way in which a product is represented visually. Descriptions automatically created about products from images taken by shops increase the visibility and findability of a certain product, and enhance customer experience. Accurate captioning will facilitate customer decision-making, providing detailed insight into product features therefore, fostering trust and conversion.

5. **Health:**
   Medical imaging-interpretation of X-rays, MRI, or CT scans-is a task solely meant for skilled professionals. Automatic image captioning can help with giving preliminary explanations for such scans to help the doctors in making a more efficient and effective diagnosis. This will also be very helpful for training medical students.

1.2 Aims & Objectives:

1.2.1 Aims:

- To create a Deep Learning based system that can Generate the Correct and Corresponding Caption for the Image in the Nepali language.
- To use a combination of the computer vision and the natural language processing to properly interpret the visual content and describe it in natural text.
- To develop domain-specific linguistic rules and visual content understanding to improve quality of Nepali image captioning.
- To help in the development of AI applications for low resource languages such as Nepali.

1.2.2 Objectives:

- Use modern Computer Vision algorithms like **Vison Transformers** for feature extraction from the images
- Use sequence generation language model such as a **transformer Decoder (GPT)** to generate Nepali captions from the visual features.
- For training and evaluation of the image captioning model, datasets in Nepali need to be preprocessed and annotated.
- Choose standard metrics (BLEU, METEOR, CIDEr) to evaluate the generated captions for the model.
- Overcome challenges related to grammar, semantics, and contextual appropriateness by Fine-tuning and optimizing the model for Nepali text generation.

2. Background:

2.1 Research on the problem scenario:

In the early years hand-crafted features and rule-based systems dominated the field of image captioning . It was more about retrieving certain visual features from the images and writing captions using predefined templates or simple probabilistic models.

1. **Rule-Based Systems (Pre 2000s):**

   Earlier approaches relied on handcrafted rules to characterize an image. An image, for example, could have been used to detect objects and attributes of an object (color, shape, or position) and then captions could be generated based on a combination of template and attributes.

   Example:

   Detected objects: “Car,” “Red,” “Road.”

   Caption produced: “A red car on the road.”

2. **Visual Features and Bag of Words (2000s):**

These approaches used manually engineered visual features like **SIFT (Scale-Invariant Feature Transform)** or **HOG (Histogram of Oriented Gradients)** to detect objects or detections in the image.

These were then compared to some fixed collection of labels/captions with simple statistical models based on K-Nearest Neighbors or Bayesian networks.

Limitations:

Must be approved upon design features requiring some domain expertise

Not generalizable at all and unable to do anything more than the base rate

3. **Early 2010s: Retrieval-Based Captioning (2000s – Early 2010s):**

Such systems relied on having a well labelled database of images and rather than generating captions from scratch, they would find the closest matching image in the database and use it caption.

One example involved matching images using global image descriptors (e.g., GIST) based on their overall similarity.

Limitation:

An inability to generate new captions and generalize with unseen images.

The above methods are the traditional ways of generating captions from images. Modern approaches take a different route in this process with:

1. **Architecture with Encoder and Decoder:** This consists of CNNs (like ResNet) for feature extraction of images and using RNNs (RNNs like LSTMs) to generate captions sequentially.

2. **Attention Mechanisms:** Works by dynamically choosing different relevant parts of the image for each word, instead of calculating one single image per caption. Thus enabling stricter accuracy and detail.

3. **Transformer:** Uses transformers (e.g. Vision Transformers, GPT) for image analysis and caption generation, greater scalability, more fluent, capacity for complex scenes.

For developing the image captioning model, I have gone through the following literature reviews:

Paper Title	Features Implemented	Algorithms Used	Link
Show and Tell: A Neural Image Caption Generator	• Uses CNNs to extract features from the image and RNNs to learn to generate a caption describing the image. It uses the encoder-decoder structure with a pre-trained Inception model encoder and a LSTM decoder.	• Convolutional Neural Networks (CNNs) with some famous models like Inception For Feature Extraction. • Long Short Term Memory (LSTM) for sequential language modeling.	https://arxiv.org/abs/1411.4555

Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention	• Adds an attention mechanism which enables the model to pay attention to specific areas of the image when generating each word. This enhances the quality and specificity of the captions generated.	• CNN (ResNet) to extract features. • Whereas a soft attention mechanism focusing on image regions but also introducing a distracting amount of noise. • LSTM for caption generation.	https://arxiv.org/abs/1502.03044
Image Captioning with Semantic Attention	• Semantic attention which fuses global semantic (e.g., latent structured attributes) and spatial visual features. This enhancement helps in the generation of captions which are semantically relevant.	• CNN for extracting spatial features. • Fuse Visual and Semantic features via Semantic Attention Mechanism. • Long short term memory (for generating words one by one).	https://arxiv.org/abs/1603.03925

Image Captioning with CNN and RNN	• It directly investigates the integration of CNN for extracting visual features and RNN for forming the actual sentences. It shows how CNN-based feature vectors can lead the RNN in caption generation.	• Extracting image features using a CNN (VGG16 / ResNet). • Sequentially using GRU (Gated Recurrent Unit) or LSTM to create the caption.	https://arxiv.org/abs/1604.03944
Image Captioning Using Vision Transformers (ViT)	• Suggests using Vision Transformers (ViT) instead of CNNs. Transformers can model the global dependencies of images better than CNNs by getting a larger context.	• Using Vision Transformer (ViT) as feature extractor. • Transformer-based Encoder-Decoder with Transformer for caption generation. • The Multi-Head Attention is used to learn dependencies not only between the image regions but also between the image regions and captions too.	https://arxiv.org/abs/2105.01928
Attention Is All You Need	• Self attention: it captures semantic relationships between all elements in a sequence. • Multi-Head Attention: It	• Scaled Dot-Product Attention. • Layer Normalization, Feedforward Networks.	https://arxiv.org/abs/1706.03762

	allows looking at different places in the input at the same time. • Positional Encoding: Helps in maintaining the order of the sequence. •Encoder-Decoder Architecture: Stacking of both an encoder and a decoder, Encoder-decoder Architecture.		

2.2 Related Works:

Most of the works related to image captioning have been carried out for linguistically rich languages like English. Recently, a few works also report progress on Hindi and Bengali, which are linguistically close to Nepali. The very first work on Nepali image captioning is that of Aashish Adhikari and Sunil Ghimire, who proposed encoder-decoder models with/without visual attention. They employed ResNet-50 and Inception V3 as the encoder and LSTM and GRU as the decoders by training them using MS COCO for generating Nepali language captions. On the other hand, Mishra et al. created a Hindi captioning dataset by employing ResNet101 for image encoding and transformer for decoding in order to model semantic relations along with long-range dependencies.

