Automatic Bengali Image Captioning using EfficientNet-Transformer Network and Vision Transformer

by

Muhammad Khubayeeb Kabir
19101168
Anindita Labonno
19101149
Sofia Amin
19101232
Fariha Tahsin
19101170

A thesis submitted to the Department of Computer Science and Engineering
in partial fulfillment of the requirements for the degree of
B.Sc. in Computer Science

Department of Computer Science and Engineering
School of Data and Sciences
Brac University
January 2023

© 2023. Brac University
All rights reserved.

Declaration

It is hereby declared that

1. The thesis submitted is my/our own original work while completing degree at Brac University.
2. The thesis does not contain material previously published or written by a third party, except where this is appropriately cited through full and accurate referencing.
3. The thesis does not contain material which has been accepted, or submitted, for any other degree or diploma at a university or other institution.
4. We have acknowledged all main sources of help.

Student's Full Name & Signature:

[illegible]	Anindita
Muhammad Khubayeeb Kabir 19101168	Anindita Labonno 19101149
Sofia	Fariha
Sofia Amin 19101232	Fariha Tahsin 19101170

Approval

The thesis/project titled "Automatic Bengali Image Captioning using EfficientNet-Transformer Network and Vision Transformer" submitted by

1. Muhammad Khubayeeb Kabir (19101168)
2. Anindita Labonno (19101149)
3. Sofia Amin (19101232)
4. Fariha Tahsin (19101170)

Of Fall, 2022 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of B.Sc. in Computer Science on January 17,2023.

Examining Committee:

Supervisor:
(Member)

[illegible]

Md. Khalilur Rahman
Associate Professor
Department of Computer Science and Engineering
Brac University

Co-Supervisor:
(Member)

[illegible]

Moin Mostakim
Senior Lecturer
Department of Computer Science and Engineering
Brac University

Thesis Coordinator:
(Member)

Md. Golam Rabiul Alam, PhD
Professor, Thesis Coordinator
Department of Computer Science and Engineering
Brac University

Head of Department:
(Chair)

Sadia Hamid Kazi, PhD
Chairperson and Associate Professor
Department of Computer Science and Engineering
Brac University

Abstract

The task of image captioning is a complex process that involves generating textual descriptions for images. This technology is extremely beneficial for a wide range of applications, such as assisting people with visual impairments, monitoring surveillance systems, content generation, image indexing, and automatic annotation of images for producing data for training AI-based image generation models. Much of the research done in this particular domain, especially using transformer models, has been focused on English language. However, there has been relatively little research dedicated to the context of the Bengali language. This study addresses the lack of research in the context of Bengali language and proposes a novel approach to automatic image captioning that involves a multi-modal, transformer-based, end-to-end model with an encoder-decoder architecture. Our approach utilizes pre-trained EfficientNet Transformer Network. To evaluate the effectiveness of our approach, we compare our model with a Vision Transformer that utilizes a non-convolutional encoder pre-trained on ImageNet.The two models were tested on the BanglaLekhaImageCaptions dataset and evaluated using BLEU metrics.

Keywords: Image Captioning; Image Encoders; EfficientNet; Vision Transformer; BanglaLekhaImageCaptions; BLEU; Transformer Architecture

Acknowledgement

We would like to begin by giving thanks to Allah for allowing us to complete our thesis without any major disruptions. We are also deeply grateful to our supervisors, Dr. Md. Khalilur Rahman and Mr. Moin Mostakim, for their guidance, feedback, support, and encouragement throughout the process. Lastly, we would like to thank our parents for their support and prayers, which have brought us to the brink of graduation.

Table of Contents

List of Figures

Nomenclature

The next list describes several symbols & abbreviation that will be later used within the body of the document

$BERT$ Bidirectional Encoder Representations from Transformers

$BLEU$ Bilingual Evaluation Understudy

CNN Convolutional neural network

GCN Graph convolutional network

GRU Gated Recurrent Unit

$LSTM$ Long short term memory

RNN Recurrent neural network

Chapter 1

Introduction

Image Captioning is the process of automatically generating, syntactically and grammatically appropriate, textual information from an image. With the advent of deep learning, computer vision, and natural language processing, the task of generating texts from images has become one of the cornerstones for a plethora of applications in this field of research. Past endeavors in neuroscience [1] have revealed a relationship between language and visual processing in the human brain. Likewise, recent advancements in artificial intelligence and computer vision have made deep learning models of various architectures capable of processing images and generating texts. However, this is a rather difficult task since it fuses two of the most important domains in artificial intelligence: Natural Language Processing and Computer Vision.

Some of the applications of Image Captioning involve the inclusion of more accessibility features on the internet to allow visually impaired people to seamlessly interact and engage with visual and multimedia content online; monitor large-scale surveillance systems to avoid undesirable events, and improve AI-based medical assistants for monitoring patients. More importantly, image captioning models, with reliable caption prediction capabilities, can be utilized for the automatic annotation of image data such that an AI-based image generation model can be trained using these large annotated datasets.

Our contributions to this study are the following:

1. **Data Cleaning**: The dataset was full of errors, with a significant portion of the image samples having mismatched captions. These samples were corrected. Both models demonstrated improved captioning capabilities as a result.

2. **Training an EfficientNet Transformer Network**: The model was the best performing one among the two.

3. **Training a Vision Transformer Network**: The model's performance exhibited relatively less capability in predicting the captions. This limitation is a common feature of transformer models when trained on limited dataset sizes.

4. **Understanding the limitations of this field of research**: We analyze the current limitations in Bengali image captioning models in terms of resources and their methodologies.

Chapter 2

Research Problem

Various taxonomy of architectures has previously been explored by researchers, most of which were inspired by an encoder-decoder architecture from neural machine translation [2]. The main objective of these models is to establish a relationship between an image and a sequence of words. Images are encoded using visual encoders, mainly deep CNNs, whereas word or subword sequences are mapped to word vectors. A Recurrent Neural Network (RNN) based, sequence to sequence, generative model is employed as a decoder [3] for generating word sequences based on image features. A simple fully connected layer is also used as a decoder in some models.

Among the different approaches proffered in research, two architectures have been attested to be the most effective in literature: the inject and merge model [4]. The merge model typically aggregates the two encoded feature vectors (image and word) using addition, concatenation, or multiplication. Two different pre-trained feature extractors can be employed, one tasked as an image feature extractor and the other as a text feature extractor to create word embeddings. The word vectors or the word embeddings are then fed to an RNN which encodes the linguistic representations. The image features are then used along with the RNN output to condition the predictions. A fully connected layer is usually employed for this prediction task. Inject models, on the other hand, tend to incorporate the image features somewhat early in the generation process. According to Tanti et al (2018), there are three different variants of this architecture: init-inject, pre-inject, and par-inject. This architecture combines the concerns of the image with each input word to the RNN, with the encoding at the encoder to incorporate linguistic and visual information together. The merge model uses the exact same representation for every word output, whereas the inject model allows the image representation in the hidden state vectors of the RNN to change after each time step. The merge model is superior to the latter due in part to the multimodal nature of its architecture. This helps the merge model avoid generating less stereotyped and generic captions.

Contemporary art, however, revolves around transformer-based architectures. These mostly comprise pre-trained visual and text encoders and a transformer-based decoder. Bidirectional Encoder Representations from Transformers (BERT) models [5], which are often incorporated in language models, are also applied to image captioning as text encoders.

2.1 Image Encoders

Convolutional neural networks (CNN) have been the staple of computer vision tasks for the past decade. They are useful for their ability to capture spatial and temporal dependencies in an image accurately, via convolution operations with relevant filters. In the task of image captioning, an effective representation for the visual encoding pipeline can be achieved using various approaches, and these approaches can be classified into four main categories, three of which are CNN-based: 1. **non-attentive methods** - these are CNNs that focus on the global features of an image. Their goal is to derive high-level, fixed-sized representations from input images. 2. **additive attentive methods** - these are proposed approaches based on additive attention [6] that make up for the shortcomings of the global representations by introducing more granularity along with time-varying visual features in their encodings, thus allowing greater flexibility. Additive attention incorporates the visual content using either grids or regions. 3. **graph-based attention** - these include hand-designed graphs that are placed on top of images to mark regions of interest that are convolved with relevant filters for object instance detection before being proceeded by a graph convolutional network (GCN) [7] . In doing so, the model is able to capture both spatial and semantic relationships between various regions of an image, using a pre-trained classifier and geometric inferences, respectively. Several variants of this GCN architecture were employed in [8] and [9] . 4. **self-attentive methods** - these are transformer-based approaches, as proposed in [10], which utilize the most relevant parts of the input sequence in a flexible manner by a weighted combination of all of the encoded input vectors to obtain refined representations and capture long-term dependencies in input sequences.

2.2 Language models

Language models allow us to predict the next word of a given sequence with a probability. For the task of image captioning, this prediction is explicitly conditioned on the visual representations of the image in concern. In addition, each word that is predicted in the sequence is also conditioned on the previous words in the sequence. The various strategies employed for generative language models include – 1. Long short term memory (LSTM), a superior variant of RNN, is tasked with word prediction at each time step by applying softmax on the hidden states when projected onto a vector space having the same dimension as the vocabulary length, where the hidden state represents the visual encodings [11] . This was further improved by introducing additive attention in [12] by replacing the static global vector representations of images, that reside in the hidden state, with time-varying representations to better align the word and visual features. The LSTM layers can also be stacked using a recurrent convolutional architecture, as proposed in [13] , to capture higher-order relations. 2. CNNs. Convolution operations with kernels on feature vectors are also used for feature extraction and hence can be utilized as a language model by combining the image feature maps with word embeddings as seen on [14] and then feeding the output to a CNN model for which the sequences are right masked. Despite being capable of leveraging parallel training, this approach falls short in terms of performance when compared to its transformer-based counterparts. 3. Transformer-based architectures, which are essentially decoders used as set-to-sequence generative language models, perform self-attention operation on the word embeddings, followed by a cross-attention operation which is finally proceeded by a feed-forward dense layer. The generative attribute of the model is incorporated by introducing a masking mechanism to make the generation of the sequence unidirectional.

2.3 Datasets

The most commonly used dataset in image captioning that is widely used for benchmarking new models and architectures includes the Microsoft COCO [15] , which has over 120,000 training examples annotated with five captions each. Flickr8k, Flickr16k, and Flickr30k are some of the datasets that were previously used for testing, with 8,000, 16,000, and 30,000 training examples respectively.
We have used the BanglaLekhaImageCaptions dataset[16] which we obtained from Mendeley.com[17] and this dataset has been used in [16]. The BanglaLekhaImageCaptions dataset[16] consists of approximately 9154 images with two reference captions per image. Figure 2.1 visualizes a few samples from the dataset.
The dataset presented some challenges, as it contained a significant amount of inaccuracies and inconsistencies that required correction. Additionally, the dataset was limited in size.

