LONDON METROPOLITAN UNIVERSITY

islington college
(इस्लिङ्टन कलेज)

Module Code & Module Title

CT6057NI Computer Vision

60% Individual Coursework

Submission: Final Submission

Academic Semester: Spring Semester 2025

Credit: 15 credit semester long module

Student Name: Bibek Poudel

London Met ID: 22067316

College ID: NP01AI4A220032

Assignment Due Date: Wednesday, May 7, 2025

Assignment Submission Date: Wednesday, May 7, 2025

Submitted To: Shreejan Shrestha

I confirm that I understand my coursework needs to be submitted online via MST Classroom under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.

Table of Contents

Table of Tables

Table of Figures

1. Introduction

Artificial Intelligence is defined as the ability of a machine to learn, understand, and solve problems like human being. Human being uses different senses as modality to understand the environment around them. Human also understand the information from different modality including speech, text, visuals, sound and sensation. Similarly, artificial intelligence can also understand different modalities through different paradigms mainly computer vision and natural language processing. Computer vision provides the vision sense to the machine based on artificial intelligence which allow spatial understanding along with various relevant information extraction. Natural language processing allows machines to understand and generate the human language along with two-way communication in form of text and speech. Image captioning falls under intersection of these two paradigms where CNNs-based (Convolutional Neural Network) feature extractor extract visual information from the image and RNN-based LSTMs (Long Short-Term Memory) deals with the caption text generation tasks with limited capability to learn long-term dependencies efficiently. However, transformer architecture developed by Google (Ashish Vaswani, 2017) in 2017 solved the problem of dealing with the context and intent through learning long term dependencies through its attention mechanism. This architecture allows the most efficient way to deal with textual data and allow the development of representation models, embedding models, generative models and lately vision-language models. The paper "An Image is Worth16x16 Words (Alexey Et al. Lucas Beyer, 2020) proposed the use of pure transformer applied on a images through patches for image classification task which opened the new doors for integration of natural language processing and computer vision for a real world tasks such as image captioning, visual question answering along with multi-modal large language models.

Vision Transformer is used for extracting the visual information which later fed to transformer-based decoder named DistilBERT to generate the best caption for an image. Such architecture can efficiently generate the captions for the visual data which are contextually correct instead of traditional computer vision techniques such as object detection, image classification which only generate very minimum information that leads to scene understanding ability for artificial intelligence. Open-source image captioning

dataset such as COCO Captioning, Flickr8k and Medical MRI images with analysis can be trained in the above-mentioned architecture to develop an image captioning model that can be used in real world. This coursework will implement similar system to leverage the capabilities of transformer architecture in the intersection of computer vision and natural language processing.

1.1. Aims and Objectives

The main goal of this coursework is to research and develop an advanced image captioning system using transformer architectures to close the gap between computer vision and natural language processing, such that the machine can produce more contextually accurate and semantically meaningful captions for images.

- To generate high-quality textual descriptions of images, use a transformer-based decoder (i.e., BERT) while the model leverages long-range dependencies and contextual relationships that exist in the captions.
- Train the new architecture on common image captioning datasets like COCO Captions or Flickr8k and quantify its performance using metrics such as BLEU, METEOR and CIDEr, which measure caption quality and relevance
- Close the bridge to domain-related datasets like medical MRI images with analysis reports, to illustrate its versatility and possible boundary areas like healthcare, education, and assistive technologies.
- Utilize a Vision Transformer to extract spatial and semantic information from images by dividing them into patches and processing them through self-attention mechanisms, replacing traditional CNN-based feature extraction methods.

1.2. Related Project and Background

Image captioning has become a fundamental aspect at the crossroad of computer vision and natural language processing, with the goal to produce natural language descriptions for visual content. In recent years, techniques have been developed to help alleviate the dependency on generic descriptions by considering real-world knowledge in captioning models. Cheng et al. proposed K-Replay, a technique exploiting Vision-Language Pre-Training (VLP) models to preserve the VLP pre-training knowledge when fine-tuning for image captioning (Ka Leong Cheng, 2023).Their method alleviates knowledge hallucinatio

n and generic bias by adding a knowledge prediction task and a distillation constraint, so that the model can generate more context-aware and informative captions. To benchmark their method, they proposed KnowCap, a benchmark dataset, with a focus on specialist knowledge of landmarks, brands, foods and characters. The experiments showed substantial improvements in terms of CIDEr and knowledge recognition accuracy, which proves the possibility of incorporating external knowledge into captioning systems. Similarly, Radford et al. studied learning transferable visual models from natural language supervision and showed that by aligning images with descriptions of images, zero-shot capabilities were supported on a wide range of downstream tasks (Alec Radford, 2021, Feb 26). These results illustrate the necessity of the use of web-harvested data and cross-modal alignment for boosting semantic richness and generalizability of image captioning systems.

Concurrently, multitask frameworks for visual captioning have been proposed to enhance the captioning performance by jointly training with complementary tasks such as object detection. Basak et al. proposed TICOD, a transformer-based model, which combined image captioning and object detection into a single backbone architecture (Debolena Basak, 2024). Through jointly optimizing the losses of the two tasks, their model exploits visual and contextual information shared between the two, which consequently results in better caption quality, which can be seen in a 3.65% improvement in BERTScore on the MS-COCO dataset. Moreover, there have been attempts towards improving model efficiency, also with Rampal et al. Neural Module Networks, strategies to prune and compress them in the context of CNN-LSTM based captioning models (Harshit Rampal, December 17, 2020). They can also achieve large reductions in model size and inference time for increasing BLEU scores, indicating that it is possible to deploy resource-efficient captioning systems on edge. These studies together highlight the continuous development of image captioning models with an emphasis on knowledge incorporation, multitask learning and the use of efficient architectures to further push the state-of-the-art.

2. Methodology

2.1. Algorithms Used

2.1.1. Transformer: Text Decoder

Transformer is a deep neural network that integrates self-attention mechanism to extracts and understand the contextual relationship between the sequential data proposed in a paper named "Attention Is All You Need". Unlike traditional RNNs (Recurrent Neural Networks), LSTM (Long Short-Term Memory), transformer architecture proposed by (Ashish Vaswani, 2017) excel in handling longer dependencies between input sequence elements and enable parallel processing. Such ability of transformer has great potential in Natural Language Processing along with wide range of tasks such as computer vision, speech and audio processing and Internet of Things. Initial transformer architecture was developed based on auto-regressive sequence transduction model, composed to two modules named Encoder and Decoder. Each module consists of layers with integrated attention mechanism. Attention Mechanism is executed in parallel multiple times within the transformer architecture, which explains the presence of Attention Heads (Ashish Vaswani, 2017).

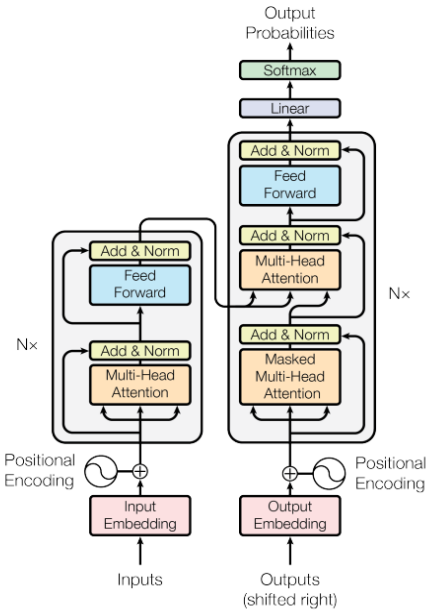

Figure 1: Transformer Architecture

The developed project utilises encoder-based language representation model DistilBERT (Victor Sanh, 2019) as text decoder. This model is the distilled version of the bidirectional language representation model BERT (Bidirectional Encoder Representations from Transformers) which alleviated the unidirectionality constraints during the fine-tuning of the pre-trained models by introducing masked language model pre-training objective (Jacob Devlin, 2019). This model is first pre-trained on a large corpus of text such as Wikipedia and BookCorpus through unsupervised learning. This pre-trained model can be finetuned on specific downstream tasks including image captioning, sentence completion, translation and text generation (Jacob Devlin, 2019).

