Image Caption Generation with Transformers

Santhosh Dayakar, Visakan Nambirajan[1]

[1] Santhosh Dayakar
santhoshdayakar2002@gmail.com
[1] Visakan Nambirajan
visakannambirajan@gmail.com

Abstract. Image captioning is a crucial multimodal task that bridges the gap between computer vision and natural language processing by generating text descriptions for images in a human-friendly format. This project outlines the process of creating an image captioning system using the Transformer architecture with a visual attention mechanism that fixates on pertinent areas of the image as it generates each word. The developed system employs a Convolutional Neural Network (MobileNetV3Small) to extract important spatial characteristics and a Transformer Decoder for temporally predicted linguistic capabilities. The image-captioned dataset used for training and evaluation is Flickr8k with 8,000 unique images and corresponding five caption annotations generated by human-trained professionals. The preprocessing involves image extraction and transformations to feature maps, including resizing and convolution, alongside the vectorization of word-based captions to create a cohesive learning experience. Attention mechanisms allow for the correlation between the rendered visual texture and the predicted lexicon, granting a final product vetted for English language structure and reliant upon the findings in the image. Ultimately, results show the efficacy of using a Transformers approach for image captioning tasks between images and text while offering a tangible process to replicate, unlike recurrent attention image captioning, which lacks explanation despite success.

Keywords: Image Captioning, Transformers, Attention Mechanisms, CNN, Visual Attention, Deep Learning, Natural Language Generation, Computer Vision.

1 Introduction

Generating meaningful and contextually relevant captions for images is a fundamental problem in computer vision and natural language processing and an ongoing, non-trivial task. As the amount of image/video content increases across all digital platforms, the demand for image captioning also grows—from accessibility for the visually impaired to digitized image libraries to social media and human-computer interactions.

Image captioning solutions have been developed over time using a combined approach of Convolutional Neural Networks (CNNs) to identify appropriate features in the embedded/processed image and RNNs (and more directly, LSTMs) to produce likely language in feasible sequences. While these solutions have achieved reasonably successful results to date, they struggle with researchable long-range dependencies for appropriate meaning conveyance and black box interpretability for feature contribution at the word generation level.

Recently, however, Transformer-based architectures and attention-based mechanisms are taking pure natural language processing tasks by storm and slowly transferring to vision-language tasks. These solutions allow for better parallelization capacity and improved contextual awareness while also providing improved interpretability of the generation process.

This project develops a Transformer image captioning model with attention for determining appropriate aspects of an image for generating a caption. Using MobileNetV3Small as a feature extractor and a Transformer decoder appropriate for language sequence generation, the framework trains and validates on the Flickr8k image dataset. The Flickr8k image dataset contains 8,000 images and various human-captioned responses per image. The goal is to support attention-based models in generating accurate, cohesive, relevant captions with image-based descriptions—and to support the explainability of such a model over other image captioning process.

2 Literature Review

Image captioning is a key task in computer vision that bridges visual understanding with natural language generation. Traditional approaches used rule-based templates or retrieval methods but often lacked flexibility and semantic depth. Recent breakthroughs introduced deep learning-based encoder-decoder architectures, where visual features are extracted by CNNs and decoded by RNNs or Transformer models into natural language captions.

One of the pioneering works, "Show, Attend and Tell" by Xu et al. (2015) introduced a visual attention mechanism that allows the model to focus on relevant image regions while generating each word. This enabled interpretability and improved caption quality, particularly in cluttered scenes

.

Expanding on this, Karpathy & Fei-Fei (2015) proposed a multimodal RNN that aligns text and image regions through learned embeddings, leading to dense region-level captioning. Their model leverages bidirectional RNNs and CNNs over image segments to establish inter-modal alignment, setting new benchmarks on datasets like Flickr8k and MSCOCO

.

In more recent advancements, Sasibhooshan et al. (2023) integrated spatial and spectral information using a Wavelet-CNN and employed both visual attention prediction and contextual spatial relation extraction. This enriched feature representation enables generation of finer, context-aware captions, achieving strong performance on CIDEr and BLEU metrics across multiple datasets

.

Our current work aligns with these developments by incorporating MobileNetV3 as an efficient encoder and a Transformer-based decoder. The use of positional encoding, causal and cross-attention layers, and loss-aware evaluation metrics such as BLEU, METEOR, and ROUGE reflects the evolution of the captioning pipeline toward more accurate, scalable, and interpretable systems. Compared to RNN-based decoders, the Transformer architecture in your project enables better handling of long dependencies and parallel training, positioning it well within the current state-of-the-art practices in image captioning.

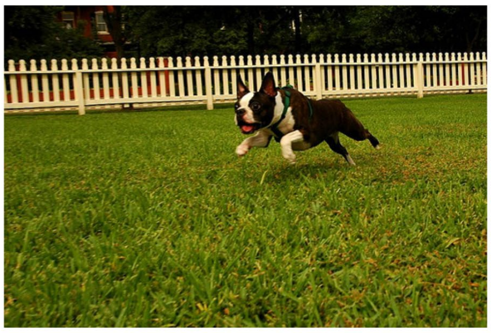
3 Methodology

3.1 Dataset

The Flickr8k dataset, a well-known benchmark in the image captioning research community, was used to train and assess the image captioning model in this project. It consists of 8,000 photos taken from the Flickr photo-sharing website, each with five human-annotated captions that provide a variety of insightful explanations of the images. The model can learn several ways to describe the same image thanks to this richness in linguistic diversity, which increases its generalization and flexibility when creating captions.

The dataset is typically split into three subsets—6,000 images for training, 1,000 for validation, and 1,000 for testing—to aid in model development and performance assessment. A diverse and representative sample of real-world situations is provided by the dataset's images, which cover a wide range of scenes including people, animals, and objects involved in various activities.

