ARABIC IMAGE CAPTIONING USING DEEP LEARNING WITH ATTENTION

by

SABRI MONAF SABRI

(Under the Direction of Frederick Maier)

ABSTRACT

Automatic image captioning is a challenging deep learning task involving computer vision to understand the contents of an image and natural language generation to compose a coherent description for that image. Image captioning for the English language is well-developed and has high precision, with some recent work surpassing human-level performance. However, Arabic image captioning work has been lacking, with few papers published having relatively low-performance results. Researchers attribute this to the Arabic language's morphological complexity and the to lack of large, robust benchmark datasets compared to those available for the English language.

Our proposed framework includes using an improved text preprocessing pipeline incorporating a word segmenter to alleviate some of the morphological complexity associated with the Arabic language. We also build neural network architectures which include techniques not previously explored in the Arabic image captioning literature, such as attention mechanisms and transformers. Our approach yields better results over the most recent published work on the subject in Arabic, improving the BLEU-1 score from 33 to 44.3 and the BLEU-4 score from 6 to 15.6.

INDEX WORDS: ARABIC IMAGE CAPTIONING, IMAGE CAPTIONING, ATTENTION, TRANSFORMERS, DEEP LEARNING

ARABIC IMAGE CAPTIONING USING DEEP LEARNING WITH ATTENTION

by

SABRI MONAF SABRI

BS, Al-Mustansiriya University, Iraq, 2016

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2021

ARABIC IMAGE CAPTIONING USING DEEP LEARNING WITH ATTENTION

by

SABRI MONAF SABRI

Major Professor: Frederick Maier

Committee: Khaled Rasheed
Sheng Li

Electronic Version Approved:

Ron Walcott
Vice Provost for Graduate Education and Dean of the Graduate School
The University of Georgia
August 2021

DEDICATION

To my parents for their continuous support and encouragement. And to my life's partner, Sarah, who has been with me at every step throughout this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

ARABIC IMAGE CAPTIONING

1.1 Introduction

Image Captioning is the process of describing the contents of an image. Automating this process is a challenging machine learning problem involving both computer vision and natural language processing. A model intended to do this needs to recognize different objects, their attributes, relevance, and relationship to one another and compose a coherent natural language sentence to describe them clearly. Image captioning has a wide range of high-impact applications, such as helping the visually impaired and automatically generating captions for web content for better content indexing and accessibility. It is also the basis for other tasks such as visual question answering.

Significant progress has been made in deep learning-based image captioning research over the last few years [12], [19], reaching high accuracy levels, and with more recent research, surpassing human-level performance in some instances [20]. However, Arabic Image Captioning (AIC) progress has been lacking, with the performance of the latest research [15] still lagging behind English Image Captioning research from 2015 [44]. El Jundi et al. [15] attribute the lack of progress to the morphologically complex nature of the Arabic language, and to the lack of large, diverse, and robust benchmark datasets similar to the ones available for the English language [25]. We use the latest research in AIC by El Jundi et al. [15] as the baseline to compare our results against. El Jundi et al. implemented an encoder-decoder model [38] using a pre-trained VGG19 [36] model as the encoder and a custom GRU-based [11] decoder with a word embedding [29] layer. Our proposed approach improves upon El Jundi et al.'s work by: 1) Using an improved tokenization technique to alleviate some of morphological complexity of the Arabic language, 2) using newer image models such as EfficientNet [41] and MobileNetV2 [35], 3) including an attention mechanism [6], [27] layer in the decoder model to improve captioning accuracy, and 4) implementing

an entirely new model based on the transformers [42] architecture.

The next section briefly explores some of the approaches that researchers have previously used in image captioning, the datasets they used for training, and the evaluation metrics they used to evaluate their results. Section 1.3 describes our contributions in more detail.

1.2 Background

1.2.1 Evaluation Metrics

The process of evaluating the output of an image captioning model is somewhat similar to evaluating machine-translated sentences (i.e., comparing a generated sentence against a reference sentence(s)). That is why early on, image captioning researchers used metrics such as bilingual evaluation understudy (BLEU) [32] which estimates accuracy based on n-gram similarities (BLEU-1 being 1-gram similarity, BLEU-2 is 2-gram, and so on.). The BLEU score is measured on a scale of 0 to 1. It is a common practice to scale BLEU scores up by 100 (to a scale of 0 to 100) for better readability.

Another machine-translation metric that image captioning researchers use is the Metric for Evaluation of Translation with Explicit Ordering (METEOR) [7] which takes explicit ordering into account in addition to n-gram similarity. Similar to BLEU, METOER is also measured on a scale of 0 to 1 but is scaled up by 100 for better readability.

1.2.2 Datasets

Two data-sets are commonly used by researchers to train and benchmark image captioning models. The first dataset is the Common Objects in Context (COCO) [25]. COCO contains data for several computer vision tasks such as object detection, image segmentation, and image captioning. The image captioning dataset contains 83k training images and 41k testing images. Each image has five different captions.

The second dataset is Flicker8k [18], which contains 8,092 images with five captions for each image. The Flicker8k data set is roughly ten times smaller than COCO. An Arabic version of this dataset was pub-

lished by El Jundi et al. [15]. The captions were automatically translated using Google's Machine Translation API and manually validated and ranked by professional Arabic translators and the best three captions are kept (out of the original five), which results in a far smaller dataset which was already much smaller than what is available for other languages. This negatively impacts the quality of the deep learning models trained on this dataset (which generally require very large amounts of data).

1.2.3 Prior Work on Image Captioning with Deep-Learning

We will briefly go over some of the most important papers in the field [3], [12], [15], [20]–[22], [31], [44], [47] and the datasets [18], [25] and evaluation metrics [7], [32] that authors have used to evaluate their work. All papers discussed here, along with their scores, languages, and datasets used, are listed in Table 1.1.

1.2.3.1 English Image Captioning

The first neural-based image captioning paper was published in 2015 by Vinyal et al. [44]. The authors used an encoder-decoder architecture [38] with a convolutional neural network-based model as the encoder that extracts image features and a recurrent neural network-based model that generates the caption. This achieved a BLUE-1 of 64.6 on the COCO dataset. Other researches subsequently built upon this architecture. E.g., in the same year, Xu et al. [47] added an attention mechanism layer [6], [27] which allowed the language generating decoder to focus on salient regions of the image to produce the caption. This approach improved the BLEU-1 score of the previous work to 70.7. In 2018, Anderson et al. [3] implemented a model with two attention steps, bottom-up attention that detects objects based on Faster R-CNN [33] in the image and top-down attention that focuses the most salient objects, this approach resulted in a BLEU-1 score of 79.8.

Newer papers use the transformer architecture for image captioning. Corina et al. [12] use a meshed-memory transformer architecture which learns a multi-level representation of the relationships between image regions to improve captioning accuracy. This approach set the state-of-the-art score on the COCO [25] benchmark dataset, with a BLEU-1 score of 80.8. Another notable transformer-based approach is by Hu et al. [20] which can recognize novel objects not available in most datasets (they focus on more gen-

eral concepts) by pre-training the model on large-scale datasets with abundant tags and then fine-tuning it on the image/caption pairs. The accuracy score on the COCO dataset didn't surpass the state-of-the-art score set by Corina et al. [12]. The authors attribute that to the small number of visual concepts present in the COCO dataset, thus diminishing the benefit of learning an extensive visual vocabulary that the pre-training step does in this model.

The current state-of-the-art BLEU-1 score on the COCO dataset in English is produced by the model by Corina et al. [12] – a score of 80.8 out of 100. In the case of the METEOR metric, He et al. [20] produced the highest score at 45.4 out of 100 (versus the Corina et al.'s second best score of 29.2).

1.2.3.2 Arabic Image Captioning

The BLEU scores achieved in the AIC literature are universally lower than those in the English image captioning literature (due to reasons discussed in subsubsection 1.2.3.4). The earliest paper we found in the AIC literature was by Vasu [21], [22] who had published two subsequent papers on the subject. The first one used an encoder-decoder architecture similar to Vinyals et al. [44]. Vasu used a pre-trained image model as the encoder, and a deep belief network (DBN) **dbm** pre-trained by Restricted Boltzmann Machines as the decoder. In his second paper, Vasu replaced the DBN-based decoder with an LSTM-based [17] one. The two approaches achieved BLEU-1 scores of 34.8 and 55.6 respectively, both trained and evaluated on two different proprietary datasets built from pairing news articles images with their captions from sites such as Al-Jazeera News.

Al-Muazini et al.[31] introduced a merge model for generating Arabic captions. The overall architecture is composed of an LSTM-based linguistic encoding encoder and an image feature extractor. The output of both models then goes into another LSTM model which generates the caption. This approach achieved a BLEU-1 score of 46 on an unpublished Arabic version of the Flicker8k dataset translated using crowd-sourcing and machine translation.

The latest AIC research is by El Jundi et al. [15], who also use an encoder-decoder architecture. They used a pre-trained image model as the encoder and an LSTM-based model as the decoder. One of the more significant contributions of this paper is publishing a new manually validated dataset based on the

Flicker8K dataset. This opens the possibility for future researchers to benchmark their work against previous work, which was not possible before. All the papers cited above (except for El Jundi et al.'s) used a proprietary, unpublished dataset. El Jundi et al.'s approach achieved a BLEU-1 score of 33 on their Arabic version of the Flicker8k dataset.

1.2.3.3 Image Captioning in Other Languages

Researchers have worked on image captioning for other languages. E.g., Mishra et al. [30] worked on Hindi image captioning by implementing a similar architecture to Xu et al. [47] using an encoder-decoder architecture with attention. They explored several attention mechanism techniques such as spatial, visual, Bahdanau-style, and Luong-style attention. They achieved the highest score on the Bahdanau-style attention (67.0 BLEU-1 on a Hindi version of the COCO dataset).

Working on on Chinese image captioning, Lu et al. [26] used an architecture similar to to Xu et al., with the primary difference being the use of bidirectional LSTMs [17]. This approach achieved a BLEU-1 score of 78.5 on the AI Challenger Dataset [46], a Chinese dataset that which like COCO contains data for several computer vision tasks including image captioning.

1.2.3.4 Low BLEU scores in AIC

Table 1.1 contains a comparison of image captioning in different languages. Looking at the table, we can we can clearly see that AIC models (except for Vasu's [21], [22] which works on root words only) produce significantly lower scores than their English, Hindi, and Chinese counterparts. Agreeing with El Jundi et al., we believe that this is mainly due to the morphological complexity of the Arabic language; a single Arabic word typically has several attached attached morphemes (as illustrated in Figure 2.12) making the sentences have a much lower number of words than most languages and thus making error penalties much higher in n-gram similarity-based metrics such as BLEU. Another (perhaps even more important) reason is lack of a high-quality benchmark dataset that is as robust and extensive as what's available for other languages such as the COCO dataset [25]. The BLEU score bias in Arabic is discussed in detail in a paper by Bouamor et al. [9] where they verify the bias by comparing BLEU scores against human judgment scores.

Table 1.1: Summary of prior work on deep learning-based image captioning

Paper	Language	Year	Dataset	BLEU-1	BLEU-4	METEOR
Vinyals et al. [44]	EN	2015	COCO	64.6	24.6	23.7
Xu et al [47]	EN	2015	COCO	70.7	25.0	23.9
Anderson et al. [3]	EN	2018	COCO	79.8	36.3	27.7
Corina et al. [12]	EN	2020	COCO	80.8	39.1	29.2
He et al. [20]	EN	2021	COCO	-	34.9	45.4
Mishra et al. [30]	HI	2021	Hindi COCO	65.9	21.0	-
Lu et al. [26]	ZH	2021	AI Challenger [46]	78.5	47.8	41.5
Vasu [21]	AR	2017	News Articles [1]	34.8	-	-
Vasu [22]	AR	2018	Flicker8k [2]	65.8	22.3	20.09
Vasu [22]	AR	2018	News Articles [3]	55.6	18.9	18.01
Al-Muzaini et al. [31]	AR	2018	Flicker8k [4]	46	8	-
El-Jundi et al. [15]	AR	2020	Flicker8k [5]	33	6	-
Our Approach (Transformers)	AR	2021	Flicker8k [5]	44.3	15.7	34.3

1.3 Our Contribution

There are four main contributions that we introduced to improve upon previous AIC work.

