Vietnam National University, Ho Chi Minh City
University of Information Technology
Computer Science

FINAL PROJECT REPORT
Advanced Computer Vision - CS331

VIETNAMESE IMAGE CAPTIONING

Lecturer: PhD.Mai Tiến Dũng

Group members:

Full name	ID	Task
Nguyễn Hà Anh Vũ	21520531	100%

12/2024, Hồ Chí Minh city

Mục lục

CHAPTER 1: Image Captioning PROBLEM

1.1 Definition of Image Captioning

IMAGE CAPTIONING is the task of describing the visual content of the image in natural language, employing a visual understanding system and a language model capable of generating meaningful and syntactically correct sentences.

"Người đàn ông mặc áo đen đang chơi guitar."

"công nhân xây dựng đang làm việc trên đường."

"Hai cô gái đang chơi xếp hình."

Figure 1: Example of Vietnamese Image Captioning

1.2 Problem identification

1.2.1 Input

$$D = \{I_k, C_k\}_{k=1}^{N} \text{ and } I\_new \in F$$

$$\text{with } \begin{cases} I_k \in F = \mathbb{R}^d, \\ C_k = \{c_{k1}, c_{k2}, \ldots, c_{km}\} \end{cases}$$

In which:

+ D: The dataset consisting of images and their corresponding captions.
+ I_k: The k-th image in the dataset.
+ C_k: The set of captions describing the image I_k.
+ N: The number of images in the dataset.
+ I_{new}: A new image that needs to be described.
+ F: The space containing the images ($\mathbb{R}^d$).

Dataset:

- A collection of images, each accompanied by one or more detailed descriptions of its content.
- Each image may contain one or more objects, actions, contexts, or other elements that the model needs to recognize and understand.

New image:

- An input image without an accompanying description.

1.2.2 Output

$$\widehat{c} = f\left(I_{new}\right) \text{ with } f \text{ is the model Image Captioning}$$

- A description in natural language that reflects the visual content of the image.
- The description should include information about:
  - Important objects in the image.
  - Actions taking place.
  - Context or other relevant details.
- Ensure that the sentence is clear, coherent, grammatically correct, and conveys the intended meaning accurately.

1.2.3 Constraints

- **Regarding images:**
  - Input images may contain overlapping objects, complex contexts, or uneven lighting.
- **Regarding descriptions:**
  - Must be accurate in grammar and meaning.
  - The length of the description should not be too short to lack meaning, nor too long to cause confusion.
  - Must not contain irrelevant information or information not present in the image.
- **Regarding processing time:**
  - The system needs to generate descriptions in a reasonable time to ensure practical usability.

1.2.4 Requirements:

Visual Understanding:

- Accurately identify objects present in the image.
- Understand the relationships between objects (e.g., relative positions, interactive actions).

Language Ability:

- Generate natural descriptions similar to how humans communicate.
- Ensure that the sentences are grammatically correct, clearly structured, and complete in content.

Generalization Capability:

- The system must perform well on images with various themes.
- Must not rely on a specific type of image content (e.g., describing only animals or only landscapes).

Effective Integration:

- Easily integrate into practical applications such as tools for assisting visually impaired individuals, image searching, or automated content generation.

1.3 Challenges of Image Captioning

1.3.1 Understanding Image Meaning

Understanding the meaning of an image is a crucial foundation in the task of image captioning, enabling the system to generate appropriate and meaningful descriptions.

- **Understanding objects in the image:** First, the model needs to identify the main objects in the image. For example, recognizing that there is a “car” or “a dog” present. This is a basic step but not sufficient for creating a complete caption.
- **Understanding relationships between objects:** The captioning task is not merely about listing objects; it requires describing how they interact. For instance, "A dog is sitting in the car" is entirely different from "A car is parked near the dog." To achieve this, the model needs to extract deeper semantic features from the image and understand the relationships between different regions of the image.

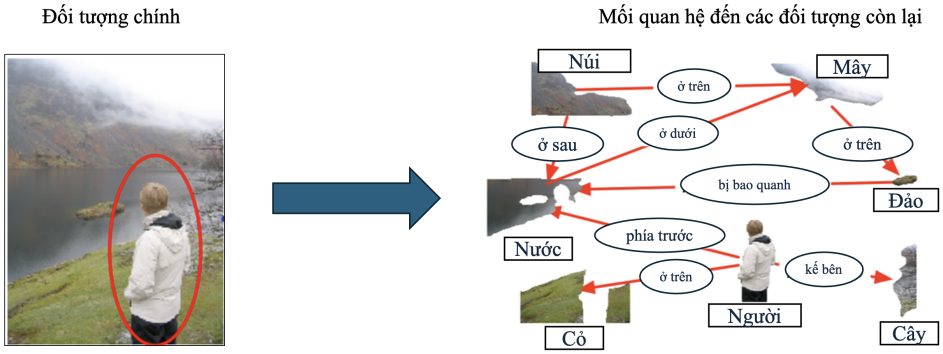
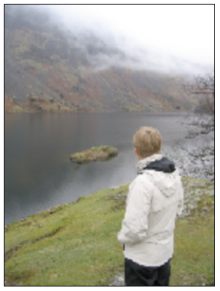


Figure 2: Semantic Understanding

1.3.2 Language Diversity

Diversifying descriptions is a significant challenge in image captioning:

- **Multiple expressions for the same image:** For an image depicting "a cat on the sofa," the model needs to generate different descriptions such as: "A cat is lying on the couch.", "There is a cat relaxing on the sofa.", "The cat is peacefully sleeping on the couch." ,...
- **Creating appropriate context:** An image can be described differently depending on the context or purpose. For instance, in a commercial context, the description might focus on product details, such as "A luxurious sofa with an adorable cat lounging on it."



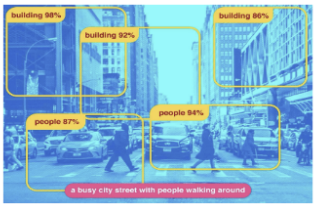
Caption 1: Người đàn ông mặc áo khoác đang nhìn xuống hồ

Caption 2: Một người đang mặc áo trắng, tóc vàng đứng trước mặt hồ

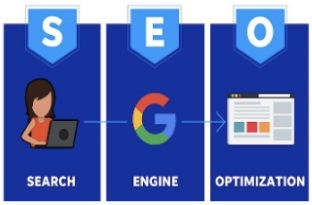
Figure 3: Alternative captions.

1.4 Applications of Image Captioning

Supporting People with Disabilities



Systems Supporting Image Description Generation



Integration into Search Tools and Image Querying Based on Descriptions

Figure 4: Applications of Image Captioning

1. **Supporting People with Disabilities:** Image captioning can play an important role in helping visually impaired individuals access visual information:
   - **Converting image content to text or audio:** Automated descriptions of images can be read aloud through screen readers, helping visually impaired individuals understand content they may struggle to see, allowing them to recognize and imagine situations.
   - **Enhancing communication experiences:** Applications in social media platforms like Facebook and Twitter allow visually impaired users to access shared visual content through automated descriptions.
   - **Supporting learning:** In education, visual materials or charts can be converted into descriptive text, helping visually impaired students grasp the content of lessons.
2. **Systems Supporting Image Description Generation:** Image captioning systems can help simplify specialized tasks:
   - **Automating image editing and captioning tasks:**
     - In creative industries such as journalism, advertising, and design, image captioning helps generate descriptions quickly, saving editing time. For example, when uploading a photo set, the system can automatically create captions for each image for publishing or sharing.
     - In e-commerce, automatically generating descriptions for images can be used to describe new products.
   - **Supporting specialized content creation:**
     - In healthcare, image captioning can describe X-ray or MRI images, assisting doctors in making diagnoses more easily with automated annotations.
     - In legal contexts, these systems can help describe images or videos related to evidence.
3. **Integration into Search Tools and Image Querying Based on Descriptions:** Image captioning enhances the ability to query images and videos through natural descriptions:
   - **Improving text-based search:** Image search tools like Google Images can utilize captioning to accurately and automatically tag images. This improves accuracy when users enter complex queries like "a red sports car parked near the beach."
   - **Image Search:** When users upload an image, the system can generate automatic captions to search for similar content on the internet.
   - **Applications in data management:** CLarge image or video libraries can automatically organize and tag content through descriptions, making searching and categorization more efficient.

CHAPTER 2: Related Works

2.1 Pipeline

The process of creating captions for images (Image Captioning) is carried out through a pipeline consisting of several sequential steps, from feature extraction to generating a detailed descriptive sentence as follows:

The image captioning process begins with the input image that the model needs to describe. This image is passed through a Visual Encoder to extract its features. The Visual Encoder, typically deep learning models, is responsible for converting the image into feature vectors that reflect the content of the image, including objects, scenery, and their relationships. These features can be a single vector or a set of vectors representing different parts of the image.

Next, the features from the Visual Encoder are fed into a Language Model to generate a sequence of descriptive words. This model uses the image features along with previously generated words (or a starting token like <start>) to predict the next word in the caption. This process occurs sequentially, with each new word generated based on the context (previous words) and image features, until the model generates an end token (usually <end>).

Finally, the Language Model outputs a caption in the form of a complete sentence. This caption describes the content of the input image, including objects, actions, and their relationships.

The entire process enables the model to automatically generate accurate and coherent semantic descriptions for any image, based on the features learned from training data.

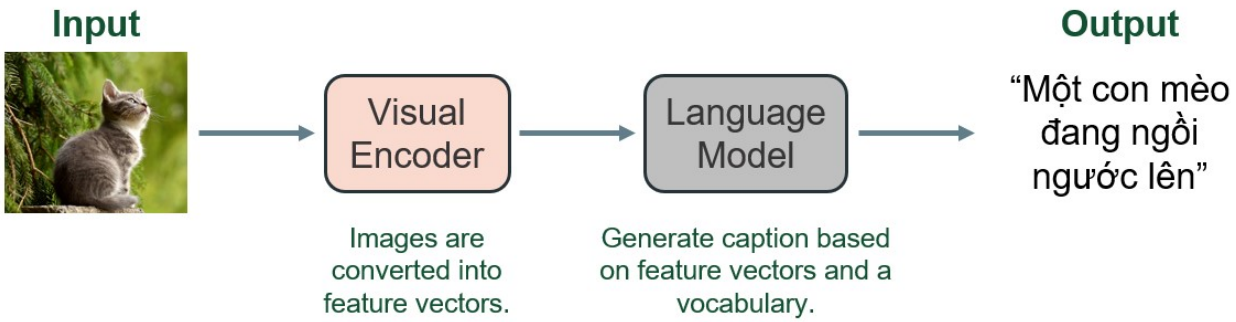


Figure 5: Pipeline.

2.2 Some Approaches to Address the Problem of Image Captioning

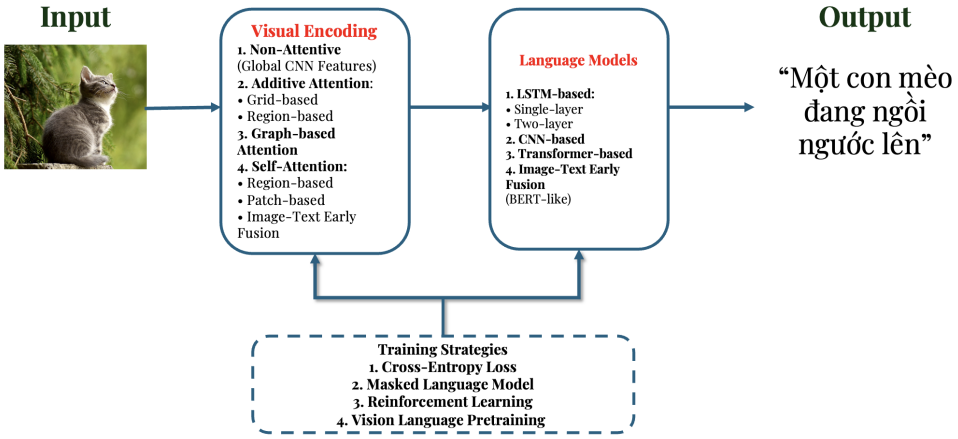


Figure 6: Overview of the image captioning task and taxonomy of the most relevant approaches.

2.2.1 Visual Encoding

Providing an effective representation of the visual content is the first challenge of an image captioning pipeline. The current approaches for visual encoding can be classified as belonging to four main categories:

1. **Non-attentive** methods based on global CNN features.
2. **Additive attentive** methods that embed the visual content using either grids or regions.
3. **Graph-based** methods adding visual relationships between visual regions.
4. **Self-attentive** methods that employ Transformer-based paradigms, either by using region-based, patch-based, or image-text early fusion solutions.

2.2.1.1 Non-attentive/ Global CNN features:

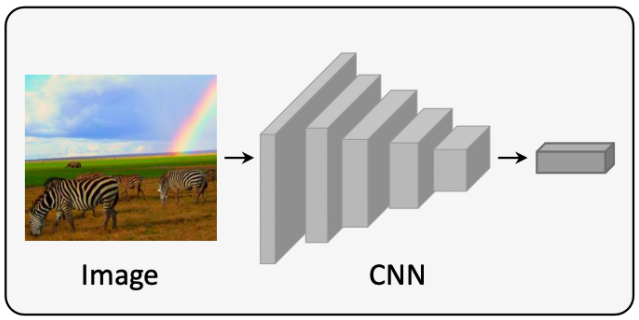


Figure 7: Non-attentive approach

CNNs have significantly improved the performance of visual input models, including image captioning. The visual encoding step typically uses high-level features from the final layers of CNNs as input to a language model. For instance:

- **"Show and Tell"** [1]: Used GoogleNet features as initial hidden state of language model.
- **Karpathy et al.** [2]: Used global features from AlexNet.
- **Mao et al.** [3], **Donahue et al.** [4]: Injected VGG features at language model time step.
- **Rennie et al.** [5]: FC model using ResNet-101 features while preserving dimensions.

Global CNN features are widely adopted due to their simplicity and ability to condense image information into a compact representation. However, this approach often compresses information excessively, limiting the granularity needed for detailed and specific image descriptions.