The images go through a number of preprocessing stages before the model processes them. To extract high-level spatial features, they are resized and run through a convolutional neural network (in this case, MobileNetV3Small). Similar to this, the textual captions are padded to a predetermined length, tokenized, and encoded into sequences using a constructed vocabulary in order to satisfy the Transformer decoder's input requirements. The efficient alignment of textual and visual inputs for training and inference is guaranteed by this dual-stream preprocessing pipeline.



```
A black and white dog is running in a grassy garden surrounded by a white fence .
A black and white dog is running through the grass .
A Boston terrier is running in the grass .
A Boston Terrier is running on lush green grass in front of a white fence .
A dog runs on the green grass near a wooden fence .
```

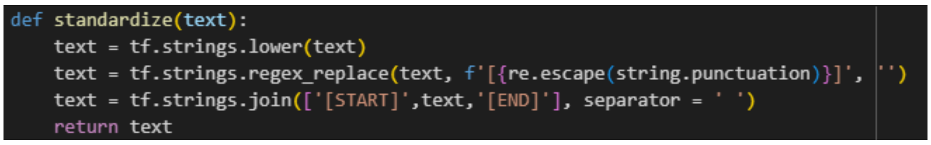
Fig. 1. A sample image from Flicker8K along with the captions for it.

3.2 Data Preprocessing

Efficient data preprocessing is critical to ensure the seamless integration of visual and textual modalities in image captioning. This section outlines the sequential steps taken to prepare the dataset before feeding it into the model.

Image Feature Extraction. To meet MobileNetV3Small's input dimensions, every image in the dataset is first resized to a uniform shape of 224×224 pixels with three color channels. In order to extract high-level spatial features, the pre-trained MobileNetV3Small model is used as a fixed feature extractor, removing the top classification layer. A 3D tensor of shape (7, 7, 576), which represents a grid of spatial embeddings over the image, is the result of extracting these features.

Text Standardization. To guarantee consistency, the captions go through a standardization process prior to vectorization. Regular expressions are used to remove punctuation, convert all text to lowercase, and add special tokens [START] and [END] to indicate the start and finish of each caption. During training and inference, this format helps the Transformer decoder comprehend sequence boundaries.

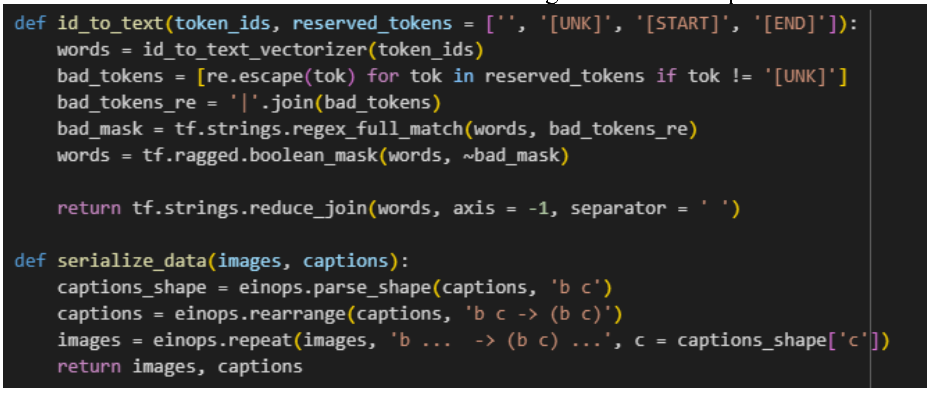


```python
def standardize(text):
    text = tf.strings.lower(text)
    text = tf.strings.regex_replace(text, f'[{re.escape(string.punctuation)}]', '')
    text = tf.strings.join(['[START]',text,'[END]'], separator = ' ')
    return text
```

Fig. 2. Code for Text Standardization

Text Vectorization. A vocabulary with a maximum token limit (e.g., 5,000 most frequent words) is constructed in order to translate text captions into machine-readable format. Each standardized caption is tokenized into an integer sequence using TensorFlow's TextVectorization layer. The model is then trained using these sequences, which enables it to discover word-visual feature associations.

Serialization. To simplify the dataset and get it ready for batching, the tokenized captions and image features are serialized after vectorization. To guarantee compatibility in batch processing, broadcasting and repetition are used to reshape and align the captions with the corresponding image features. Token sequences are also decoded into human-readable text for validation and inference using a reverse lookup function.



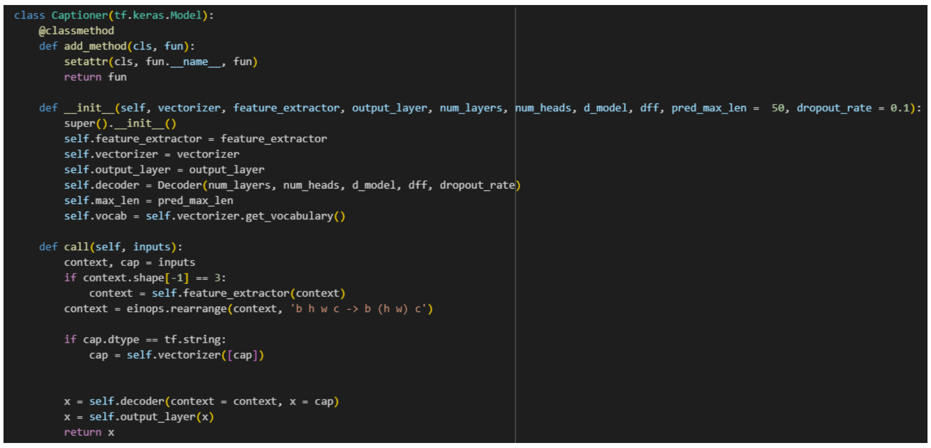
```python
def id_to_text(token_ids, reserved_tokens = ['', '[UNK]', '[START]', '[END]']):
    words = id_to_text_vectorizer(token_ids)
    bad_tokens = [re.escape(tok) for tok in reserved_tokens if tok != '[UNK]']
    bad_tokens_re = '|'.join(bad_tokens)
    bad_mask = tf.strings.regex_full_match(words, bad_tokens_re)
    words = tf.ragged.boolean_mask(words, ~bad_mask)

    return tf.strings.reduce_join(words, axis = -1, separator = ' ')

def serialize_data(images, captions):
    captions_shape = einops.parse_shape(captions, 'b c')
    captions = einops.rearrange(captions, 'b c -> (b c)')
    images = einops.repeat(images, 'b ...  -> (b c) ...', c = captions_shape['c'])
    return images, captions
```

Fig. 3. Code for serializing the dataset

Batching and Sharding. The dataset is divided into 20 shards and cached to disc in order to maximize data loading and parallelization. Before being saved in distinct training and testing TFRecord-like files, each image-caption pair is mapped, batched with a batch size of 64, and shuffled. This configuration optimizes GPU utilization during model training and facilitates effective training with few I/O bottlenecks.