1. We improve preprocessing by incorporating the AraBERT segmenter [4]. This step alleviates some of the morphological complexity of the Arabic language, specifically where the same word appears in multiple forms (e.g., definite vs. indefinite, masculine vs. feminine, conjunct vs. non-conjunct, etc.). This step reduced the number of unique tokens in the dataset's vocabulary from 10,396 tokens to 5,208.

[1] 10,000 manually translated images from ImageNet and 100,00 images from Al Jazeera website which paired article titles with featured images.

[2] Unpublished manually translated version of the Flicker8k dataset.

[3] 405,000 images from Middle Eastern news websites, pairing articles titles with featured images.

[4] A subset of the original dataset that includes 150 manually translated captions and 2111 captions translated using Google Translator.

[5] All images from the original dataset are used, each of the 5 captions are automatically translated and ranked by professional translators, only the top 3 captions are kept.

2. We replace the VGG19 encoder with newer pre-trained models such as EfficientNet [41] for better accuracy (80.1% classification accuracy versus VGG16's 71.3%) and MobileNetV2 [35] for faster performance (20.51 ms/image versus VGG16's 151.52 ms/image)[6].
3. We add an attention mechanism layer [6], [27] in the decoder which provides a context vector along with the feature map produced by the encoder. The additional context vector allows the RNN-based decoder to focus on specific regions of the image for each word, thus improving the caption's accuracy. Adding an Bahdanau-style [6] attention mechanism layer to the previous work by El Jundi et al. improved BLEU-1 score from 33 to 35.3 and BLEU-4 score from 6 to 7.8.
4. We implement an entirely new model based on the transformers [42] architecture.

The transformer-based model was our best performing model in terms of captions accuracy, which improved results over El Jundi et al.'s method on the Arabic Flicker8k dataset with a BLEU-1 and BLEU-4 score of 44.3 and 15.6 (compared to El Jundi et al.'s 33 and 6 on the same training/testing split).

1.4 Outline of the Thesis

Chapter 2 introduces additional background information on the prior work in image captioning, including the building blocks that go into creating the models such as Convolutions Neural Networks, Recurrent Neural Networks, Attention, and Transformers, as well as providing more details on the evaluation metrics and datasets that we used in our experiments. Chapter 3 desribes the data preprocessing pipeline and the implementation of our AIC models. Chapter 4 describes the training and experimentation setup, and analyzes the results of the experiments, providing information on captioning accuracy and runtime. Chapter 5 concludes our work and makes suggestions on how these experiments could be extended and discusses potential alternative methods for AIC. .

[6] Tested on an Jetson TX1 board. Source: Bianco et al. [8]

CHAPTER 2

BACKGROUND

We will go over the building blocks that go into building neural network-based image captioning models, the datasets used in training, the evaluation metrics used for evaluation, and prior work in this field in English, Arabic, and other languages.

2.1 Neural Network Architectures

Deep learning-based image captioning techniques require the use of many different neural network-based components and architectures. This section introduces the basic building blocks for neural networks and discusses more complex architectures (e.g., Convolutional Neural Networks, Recurrent Neural Networks, Attention Mechanisms, Transformers) that will be used in our work later on.

2.1.1 Feedforward Neural Networks

Feedforward neural networks, also called Multi-Layer Perceptrons, are the most basic form of deep learning models. The Feedforward network's goal is to approximate some function f that maps an input x to an output y. Feedforward networks are built using artificial neurons or perceptions. Feedforward neural networks are built by placing one or more perceptrons (also called neural units) in parallel to form a single layer and stacking multiple layers to form the network. Feedforward networks consist of an input layer which takes in the input data x, and an output layer which produces the output $\hat{y}$, and one of more layers in between called the hidden layers. Data flows forward (hence the name) from the input layer through the network, where the output of each layer becomes the input of the following layer until reaching the output layer and producing the output. Figure 2.1 illustrates an example of a simple feedfor-

ward network.

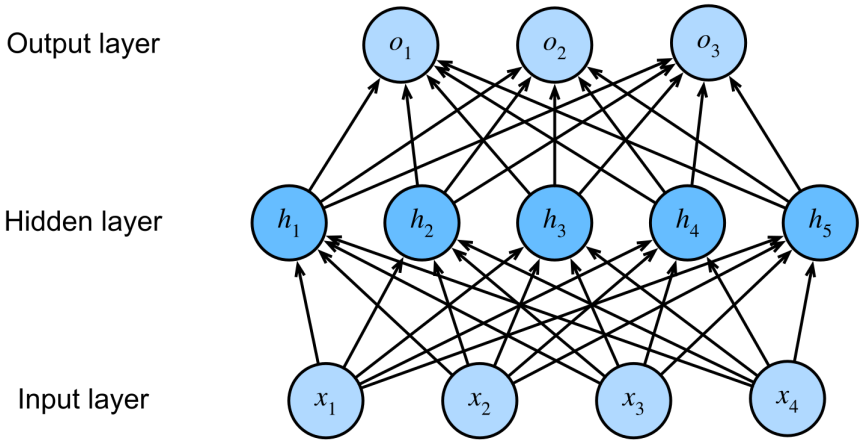


Figure 2.1: An example of Multi-Layer Perceptron with a single hidden layer. Image Source: d2l.ai [48]

A single perceptron calculates the weighted sums of the input(s) and passes the sum through an activation function. Equation 2.1 shows the equation of a perceptron where x_i is the i^{th} input and W_i is the corresponding weight, and σ is the activation function which introduces non-linear properties to the network allowing the perceptron to perform more complex operations. operations. Generally, the activation function is differentiable, which is crucial to updating weights based on backpropagation of error [34].

$$\hat{y} = \sigma(\sum_{i=0}^{n} W_i * x_i) \tag{2.1}$$

Some of the activation functions often used in deep learning are the Hyperbolic Tangent function (Equation 2.3), Rectified Linear Unit (Equation 2.2), and Softmax function (Equation 2.4).

$$ReLU(x) = max(0, x) \tag{2.2}$$

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

$$Softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} x^{z_j}} \tag{2.4}$$

2.1.2 Convolutional Neural Networks

In deep learning, convolutional neural networks (CNNs) are used to extract image features automatically. The layers of these networks are composed of convolutions instead of regular perceptions. These convolutions are two-dimensional tensors (also called kernels or filters) that learn to recognize different features in the image; shallower layers learn to recognize low-level features such as edges while deeper layers learn to recognize more high-level features such as faces and other objects (in practice, deeper filters typically learn more abstract features that are not immediately comprehensible by humans). In classification tasks, the output of the convolutions is passed to a classifier (a feedforward network).

CNNs take in an image in a tensor format as input, where the elements of the tensor represent pixel values. The kernels (i.e., filters) slides over the input image, multiplying regions of the image with the values of the filter in an element-wise multiplication and then summing the values of the output matrix. The process continues over the whole input tensor by passing the kernel over image regions from left to right and top to bottom. This process produces a feature map tensor that is used as input to deeper layers in the network.

In image captioning tasks, a convolutional neural network is typically used as the decoder in the encoder-decoder models (more details on this architecture are discussed in subsection 2.1.5). Image captioning researchers often use pre-trained image models (e.g., VGG16 by Simonyan et al. [36]) to save time and resource on training image models from scratch.

We will be using newer image models in our model's decoder, like MobileNetV2 by Sandler et al. [35], which uses bottleneck layers (layers with 1x1 kernels) [40] to reduce the number of channels in features maps with inverted residual connections [16]. Another model that we will use is EfficientNet-B2 by Tan et al., which carefully tuning the network's depth (the number of layers in the network), width (the number of kernels and each layer and their sizes), and resolution (the resolution of the input image), as well

as using similar techniques as MobileNetV2 to optimize size and performance [41].

MobileNetV2 delivers comparable results to VGG16 while being 2% the size, while EfficientNet-B2 provides accuracy on par with more complex models such as InceptionResNetV2 [39] while still being far more efficient. Table 2.1 breaks down the image models discussed in this section and compares them in terms of accuracy and size. Accuracy score is reported in Top-N format, meaning that the model's output is considered "correct" when the correct class is within top N probabilities produced by the model.

Table 2.1: Comparison of different image models trained on ImageNet [13] in terms of model size and accuracy. Data Source: keras.io [10]

Model	Size	Top-1 Accuracy	Top-5 Accuracy
VGG16 [36]	528 MB	71.3%	90.1%
MobileNetV2 [35]	14 MB	71.3%	90.1%
InceptionResNetV2 [39]	215 MB	80.3%	95.3%
EfficientNetB2 [41]	36 MB	80.1%	94.9%

2.1.3 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), first introduced by Michael I. Jordan [23] in 1979, are a special type of neural network that includes a hidden internal state that serves as a memory. RNNs are used in sequence-based tasks where the input or the output (or both) are a sequence (e.g. words in a sentence or data in a time series). Equation 2.5 show how RNNs work where t is the current time step, x_t is the input at time step t, h_t is the hidden state value at time step t, y_t the output for t, W, U and b are the parameters' weights, and σ_h, σ_y are the activation functions.

$$\begin{aligned} h_t &= \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma_h(W_y h_t + b_y) \end{aligned} \tag{2.5}$$

There are some problems with vanilla RNNs, such as being unable to learn long-term dependencies (remembering earlier time steps in longer sequences) and, more importantly, the problem of vanishing and exploding gradients [17] while training when the gradients that are calculated during backpropagation (also called backpropagation through time in RNNs [45]), where gradients are calculated back

through the time steps, the values either gets infinitesimally small to the point that the weight stop being updated (vanishing gradients), or exponentially large to the point that the weights overflow (exploding gradients).

2.1.3.1 Long Short-Term Memory

Introduced by Hochreiter & Schmidhuber in 1997 [17], LSTMs is a variation on RNN that allows the network to learn long-term dependencies by introducing a cell state and special gates that allows "forgetting" unimportant parts of the sequence and retain information significant to the desired output. Figure 2.2 and Equation 2.6 how the output and cell state are calculated within an LSTM cell where t is the current time step, i is the input gate, f is the forget gate, o is the output gate, $\tilde{c}$ is the memory cell value candidate, and c is the cell value, x is the input, and H is the hidden state value. W, U, b are the trainable parameters.

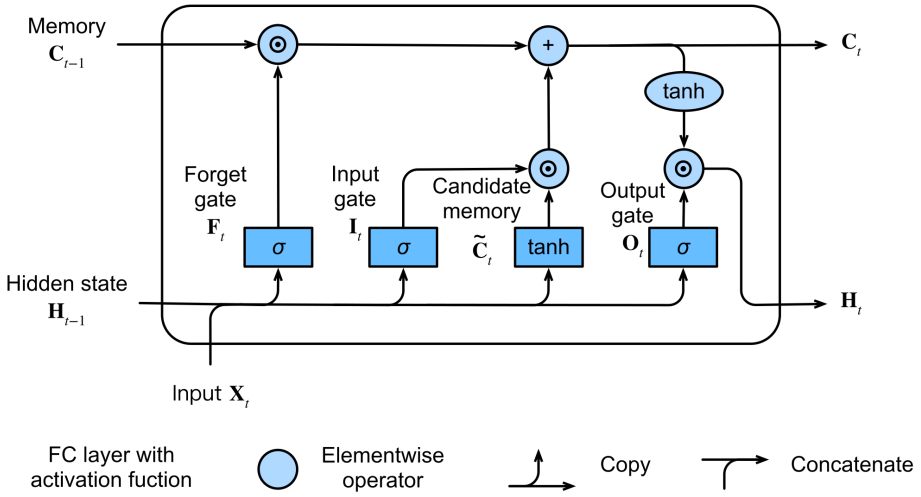


Figure 2.2: The internals of a Long Short-Term Memory cell. Image Source: d2l.ai [48]

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c_t} &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c_t}
\end{aligned}
\tag{2.6}
$$

2.1.3.2 Gated Recurrent Units

Developed by Chung et al. in 2014 [11], gated recurrent units (GRUs) are also a special variant of RNNs that, similar to LSTMs, implement internal gates to control the flow of sequence data. However, unlike LSTMs, GRUs do not maintain an internal cell state. Instead, it relies only on the hidden state to maintain a memory of earlier parts of the sequence. It also uses only two gates (reset and update gates) instead of LSTM's three gates (input, forget, and output). Figure 2.3 and Equation 2.7 how the output and cell state are calculated within an LSTM cell where t is the current time step, r is the reset gate, z is the update gate, $\hat{h}$ is the hidden state candidate, and h is the hidden state value, x is the input, and.W, U, b are the trainable parameters.