Based on these advances, Palash et al. and Shah et al. employed CNN-Transformer encoder-decoder models for Bengali image captioning. Authors have exploited architectures such as ResNet101-Transformer and InceptionV3-Transformer, yielding promising results inspired by earlier works on English. Later, Subedi and Bal proposed a CNN-Transformer model for Nepali captions using ResNet101 and EfficientNetB0. Their BLEU score analysis has shown that EfficientNetB0 outperformed ResNet101 by a very small margin. Liu et al. extended this further by proposing an architecture of only a transformer-based, thereby eliminating the need to incorporate both the CNN and RNN components individually. Their end-to-end transformer relied on self-attention and cross-modal attention mechanisms and provided competitive accuracy while generating contextually relevant image captions.

Although CNN-Transformer models have presented quite remarkable potential so far, the architecture of a full transformer is yet to be explored for Nepali image captioning and its low-resource linguistically similar versions. The research work presented here applies an end-to-end transformer model motivated by earlier works with an aim to go beyond the state-of-the-art results in Nepali image captioning.

2.3 System architecture & block diagram :

Our model is made up of **a Vision Transformer** and **GPT**. The Vision Transformer acts as the encoder, and GPT acts as the decoder.

Lets see Individual Encoder and Decoder Archicture diagram:

Encoder Archicture Diagram:

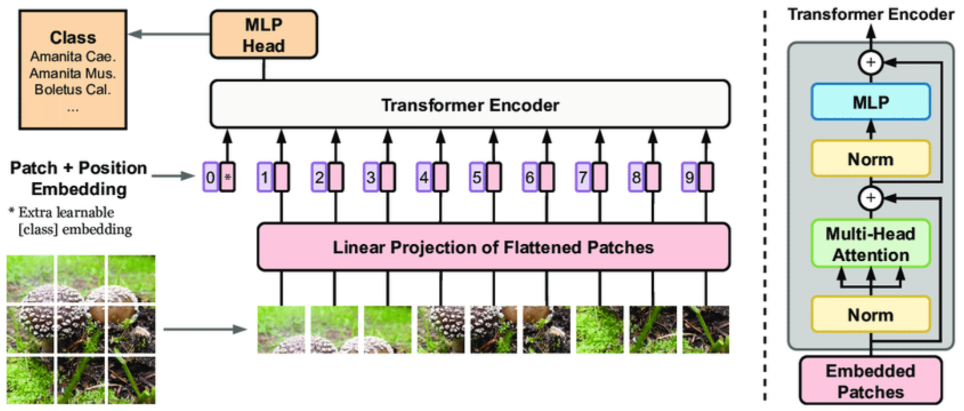


Figure 1: Encoder

Decoder Archicture Diagram:

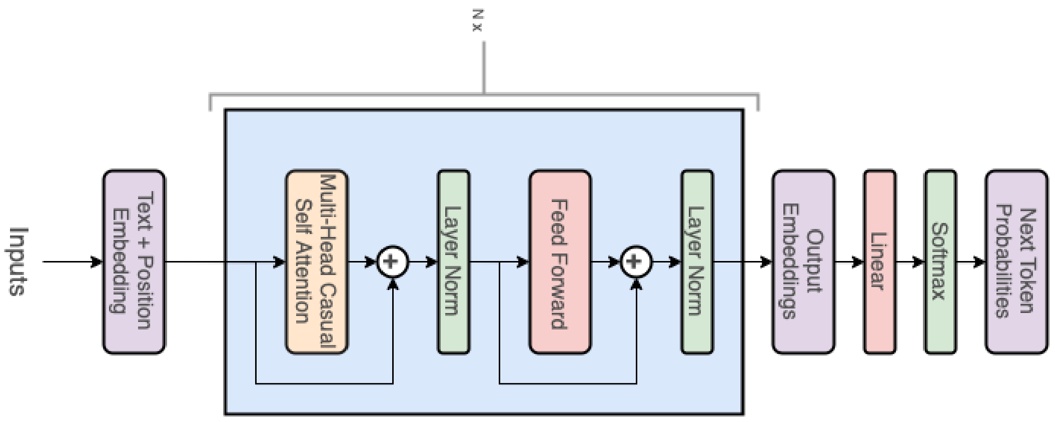


Figure 2: Decoder

Entire Archicture Diagram:

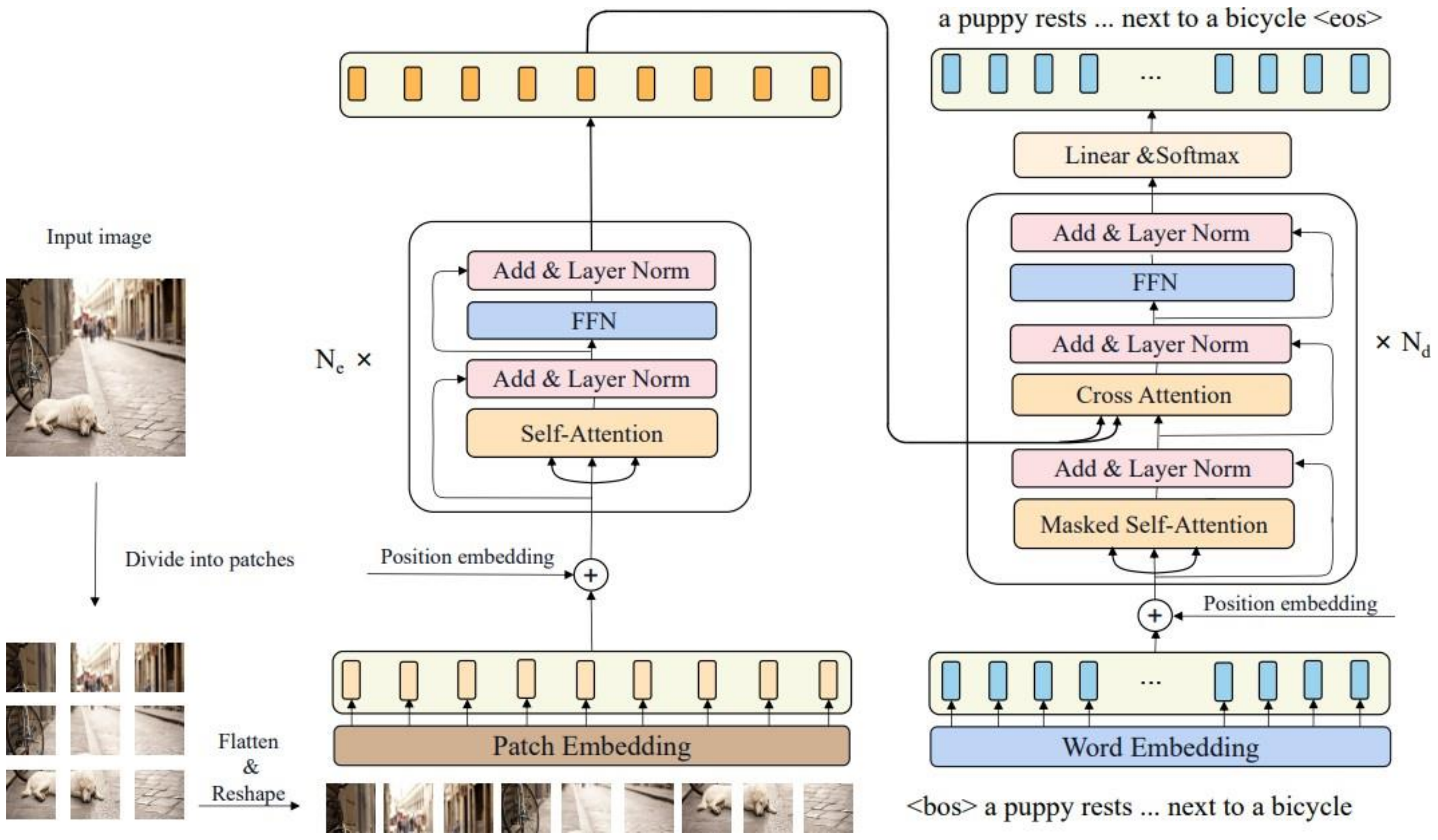
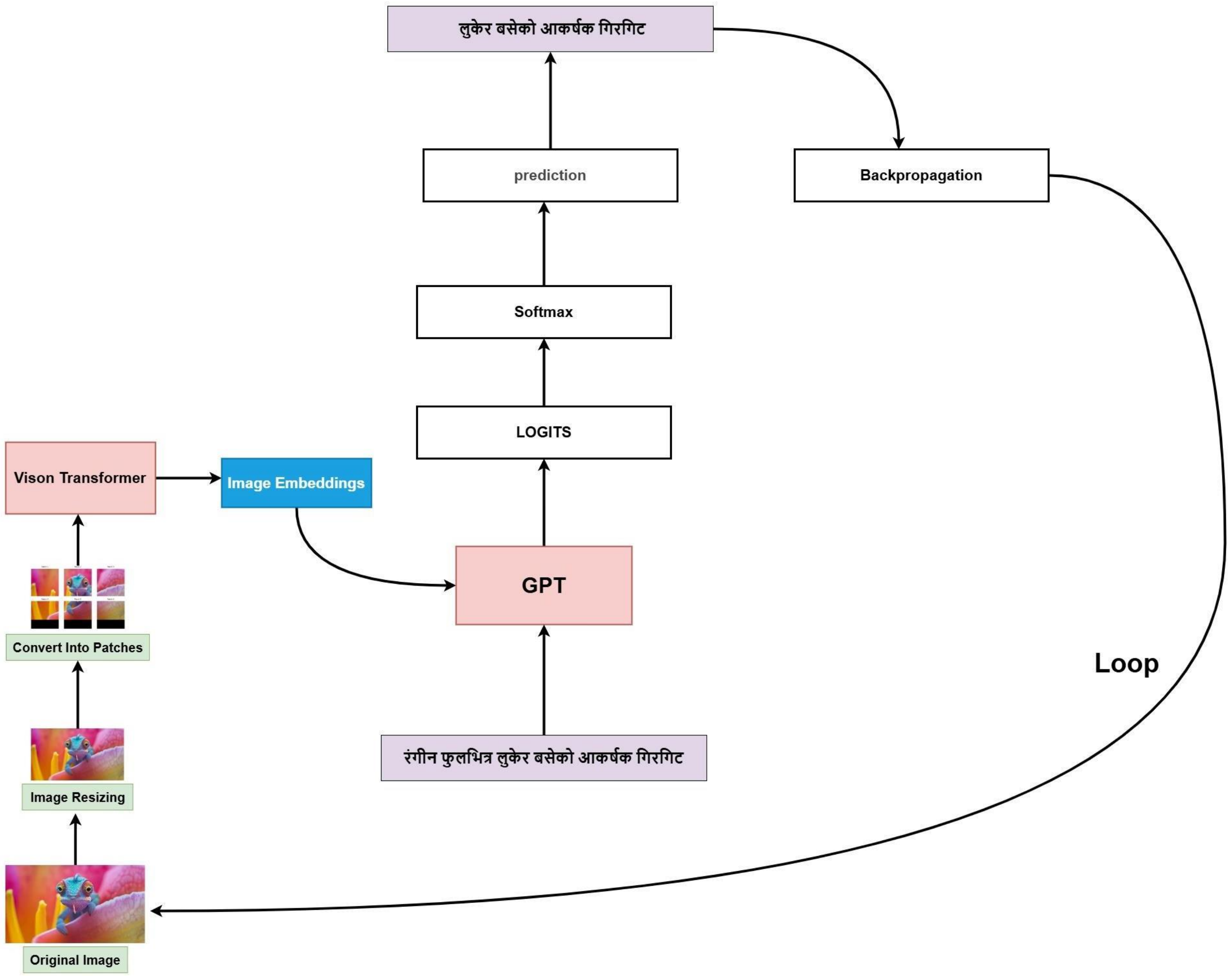


Figure 3: Entire Archicture Diagram

Flow Chart Diagram:



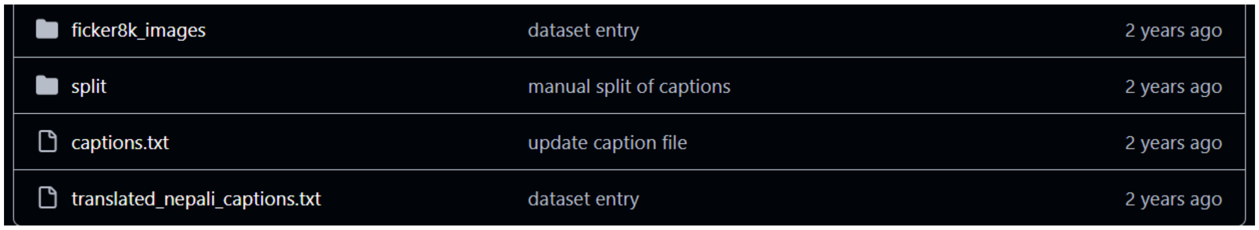
2.4 Designed DataSet:

For this work, the Nepali Flickr8k dataset was utilized. It consists of 8,000 images, each with fone different captions in the Nepali language. The dataset is split into training, validation, and test sets as described below:

Data	Training	Validation	Testing
Images	6000	1000	1000
Captions	6000	1000	5000

Table 1: DataSet

The Datasets undergoes rigid preprocessing so that it may be ready to be used with the model. With the caption data, the first step carried out is tokenization into individual words, followed by the removal of all punctuations, special characters, and numbers, as well as unnecessary white space. The images are scaled down to a fixed size, and then the pixel values are normalized to get the images ready to be used as input to the model. A unique word dictionary is thereafter created from these now cleaned-up captions by recording the vocabulary envisaged to be met during training. Continuing with the processing for the words in the caption, each token is translated to a number in a vector for tokenization and further padded for all tokens of the different images to be kept with the same length across all the samples. In short, this vast amount of preprocessing: consistency provided by making both the text data and the images ready for proper training.



ficker8k_images	dataset entry	2 years ago
split	manual split of captions	2 years ago
captions.txt	update caption file	2 years ago
translated_nepali_captions.txt	dataset entry	2 years ago

Figure 4: Dataset Folder Structure

2.5 Review and Analysis:

This document discusses the Encoder-Decoder model with a transformer architecture. In the model under discussion, the encoder has to process the input image for the extraction of its essential features, and the decoder generates meaningful captions from the features provided. The basic framework behind any model of image caption generation could be summarized as the translation of visual information into feature representations by the encoder and the conversion of these representations into coherent and contextually relevant textual descriptions by the decoder.