2.2.1.2 Additive Attention:

a) Attention Over Grid of CNN Features:

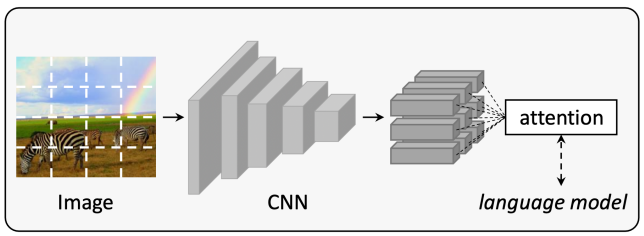


Figure 8: fine-grained features extracted from the activation of a convolutional layer, together with an attention mechanism guided by the language model;

Motivated by the limitations of global representations, recent approaches have enhanced the granularity of visual encoding. For example, Dai et al. [6] used 2D activation maps instead of 1D global feature vectors to incorporate spatial structure into language models. Drawing from machine translation, many image captioning models adopted the additive attention mechanism, allowing time-varying, fine-grained visual feature encoding. A single-layer feed-forward neural network with a hyperbolic tangent non-linearity is used to compute attention weights. Formally, given two generic sets of vectors $x_1,...,x_n$ and $h_1,...,h_m$, the additive attention score between h_i and x_j is computed as follows:

$$f_{\text{att}}(h_i, x_j) = W_3^\top \tanh(W_1 h_i + W_2 x_j)$$

where W_1, W_2 are weight matrices, and W_3 is a weight vector. A softmax function then provides a probability distribution, denoting the relevance of each element x_j to h_i. Originally designed for sequence alignment, this mechanism now connects visual representations to language model hidden states.

b) Attention Over Visual Regions:

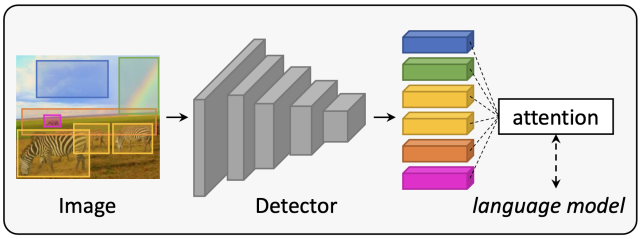


Figure 9: image region features coming from a detector, together with an attention mechanism

Neuroscience suggests that the brain combines a top-down reasoning process (predicting sensory input based on knowledge) with a bottom-up flow (visual stimuli refining predictions). Additive attention acts as a top-down mechanism, where the language model predicts the next word by attending to a feature grid, regardless of image content.

Bottom-Up and Top-Down Attention: Unlike saliency-based methods [7], Anderson et al. [8] define the bottom-up path using an object detector (Faster RCNN) to propose image regions. A top-down mechanism then assigns weights to these regions for word predictions. Pretraining on the Visual Genome dataset, the model learns to predict object and attribute classes, improving detection and feature representation by capturing salient objects and contextual regions.

2.2.1.3 Graph-based Encoding

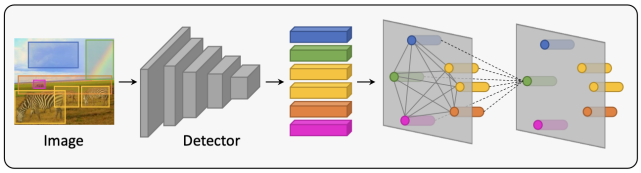


Figure 10: Graph-based encoding of visual regions

To enhance the encoding of image regions and their relationships, some studies employ graph structures that incorporate semantic and spatial connections.

Yao et al. [9] and Guo et al. [10] used graph convolutional networks (GCNs) to integrate semantic and spatial relationships. **Semantic Graphs** was built by using a classifier pre-trained on Visual Genome to predict actions or interactions between object pairs. **Spatial Graphs** was derived from geometric measures like intersection over union (IoU), relative distance, and angles between object bounding boxes. **Scene Graphs**: Yang et al. [11] incorporated semantic priors into image encoding by leveraging scene graphs—directed graphs connecting objects, attributes, and relations. Shi et al. [12] refined this by training predicate prediction directly

on ground-truth captions instead of external datasets. Yao et al. [13] proposed using hierarchical trees as a specialized graph representation.

- Structure: The root represents the entire image, intermediate nodes represent regions and sub-regions, and leaves represent segmented objects.
- Advantages: Graph-based encodings facilitate local information exchange between nodes and integration of external semantic data.

Limitations and Self-Attention solves problem: Manually building graph structures can restrict interactions between visual features. Self-attention overcomes this by connecting all elements as a complete graph, enabling richer and more flexible feature representations.

2.2.1.4 Self-Attention Encoding

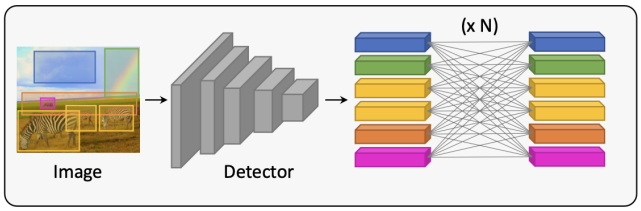


Figure 11: Self-attention-based encoding over image region features

Self-attention connects all elements in a set, refining their representation through residual connections. Introduced by Vaswani et al. [14] for machine translation, it forms the basis of the Transformer architecture, which has dominated NLP and later Computer Vision. The scaled dot-product mechanism is used:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Here, Q, K, V are query, key, and value vectors, respectively, and d_k is a scaling factor.

Early Applications:

- Yang et al. [15]: Introduced self-attention to model relationships between object features.
- Li et al. [16]: Proposed a Transformer with visual and semantic encoders, fused via a gating mechanism.

Variants of Self-Attention:

- Geometry-Aware Encoding: Herdade et al. [17]: Incorporated spatial relationships into attention weights. Guo et al. [18]: Normalized geometry-aware self-attention using relative spatial relationships. He et al. [19]: Developed a spatial graph transformer with categorized spatial relationships.
- Attention on Attention: Huang et al. [20]: Weighted self-attention outputs with context-guided gates.
- X-Linear Attention: Pan et al. [21]: Enhanced region-level features using bilinear pooling for higher-order interactions.
- Memory-Augmented Attention: Cornia et al. [22]: Augmented self-attention with learnable memory vectors for multi-level relationships.

Other Advances:

- Global Context: Ji et al. [23] added a global vector to feature sequences for inter-layer representation.
- Hybrid Approaches: Luo et al. [24] combined region and grid features via self- and cross-attention.
- Semantic Alignment: Liu et al. [25] aligned grid or detection features with visual words for semantic encodings.

Grid Features and Patches:

- Zhang et al. [26]: Applied self-attention directly to grid features with geometric relationships.
- Liu et al. [27]: Introduced the first convolution-free image captioning model using Vision Transformer (ViT).

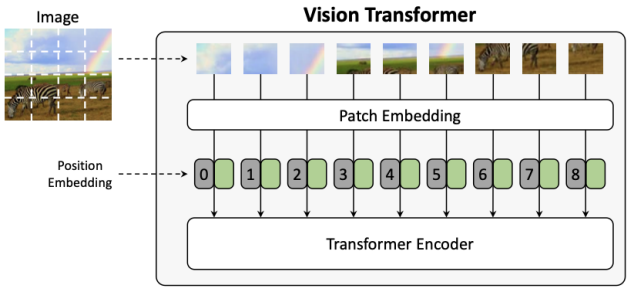


Figure 12: Vision Transformer encoding. The image is split into fixed-size patches, linearly embedded, added to position embeddings, and fed to a standard Transformer encoder.

Vision-and-Language Pretraining:

- Early Fusion: Zhou et al. [28]: Unified region and word tokens in a Transformer for joint image-caption pertaining.
- BERT-Inspired Architectures: OSCAR (Li et al. [29]): Added object tags as semantic anchors, pre-trained on 6.5M image-text pairs. VinVL (Zhang et al. [30]): Improved visual features and pretraining objectives. Scalable Models (Hu et al. [31]): Enhanced VinVL with larger data and model size.

2.2.1.5 Discussion on Visual Encoding

Region-based features have been dominant in image captioning for years. Recently, advancements like better-trained grid features, self-attentive encoders, multi-modal models like CLIP, improved object detectors, and end-to-end visual models are challenging this dominance. BERT-like early-fusion strategies further highlight the potential of integrating visual and textual representations.

2.2.2 Language models

The language model in image captioning predicts the probability of a word sequence $P(y_1, y_2, ...y_n|X)$ where X represents the visual encoding. It operates auto-regressively, conditioning each word on the previous ones and deciding when to stop using an end-of-sequence token.

$$P(y_1, y_2, ...y_n|X) = \prod_{i=1}^{n} P(y_i|y_1, y_2, ...y_{i-1}, X)$$

Key strategies in language modeling for image captioning:

1. LSTM-based: Single-layer or two-layer architectures.
2. CNN-based: Early attempts to move beyond recurrent models.
3. Transformer-based: Fully attentive mechanisms.
4. BERT-like (early fusion): Directly integrates visual and textual inputs.

2.2.2.1 LSTM-based Models

As language has a sequential structure, RNNs are naturally suited to deal with the generation of sentences. Among RNN variants, LSTM has been the predominant option for language modeling.

a) Single-Layer LSTM:

Baseline: Vinyals et al. [1] introduced a simple model where the visual encoding initializes the LSTM hidden state, generating captions word by word through softmax activation. Xu et al. later incorporated additive attention to guide the model over visual features.

Extensions:

- Visual Sentinel: Adds a learnable vector for non-visual words (e.g., "the," "of").
- Hidden State Reconstruction: Improves context modeling with bidirectional LSTMs or auxiliary modules.
- Multi-Stage Generation: Splits captioning into coarse skeleton creation and detailed refinement.
- Semantic-Guided LSTM: Incorporates semantic image features directly into the LSTM gates.

b) Two-Layer LSTM: Architecture: Stacks two LSTMs, with the first layer focusing on attention and the second generating captions. Improvements:

- Neural Baby Talk: Grounds words in image regions using a pointing mechanism.
- Reflective Attention: Refines syntax and word relevance based on past predictions.
- Look Back and Predict Forward: Alleviates accumulated errors by predicting multiple words at once.
- Adaptive Attention Time: Dynamically adjusts attention steps for each word.

c) Boosting LSTMs with Self-Attention: Key Advances: Replaces additive attention with self-attention to improve contextual understanding.

- Attention on Attention: Enhances visual self-attention with an additional attention layer.
- X-Linear Attention: Incorporates second-order interactions for better feature encoding.
- Neural Architecture Search: Optimizes RNN structure and self-attention integration.

2.2.2.2 Convolutional Language Models

Approach: Combines global image features and word embeddings, processed by a CNN.

Mechanism: Operates on all words in parallel during training; uses right-masked convolutions to prevent future word token information leakage.

Drawbacks: Limited adoption due to subpar performance compared to other models and the rise of Transformer architectures.

2.2.2.3 Transformer-based Architectures

Core Concept: Based on the Transformer architecture by Vaswani et al. [14], featuring a masked self-attention for words, cross-attention (words as queries, encoder outputs as keys/values), and a feed-forward network. Unidirectional word generation is enforced via a masking mechanism.

Key Variants and Improvements:

- Gating Mechanisms: Cross-Attention Gating: Modulates flow of visual and semantic info (Li et al. [16]). Context Gating: Adjusts influence of global image representation via multi-head attention (Ji et al. [23]).
- Meshed Decoder (Cornia et al. [22]): Considers all encoding layers instead of only the last one. Includes a mesh operator for independent modulation and a gate for text-query-guided weighting.
- Textual Prefixes: Uses pre-trained visual-semantic models to generate visual tags and prefixes for the decoder ([32]), [33])).

2.2.2.4 BERT-like Architectures

Overview: BERT-like models integrate visual and textual modalities early in the architecture, leveraging pre-trained parameters from large textual corpora. This approach enables strong pre-training advantages for both vision and language tasks.
Key Features:

- **Early Fusion**: Combines visual and textual inputs in a shared multi-layer Transformer encoder.
- **Pre-training**: Models are pre-trained on large-scale image-caption pairs, then fine-tuned for unidirectional caption generation using right-masked token sequences.

Notable Approaches:

- Zhou et al. [28]: Developed a unified BERT-like encoder-decoder model for image captioning.
- Li et al. [29]: Enhanced vision-language alignment by using object tags detected in images as anchor points. Inputs represented as word tokens, object tags, and region features, where object tags serve as textual labels from object detectors.

2.2.2.5 Non-autoregressive Language Models

Concept: Non-autoregressive models leverage parallelism in Transformers to reduce inference time by generating all words simultaneously.
Approaches:

- Multi-Stage Refinement: Initial methods generated words in parallel, refining outputs over multiple stages.
- Reinforcement Learning: Generation viewed as a cooperative multi-agent system, where positions of words act as agents maximizing a sentence-level reward. Techniques include knowledge distillation on unlabeled data and post-processing to eliminate duplicate tokens.

Applications: While inspired by advancements in machine translation, these approaches aim to combine speed and quality in image captioning.

2.2.2.6 Discussion on Language model

Recurrent models, once dominant in image captioning, have been surpassed by autoregressive and Transformer-based solutions due to their faster training and ability to handle long-term dependencies. Massive pre-training, inspired by NLP advancements, has significantly improved performance by leveraging large, less-curated datasets with encoder-decoder or BERT-like architectures, often integrating textual tags. While BERT-like models excel in semantic learning, they are not inherently generative. Generative architectures with extensive pre-training now offer comparable or superior results, making them a promising focus for future research.

CHAPTER 3: Models Utilized in This Project

3.1 CLIPCap

ClipCap consists of three main components: CLIP, Mapping Network, and GPT-2:

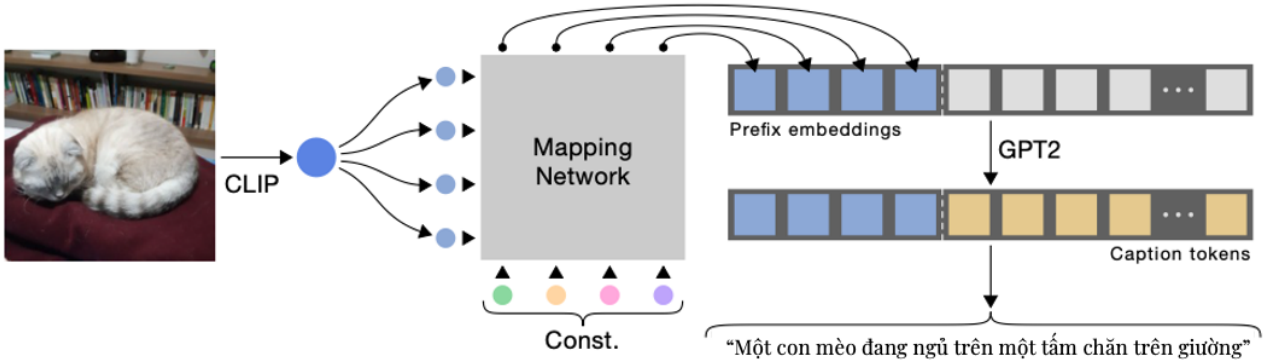


Figure 13: Overview of CLIPCap architecture, while both CLIP and GPT-2 are frozen. To extract a fixed length prefix, I train a lightweight transformer-based mapping network from the CLIP embedding space and a learned constant to GPT-2. At inference, I employ GPT-2 to generate the caption given the prefix embeddings.