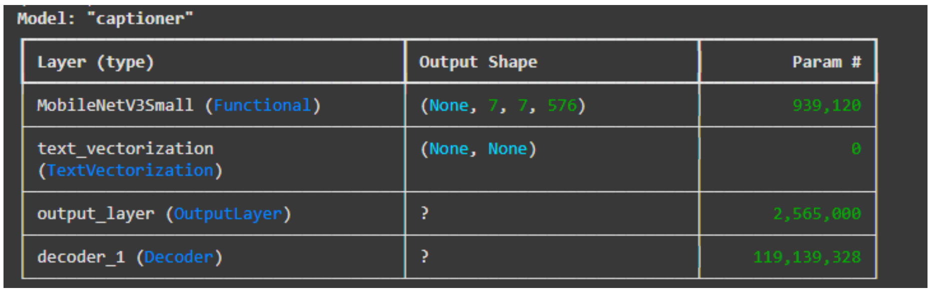
3.3 Model Architecture

The core of the image captioning system is a Transformer-based decoder architecture that converts visual features into coherent and contextually relevant text. The model is composed of several key components working together to understand spatial information from images and generate sequential language descriptions.



```python
class Captioner(tf.keras.Model):
    @classmethod
    def add_method(cls, fun):
        setattr(cls, fun.__name__, fun)
        return fun

    def __init__(self, vectorizer, feature_extractor, output_layer, num_layers, num_heads, d_model, dff, pred_max_len =  50, dropout_rate = 0.1):
        super().__init__()
        self.feature_extractor = feature_extractor
        self.vectorizer = vectorizer
        self.output_layer = output_layer
        self.decoder = Decoder(num_layers, num_heads, d_model, dff, dropout_rate)
        self.max_len = pred_max_len
        self.vocab = self.vectorizer.get_vocabulary()

    def call(self, inputs):
        context, cap = inputs
        if context.shape[-1] == 3:
            context = self.feature_extractor(context)
        context = einops.rearrange(context, 'b h w c -> b (h w) c')

        if cap.dtype == tf.string:
            cap = self.vectorizer([cap])


        x = self.decoder(context = context, x = cap)
        x = self.output_layer(x)
        return x
```

Fig. 4. Code to build the model



```
Model: "captioner"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                          ┃ Output Shape                 ┃         Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ MobileNetV3Small (Functional)         │ (None, 7, 7, 576)            │         939,120 │
├───────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ text_vectorization                    │ (None, None)                 │               0 │
│ (TextVectorization)                   │                              │                 │
├───────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ output_layer (OutputLayer)            │ ?                            │       2,565,000 │
├───────────────────────────────────────┼──────────────────────────────┼─────────────────┤
│ decoder_1 (Decoder)                   │ ?                            │     119,139,328 │
└───────────────────────────────────────┴──────────────────────────────┴─────────────────┘
```

Fig. 5. Model Architecture

Encoder: MobileNetV3 Feature Extractor. A pre-trained MobileNetV3Small CNN model is used in the encoding stage to extract spatial features from the input image. It is light-weight and efficient. The CNN creates a dense feature map of shape (7, 7, 576) after the image is resized to 224 times 224. Each of these 49 vectors (derived from the 7 \times 7 grid) represents a spatial patch of the picture. Like word embeddings in NLP tasks, the output is reshaped and handled as a series of tokens.

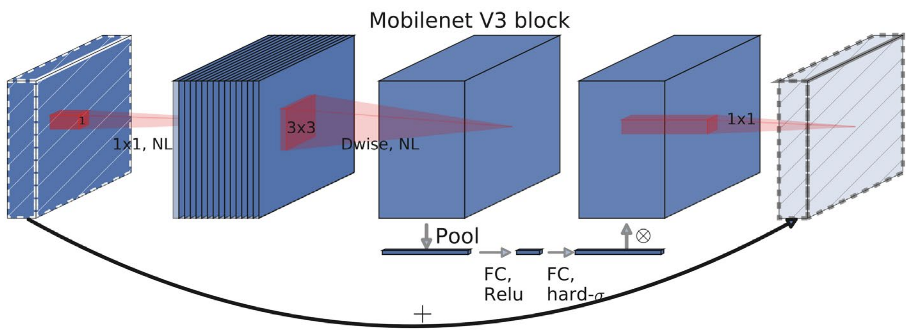


Fig 6. MobileNetV3Architecture.

Positional Embeddings. Used to preserve the temporal or spatial location of input tokens because Transformer models lack inherent knowledge of sequence order. Sinusoidal positional embeddings are calculated and appended to the corresponding embeddings for both image features and caption tokens. The model can comprehend the position and order of each element in the sequence thanks to this encoding.