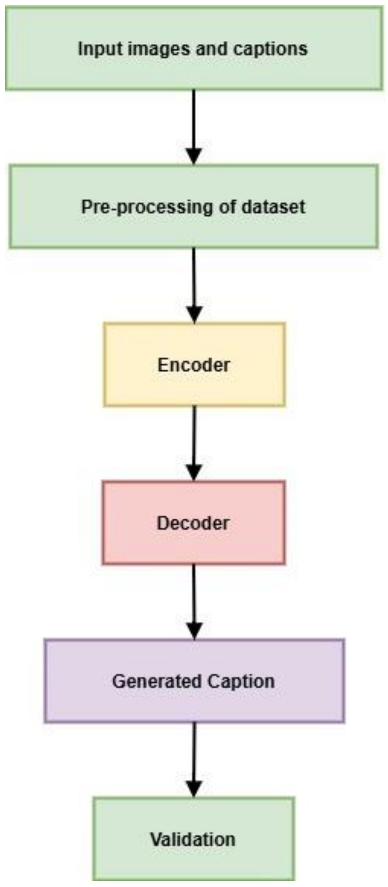


Figure 5: General block diagram

2.5.1 Model Design:

This work explores an end-to-end transformer model based on the encoder-decoder approach. First, the images are resized to a fixed resolution $X = R^{3\times H\times W}$ where *H* and *W* are the height and width of an image. The image is then divided into *N* patches, where, $N = \frac{H}{P} \times \frac{W}{P}$ with *P* as the patch size. Each patch is flattened and reshaped into a 1D patch sequence represented as $P_a=[p_1,p_2,...,p_N]$ .After flattening, a learnable 1D positional embedding is added to the patch features. The linear embedding layer further maps these patch features to the latent space, and this forms an input to the transformer encoder where the transformer encoder itself contains a positional feedforward sublayer and a multihead self-attention sublayer, each followed by layer normalization.

The captions get divided into words, which receive tokenization from the decoder side. These get combined with positional embeddings post-word embeddings, and this output of the encoder and the embeddings would provide the input for the decoder. In the architecture of the transformer decoder, the components follow this order: positional feedforward sublayer, multi-head cross-attention sublayer, and masked multi-head self-attention sublayer, with each of these sublayers followed by a normalization layer. The output from the last decoder layer passes through the linear layer to predict the next word. Its output dimension would thereby be set equal to the size of the vocabulary. In this research, two transformer-based models have been explored.

3. Solution and Problem-Solving Technique:

Image captioning can be done using different architectures and techniques. But the latest one is using the **transformer architecture**. The transformer architecture consists of the encoder and decoder parts.

The **encoder** part acts as a **vision transformer**, and the **decoder** part acts as the **GPT-2 architecture**, so overall we can call it **(Vision Transformer + GPT-2) image captioning**. The encoder part takes the images, and it captures the features of the image, and the decoder part, GPT, takes the text with the vision transformer out of the image features and generates the captions of the image.

Image captioning is used for e-commerce, social media like Facebook, scientific research, self-moving vehicles, OCR technologies, medical fields, and many more. The traditional rule and the probabilistic approach don’t help to generalize the model. The traditional models are unable to generate the captions for any language; the entire process also involves a lot of time for making the rules and framework. The images are also not processed properly since they consist of heavy matrices, and the bag of words doesn't capture the semantic meaning of words, and the output may always vary for the same image, so the transformer-based models come into play.

3.1 Algortithms Used in Our System:

The ViT & GPT models consist of various algorithms for image captioning, which are listed below:

1. **Positional Encoding:**
   Positional Encoding injects the positional information into the transformers that handle inputs in a paralleled manner. It does so by simply adding a positional embedding to learnt word vectors that consists of sin and cos of different frequencies, allowing the models to understand the order of tokens.

2. **Self-Attention:**

   Self-Attention enables every token (or image patch) to attend to other tokens in the sequence to build its representation. It is used to capture the semantic meaning between words in sentences. For every token, we convert them to queries, keys, and values vectors. The dot product of the query with all the keys gives the attention scores, which are softmaxed to get weights for the values. It allows the model to relate to each token.

3. **Multi-Head Attention:**

   Multi-Head Attention divides the query, key, and value vectors into separate subspaces for independent processing via different attention heads. The concatenated outputs are reshaped to form a single vector which allows the model to attend to different connections in the input.

4. **Masked Attention:**

   Masked Attention is a specific type of self-attention which masks some positions so that the model cannot attend to future tokens in the sequence. This is very useful for autoregressive tasks such as text generation.

5. **Normalization:**

   We use Layer Normalization to normalize the inputs of a layer by normalizing within the activations of the feature, which has been shown to increase training stability and convergence.

6. **Cross-Attention:**

   Cross-Attention is like self-attention but works over different sequences (i.e. when a decoder attends to the outputs of an encoder in seq-to-seq).

7. **Fully Connected Network:**

   A Fully Connected Network (FCN) is also known as a Dense Network, In this architecture, all the neurons from one layer are connected to all the neurons from the next layer. In this structure, all the nodes of one layer receive input from every node of the previous layer, which makes this a densely interconnected structure.

8. **Linear Layer & Softmax:**

   Linear Layer & Softmax A linear layer projects the final embeddings to logits, and the softmax layer passes these logits to probabilities used in classification or for the generation of outputs.