- Pre-trained CLIP model: This component extracts the semantic features from images, enabling the model to understand and encode the visual content effectively.
- Pre-trained GPT-2: Using the semantic information provided by CLIP, GPT-2 generates descriptive captions that correspond to the given images.
- Mapping Network: This is the key component of the model. It transforms the encoding output from CLIP into a format compatible with GPT-2's word embeddings, serving as a bridge between the two models.

The training process of ClipCap primarily focuses on optimizing the Mapping Network. Since CLIP and GPT-2 are large-scale, pre-trained models, they do not require additional training. This design significantly reduces computational complexity while achieving high performance in image captioning tasks.

3.1.1 CLIP model

CLIP (Contrastive Language–Image Pre-training) is a powerful multimodal vision-language deep learning model developed by OpenAI in 2021. It was trained on an extensive dataset of 400 million (image, text) pairs.

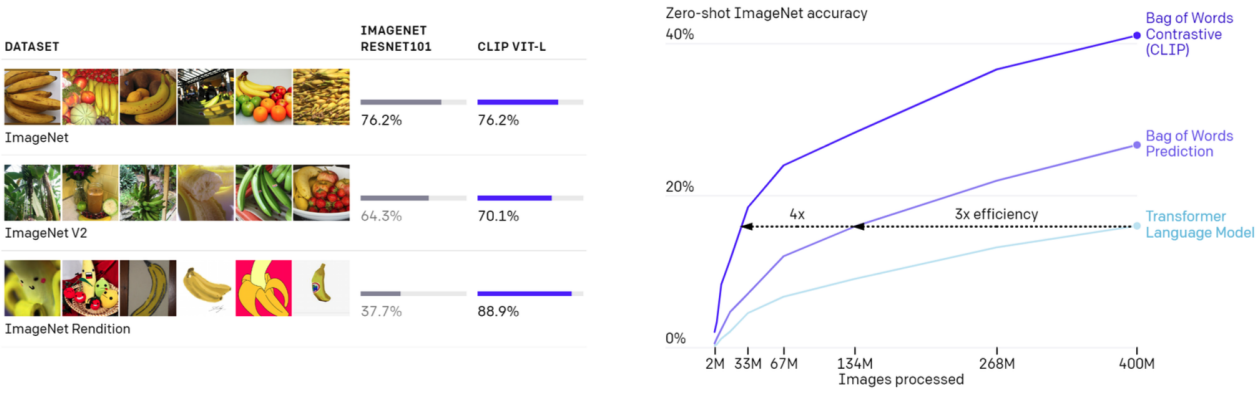


Figure 14: Overview of CLIP model

CLIP has the remarkable ability to intuitively learn a wide range of concepts, including objects and actions, from images and link them with their corresponding textual descriptions. This capability makes CLIP highly versatile:

- It can be used for various image classification tasks across a diverse range of classes.
- It supports zero-shot learning, enabling it to perform recognition tasks on data that it has not been explicitly trained or fine-tuned for—similar to OpenAI's GPT-2 and GPT-3 models in the domain of natural language processing.

Foundational Idea: The core concept of CLIP lies in encoding both images and text into a shared representation space. This shared encoding allows for the comparison of these modalities in a meaningful way. For instance, much like summarizing a book (text) and a movie (a combination of images and sound) to compare their content, CLIP aligns image and text representations, effectively bridging the gap between Computer Vision and Natural Language Processing (NLP).

$$max_\theta \sum_{i=1}^{N} logp_\theta(c_i|x_i)$$

By optimizing the connection between visual and textual data, CLIP serves as a robust bridge that facilitates seamless integration between these two traditionally separate domains, pushing the boundaries of multimodal AI.

3.1.1.1 How CLIP Works

The CLIP model comprises two sub-models: an image encoder and a text encoder, which process inputs into feature vectors to construct a similarity matrix (where $I \times T$ is computed as an inner product).

Similarity Matrix: $S = I \cdot T$

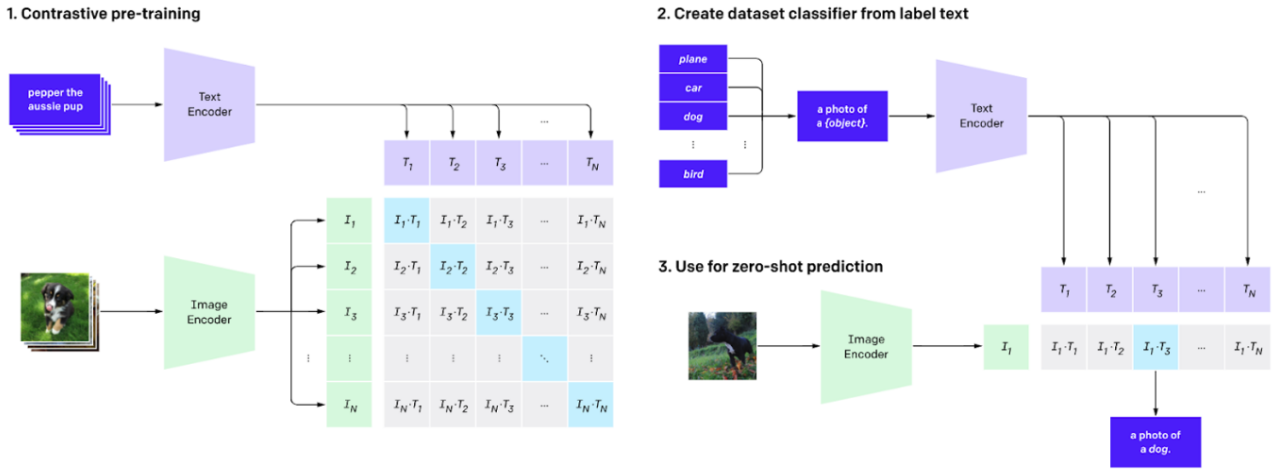


Figure 15: How CLIP model works

- Each row in the similarity matrix represents a classification task: given an image I, the output is the text corresponding to that image.
- Each column represents another classification task: given text T, the output is the image corresponding to that text.

During inference, the process involves:

1. Taking a set of labels and generating textual descriptions based on these labels.
2. Passing these textual descriptions through the text encoder to create text embeddings.
3. Matching the text embeddings with the image representations to determine the similarity between the input image and the generated text.

3.1.1.2 How CLIP Works in ClipCap

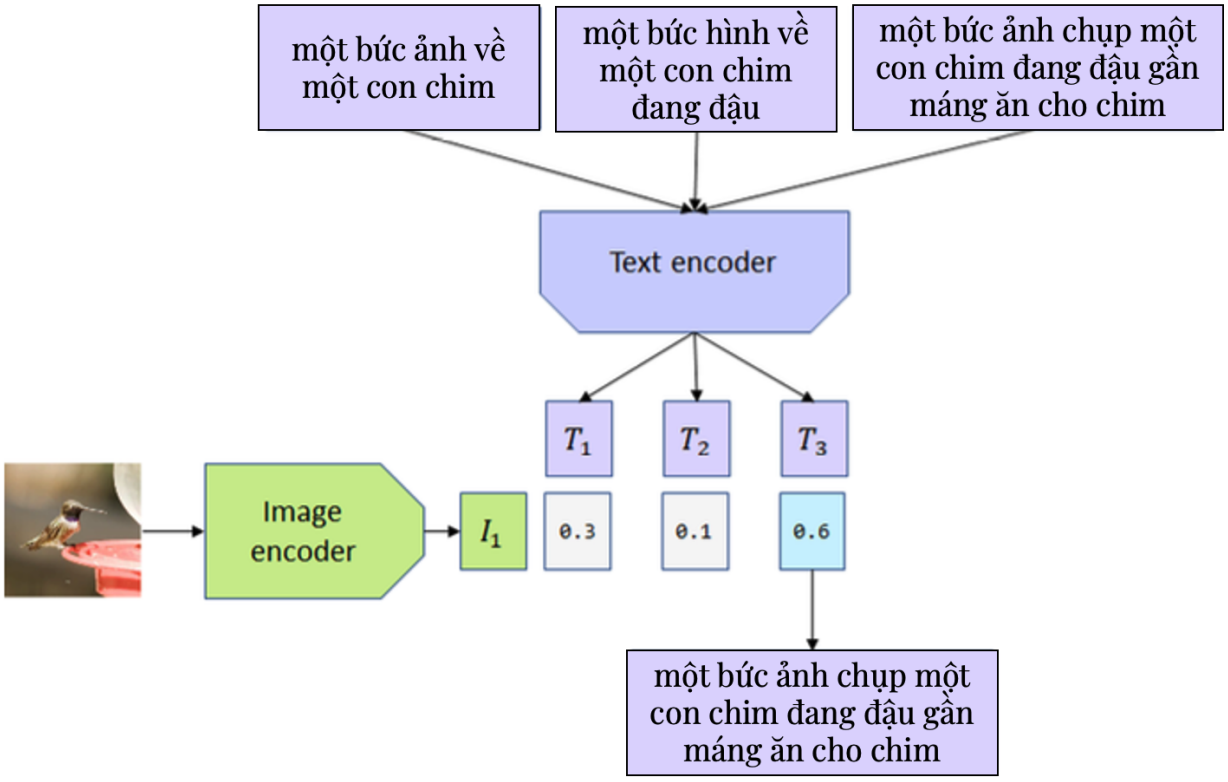


Figure 16: How CLIP model works in CLIPCap

In the ClipCap model, CLIP functions as a visual encoder, extracting semantic information (semantic encoding) from the input image. This role is crucial due to CLIP's exceptional ability to generalize beyond standard classification models. By understanding the underlying meaning of class labels in an image, CLIP provides remarkable flexibility and generalization, enabling it to handle a wide range of tasks effectively.
Key Benefits of CLIP in ClipCap:

- Generalization: CLIP's robust understanding of visual semantics makes it adaptable to various contexts without being constrained to specific datasets or tasks.
- Efficiency: Since CLIP is pre-trained on a massive dataset of image-text pairs, it can encode semantics for virtually any image without requiring additional supervision or annotations. This reduces the need for extensive labeled datasets, saving time and computational resources.

In ClipCap, these capabilities make CLIP an ideal choice for extracting meaningful visual information that can then be used to generate descriptive captions, highlighting its pivotal role in bridging the gap between vision and language tasks.

3.1.2 GPT-2

GPT-2 (Generative Pretrained Transformer 2) is a language model based on the Transformer architecture, developed by OpenAI in February 2019. Its primary purpose is to predict the next word in a sequence of

text. The model is open-source and trained with 1.5 billion parameters, enabling it to generate coherent text sequences based on a given input. The diversity of the training data allows GPT-2 to produce text across a wide range of domains.

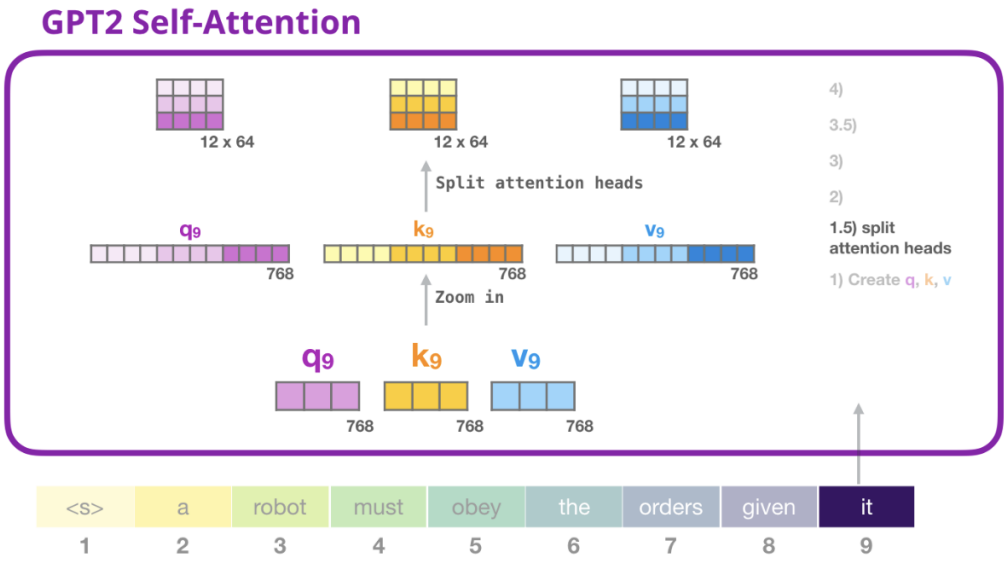


Figure 17: GPT-2

Compared to its predecessor, GPT, GPT-2 is significantly more powerful, featuring 10 times the parameters and trained on 10 times the data. There are four versions of GPT-2, categorized by size as follows:

- GPT-2 Small: 117 million parameters
- GPT-2 Medium: 345 million parameters
- GPT-2 Large: 762 million parameters
- GPT-2 Extra Large: 1.542 billion parameters

GPT-2 was trained on a large corpus called WebText, consisting of 40GB of text data collected by crawling web pages linked to Reddit posts with at least three upvotes (as of December 2017). According to the authors, WebText is of higher quality compared to other commonly used datasets, such as Common Crawl, frequently employed for training language processing models.

GPT-2 can learn and perform language tasks such as reading, summarization, and translation directly from raw text, without the need for task-specific datasets.

The model is trained using a next-word prediction task, where it predicts the next token based on the preceding context. The input data format requires a special token marking the start of a sequence. Each new token generated by the model is appended to the input sequence and used as input for the next iteration. This process repeats until a special token signaling the end of the caption is produced.

Leveraging the strengths of the Transformer architecture and its rich training dataset, GPT-2 demonstrates excellent performance across a variety of tasks beyond text generation, including: Question answering, Text summarization, Machine translation,...

In the original paper, the authors showcased GPT-2's impressive results on widely-used datasets, such as: WMT-14 Fr-En for machine translation, CoQA for text comprehension, CNN/Daily Mail for text summarization. These results were unexpected, as GPT-2 was initially designed solely for next-word prediction tasks. Its ability to generalize to other language-related tasks highlights its versatility and robustness, making it one of the significant breakthroughs in natural language processing.