The advantage of GRU's simpler implementation is having fewer parameters making the network smaller in size and faster to train and execute while maintaining the benefits of learning long-term dependencies and reducing the effects of vanishing and exploding gradients.

This simplified implementation is not without its drawbacks, where it can struggle in datasets with longer sequences where LSTMs would typically produce more accurate results.

$$
\begin{aligned}
z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\
\tilde{h_t} &= tanh(W_h x_t + U_h(r_t * h_{t-1}) + b_h) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \hat{h_t}
\end{aligned}
\tag{2.7}
$$

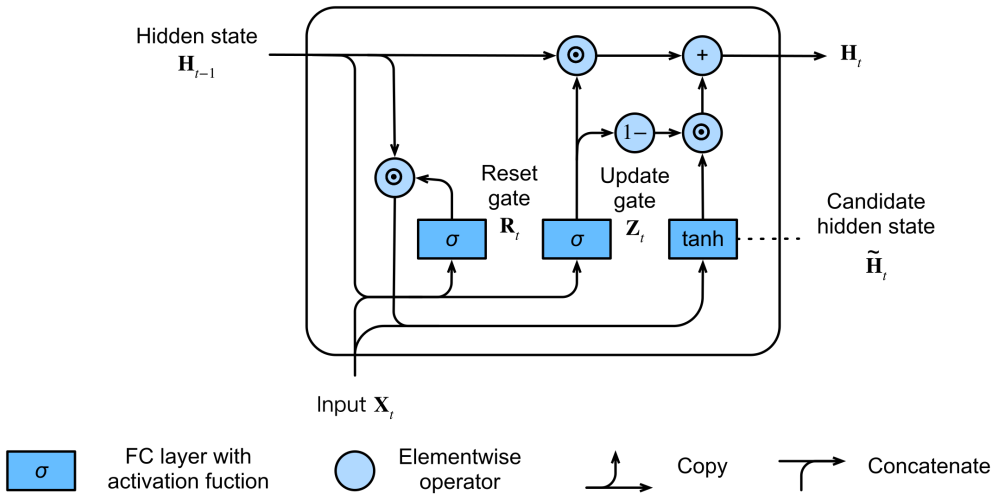


Figure 2.3: The internals of a Gated Recurrent Unit cell. Image Source: d2l.ai [48]

2.1.4 Word Embeddings

Word embeddings refer to the representation of words as vectors. In embedding space, words that are close together typically have a similar meaning. There are several algorithms that compute the word embeddings, such as the Bag of Words and Skip Grams models introduced by Mikolov et al. [28], [29]. These models are trained on massive text corpora to find a vector representing each word such that the cosine similarity between the vectors indicates the level of semantic similarity between the words they represent.

Word embeddings are often used in deep learning tasks that involve natural language to provide a more robust representation of words that encodes the words' meaning and the association between different words.

2.1.5 Encoder-Decoder Architecture

The encoder-decoder architecture is a neural network architecture originally introduced for the machine translation task by Sutskever et al. [38]. The main goal of the model is to handle variable-length inputs and outputs by encoding the input into an intermediary representation in a fixed-sized vector

which is then used by the decoder to generate the output.

Figure 2.4: A high-level diagram of the encoder-decoder architecture where both the encoder and decoder are neural networks. Image Source: d2l.ai [48]

In machine translation, both the encoder and decoder are RNN-based models that deal with sequential inputs and outputs of variable size (i.e., sentences). However, in image captioning, the encoder is replaced with a CNN-based model that extracts high-level image features. The intermediary representation, in this case, is a vector representation of the image's features which the RNN-based decoder uses to generate a sequence of words that describes the input image.

2.1.6 Attention

Attention in neural networks is first introduced in the context of machine translation by Bahdanau et al. [6] and Luong et al. [27] to solve an issue in encoder-decoder models using RNN-based models where the decoder cannot process long input sequences since only the last hidden state is used as the input to the decoder. Attention mechanisms offer a solution to this problem by calculating alignment scores between the encoder's output state (i.e. the representation of the input sentence in machine translation) and the decoder's hidden state in the current time-step. The produced alignment score represents how much each word in the input sequence is important to the word that's currently being generated. The alignment score is passed through a softmax function producing a vector with values between 0 and 1 which is then multiplied by the original input from the encoder to create a context vector where important parts of the input retain most of their original values and less important parts are now closer to 0. The context vector is then used by the following RNN layer instead of the original input from the encoder.

Attention is also used in image captioning where the input is an image instead of a sentence and the encoder is a CNN-based image model instead of an RNN-based language model. The alignment score in

image captioning models is calculated between image regions and the word being generated. Figure 2.10 describes in high-level how attention works in image captioning models.

There are mainly two methods to calculate the alignment score: The Bahdanau-style, also known is additive attention which is the most widely used in image captioning due to its more complex structure which allows it to learn better representations [6]. And Luong-style, which offers three different scoring techniques. Bahadanu attention is described in Equation 2.8 and the three Luong attention techniques are described in Equation 2.9 where W, W_c, W_d, W_e are learnable parameters (weights), h_e the the input from the encoder, and h_d is the decoder's hidden state.

$$score = W_c tanh(W_d h_d + W_e h_e) \tag{2.8}$$

$$\begin{aligned} dot &= h_e h_d \\ general &= W(h_e h_d) \\ concat &= W tanh(W_c(h_e + h_d) \end{aligned} \tag{2.9}$$

2.1.7 Transformers

The transformer architecture introduced by Vaswani et al. in 2017 aims to replace the recurrent based techniques in sequence modeling tasks such as machine translation by making extensive use of attention mechanisms. Transformers use a scoring technique similar to the Luong-style dot product attention described in Equation 2.9 with the addition of a scaling factor which is $\frac{1}{\sqrt{d_k}}$ length of the key vector (i.e., the hidden state). However, what gives the transformer architecture is the introduction of a concepts in attention called Multi-head Attention, where instead of performing a single attention function, the inputs of the attention are passed through h linear layers (h being the number of attention heads) in parallel, learning h different representations of the inputs and attending to them separately producing h attention scores. The scores are then concatenated and passed through a final linear layer. Multi-head attention allows the model to jointly attend to information from different representation subspaces at

different positions.

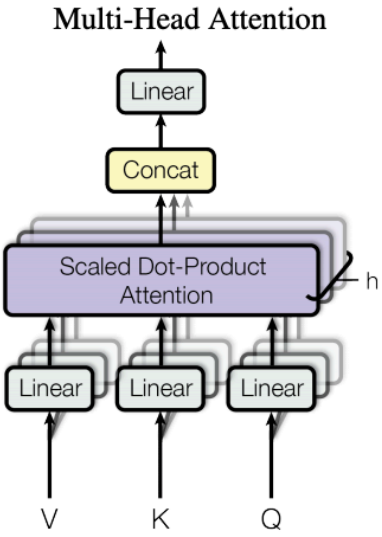


Figure 2.5: Visualizing the Multi-Head Attention which consists of h attention layers running in parallel. Image Source: Vaswani et al. [42]

Since the architecture does not make use of recurrence, it needs a different approach to provide information about the relative or absolutes position of tokens in the sequence, and it does so by using Position Encoding, which is an embedding layer that calculates embeddings for each position in the sequence instead of each words.

Using the concepts described above, Vaswani et al. propose an encoder-decoder model for machine translation. The encoder starts by calculating the word embeddings and positional encoding of the input sentences. The results are concatenated and passed to the first layer of a stack of 6 identical layers. Each layer consists of two sub-layers, a multi-head attention layer and a feedforward layer. With residual connection [16] around each of the multi-head attention and feedforward layers followed by layer normalization [5]. The linear layers in the Multi-head attention and the feedforward layer are of size 512 units.

The decoder is also composed of a stack of 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the encoder, residual connections around each of the sub-layers are used, followed by layer normalization [42]. The output of the final layer in the decoder stacked is passed to a feedforward layer with a softmax activation function which produces a probability distribution over the words in the vocabulary. Similar to the encoder, the linear layers in the Multi-head attention and the feedforward layer are of size 512 units, while the final feedforward layer is of size n units where n is the number of words in the vocabulary. In addition to the output of the encoder and similar to the hidden state in RNN-based decoders, the Transformer decoder takes in the previous outputs of the decoder as input, passing them through a word embedding and positional encoding layer and concatenating them before entering the decoder stack. The encoder-decoder model is visualized in Figure 3.6 where N = 6

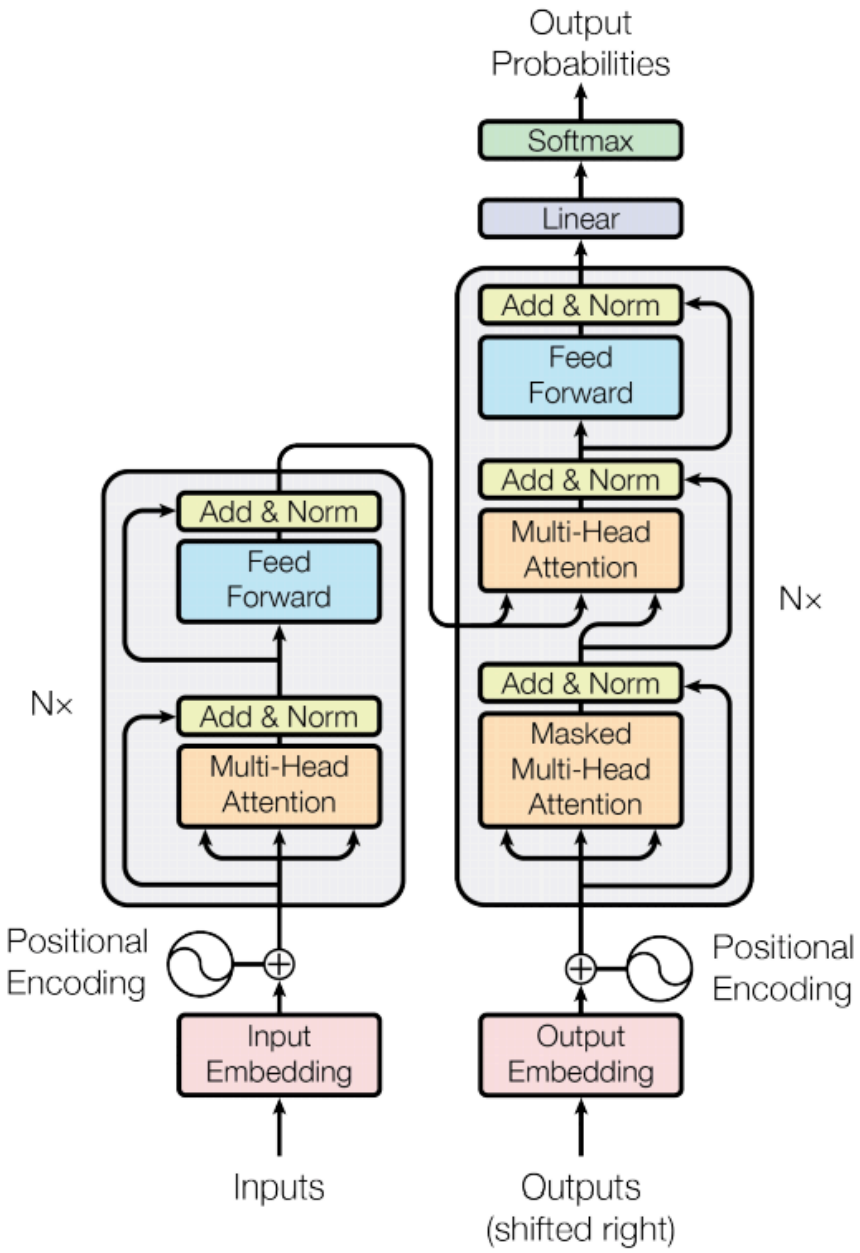


Figure 2.6: The Transformer model architecture by Vaswani et al. [42]

2.2 Prior Image Captioning Work

2.2.1 Evaluation Metrics

The process of evaluating the output of an image captioning model is somewhat similar to evaluating machine-translated sentences (i.e., comparing a generated sentence against a reference sentence(s)). That is why early on, image captioning researchers used metrics such as Bilingual Evaluation Understudy (BLEU) score [32] which estimates accuracy based on cumulative n-gram precision (n-grams being continuous sequences of *n* words). BLEU-1 being 1-gram precision which matches single words occurrences between the hypothesis sentence and reference sentences, and BLEU-2 being the cumulative precision of 1-grams and 2-grams. Figure 2.14 shows the BLEU 1-4 scores for sentences in English and Arabic

Another machine-translation metric that image captioning researchers use is the Metric for Evaluation of Translation with Explicit Ordering (METEOR) [7] which takes explicit ordering into account in addition to n-gram similarity. These metrics are language-agnostic and are used by image captioning researchers working on most languages

Newer metrics specialized in image captioning such as Consensus-based Image Description Evaluation (CIDEr) [43] and Semantic Propositional Image Caption Evaluation (SPICE) [2] provide a more robust evaluation. CIDEr and SPICE are used in newer research, but are not used in AIC literature due to not generalizing well to the Arabic language.