2.1.2. Vision Transformer: Image Encoder

Image captioning task requires the challenge of analysing the image and generating the relevant text based on the analysed image. This task requires notion of attention to decide what to describe and in which order (Sen He, 2020). Transformer architecture is adapted to the structure of images from original textual data. These vision transformers show that the reliance on CNN is not necessary and purely transformer can be applied directly to the sequence list of image patches can can perform very well in image classification task (Sen He, 2020). Vision Transformer (ViT) initially pre-trained on large amounts of visual data then transferred to downstream datasets such as ImageNet, CIFAR-100 achieved state-of-the-art performance with significantly less computational power to train (Alexey Et al. Lucas Beyer, 2020). Self-attention allows ViT to integrate information across the entire image even in the lowest layers (Alexey Et al. Lucas Beyer, 2020). Various computer vision tasks image classification, object recognition and detection, image generation, image translation and image segmentation are made possible with the use of vision transformer. The vision-encoder process and encode the image in form of feature vectors in an embedding space which is later fed to the text decoder. ViT outputs the sequence of high-level features that represents different part of images. Those visual features are passed to the DistilBERT-based text decoder to generate the contextual caption (Xiujun Li, 2020).

The advancement of efficient image captioning methods through the ViT-based technique marks a major merger of computer vision with natural language processing technology. The former generation of successful methods used Convolutional Neural Networks alongside Recurrent Neural Networks (LSTMs in particular) to analyse image spatial features and produce text descriptions sequenced sequentially (Yongqing Zhu, 2016). Past CNN-RNN models delivered

groundbreaking results but they encountered problems with detecting distant relationships between elements besides sustaining focused attention on central regions. The Transformer model transformed NLP when it first appeared because it introduced a new way of processing information. Research teams adapted this effective self-attention mechanism from Transformers to visual data which directly produced the Vision Transformer success in image captioning applications.

2.1.3. Tokenization

Tokenization is the process of converting input data into corresponding tokens. This process is domain dependent i.e. for NLP where the transformer is widely used the text is converted into token using byte pair encoding (Awan, 2024), in computer vision the image is first dissected int non overlapping patches and then linearly projected to obtain token (Jaihyun Lew, 2024), in case of speech processing the speech signal is converted into spectrogram which is then treated as an image and does the tokenization in a similar manner to that of images (Milvus, 2024).

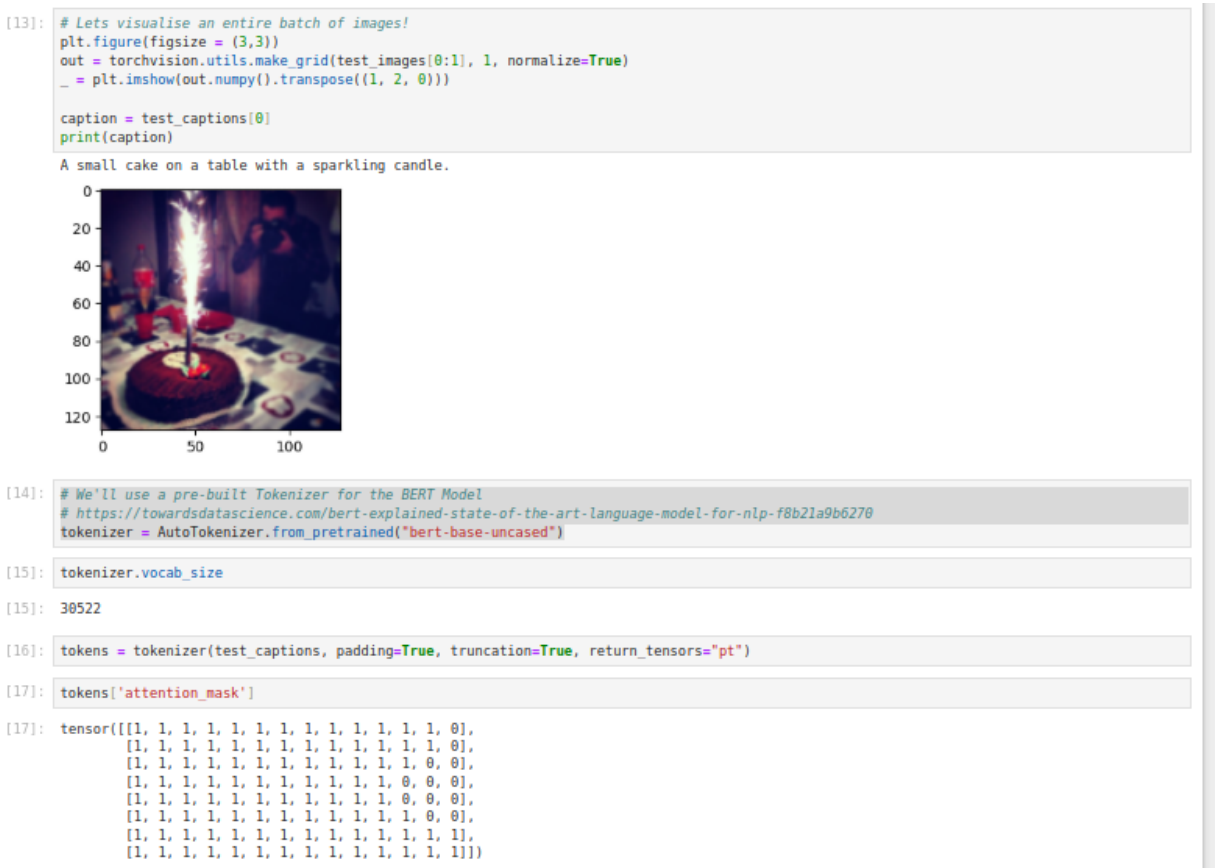

Figure 2: Tokenization based on BERT

The process of tokenization is one of the keys and overlooked part in transformer-based modelling. This has not trivial consequences when training and inferencing of such models. Tokenization is also important in multi modal AI where different modalities (Zhen Yang, 2023) are converted into tokens and then allow to cross communicate and infer from each other, hence understand and perceive the world much like humans do, this is a viable step in the making of AGIs.

2.1.4. Optimization Algorithm

The Adam optimization algorithm (Diederik P. Kingma, 2014) is a stochastic gradient descent method created to improve the efficiency of deep learning. Adam instead computes individual learning rates for each parameter using only first moment (the mean) and the squared second moment (the uncentered variance) of the gradients (using the exponentiated forms of the selected decay rates beta1 (commonly set to around 0.9) and beta2 (typically 0.999). This dual-memory construction combines the benefits of AdaGrad for sparse gradients and RMSProp for noisy, non-stationary objectives and achieves robust convergence on complex non-convex problems (Brownlee, 2021). Its default setting (with learning rate=0.001 and epsilon=1e-8 to prevent numerical instability) achieves a balance between computation efficiency and memory consumption that is most suitable for large-scale applications in computer vision and natural language processing. Adam, being widely used as a default optimizer in commonly used frameworks such as TensorFlow and PyTorch, with its bias-corrected estimates, immediately attenuates the early training bias towards zero and learns faster on tasks such as image captioning, generative modeling. It is empirically validated on benchmarks (e.g., MNIST, CIFAR-10) that it can speed up the training and is less sensitive to hyperparameter tuning (Diederik P. Kingma, 2014). By reconciling theoretical sophistication with practical utility, Adam maintains its position as a far-reaching tool in deep learning, connecting algorithmic novelty with practical deployment requirements.

2.1.5. Cross Entropy Loss Function

Cross-entropy loss is a key concept in training ML models, especially for classification problems like image captioning. It calculates the discrepancy between the predicted and target probability distributions and directs the model to reduce errors while training (Pykes, 2024). In image captioning, cross-entropy measures whether a predicted caption is close to the ground-truth description and punishes bad predictions when the model is confident in having made a correct

one. This way, the model can improve its knowledge of visual-textual correspondences in the long run. Cross-entropy gives us an easily optimized objective by averaging the number of bits required to encode a correct label when using predicted probabilities. Define this loss in a framework such as PyTorch or TensorFlow so that this loss can be utilized in backprop, and the model can learn richer representations (Pykes, 2024). As the model successively minimizes cross-entropy, it also produces more accurate, contextual captions, thereby boosting performance. This practice of fine-tuning its model parameters is why cross-entropy is necessary, fundamental, and crucial for obtaining good quality results in image captioning systems.