3.1.3 Language model fine-tuning

$$\max_{\theta} \sum_{i=1}^{N} \sum_{j=1}^{l} \log p_{\theta}(c_j^i \mid x_i, c_1^i, \ldots, c_{j-1}^i)$$

One key challenge in training the ClipCap model lies in effectively bridging the semantic representation gap between CLIP (output from the CLIP visual encoder) and GPT-2 (input to the GPT-2 model). While both CLIP and GPT-2 are powerful models capable of generating diverse and rich outputs, the differences in their semantic representations—arising from their independent training processes—can lead to variations in the predicted caption concepts.
To address this challenge, the authors proposed two distinct approaches:

- **Approach 1: Fine-tuning GPT-2 During Mapping Network Training**: This method involves fine-tuning the language model (GPT-2) alongside training the Mapping Network.
  - Advantages: Fine-tuning GPT-2 improves its performance, leading to better language generation results.
  - Disadvantages: Fine-tuning significantly increases the number of trainable parameters, which affects both the model's size and efficiency. And the additional computational overhead can make the model less practical for large-scale or resource-constrained applications.
- **Approach 2: Prefix-Tuning with a Frozen GPT-2**: To mitigate the disadvantages of fine-tuning, the authors adopted a second approach: training only the Mapping Network while keeping GPT-2 frozen. This involves using a technique called prefix-tuning, where the model learns a prefix during training.
  - Advantages:**Lighter Model:** This approach drastically reduces the number of trainable parameters, making the model smaller and more efficient.**Faster Training:** By freezing GPT-2, training time is significantly reduced.**Comparable or Superior Performance:** Despite being more efficient, prefix-tuning achieves results comparable to, or even better than, state-of-the-art (SOTA) models in certain cases.

The authors also experimented with fine-tuning the CLIP model, but this approach did not yield any improvement in results. Instead, it increased training time and model complexity. The findings suggest that CLIP's representation space already encapsulates sufficient semantic information about the images, making additional adaptation for specific captioning styles unnecessary.

3.1.3.1 Apdater-tuning

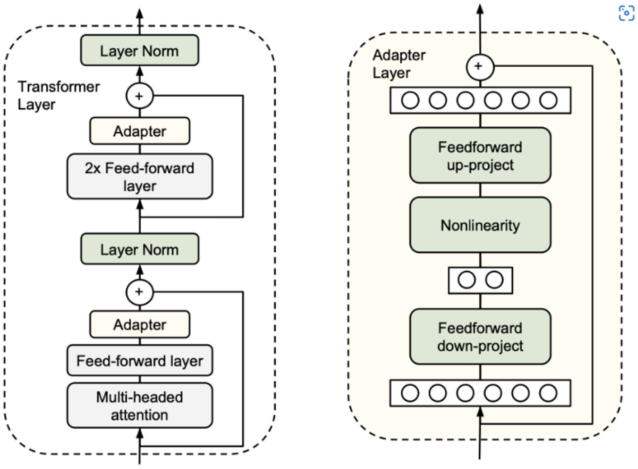


Figure 18: Adapter-tuning

Adapter-tuning: This is a technique used during the fine-tuning of language models by adding task-specific layers while keeping the parameters of the frozen language model unchanged. It helps save storage space while still achieving performance comparable to full fine-tuning.

By retaining the original parameters of the language model, adapter-tuning addresses the issue of storage inefficiency for different downstream tasks. However, the fine-tuning process still requires sufficient data to achieve good results, posing challenges in resource-constrained scenarios involving limited hardware and storage.

3.1.3.2 Prefix-tuning

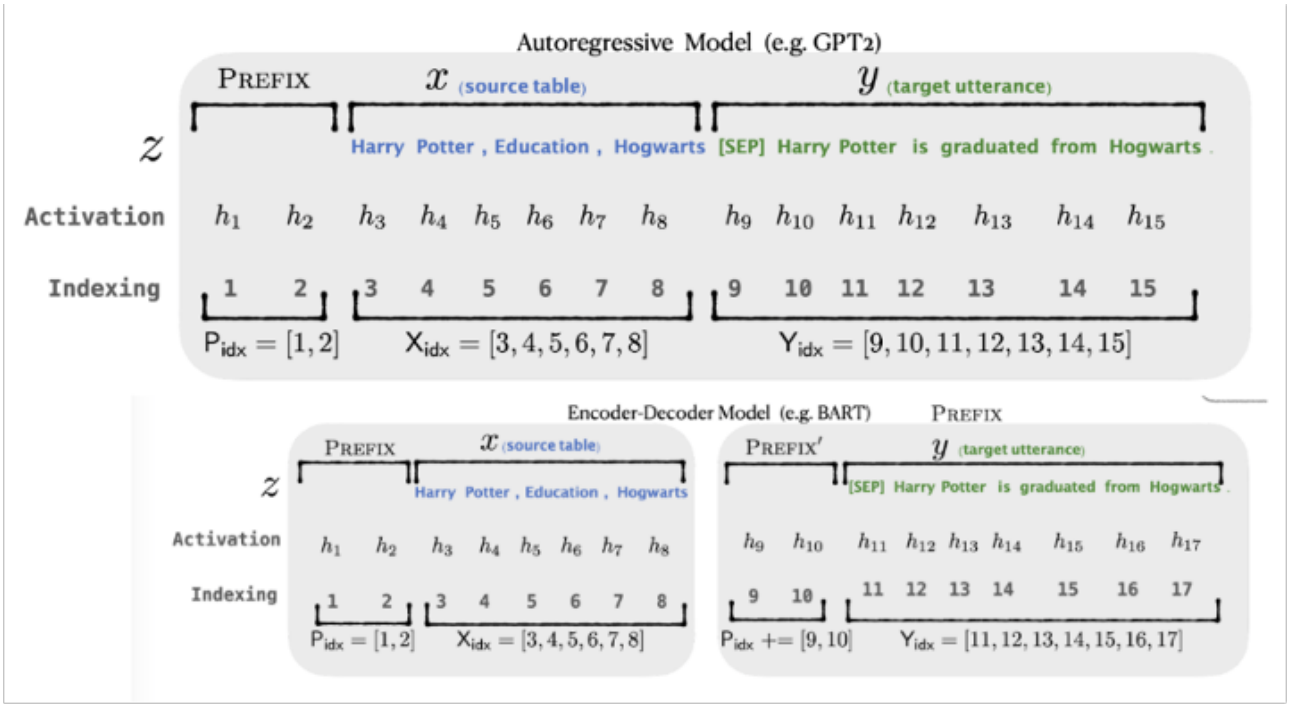


Figure 19: Prefix-tuning

If we aim to steer a language model toward generating a specific target word, prepending or preparing relevant phrases in advance can effectively increase the conditional probability of the desired output. This technique leverages the model's inherent ability to predict subsequent tokens based on its context.

The authors propose an optimized contextual prompting approach by defining prompts as continuous vectors rather than discrete words or pre-trained word embeddings. The main objective is to fine-tune these continuous prompts to align with the requirements of specific downstream tasks.

The process can be formalized as follows:

$$h_i = \begin{cases} P_\theta[i,:], & \text{if } i \in P_{\text{idx}} \\ LM_\phi(z_i, h_{<i}), & \text{otherwise} \end{cases}$$

h_i represents the hidden state at position i, determined either by the learned prefix P_{idx} or the standard autoregressive computation of the language model LM_ϕ, depending on the position index i

In an autoregressive model like GPT-2, the prefix-tuning method involves adding learned prefixes before the input tokens x and y. This generates activations h_1 and h_2, corresponding to the prefix length (e.g., prefix length = 2, as illustrated in diagrams).

The learned prefix is represented by a trainable matrix P_θ, whose dimensions are defined by the prefix length and the size of the activation vectors. For all indices outside the prefix, the activations are computed as in standard fine-tuning, relying on the model parameters ϕ.

3.1.4 Mapping Network

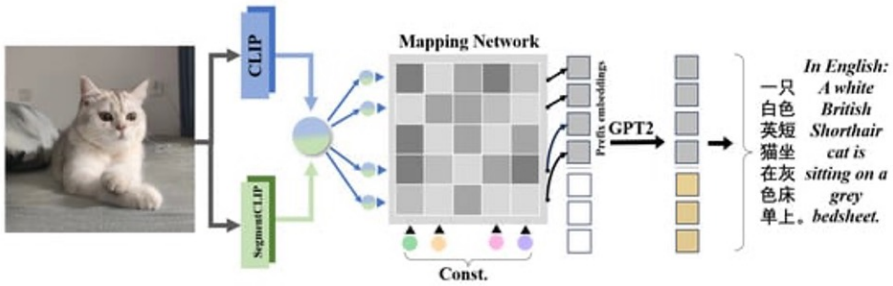


Figure 20: Mapping network

The Mapping Network is a key component of the ClipCap model, responsible for transforming/ matching the CLIP encoding (the output of CLIP) and the word embedding (the input of GPT-2).

$$p_{i1}, \ldots, p_{ik} = F(CLIP(x_i))$$

where $p_{i1}, ..., p_{ik}$ are the prefix embeddings, F is the mapping network, and $CLIP(x_i)$ represents the output of the CLIP visual encoder.

When fine-tuning the language model as in the initial approach described above, the mapping process becomes easier since the two models/networks can be easily controlled. Based on this, the authors proposed the first architecture for the Mapping Network as a single Multi-Layer Perceptron (MLP). Despite having only one hidden layer, this architecture yielded good results because the pre-trained CLIP already performs optimally for vision-language tasks.

Additionally, for the second approach (prefix-tuning, freezing the model), the authors proposed using a Transformer architecture for the Mapping Network. The Transformer network leverages a global attention mechanism on the input tokens, which helps reduce the number of parameters required for long sequences and increases the prefix size, thereby improving the model's performance.

3.1.4.1 Transformer

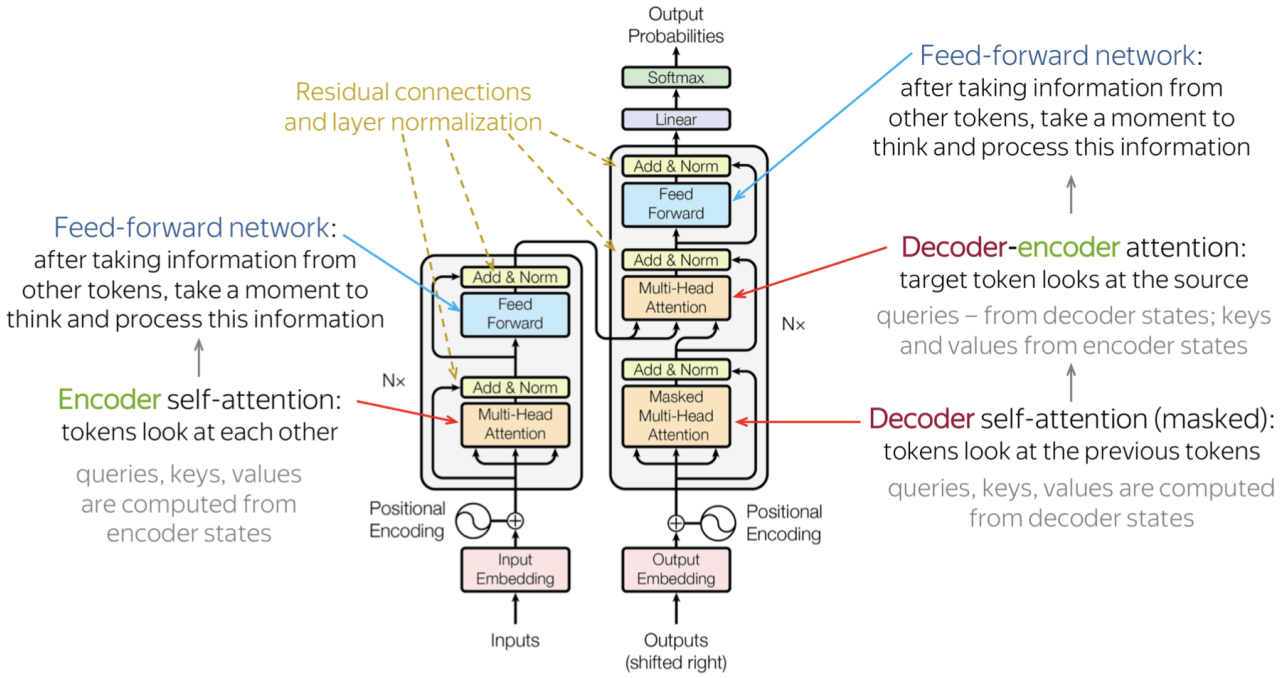


Figure 21: Transformer Architecture

Transformers were first introduced in the 2017 paper “Attention Is All You Need.” The architecture of the Transformer is described as follows: on the left side is the encoder, which typically consists of N_x=6 stacked layers. Each layer includes multi-head attention, as previously explored, and a feed-forward block. Additionally, residual connections, similar to those in ResNet, are present. On the right side is the decoder, also consisting of N_x=6 stacked layers. The architecture is quite similar to the encoder but includes an additional masked multi-head attention block at the beginning.

- **Positional Encoding**: Since the Transformer does not use recurrent or convolutional networks, it lacks awareness of the order of input tokens. Therefore, positional encoding is required to provide the model with this information. After the embedding layer generates token embeddings, positional vectors representing the position of words in the sentence are added to the embeddings.
- **Normalization Layer**: The term "Add & Norm" in the Transformer architecture refers to the presence of a normalization layer. This layer normalizes the output of the multi-head attention, improving the model's convergence during training.
- **Residual Connections**: Residual connections are straightforward: the input of a block is added to its output. This connection enables stacking multiple layers in the network. Residual connections are used after the FFN block and the attention block. The "Add" in "Add & Norm" signifies the residual connection.
- **Feed-Forward Block**: The feed-forward block (FFN) is fundamental. After the computations in the attention block in each layer, the subsequent processing occurs in the FFN. While attention mechanisms gather information from input tokens, the FFN processes this information.

Transformer Mapping Network in ClipCap: In the ClipCap model, the Mapping Network with a Transformer architecture takes two inputs: CLIP visual encoding and learned constant input. The learned constant input contains information derived from the CLIP encoding through the multi-head attention mechanism. This helps align and fine-tune the language model with new data.

The authors' experiments demonstrated the effectiveness of the Transformer architecture with the prefix-tuning approach. The learned constant effectively captures critical embeddings and detailed aspects of the CLIP encoding, improving the ability to interpret and generalize prefixes.

3.2 EfficientNet_v2 and Transformer

3.2.1 Encoder

3.2.1.1 EfficientNetv1

The idea behind EfficientNet is to employ a controller (a network such as RNN) to sample network architectures from a search space with a probability p. This architecture is then evaluated by training the network first, and then validating it on a test set to obtain an accuracy R. The gradient of p is calculated and scaled according to the accuracy R. The result (reward) is fed back to the RNN controller. The controller acts as an agent, the training and testing process acts as the environment, and the result acts as the reward. This is a common loop in Reinforcement Learning. This loop runs multiple times until the controller finds a network architecture that yields a high reward (high test accuracy).