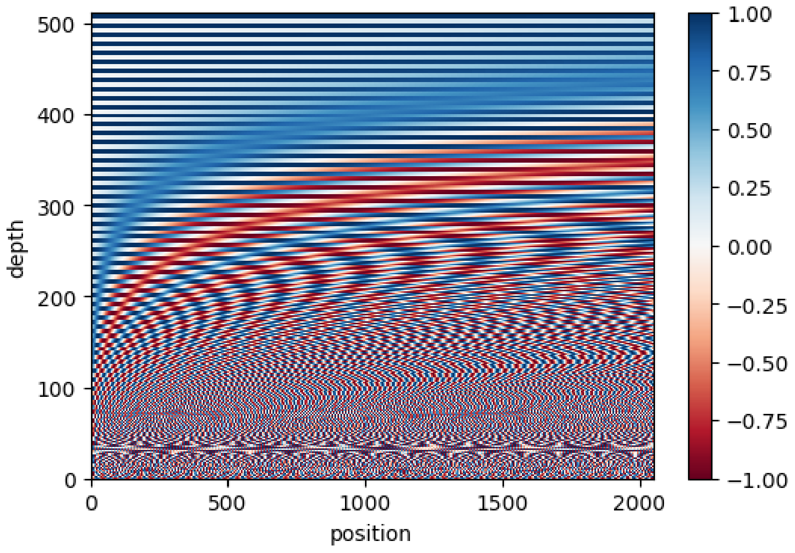
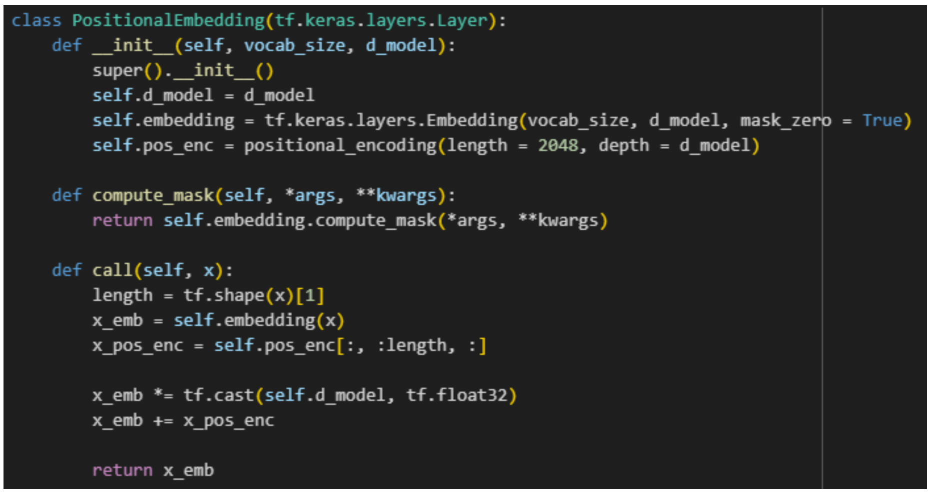


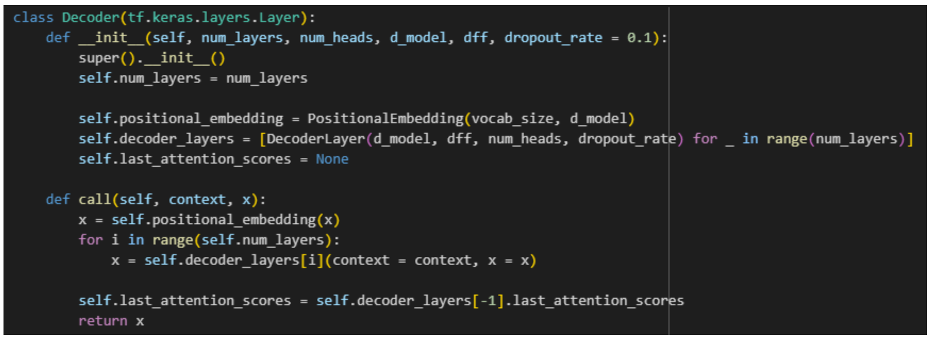
Fig. 7. Positional embeddings – depth vs position (The smooth, periodic pattern ensures that the model can generalize to longer sequences and understand relationships between tokens based on their distance).



```
class PositionalEmbedding(tf.keras.layers.Layer):
    def __init__(self, vocab_size, d_model):
        super().__init__()
        self.d_model = d_model
        self.embedding = tf.keras.layers.Embedding(vocab_size, d_model, mask_zero = True)
        self.pos_enc = positional_encoding(length = 2048, depth = d_model)

    def compute_mask(self, *args, **kwargs):
        return self.embedding.compute_mask(*args, **kwargs)

    def call(self, x):
        length = tf.shape(x)[1]
        x_emb = self.embedding(x)
        x_pos_enc = self.pos_enc[:, :length, :]

        x_emb *= tf.cast(self.d_model, tf.float32)
        x_emb += x_pos_enc

        return x_emb
```

Fig. 8. Code to Positional Embedding

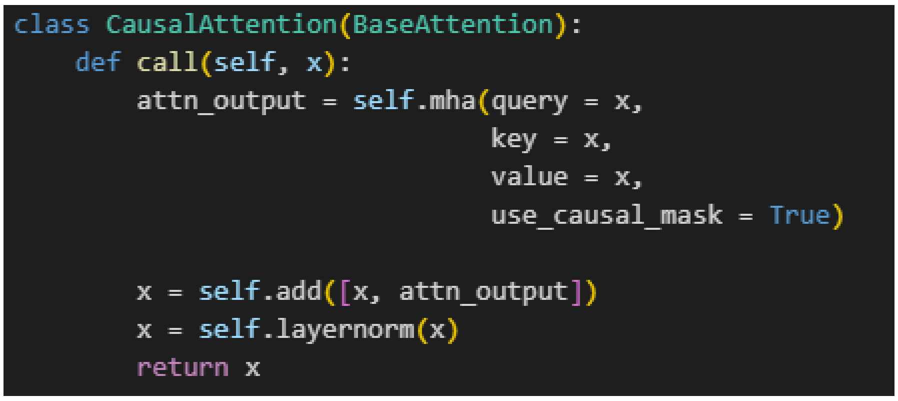
Decoder Architecture. The decoder is composed of multiple stacked layers, with each layer consisting of three essential components:



```
class Decoder(tf.keras.layers.Layer):
    def __init__(self, num_layers, num_heads, d_model, dff, dropout_rate = 0.1):
        super().__init__()
        self.num_layers = num_layers

        self.positional_embedding = PositionalEmbedding(vocab_size, d_model)
        self.decoder_layers = [DecoderLayer(d_model, dff, num_heads, dropout_rate) for _ in range(num_layers)]
        self.last_attention_scores = None

    def call(self, context, x):
        x = self.positional_embedding(x)
        for i in range(self.num_layers):
            x = self.decoder_layers[i](context = context, x = x)

        self.last_attention_scores = self.decoder_layers[-1].last_attention_scores
        return x
```