2.2.2 Datasets

Image captioning researchers mostly use two datasets to train and benchmark image captioning models.

The first dataset is the Common Objects in Context (COCO) [25], COCO contains data for several computer vision tasks such as object detection, image segmentation, and image captioning. The image captioning dataset contains 83k training images and 41k testing images. Each image has five different captions.

looking up at a clock tower with an iron railing around it.
A very tall clock tower with a minty roof.
A clock tower has a balcony around it.
the top of a tower with a balcony and clock.
A building with a balcony and a clock.

Figure 2.7: An example of an image and its five captions from COCO dataset

There is an Arabic version of the dataset translated automatically using Google Machine Translation API. However, the Arabic dataset is not manually validated, and in our exploration of a randomly sampled subset of 150 captions, we found that 46% of the translations have some grammatical and semantic errors to completely unintelligible translations. Table 2.2 breaks down the results in more detail.

Table 2.2: The results from our manual evaluation of a random sample of 150 captions from the Arabic COCO dataset.

Score Value	Score Meaning	Frequency	Percentage
5	The caption is accurate	135	54%
4	The caption has minor grammatical/semantic errors	77	30.8%
3	The caption has major grammatical/semantic errors	28	11.2%
2	The caption is barley understandable	9	3.6%
1	The caption is unintelligible	1	0.4%

Another popular dataset is the Flicker8k dataset [18], this data set contains 8,092 images with five captions for each image which results in 40,460 unique captions (compared to COCO's 413,195 unique captions). An Arabic version of this dataset was published by El Jundi et al., which contains machine-translated captions manually validated by Arabic translators. Only the best three captions are kept instead of the original five.

A black dog and a spotted dog are fighting.
A black dog and a tri-colored dog playing with each other on the road.
A black dog and a while dog with brown spots are staring at each other in the street.
Two dogs of different breeds looking at each other on the road.
Two dogs on pavement moving toward each other

Figure 2.8: An example of an image and its five captions from Flicker8k dataset

COCO is the preferable dataset due to its large size, allowing it to cover more visual concepts and scenarios. All English image captioning papers mentioned in the previous section use COCO. However, it is not trivial to manually translate or validate the machine-translated version of the COCO dataset due to its large size; that is why it is yet to be used by AIC researchers. Instead, researchers have used some version of the Flicker8K [15], [22], [31].

2.2.3 English Image Captioning

Deep learning based image captioning research started to gain momentum with the paper by Vinyals et al. [44] in 2015, this paper was the first neural network based approach for image captioning. The authors proposed an end-to-end system based on the encoder-decoder architecture [38] that was already in use in the machine translation field. The authors used a CNN-based decoder that serves as a feature extractor and an RNN encoder that serves as natural language generator to compose the caption. The end-to-end network works by passing an image to the encoder that extracts features from the images. The image features are then passed to the RNN decoder which generates a sequence of words that forms the predicted caption. This approach improves upon the previous state-of-the-art scores on many datasets (e.g., improving the BLEU-1 score on the Pascal dataset from 25 to 59 and setting the state-of-the-art BLEU-1 for the COCO dataset at 64.6).

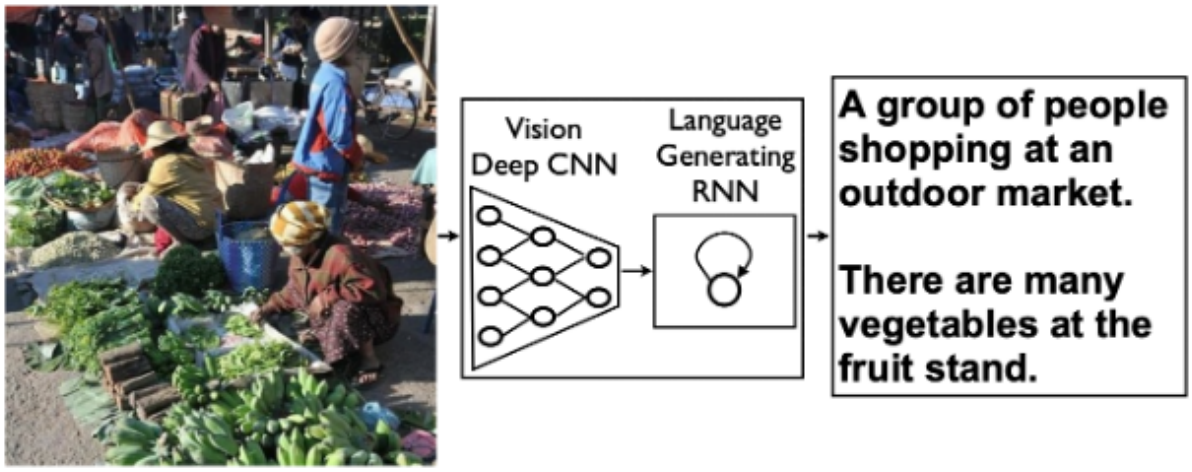


Figure 2.9: High-level view of the Encoder-Decoder Architecture for Neural Image Captioning by Vinyal et al.

In the same year, Xu et al. were working on a model with similar architecture to Vinyal et al.'s with the addition of an attention layer on top of the RNN-based decoder. Attention mechanisms previously were used in encoder-decoder-based machine translation tasks [6]. The addition of the attention mechanism provides additional context to the decoder by identifying specific regions in the image feature maps that contains salient objects to help the decoder focus on relevant details while composing the caption sequence. Figure 2.10 shows a high-level illustration of this approach. This approach made some improvement over the previous state-of-the-art score set by Vinyal et al. [44], increasing BLEU-1 score from 64.6 to 70.7 and METEOR score from 23.7 to 23.8 on the COCO dataset.

In 2018, Anderson et al. [3] introduced an image captioning model that used two steps of attention, bottom-up and top-down. The bottom-up attention used an object detection model called Faster R-CNN [33] to detect objects and salient regions in the image, Figure 2.11 shows the output of this step in comparison with the uniform feature map grid that regular CNN models produce. The top-down attention step focuses on the regions/objects that best correlate with every step of the sequence that is

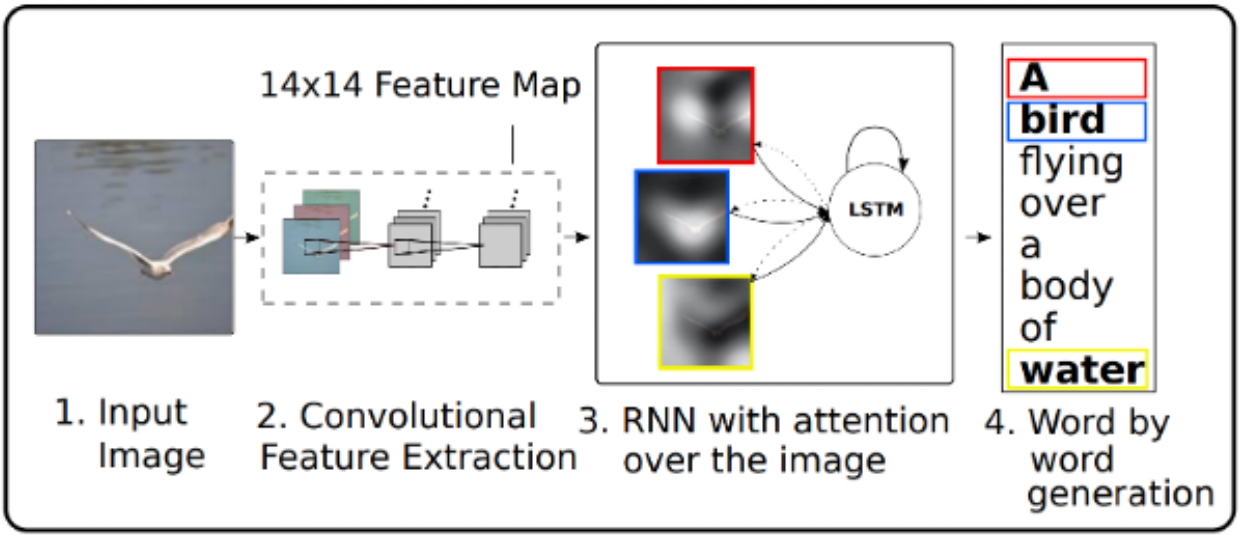


Figure 2.10: Overview of Neural Image Captioning with Attention architecture by Xu et al. Step 3 and 4 show how the attention mechanism works.

being generated. This approach made significant improvements to the previous state-of-the-art score set by Xu et al., improving the BLEU-1 score from 70.7 to 79.8 and METEOR score from 23.9 to 27.7 on the COCO dataset.

Corina et al. published a paper in 2020 [12] which introduced an image captioning model that leveraged the recent advancement in Transformers [42] that had gained a lot of momentum in sequence modeling tasks after its introduction in 2017. The transformer-based architecture improves both the image encoding and the language generation steps. It learns a multi-level representation of the relationships between image regions integrating learned apriori knowledge and uses a mesh-like connectivity at the decoding stage to exploit low- and high-level features. This approach sets the current state-of-the-art on most benchmark scores, with a BLEU-1 score of 80.8 and a METEOR score of 39.1.

In late 2020, Hu et al. [20] surpassed human-level performance in Image Captioning tasks. The model is also based on the Transformer architecture, but its strength lies in its ability to recognize and

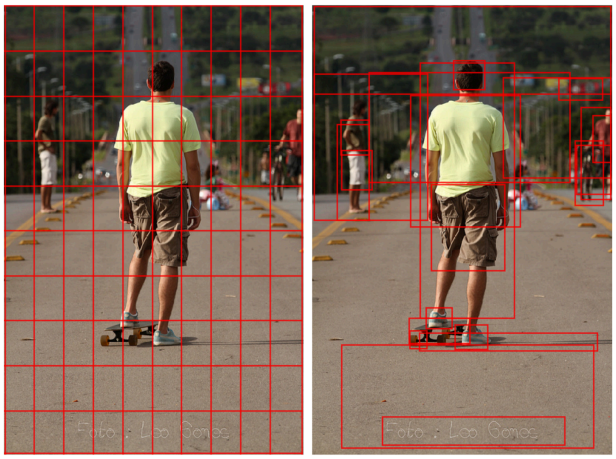


Figure 2.11: Left: Uniform image feature grid produced by the CNN-decoder that is typically used by attention mechanisms. Right: The output produced by the bottom-up attention step used by Anderson et al [3]. which enabled attention to be calculated at the level of objects and other salient regions of the image.

caption novel objects that are not seen in the image/caption pairs provided by the captioning datasets; this is achieved by pre-training the model on large-scale datasets with abundant tags. The model is pre-trained solely on the image-tag pairs. After pre-training, the model is fine-tuned on the image captioning dataset.

The score on the COCO dataset didn't surpass the state-of-the-art score set by Corina et al. [12]. The authors attribute that to the small number of visual concepts present in the COCO dataset, thus diminishing the benefit of learning an extensive visual vocabulary that the pre-training step does in this model.