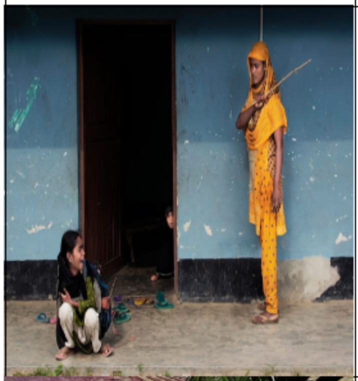
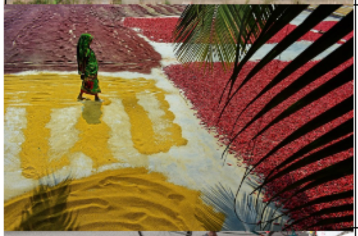
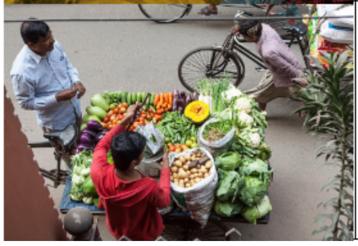
Image	Reference caption
	তিন জন মেয়ে মানুষ আছে। এক জন দাড়িয়ে আছে আর দুই জন বসে আছে। একটি হলুদ জামা পায়জামা পরা মহিলা দাড়িয়ে হাতে একটি বেত নিয়ে পিটানোর ভাব দেখাচ্ছে আর ছোট একটি মেয়ে পিছনে ব্যাগ নিয়ে বসে কাঁদছে।
	এক জন মেয়ে মানুষ মাথায় ঘোমটা দিয়ে কাজ করছে। মাটিতে বিভিন্ন রঙের মসলা আছে। একটি মহিলা হালকা পানির উপরে দাড়িয়ে আছে শারি পরে, মহিলার মুখ ডানদিকে ঘুরানো, পানির রং হলুদ দেখাচ্ছে।
	তিনজন ছেলে মানুষ আছে। একজন মানুষ সবজি বিক্রি করছে। একজন মানুষ রাস্তা দিয়ে রিক্সা নিয়ে যাচ্ছে। ৩ জন পুরুষ যার মাঝে ১ জন সবজী পাল্লা দিয়ে মাপছে তার প্রনে লাল ফুলহাতা গেঞ্জি, ১ জন সবজির ব্যানের কাছে সাদা শার্ট পরে দাড়িয়ে আছে পাশে একজন রিক্সা চালক একটি রিক্সা নিয়ে যাচ্ছে যার পরনে শার্ট, লুঙ্গী।

Figure 2.1: BanglaLekhaImageCaptions Dataset

2.4 Evaluation metrics

Various automatic scoring methods are exercised in literature for evaluation with the most popular one being BLEU [18] (Bilingual Evaluation Understudy), which is also widely used in machine translation tasks to evaluate machine-translated text. BLEU is considered a robust metric as it is language-independent, intuitive, and easy to compute. It is defined by the total number of n-gram occurrences in the reference text divided by the total number of n-grams in the predicted word sequence. An n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be words, phonemes, or other units of speech or text. In BLEU metrics, n-grams are used to compare a machine-generated translation to a human-generated reference translation. The idea is that a good translation should have a high degree of lexical and phrasal overlap with the reference translation. By comparing n-grams of the machine-generated translation to the reference translation, the BLEU metric can estimate the extent to which the two translations match. The n-grams are usually calculated up to a length of four, e.g. - unigram, bigram, trigram, and 4-gram. For example, in the sentence "the cat in the hat," the bigrams are "the cat", "cat in", "in the", and "the hat". Similarly, trigrams would be "the cat in", "cat in the", "in the hat".
The BLEU-1 metric calculates the precision using only unigrams (single words). The BLEU-2 metric calculates the precision using bigrams (pairs of words). The BLEU-3 metric calculates the precision using trigrams (triplets of words), and the BLEU-4 metric calculates the precision using 4 grams (sets of four words). For example, if a translation contains 50 percent unigrams, 30 percent bigrams, 10 percent trigrams, and 10 percent 4-grams that match the reference translation, the BLEU score would be calculated as follows:
BLEU-1 score = 50 percent, BLEU-2 score = 30 percent, BLEU-3 score = 10 percent, BLEU-4 score = 10 percent. The final BLEU score is a weighted average of the individual scores, with higher n-grams given more weight. BLEU scores are commonly used to evaluate the performance of machine translation systems, but they can also be used to evaluate the quality of other types of text generation systems.
METEOR [19] is another evaluation system based on unigram precision and recall. This is calculated as follows

$$Score = Fmean \cdot (1 - Penalty) \tag{2.1}$$

where, Fmean is the weighted recall and precision of the total matching unigram occurrences in the reference text and the predicted text, and the penalty is the cost for a given alignment of a set of unigrams. Another popular metric is CIDEr [20] which is based on cosine similarity between TF-IDF weighted n-gram of the predicted caption and its corresponding reference captions.

Chapter 3

Research Objectives

In this study, we implemented a total of two models that are entirely transformer-based with slight modifications to their architecture. The main challenge we faced was the lack of variety in training examples while working with the BanglaLekhaImageCaptions dataset. The captions were not standardized, meaning annotations of the dataset did not seem to follow a particular scheme i.e. some captions were too verbose whereas, others were too short, and lacked consistency in their ways of describing the pictures. Our workflow can be summarized as follows:

1. Clean and preprocess the available data.
2. Apply augmentations to images.
3. Implement the EfficientNet-Transformer network.
4. Implement the Vision Transformer model.
5. Tune hyperparameters.
6. Evaluate the models using the metrics discussed in the previous section.
7. Understand and analyze the models' limitations.

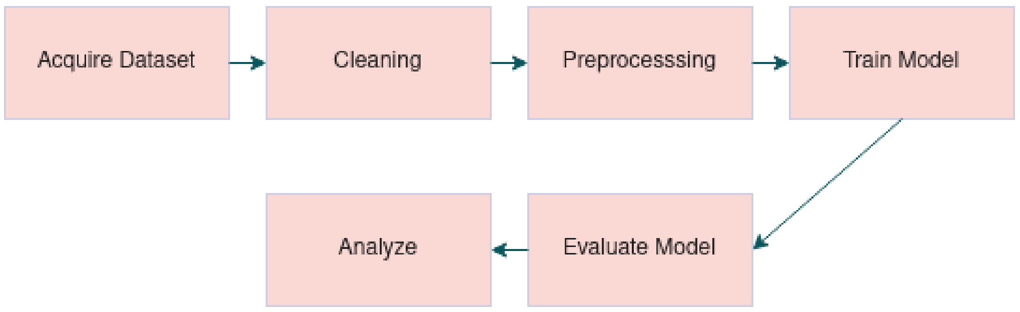


Figure 3.1: A summary of the research workflow.

Chapter 4

Literature Review

This section aims for a detailed review of some of the past research in image captioning in Bengali language and some transformer-based image captioning models in English language. Additionally, we also review some unconventional image captioning model architectures reported in some previous works.

4.1 Bengali

A total of 9 different approaches, to our knowledge, have previously been explored for captioning in Bengali language.

The authors in [21] employed a CNN-LSTM based architecture that pursues the state-of-the-art top-down approach for automatic image captioning. Training examples were subsampled based on a goodness score to achieve better generalization. FastText's [22] pre-trained word embeddings were used for text feature extraction and image features were extracted via InceptionResnet [23] and VGG-16 [24] . Par-inject and merge architectures were adapted to achieve competitive scores.

[25] is the first paper where the authors constructed their very own annotated dataset called 'BanglaLekhaImageCaptions' [17] in Bangla which mostly represents the context of Bengali culture and its people. A pre-trained VGG16 model was used for image feature extraction and encoding. A stacked LSTM layer was tasked to sequentially predict the captions from the encoded image features and the textual features (word embeddings) obtained using an embedding layer.

In [16] a pre-trained VGG16 model was used as a feature extractor. The images in the dataset were mapped to 25 distinct classes of the pre-trained weights. A LSTM layer was used to generate text sequences from the image vectors. The CNN was trained for some epochs and its weights were transferred to the LSTM layer for further training. And finally, the previously trained CNN was fitted on top of the LSTM layer and was trained for a final few epochs to yield results.

[26] proposed a deep learning pipeline that uses two CNN models as its image feature extractors. These include a pre-trained InceptionV3 [27] and Xception [28] on the ImageNet dataset. Local attention mechanisms(or Bahdanau attention) were employed to align the decoder RNN with the relevant input sequences from CNN-

based encoders. A Gated Recurrent Unit(GRU) was used as the RNN decoder.

A noble encoder-decoder architecture was proposed in [29] . A new dataset called BNATURE was constructed and annotated. InceptionV3 was used as the encoder followed by a bidirectional GRU layer as the decoder to generate the captions. Furthermore, the authors used beam search and argmax to refine the quality of the captions generated by the model.

The authors in [30] proposed a multimodal approach to image captioning where the text and image features were extracted using CNN-based encoders. Images were encoded using pre-trained ResNet-50 [31] , and text features were obtained using one-dimensional convolution layers with a Global Max Pooling layer at the end. Both the features were then fed to a fully connected (dense) layer which acts as a simple decoder.

A hybrid encoder-decoder model with two embedding layers side-by-side, along with two other models with a single unit embedding layer, was proposed in [32] . Two word embedding models: fastText and a pre-trained GloVe [33] embedding model were employed as text feature extractors for the reference captions. Likewise, two pre-trained models, InceptionResnetV2 and Xception, both trained on ImageNet, were used separately to encode the images. The image feature vectors and the text feature vectors were concatenated before being fed to a fully connected layer (or the decoder layer) to generate the corresponding captions.

[34] is the first paper (to our knowledge) that employed a transformer model for image captioning in Bengali language. The authors proffered an encoder-decoder architecture that comprised the InceptionV3 model as an image encoder, a transformer encoder block, and a decoder block. This model was compared with a second model which utilized a visual attention-based encoder-decoder architecture that comprises an attention mechanism-infused CNN as an image feature extractor, an embedding layer as a text feature extractor, and a Gated Recurrent Unit (GRU) as a decoder.