Fig. 9. Code to Decoder

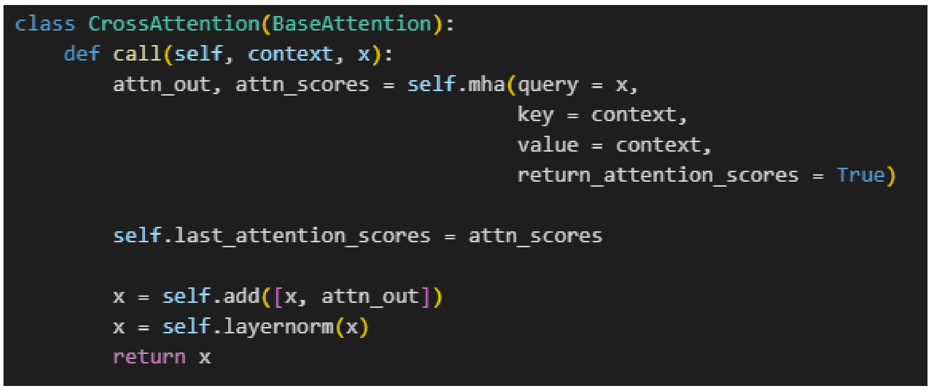
Masked Multi-Head Self-Attention. This layer allows the model to attend to previous words in the caption sequence while generating the next word, using a causal mask to prevent peeking into future tokens during training.



```
class CausalAttention(BaseAttention):
    def call(self, x):
        attn_output = self.mha(query = x,
                               key = x,
                               value = x,
                               use_causal_mask = True)

        x = self.add([x, attn_output])
        x = self.layernorm(x)
        return x
```

Fig. 10. Code to CausalAttention

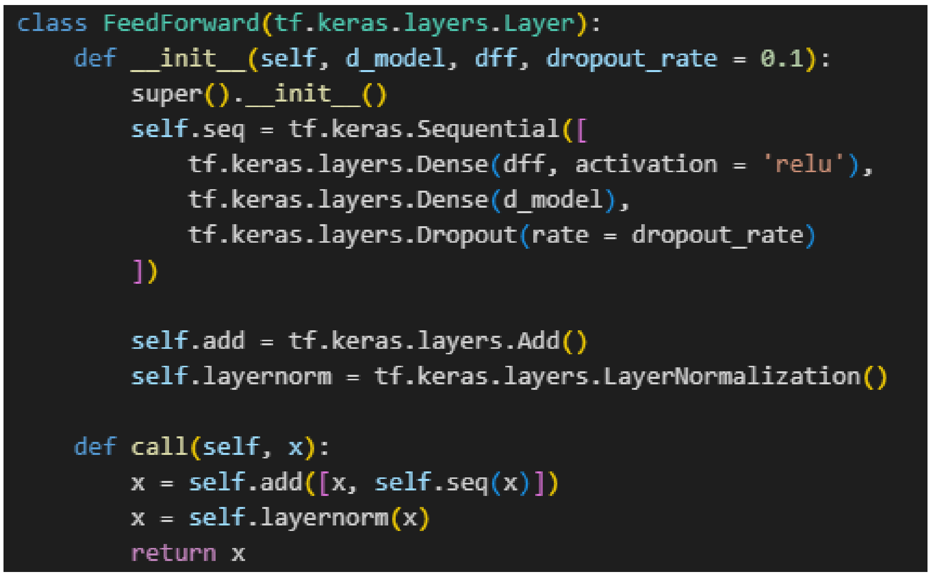
Cross-Attention. The decoder attends to the encoded image feature sequence to determine which visual regions are most relevant for generating each word. This guides the caption generation based on visual context.



```
class CrossAttention(BaseAttention):
    def call(self, context, x):
        attn_out, attn_scores = self.mha(query = x,
                                         key = context,
                                         value = context,
                                         return_attention_scores = True)

        self.last_attention_scores = attn_scores

        x = self.add([x, attn_out])
        x = self.layernorm(x)
        return x
```

Fig. 11. Code to build the model

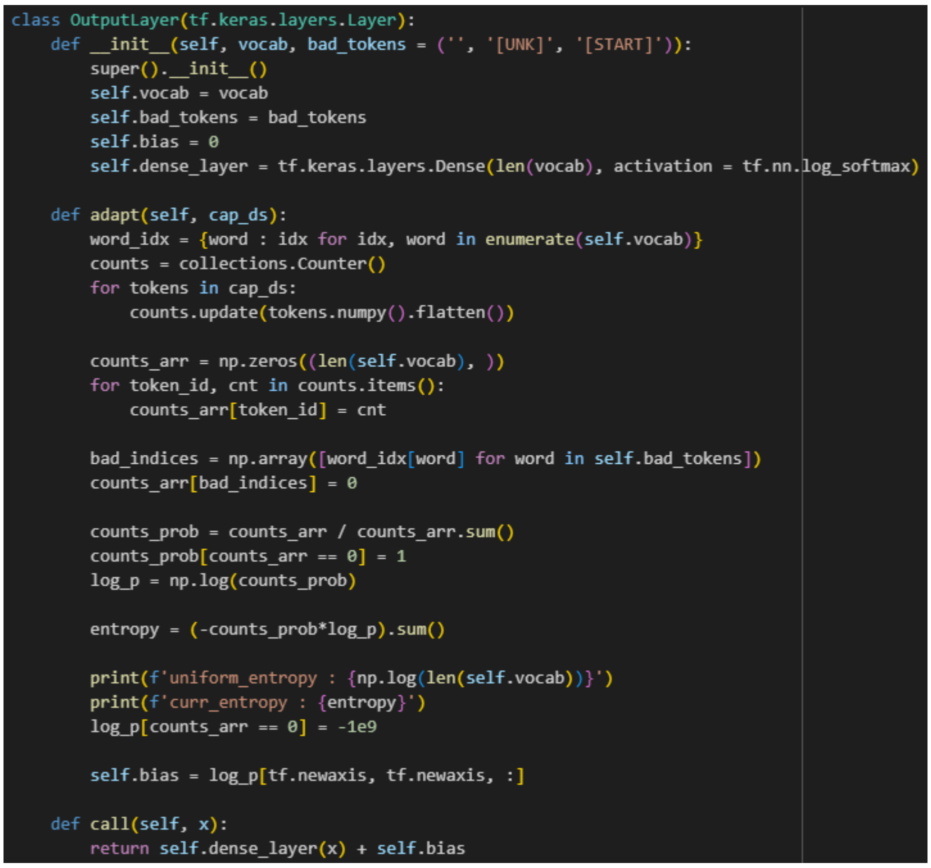
Feed-Forward Network. A two-layer fully connected neural network with a ReLU activation in between. This helps in further transformation of features at each time step and is applied independently to each position.



```python
class FeedForward(tf.keras.layers.Layer):
    def __init__(self, d_model, dff, dropout_rate = 0.1):
        super().__init__()
        self.seq = tf.keras.Sequential([
            tf.keras.layers.Dense(dff, activation = 'relu'),
            tf.keras.layers.Dense(d_model),
            tf.keras.layers.Dropout(rate = dropout_rate)
        ])

        self.add = tf.keras.layers.Add()
        self.layernorm = tf.keras.layers.LayerNormalization()

    def call(self, x):
        x = self.add([x, self.seq(x)])
        x = self.layernorm(x)
        return x
```