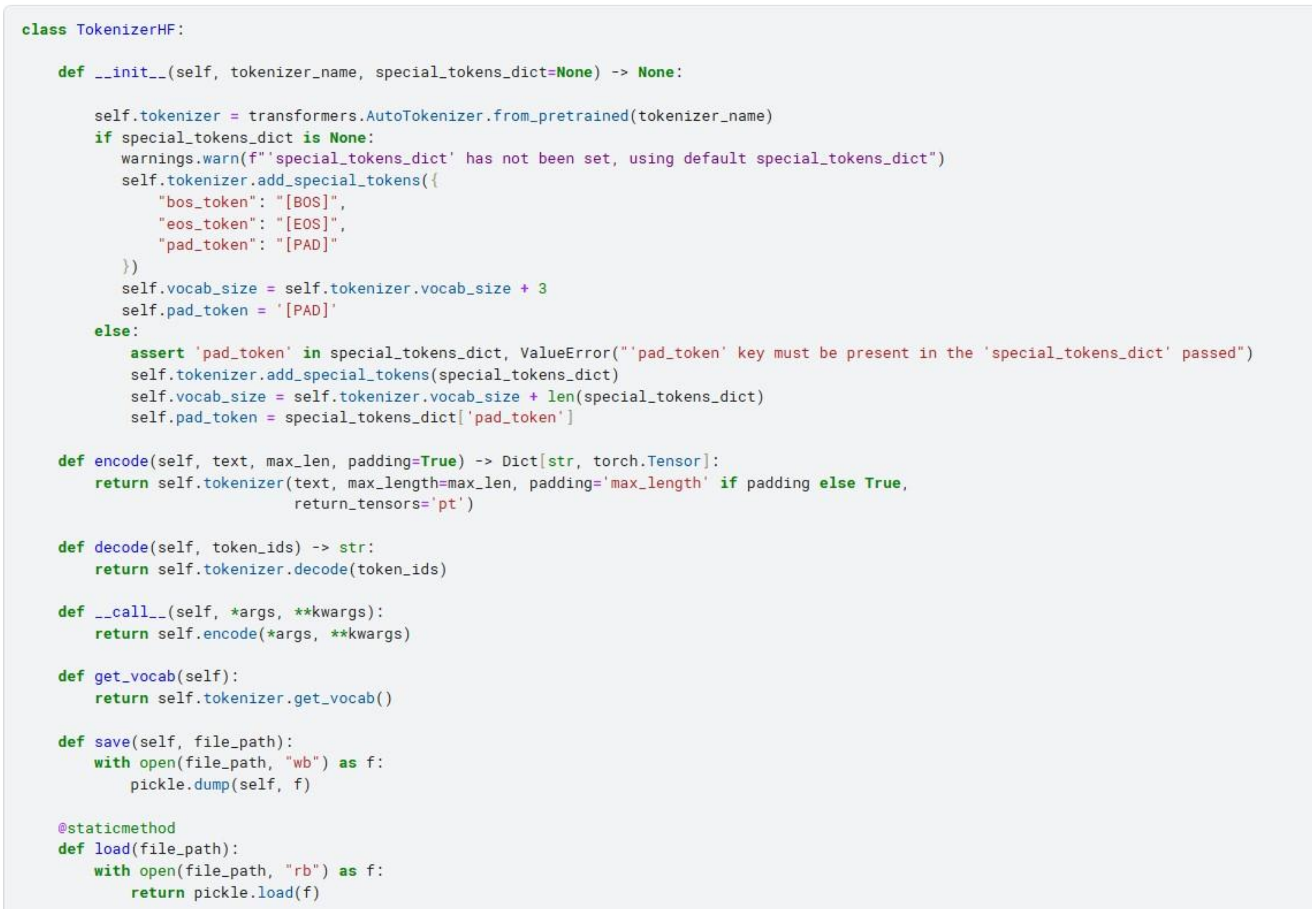
3.2 Pseudocodes:

```
CREATE a class ImageCaptioningSystem
DO
  DECLARE an instance variable imageEncoder as Vision Transformer
  DECLARE an instance variable textDecoder as GPT
  DECLARE an instance variable tokenizer for tokenization and detokenization

  CREATE a constructor ImageCaptioningSystem
  DO
    INITIALIZE imageEncoder with a pretrained Vision Transformer model
    INITIALIZE textDecoder with a pretrained GPT model
    INITIALIZE tokenizer with the tokenizer compatible with GPT
  END DO

  CREATE a function preprocessImage
  DO
    DECLARE a parameter image
    RESIZE the image to fixed dimensions (e.g., 224x224)
    CONVERT the image to a tensor
    NORMALIZE the image tensor to match Vision Transformer input requirements
    RETURN the preprocessed image
  END DO
```

```
CREATE a function generateCaption
DO
    DECLARE a parameter image
    CALL preprocessImage with image and store the result in preprocessedImage
    PASS preprocessedImage through imageEncoder to get imageFeatures
    INITIALIZE a sequence with a special token <BOS> (beginning of sequence)

    WHILE the sequence does not contain <EOS> and the length is less than a maximum
    DO
        PASS the sequence and imageFeatures to the textDecoder to generate the next token
        APPEND the generated token to the sequence
    END WHILE

    CONVERT the token sequence to text using tokenizer
    RETURN the generated caption as a string
END DO

CREATE a function trainModel
DO
    LOAD a dataset of image-caption pairs
    DECLARE parameters for training such as batch size, learning rate, and epochs
    FOR each epoch in total epochs
    DO
        FOR each batch of image-caption pairs
        DO
            PREPROCESS images in the batch
            TOKENIZE captions in the batch
            PASS the images through imageEncoder to get imageFeatures
            PASS imageFeatures and tokenized captions through textDecoder for training
            CALCULATE the loss between predicted and actual captions
            UPDATE model parameters using backpropagation
        END FOR
    END FOR
    SAVE the trained model
END DO

CREATE a function testModel
DO
    DECLARE a parameter testDataset
    FOR each test image in testDataset
    DO
        CALL generateCaption with the test image
        PRINT the test image and generated caption
    END FOR
    END DO
END DO

CREATE a main function
DO
    DECLARE an input image as inputImage
    CREATE an instance captioningSystem of ImageCaptioningSystem
    CALL captioningSystem.trainModel to train the model
    CALL captioningSystem.generateCaption with inputImage to generate a caption
    DISPLAY the caption
    CALL captioningSystem.testModel with a test dataset to evaluate the model
```