The authors in [35] proposed a ResNet-transformer-based encoder-decoder model that employs a pre-trained ResNet-101 model for image feature extraction. A simple word embedding model was used for encoding word sequences. The image feature vectors are passed on to a transformer encoder whereas the word embeddings are fed to the decoder. The image feature vectors are then combined using cross attention operation with the output of the masked self-attention in the decoder.

4.2 English

Conventional encoder-decoder architectures use a CNN encoder for the images but in [36] , the authors proffered a sequence to sequence CNN-free architecture, namely the Vision Transformer [37] , where a noble image encoder pre-trained on ImageNet was utilized as an image encoder. The images were, at first, divided into fixed-sized patches and transformed into 1D patch sequences which are then passed through an embedding layer, followed by a learnable positional embedding layer. The resulting output was then used to perform cross attention operations with the word feature vectors to predict words.

[38] proposed a noble approach to the transformer-based encoder-decoder architecture where a stack of memory-augmented encoding layers was used to encode multi-level visual relationships from the input image, with a priori knowledge. This was followed by a stack of (meshed) decoder layers that read from the output of each encoding layer and sequentially generate the output caption with the intra-modality and the cross-modality interactions modeled via scaled dot product attention.

In [39] a multi-view image encoder architecture that utilized the Faster RCNN [40] architecture with three different backbone CNN models was proposed along with the base transformer encoder and decoder. The image representations from the multi-view encoder output were then passed to the transformer encoder. In addition, the word representations were preprocessed using GloVe word embedding followed by an LSTM layer before being fed to a transformer decoder. The model was tested with both single and multi-view encoders and evaluated based on appropriate caption metrics.

Chapter 5

Methodology

5.1 Neural Networks

Neural networks are composed of layers of interconnected nodes, called artificial neurons or simply "neurons." A neural network typically consists of three types of layers: the input layer, one or more hidden layers, and the output layer. The input layer is responsible for receiving and passing the input data onto the hidden layers. Each neuron in the input layer receives a single element of the input data as its input and passes that input on to the next layer. The hidden layers are responsible for processing the input data and producing intermediate output. Each neuron in a hidden layer receives input from some number of other neurons in the previous layer and uses that input to compute and output a single value. It can be mathematically represented as:

$$output = f(\sum_{i=1}^{n}(weight_i * input_i) + bias)$$

Here, the function f is the activation function, which determines the output of the neuron given the weighted sum of its inputs and the bias term. The weights (w1,w2,.....,wn) are the parameters of the neuron that determine how much each input contributes to the output, and the bias is an additional parameter that allows the neuron to shift the output along the y-axis. The output layer is responsible for producing the final output of the neural network. Like the hidden layers, each neuron in the output layer receives input from some number of neurons in the previous layer and uses that input to compute and output a single value using the same equation as before. The number of neurons in the input layer is determined by the size of the input data, and the number of neurons in the output layer is determined by the number of output classes or the number of output variables that the neural network is intended to predict. The number of hidden layers and the number of neurons in each hidden layer are typically chosen through a process of trial and error, based on the complexity of the problem being solved.

The key steps that are involved in training a neural network are as the following:
Input pipeline and preprocessing: The input pipeline is responsible for loading

the data and getting it into the right format for the neural network to consume. This typically involves loading the data from a file or database, splitting it into training and validation sets, and normalizing or preprocessing the data as necessary. Preprocessing may include steps such as scaling, centering, or changing the data format to fit the model's requirement.
Forward Propagation: Once the data is prepared, the next step is forward propagation, where the input is passed through the network to compute the output. During forward propagation, the input is passed through the different layers of the network, where the weights and biases of the neurons are applied to the input to produce the output. The output is then used to compute the loss.
Backpropagation: After the forward propagation, comes the backpropagation where the error is calculated and propagated back through the network to update the weights. The goal of backpropagation is to adjust the weights of the neurons in the network so that the output produced is closer to the desired output, which reduces the error.
Gradient descent: Backpropagation computes the gradients of the parameters (weights) with respect to the loss function, which describes how much the parameters contribute to the error. The optimizer uses these gradients to update the parameters in a direction that minimizes the error. Gradient descent is a commonly used optimizer, which iteratively updates the parameters of the network by subtracting the gradient of the loss function with respect to the parameters, multiplied by a learning rate.
There are other types of optimizers as well like Stochastic Gradient Descent, Adam, RMSprop etc, and depending on the problem, dataset, and model architecture one may suit better than others. It has been further discussed in section 5.2.

5.1.1 Activation Function

An activation function is a mathematical function that is applied to the output of a neuron in a neural network. It determines whether the neuron should be activated or not, based on whether the output meets certain criteria. Activation functions are used to introduce non-linearity into the network, as most

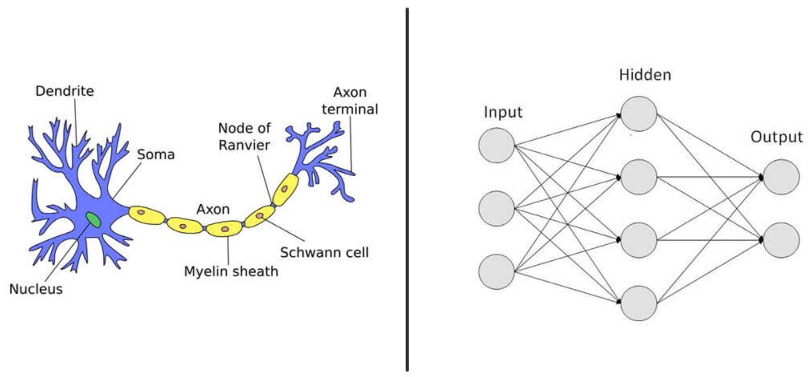


Figure 5.1: The Biological Neuron Graph and the Neural Network

real-world data is non-linear. There are several different types of activation functions that are commonly used in neural networks. Some common activation functions include:
Sigmoid: This is a smooth, S-shaped function that maps any input to a value between 0 and 1. It is often used in the output layer of a binary classification model, as it can be interpreted as a probability. It is mathematically represented as follows:

$$\textit{sigmoid: } f(x) = \frac{1}{1 + e^{-x}}$$

Tanh: This function is similar to the sigmoid function, but it maps the input to a value between -1 and 1. It is often used in the hidden layers of a neural network. Mathematically it is represented as:

$$\textit{tanh: } f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

ReLU (Rectified Linear Unit): This function is a simple linear function that returns the input if it is positive, and returns 0 if it is negative. It is fast to compute and has been shown to work well in practice. Mathematically, it can be represented as follows:

$$ReLU\textit{: } f(x) = max(0, x)$$

Leaky ReLU: This is a variant of the ReLU function that allows a small, non-zero gradient when the input is negative. It can help to prevent the "dying ReLU" problem, where some neurons in the network become inactive and stop learning. Its mathematical representation is as follows:

$$\textit{Leaky } ReLU\textit{: } f(x) = max(0.1x, x)$$

It's important to choose an appropriate activation function for the neural network, as it can significantly impact the model's performance. It's also possible to use different activation functions for different layers in a neural network, depending on the task at hand.

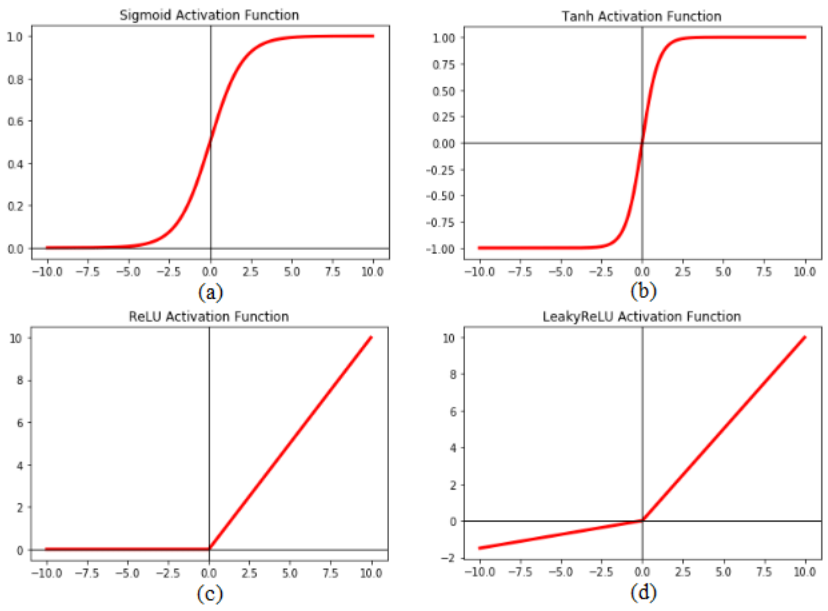


Figure 5.2: Commonly used activation functions in Neural Networks

5.2 Optimizers

Optimizers are algorithms or methods used to adjust the parameters of a machine learning model in order to minimize a loss function. The loss function measures how well the model is able to predict the correct output for a given input, and the goal of the optimization process is to find the set of model parameters that result in the lowest possible loss. There are many different types of optimizers available, and each has its own set of strengths and weaknesses. Some common optimizers include stochastic gradient descent (SGD), Adam, RMSprop, and AdaGrad.
SGD is a simple and widely-used optimizer that updates model parameters by taking the gradient of the loss function with respect to the parameters and moving in the opposite direction.
Adam is a variant of SGD that uses adaptive learning rates and is generally considered to be more efficient and effective.Adam optimizer is efficient as it combines the benefits of both gradient descent and momentum optimization. Gradient descent is a simple optimization algorithm that updates the model parameters by taking a step in the opposite direction of the gradient of the objective function with respect to the parameters. This is a good optimization strategy when the objective function is relatively smooth and the gradient is well-behaved. However, when the objective function has many local minima, or when the gradient is noisy, gradient descent can be slow to converge. Momentum optimization is a variation of gradient descent that tries to alleviate these problems by adding a momentum term to the update rule. The momentum term causes the optimization to continue in the direction of the gradient for a certain number of steps, which can help the optimization escape from local minima and converge faster. Adam combines the ideas of gradient descent and momentum optimization by keeping track of an exponentially decaying average of the gradient, as well as an exponentially decaying average

of the squared gradient. These moving averages are used to update the model parameters, with the idea that the moving averages will give a more accurate estimate of the gradient and its variance, even in the presence of noise. This makes Adam algorithm adapts to the structure of the data and converges faster.
RMSprop is another variant of SGD that uses a moving average of the squared gradient to scale the learning rate, which can help to prevent oscillations and divergence during optimization.
AdaGrad is an optimizer that adaptively adjusts the learning rate for each model parameter based on the historical gradient information, which can help to prevent overfitting and improve convergence.
We have used Adam optimizer because image captioning is a high-dimensional problem where the objective function can be very complex and have many local minima.