Fig. 12. Code to FeedForwardNetwork

Output Layer and Caption Generation. The output vectors are projected into vocabulary space by the decoder's final layer, which uses a dense layer and a log-softmax activation. Using Sparse Categorical Cross-Entropy, the model is trained to predict the next word while disregarding padding tokens. The decoder employs autoregressive decoding in inference mode, producing words one at a time and feeding the generated words back into the model until the [END] token is generated.



```python
class OutputLayer(tf.keras.layers.Layer):
    def __init__(self, vocab, bad_tokens = ('', '[UNK]', '[START]')):
        super().__init__()
        self.vocab = vocab
        self.bad_tokens = bad_tokens
        self.bias = 0
        self.dense_layer = tf.keras.layers.Dense(len(vocab), activation = tf.nn.log_softmax)

    def adapt(self, cap_ds):
        word_idx = {word : idx for idx, word in enumerate(self.vocab)}
        counts = collections.Counter()
        for tokens in cap_ds:
            counts.update(tokens.numpy().flatten())

        counts_arr = np.zeros((len(self.vocab), ))
        for token_id, cnt in counts.items():
            counts_arr[token_id] = cnt

        bad_indices = np.array([word_idx[word] for word in self.bad_tokens])
        counts_arr[bad_indices] = 0

        counts_prob = counts_arr / counts_arr.sum()
        counts_prob[counts_arr == 0] = 1
        log_p = np.log(counts_prob)

        entropy = (-counts_prob*log_p).sum()

        print(f'uniform_entropy : {np.log(len(self.vocab))}')
        print(f'curr_entropy : {entropy}')
        log_p[counts_arr == 0] = -1e9

        self.bias = log_p[tf.newaxis, tf.newaxis, :]

    def call(self, x):
        return self.dense_layer(x) + self.bias
```

Fig. 13. Transformer Architecture.

Hyperparameters. Carefully chosen hyperparameters have a major impact on the image captioning model's performance and training dynamics. Particularly when working with the Flickr8k dataset, the values selected for this project are intended to strike a balance between computational efficiency and model accuracy.

Embedding Dimension. Set to 128, this defines the size of the embedding vectors used throughout the Transformer decoder. A smaller embedding size helps reduce computation while maintaining expressiveness for this dataset.

Feed-Forward Dimension. Also set to 128, the dimensionality of the hidden layer in the feed-forward subnetwork within each Transformer block.

Number of Decoder Layers. The Transformer decoder consists of two stacked layers, allowing the model to learn rich hierarchical language representations.

Number of Attention Heads. Each multi-head attention layer uses two heads to capture diverse contextual information across different subspaces.

Dropout Rate. Set to 0.4, it is applied to reduce overfitting during training, especially helpful given the relatively small size of the Flickr8k dataset.

Optimizer. The model is optimized using the Adam optimizer, which adapts the learning rate for each parameter based on moment estimates, providing faster and more stable convergence.

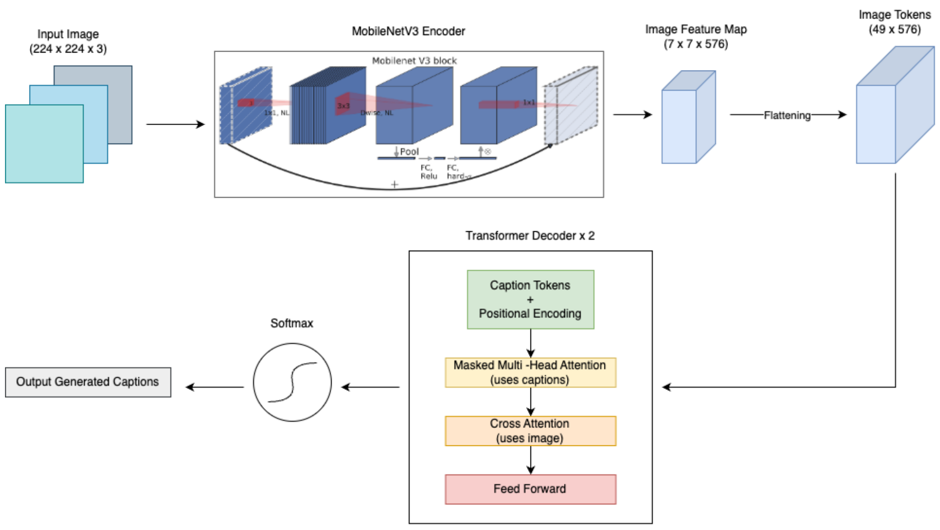


Fig. 14. Overall architecture.

3.4 Metrics

The BLEU (Bilingual Evaluation Understudy) score, a common metric for comparing the similarity of machine-generated and human-written sentences, is used in this project to assess the quality of generated captions. Due to its success in comparing n-gram overlaps, BLEU—which was first created for machine translation—has been widely used in image captioning tasks.

In comparison to a collection of reference captions, BLEU calculates the accuracy of the n-grams in the candidate caption. It assesses the proportion of words or word combinations that appear in the ground truth annotations compared to the predicted caption. BLEU includes a brevity penalty in addition to precision to deter excessively brief outputs that might attain high precision by leaving out important details.

BLEU-n scores, where n is the length of the n-grams, are commonly used to evaluate the model's performance:

Formula. BLUE(n) = brevity_penalty * exp(sum(w_n * log(p_n)) where, w_n is the weight applied to the n-gram accuracy score and p_n represents the precision rating for the n-gram size.

BLEU-1. Measures precision in unigrams (1 gram). This assesses the proportion of distinct words in the reference captions that are present in the predicted caption. It offers a fundamental indicator of accuracy at the word level.