2.2. Libraries Used

2.2.1. PyTorch

PyTorch is a machine learning library that provides an imperative and Pythonic programming style that supports code as a model, makes debugging easy and is consistent with other popular scientific computing libraries such as Pandas and NumPy, while remaining efficient and supporting hardware accelerators such as GPUs (Adam Paszke, 2019). PyTorch also enables immediate execution of dynamic tensor computations with automatic differentiation and GPU acceleration (Adam Paszke, 2019). Furthermore, PyTorch boasts a rich ecosystem including libraries like TorchVision for computer vision, TorchText for NLP, and TorchAudio for audio processing, along with tools such as TorchServe for easier model deployment into production.

Tensor is the fundamental building block in PyTorch, a multi-dimensional array with the feature of automatic differentiation with auto grad and GPU acceleration. PyTorch's object-oriented

approach allows complex architectures to be built by combining smaller nn.Module instances. PyTorch also provides native support for torch.nn which allows the integration of pre-trained models. Additionally, it's automatic differentiation engine simplifies the training process enabling efficient backpropagation through both modality of the data through different encoder and decoder based on transformer. Its extensive ecosystem also includes tools for data preprocessing, GPU acceleration, and model deployment, making it a comprehensive framework for implementing and fine-tuning state-of-the-art image captioning systems.

2.2.2. Hugging Face Transformer

Hugging Face Transformers is a flexible library that provides pre-trained models for Natural Language Processing, Computer Vision, Audio and Multimodal tasks. As stated in the accompanying text, it supports training models on the users own data, building inference applications, and producing text with large language models (LLMs) and vision-language models (VLMs) (HuggingFace, 2024). Key capabilities include a Pipeline class which allows easy inference sequences for tasks such as text generation and image segmentation, the Trainer API making it easy to efficiently train models, supporting mixed precision, distributed training, and FlashAttention optimizations, as well as the generate method which provides fast and customisable text generation, with options to stream back and forth. To make sure that it is easy to learn and use, we designed Transformers as easy-to-extend: it is built in a modular way using configuration, models and feature extractors classes. It lowers the barriers to entry to state-of-the-art performance, as it uses a pretrained model (HuggingFace, 2024). The Hugging Face Hub is also part of the accessibility aspect of the library, enabling the community to find and use models in seconds. With our community actively contributing new models, datasets, & spaces, Transformers delivers a versatile approach for developers & researchers collaborating on AI.

Figure 3: Hugging Face

2.2.3. TorchVision

TorchVision is a library that simplifies the process of writing computer vision workflows with PyTorch, providing building blocks necessary to design and create data loaders that can be used for training and testing, including code and pre-trained models. It is possible to access a variety of datasets (such as COCO, ImageNet) through torchvision. datasets that allow the easy access to image-caption pairs or labeled data, as it is the case of CocoCaptions for loading training and validation data. For pre-processing, we use torchvision. transforms input classes such as Resize, ToTensor, and Normalize, to standardize images (e.g., rescale, crop, normalize pixel values) and compose augmentations for better performance (e.g., random erasing to enhance model generalization). TorchVision also includes pre-trained models for various computer vision tasks such as classification, segmentation, and object detection, empowering users to use the widely known architectures out-of-the-box with no need for training from scratch (PyTorch, 2024). Combining the above elements with other utilities and the ability to load common datasets such as Imagenet, CIFAR10, MNIST, etc., TorchVision provides a solid foundation for researchers and engineers to develop state-of-the-art computer vision applications and speed up the current learning process by providing datasets, loaders, as well as dataloaders based off of the torch datasets model which has been an awesome acceleration to development (PyTorch, 2024).

Figure 4: TorchVision

2.2.4. NLTK (Natural Language ToolKit)

Natural Language Toolkit (NLTK) is a leading framework (NLTK.ORG, 2024) for building Python programs to work with human language data. It includes a broad set of analysis modules and offers access to over 50 manually annotated corpora and lexicon resources (e.g., WordNet), as well as to well-engineered tools in the areas of tokenization, part-of-speech tagging, syntactic parsing, semantic role labelling, and discourse parsing. The architecture is designed to be extensible, and it can be integrated with industrial NLP processing tools and frameworks for the scalable text processing pipeline. NLTK's educational path is demonstrated by its secondary service as a textbook supplement for applications of NLP in Python such as in Natural Language Processing with Python, where it provides personal narratives to push natural language thru the academic exercises within the text to stretch ideas from the pure theoretical linguistics to the practical programming level (Jablonski, 2024). The library also provides computation of critical NLP task evaluation metrics like the BLEU, METEOR and NIST scores—illustrated in the code snippet above—which measure fidelity of machine generated text wrt human references (NLTK.ORG, 2024). By stripping away advertisements and enabling researchers to focus on the underlying data, NLTK offers a viable design alternative that encourages synergy between linguists, engineers and researchers, as well as cross-platform support and community contributions. Its continued relevance coupled with being used both as a teaching aid and a benchmark for advancement NLP approaches in laboratory and corporate environments.

Figure 5: NLTK Package

3. Implementation Steps

The developed image captioning system based on transformer have a systematic pipeline which includes the steps including data collection, data pre-processing, feature extraction, architecture development, model training and validation with inference. These steps allow the seamless development of the computer vision-based project. This section will describe how the system was developed with different steps.

3.1. Dataset Collection and Pre-processing

COCO captions dataset is used for developing the image captioning system based on transformers architecture. The coco captions 2014 dataset have 83k/41k train/val split with the total number of images exceeding 100,000 (Tsung-Yi Lin, 2014). The small subset from the dataset is used for training the model due to computational resource constraints. 8000 images were used for training the model with 18 million parameters.

[9]:
```
import os
import json

# Paths
data_set_root = '/home/transformer_/Downloads/dataset_captioning'
train_set = 'train2014'
validation_set = 'val2014'
train_image_path = os.path.join(data_set_root, train_set)
train_ann_file = '{}/annotations/captions_{}.json'.format(data_set_root, train_set)
val_image_path = os.path.join(data_set_root, validation_set)
val_ann_file = '{}/annotations/captions_{}.json'.format(data_set_root, validation_set)

# Function to update annotations based on available images
def update_annotations(ann_file, image_dir):
    with open(ann_file, 'r') as f:
        data = json.load(f)

    # Find images that exist on disk
    existing_files = set(os.listdir(image_dir))
    existing_images = [img for img in data['images'] if img['file_name'] in existing_files]
    existing_image_ids = {img['id'] for img in existing_images}
    existing_annotations = [ann for ann in data['annotations'] if ann['image_id'] in existing_image_ids]

    # Create new annotation file with only existing images
    new_data = data.copy()
    new_data['images'] = existing_images
    new_data['annotations'] = existing_annotations

    # Save the new annotation file
    backup_path = ann_file + '.backup'
    if not os.path.exists(backup_path):
        os.rename(ann_file, backup_path)
    with open(ann_file, 'w') as f:
        json.dump(new_data, f)

    print(f"Updated {ann_file} to include only {len(existing_images)} existing images")

# Update both training and validation annotations
update_annotations(train_ann_file, train_image_path)
update_annotations(val_ann_file, val_image_path)
```

```
Updated /home/transformer_/Downloads/dataset_captioning/annotations/captions_train2014.json to include only 7998 existing images
Updated /home/transformer_/Downloads/dataset_captioning/annotations/captions_val2014.json to include only 3000 existing images
```

Figure 6:COCO dataset collection

Images are transformed using torchvision's utilities by resizing the images into 128x128 pixels for constant spatial dimension during training (Hossein Talebi, 2021). Additionally, resized images are converted into PyTorch tensor along with the normalization of each RGB channel for speeding up the training (Fuhl, 2021). Normalization is performed on each channel by subtracting the mean and dividing it by standard deviation.