5.3 FeedForward Neural Networks

A feedforward neural network is a type of artificial neural network in which the connections between the units do not form a cycle. This means that information flows through the network in only one direction, from the input layer to the output layer, without looping back. In a feedforward neural network, the input data is processed through a series of hidden layers, and finally, the output is produced. Each layer consists of units, also known as neurons, which perform calculations on the data. These calculations are based on the weights and biases associated with each unit, which are adjusted during the training process to optimize the network's performance. Feedforward neural networks are widely used for tasks such as classification, regression, and prediction. They are simple to understand and easy to implement, and they can be trained using a variety of algorithms, such as backpropagation and stochastic gradient descent. However, they are limited in their ability to process temporal or sequential data and can struggle to model complex relationships between variables.

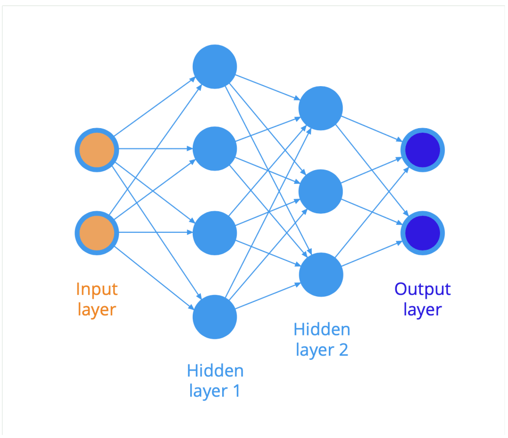


Figure 5.3: Fully-Connected, FeedForward Neural Network

5.4 Layer Normalization

Layer normalization is a technique that is used to scale the input layer of a neural network. It is similar to batch normalization, but rather than normalizing the activations of the batch, it normalizes the activations of the hidden units in a layer. The layer normalization method is designed to overcome the drawback of dependency on batch size. It is not sensitive to the scale of the input data, and hence, stabilizes the hidden units of the network, eventually improving the network's performance. The outcome from the previous layer significantly affects the outcome of the present layer. Therefore the layer normalization statistics are computed over all hidden sections in the same layer to overcome the covariate shifting problem by fixing the mean and variance of the summed input within a layer. The computed layer normalization statistics are as follows:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{5.1}$$

$$\sigma_l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{5.2}$$

Here H is the number of hidden units in a layer and, μ and σ are the normalization terms

5.4.1 Weight re-centering and re-scaling:

Layer normalization is successfully achieved by re-centering and re-scaling the normalization invariant. Unlike other normalization methods, batch normalization and weight normalization, the layer normalization can vary with the individual scaling of the single weight vectors, rather it is invariable to scaling of the whole matrix and a relocation of the inputted weights in the weight matrix. For instance, if there are two sets of model parameters with weight W and W' which varies by a scaling factor δ with addition to all the inputted weights are moved by a constant vector γ it leads to

$$W' = \delta W + 1\gamma^T$$

While implementing layer normalization the two models determine the same outcome:

$$\begin{aligned}\mathbf{h}' = f(\frac{\mathbf{g}}{\sigma'}(W'\mathbf{x} - \mu') + \mathbf{b}) &= f(\frac{\mathbf{g}}{\sigma'}((\delta W + 1\gamma^T)\mathbf{x} - \mu') + \mathbf{b}) \\ &= f(\frac{\mathbf{g}}{\sigma}(W\mathbf{x} - \mu) + \mathbf{b}) = \mathbf{h}.\end{aligned} \tag{5.3}$$

However, if layer normalization is executed for the input before the weights the outcome will not remain the same after re-scaling and re-centering.

5.4.2 Dataset re-scaling and re-centering:

This method is also invariant to dataset re-scaling and re-centering by different aggregated inputs of neurons remains unchanged in varying conditions and under individual training cases. The normalization scalars µ and r are used to re-scale x by δ to a new point to x'.

$$h_i' = f(\frac{g_i}{\sigma'}(w_i^T \mathbf{x}' - \mu') + b_i) = f(\frac{g_i}{\delta\sigma}(\delta w_i^T \mathbf{x} - \delta\mu) + b_i) = h_i \tag{5.4}$$

5.5 Batch Normalization

For training deep neural networks, batch process regulates the inputs within each layer by deducting the mean value of the batch while the batch standard deviation is divided. This increases the ability of generalizing of the model. Thus stabilize the learning process. It is particularly and most effectively used for training deep convolutional neural network and is been used as a regularization method.
Batch normalization does not normalize the activation of the entire layer, it works in small batches of training data and normalizes the activation of a layer. To make sure that each feature map is independent, each feature map within the layer is normalized separately. Training is done by computing the mean and standard deviation of the activation for each small batch and normalizing the activation. The normalization is done using the mean and standard deviation of the activation of the whole set while speculating.
Computing mean and Standard Deviation:

$$\mu = \frac{1}{m}\sum_i z_i^{[l]} \tag{5.5}$$

$$\sigma^2 = \frac{1}{m}\sum_i \left(z_i^{[l]} - \mu\right)^2 \tag{5.6}$$

Normalizing the vectors $z^{[l]}$ as follows:

$$z_{norm}^{(i)} = \frac{z^{(i)} - \mu}{\sqrt{\sigma^2 + \varepsilon}} \tag{5.7}$$

Here, the epsilon, ε, is a small constant added to the variance to prevent division by zero.

5.6 Softmax

N-dimensional vector with real values in the range (0, 1), where the output vector is scaled to make sum of all the elements to 1 which makes it a probability distribution over the N classes, is produced from an N-dimensional vector of real numbers in Softmax function. Softmax provides a smooth gradient which makes it easier to optimize during training a set of data.

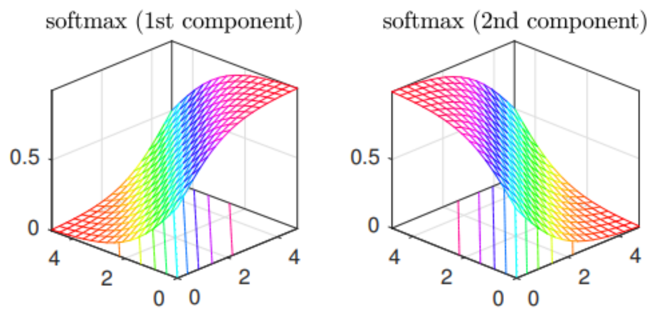


Figure 5.4: components of Softmax function over $\mathbf{R}^2$ with $\lambda = 1$.

$$softmax(z_i) = \frac{exp(z_i)}{\sum_j exp(z_j)} \tag{5.8}$$
$$1 \leq i \leq N$$

Here, the values from the neurons of the output layer, also known as logits, are represented by $\mathbf{z}$. The exponential e acts as the non-linear function. Lastly, the values are divided by the sum of all the exponential values in order to normalize.
If $\lambda = 1$, it is referred to as the standard softmax function, which is visualized in Figure 5.4

In multi-class classification problems, the softmax function is implemented in conjunction with the cross-entropy loss function. The difference between the predicted probability distribution and the true distribution is determined by the cross-entropy loss function which makes it easy to be optimized.

5.7 Categorical Cross Entropy

During model training, loss functions are utilized in the forward propagation stage as the last step to evaluate the model's goodness of fit on the training batch. Categorical cross entropy (CCE) is used as a loss function in multi-class classification problems. It is inspired by the concept of entropy from information theory and bears similarities with the concepts of KL (Kullback-Leibler) divergence and log loss.
The entropy H of a discrete random variable X is a measure of the amount of uncertainty or randomness in the variable, with a probability distribution $p(x)$, and is defined as follows:

$$H(X) = -\sum_x \log(p(x)) \tag{5.9}$$

In the equation above, a larger value of $H(x)$ corresponds to a greater uncertainty and vice-versa. Consequently, the term "cross-entropy" refers to a measure or a means for quantifying the differences between two probability distributions for a given random variable. Hence, categorical cross-entropy is a measure of the differences between the predicted probability distribution over the classes and the true probability distribution. It is a probabilistic loss function that is calculated as the negative log of the predicted probability for the true class:

$$CCE(\hat{y}, y) = -\sum_{i=1} y_i \log(\hat{y}_i) \tag{5.10}$$

where, y is the target label and $\hat{y}$ is the softmax induced model output or logits. As mentioned in the previous subsection, this loss function is used with a softmax activation function to compute the loss. In the case of an image captioning model, this is useful for optimizing a conditional probability distribution which allows the model to compute a target token given an image and the previous tokens in its caption sequence.

5.8 Convolution

One of the key challenges in image recognition is the variability in the position, scale, and appearance of objects in an image. Convolutional Neural Networks (CNNs) is a deep learning neural network developed to address this challenge. CNNs are designed to process multidimensional spatial data, such as an image, which allows them to learn spatially hierarchical representations of the input data. The architecture of a CNN consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers.
In a CNN, the convolutional layers utilize kernels, also known as filters, to extract features from the input image. These filters are intended to identify specific characteristics such as edges, textures, and shapes from the image. The filters are applied using a sliding window approach, moving across the entire image to create a feature map. After that, the feature map goes through a non-linear activation function like ReLU to add non-linearity to the model. The feature map is then reduced in dimension by using a pooling technique like max-pooling which preserves the essential features while reducing the spatial dimensions of the feature map. This process is repeated multiple times, with each successive layer detecting increasingly complex features in the image. The final output of the convolutional layer is usually passed through one or more fully connected layers for classification or further processing. Figure 5.5 demonstrates the convolution operation performed for a 3×3 kernel on a zero-padded 5×5 image.
The convolutional mathematical expression for a 2D image, I of size $M \times N$ is as follows:

$$F \star I(x, y) = \sum_{j=-M}^{M} \sum_{i=-N}^{N} F(i, j) I(x - i, y - j) \tag{5.11}$$

Here, F represents the convolution filter, the star notation between F and I represents convolution and i represents the number of neighboring horizontal pixels that will be considered on each side of the pixel and j represents the number of neighboring vertical pixels that will be considered on each side of the pixel.