3.3 Screenshots & Descriptions:

3.3.1 Tokenizer:

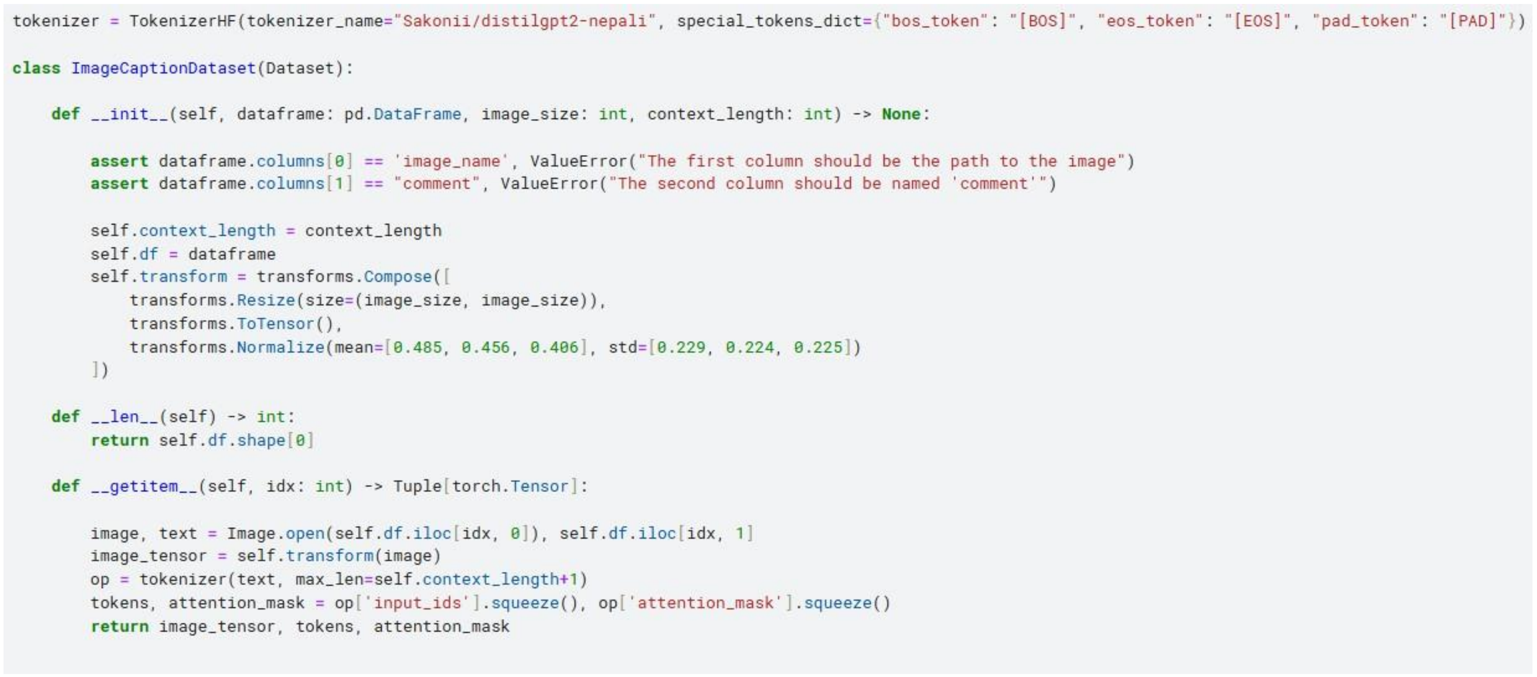


```python
class TokenizerHF:

    def __init__(self, tokenizer_name, special_tokens_dict=None) -> None:

        self.tokenizer = transformers.AutoTokenizer.from_pretrained(tokenizer_name)
        if special_tokens_dict is None:
           warnings.warn(f"'special_tokens_dict' has not been set, using default special_tokens_dict")
           self.tokenizer.add_special_tokens({
               "bos_token": "[BOS]",
               "eos_token": "[EOS]",
               "pad_token": "[PAD]"
           })
           self.vocab_size = self.tokenizer.vocab_size + 3
           self.pad_token = '[PAD]'
        else:
            assert 'pad_token' in special_tokens_dict, ValueError("'pad_token' key must be present in the 'special_tokens_dict' passed")
            self.tokenizer.add_special_tokens(special_tokens_dict)
            self.vocab_size = self.tokenizer.vocab_size + len(special_tokens_dict)
            self.pad_token = special_tokens_dict['pad_token']

    def encode(self, text, max_len, padding=True) -> Dict[str, torch.Tensor]:
        return self.tokenizer(text, max_length=max_len, padding='max_length' if padding else True,
                              return_tensors='pt')

    def decode(self, token_ids) -> str:
        return self.tokenizer.decode(token_ids)

    def __call__(self, *args, **kwargs):
        return self.encode(*args, **kwargs)

    def get_vocab(self):
        return self.tokenizer.get_vocab()

    def save(self, file_path):
        with open(file_path, "wb") as f:
            pickle.dump(self, f)

    @staticmethod
    def load(file_path):
        with open(file_path, "rb") as f:
            return pickle.load(f)
```

Description:

The TokenizerHF class is a wrapper class that deals with Hugging Face's tokenizer; it's supposed to convert raw text into tokenized sequences that can be fed into an NLP model. It also supports special tokens, such as **[BOS]**, **[EOS],** and **[PAD],** with provisions for their customization. This class contains methods for encoding text into token IDs as PyTorch tensors and decoding token IDs back to text. It also handles padding, retrieves the tokenizer's vocabulary, and offers methods to save and load the tokenizer. This class therefore simplifies text tokenization while providing flexibility for various NLP tasks.

3.3.2 Prepare Dataset:



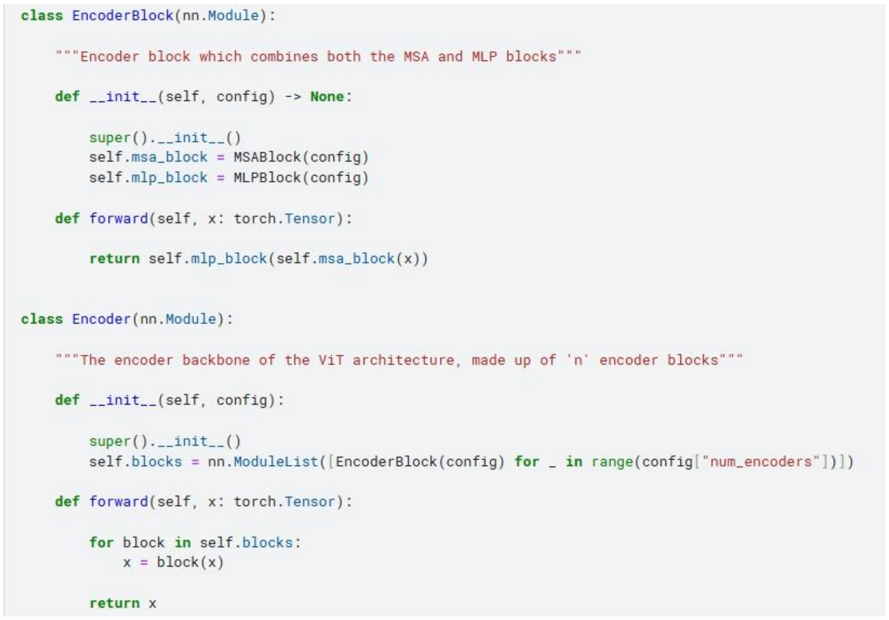
```python
tokenizer = TokenizerHF(tokenizer_name="Sakonii/distilgpt2-nepali", special_tokens_dict={"bos_token": "[BOS]", "eos_token": "[EOS]", "pad_token": "[PAD]"})

class ImageCaptionDataset(Dataset):

    def __init__(self, dataframe: pd.DataFrame, image_size: int, context_length: int) -> None:

        assert dataframe.columns[0] == 'image_name', ValueError("The first column should be the path to the image")
        assert dataframe.columns[1] == "comment", ValueError("The second column should be named 'comment'")

        self.context_length = context_length
        self.df = dataframe
        self.transform = transforms.Compose([
            transforms.Resize(size=(image_size, image_size)),
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
        ])

    def __len__(self) -> int:
        return self.df.shape[0]

    def __getitem__(self, idx: int) -> Tuple[torch.Tensor]:

        image, text = Image.open(self.df.iloc[idx, 0]), self.df.iloc[idx, 1]
        image_tensor = self.transform(image)
        op = tokenizer(text, max_len=self.context_length+1)
        tokens, attention_mask = op['input_ids'].squeeze(), op['attention_mask'].squeeze()
        return image_tensor, tokens, attention_mask
```