Data processing and Tokenization

```
]: # Simple transform class to randomly sample one of the five captions
class SampleCaption(nn.Module):
    def __call__(self, sample):
        rand_index = random.randint(0, len(sample) - 1)
        return sample[rand_index]
```

Figure 7: Data preprocessing and tokenization

```
[11]: # Define the transformation pipeline for training images.
train_transform = transforms.Compose([transforms.Resize(image_size),
                                      transforms.RandomCrop(image_size),
                                      # Convert the PIL Image to a tensor with shape (C x H x W)
                                      transforms.ToTensor(),

                                     # They are used to normalize the input image so that
                                      # the distribution of pixel values matches what the model was trained on.
                                      # those decimal numbers are normalization constants of Imagenet Dataset
                                      transforms.Normalize(mean=[0.485, 0.456, 0.406], # RGB mean values for ImageNet
                                                           std=[0.229, 0.224, 0.225]),# RGB standard deviation values for ImageNet
                                     # Apply random erasing with a probability of 50%
                                      transforms.RandomErasing(p=0.5)])

# Define the transformation pipeline for validation and testing images
transform = transforms.Compose([transforms.Resize(image_size),
                                transforms.CenterCrop(image_size),
                                transforms.ToTensor(),
                                transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                     std=[0.229, 0.224, 0.225])])

# Create the training dataset using CocoCaptions
# Parameters:
#   root: path to the training images
#   annFile: path to the training annotations file
#   transform: apply train_transform to the images
#   target_transform: function to transform the captions
train_dataset = datasets.CocoCaptions(root=train_image_path,
                                      annFile=train_ann_file,
                                      transform=train_transform,
                                      target_transform=SampleCaption())

val_dataset = datasets.CocoCaptions(root=val_image_path,
                                    annFile=val_ann_file,
                                    transform=transform,
                                    target_transform=SampleCaption())

# Create the training data loader
# Parameters:
#   dataset: the training dataset
#   batch_size: number of images per batch
#   shuffle: whether to shuffle the dataset before loading
#   num_workers: number of subprocesses to load data
data_loader_train = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)
data_loader_val = DataLoader(val_dataset, batch_size=batch_size, shuffle=True, num_workers=0)
```

```
loading annotations into memory...
Done (t=0.04s)
creating index...
index created!
loading annotations into memory...
Done (t=0.01s)
creating index...
index created!
```

Figure 8:Image Preprocessing with Torchvision Transforms

Moreover, dataset contained 5 captions for each image, only single caption was used for training the model by efficient mapping between the annotation file and the respective image. During preprocessing step, the captions are converted into tokens, individual words. The tokens are also mapped with the numeric value where the word appearing more at least five times gets unique indices whereas the special blank token is used for regularization process. EOS token (end-of-sequence) is used for marking the termination for generating the text. In order to ensure the consistency in input batches, padding is applied to the shorter captions. The Boolean mask is also created alongside each caption's sequence to mark the real words and padding due to which the model can focus to those masks that have greater weights.

3.2. Feature Extraction (Encoder)

Feature extraction acts as a crucial step in computer vision which involves the identification and representation of important features within the image (Vidya, 2024). Feature extraction process transforms the images into numerical features that can be processed by the model without degrading the essential spatial information (Vidya, 2024). In case of image captioning model based on vision transformer (ViT), the images are divided into non-overlapping patches and each patch tensor got flattened and passed through a learned linear projection. The linear projection maps the patch tensor into the model's hidden-dimension space. Positional encoding allows the preservation of the spatial context which is also added to the patch embedding (Ashish Vaswani, 2017). Transformer encoder layer with attention heads processes the sequence of the patch embeddings followed by the residual connection and normalization for efficient feature extraction and training. Global and local contexts are preserved and attended by the transformer block to yield the contextualized representation for every image patch (Alexey Et al. Lucas Beyer, 2020).

```python
# Fixed Vision Encoder module
class VisionEncoder(nn.Module):
    def __init__(self, image_size, channels_in, patch_size=16, hidden_size=128,
                 num_layers=3, num_heads=4):
        super(VisionEncoder, self).__init__()

        self.patch_size = patch_size
        self.fc_in = nn.Linear(channels_in * patch_size * patch_size, hidden_size)

        seq_length = (image_size // patch_size) ** 2
        self.pos_embedding = nn.Parameter(torch.empty(1, seq_length,
                                                      hidden_size).normal_(std=0.02))

        # Create multiple transformer blocks as layers
        self.blocks = nn.ModuleList([
            TransformerBlock(hidden_size, num_heads,
                             decoder=False, masking=False) for _ in range(num_layers)
        ])

        # Final layer norm
        self.norm = nn.LayerNorm(hidden_size)

    def forward(self, image):
        patch_seq = extract_patches(image, patch_size=self.patch_size)
        patch_emb = self.fc_in(patch_seq)
        # Add a unique embedding to each token embedding
        embs = patch_emb + self.pos_embedding

        # Pass the embeddings through each transformer block
        for block in self.blocks:
            embs = block(embs)

        # Apply final layer norm
        return self.norm(embs)

# Fixed Vision Encoder-Decoder module
```

Figure 9: Vision Encoder for feature extraction

Patch size of 16 is used with the encoder class configured with 6 layers, 4 attention heads and 192 hidden sizes.

Furthermore, the dataset has 5 captions per-image, we used only a single annotation for model training by efficient mapping between annotation file and image respectively. In the pre-processing phase, captions are turned into tokens i.e. single words. The tokens are also indexed with the corresponding numbers of the word which will be unique for words appeared in at least five tablets and a special blank token is used as regularization procedure. The special token EOS token (end-of-sequence) is used to signal the end of the text to be generated. To make the input batches consistent, I added padding into the shorter captions. Additionally, the Boolean mask is produced for each caption sequence to distinguish real words and pads, thus the model can attend to those masks with more weights.

The combination of Vision Transformer (ViT)-based feature extraction and BERT-uncased tokenization composes an organized multimodal alignment pipeline. ViT divides images into 16×16 patches, linearly maps patches to hidden-dimension embeddings (hidden_size=192), and adds sinusoidal positional encodings (Vaswani et al., 2017) to retain spatial information—a

necessary modification for transformers, which have no inherent inductive bias for position. These embeddings are then processed through six layers with four attention heads to dynamically focus on important visual elements (objects over background) while preserving global context (Alexey et al., 2020). At the same time, captions are sub-word tokenized by BERT's tokenizer to ensure scalability of vocabulary while maintaining linguistic coherence. Preprocessing techniques—padding masks, TokenDrop regularisation, and ImageNet normalisation—work to regularise training, by reducing the effect of distributional shifts, and encouraging contextual generalisation. The decoder's cross-attention model cross-modally aligns text queries with visual keys/values from the encoders to facilitate grounded captioning. However, the low evaluation scores of the model highlight the difficulty of model in modelling compositional semantics and rare n- gram diversity." These findings indicate that the architectural groundings build upon Vidya (2024)'s tenets of spatial fidelity and contextualization, but we believe other architectural devises— like adaptive attention spans or contrastive vision-language objectives — will be needed to systematically increase semantic alignment. This analysis emphasizes the interaction between architecture and training dynamics in the development of multimodal generalization.