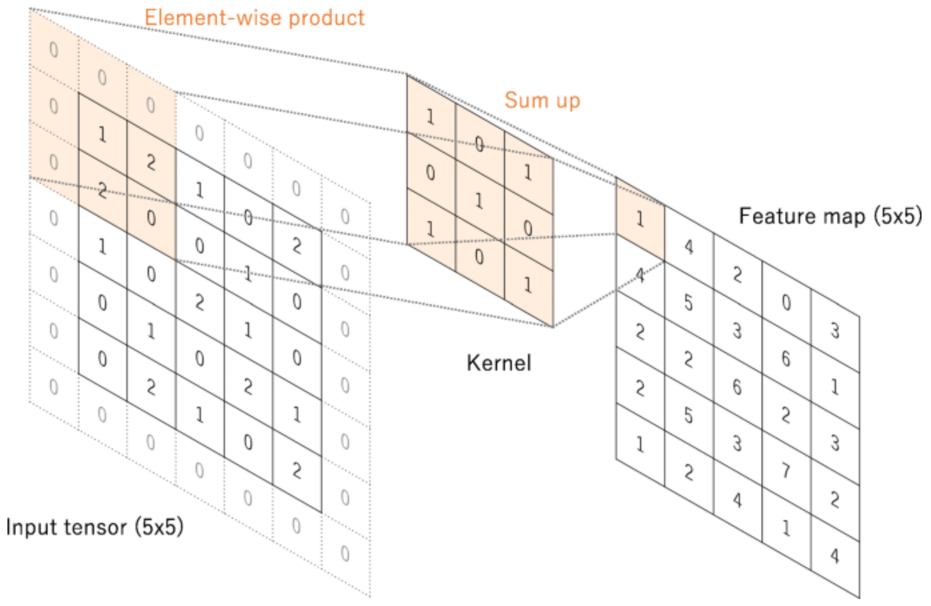


Figure 5.5: The convolution operation

5.9 Supervised Learning

Using a labeled training dataset, a neural network is trained to, give a desired output to be mapped from the input in supervised learning. The training data consists of pairs of input data and the corresponding correct output. The objective here is to determine a general rule that maps input to expected and accurate output.
During the training phase of a supervised learning neural network, the network makes predictions about the output given a specific input, and the network adjusts its internal parameters (weights and biases) to minimize the difference between the predicted labels and the ground truth, as indicated by the training data. This process is repeated for multiple input-output pairs in the training data. Once the network has been trained, it can then make predictions on new, unseen data and generalize to inputs it has not seen before.

On basis of prior experiences, by using supervised learning the model can

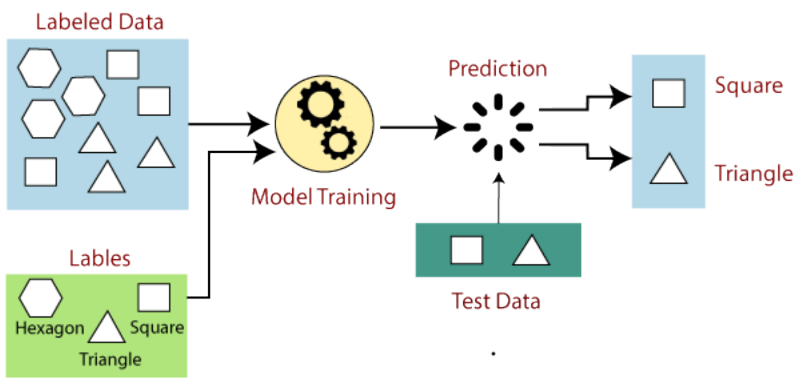


Figure 5.6: Supervised Learning Pipeline

predict the labels on unseen data which is one of the main advantages of supervised learning. It is a widely used learning method in real-world problems i.e. fraud detection, spam filtering, etc.
However, these models are not perfect for taking care of complex tasks. On times when the test dataset is different from the training dataset, it fails to predict the correct label. Furthermore, supervised learning is time-consuming as it needs a lot of computation time.

5.10 Transfer Learning

Transfer learning allows a machine learning model that has already been trained on one task to be used as a starting point for a new, related task, rather than training a new model from scratch. This can be useful when there are limited resources, such as data or computing power, available for the new task. Transfer learning allows the model to utilize its knowledge from the original task as a foundation for learning the new task, rather than starting with no prior knowledge.
Some of the layers of the model are mobilized and trained using the new data to fine-tune a pre-trained model for downstream tasks. The layers that are immobilized are still used in the model, but their weights remain unchanged during training.
Although it is possible to train CNNs from scratch for small datasets, it takes extremely huge amounts of processed data, and computational power, while being time-consuming and expensive to make them reliable for practical use. To overcome this problem transfer learning has been introduced and is widely used across the domain. The benefit of using transfer learning is that a large dataset is not needed for training. Moreover, as pre-trained weights can be used and only the weights of the last couple of layers, transfer learning requires less computational power.

5.11 Word Vectors and Embeddings

Word vectors and word embeddings help us represent the meaning of a word mathematically. Word embeddings allow computers to discern words with similar meanings and group them by representing them as real-valued vectors. A low-dimensional embedding space for words allows lexical and semantic meanings to be learned and represented in a distribution. The similarities between such words can be computed using the cosine dot product or cosine similarity.

$$sim(w_1, w_2) = \cos(\theta) = \frac{\vec{w_1}.\vec{w_2}}{||\vec{w_1}||\,||\vec{w_2}||} \tag{5.12}$$

Word vectors, on the other hand, transform text sequences into sequences of integers. When working with a small corpus, it is desirable to have a simple yet efficient input pipeline to pre-process text data and extract meaningful features. Hence we resort to the latter for our use case.

5.12 Vision Transformer Network

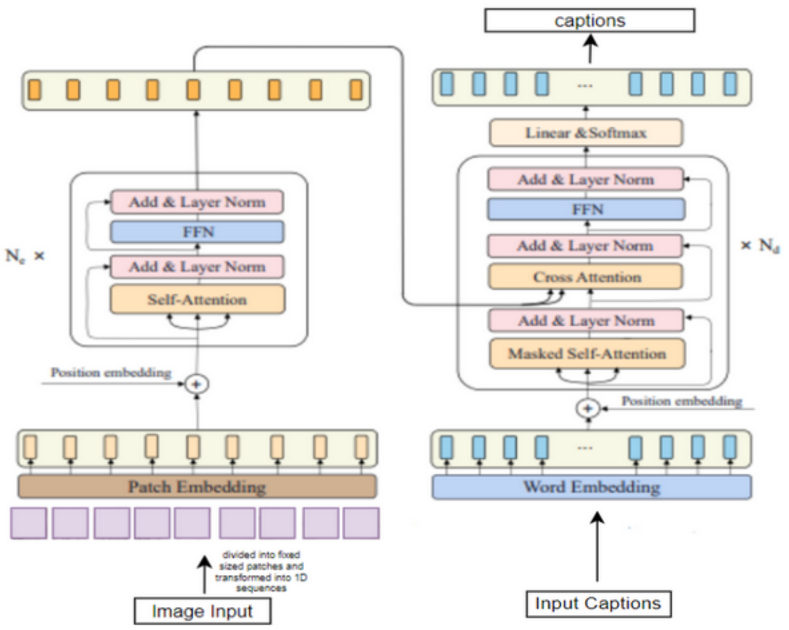


Figure 5.7: Vision-Transformer architecture

5.13 EfficientNet-Transformer Network

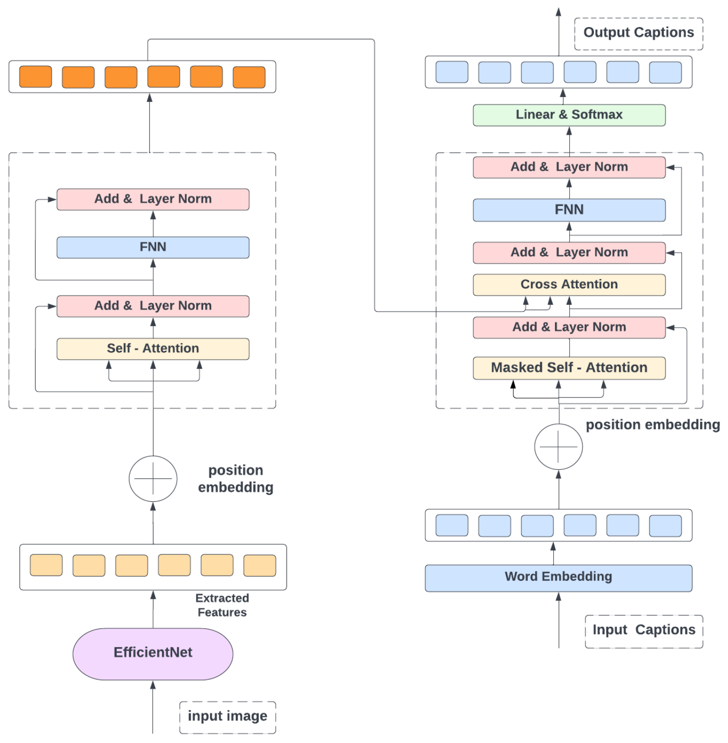


Figure 5.8: EfficientNet-Transformer architecture

5.14 Image Encoders

a. EfficientNet

EfficientNets, as proposed in [41], are based on convolutional neural networks which utilize a carefully balanced and scaled architecture to improve performance and optimizations. The scaling method offers uniform scaling of depth, width, and resolution by utilizing the concept of compound coefficients. The common scaling parameters for a baseline model may be defined as F_i, L_i, H_i, W_i and C_i where F_i denotes the operator, L_i denotes the length of the network, H_i and W_i denotes the spatial dimensions of the input tensors and finally Ci denotes the number of channels. A simple ConvNet N composed of multiple layers can hence be defined as:

$$N = \odot_{i=1,2,\ldots s} F_i^{L_i}(X\langle H_i, W_i, C_i\rangle) \tag{5.13}$$

and, the parameters for the scaled model are defined as width (w), depth (d), and resolution (r), which are all dependent on one another. Deeper ConvNets have the ability to extract complex and lower-level features for images of higher resolutions. However, such models may have the tendency to suffer from the vanishing gradient problem. Wider models are often implemented at smaller scale and can capture fine-grained features. Higher-resolution input images correspond to better accuracy with the appropriate depth and width of the ConvNet. Hence, it is important to coordinate a balance between the three parameters. This implies a multidimensional scaling of the model rather than the traditional single-dimensional scaling. The compound scaling method uses a compound coefficient to uniformly scale the network in a principled way:

$$depth(d) = \alpha^{\phi} \tag{5.14}$$

$$width(w) = \beta^{\phi} \tag{5.15}$$

$$resoloution(r) = \gamma^{\phi} \tag{5.16}$$

where α, β, and γ are hyperparameters that are to be determined via grid search. The compound coefficient ϕ, on the other hand, is defined by the user and it controls the number of resources allocated to the parameters whereas, α, β, and γ these three parameters, themselves, are indicators that determine how these resources are to be assigned. Our model employs the EfficientNetB1 architecture, which was pre-trained on ImageNet. The input images were resized to 224 x 224 as per the requirement stated by its authors in the literature. The EfficientNetB1 architecture is visualized figure 5.9 where the MBConv block