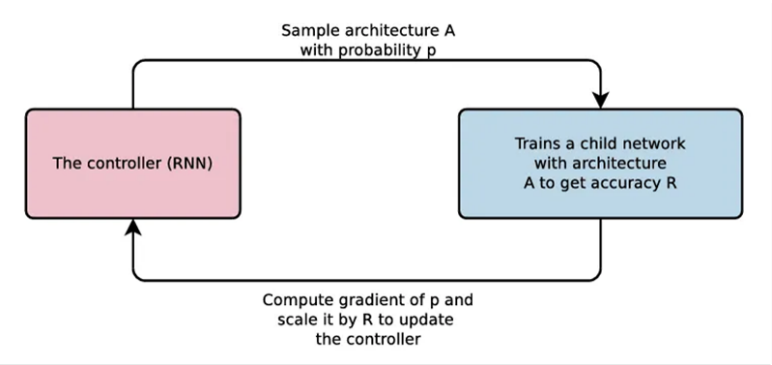


Figure 22: How controller works.

The RNN controller samples different network architecture parameters—such as the number of filters, filter height, filter width, stride height, and stride width for each layer. These parameters 1 can vary for each layer of the network. Ultimately, the network with the highest reward (accuracy) is selected as the final network architecture.

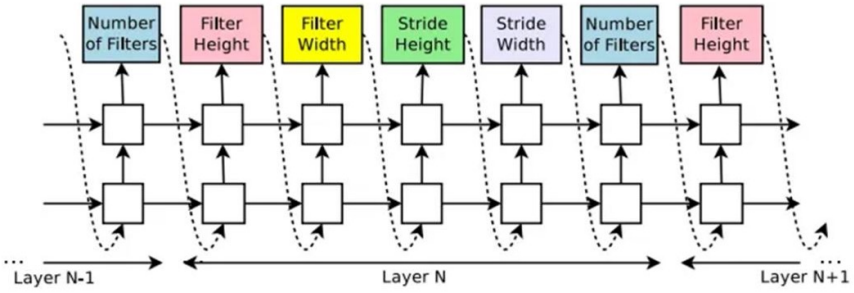


Figure 23: How controller works specifically

While this method works well, one of the issues with this approach is that it requires a significant amount of computational power and time. These architectures do not have different parameters in each layer, but rather have a cell with multiple convolutional layers (also known as ConvNet/CNN) and pooling layers, and throughout the entire network architecture, these cells are used multiple times. The authors utilized this idea to find such cells using a Reinforcement Learning controller and simply repeated these cells N times to create a scalable NASNet architecture. In this network, the authors selected 7 cells and one layer of the cell was sampled and repeated for each cell. In addition to these parameters, another crucial parameter considered when determining the reward was latency, which was fed into the controller. Thus, for MnasNet, the authors considered both accuracy and latency to find the best model architecture. This is illustrated in the figure below. This makes the architecture more compact and it can run on mobile or edge devices.

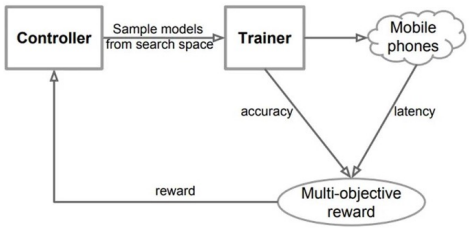


Figure 24: Architecture

The EfficientNet architecture search process is quite similar to MnasNet, but instead of considering 'latency' as the reward parameter, 'FLOPs (floating-point operations per second)' were considered. Searching with this criterion yielded a base model called EfficientNetB0. Next, scaling the depth, width, and image resolution of the base model (using a grid search) resulted in six additional models, from EfficientNetB1 to EfficientNetB7. The scaling ratios are shown in the figure below.

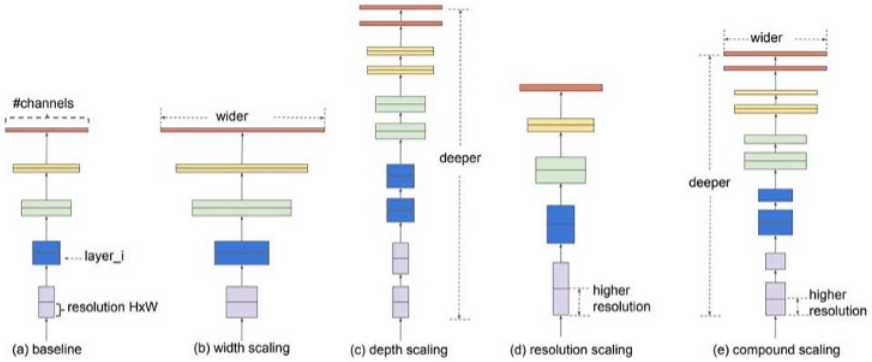
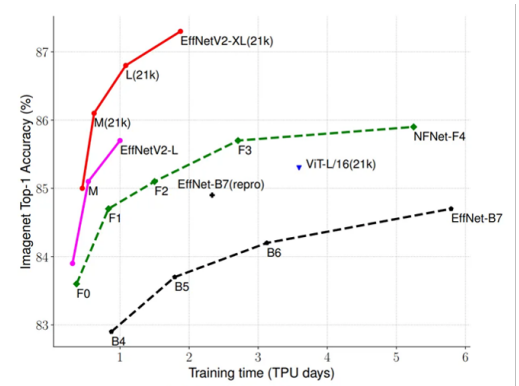


Figure 25: Ratio of depth, width and image resolution

3.2.1.2 EfficientNetv1 problems

EfficientNetV2 takes a step further than EfficientNet to accelerate training speed and improve parameter efficiency. This network is created by using a combination of scaling (width, depth, resolution) and neural architecture search. The primary goal is to optimize training speed and parameter efficiency. Additionally, the search space this time also includes new convolutional blocks like Fused-MBConv. Ultimately, the authors obtained EfficientNetV2 architectures that are much faster than previous and newer state-of-the-art models, while being significantly smaller (up to 6.8 times). This is illustrated in the figure below.



(a) Training efficiency.

	EfficientNet (2019)	ResNet-RS (2021)	DeiT/ViT (2021)	EfficientNetV2 (ours)
Top-1 Acc.	84.3%	84.0%	83.1%	83.9%
Parameters	43M	164M	86M	24M

(b) Parameter efficiency.

The Parameter efficiency figure clearly shows that EfficientNetV2 has 24 million parameters, while Vision Transformer (ViT) has 86 million parameters. The V2 version also has nearly half the parameters of the original EfficientNet. Despite significantly reducing the parameter size, it still maintains similar or higher accuracy compared to other models on the ImageNet dataset.

Additionally, progressive learning is also implemented, which is a method to gradually increase the image size along with regularizations like dropout and data augmentation. This method further accelerates training.

EfficientNets generally train faster than other large CNN models. However, when large image resolutions are used to train the models (models B6 or B7), the training process becomes slow. This is because larger EfficientNet models require larger image sizes to get optimal results and when using larger images, the batch size needs to be lowered to fit these images into GPU/TPU memory, slowing down the overall process.

In the early layers of the network architecture, depthwise separable convolutions (MBConv) are slow. Depthwise separable convolutions generally have fewer parameters than regular convolutions, but the problem is that they cannot fully utilize modern accelerators. To address this issue, EfficientNetV2 uses a combination of MBConv and Fused MBConv to train faster without increasing parameters.

An equal ratio is applied to height, width, and image resolution to create different EfficientNet models from B0 to B7. An equal ratio for all these layers is not optimal. For example, if the depth is scaled by 2, all blocks in the network will be scaled by 2 times, making the network very large/deep. It might be more optimal to scale one block by 2 times and another block by 1.5 times (uneven scaling), to reduce the model size while still maintaining good accuracy.

3.2.1.3 EfficientNetV2

As mentioned above, MBConv blocks often cannot fully utilize modern accelerators. Fused-MBConv layers can better utilize server/mobile accelerators.

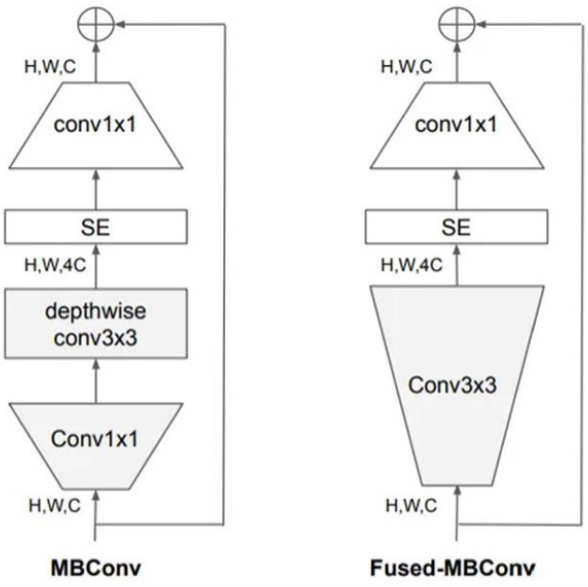


Figure 26: Architectures of MBConv and Fused MBConv

MBConv vs Fused-MBConv

As mentioned previously, the MBConv block, first introduced in MobileNets, consists of a depthwise separable convolution followed by a 1x1 convolution. EfficientNetV2 introduces the Fused-MBConv block, which replaces these two operations with a single 3x3 convolution. While MBConv blocks can be fused to accelerate training with only a small increase in parameters, using too many of them can slow down training due to the added parameters. To address this, the authors incorporated both MBConv and Fused-MBConv blocks into the neural architecture search, allowing the model to automatically determine the optimal combination for the best performance and training speed.

Neural Architecture Search

The neural architecture search was conducted to jointly optimize accuracy, parameter efficiency, and training efficiency. The EfficientNet model was used as a backbone, and the search process explored various design choices, including convolution blocks, the number of layers, filter sizes, expansion ratios, and more. Nearly 1000 models were sampled and trained for 10 epochs, and their results were compared. The model that was best optimized in terms of accuracy, training step time, and parameter size was selected as the final base model for EfficientNetV2.

EfficientNetV2 Architecture

The figure below shows the base model architecture of EfficientNetV2 (EfficientNetV2-S). The model incorporates Fused-MBConv layers at the beginning but later transitions to MBConv layers. For comparison, the base model architecture for the previous EfficientNet paper is also shown in Figure 9. The previous version only had MBConv layers and no Fused-MBConv layers.

Scaling EfficientNetV2

EfficientNetV2-S also has a smaller scaling ratio compared to EfficientNet-B0. EfficientNetV2 does not use 5x5 filters and only uses 3x3 filters.

After obtaining the EfficientNetV2-S model, it was scaled to obtain the EfficientNetV2-M and EfficientNetV2-L models. A compound scaling method was used, similar to EfficientNet, but with some modifications to make the models smaller and faster.

First, the maximum image size was limited to 480x480 pixels to reduce GPU/TPU memory usage, thus accelerating training. Second, more layers were added to the later stages to increase the network capacity without significantly increasing runtime costs.

Progressive Learning with Adaptive Regularization

Larger image sizes generally tend to yield better training results but increase training time. Some previous papers have proposed dynamically changing image sizes, but this often leads to accuracy loss during training.

The authors of EfficientNetV2 show that when the image size is changed dynamically during network training, the normalization should also be changed accordingly. Changing the image size but keeping the normalization constant leads to accuracy loss. Moreover, larger models require more regularization than smaller models.

The authors test their hypothesis by using different image sizes and different augmentations. As seen below, when the image size is small, weaker augmentations yield better results, but when the image size is large, stronger augmentations yield better results.

Considering this hypothesis, the authors of EfficientNetV2 used a Progressive Learning with Adaptive Regularization method. The idea is simple. In the initial steps, the network is trained on small images and weak regularization. This allows the network to learn features quickly. Then, the image size is gradually increased, and so are the regularizations. This makes the network harder to learn. Overall, this method achieves higher accuracy, faster training speed, and less overfitting.

The initial image size and normalization parameters are user-defined. Linear interpolation is then applied to increase the image size and normalization after a specific stage (M), as shown in the figure below. This is explained more visually in the end. As the number of epochs increases, the image size and augmentations increase gradually. EfficientNetV2 uses 3 different types of regularization - Dropout, RandAugment, and Mixup.

3.2.2 Decoder Transformer

3.2.2.1 Transformer

Transformer is a deep learning model introduced in 2017, primarily used in the field of natural language processing (NLP). It is considered a state-of-the-art deep learning model and offers high performance.

After obtaining image features in the form of image embedding matrices through the encoder, I will use the decoder architecture of the Transformer deep learning model to generate captions for images.

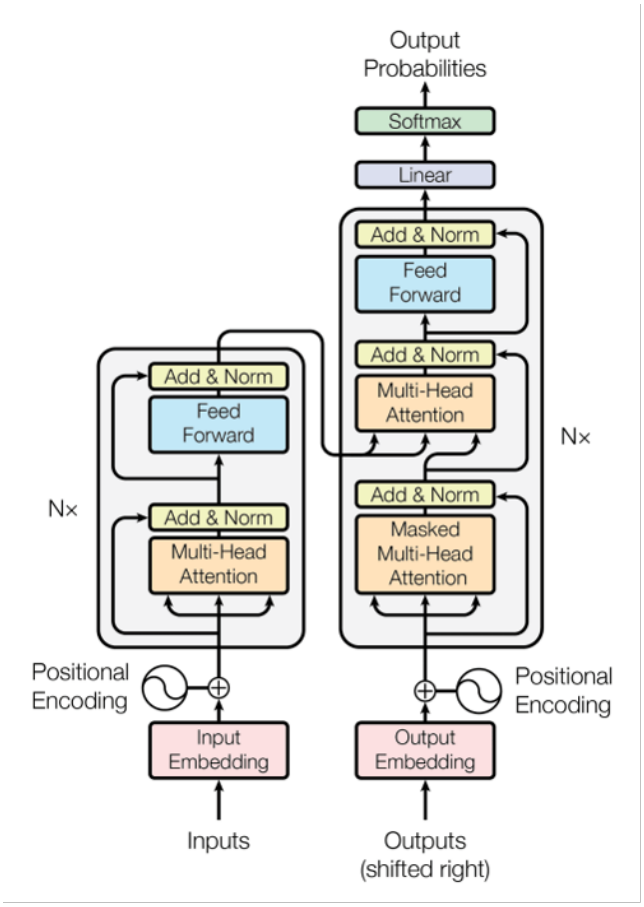


Figure 27: Transformer Architecture

3.2.2.2 Positional Encoding

Positional Encoding is a technique used to inject positional information into the input embeddings of a neural network model. The goal is to provide the model with information about the order and position of words in a sentence. This is crucial because neural networks typically operate on fixed-size vectors and do not have an inherent understanding of the order of elements in an input sequence. Therefore, without positional encoding, model would struggle to differentiate between different sequences containing same words in different orders.