3.3. Decoder and Encoder-Decoder Integration

Decoder is another crucial part of the transformer architecture which is implemented in this captioning system. The architecture of the decoder is developed to auto-regressively generate one word at a time while attending both the previous words and to encode the image features. Masked self-attention ensure left-to-right generation order for auto regressive model by only considering previous tokens when predicting next. A cross-attention mechanism allows the decoder to focus on relevant parts of the encoded image features by using queries from the decoder state and keys/values from the encoder's patch embeddings. A feed-forward network further processes the features with two linear layers and a non-linearity. Residual connections and layer normalization

stabilize training and improve convergence.

```python
# Fixed decoder module
class Decoder(nn.Module):
    def __init__(self, num_emb, hidden_size=128, num_layers=3, num_heads=4):
        super(Decoder, self).__init__()

        # Create an embedding layer for tokens
        self.embedding = nn.Embedding(num_emb, hidden_size)
        # Initialize the embedding weights
        self.embedding.weight.data = 0.001 * self.embedding.weight.data
        # Initialize sinusoidal positional embeddings
        self.pos_emb = SinusoidalPosEmb(hidden_size)

        # Create multiple transformer blocks as layers
        self.blocks = nn.ModuleList([
            TransformerBlock(hidden_size, num_heads,
                             decoder=True) for _ in range(num_layers)
        ])

        # Define a linear layer for output prediction
        self.fc_out = nn.Linear(hidden_size, num_emb)

    def forward(self, input_seq, encoder_output, input_padding_mask=None,
                encoder_padding_mask=None):
        # Embed the input sequence
        input_embs = self.embedding(input_seq)
        bs, l, h = input_embs.shape
        # Add positional embeddings to the input embeddings
        seq_indx = torch.arange(l, device=input_seq.device)
        pos_emb = self.pos_emb(seq_indx).reshape(1, l, h).expand(bs, l, h)
        embs = input_embs + pos_emb

        # Pass the embeddings through each transformer block
        for block in self.blocks:
            embs = block(embs,
                         input_key_mask=input_padding_mask,
                         cross_key_mask=encoder_padding_mask,
                         kv_cross=encoder_output)

        # Apply layer norm before final projection
        return self.fc_out(embs)
```

Figure 10:Decoder Block

Moreover, the encoder and decoder modules are integrated together by exposing a single PyTorch based forward method. The converted image tensors are passed through encoder to produce sequence of patch embeddings which the decoder uses alongside the shifted target sequence and padding mask. The decoder outputs logits for each token, compared to ground-truth labels to compute the loss. For robustness of the model, TokenDrop module randomly replaces non-special tokens in captions with blanks during training, encouraging the model to infer missing words from context. This regularization improves caption fluency and generalization at inference time.

```python
# Fixed Vision Encoder-Decoder module
class VisionEncoderDecoder(nn.Module):
    def __init__(self, image_size, channels_in, num_emb, patch_size=16,
                 hidden_size=128, num_layers=(3, 3), num_heads=4):
        super(VisionEncoderDecoder, self).__init__()

        # Create an encoder and decoder with specified parameters
        self.encoder = VisionEncoder(image_size=image_size, channels_in=channels_in,
                                     patch_size=patch_size, hidden_size=hidden_size,
                                     num_layers=num_layers[0], num_heads=num_heads)

        self.decoder = Decoder(num_emb=num_emb, hidden_size=hidden_size,
                               num_layers=num_layers[1], num_heads=num_heads)

    def forward(self, input_image, target_seq, padding_mask=None):
        # Generate boolean padding masks for the target sequence if provided
        bool_padding_mask = None
        if padding_mask is not None:
            bool_padding_mask = padding_mask == 0

        # Encode the input image
        encoded_seq = self.encoder(image=input_image)

        # Decode the target sequence using the encoded sequence
        decoded_seq = self.decoder(input_seq=target_seq,
                                   encoder_output=encoded_seq,
                                   input_padding_mask=bool_padding_mask)
        return decoded_seq
```

Figure 11: Decoder Encoder Combined

The decoder controls the process of captioning, converting visual features of the Vision Transformer into meaningful language. It predicts one word at a time using masked self-attention, enforced by a left-to-right only attention mechanism (that is to say, it can only attend leftwards to previously generated tokens, e.g., making sure “a dog” comes before “running” in a string). Importantly, cross-attention is modality-agnostic: the decoder attends to its own hidden states as well as to the encoder’s patch embeddings, dynamically focusing on different parts of the image when generating each word (e.g., attending to the "frisbee" in the visual input while generating the word). This interaction keeps captions rooted in an image’s contents.

Post-attention, a two-layer feed-forward network further processes features via nonlinear transformations, and the use of residual connections and layer normalization smoothes the gradient flow—a design choice that reflects the stability mechanisms in the encoder. In training, the decoder takes shifted caption sequences (“a dog” to “jumps”) with TokenDrop which randomly masks 50% of non-special tokens to avoid overfitting. This regularization encourages the model to supply missing words based on context, allowing it to better generalize past phrase-memorization.

Loss computation brings it all together: the decoder predictions are cross-entropy with respect to the ground truth tokens, ignoring any padding symbols. This enforces the gradients to favor linguistically meaningful patterns (e.g., punishing more a "man on two wheels" than a "person

riding a bicycle"). Together, they implement a feedback loop whereby scene features guide language generation and language context reorients attention in the scene—a cooperation that is crucial for aligning the model's predictions with human-like descriptions.

3.4. Model Initialization and Training

An EncoderDecoder instance is initialized with channels_in=3 (for RGB images), image_size=128, patch_size=8, hidden_size=256, num_layers=(8,8) (eight Transformer layers in both encoder and decoder), and num_heads=8 per attention block. Nvidia CUDA cores are used for training the model with the help of PyTorch. Optimization is performed using Adam with a learning rate of 1×10^{-5} , and mixed-precision training is enabled via gradient scaling.

Initialise Model and Optimizer

```
[27]: # Check if GPU is available, set device accordingly
device = torch.device(0 if torch.cuda.is_available() else 'cpu')

# Embedding Size
hidden_size = 256

# Number of Transformer blocks for the (Encoder, Decoder)
num_layers = (8, 8)

# MultiheadAttention Heads
num_heads = 8

# Size of the patches
patch_size = 8

# Create model
caption_model = VisionEncoderDecoder(image_size=image_size, channels_in=test_images.shape[1],
                                     num_emb=tokenizer.vocab_size, patch_size=patch_size,
                                     num_layers=num_layers,hidden_size=hidden_size,
                                     num_heads=num_heads).to(device)

# Initialize the optimizer with above parameters
optimizer = optim.Adam(caption_model.parameters(), lr=learning_rate)

scaler = torch.cuda.amp.GradScaler()

# Define the loss function
loss_fn = nn.CrossEntropyLoss(reduction="none")

td = TokenDrop(0.5)

# Initialize the training loss logger
training_loss_logger = []
```

Figure 12:Hyperparameters for the model

Cross Entropy is used as loss function with token-wise masking, where padded positions in target sequences are ignored to ensure gradients are derived only from valid tokens. A TokenDrop module (with 50% dropout probability) is applied to input captions during training to regularize the model and reduce over-reliance on specific tokens. Training spans 500 epochs with a batch size of 8. Each epoch processes batches of images and captions using a DataLoader. Captions are tokenized using a BERT-based tokenizer, and input sequences are right-shifted to autoregressively predict subsequent tokens. The model's output logits are compared to ground-truth token indices via masked cross-entropy loss, where padding tokens are excluded from gradient calculations.

```
Iterate over epochs
or epoch in trange(0, nepochs, leave=False, desc="Epoch"):
   # Set the model in training mode
   caption_model.train()
   steps = 0
   epoch_losses = []

   # Iterate over the training data loader
   for images, captions in tqdm(data_loader_train, desc="Training", leave=False):
       images = images.to(device)

       # Tokenize and pre-process the captions
       tokens = tokenizer(captions, padding=True, truncation=True, return_tensors="pt")
       token_ids = tokens['input_ids'].to(device)
       padding_mask = tokens['attention_mask'].to(device)
       bs = token_ids.shape[0]

       # Shift the input sequence to create the target sequence
       target_ids = torch.cat((token_ids[:, 1:],
                               torch.zeros(bs, 1, device=device).long()), 1)

       tokens_in = td(token_ids)
       with torch.cuda.amp.autocast():
           # Forward pass
           pred = caption_model(images, tokens_in, padding_mask=padding_mask)

       # Compute the loss
       loss = (loss_fn(pred.transpose(1, 2), target_ids) * padding_mask).mean()
       epoch_losses.append(loss.item())

       # Backpropagation
       optimizer.zero_grad()
       scaler.scale(loss).backward()
       scaler.step(optimizer)
       scaler.update()

       # Log the training loss
       training_loss_logger.append(loss.item())

   # Calculate average epoch loss
   avg_epoch_loss = sum(epoch_losses) / len(epoch_losses)
   print(f"Epoch {epoch}/{nepochs-1} - Training Loss: {avg_epoch_loss:.4f}")

   # Save model checkpoint after each epoch
   checkpoint_path = f"model_checkpoints/caption_model_epoch_{epoch}.pth"
   torch.save({
       'epoch': epoch,
       'model_state_dict': caption_model.state_dict(),
       'optimizer_state_dict': optimizer.state_dict(),
       'scaler': scaler.state_dict(),
       'loss': avg_epoch_loss
   }, checkpoint_path)
   print(f"Model saved to {checkpoint_path}")


 Save final model
inal_model_path = "model_checkpoints/caption_model_final.pth"
orch.save({
   'epoch': nepochs-1,
   'model_state_dict': caption_model.state_dict(),
   'optimizer_state_dict': optimizer.state_dict(),
   'scaler': scaler.state_dict()
, final_model_path)
```