2.2.4 Arabic Image Captioning (AIC)

AIC research is yet to produce results on par with the English counterparts; this is due to the morphologically complex nature of the Arabic language that is illustrated in Figure 2.12, but more importantly, due to the lack of standardized and high-quality benchmark datasets such as the ones available for the English language.

Figure 2.12: A word that illustrates the morphological complexity in Arabic and how it translates to English.

The first attempt at neural-based image captioning in Arabic was in 2017 by Jindal Vasu [21]. Vasu worked on composing captioning using root words only instead of full words to avoid the morphological complexity of the Arabic language. He used a pre-trained model to extract image features, then passed the features to a deep belief network pre-trained by Restricted Boltzmann Machines to predict different root words associated with the image fragments and extract the most appropriate words for the image. And finally, the words are arranged to form a sentence using dependency tree relation. Vasu used manually translated ImageNet class names (10,000 images) and images associated with news articles from Al-Jazeera news website (100,000 images), creating a total of 110,000 images, 80,000 used for training and 30,000 used for testing. The BLEU-1 score reported on this on the dataset is 34.8.

A year later in 2018, Vasu [22] updated his work to use an LSTM instead of a deep belief network, but his work was still limited to using root words. Vasu used two different datasets for this paper. The first one is the Flicker8k dataset with manually written captions. However, the exact number of images used, and number of captions written for each image isn't mentioned. We will assume that the entire dataset is used and one caption per image is written instead of the original five captions per image). The second dataset is 405,000 images gathered from middle eastern news websites, pairing article images and titles. This updated approach reported a BLEU-1 score of 65.8 on the Flicker8k dataset and 55.6 on the

middle eastern news websites dataset. The BLEU-1 score did improve the previous results (from 34.8 to 65.8/55.6), but these results were reported on entirely different datasets so the exact amount of improvement cannot be measured.

In 2018, Al-Muzaini et al. [31] proposed an image captioning model that uses a merge model for generating Arabic captions. The overall architecture is composed of an LSTM-based model that encodes linguistic sequences and a CNN-based image model that extracts image features. The outputs of both these models is concatenated into the decoder that composes the caption. The authors created a dataset using 1,166 image (5 captions each, totaling 5,830 captions), 150 images from Flicker8k translated manually and 2,111 translated automatically suing Google translator (also 5 captions each, totaling 750 and 10,555 captions respectively). 2,400 images were used for training, 411 for validation, and 616 for testing. This approach reported a BLEU-1 score of 46, which is the highest reported BLEU-1 score in AIC for complete sentences (non-root word-only approaches). However, neither the code nor the datasets are published, making it impossible to verify the results or test other models against this approach using the same dataset.

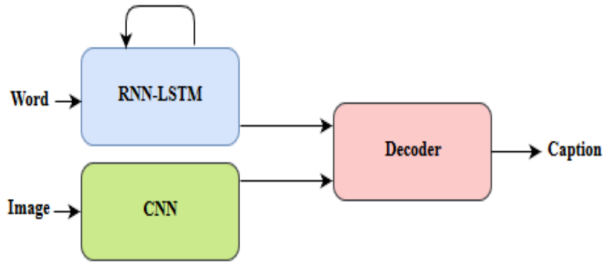


Figure 2.13: High-level overview of the merge model architecture proposed by Al-Muazini [31]

In 2020, El Jundi et al.[15] used an encoder-decoder architecture similar to the one introduced by Vinyal et al. [44]. They used a pre-trained VGG19 [36] model as the CNN decoder and a custom LSTM-based decoder. The dataset used for this work is a version of the Flicker8k dataset translated using Google Machine Translation API. The translated captions are then validated and ranked by professional Arabic translators, and only the top-3 captions out of the overall 5 captions are kept in the dataset. 7,292 out of the 8,092 images were used for training and the remaining 800 images were used for testing. The pro-

posed model achieved a BLEU-1 score of 33 on that dataset.

On paper, the state-of-the-art score in AIC (for non-root word only approaches) is El Muazini et al.'s BLEU-1 score of 46 and BLEU-4 score of 8 (tied with El Jundi et al's score). [31]. However, the scores reported in the papers discussed above are not a reliable source of identifying the current state-of-the-art as all papers (except for El Jundi et al.'s) used a different dataset that isn't publicly available for other researchers to benchmark against. However, El Jundi et al. open-sourced their dataset as well as their source code, which means their work is reproducible and it can serve as a benchmark for future research.

2.2.5 Other Languages

Researchers have worked on image captioning for other languages. Mishra et al. [30] worked on Hindi image captioning by implementing a similar architecture to Xu et al. [47] using an encoder-decoder architecture with attention. Mishra et al. explored many attention mechanisms such as spatial, visual, Bahdanau-style, and Luong-style attention. They achieved the highest score on the Bahdanau-style attention (67.0 BLEU-1 on a Hindi version of the COCO dataset).

Lu et al. [26] worked on Chinese image captioning. Lu et al. also used a similar architecture to Xu et al., with the difference of using a bidirectional LSTM. This approach achieved a BLEU-1 score of 78.5 on the AI Challenger Dataset [46], a Chinese dataset that, like COCO, contains data for several computer vision tasks including image captioning.

2.2.6 Discussing the Relatively Low BLEU Scores of AIC

Looking at Table 1.1 we can clearly see that AIC models produce significantly lower score than their English, Hindi, and Chinese counterparts. Researchers [9], [15], [31] have attributed this the morphological complexity of the Arabic language; a single Arabic word typically have several attached attached articles (as illustrated in Figure 2.12) making the sentences have a much lower number of words than most language and thus making error penalties much higher in n-gram similarity-based metrics like BLEU. This issue also explains why the results by Vasu [21], [22] which dealt with root words only have more compa-

rable scores to other languages. Figure 2.14 shows an example of two sentences in Arabic and English where the only difference between the sentences is using a different gender pronoun. The bias of the BLEU score in Arabic is discussed in a paper by Bouamor et al. [9] where they verify the bias by comparing BLEU scores against human judgment scores as well as introducing a new metric that provides partial credits for stem and morphological matching of hypothesis and reference words. We also experimented with training the same model once on the Arabic Flicker8k dataset and once on the original English Flicker8k dataset, the results are discussed later in section 4.4.

Figure 2.14: An example to illustrate how similar mistakes produce significantly lower BLEU scores in Arabic captions versus English captions.

Another reason for lower scores in AIC is the smaller size of data available for AIC research. The best available resource for AIC is El Jundi et al.'s Arabic Flicker8k dataset which contains an overall 8,092 images and 24,276 captions (3 captions per image) with no dedicated test/validation dataset. The COCO dataset on the other hand has 82,783 images 413,915 captions (5 captions per image) in the training dataset

with dedicated datasets for testing and validation. An Arabic COCO dataset does exist but it is translated using Google's Machine Translation API and is not manually validated, our analysis of a sub-sample of the dataset (results broken down in detail in Table 2.2) showed that only around half of the captions are accurate while the rest range from having minor grammatically mistakes to being barley understandable rending the dataset unusable in research.

CHAPTER 3

METHODOLOGY

3.1 Data Preprocessing

Before building and training the model, we need to prepare the dataset for consumption by the neural network. We will be using the Arabic Flicker8k dataset published by El Jundi et al. [15] which is based on the English Flicker8k [18] dataset. This dataset consists of 8,092 images with three captions for each image, totaling 24,276 unique image-caption pairs with a total vocabulary size of 11,384 unique tokens. Figure 3.1 shows a couple of samples from the dataset along with the original English captions.

-Black dog in the water
-Black dog runs through the water with ball in its mouth
-Black laborador with ball in his mouth coming out of the surf
-Large black dog with something in its mouth is coming out of pool of water
-The black dog waring red collar is running through the water with ball in its mouth

-كلب اسود في الماء
-كلب اسود يمر عبر الماء مع كره في فمه
-كلب اسود كبير مع شيء في فمه يخرج من بركه من الماء

-Boy in an orange shirt and red hat is standing next to girl with white shirt and blue pants in the sand next to the ocean
-child and grownup each wearing hats play by the ocean
-woman and boy playing on beach
-Two children are running towards the ocean on beach
-Two children on the beach

-امراه وفتي يلعبان علي الشاطئ
-يركض طفلان نحو المحيط علي الشاطئ
-طفلان علي الشاطئ

Figure 3.1: Samples from the Arabic Flicker8k dataset and the original English captions [15], [18]

3.1.1 Image Preprocessing

The dataset's images are in RGB format (there is no alpha channel) and come in different sizes. Since we are working with pre-trained image models, we will follow the same preprocessing steps that the models use since the model structure is already defined. The preprocessing steps are shown below.

1. Resize the images into one fixed size tensors of shape $(224, 224, 3)$. Elements in the tensor represent pixel values of the image. The color channels are always three since we are working with RGB images.
2. Pixel values are scaled between -1 and 1. This step is required because it was part of the preprocessing steps for data used to train both EfficientNet [41] and MobileNetV2 [35].
3. We then pass the resulting tensors through the pre-trained CNN models [35], [41] to extract the feature maps, and we cache them on disk. This step saves us time during training since the model is pre-trained and the weights won't be updated during training, so we do this step once instead of having to do it in every epoch.

3.1.2 Text Preprocessing

Text preprocessing is required to convert the words sequences with varying sizes into a uniform tensor that a neural network can consume. The following are the preprocessing steps taken (the first three are specific to the Arabic language, the rest are general steps applied to text preprocessing in neural network tasks).

1. Normalize Arabic words by replacing letter variants with the original form (e.g., convert "إ أ آ" to "ا".
2. Remove diacritics.
3. Segment words to alleviate some of the morphological complexity using the AraBERT segmenter [4], this is discussed in more detail in subsubsection 3.1.2.1
4. Remove punctuation.
5. Add "<start> " to the beginning of each sentence. This will be used as the seed to kick start the recurrent word generation model (the decoder).

6. Add " <end>" to the end of the sentence. The decoder will produce this token to signal that the sentence has ended thus triggering the generation loop to be stopped.
7. Tokenizing the words by assigning a unique index number for each word, creating a vector or integer for each caption.
8. Unifying the caption vectors' length to a maximum of 20 tokens, cutting off captions that are longer, and padding the captions that are shorter with zeros. The decision to use a cut-off of 20 tokens (24 tokens in AraBERT variants due to segmentation which leads to more tokens per sentence.) is based on analyzing the dataset captions where the length of the sequences has a mean of 10.8, a median of 10, and a standard deviation of 4.2. Figure 3.2 show a histogram of sequence lengths in the dataset.

The output of the preprocessing steps is a Tensor of shape (C, S), where C is the number of captions and S is the maximum caption length in the dataset.

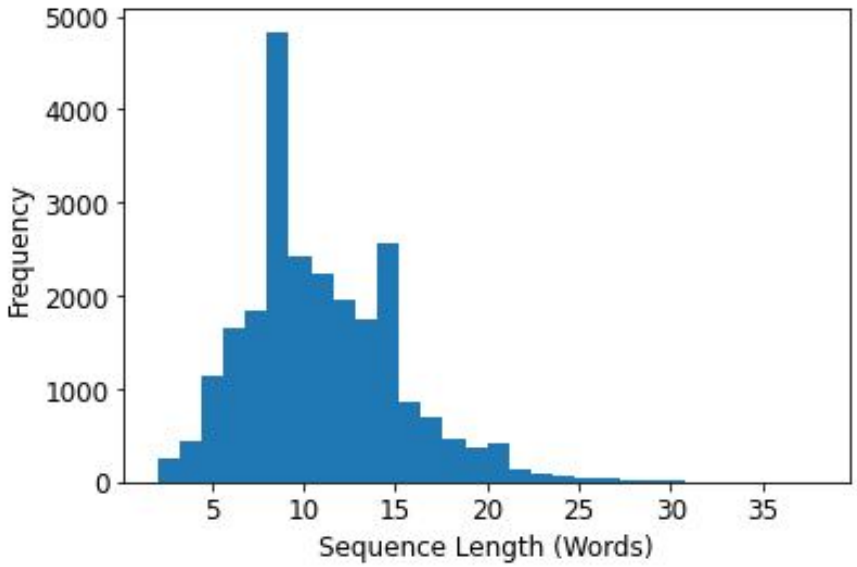


Figure 3.2: Histogram of sentence length in the Arabic Flicker8k Dataset.