Description:

ImageCaptionDataset is the custom dataset class for image captioning, inheriting from the base PyTorch class of Dataset. It initializes a dataset with a given dataframe that may contain image paths and corresponding captions: the first column of this dataframe should contain the path to an image, and the second one contains the corresponding captions. The **__getitem__** method loads an image and converts it into a tensor, the caption text is tokenized with the previously initialized TokenizerHF object, which encodes this text into tokens and attention masks that will later be used in NLP tasks.

This Dataset object will return an image tensor along with its tokenized caption in a tuple of input IDs and attention masks that can serve as input in a neural network for an image captioning task. The images are resized and normalized before passing into the model.

3.3.3 Vison Transformer (Encoder):

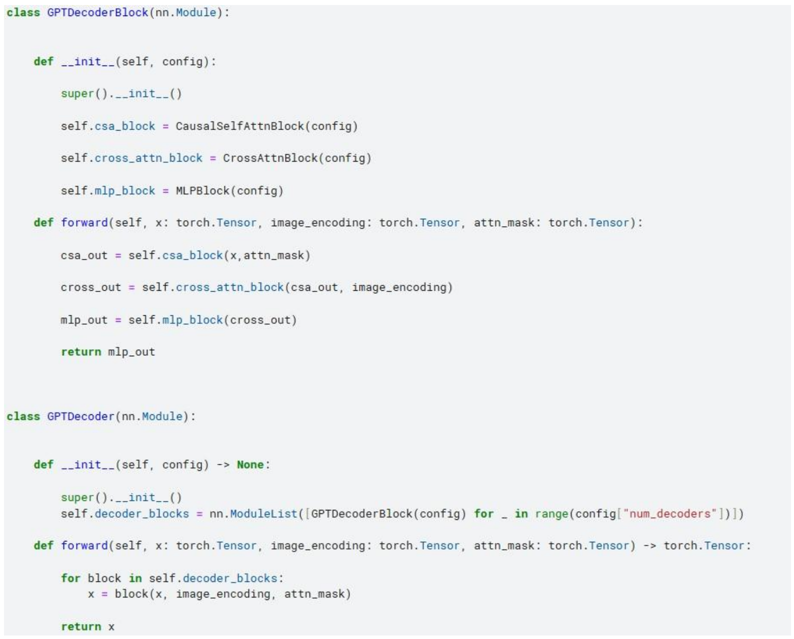


```python
class EncoderBlock(nn.Module):

    """Encoder block which combines both the MSA and MLP blocks"""

    def __init__(self, config) -> None:

        super().__init__()
        self.msa_block = MSABlock(config)
        self.mlp_block = MLPBlock(config)

    def forward(self, x: torch.Tensor):

        return self.mlp_block(self.msa_block(x))


class Encoder(nn.Module):

    """The encoder backbone of the ViT architecture, made up of 'n' encoder blocks"""

    def __init__(self, config):

        super().__init__()
        self.blocks = nn.ModuleList([EncoderBlock(config) for _ in range(config["num_encoders"])])

    def forward(self, x: torch.Tensor):

        for block in self.blocks:
            x = block(x)

        return x
```

Description:

The code handles the ViT model:one class, PatchEmbeddings, is developed for cutting the image into patches and projecting this embedding into higher dimensional space, class ViTEmbedding class just adds class and position tokens to those patch embeddings, class MSABlock has multi-self attention, whereas the MLPBlock applies the feed-forward neural networks with residual, EncoderBlock then combines the attention-instituted layer along with feed forward and the Encoder stacked multiple of the encoder blocks. The final ViT outputs the representation of the [CLS] token for image classification.

3.3.4 GPT 2 (Decoder):



```python
class GPTDecoderBlock(nn.Module):


    def __init__(self, config):

        super().__init__()

        self.csa_block = CausalSelfAttnBlock(config)

        self.cross_attn_block = CrossAttnBlock(config)

        self.mlp_block = MLPBlock(config)

    def forward(self, x: torch.Tensor, image_encoding: torch.Tensor, attn_mask: torch.Tensor):

        csa_out = self.csa_block(x,attn_mask)

        cross_out = self.cross_attn_block(csa_out, image_encoding)

        mlp_out = self.mlp_block(cross_out)

        return mlp_out


class GPTDecoder(nn.Module):


    def __init__(self, config) -> None:

        super().__init__()
        self.decoder_blocks = nn.ModuleList([GPTDecoderBlock(config) for _ in range(config["num_decoders"])])

    def forward(self, x: torch.Tensor, image_encoding: torch.Tensor, attn_mask: torch.Tensor) -> torch.Tensor:

        for block in self.decoder_blocks:
            x = block(x, image_encoding, attn_mask)

        return x
```

Description:

This architecture is based on transformers. It creates token embeddings by using GPTEmbedding and further adds positional encodings and dropout for regularization. CausalSelfAttnBlock implements causal self-attention together with safe softmax for numerical stability and handles padding tokens by means of a mask. CrossAttnBlock introduces cross-attention in order to concatenate token embeddings with image encoding. GPTDecoderBlock includes a causal self-attention, encoder cross-attention, and feed-forward neural network, whereas GPTDecoder does the same but stacks several decoder blocks to process sequentially. Finally, the last version of the GPT model will be equipped with a classification head to predict tokens and a mask generation method in order to handle causality and padding masks. Herein, an architecture has been developed that can handle natural language and image data jointly.