By adding positional encoding to the input embeddings, the model can better understand the structure and meaning of the input text sequence.

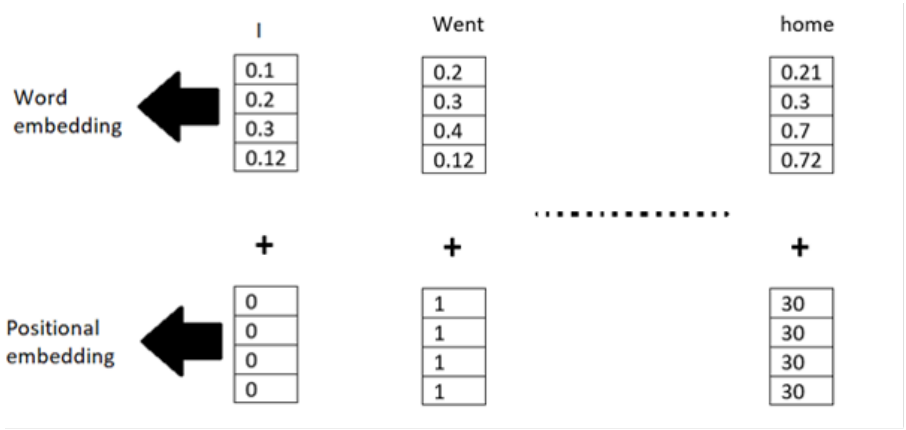
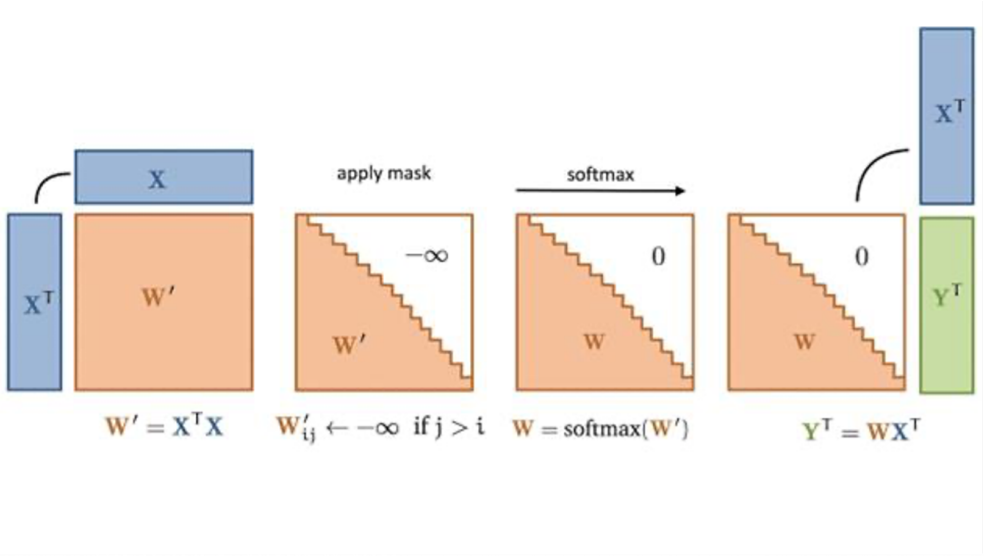


Figure 28: Positional embedding and word embedding

3.2.2.3 Masked Multi-Head Attention



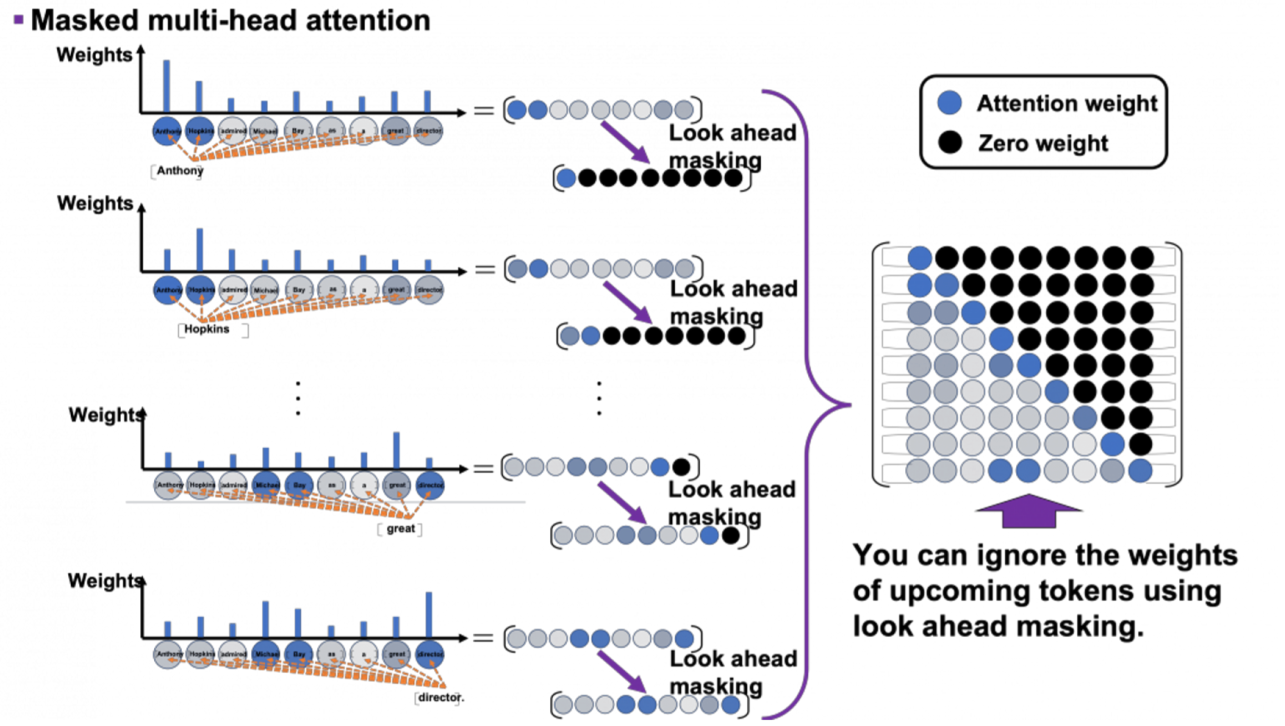


Figure 29: Masked Multi-Head Attention training progress

Masked Multi-Head Attention is an attention mechanism used in sequence-to-sequence neural network architectures. Its purpose is to allow the decoder to focus only on previously generated tokens rather than the entire input sequence. The Masked Multi-Head Attention mechanism works as follows:

1. Input to the mechanism is a sequence of embeddings, representing the previously generated output tokens.
2. The embeddings are transformed by three separate linear projections, creating query, key, and value vectors for each token. These vectors are then split into multiple heads, allowing the model to focus on different parts of the sequence.
3. Attention scores between query and key vectors are computed for each head, using the scaled dot-product attention mechanism. These attention scores are then used to weight the value vectors, and the weighted values are combined to produce a set of attention outputs.

4. The attention outputs are concatenated and passed through another linear projection, producing the final output of the mechanism. During training, the input sequence is masked to prevent the decoder from "seeing" future tokens. This is done by setting the attention scores for future tokens to negative infinity.

3.2.2.4 Multi-Head Attention

Multi-Head Attention in the Transformer decoder is used to generate words in image captions. Specifically, it calculates the importance of words in the caption relative to the encoded image. This allows the model to focus on different parts of the image to generate more accurate descriptive words in the caption.

To perform Multi-Head Attention, the input to the attention mechanism includes query vectors and value vectors. Query vectors are generated by taking the output of the previous linear layer (Masked Multi-Head Attention) and passing it through another linear layer to create query vectors (Add & Norm). Value vectors are generated by taking the output of the image encoder and passing it through a linear layer to create value vectors (image embedding).

Subsequently, the query and value vectors are split into multiple heads and passed into the attention function. Attention scores between query and value vectors are computed for each head, and the weighted values are combined to produce a set of attention outputs. These attention outputs are then concatenated and passed through another linear layer to produce representations for words in the image caption.

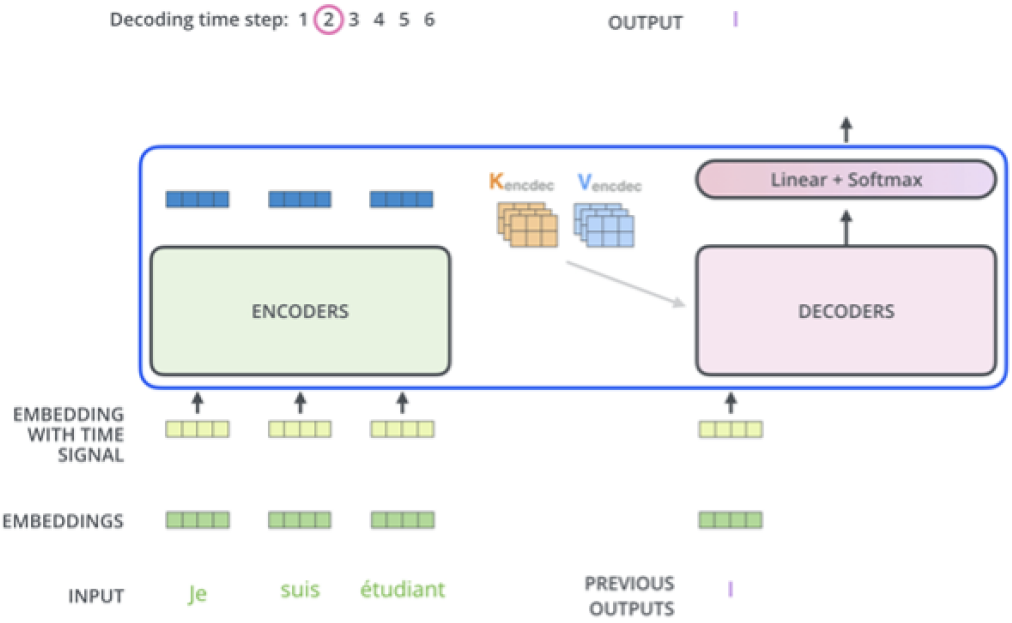


Figure 30: How Multi-Head Attention works

CHAPTER 4: How to use those models

4.1 CLIPCap

4.1.1 Preparing the Model and Data

4.1.1.1 Load the CLIP Model and Preprocessing Function

CLIP Model:

- The model used is ViT-B/16, a vision-language model developed by OpenAI.
- It transforms input images into feature vectors in a shared semantic space, enabling connections between visual and textual representations.
- CLIP not only encodes the visual content but also captures its semantic meaning, facilitating natural language generation.

Preprocessing:

- Input images are processed using the `preprocess` function to standardize size, aspect ratio, and color format.
- This ensures compatibility with CLIP's requirements and optimizes feature extraction.

4.1.1.2 Load the GPT-2 Tokenizer

- The GPT-2 Tokenizer is used to encode textual input into numerical tokens and decode token sequences into human-readable text.
- The customized version (`imthanhlv/gpt2news`) is specifically tailored for generating text descriptions related to visual content.
- The tokenizer also removes padding tokens (`<pad>`) to improve output accuracy.

4.1.1.3 Custom Model - CLIPCap

CLIPCap (ClipCaptionPrefix):

- Model maps embeddings from CLIP (image space) to embeddings compatible with GPT-2 (language space).
- **How it works:**
  - **Input:** Embeddings from CLIP representing the image.
  - **Output:** Sequential embeddings used by GPT-2 for text generation.

4.1.2 Encoding the Image into Feature Vectors

4.1.2.1 Image Preprocessing

- The input image is loaded and preprocessed using `preprocess`.
- The processed image is converted into a tensor with a compatible format and size for the CLIP model.

4.1.2.2 Extracting Embeddings from CLIP

- The CLIP model generates feature embeddings for the preprocessed image.
- These embeddings represent the semantic information of the image.

4.1.2.3 Mapping CLIP Embeddings to GPT-2 Space

- The embeddings from CLIP are passed through the CLIPCap model (`ClipCaptionPrefix`) to map them into GPT-2-compatible space.
- The resulting sequential embeddings represent image in a format suitable for natural language generation.

4.1.3 Generating Captions

4.1.3.1 Beam Search Method (`generate_beam`)

Beam Search is an optimization technique for text generation. It evaluates not only the next token's probability but also the quality of the entire sequence.

How Beam Search Works

- **a) Initialization:**
  - The method starts by creating `beam_size` candidates based on the highest-probability tokens.
  - Each candidate is scored using cumulative probabilities.
- **b) Beam Expansion:**
  - At each step, the method expands the current sequences by adding the most probable next tokens.
  - It retains only the top `beam_size` sequences based on average scores (cumulative probability divided by sequence length).
- **c) Stopping Criteria:**
  - The process stops when all sequences contain the stop token (`stop_token`) or reach the maximum sequence length (`entry_length`).
  - The best sequence (highest-scoring) is selected as the output.

4.1.3.2 Advantages of Beam Search

- **Higher Quality:** It considers multiple possibilities instead of greedily selecting the most probable token at each step.
- **Reduced Errors:** Illogical or incomplete sequences are less likely to be selected.

4.1.4 Results

- The input image is displayed alongside the generated caption.
- The caption is cleaned to remove padding tokens (`<pad>`) for better readability.

4.2 EfficientNetV2 + Transformer

4.2.1 Data Annotation Preprocessing

- **Text Normalization**: Remove special characters, digits, and convert all text to lowercase.
- **Tokenization and Padding**: Split the normalized captions into tokens and add a special `<pad>` token to ensure a fixed maximum length for all captions.
- **Vocabulary Encoding**: Construct a mapping dictionary between words and indices:

$$\text{word\_to\_idx} : \text{word} \to \text{index}, \quad \text{idx\_to\_word} : \text{index} \to \text{word}.$$

4.2.2 Feature Extraction from Images Using EfficientNetV2

- **Utilize a Pre-trained EfficientNetV2 Model**:
  - Extract features from the last convolutional layer of the network.
  - Transform the resulting tensor into a format compatible with the Transformer model, maintaining dimensional consistency.