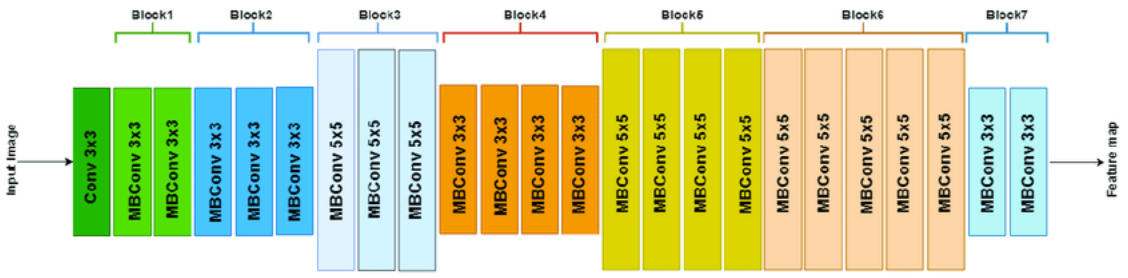


Figure 5.9: EfficientNetB1 architecture

refers to a variant of the inverted residual block that was originally proposed for the MobileNetV2 architecture [42]. The MBConv block aims to increase the efficiency of a network by reducing computational requirements while preserving its ability to extract relevant features. This is achieved through the use of a depthwise convolution, which applies unique filters to each input channel, and a pointwise convolution, which combines the output from the depthwise convolution using a 1x1 convolution to adjust the number of channels. A technique called "bottleneck" is also employed, which reduces the number of input channels before the depthwise convolution through a 1x1 convolution, further decreasing computational costs. MBConv is a key component of EfficientNet, a model that utilizes this block while also scaling network depth, width, and resolution to achieve improved image classification results with increased efficiency.

b. Vision Transformer (ViT) Encoder

Vision Transformer encoders, as proffered in [37] , are fundamentally different

from ConvNets (CNN-based architectures) in the sense that they lack the inductive biases of ConvNets. In other words, they are not translation invariant and, they lack a locally restricted receptive field which helps reduce the input region of the deeper layers. Transformers, furthermore, are by design permutation invariant meaning - the model does not assume any spatial relationship between features (unlike ConvNets). This is largely due to their inability to process grid-structured spatial inputs. They are, instead, designed to process sequential data.
Vision transformers are designed to convert these spatial inputs of shape $R^{H \times W \times C}$ to sequential data by dividing the spatial input into fix sized 2D patches.

$$x_p \in^{N \times (P^2 \cdot C)} \tag{5.17}$$

$$N = HW/P^2 \tag{5.18}$$

where P is the patch size, C is the number of color channels, and N is the value for the resulting number of patches that is computed using the $height(H) \times width(W)$ resolution of the image. Each patch is treated as a sequence token. These patches are then flattened and fed to a linear layer to obtain their projection embeddings, also known as patch embeddings. The positions of these embeddings are encoded using a positional embedding layer to retain the positional information(e.g. patch position) of the patches. Finally, the resulting outputs are fed to a standard transformer encoder.

5.15 The Transformer Architecture

a. Self-Attention

As we have seen in Chapter 1, sequence-to-sequence models utilize the encoder-decoder architecture by encoding the input sequences in the intermediate or hidden state vector and then passing the context vector derived from the hidden state vectors to the decoder to generate the output sequences. This context vector c, which is simply the sum of the encoder hidden states (h_1, ... , ... , h_T), stores all the information in the hidden state in a compressed form. A standard sequence-to-sequence model computes a sequence of outputs (y_1, ... , ... , y_T) from a given sequence of inputs (x_1, ... , ... ,x_T) by iterating over the following equations.

$$h_t = sigm(W^{hx} x_t + W^{hh} h_{t-1}) \tag{5.19}$$

$$y_t = W^{yh} h_t \tag{5.20}$$

This vanilla architecture, however, suffers from its inability to capture proper information from input sequences beyond a certain length. Attention is incorporated, as a workaround, with these sequential inputs to overcome the bottlenecking of the maximum sequence length and its inability to encode information from all time steps in the context vector c by introducing a direct

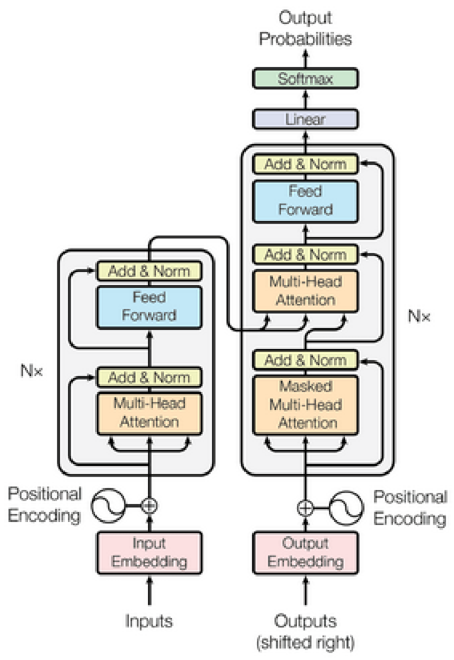


Figure 5.10: The transformer model architecture

connection to each time step. This is essentially a scalar value or a score, assigned to each hidden state, that describes the relationship or the "alignment score" between the all hidden states in the encoder and the previous hidden state in the decoder. In other words, for each hidden state h_j of the encoder we compute a scalar weight.

$$c = q(\{h_1, ..., h_{T_x}\}) \tag{5.21}$$

where, q is some non-linear function. Each element in the context vector is computed as follows

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{5.22}$$

and the weight $\alpha_i j$ of each hidden state h_j is computed as follows:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{5.23}$$

where,

$$e_{ij} = a(s_{i-1}, h_j) \tag{5.24}$$

where, a is some alignment score function, which determines the type of attention employed and s_{i-1} denotes the previous hidden state of the decoder which is computed in the previous time step as s_i by

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{5.25}$$

where f is some non-linear function. The resultant context vector c is thus a weighted sum of the hidden states in the encoder. During inference, a predicted output sequence element y_i is computed as

$$p(y_i|y_1, ..., y_{i-1}, X) = g(y_{i-1}, s_i, c_i) \tag{5.26}$$

where g is some non-linear function, which is the softmax function in our case. Self-attention is a key component of the transformer architecture, which, instead of looking for an input-output sequence alignment, computes the alignment scores between the elements of the sequence itself, which is simply implemented by tweaking the alignment score function a and its parameters.

b. Scaled Dot Product Attention

It is a scaled attention function or alignment score function that can be parameterized with three different representations as inputs, namely queries(Q), keys(K), and values(V). These three representations are initialized by a simple matrix multiplication between the token embedding representation of sequences X and three different arbitrarily initialized, learnable weight matrices:

$$Q = XW_Q \tag{5.27}$$

$$K = XW_K \tag{5.28}$$

$$V = XW_V \tag{5.29}$$

and the attention score is calculated as follows:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{5.30}$$

where d_k denotes the dimension of the input keys, queries and values. Here, $softmax\left(\frac{QK^T}{\sqrt{d_k}}V\right)$ is the self-attention matrix which is a softmax function mapping of the value of the dot product of Q and K scaled by $\frac{1}{\sqrt{d_k}}$ and, V is the value matrix.

c. Multi-head Attention and Masked Multi-head attention

Multi-head attention expands on the idea of self-attention, where multiple sets of attention scores are computed in parallel using multiple sets of learned, linear projections. The output vectors, also called attention heads, are concatenated and the resultant vector is linearly projected (W^O) one final time to aggregate the attention scores before being passed on to the next layer.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_k)W^O \tag{5.31}$$
$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

Each multi-head attention block is followed by a normalization layer and a residual connection whose outputs are then fed to a feed-forward network

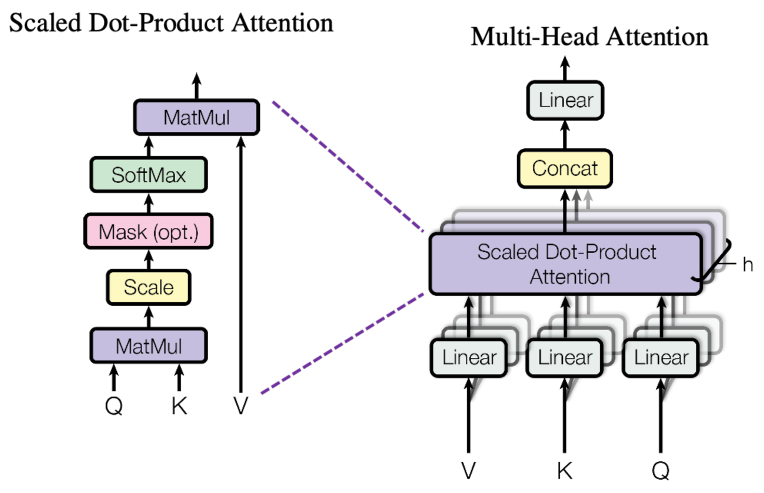


Figure 5.11: Multi-head Attention

which is also followed by a normalization layer and a residual connection. This whole concept is visualized in figure 5.11.