3.1.2.1 AraBERT Segmenter

AraBERT [4] is a multi-purpose language model developed by Antoun et al. and based on the BERT model [14]. AraBERT provides several NLP tools, including an Arabic word segmenter. The segmenter

uses the Farasa segmenter [1] under the hood while using AraBERT to identify the root/non-root words and apply a special pre/post-fixed symbol for the segmented articles, which allows users to re-attach the segments to the original word. This feature is handy since our model would output segmented words, and having these pre/post-fixed symbols allows us to put the sentence together and produce a complete sentence. Figure 3.3 shows a couple of example of the sentences before and after applying the AraBERT segmenter.

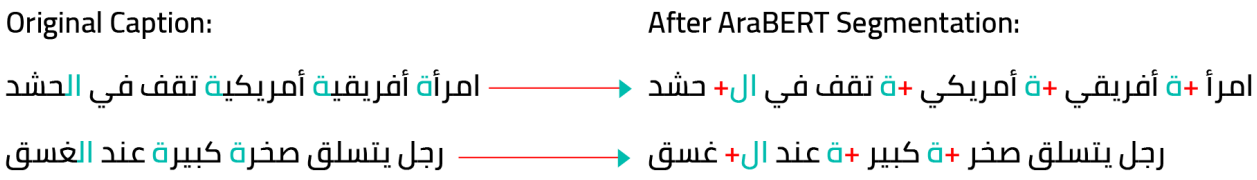


Figure 3.3: Example of sentences before and after passing through the AraBERT segmenter, the additional "+" sign represents where the segment was cut off which allows us to de-segment the sentence later.

Using this technique reduced the number of tokens in the vocabulary from 10,396 to 5,208 tokens. This is largely due to three articles. The definite article " ال " (equivalent to *the* in English) which in Arabic is attached to the word, so most words appear in the vocabulary twice in the definite and indefinite forms. Another one is the " ة " article which is the feminine article in Arabic is applied to most words when used with a female subject or object. And finally, the " و " article (equivalent to *and* in English). This article is attached to the word that comes after it. Figure 3.4 shows the top 10 most frequent tokens in the dataset before and after applying the AraBERT segmenter.

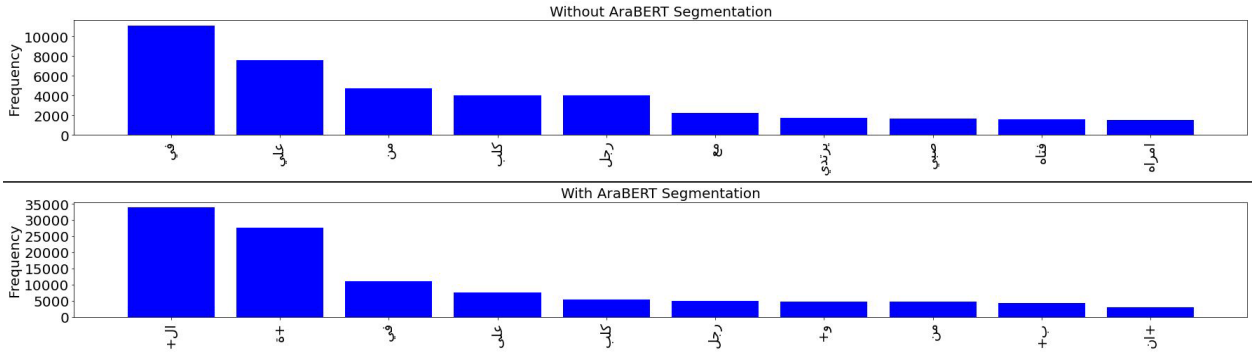


Figure 3.4: The top 10 frequently appearing words in the dataset before (Top) and after (Bottom) applying the AraBERT Segmenter.

3.2 Models Definition

All of our proposed models use an encoder-decoder architecture [38] similar to previous work (El Jundi et al., and Xu et al. [15], [47]). We propose three different model architectures. These models are summarized below and discussed in detail in upcoming sections:

- **LSTM Model**: MobileNetV2 [35] pre-trained on ImageNet is used as the encoder, and an LSTM-based model with an attention layer [6] is used as the decoder.
- **GRU Model**: MobileNetV2 [35] pre-trained on ImageNet is used as the encoder, and a GRU-based model with an attention layer [6] is used as the decoder.
- **Transformers Model**: EffeceintNet [41] pre-trained on ImageNet is used as the encoder, and a Transformers-based model is used as the decoder.

The GRU and LSTM-based models with MobileNetV2 are designed to deliver good accuracy while maintaining a lightweight model with fast performance. In contrast, the Transformers-based models focus on delivering captions with state-of-the-art accuracy while having significantly slower running time and larger model size. All models are built with the Keras framework [10] using sub-classing, which offers granular control over the model architecture and dataflow.

3.2.1 LSTM Based Model

Our first model is an encoder-decoder model that uses a MobileNetV2-based encoder [35] and an LSTM-based decoder [17] and consists of the following layers:

1. Encoder: This part of the network runs once when the generation starts. It takes in an image and produces a vector of size 256 that describes that image.
    1.1. The input image of size 224x224x3 (which is the default input size for our pre-trained model) goes through a MobileNetV2 [35] model with no top (i.e., no classification/fully connected layers, only the convolutional layers) pre-trained on the ImageNet [13] dataset, the model extracts image features from the image and produces an image map.
    1.2. The image features pass through a feed-forward layer with a ReLU activation function, the layer is trainable and consists of 256 units and produce
2. Decoder: This part of the network runs recursively for each generated word. It starts with the token "<start>" and ends when the token "<end>" is produced or when the maximum sequence length (20) is reached.
    2.1. A word embedding [29] layer with an embedding dimension of 256 (i.e., each word is represented in a vector of length equal to 256) takes the previously produced word as input (when first starting generation it takes in the "<start>" token).
    2.2. A Bahdanau-style Attention layer [6] with a size of 512 units, the attention layer takes in the image embeddings produced by the decoder concatenated with the current hidden state of the RNN-based layer (the LSTM layer in this case) and calculates a score for the image regions between zero and one based on how relevant the region is to the current hidden state. The attention scores are then multiplied by the image features vector produced by the decoder to create a context vector.
    2.3. The context vector and the output of the word embeddings layer are concatenated and passed to an LSTM layer of size 512 recurrent unit.

2.4. The output of the LSTM then goes through a Fully Connected layer of size 512 and ReLU activation function. This layer uses Dropout with a drop rate of 0.5. Dropout randomly turns off the layer's units (i.e., make their output equal to zero) with a probability equal to the drop rate.

2.5. Finally, the output layer is a feed forward layer with a number of units equal to the vocabulary size (8,000) with a softmax activation function. The output of this layer is a probability distribution of all words in the vocabulary, and the word with the highest probability is chosen as the output of this iteration of the recurrent decoder.

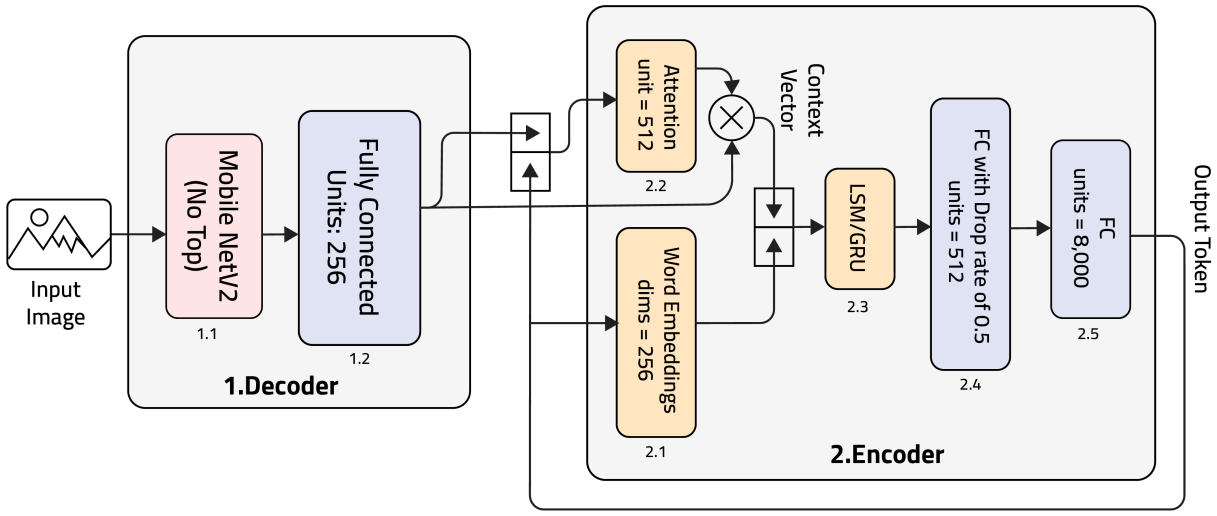


Figure 3.5: The Architecture of our LSTM-based model.

3.2.2 GRU Based Model

This model is almost identical to the previous model discussed in subsection 3.2.1, the only difference is replacing the LSTM layer described in item 2.3 with a GRU layer of equal size. GRUs are more efficient and smaller in size than LSTMs because they have one less gate and thus fewer parameters [11]. The difference between them is discussed in more detail in subsection 2.1.3. We experiment with replacing the LSTM layer with a GRU to evaluate if the accuracy difference between them (if any) is worth the extra complexity and performance drawbacks.

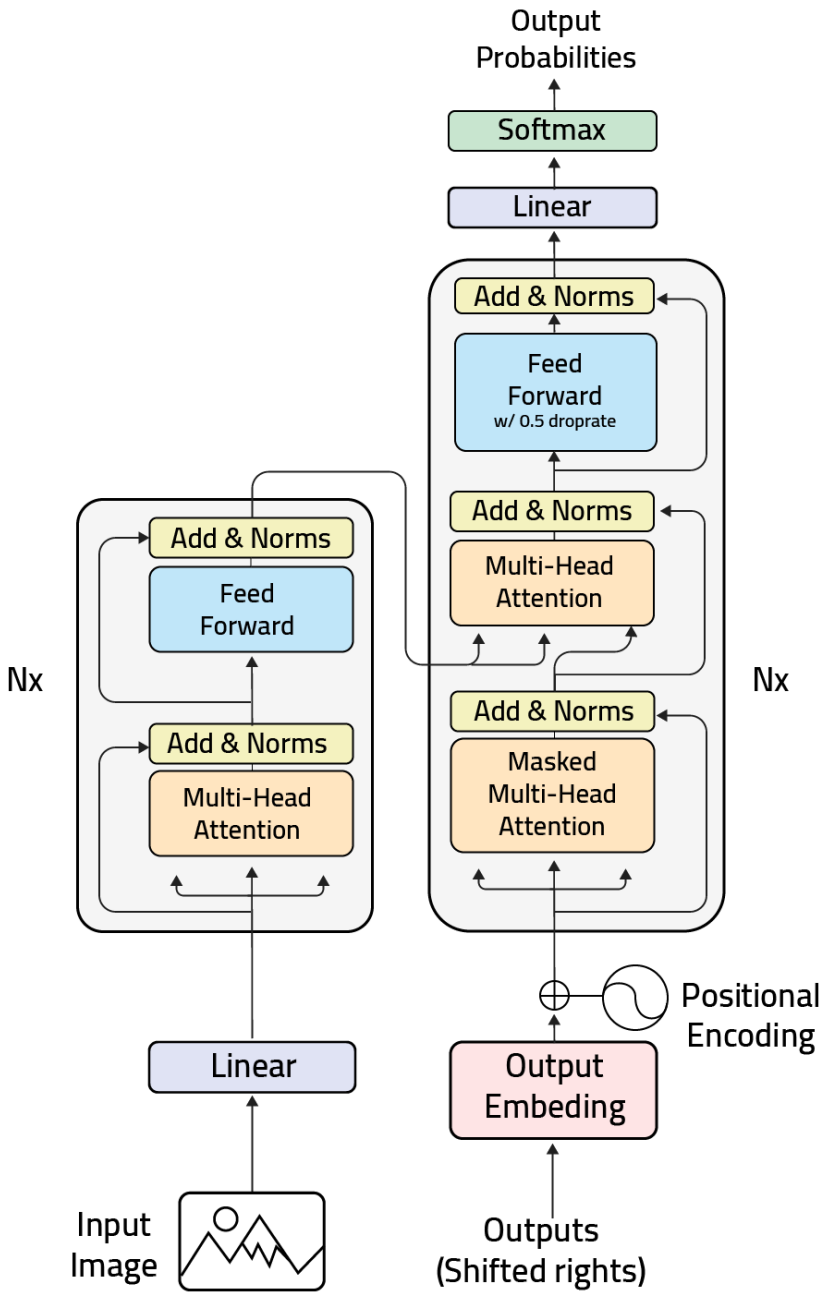


Figure 3.6: Our transformer model, a modified version of the original transformer model developed by Vaswani et al. [42]

3.2.3 Transformers Based Model

This model uses the EfficientNet-B2 [41] model pre-trained on ImageNet [13] as the feature extractor, and uses a modified version of the encoder-decoder transformer architecture proposed originally by Vaswani et al. [42] and discussed in detail in subsection 2.1.7. The modifications include modifying the decoder that would take in the feature map output from the EfficientNet-B2 instead of taking in a sequence of words. The transformer encoder uses a Fully Connected layer of size 512 units and ReLU activation function (similar to the image embeddings layer our GRU/LSTM based models) followed by a Multi-Head Attention layer with six attention heads, and finally, a Layer Normalization [5] layer that applies a transformation that maintains the mean activation of the previous layer within each sample close to 0 and the activation standard deviation close to 1. Layer Normalization is a technique commonly used in transformers architecture to reduce the effects of the variance in distribution between different training samples, which causes the network to take longer to converge.