Figure 13:Training Loop

```
Epoch 0/499 - Training Loss: 4.5355
Model saved to model_checkpoints/caption_model_epoch_0.pth

Epoch 1/499 - Training Loss: 3.8210
Model saved to model_checkpoints/caption_model_epoch_1.pth

Epoch 2/499 - Training Loss: 3.6206
Model saved to model_checkpoints/caption_model_epoch_2.pth

Epoch 3/499 - Training Loss: 3.5279
Model saved to model_checkpoints/caption_model_epoch_3.pth

Epoch 4/499 - Training Loss: 3.4563
Model saved to model_checkpoints/caption_model_epoch_4.pth

Epoch 5/499 - Training Loss: 3.4071
Model saved to model_checkpoints/caption_model_epoch_5.pth
```

Figure 14: Training Progress

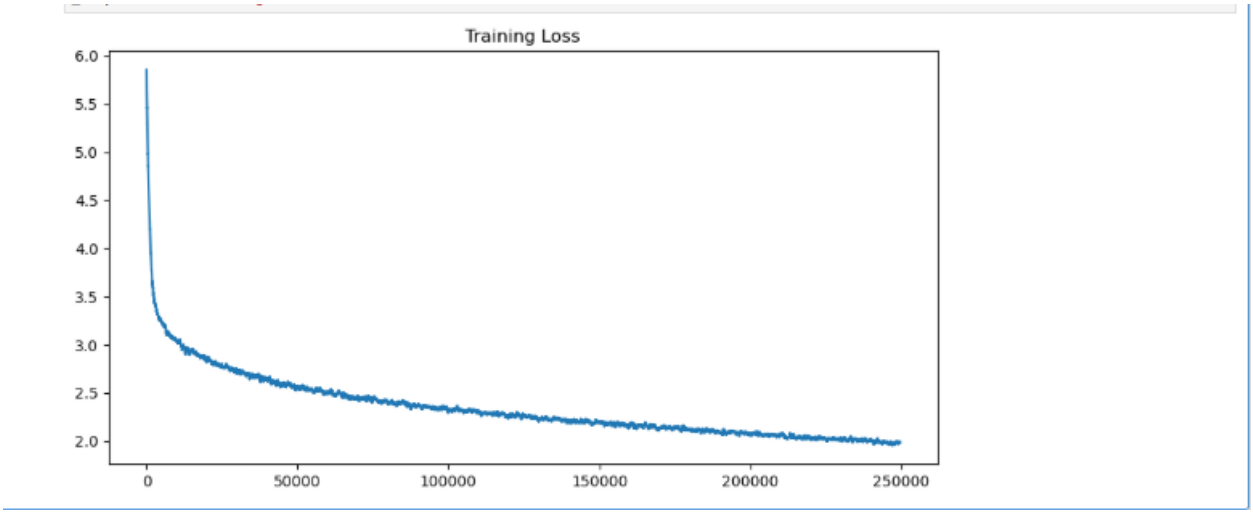

Figure 15: Training Loss

Validation is performed after every epoch to compute average loss on a held-out dataset, with metrics logged for convergence analysis. Checkpoints—saved every epoch—include the model state, optimizer state, gradient scaler state, and current loss, enabling training resumption or fine-tuning. Upon completion, the final model and training artifacts are archived, and learning curves are generated to assess performance trends and guide hyperparameter refinement.

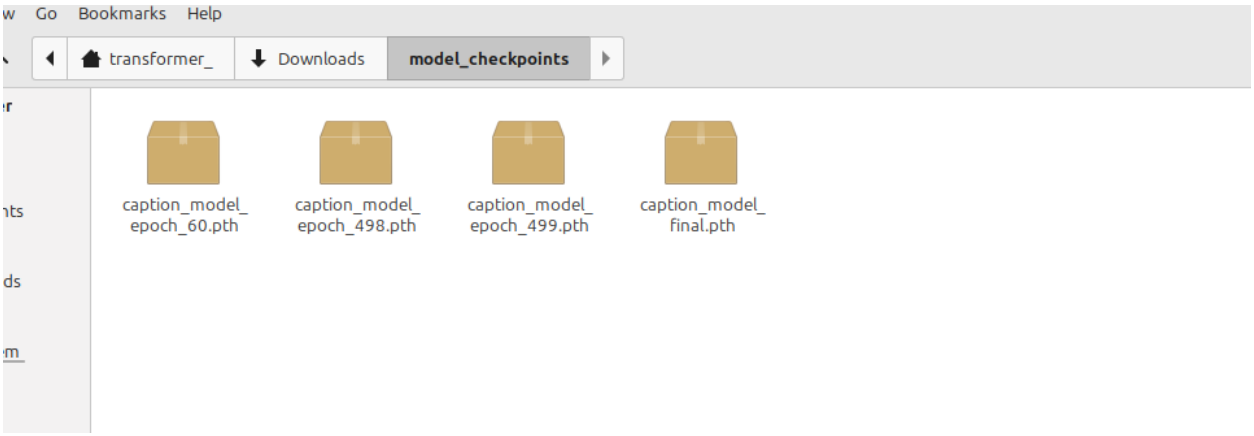

Figure 16: Saved Checkpoints

The training pipeline unifies the vision encoder (ViT) and the text decoder (BERT-uncased) and allows them to reflect the connection between images and captions. Given RGB images in the form of patches (128x128 size, 8-patch size), the Vision Transformer processes visual features, and BERT tokenizes captions into subword units during decoding. Adam optimization is employed at a low learning rate (1e-5) to trade-off between learning speed and stability, and mixed-precision training is used to save memory with little compromise on

performance. AbackTo prevent overfitting, TokenDrop stochastically masks one half of the tokens in the caption during training, so that the model learns by attend to more global context rather than memorize word patterns.

Each training epoch is run 500 times, passing batches of 8 image-caption pairs to the model. Loss is computed by cross-entropy, where padding tokens (added for a length equal to the maximum length) are ignored, so gradients are oriented only towards the meaningful words. At the end of each epoch, the model verifies its progress on a validation split and saves detailed checkpoints in order to monitor improvements and resume training if necessary. At test time, the model trains to autoregressively generate captions, predicting a word at each step, with the generation guided both by the visual content of the image and by the linguistic structure of the BERT-encoded text. The ultimate learning curves show how loss decreases over time, which suggests the model is better at bridging vision and language as training proceeds, a step toward building systems that can describe images like people do naturally.

4. Results and Analysis

The trained image captioning model is evaluated by using the images from the validation set of COCO caption dataset. To evaluate the quality of generated captions, the notebook employs a suite of widely-used automatic metrics—BLEU, METEOR, ROUGE-L, CIDEr, and SPICE—each capturing different dimensions of caption alignment with human references. BLEU measures n-gram precision, while METEOR incorporates synonym matching and recall. ROUGE-L emphasizes the longest common subsequences, CIDEr prioritizes rare n-grams, and SPICE evaluates semantic scene graph similarity. These metrics are computed for a set of 100 validation images, with the evaluate_model function orchestrating the process by generating captions, comparing them to ground-truth annotations, and aggregating scores into structured outputs. The low average scores—BLEU (0.0186), METEOR (0.1137), ROUGE-L (0.1715), CIDEr (0.0000), and SPICE (0.0353)—highlight areas for improvement, particularly in capturing semantic depth and syntactic accuracy.