Masked multi-head is a modified version of multi-head attention which makes its appearance in the bottom-most layer of the decoder. It processes the token embedding output sequences in a similar fashion as the multi-head attention, but it hides the information regarding the prediction of the token in the next position using a masking matrix M which is composed of zeros and $-\infty$.

$$MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T + M}{\sqrt{d_k}}\right)V \tag{5.32}$$

This forces the decoder to only make predictions based on the previously predicted tokens in the output sequence (refer to the last equation of 5.4). Hence, the mask changes accordingly for every new token that is computed in the output sequence.

d. Layer Normalization and Residual Skip Connections

Layer normalization [43] aims to reduce the covariate shift, the shift in the distribution of data between the training set and the test set, across batches of the training set by normalizing the summed inputs of the hidden layers independently and, hence, fixing the mean and variance of the summed inputs within each layer. The layer normalization statistics over the lth hidden layer in a feed-forward network are computed as follows:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \tag{5.33}$$

$$\sigma_l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \tag{5.34}$$

where a^l is the vector representation of the summed inputs to the neurons in that layer and H is the dimension of that hidden layer, alongside μ and σ being the mean and standard deviation of the distribution across that layer respectively.
Residual skip connections are used to allow the representations of different levels of processing to interact. In a transformer model, it introduces a linear component to the sub-layers which improves the optimization and performance of the model and helps eliminate the degradation problem with deeper networks [23][31]. The residual skip connections along with layer normalization are implemented as follows:

$$y = LN(F(x_i, W_i) + \lambda x) \tag{5.35}$$

where y is the normalized output, x is the input tensor, F denotes the weighted feed-forward layer parameterized by W, and λ denotes the modulating factor that controls the relative importance of the skip connection. LN is the normalization function, with learned parameters γ and β:

$$LN_{\gamma,\beta}(a^l) = \gamma \hat{a}^l + \beta, \text{ where } a^l \in R^H \tag{5.36}$$

$$\text{and } \hat{a}^l = \frac{a^l - \mu^l}{\sqrt{\sigma^{l2} + \in}}, \text{ where } \hat{a}^l \in R \tag{5.37}$$

Note: a^l and x both denote the same input tensor in the equations above.

e. Cross-Attention

Cross-attention, performed in the decoder - which is the same operation as multi-head attention but with inputs of different modalities - mixes the embedding sequences of the image representations and the text representations. The text embedding sequences are used as input queries and the image embedding sequences play the role of the key and value inputs, thus, introducing information from the input sequence to the decoding layers by doing so. This allows for the decoder to predict the next token in the output sequence. This last step is repeated with every new output token that is generated, and is added to the output sequence which is used as key input in the next decoder layer.

f. Sinusoidal Positional Encoding

Sinusoidal positional encoding is used to provide the model with positional information of the words in the sequences. This enables the model to deal with relative positions of words in a sequence while maintaining the consistency of the length each of time step between words across sequences of different lengths and, to capture long term dependencies of words in sentences. Positionally encoding the input vectors compensates for the model's lack of recurrence.

Sine and cosine functions of different frequencies were employed to positionally encode the word embedding representations. The position vector $P \epsilon R^d$ for the

k^{th} token within a sequence of length d is given as follows:

$$\vec{p} = f(k) \tag{5.38}$$

where, the i^{th} positions in $\vec{p}$ is computed as follows

$$p_{k,2i} = \sin\left(\frac{k}{10000^{2i/d}}\right), \tag{5.39}$$

$$p_{k,2i+1} = \cos\left(\frac{k}{10000^{2i/d}}\right) \tag{5.40}$$

The result is a d dimensional vector for each token k consisting of pairs of sines and cosines defined over a set of frequencies which decrease across its dimension.

5.16 Experimental Setup

The dataset required some cleaning as there were several samples with caption mismatch. A total of 816 such samples were discarded. For each model, we used an 80:20 ratio for splitting the BanglaLekhaImageCaptions dataset into training and test sets, 6670 samples belonging to the training set and 1668 samples in the test set, with a batch size of 32. The captions were tokenized, with a '$< start >'$ and an '$< end >'$ token added to the beginning and the end of each caption sequence, and then encoded to their corresponding vector representations using the TextVectorization class from Keras. The training set consisted of 4646 unique tokens (vocabulary size). An Adam optimizer was employed with a variable learning rate over the course of training using a custom learning rate schedule with warmup steps set to 4000 and the post warmup learning rate set to 1×10^{-3}. Callbacks such as ModelCheckpoint and EarlyStopping were used to halt training in case the model starts to overfit and to save the best model for evaluating the caption metrics, respectively.
The models were trained for a total of 50 epochs each for an entire day on a single NVIDIA Tesla P100 GPU along with 16 GB of RAM and a Ryzen 7 3700x CPU.

a. Efficient-Net Transformer

Images were resized to 299 x 299. 3 encoders and 3 decoders, two attention heads each, were employed, with the last layer being a linear layer with its dimension equal to the size of the vocabulary of the training set captions from which the sequence was generated. The projection layer dimension and the hidden state dimension were both set to 2048.

b. Vision Transformer

A Huggingface vision transformer model (vit-base), pre-trained on the ImageNet dataset with 14 million images and 21 thousand classes by Google researchers [44] , was used to implement this model. The model used a patch size of 16 x 16, 768 hidden state dimensions, 3072 projection dimensions, and

12 encoders with 6 attention heads each. For our task 4 decoders with 6 attention heads each were used. The dimension of the output of the decoder layer was equal to the vocabulary size. Finally, the hidden state dimension was kept the same as the encoder. Input images had to be resized to 224 x 224 to satisfy the input size requirement of the pre-trained model.

Chapter 6

Results and Discussions

6.1 EfficientNet

The EfficientNet Transformer was able to predict the captions with reasonable accuracy. The reference captions appeared to be more subjective and specific in some instances whereas the model predicted the captions in a more generalized manner. This demonstrates a primary limitation of the dataset not having enough variations to its reference captions for each image. The first image depicts a man standing in

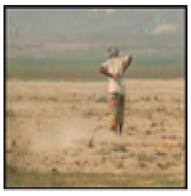
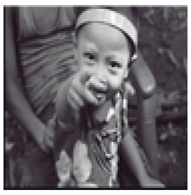
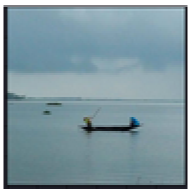
Image	Captions	
	Predicted text	একজন পুরুষ হেঁটে যাচ্ছে
	Reference text 1	একজন পুরুষ কাজ করছে
	Reference text 2	লুঙ্গী শার্ট পরা একজন পুরুষ মইয়ের দড়ি টানছে একটি ক্ষেতে।
	Predicted text	একটি শিশু আছে
	Reference text 1	একটি শিশু হাঁত তুলে আছে।
	Reference text 2	একজন চেয়ার এর উপরে বসে থাকা ছেলের কোলে একজন বাচ্চা মেয়ে আঙুল দিয়ে সামনে তাকিয়ে আছে।
	Predicted text	একটি নৌকায় একজন মানুষ আছে মানুষ আছে
	Reference text 1	একটা নৌকায় একজন লোক মাথায় কাগজ ধরে বসে আছে আর মাঝি মাথায় ছাতা ধরে নৌকা বাইচে।
	Reference text 2	নৌকায় দুইজন মানুষ আছে।

Figure 6.1: Examples of predicted sequences by the EfficientNet Transformer network

the middle of a paddy field. The model describes it as a man walking by whereas the reference text also addresses the environment in which the person is pictured. The second image portrays a child pointing at a camera. The model was able to recognize the presence of the child. However, the reference captions acknowledge the child pointing at the camera while sitting on a chair. The last image is a picture of multiple boats in a river. The model predicts that there are a bunch of people riding a boat whereas the reference text describes it in a similar manner.

6.2 Vision Transformer pre-trained on ImageNet Dataset

The vision transformer model, on the other hand, performed poorly due to its architectural complexity. The predicted captions lacked proper grammar and sentence semantics. A certain set of words appeared in pairs repeatedly at the end of the predicted sequence in multiple instances. While the sentences bore some correspondence to their image, they lacked proper grammatical syntax.

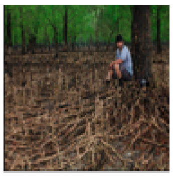
Image	Captions	
	Predicted text	একটি নৌকায় কয়েকজন মানুষ আছে ও অনেকগুলো মানুষ আছে দেখা যাচ্ছে দেখা যাচ্ছে দেখা যাচ্ছে
	Reference text 1	অনেকগুল নৌকা ও কয়েকটি লঞ্চ আছে।
	Reference text 2	নদীতে অনেক নৌকা চলছে আর কয়েকটা লঞ্চ পাড়ে ভিড়ানো আছে।
	Predicted text	একজন পুরুষ ও একজন নারী আছে শিশু আছে শিশু আছে সামনে কয়েকজন মানুষ আছে
	Reference text 1	বোরখা পরিহিত একজন নারী হেঁটে আসছে।
	Reference text 2	একজন কালো বোরকা পরা মহিলা হেটে যাচ্ছে যার পাশে লাল টিনের বেড়া।
	Predicted text	একজন পুরুষ ও একজন নারী আছে শিশু আছে হেঁটে যাচ্ছে
	Reference text 1	একজন পুরুষ বসে আছে।
	Reference text 2	একটি গাছের অনেক শিকরের উপরে বসে আছে একজন পুরুষ।

Figure 6.2: Examples of predicted sequences by the pre-trained Vision Transformer network

6.3 Summary of training result

Table 6.1: Summary of Training Results

Model	***BLEU1***	***BLEU2***	***BLEU3***	***BLEU4***
EfficientNet Transformer	0.425	0.264	0.174	0.134
Vision Transformer(vit-base)	0.185	0.0940	0.0487	0.0339

EfficientNet Transformer, with its simpler architecture, outperformed the more complex Vision Transformer Network. The former had relatively far less training parameters than the latter. Shallower models outperform deeper models when trained on small datasets [45]. Both models suffered from overfitting when trained with one decoder and one encoder (in the case of EfficientNet). However, increasing the number of encoders and decoders for the EfficientNet transformer network seemed to yield better results. The Vision transformer model, on the other hand, still suffered from severe overfitting as the pre-trained transformer encoder came with 12 encoders by default. The EfficientNet Transformer Network achieved a BLEU-1 score of 0.42459 whereas the Vision Transformer model scored lower, 0.18521 in BLEU-1. A similar trend is observed across BLEU-2, BLEU-3, and BLEU-4.

6.4 Comparison with Prior Art

Table 6.2 details a comparison of our results against other methods explored in literature. The best-performing model, EfficientNet Transformer, achieved results similar to the InceptionV3 GRU merge model with a BLEU-1 of 0.425 against 0.425. In previous literature, the authors annotated and added more samples and, hence, increased the size of their dataset to improve their models' performances.