The decoder uses a similar architecture to the one introduced by Vaswani et al. [42] using six attention heads in each Multi-Head Attention sub-layer with the addition of a Dropout [37] layer with a drop rate of 0.1 for the inputs of the first Multi-Head Attention layer. Another Dropout layer is applied to the outputs of the third and final Multi-Head Attention sub-layer. Both the encoder and decoder use a single transformer layer instead of the stack of six layers used by Vaswani et al.

3.3 Conclusion

Of the three model architectures discussed, the first two (using an MobileNetV2-based encoder and a LSTM/GRU-based decoder) are similar to previous AIC models such as the one introduced by El Jundi et al. [15] with the addition of a more modern image model and an attention mechanism layer. The third model is an entirely new model based on the Transformers architecture. As will be seen in the next chapter, the Transformer-based model generates the best results in terms of captioning accuracy. In contrast, the LSTM/GRU-based models generate relatively lower quality captions but do so significantly faster than the transformer-based model.

CHAPTER 4

EXPERIMENTS SETUP AND ANALYSIS

4.1 Experiments Setup

To train the models introduced in chapter 3, Categorical Cross Entropy is used as the loss function, and the Adam optimizer [24] is used for updating the network's weights. The following values are used for the optimizer's parameters.: $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e^{-7}$. Here, α is the learning rate, β_1 and β_2 are the exponential decay rate for the first and second moment estimates, and ϵ is a constant value that prevents the denominator in the weight update formula from becoming zero. The complete Adam algorithm by Kingma et al. [24] is described in Figure 5.1.

The LSTM and GRU-based models are trained over 20 epochs; training over 20 epochs causes the testing loss to increase, which indicates that the model is overfitting the training data. The Transformers-based model is trained over 30 epochs due to its significantly more complex architecture and higher number of trainable parameters. The model starts to improve slowly after the 25th epoch, but there are no signs of overfitting, so training for more epochs might be beneficial but is very time-consuming. Both models are trained using mini-batching with batch size of 64.

The three models described in the previous chapter were trained on the Flicker8k dataset, using 90% of the images (7,281 images) for training and 10% (810 images) for testing. The results below are reported on the testing dataset unless stated otherwise. We also trained the best-performing model using 4-fold cross-validation to ensure that the results are stable. The results are discussed in the following sections.

4.2 Accuracy Analysis

The transformer-based model achieved the best results on the BLEU metric, while the LSTM and GRU models achieved lower scores that are fairly similar. The models trained on data which included using AraBERT segmenter consistently produced better results. Figure 4.1 show a comparison of the BLEU-1 and BLEU-4 scores achieved by our models. Figure 4.2 compares the BLEU-1 results evaluated on the training an testing data which shows minimal overfitting due to the use of dropout layers throughout our models. The primary results are given below

- The LSTM-based model achieved a BLEU-1 and BLEU-4 score of 38.3 and 8.2 with the AraBERT segmenter, and 35.1 and 8 without the AraBERT segmenter.
- The GRU-based model achieved a BLEU-1 and BLEU-4 score of 37.6 and 7.9 with the AraBERT segmenter, and 35.3 and 7.8 without the AraBERT segmenter.
- The Transformer-based model achieved a BLEU-1 and BLEU-4 score of 44.3 and 15.7 with the AraBERT segmenter, and 42.7 and 15.2 without the AraBERT segmenter.

All of our models exceed the previous scores set set by El Jundi et al. [15] on the Arabic Flicker8k dataset with the same training/testing split. The scores reported by El Jundi et al. were a BLEU-1 score of 33 and BLEU-4 score of 6. Table 4.1 breaks down the scores of the previous work along with our models.

Table 4.1: BLEU score comparison of our models and previous work.

Model	BLEU-1	BLEU-4
El Jundi et al. [15]	33	6
Ours: Transformers + AraBERT	**44.3**	**15.7**
Ours: Transformers	42.7	15.2
Ours: LSTM + AraBERT	38.3	8.2
Ours: LSTM	35.1	8
Ours: GRU + AraBERT	37.6	7.9
Ours: GRU	35.3	7.8

In order to make sure that the results are stable, we performed 4-fold cross validation on the best performing model (Transformer + AraBERT segmentation). The average of the folds was a BLEU-1 score

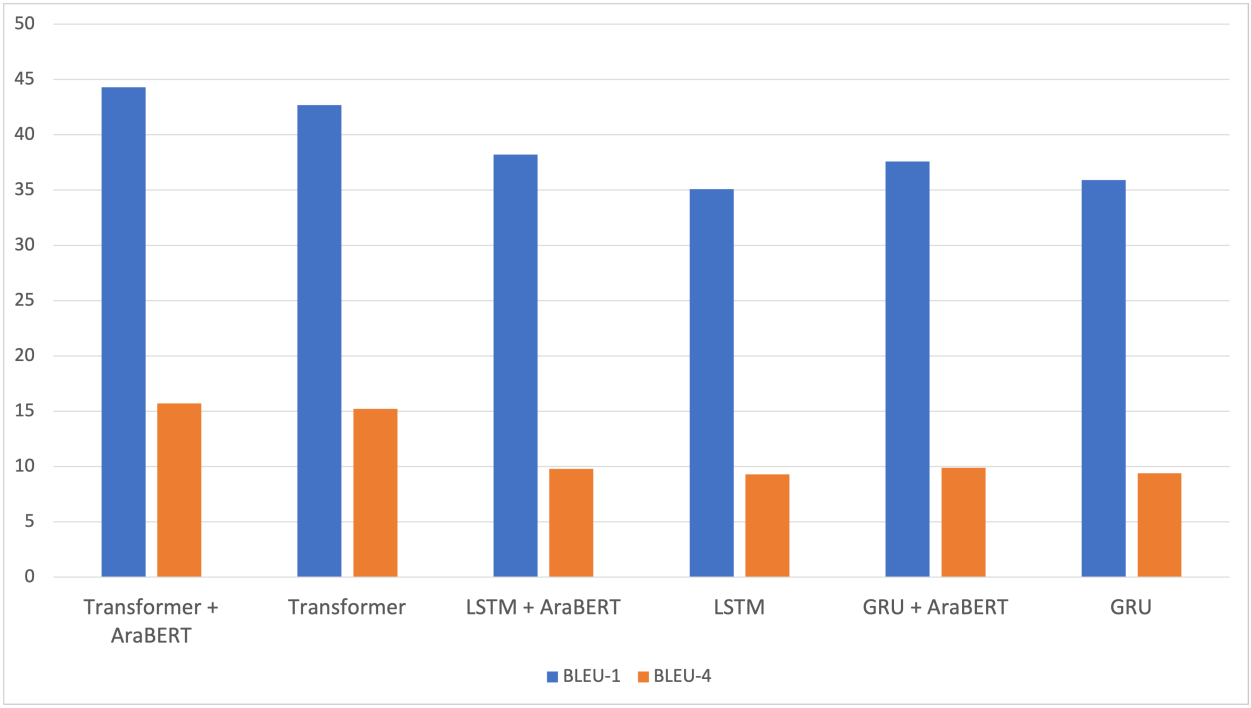


Figure 4.1: BLEU Scores of our models on the Arabic Flicker8k Dataset.

of 43.8 and a BLEU-4 score of 15.3. Table 4.2 breaks down the cross validation results per fold.

Captions generated using the transformers-based architecture are mostly accurate, with some captions having minor and sometimes major grammatical errors, and the occasional entirely unrelated captions. Figure 4.3 shows a histogram of BLEU-1 score of the validation dataset, with most of the captions producing BLEU-1 scores between 20-60, with some captions having a score between 80 and 100.

Table 4.2: The results of the 4-fold cross-validation on our Transformers + AraBERT Segmenter model.

Fold	BLEU-1	BLEU-4
Fold 1	44.3	15.7
Fold 2	43.1	15.3
Fold 3	44.9	15.4
Fold 4	42.8	14.9

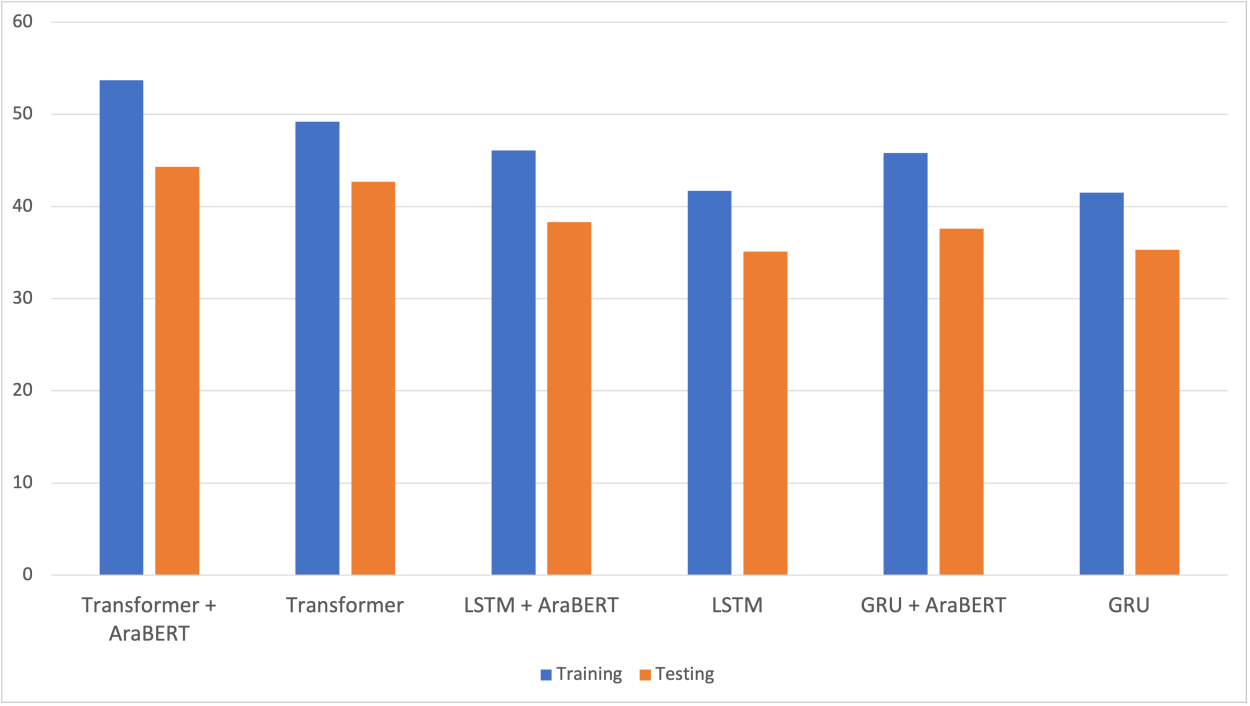


Figure 4.2: BLEU-1 scores of our models for the training and testing set.