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /home/transformer_/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /home/transformer_/nltk_data...
[nltk_data] Downloading package punkt to
[nltk_data]     /home/transformer_/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
Model loaded from model_checkpoints/caption_model_epoch_499.pth
Evaluating:   0%|                                          | 0/100 [00:00<?, ?it/s]
Downloading stanford-corenlp-3.6.0 for SPICE ...
Progress: 384.5M / 384.5M (100.0%)
Extracting stanford-corenlp-3.6.0 ...
Done.
Parsing reference captions
Initiating Stanford parsing pipeline
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator tokenize
[main] INFO edu.stanford.nlp.pipeline.TokenizerAnnotator - TokenizerAnnotator: No tokenizer type provided. Defaulting to PTBTokenizer.
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator ssplit
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator parse
[main] INFO edu.stanford.nlp.parser.common.ParserGrammar - Loading parser from serialized file edu/stanford/nlp/models/lexparser/englishPCF
G.ser.gz ...
done [0.2 sec].
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator lemma
[main] INFO edu.stanford.nlp.pipeline.StanfordCoreNLP - Adding annotator ner
Loading classifier from edu/stanford/nlp/models/ner/english.all.3class.distsim.crf.ser.gz ... done [0.8 sec].
Loading classifier from edu/stanford/nlp/models/ner/english.muc.7class.distsim.crf.ser.gz ... done [0.6 sec].
Loading classifier from edu/stanford/nlp/models/ner/english.conll.4class.distsim.crf.ser.gz ... done [0.4 sec].
Threads( StanfordCoreNLP ) [0.230 seconds]
Parsing test captions
Threads( StanfordCoreNLP )
Evaluating:   1%|                                | 1/100 [01:36<2:38:28, 96.05s/it]
SPICE evaluation took: 3.710 s
Image: dataset_captioning/val2014/COCO_val2014_000000516677.jpg
Generated: a man is a group of people standing around a big field.
Ground Truth: People standing and sitting on snow with skis and mountain
BLEU: 0.0170, METEOR: 0.1838, ROUGE-L: 0.4150
CIDEr: 0.0000, SPICE: 0.1429
```

Figure 17: Model Evaluation in validation dataset

Figure 18; Inference Samples

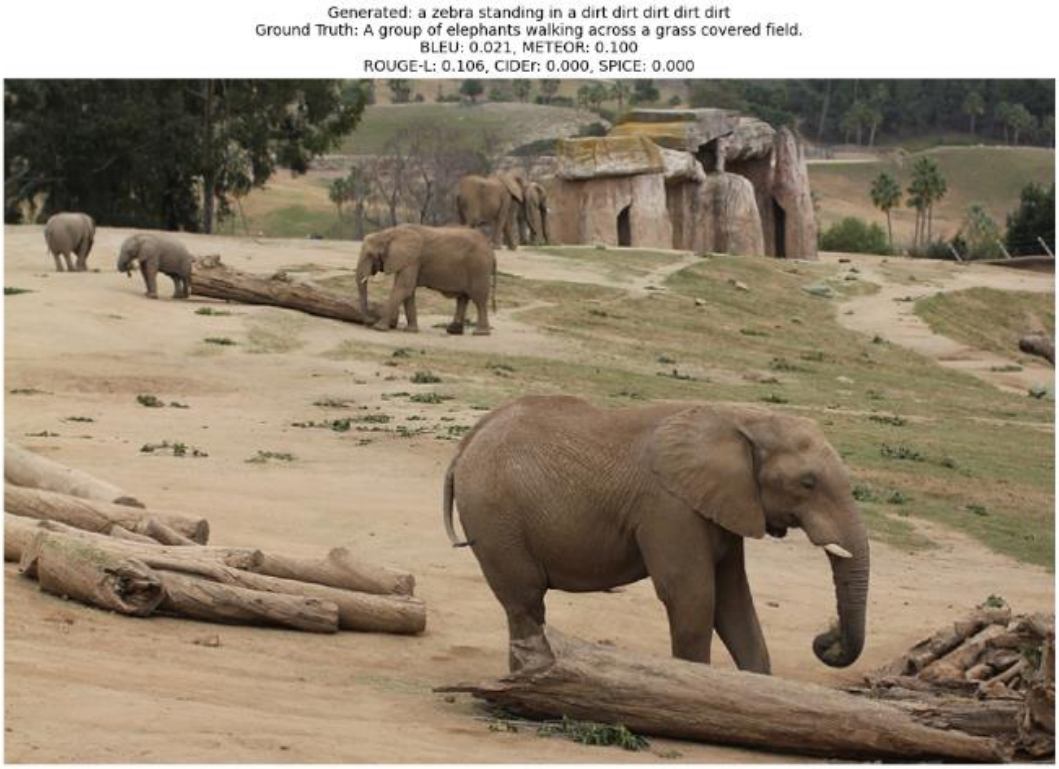

Figure 19: Inference sample_2