Table 6.2: Comparison with prior art

Model	***BLEU1***	***BLEU2***	***BLEU3***	***BLEU4***
Our Proposed model (EfficientNet Transformer)	0.425	0.264	0.174	0.134
Inception V3 GRU [29]	0.425	0.279	0.236	0.164
Inception V3 Transformer [34]	0.567	0.460	0.385	0.319
CNN+ResNet-50[merge] [30]	0.651	0.426	0.278	0.175

Chapter 7

Conclusion and Future Work

Previous literature in English suggests using reinforcement learning [36] to fine-tune the transformer model for improved performance. Furthermore, more data on Bengali image captioning must be produced and made available in public domains. In addition, reference caption per image in the publicly accessible Bengali dataset should be increased to a number close to or greater than what is available in English for better performance.
The purpose of this study was to investigate the effectiveness of using transformer models with pre-trained EfficientNet and Vision Transformer image encoders to generate captions for images. However, due to the limited amount of training data, both models experienced significant overfitting. Despite the EfficientNet transformer model showing better results than the Vision Transformer due to the latter's excessive depth, the lack of availability of Bengali image captioning dataset in terms of quality and quantity remains a significant obstacle to improving the models' performance.

References

[1] A. Ardila, B. Bernal, and M. Rosselli, "Language and visual perception associations: Meta-analytic connectivity modeling of brodmann area 37," *Behavioural Neurology*, vol. 2015, p. 565 871, Jan. 2015, ISSN: 0953-4180. DOI: 10.1155/2015/565871. [Online]. Available: https://doi.org/10.1155/2015/565871.

[2] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. arXiv: 1406.1078. [Online]. Available: http://arxiv.org/abs/1406.1078.

[3] M. Tanti, A. Gatt, and K. P. Camilleri, "What is the role of recurrent neural networks (rnns) in an image caption generator?" *CoRR*, vol. abs/1708.02043, 2017. arXiv: 1708.02043. [Online]. Available: http://arxiv.org/abs/1708.02043.

[4] M. TANTI, A. GATT, and K. P. CAMILLERI, "Where to put the image in an image caption generator," *Natural Language Engineering*, vol. 24, no. 3, pp. 467–489, 2018. DOI: 10.1017/S1351324918000098.

[5] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[6] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2014. DOI: 10.48550/ARXIV.1409.0473. [Online]. Available: https://arxiv.org/abs/1409.0473.

[7] L. Guo, J. Liu, J. Tang, J. Li, W. Luo, and H. Lu, "Aligning linguistic words and visual semantic units for image captioning," in *Proceedings of the 27th ACM International Conference on Multimedia*, ser. MM '19, Nice, France: Association for Computing Machinery, 2019, pp. 765–773, ISBN: 9781450368896. DOI: 10.1145/3343031.3350943. [Online]. Available: https://doi.org/10.1145/3343031.3350943.

[8] X. Yang, K. Tang, H. Zhang, and J. Cai, *Auto-encoding scene graphs for image captioning*, 2018. DOI: 10.48550/ARXIV.1812.02378. [Online]. Available: https://arxiv.org/abs/1812.02378.

[9] T. Yao, Y. Pan, Y. Li, and T. Mei, *Hierarchy parsing for image captioning*, 2019. DOI: 10.48550/ARXIV.1909.03918. [Online]. Available: https://arxiv.org/abs/1909.03918.

[10] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762.

[11] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, *Show and tell: A neural image caption generator*, 2014. DOI: 10.48550/ARXIV.1411.4555. [Online]. Available: https://arxiv.org/abs/1411.4555.

[12] K. Xu, J. Ba, R. Kiros, *et al.*, *Show, attend and tell: Neural image caption generation with visual attention*, 2015. DOI: 10.48550/ARXIV.1502.03044. [Online]. Available: https://arxiv.org/abs/1502.03044.

[13] J. Donahue, L. A. Hendricks, M. Rohrbach, *et al.*, “Long-term recurrent convolutional networks for visual recognition and description,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, 2017. DOI: 10.1109/TPAMI.2016.2599174.

[14] J. Aneja, A. Deshpande, and A. Schwing, *Convolutional image captioning*, 2017. DOI: 10.48550/ARXIV.1711.09151. [Online]. Available: https://arxiv.org/abs/1711.09151.

[15] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2014. DOI: 10.48550/ARXIV.1405.0312. [Online]. Available: https://arxiv.org/abs/1405.0312.

[16] A. H. Kamal, M. A. Jishan, and N. Mansoor, *Textmage: The automated bangla caption generator based on deep learning*, 2020. DOI: 10.48550/ARXIV.2010.08066. [Online]. Available: https://arxiv.org/abs/2010.08066.

[17] N. Mansoor, A. H. Kamal, N. Mohammed, S. Momen, and M. M. Rahman, *Banglalekhaimagecaptions*, Jul. 2019. [Online]. Available: https://doi.org/10.17632/rxxch9vw59.2.

[18] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “Bleu: A method for automatic evaluation of machine translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL ’02, Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. [Online]. Available: https://doi.org/10.3115/1073083.1073135.

[19] S. Banerjee and A. Lavie, “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments,” in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72. [Online]. Available: https://aclanthology.org/W05-0909.

[20] R. Vedantam, C. L. Zitnick, and D. Parikh, “Cider: Consensus-based image description evaluation,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4566–4575. DOI: 10.1109/CVPR.2015.7299087.

[21] T. Deb, M. Z. A. Ali, S. Bhowmik, *et al.*, “Oboyob: A sequential-semantic bengali image captioning engine,” *Journal of Intelligent & Fuzzy Systems*, vol. 37, pp. 7427–7439, 2019, 6, ISSN: 1875-8967. DOI: 10.3233/JIFS-179351. [Online]. Available: https://doi.org/10.3233/JIFS-179351.

[22] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, *Enriching word vectors with subword information*, 2016. DOI: 10.48550/ARXIV.1607.04606. [Online]. Available: https://arxiv.org/abs/1607.04606.

[23] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, *Inception-v4, inception-resnet and the impact of residual connections on learning*, 2016. DOI: 10.48550/ARXIV.1602.07261. [Online]. Available: https://arxiv.org/abs/1602.07261.

[24] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2014. DOI: 10.48550/ARXIV.1409.1556. [Online]. Available: https://arxiv.org/abs/1409.1556.

[25] M. Rahman, N. Mohammed, N. Mansoor, and S. Momen, "Chittron: An automatic bangla image captioning system," *Procedia Computer Science*, vol. 154, pp. 636–642, 2019, Proceedings of the 9th International Conference of Information and Communication Technology [ICICT-2019] Nanning, Guangxi, China January 11-13, 2019, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.06.100. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919308701.

[26] A. S. Ami, M. Humaira, M. A. R. K. Jim, S. Paul, and F. M. Shah, "Bengali image captioning with visual attention," in *2020 23rd International Conference on Computer and Information Technology (ICCIT)*, 2020, pp. 1–5. DOI: 10.1109/ICCIT51783.2020.9392709.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, *Rethinking the inception architecture for computer vision*, 2015. DOI: 10.48550/ARXIV.1512.00567. [Online]. Available: https://arxiv.org/abs/1512.00567.

[28] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, 2016. DOI: 10.48550/ARXIV.1610.02357. [Online]. Available: https://arxiv.org/abs/1610.02357.

[29] A. M. Faruk, H. A. Faraby, M. M. Azad, M. R. Fedous, and M. K. Morol, *Image to bengali caption generation using deep cnn and bidirectional gated recurrent unit*, 2020. DOI: 10.48550/ARXIV.2012.12139. [Online]. Available: https://arxiv.org/abs/2012.12139.

[30] M. F. Khan, S. M. S.-U.-R. Shifath, and M. S. Islam, *Improved bengali image captioning via deep convolutional neural network based encoder-decoder model*, 2021. DOI: 10.48550/ARXIV.2102.07192. [Online]. Available: https://arxiv.org/abs/2102.07192.

[31] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. DOI: 10.48550/ARXIV.1512.03385. [Online]. Available: https://arxiv.org/abs/1512.03385.

[32] M. Humaira, S. Paul, M. A. R. K. Jim, A. S. Ami, and F. M. Shah, "A hybridized deep learning method for bengali image captioning," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 2, 2021. DOI: 10.14569/IJACSA.2021.0120287. [Online]. Available: http://dx.doi.org/10.14569/IJACSA.2021.0120287.

[33] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. [Online]. Available: https://aclanthology.org/D14-1162.

[34] F. M. Shah, M. Humaira, M. A. R. K. Jim, A. S. Ami, and S. Paul, "Bornon: Bengali image captioning with transformer-based deep learning approach," *CoRR*, vol. abs/2109.05218, 2021. arXiv: 2109.05218. [Online]. Available: https://arxiv.org/abs/2109.05218.

[35] M. A. H. Palash, M. D. A. A. Nasim, S. Saha, F. Afrin, R. Mallik, and S. Samiappan, "Bangla image caption generation through cnn-transformer based encoder-decoder network," *CoRR*, vol. abs/2110.12442, 2021. arXiv: 2110.12442. [Online]. Available: https://arxiv.org/abs/2110.12442.

[36] W. Liu, S. Chen, L. Guo, X. Zhu, and J. Liu, "CPTR: full transformer network for image captioning," *CoRR*, vol. abs/2101.10804, 2021. arXiv: 2101.10804. [Online]. Available: https://arxiv.org/abs/2101.10804.

[37] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. arXiv: 2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[38] M. Cornia, M. Stefanini, L. Baraldi, and R. Cucchiara, "M^2: Meshed-memory transformer for image captioning," *CoRR*, vol. abs/1912.08226, 2019. arXiv: 1912.08226. [Online]. Available: http://arxiv.org/abs/1912.08226.

[39] J. Yu, J. Li, Z. Yu, and Q. Huang, "Multimodal transformer with multi-view visual representation for image captioning," *CoRR*, vol. abs/1905.07841, 2019. arXiv: 1905.07841. [Online]. Available: http://arxiv.org/abs/1905.07841.

[40] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. arXiv: 1506.01497. [Online]. Available: http://arxiv.org/abs/1506.01497.

[41] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. arXiv: 1905.11946. [Online]. Available: http://arxiv.org/abs/1905.11946.

[42] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2018. DOI: 10.48550/ARXIV.1801.04381. [Online]. Available: https://arxiv.org/abs/1801.04381.

[43] J. L. Ba, J. R. Kiros, and G. E. Hinton, *Layer normalization*, 2016. DOI: 10.48550/ARXIV.1607.06450. [Online]. Available: https://arxiv.org/abs/1607.06450.

[44] B. Wu, C. Xu, X. Dai, *et al.*, *Visual transformers: Token-based image representation and processing for computer vision*, 2020. arXiv: 2006.03677 `[cs.CV]`.

[45] K. Pasupa and W. Sunhem, "A comparison between shallow and deep architecture classifiers on small dataset," in *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2016, pp. 1–6. DOI: 10.1109/ICITEED.2016.7863293.