3.3.5 ImageCaption Model:

```
class ImageCaptionModel(nn.Module):

        """This class is the main class. Puts together both ViT and GPT. Lot more useful functions need to be added if
someone uses them"""

        def _init_(self, config) -> None:

                super()._init_()

                self.device = config['device']

                self.is_vit_pretrained = False

                if config['vit_kwargs']["pretrained_model_name"] is not None:

                        self.is_vit_pretrained = True

                        self.vit = timm.create_model(
                                model_name = config['vit_kwargs']["pretrained_model_name"], pretrained=True,
                                num_classes = 0,
                                global_pool = 'avg'
                        )
                        config["vit_kwargs"]["d_model"] = self.vit.embed_dim else:
                        self.vit = ViT(config['vit_kwargs'])

                self.gpt = GPT(config['gpt_kwargs'])

                self.dimension_mapping_layer                    =                    nn.Linear(config["vit_kwargs"]['d_model'],
config["gpt_kwargs"]['d_model'])

        def forward(self, image: torch.Tensor, tokens: torch.Tensor, attn_mask: torch.Tensor): image =
                image.to(self.device)

                image_encoding = self.vit(image)

                    dimension_mapped_image_encoding = self.dimension_mapping_layer(image_encoding[:, None, :])

                return self.gpt(tokens, dimension_mapped_image_encoding, attn_mask)


        @torch.inference_mode() def
        generate(self,
                          image: torch.Tensor,
                          sos_token: int, eos_token:
                          int, max_len: int = 40) :

                image = image.to(self.device)

                image_encoding = self.vit(image)

                    dimension_mapped_image_encoding = self.dimension_mapping_layer(image_encoding[:, None, :])

                tokens = torch.tensor(data=[[sos_token]], requires_grad=False).to(self.device)
        attn_mask = torch.tensor(data=[[1]], requires_grad=False).to(self.device)

        while tokens.shape[1] < max_len and tokens[0, -1] != eos_token:

            logits = self.gpt(tokens, dimension_mapped_image_encoding, attn_mask)  # 1, N+1, D_MODEL

            next_token = torch.argmax(logits[0, -1, :], dim=0).item()

            # Ensure next_token is on the correct device before concatenating
            next_token_tensor = torch.tensor([[next_token]], requires_grad=False).to(self.device)

            tokens = torch.cat((tokens, next_token_tensor), dim=-1)

            attn_mask             =             torch.cat((attn_mask,             torch.tensor([[1]],
requires_grad=False).to(self.device)),  dim=-1)

        return list(tokens[0])

    @classmethod
    def from_pretrained(cls, checkpoint, device):

        if not os.path.exists(checkpoint):
            raise FileNotFoundError(f"{checkpoint} does not exist")

        cp = torch.load(checkpoint, map_location=device)
        cp['model_config']['device'] = device
        cp['model_config']['vit_kwargs']['device'] = device
        cp['model_config']['gpt_kwargs']['device'] = device

        model = cls(cp['model_config'])
        model.load_state_dict(cp['model_state_dict'])
        model = model.to(device)
        return model
```

Description:

The ImageCaptionModel class performs image captioning using Vision Transformer (ViT) combined with GPT. It initializes either a pre-trained ViT or a custom ViT model. A mapping layer is added on top of ViT to map its output to the input of GPT. The forward method takes an image through the ViT and the mapping layer and feeds it into GPT with tokens and an attention mask. This is done by the generate method by making iterative predictions of tokens until it reaches either an end-of-sequence token or the maximum length. The from_pretrained method loads a model from a checkpoint. It updates the configuration and device settings.

3.4 Development Platform & Libraries Used:

This model was created using Google Colab, a free cloud-based platform to get access to high-end GPUs and TPUs to train such high-scale models like a Vision Transformer and GPT. Google Colab provides an interactive Jupyter notebook-like environment, where one would feel free to mix and match Python code, perform its execution along with data, and train the model by not feeling any obstruction at the local resource level. Besides, it enables direct interaction with Google Drive-stored files, which is very handy regarding dataset and model checkpoints storage and sharing.

The following libraries were used when developing the model for Image Captioning. These are enumerated below.

PyTorch:

Purpose: This is the main deep learning framework that is used in the building, training, and testing of a neural network model. PyTorch supports automatic differentiation. It is highly efficient in tensor computation and is GPU-accelerated, allowing for deep learning on a large number of data examples.

timm (PyTorch Image Models):

Purpose: This is a library hosting several pre-trained Vision Transformers models, ViT, used for feature extraction of the images. It provides different state-of-the-art vision models that can be easily integrated with your specific task.

torchvision:

Purpose: These are generally used for image processing tasks, such as image transformations, data augmentation, and the use of pre-trained models in conjunction with PyTorch on vision tasks.

Hugging Face Transformers:

Purpose: For implementing the GPT model to perform text generation. The Hugging Face Transformers library is widely used when doing Natural Language Processing tasks since it allows the user to painlessly access pre-trained models and efficient fine-tuning workflows.

numpy:

Numerical operations like manipulating or transforming arrays of data - in this case, image tensors and tokens of text, have been used.

Matplotlib:

Purpose: Very useful to visualize data, performance, and results of models; producing training curves, for instance, or displaying images with their generated captions.

4) Conclusion:

This project on Image captioning with Vision Transformers using GPT models is an effective demonstration of how we can combine Machine learning and Deep Learning technologies to create a solution for a real-world problem. By understanding both visual and textual information, the solution produces accurate and contextually relevant captions for a variety of images.

4.1 Analysis of the Work Done:

The implementation build the image captioning model all from scratch, showing good practice in building and training models using deep learning. The Vision Transformers were used for obtaining relevant visual features whereas the text generation part made use of GPT architecture for generating a semantically relevant human-readable description.

To enhance the tokenization process and the generation of captions specifically for the Nepali language, a pre-trained BERT Nepali tokenizer was incorporated. This emphasizes the point that the highly personalized solution was crafted without iterative use of tuned-focused, pre-trained models, and that it reflects new, improved capabilities in model design and performance over the rule-based or probabilistic approaches of the past.

4.2 Addressing Real-World Problems:

This solution supports many applications including:

Visual Impairments: Helping visually impaired users of all kinds by providing relevant image descriptions.

E-commerce: Cataloging and searching the products with product descriptions automation.

Healthcare sector: Used to support the analysis of medical imaging by providing descriptive outputs

Autonomous Systems: Improves perception and decision-making for self-driving cars and robotics.

Social Media: Facebook and Instagram could apply this technology to auto-generate captions for all uploaded images to increase accessibility and user attraction. And automatic captions help people discover your content as well as improve its personalization.

4.3 Further Work:

Further improvements may be done on:

More Multilingual Generating: Scaling up the model to output captions in various languages with increased fluency and accuracy.

Real-time Captioning: Transforming the work done in time-efficient manner so that it can be used in some real-time scenarios.

Multi-Modal Integration: Audio-Visual streams for Multimodal Captioning

Domain Adaptation: Finetuning the model for specific domains, including scientific, surveillance, or educational content.

5) References:

Computer Vision for Visual Computing: Techniques and Applications. (1998). *Computer Vision and Image Understanding*, 71(2), p.153. doi:https://doi.org/10.1006/cviu.1998.0714.

Rao, D. and McMahan, B. (2019). *Natural Language Processing with PyTorch*. 'O'Reilly Media, Inc.'