4.2.3 Transformer Decoder Model Construction

- **Positional Encoding**: Add positional information to token embeddings, defined as:

$$\text{PE}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad \text{PE}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right).$$

- **Transformer Decoder**: A stack of multiple transformer decoder layers processes both image features and caption sequences to predict the next token.
- **Masking Mechanism**: Create masks to ensure the model only considers past and present tokens during prediction, adhering to causal constraints.

4.2.4 Caption Generation

- **Generate Captions**: Use a beam search algorithm to produce captions by selecting the sequence with the highest overall probability.
  - **a) Initialization:**
    - Create the initial beams starting with the `<start_token>`.
    - Store the model states for each beam.
    - Maintain cumulative scores for each sequence in the beam.
  - **b) Beam Expansion:**
    - At each step, compute the probabilities for the next tokens based on the current beam states.
    - Select the most probable tokens to extend the current sequences.
    - Update the average scores for each sequence based on cumulative probabilities.
  - **c) Stopping Criteria:**
    - Terminate when:
      - All beams contain the `<end_token>`.
      - Or the maximum sequence length is reached.
    - The sequence with the highest score is selected as the final output.
- **Integrate Image Features**: Combine the extracted image embeddings with token sequences to iteratively predict the next token in the sequence.

CHAPTER 5. EXPERIMENTS

5.1 Dataset

In this report, I use the KTVIC dataset (Knowledge Technology Lab's Vietnamese Image Captioning) to train an image captioning model tailored for the Vietnamese language context.

KTVIC is an image captioning dataset developed with a focus on daily life and activities. The dataset was created by a group of researchers, including Anh-Cuong Pham, Van-Quang Nguyen, Thi-Hong Vuong, and Quang-Thuy Ha, and was officially published in 2024. The images in KTVIC are sourced from the UIT-EVJVQA dataset and are annotated with five distinct captions per image, resulting in a total of 21,635 high-quality captions. This extensive annotation allows the model to capture the diversity and richness of real-life image contexts.

Compared to existing Vietnamese image captioning datasets, such as VieCap4H, KTVIC stands out due to its diversity of objects and scenes within the images and the inclusion of multiple captions per image. This diversity contributes significantly to the quality and accuracy of the resulting image captioning models. Furthermore, the dataset supports the development of models that better understand and generate natural and contextually relevant Vietnamese captions, advancing the progress in Vietnamese natural language processing and computer vision research.

KTVIC's comprehensive nature not only ensures better representation of real-world scenarios but also addresses some limitations of prior datasets, such as limited variety in scenes and less emphasis on contextual richness. By using this dataset for image captioning tasks, this research aims to push the boundaries of Vietnamese image captioning, opening opportunities for further applications in AI-driven tools for the Vietnamese-speaking community.

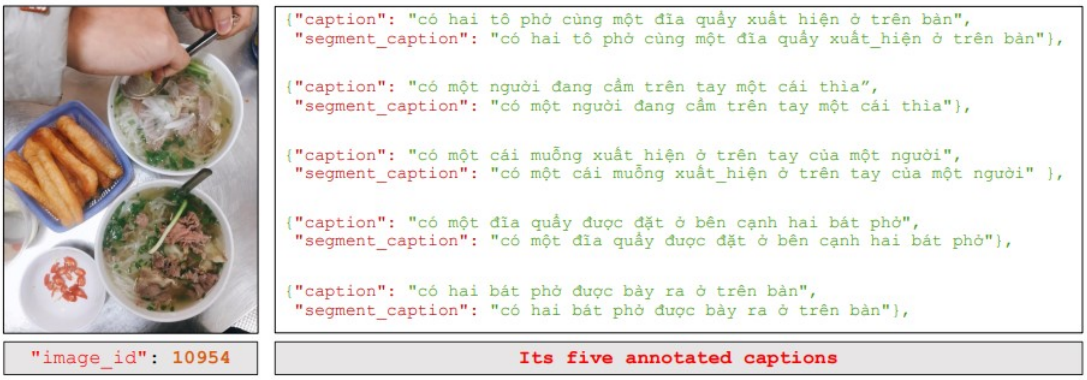


Figure 31: An example of a sample in the KTVIC dataset, where each image is paired with five captions.

5.2 Evaluation Metrics

Common metrics used to evaluate image captioning models can be categorized into two main groups: NLP-based metrics and image-specific metrics. In the NLP-based metrics group, measures such as BLEU and ROUGE are frequently used, while CIDEr is the most popular among image-specific metrics. These metrics assess the quality of image captioning models, each providing a different perspective, resulting in a comprehensive evaluation of the model's performance.

5.2.1 BLEU

BLEU (Bilingual Evaluation Understudy) is one of the most widely used metrics to evaluate the quality of text generation models, including image captioning systems. BLEU measures the similarity between candidate captions generated by the model and reference captions written by humans.

While BLEU is not a perfect metric, it has advantages such as fast computation, low cost, language independence, and most importantly, a high correlation with human evaluation. This makes BLEU an essential tool for assessing the quality of automatic text generation systems.

BLEU operates by comparing n-grams (continuous sequences of n words) between the predicted caption and reference captions. Higher BLEU scores indicate closer matches to reference captions.
Key components of BLEU include:

- n-gram Precision: The proportion of n-grams in the predicted caption that appear in the reference captions.
- Brevity Penalty (BP): A penalty applied if the predicted caption is too short compared to the reference, to discourage overly short but precise outputs.

The BLEU score is calculated as follows:

$$\textbf{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where Brevity Penalty (BP) is defined as:

$$BP = \begin{cases} 1 & \text{if } c > r, \\ \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r, \end{cases}$$

Where:

p_n : n-gram precision,

w_n : Weight for each n-gram (commonly $w_n = \frac{1}{N}$),

c : Length of the predicted caption,

r : Average length of the reference captions.

5.2.2 ROUGE-L

ROUGE-L (Recall-Oriented Understudy for Gisting Evaluation using Longest Common Subsequence) is a variant of the ROUGE metric specifically designed to evaluate the quality of text generation models by considering the longest common subsequence (LCS) between candidate captions and reference captions. This approach captures both precision and recall, emphasizing the sequential structure of text.

ROUGE-L is particularly useful for tasks like image captioning, where the order of words contributes to the overall coherence and meaning of the generated captions. By focusing on subsequences, ROUGE-L is better suited to evaluate linguistic and structural alignment compared to simple n-gram overlap.
ROUGE-L is calculated as follows:

$$\text{ROUGE-L} = F_{\text{LCS}}$$

Where the F-measure for LCS is:

$$F_{\text{LCS}} = \frac{(1 + \beta^2) \cdot R_{\text{LCS}} \cdot P_{\text{LCS}}}{R_{\text{LCS}} + \beta^2 \cdot P_{\text{LCS}}}$$

Where:

P_{LCS} : Precision based on the LCS, calculated as $\frac{\text{LCS(candidate, reference)}}{\text{Length of candidate}}$,

R_{LCS} : Recall based on the LCS, calculated as $\frac{\text{LCS(candidate, reference)}}{\text{Length of reference}}$,

β : Weighting factor (commonly set to $\beta = 1$ to equally weight precision and recall).

The Longest Common Subsequence (LCS) is defined as the longest sequence of words that appears in both the candidate and reference captions in the same order. ROUGE-L emphasizes the importance of maintaining the original sequence, providing a robust measure for assessing the quality of generated captions.

5.2.3 CIDEr

CIDEr (Consensus-based Image Description Evaluation) is a metric specifically designed to evaluate image captioning models. CIDEr was developed to enhance the evaluation of captions generated by models, considering similarity with multiple reference captions while also factoring in semantic richness and diversity in image descriptions.

CIDEr is particularly useful when evaluating models that generate longer or more creative captions, where metrics like BLEU or ROUGE may fall short. For a comprehensive evaluation of model quality, CIDEr is often used in combination with other metrics.

CIDEr evaluates image captions based on the consensus between predicted captions and reference captions. The CIDEr score is calculated based on the following factors:

- n-gram Matching: Assesses the overlap of n-grams between predicted and reference captions.
- Multi-reference Consensus: Considers similarity across multiple reference captions to reduce bias.
- Coverage: Weights n-grams that frequently appear in reference captions.
- Semantic Fit: Evaluates the richness and accuracy of semantic description in captions.

$$\text{CIDEr} = \frac{1}{N} \sum_{i=1}^{N} \text{IDF}_i \cdot \text{TF-IDF}_i$$

Where:

N : Number of n-grams in the caption.

TF-IDF_i : Term frequency-inverse document frequency of the ith n-gram in the predicted caption.

IDF_i : Inverse Document Frequency, indicating the importance of the n-gram in the reference set.

5.2.4 METEOR

METEOR (Metric for Evaluation of Translation with Explicit ORdering) is a metric designed to improve upon the limitations of BLEU by addressing issues such as synonymy, stemming, and word order. Unlike BLEU, which focuses solely on precision, METEOR incorporates both precision and recall to provide a more balanced evaluation of text similarity.

METEOR is particularly effective in evaluating text generation tasks like image captioning, where flexibility in phrasing and semantic similarity are important. By considering synonyms and morphological variations, METEOR offers a more nuanced assessment of the generated captions.
Key components of METEOR include:

- Unigram Matching: Matches unigrams between the predicted and reference captions, including exact matches, stemmed matches, and synonym matches.
- Precision and Recall: Calculates the harmonic mean of precision and recall to balance the evaluation.
- Fragmentation Penalty: Penalizes disjointed matches to reward better word order and coherence.

The METEOR score is calculated as follows:

$$\text{METEOR} = F_{\text{mean}} \cdot (1 - P_{\text{frag}})$$

Where:

$F_{\text{mean}} = \dfrac{10 \cdot P \cdot R}{R + 9 \cdot P}$ (harmonic mean of precision and recall),

P : Precision (proportion of matched words in the predicted caption),

R : Recall (proportion of matched words in the reference captions),

P_{frag} : Fragmentation penalty to penalize disjoint matches.

5.2.5 SPICE

SPICE (Semantic Propositional Image Caption Evaluation) is a semantic-based metric specifically designed for evaluating image captions. Unlike n-gram-based metrics like BLEU or ROUGE, SPICE evaluates the quality of captions by assessing the semantic content, focusing on how well the generated captions capture the relationships and objects in the image.

SPICE calculates similarity based on scene graphs, which represent objects, attributes, and relationships in the captions. This approach makes SPICE particularly effective for capturing semantic richness and coherence, making it an essential metric for tasks requiring detailed and contextually accurate descriptions.
Key features of SPICE include:

- Scene Graph Matching: Constructs semantic scene graphs from both predicted and reference captions to evaluate their similarity.
- Semantic Accuracy: Focuses on capturing relationships and attributes, providing a deeper understanding of caption quality.
- Interpretability: Offers an intuitive evaluation by focusing on semantic content rather than n-grams alone.

The SPICE score is calculated as follows:

$$\text{SPICE} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

Where:

TP : True Positives (correctly matched semantic tuples),

FP : False Positives (incorrect semantic tuples in the predicted caption),

FN : False Negatives (missing semantic tuples in the predicted caption).

SPICE provides a complementary perspective to n-gram-based metrics by prioritizing semantic fidelity, making it an invaluable tool for evaluating the effectiveness of image captioning systems.

5.3 Training strategies

Parameter	**EfficientNet_v2+ Transformer**	**CLIPCap**
Learning rate	1e-6	1e-6
Scheduler	ReduceLROnPlateau factor: 0.1; patience: 1	ReduceLROnPlateau factor: 0.1; patience: 1
Optimizer	Adam	Adam
Loss Function	CrossEntropyLoss	CrossEntropyLoss
Batch size	64	16
Early stopping	patience: 3	patience: 3
Epochs	200 (Stop at 117)	50 (Stop at 40)
Accelerator	Kaggle (GPU P100)	Kaggle (GPU P100)
Training Time	9064s	13771s

Table 1: Training hyperparameters & environment

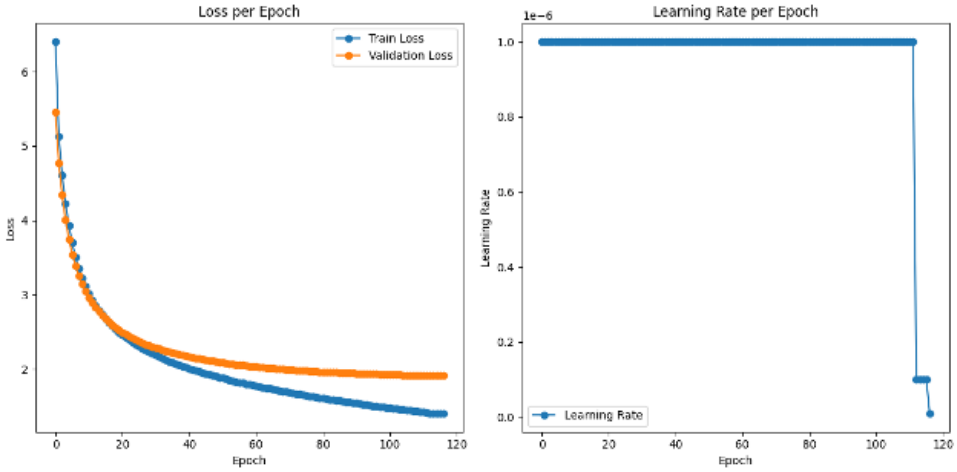


Figure 32: Loss and learning rate on EfficientNetV2+ Transformer

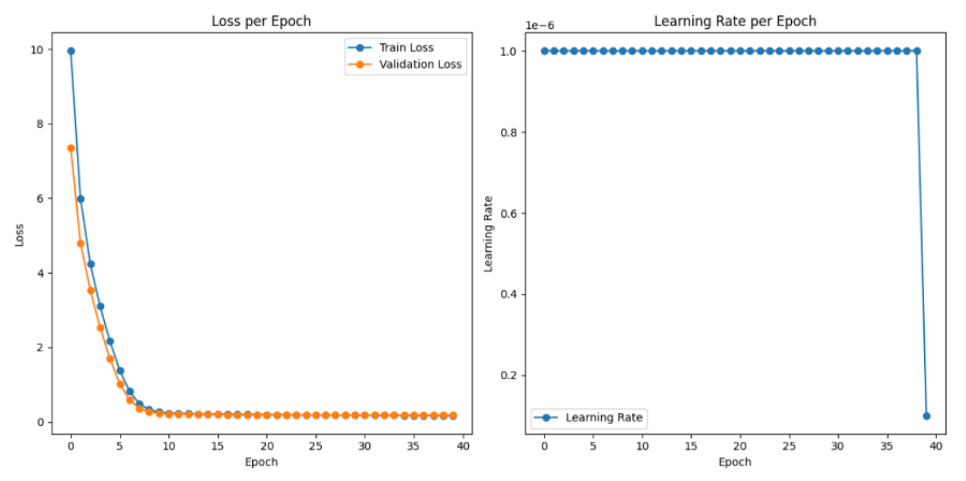


Figure 33: Loss and learning rate on CLIPCap

The table above summarizes the training settings for **EfficientNetV2 + Transformer** and **CLIPCap**, both trained on Kaggle using NVIDIA P100 GPUs.