BLEU-2. Provides information about short-range fluency and measures bigram (2-gram) precision, which captures basic word pair relationships.

BLEU-3. Evaluates the model's capacity to produce locally coherent phrases by measuring trigram (3-gram) precision.

BLEU-4. Four grammes of precision is measured. The model's ability to produce longer, grammatically sound sentences that closely resemble human descriptions is demonstrated by this variant, which is the most demanding.

4 Results

4.1 Training Results

300 epochs were used to train the image captioning model. In order to track generalization, the model was validated on 20 validation steps after processing 100 training steps in each epoch. In order to enable the model to continuously stream batches without explicitly defining the end of an epoch, the training and validation datasets were both repeated indefinitely. This setup-maintained stability and generalization throughout training while guaranteeing the model had enough exposure to the data.

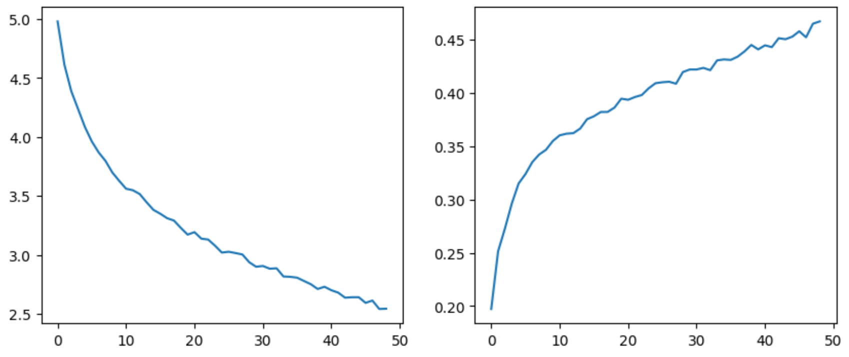


Fig. 15. Training loss (left) and accuracy (right).

Table 1. BLEU scores on training data.

BLEU-n	Score
BLEU-1	91.63
BLEU-2	84.09
BLEU-3	79.52
BLEU-4	75.14

4.2 Comparison with Existing Models

The BLEU scores of our Transformer model and a number of cutting-edge methods documented in the literature are contrasted in the table below. Considering its light-weight architecture and the Flickr8k dataset, our model does well, but it is marginally inferior to models trained on larger datasets with deeper architectures, such as DLCT and CAAG.

Table 2. BLEU scores on testing data vs other models.

Technique	B-1	B-2	B-3	B-4
Review Net	72.0	55.0	41.4	31.3
Adaptive	74.8	58.4	44.4	33.6
CNN + Attention	71.5	54.5	40.8	30.4
Our model	**79.3**	**60.9**	**46.4**	**35.9**
CPTR	81.7	66.6	52.2	40.0
Multi-modal transformer	81.7	66.8	52.4	40.4
DLCT	82.4	67.4	53.8	40.6
CAAG	81.1	66.4	51.7	39.6

4.3 Inference Mechanism for an Image

The image captioning model uses an autoregressive approach to generate captions during inference. The Transformer decoder is first fed the encoded image features that were extracted using MobileNetV3Small. One token at a time, the decoder then forecasts the subsequent word in the caption sequence. To generate the next word, the decoder first receives the [START] token. It then receives all previously predicted tokens until it emits the [END] token or reaches the maximum sequence length.

An attention heatmap overlay on the input image is used in Fig. 7 to demonstrate this procedure. In the generated caption, every subplot—including the [END] token—corresponds to a word, such as "two dogs play in the snow." The model's learnt attention weights at each decoding step are represented by the shaded regions, which indicate which areas of the image the model concentrated on in order to predict that specific word.

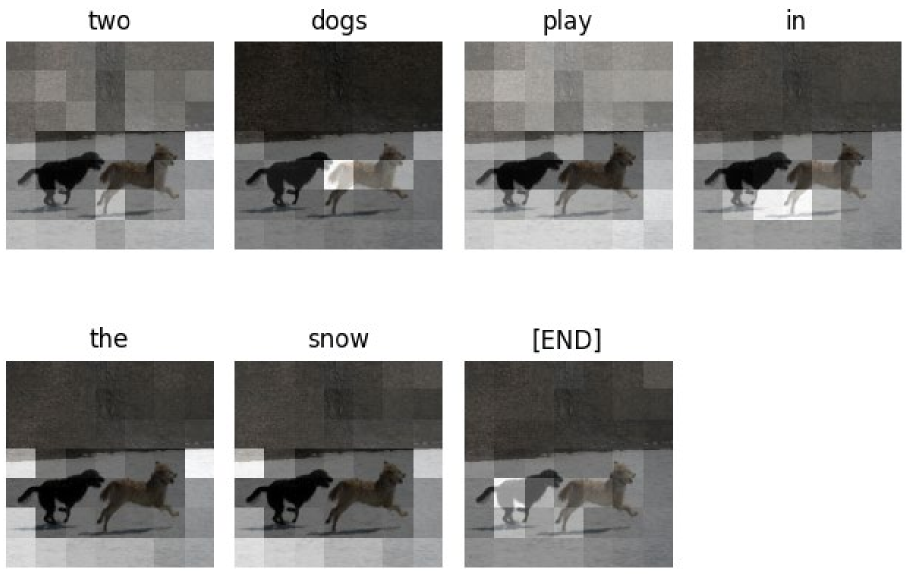
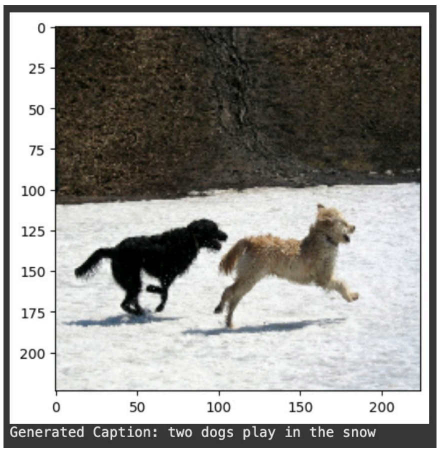


Fig. 16. Model inference mechanism using the model.