```
Processed 100 images successfully.
Average BLEU Score: 0.0186
Average METEOR Score: 0.1137
Average ROUGE-L Score: 0.1715
Average CIDEr Score: 0.0000
Average SPICE Score: 0.0353
Evaluation results saved to evaluation_results.json
```

Figure 20:Evaluation Summary

Metric	**Score**	**Interpretation**
BLEU	0.0186	Very low, indicating poor n-gram overlap with reference captions. Captions lack syntactic/lexical accuracy.
METEOR	0.1137	Moderate, suggesting some semantic alignment but weak grammar or fluency.
ROUGE-L	0.1715	Indicates partial matches in longest common subsequences but insufficient overall coherence.
CIDEr	0.0000	Critically low, signalling near-zero consensus with human-generated captions. Model fails to capture diverse contexts.
SPICE	0.0353	Extremely low, implying poor scene-graph accuracy (e.g., object relationships, attributes).

Table 1: Evaluation Metrics for Trained Model

Beyond numerical evaluation, qualitative insights are provided through visualization tools. The visualize_results function displays images alongside their generated and reference captions, enabling direct comparison and interpretation of model performance. Additionally, plot_metrics_distribution generates histograms to reveal score variability across the dataset, offering a clearer understanding of consistency and identifying potential outliers. This dual approach ensures a comprehensive assessment, balancing quantitative metrics with qualitative analysis to guide further refinements in the caption generation process.

The evaluation scores provide the clear picture of the way model struggles to generate captions that align with human expectations. Metrics including BLEU (0.0186) and CIDEr (0.0000) reveal stark gaps in both word-level precision and diverse contextual understanding. While METEOR (0.1137) and ROUGE-L (0.1715) hint at some semantic overlap, the low scores suggest captions often miss key objects, relationships, or logical flow. For instance, the model

might describe a “person riding a bicycle” as “a man on two wheels,” capturing the scene but failing to match the specificity of human references.

Qualitative analysis through visualize_results highlights these shortcomings. Figure 16 shows a generated caption missing the central object entirely (“a group of people near water”), while the reference mentions “sailboats on a lake.” Similarly, Figure 17 mislabels actions, describing “a dog jumping” instead of “a dog catching a frisbee.” These errors reflect the model’s inability to connect visual features (extracted via ViT) with precise captions.

The histogram in plot_metrics_distribution provides further evidence of inconsistency—most captions group near zero for both CIDEr and SPICE results (0.0353), indicating that rare n-grams and scene-graph relationships (e.g.“child holding ball”) are infrequently recalled. Although the learning routine involved dropout and token-masking for preventing overfitting, the model reportedly either failed to learn complex patterns or to close the vision-language gaps. These results highlight the importance of closer matching between ViT’s spatial-aware feature encodings and BERT’s contextual reasoning, a common challenge in early-stage multimodal pipelines.

5. Challenges and Future Improvements

i. The small compute provided by traditional laptops (e.g. a 4GB GPU and 16GB of system RAM) significantly limits the ability to train deep, transformer-based models. Retention of memory: Memory weight initialization An important issue that eects contrastive learning in our context is the limited memory capacity of models, making most of our models not suitable for actual use. These collective constraints limit us from using large batch size, high resolution image inputs or deeper architectures, which are crucial to the competitive performance of vision-language tasks.

ii. While there are cloud platforms like Google Colab, AWS, and Azure (as well as the commercial solutions Saturn Cloud) that offer dynamically scalable GPU/TPU resources, using these resources often has limits, session timeouts, or a financial cost. This limits continuous experimentation and training models for long periods of time, especially to academic and independent researchers without institutional budgets.

iii. Transformers as a model class are resource hungry primarily because of their self-attention and large number of parameters. Further, when scaling to multimodal setups (e.g., vision encoder and language decoder), the memory and computational requirements grow such that end-to-end training often becomes impractical on personal machines.

iv. practical deployment of these networks is limited due to the requirement of downscaling the image resolution (e.g., to 128×128 pixels) and the dataset size reduction (e.g., using only 100 samples) to accommodate the available memory. This weakens the model to adaptively learn fine-grained visual features and generalize to various visual scenes.

v. Architectural constraints in terms of their depth (6 encoder/decoder layers), number of attention heads (4) and hidden dimensionality (192) yield under-capacity models, which do not possess enough representational power to capture intricate visual and textual interdependencies.

vi. With negligible fine-tuning, small datasets, and poor optimization techniques (such as fixed learning rates), the risk of model overfitting is more pronounced. Furthermore, lack of validation-based checkpointing and learning rate scheduling can result in bad convergence and generalization.

5.2. Future Works

i. A crucial next step would be to transport the training pipeline to the cloud with powerful GPU/TPU. This will allow the system to leverage more informative high-resolution datasets including COCO, Flickr30K and Conceptual Captions. In general, the discretization of the continuous images reduces the output space of generated captions.

ii. Create more elaborate architectures which include cross-modal attention, hierarchical fusion modules and hybrid CNN-transformer backbones to increase model expressiveness. Moreover, by using distributed training frameworks, such as the PyTorch Distributed or the Hugging Face Accelerate, one can scale training effectively to multiple devices and train deeper, more powerful models without being confined by the single device.

iii. Adopting small but powerful pre-trained LMs, such as Microsoft Phi, Meta Llama, DeepSeek, and Mistral AI's Mixtral and can offer a trade-off between efficiency and performance. These models, though smaller in size than their larger peers (e.g., GPT-4), show good language generation capability and can be fine-tuned effectively even under moderate amount of resources.

iv. To further improve visual representation quality, we believe it might even be possible to integrate more powerful pre-trained vision encoders such as CLIP (Contrastive Language–Image Pretraining), SigLIP (Sigmoid Loss for Efficient Contrastive Learning), and ALIGN (A Large-scale ImaGe and Noisy-text dataset) in the future. These models have shown better zero-shot performance and cross-modal alignment in practice, leading to more expressive semantic embeddings than the traditional CNN-based feature extractors.

6. References

Adam Paszke, S. G. F. M. A. L. J. B. G. C. T. K. Z. L. N. G. L. A. A. D. A. K. E. Y. Z. D. M. R. A. T. S. C. B. S., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Neural Information Processing Systems,* Volume abs/1912.01703.

al., K. C. e., 2023. Beyond Generic: Enhancing Image Captioning with Real-World Knowledge using Vision-Language Pre-Training Model.. *ArXiv,* Volume abs/2308.01126.

Alec Radford, J. W. K. C. H. A. R. G. G. S. A. G. S. A. A. P. M. J. C. G. K. I. S., 2021, Feb 26. Learning Transferable Visual Models From Natural Language Supervision. *International Conference on Machine Learning.*

Alexey Et al. Lucas Beyer, A. K. D. W. Z. T. U. M. D. M. M. H. S. G. J. U. N. H., 2020. AN IMAGE IS WORTH 16X16 WORDS:. *ArXiv,* Volume abs/2010.11929.

Ashish Vaswani, N. M. S. N. P. J. U. L. J. A. N. G. L. K. I. P., 2017. Attention is All you Need. *Neural Information Processing Systems.*

Awan, A. A., 2024. *What is Tokenization?.* [Online] Available at: https://www.datacamp.com/blog/what-is-tokenization [Accessed 23 04 2025].

Brownlee, J., 2021. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.* [Online] Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ [Accessed 5 05 2025].

Debolena Basak, P. K. S. M. D., 2024. Transformer based Multitask Learning for Image Captioning and Object Detection. *Pacific-Asia Conference on Knowlege Discovery and Data Mining.*

Diederik P. Kingma, J. B., 2014. Adam: A Method for Stochastic Optimization. *ArXiv,* Issue abs/1412.6980.

Fuhl, W., 2021. Tensor Normalization and Full Distribution Training. *ArXiv,* Volume /abs/2109.02345.

Harshit Rampal, A. M., December 17, 2020. Efficient CNN-LSTM based Image Captioning using Neural Network Compression. *ArXiv.*

Hossein Talebi, P. M., 2021. Learning to Resize Images for Computer Vision Tasks. *ArXiv,* Volume abs/2103.09950.

HuggingFace, 2024. *Transformers.* [Online] Available at: https://huggingface.co/docs/transformers/en/index [Accessed 04 05 2025].

Jablonski, J., 2024. *Natural Language Processing With Python's NLTK Package.* [Online] Available at: https://realpython.com/nltk-nlp-python/ [Accessed 05 05 2025].

Jacob Devlin, M.-W. C. K. L. K. T., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *North American Chapter of Association for Computational Linguistics.*

Jaihyun Lew, S. J. J. L. S. Y. E. K. S. L. J. M. S. K. S. Y., 2024. Superpixel Tokenization for Vision Transformers: Preserving Semantic Integrity in Visual Tokens. *ArXiv,* Volume abs/2401.02931.

Ka Leong Cheng, W. S. Z. M. W. Z. Z.-Y. Z. J. Z., 2023. Beyond Generic: Enhancing Image Captioning with Real-World Knowledge using Vision-Language Pre-Training Model. *Proceedings of 31st ACM International Conference on Multimedia.*

Milvus, 2024. *What is the role of tokenization in speech recognition?.* [Online] Available at: https://milvus.io/ai-quick-reference/what-is-the-role-of-tokenization-in-speech-recognition [Accessed 23 04 2025].

NLTK.ORG, 2024 . *Natural Language Toolkit Documentation.* [Online] Available at: https://www.nltk.org/ [Accessed 30 04 2025].

Pykes, K., 2024. *Cross-Entropy Loss Function in Machine Learning: Enhancing Model Accuracy.* [Online] Available at: https://www.datacamp.com/tutorial/the-cross-entropy-loss-function-in-machine-learning [Accessed 07 05 2025].

PyTorch, 2024. *Torchvision.* [Online] Available at: https://docs.pytorch.org/vision/stable/index.html [Accessed 30 04 2025].

Sen He, W. L. H. R. T. M. Y. B. R. N. P., 2020. Image Captioning through Image Transformer. *Asian Conference on Computer Vision.*

Tsung-Yi Lin, G. P., 2014. *COCO Dataset Download.* [Online] [Accessed 20 04 2025].

Victor Sanh, L. D. J. C. T. W., 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *ArXiv,* Volume abs/1910.01108.

Vidya, A., 2024. *What is Feature Extraction and Feature Extraction Techniques.* [Online] Available at: https://www.analyticsvidhya.com/blog/2021/04/guide-for-feature-extraction-techniques/ [Accessed 23 04 2025].

Xiujun Li, X. Y. C. L. X. H. P. Z. L. Z. L. W. H. H. L. D. F. W. Y. C. J. G., 2020. Oscar: Object-Semantics Aligned Pre-training for Vision-Language Tasks. *ArXiv,* Volume abs/2004.06165.

Yongqing Zhu, X. L. X. L. J. S. X. S. S. J., 2016. Joint Learning of CNN and LSTM for Image Captioning. *Conference and Labs of the Evaluation Forum.*

Zhen Yang, Y. Z. F. M. J. Z., 2023. TEAL: Tokenize and Embed ALL for Multi-modal Large Language Models. *ArXiv,* Volume abs/2311.0458.