Common Parameters:

- **Learning Rate:** Both models start with an initial learning rate of 1×10^{-6}, adjusted using the **ReduceLROnPlateau** scheduler.
  - **Scheduler Settings:**
    - `factor`: 0.1 (reduces the learning rate by 10% when performance plateaus).
    - `patience`: 1 epoch (waits for one epoch of no improvement before reducing the learning rate).
- **Optimizer:** Both models use the **Adam** optimizer for efficient weight updates.
- **Loss Function:** The **CrossEntropyLoss** is used to measure prediction errors.
- **Early Stopping:** Both models employ early stopping with a patience of 3 epochs to prevent overfitting.
- **Accelerator:** Both models were trained on Kaggle using NVIDIA P100 GPUs.

Differences:

- **Batch Size:**
  - **EfficientNetV2 + Transformer:** Uses a larger batch size of 64, leveraging its architecture and memory efficiency.
  - **CLIPCap:** Uses a smaller batch size of 16 due to its higher memory requirements.
- **Epochs and Early Stopping:**
  - **EfficientNetV2 + Transformer:** Trained for up to 200 epochs but stopped early at epoch 117. The learning rate was reduced to 1×10^{-7} at epoch 113 and further reduced to 1×10^{-8} at epoch 117.
  - **CLIPCap:** Trained for up to 50 epochs but stopped early at epoch 40. The learning rate was reduced to 1×10^{-7} at epoch 40.
- **Training Time:**
  - **EfficientNetV2 + Transformer:** Total training time was 9064 seconds ($\sim$ 2.5 hours).
  - **CLIPCap:** Total training time was 13771 seconds ($\sim$ 3.8 hours), reflecting differences in batch size and architecture.

Both models share many similarities in their training setup, such as learning rate schedules, optimizers, loss functions, and early stopping strategies. The primary differences lie in the batch size, number of epochs, and training times, reflecting the distinct architectural requirements and computational demands of each model.

5.4 Results

Metric	EfficientNet_v2 + Transformer	CLIPCap
BLEU-1	0.6120	**0.6731**
BLEU-2	0.4604	**0.5437**
BLEU-3	0.3387	**0.4316**
BLEU-4	0.2572	**0.3431**
ROUGE-L	0.4895	**0.5204**
CIDEr	0.6282	**0.8127**
METEOR	0.2995	**0.3194**
SPICE	0.0782	**0.0829**

Table 2: Evaluation results table according to some metrics

Observations on the Results:

BLEU Scores (BLEU-1 to BLEU-4): CLIPCap outperforms EfficientNet_v2 + Transformer across all BLEU metrics, with a notable margin in BLEU-4 (0.3431 vs. 0.2572). This indicates that CLIPCap generates predictions with higher semantic accuracy compared to the reference text.

ROUGE-L: CLIPCap achieves a higher ROUGE-L score (0.5204 vs. 0.4895), suggesting it better preserves the overall structure and key information of the reference.

CIDEr: CLIPCap has a significantly higher CIDEr score (0.8127 vs. 0.6282). This highlights its superior alignment with the reference text from a general semantic perspective, especially in scenarios where textual relevance matters.

METEOR: CLIPCap also leads in METEOR (0.3194 vs. 0.2995), showing improved consistency in semantic and lexical overlap with the reference text.

SPICE: CLIPCap achieves a higher SPICE score (0.0829 vs. 0.0782). SPICE is sensitive to semantic relationships, indicating that CLIPCap better captures these aspects.

Summary:

- CLIPCap consistently outperforms EfficientNet_v2 + Transformer across all metrics, demonstrating its ability to generate more accurate, coherent, and semantically rich text outputs.
- EfficientNet_v2 + Transformer, while less effective than CLIPCap, still performs reasonably well, indicating competitive capability in generating textual descriptions.

CHAPTER 6: CONCLUSION & DIRECTIONS

6.1 Conclusion

In this study, I compared two Image Captioning models: EfficientNet_v2 + Transformer and CLIPCap, with the aim of evaluating the performance of these models on popular metrics such as BLEU, ROUGE, CIDEr, METEOR and SPICE.

EfficientNet_v2 + Transformer:

Advantages:

- Efficient feature extraction: Utilizes EfficientNet_v2, which is known for its efficiency in feature extraction, combining high accuracy with computational efficiency.
- Transformer's capability: Leverages the power of the Transformer architecture, which excels in capturing long-range dependencies and contextual information.
- Moderate training time: Faster training time (9064 seconds) compared to CLIPCap, making it more suitable for scenarios with limited computational resources.

Disadvantages:

- Lower performance: Underperforms CLIPCap across all evaluation metrics (BLEU, ROUGE, CIDEr, METEOR, SPICE), indicating less effectiveness in generating high-quality text.
- Limited semantic understanding: Struggles to capture deep semantic relationships as reflected in its lower SPICE and CIDEr scores.
- Potential for overfitting: Requires careful tuning to avoid overfitting, as Transformers can be data-hungry.

CLIPCap

Advantages:

- Superior text generation: Outperforms EfficientNet_v2 + Transformer on all metrics, demonstrating its ability to generate coherent, semantically rich, and contextually accurate text.
- Pretrained CLIP embeddings: Leverages CLIP embeddings, which are highly effective at capturing cross-modal (image-text) relationships, enhancing text generation quality.
- Strong semantic understanding: High SPICE and CIDEr scores indicate superior understanding and representation of semantic relationships.

Disadvantages:

- Higher computational cost: Requires significantly longer training time (13771 seconds), which may not be ideal for environments with constrained resources.
- Dependency on CLIP: Relies on the quality of pretrained CLIP embeddings, which may limit flexibility for certain tasks or domains not well represented in the CLIP training data.
- Larger batch size limitations: Uses a smaller batch size (16 vs. 64 for EfficientNet_v2 + Transformer), which might slow down training further in specific setups.

The experimental results show that CLIPCap outperforms EfficientNet_v2 + Transformer on all evaluation metrics. This may be due to CLIPCap taking better advantage of the relationship between images and language thanks to its architecture.

The CLIPCap model achieved the highest BLEU-4 score (**0.3431**) and had the highest CIDEr score (**0.8127**), demonstrating its ability to describe images in more detail and accurately.

However, the major weakness of CLIPCap is that the training time is much longer than EfficientNet_v2 + Transformer (**13771** seconds compared to **9132** seconds), which may cause difficulties in practical implementation, especially when there are limitations in computational resources.

In total:

- EfficientNet_v2 + Transformer is a more resource-efficient model suitable for environments with limited computational power but compromises performance in text generation quality.
- CLIPCap excels in generating accurate and semantically meaningful text but requires more training time and computational resources, making it ideal for high-quality applications where resources are less constrained.

6.2 Future Directions

Trainer Time Optimization: Investigate techniques such as pruning, knowledge distillation, or architecture transformation to reduce training time without sacrificing performance.

Variations in loss functions: Experiment with loss functions such as Focus Loss, Label Smoothing Loss, or custom ones to improve performance, especially for imbalanced or complex data.

Data Augmentation: Apply techniques such as cropping, rotating, or adding noise to images to improve performance, preferably with small batch models like CLIPCap.

Hyperparameter Tuning: Prioritize learning rate, scheduler, or patience to improve trainer training.

Additional research on early stopping: Prioritize stopping or extend long training times (especially CLIPCap) to increase overall performance.

Tài liệu

[1] O.Vinyals, A.Toshev, S.Bengio, D.Erhan, *"Show and tell: A neural image caption generator"*, CVPR, 2015.

[2] A.Karpathy & L.Fei-Fei, *"Deep visual-semantic alignments for generating image descriptions"*, CVPR, 2015.

[3] J.Mao, W.Xu, Y.Yang, J.Wang, Z.Huang, and A.Yuille, *"Deep Captioning with Multimodal Recurrent Neural Networks (m-RNN)"*, in ICLR, 2015

[4] J.Donahue, L.Anne Hendricks, S.Guadarrama, M.Rohrbach, S.Venugopalan, K.Saenko, and T.Darrell, *"Long-term recurrent convolutional networks for visual recognition and description"* in CVPR, 2015.

[5] S.J.Rennie, E.Marcheret, Y.Mroueh, J.Ross, and V.Goel, *"Self-critical sequence training for image captioning"* in CVPR, 2017.

[6] B.Dai, D.Ye, and D.Lin, *"Rethinking the form of latent states in image captioning"*, in ECCV, 2018.

[7] S.Chen & Q.Zhao, *"Boosted attention: Leveraging human attention for image captioning"*, ECCV, 2018.

[8] P.Anderson, X.He, C.Buehler, D.Teney, M.Johnson, S.Gould, and L.Zhang, *"Bottom-up and top-down attention for image captioning and visual question answering"* in CVPR, 2018.

[9] T.Yao, Y.Pan, Y.Li, and T.Mei, *"Exploring Visual Relationship for Image Captioning"* in ECCV, 2018.

[10] L.Guo, J.Liu, J.Tang, J.Li, W.Luo, and H.Lu, *"Aligning linguistic words and visual semantic units for image captioning"* in ACM Multimedia, 2019.

[11] X.Yang, K.Tang, H.Zhang, & J.Cai, *"Auto-Encoding Scene Graphs for Image Captioning"*, CVPR, 2019.

[12] Z.Shi, X.Zhou, X.Qiu, & X.Zhu, *"Improving Image Captioning with Better Use of Captions"*, ACL, 2020.

[13] T.Yao, Y.Pan, Y.Li, and T.Mei, *"Hierarchy Parsing for Image Captioning"* in ICCV, 2019.

[14] A.Vaswani, N.Shazeer, N.Parmar, J.Uszkoreit, L.Jones, A.N.Gomez, L.Kaiser, and I.Polosukhin, *"Attention is all you need"*, in NeurIPS, 2017.

[15] X.Yang, H.Zhang, & J.Cai, *"Learning to Collocate Neural Modules for Image Captioning,"*, ICCV, 2019.

[16] G.Li, L.Zhu, P.Liu, and Y.Yang, *"Entangled Transformer for Image Captioning"*, in ICCV, 2019

[17] S.Herdade, A.Kappeler, K.Boakye, and J.Soares, *"Image Captioning: Transforming Objects into Words"*, in NeurIPS, 2019.

[18] L.Guo, J.Liu, X.Zhu, P.Yao, S.Lu, and H.Lu, *"Normalized and Geometry-Aware Self-Attention Network for Image Captioning"*, in CVPR, 2020.

[19] S.He, W.Liao, H.R.Tavakoli, M.Yang, B.Rosenhahn, and N.Pugeault, *"Image captioning through image transformer"* in ACCV, 2020.

[20] L.Huang, W.Wang, J.Chen, & X.-Y.Wei, *"Attention on Attention for Image Captioning"*, ICCV, 2019.

[21] Y.Pan, T.Yao, Y.Li, and T.Mei, *"X-Linear Attention Networks for Image Captioning"*, in CVPR, 2020.

[22] M.Cornia, M.Stefanini, L.Baraldi, and R.Cucchiara, *"Meshed-Memory Transformer for Image Captioning"*, in CVPR, 2020.

[23] J.Ji, Y.Luo, X.Sun, F.Chen, G.Luo, Y.Wu, Y.Gao, and R.Ji, *"Improving Image Captioning by Leveraging Intra- and Inter-layer Global Representation in Transformer Network"*, in AAAI, 2021.

[24] Y.Luo, J.Ji, X.Sun, L.Cao, Y.Wu, F.Huang, C.-W.Lin, and R.Ji, *"Dual-Level Collaborative Transformer for Image Captioning"*, in AAAI, 2021.

[25] F.Liu, Y.Liu, X.Ren, X.He, and X.Sun, *"Aligning visual regions and textual concepts for semantic-grounded image representations"*, in NeurIPS, 2019.

[26] X.Zhang, X.Sun, Y.Luo, J.Ji, Y.Zhou, Y.Wu, F.Huang, and R.Ji, *"RSTNet: Captioning with Adaptive Attention on Visual and Non-Visual Words"*, in CVPR, 2021.

[27] W.Liu, S.Chen, L.Guo, X.Zhu, & J.Liu, *"CPTR: Full Transformer Network for Image Captioning"*, 2021.

[28] L.Zhou, H.Palangi, L.Zhang, H.Hu, J.J.Corso, and J.Gao, *"Unified Vision-Language Pre-Training for Image Captioning and VQA"*, in AAAI, 2020.

[29] X.Li, X.Yin, C.Li, P.Zhang, X.Hu, L.Zhang, L.Wang, H.Hu, L.Dong, F.Wei et al., *"Oscar: Object-semantics aligned pre-training for vision-language tasks"*, in ECCV, 2020.

[30] P.Zhang, X.Li, X.Hu, J.Yang, L.Zhang, L.Wang, Y.Choi, and J.Gao, *"VinVL: Revisiting visual representations in vision-language models,"*, in CVPR, 2021.

[31] X.Hu, Z.Gan, J.Wang, Z.Yang, Z.Liu, Y.Lu, and L.Wang, *"Scaling Up Vision-Language Pre-training for Image Captioning"*, 2021.

[32] Z.Wang, J.Yu, A.W.Yu, Z.Dai, Y.Tsvetkov, and Y.Cao, *"SimVLM: Simple visual language model pretraining with weak supervision"*, 2021.

[33] M.Cornia, L.Baraldi, G.Fiameni, and R.Cucchiara, *"Universal Captioner: Long-Tail Vision-and-Language Model Training through Content-Style Separation"*, 2021.

[34] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo, *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*, 2021.

[35] Ron Mokady, Amir Hertz, Amit H.Bermano *ClipCap: CLIP Prefix for Image Captioning*, 2021

[36] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, Jifeng Dai, *DEFORMABLE DETR: DEFORMABLE TRANSFORMERS FOR END-TO-END OBJECT DETECTION* , 2020.

[37] Matteo Stefanini, Marcella Cornia, Lorenzo Baraldi, Silvia Cascianelli, Giuseppe Fiameni, and Rita Cucchiara, *From Show to Tell: A Survey on Deep Learning-based Image Captioning*, 2021.

[38] Anh-Cuong Pham, Van-Quang Nguyen, Thi-Hong Vuong, Quang-Thuy Ha, *KTVIC: A VIETNAMESE IMAGE CAPTIONING DATASET ON THE LIFE DOMAIN*, 2024.