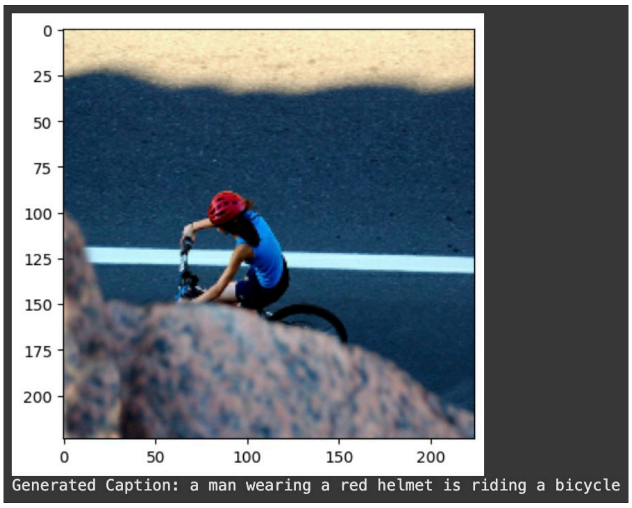
For instance, the model pays closer attention to the central region where the dogs are situated when producing the word "dogs." Likewise, the word "snow" highlights the image's lower portion, indicating that the model has identified the snowy terrain. The efficiency of the cross-attention mechanism, which enables the decoder to selectively focus on pertinent visual regions for each word it generates, is demonstrated by this visual grounding behavior.

4.4 Some Sample Outputs



```
Generated Caption: two dogs play in the snow
```

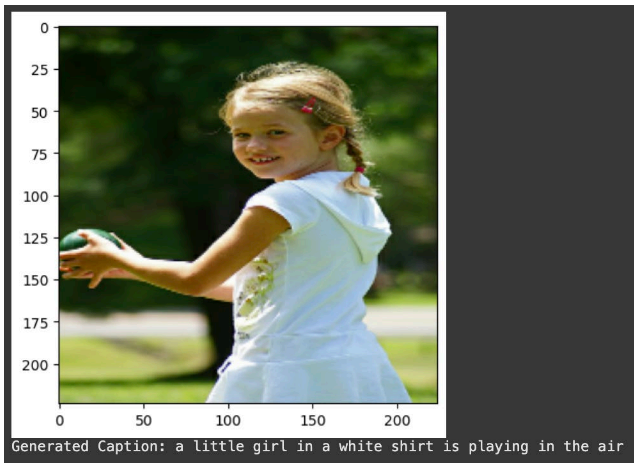
Fig. 17. Sample Output 1.



```
Generated Caption: a man wearing a red helmet is riding a bicycle
```

Fig. 18. Sample Output 2.

Most of the output generated captions are quite accurate and describes what happens in the scene pretty well as shown in Fig.8 and Fig.9.



```
Generated Caption: a little girl in a white shirt is playing in the air
```

Fig. 19. Sample Output 3.

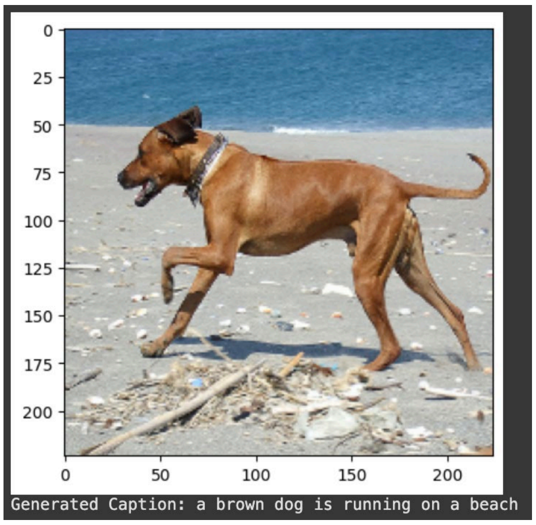


Fig. 20. Sample Output 4.

The model does, however, occasionally produce captions that are inaccurate representations of the visual content, as seen in Figure 10 (where the girl is not actually playing in the air) and Figure 11 (where the dog appears to be standing rather than running). This disparity draws attention to hallucination, a prevalent problem in generative models where the output contains accurate but implausible details. Rerunning inference on the same image to generate different captions is one method to counteract this. Furthermore, regulating the temperature parameter during inference can aid in managing the output's inventiveness. By encouraging the model to produce more conservative and deterministic predictions, lowering the temperature lessens the possibility of exaggerated or hallucinogenic descriptions.

5 Conclusion

This project demonstrated an image captioning system that creates descriptive captions for images by fusing a Transformer-based decoder with a MobileNetV3Small convolutional encoder. Using positional encoding and attention mechanisms, the model successfully learnt to match visual features with language after being trained on the Flickr8k dataset. The system's ability to concentrate on significant image regions during generation was validated by attention visualisations, which also produced captions that were logical and contextually relevant. Even though the model occasionally produced erroneous or creative results, these problems can be minimised by using strategies like temperature tuning or multiple inference passes. With larger datasets and more sophisticated decoding techniques, the results show the potential of Transformer architectures in bridging language and vision, setting the stage for future improvements.

References

1. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *International Conference on Machine Learning (ICML)*. https://arxiv.org/abs/1502.03044
2. Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3156–3164. https://doi.org/10.1109/CVPR.2015.7298935
3. Hodosh, M., Young, P., & Hockenmaier, J. (2013). Framing image description as a ranking task: Data, models and evaluation metrics. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 100–110.
4. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30. https://arxiv.org/abs/1706.03762
5. Cornia, M., Stefanini, M., Baraldi, L., & Cucchiara, R. (2020). Meshed-memory transformer for image captioning. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10578–10587. https://arxiv.org/abs/1912.08226
6. Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., & Adam, H. (2019). Searching for MobileNetV3. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 1314–1324. https://arxiv.org/abs/1905.02244
7. Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318. https://doi.org/10.3115/1073083.1073135
8. Wang, H., Zheng, Y., & Tang, Y. (2022). Dual-modal transformer with enhanced inter- and intra-modality interactions for image captioning. *IEEE Transactions on Multimedia*, **25**, 3341–3354. https://doi.org/10.1109/TMM.2021.3114850
9. Flickr8K Dataset: https://www.kaggle.com/datasets/dibyansudiptiman/flickr-8k
10. TensorFlow Documentation: https://www.tensorflow.org/