4.3 Performance Analysis

Captioning speed is one of the aspects that we focused on while building our models. The RNN-based models are significantly faster in captioning speed and model size compared to the transformers-based models, with the GRU-based model being slightly faster than the LSTM-based one. Table 4.4 breaks down the speed and size of each of our models. The models that incorporate AraBERT segmentation have smaller sizes due to using a smaller vocabulary (5,000 tokens instead of 8,000), which affects the number of parameters in both the embedding layer and the final output layer.

The speed difference between transformers-based models and RNN-based models is significant and becomes even more significant when the model is used in lower-end devices such as mobile phones. If the models were to be used in lower-end devices we recommend using a client-server architecture where the

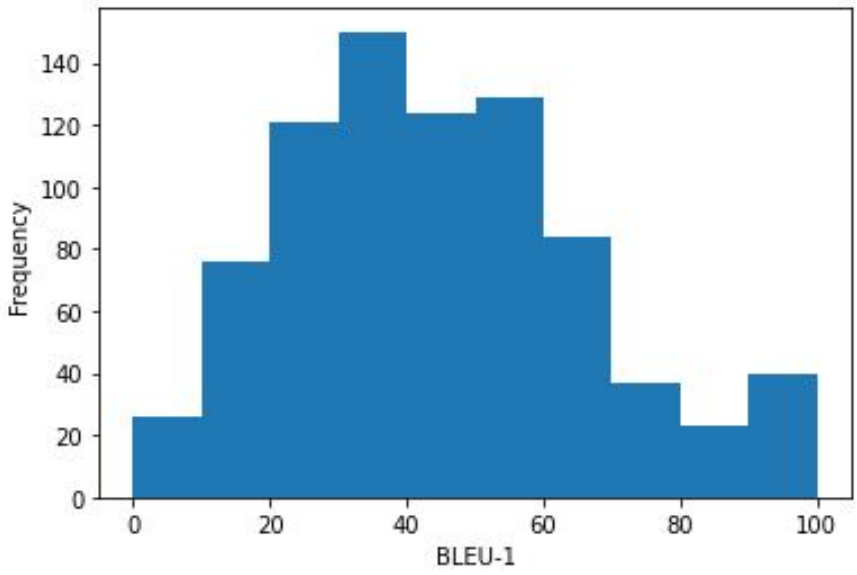


Figure 4.3: Histogram of the BLEU-1 scores of captions produced by the Transformer-based model.

Table 4.3: Breakdown of number of epochs to converge and time per epoch for each model.

Model	Epochs to Converge	Time per Epoch [1]
Transformers + AraBERT	37	1040 seconds
Transformers	35	1110 seconds
LSTM + AraBERT	20	215 seconds
LSTM	20	227 seconds
GRU + AraBERT	19	203 seconds
GRU	20	210 seconds

client (i.e. the mobile device) runs one of the RNN-based models to provide captions when there's no network connection of high network latency, and a server that uses a high-end GPU to run the transformers-based model and provide higher quality captions to the lower-end device via an REST API (Application Programmable Interface).

[1]Reporting the average speed of captioning the entire testing set (810 images) on a workstation with the following specifications: Intel i7 5960K CPU, Nvidia RTX 2080 TI GPU, 64GB DDR4 RAM, and 2TB HDD.

Table 4.4: Model size and captioning speed comparison between our models.

Model	Size	Speed [2]
Transformers + AraBERT	390 MB	1.22 second/image
Transformers	426 MB	1.1 second/image
LSTM + AraBERT	48.7 MB	0.16 second/image
LSTM	50.8 MB	0.14 second/image
GRU + AraBERT	49.3 MB	0.15 second/image
GRU	48.2 MB	0.14 second/image

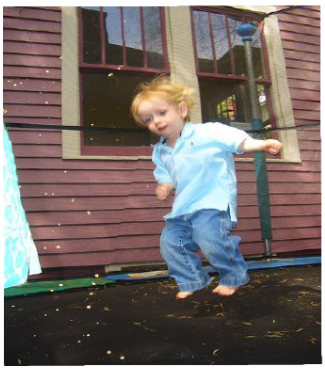
4.4 Training on the English Flicker8k Dataset

As discussed in subsection 2.2.6, the BLEU scores of Arabic image captioning work, including ours, is still low compared to English alternatives. In order to verify whether the score gap is caused by the morphological complexity of Arabic and the lower quantity of data (3 captions per image instead of 5 captions in the Flicker8k dataset), we trained the same Transformer-based model on the original English version of the Flicker8k dataset using the same training parameters and the number of epochs. The model produced a BLEU-1 score of 83.7 and a BLEU-4 score of 27.9, indicating that the low scores are due to the training data and the complex nature of the Arabic language and not due to the model architecture or the training setup. We also experimented with artificially reducing the English training data to match the size of the Arabic data by using only three captions per image instead of five. The experiment resulted in a reduced BLEU score, with a BLEU-1 score of 73.2 and BLEU-4 score of 25.6.

كلب بني يركض في حقل
Brown dog running in a field

صبي صغير في قميص ازرق وجينز ازرق
Little boy in blue shirt and blue jeans

امراه في ثوب السباحه تمشي في بركه
Woman in swimming suit walking on a puddle



صبي صغير يقفز في الهواء
Little boy jumping in the air

صبيان يستعدان للقفز من رصيف يقع علي جسم كبير من الماء
Two boys getting ready to jump from a pier on a large body of water

امراه في ستره برتقاليه تتحدث علي هاتفها المحمول
Women in orange jacket talking on a mobile phone

صبي صغير يرتدي زي القراصنه يرفع علم القراصنه
Little boy wearing pirate outfit and holding a pirate flag

صبي صغير يلعب كره السله في ملعب رياضي
Little boy playing basket ball in sport field

رجل يعزف علي الغيتار في الشارع
A man playing the guitar on the street

Describes without errors | Describes with minor errors | Unrelated to the image

Figure 4.4: Examples of captions generated by the Transformer-based model with AraBERT segmentation, grouped by manual rating.

CHAPTER 5

CONCLUSION

In this work, we presented three model architectures for Arabic image captioning: an LSTM-based model, a GRU-based model, and a Transformer-based model. The Transformer-based model produced the most accurate captions while the LSTM/GRU-based models yielded faster performance while still producing relatively good captions. We also investigated and implemented a new preprocessing technique that alleviates some of the morphological complexity that is inherit in Arabic language by using a word segmenter as part of preprocessing pipeline. Our proposed image captioning models incorporated techniques that were not previously explored in Arabic image captioning such as attention mechanism and transformers in the natural language generator. These enhancements yielded better results over most of recent works done on AIC, improving BLEU-1 score from 33 to 44.3 and BLEU-4 score from 6 to 15.6. All of our experiments produced better BLEU scores than the previous work on the Arabic Flicker8k dataset by El Jundi et al [15].

5.1 Future Research Directions

Possible avenues for further research include improving the transformer-based model by exploring more complex architectures, and spending more time on tuning the hyper-parameters. More importantly, AIC researches desperately need a solid benchmark dataset such as the COCO dataset. While the Arabic Flicker8k dataset allowed us to perform this research and achieve comparatively good results, it is more than an order of magnitude smaller than the COCO dataset, thus covering a much smaller number of visual concepts and scenarios. And while the machine translated captions are manually ranked and only the best three captions are kept in the data, there are still some minor errors. A manually translated Arabic COCO dataset is a significant undertaking but will open the doors for much higher quality AIC research

that can produce better results and allow researchers to better compare their work to that of others in the research community.

5.2 Alternative Approach to AIC

Another approach to AIC would be to use an English captioning model along with a translation model. This seems like a reasonable approach given the higher quality data and richer research available in the English image captioning field. It would, however, have the following problems:

- Images provide richer information that the Arabic translation model would lose when working only with English text (i.e., a picture is worth a thousand words).
- Captioning errors and translation errors would compound, potentially producing a much higher error.
- Runtime would significantly increase since we are doing two operations, captioning and translating, making it more difficult to run the model on lower resource devices such as mobile phones.

While this approach is yet to be explored, if we consider Google Translator to be one of the best Arabic translation models currently available, then the results that we explored on this translation service by examining the Arabic COCO dataset in subsection 2.2.2 (which used Google's Machine Translation API to translate the original English captions) showed a significant rate of error, leading us to believe that this approach will not produce the best results with the currently available resources.

BIBLIOGRAPHY

[1] A. Abdelali, K. Darwish, N. Durrani, and H. Mubarak, "ŋNJßßþÞFarasa: ŋNJßßþÞA fast and furious segmenter for arabic," in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: Demonstrations*, pp. 11–16.

[2] P. Anderson, B. Fernando, M. Johnson, and S. Gould, "ŋNJßßþÞSpice: ŋNJßßþÞSemantic propositional image caption evaluation," in *European Conference on Computer Vision*, Springer, pp. 382–398.

[3] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, and L. Zhang, "ŋNJßßþÞBottom-up and top-down attention for image captioning and visual question answering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6077–6086.

[4] W. Antoun, F. Baly, and H. Hajj, "ŋNJßßþÞArabert: ŋNJßßþÞTransformer-based model for arabic language understanding," *arXiv preprint arXiv:2003.00104*, 2020.

[5] J. L. Ba, J. R. Kiros, and G. E. Hinton, *ŋNJßßþÞLayer normalization*, 2016. arXiv: 1607.06450 [stat.ML].

[6] D. Bahdanau, K. Cho, and Y. Bengio, "ŋNJßßþÞNeural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[7] S. Banerjee and A. Lavie, "ŋNJßßþÞMeteor: ŋNJßßþÞAn 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*, pp. 65–72.

[8] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "ŋNJßßþÞBenchmark analysis of representative deep neural network architectures," *IEEE Access*, vol. 6, pp. 64 270–64 277, 2018, ISSN: 2169-3536. DOI: 10.1109/access.2018.2877890. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2018.2877890.

[9] H. Bouamor, H. Alshikhabobakr, B. Mohit, and K. Oflazer, "ŋNJßßþÞA human judgement corpus and a metric for arabic mt evaluation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 207–213.

[10] F. Chollet *et al.*, *ŋNJßßþÞKeras*, https://keras.io, 2015.

[11] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "ŋNJßßþÞEmpirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[12] M. Cornia, M. Stefanini, L. Baraldi, and R. Cucchiara, "ŋNJßßþÞMeshed-memory transformer for image captioning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10 578–10 587.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ŋNJßßþÞImagenet: ŋNJßßþÞA large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 2009, pp. 248–255.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "ŋNßßþÞBert: ŋNßßþÞPre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[15] O. ElJundi, M. Dhaybi, K. Mokadam, H. M. Hajj, and D. C. Asmar, "ŋNßßþÞResources and end-to-end neural network models for arabic image captioning," in *VISIGRAPP (5: VISAPP)*, pp. 233–241.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "ŋNßßþÞDeep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/abs/1512.03385.

[17] S. Hochreiter and J. Schmidhuber, "ŋNßßþÞLong short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] M. Hodosh, P. Young, and J. Hockenmaier, "ŋNßßþÞFlickr8k dataset,"

[19] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "ŋNßßþÞA comprehensive survey of deep learning for image captioning," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–36, Feb. 2019, ISSN: 1557-7341. DOI: 10.1145/3295748. [Online]. Available: http://dx.doi.org/10.1145/3295748.

[20] X. Hu, X. Yin, K. Lin, L. Wang, L. Zhang, J. Gao, and Z. Liu, "ŋNßßþÞVivo: ŋNßßþÞSurpassing human performance in novel object captioning with visual vocabulary pre-training," *arXiv preprint arXiv:2009.13682*, 2020.

[21] V. Jindal, "ŋNßßþÞA deep learning approach for arabic caption generation using roots-words," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, ISBN: 2374-3468.

[22] ——, "ŋNßßþÞGenerating image captions in arabic using root-word based recurrent neural networks and deep neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, ISBN: 2374-3468.

[23] M. I. Jordan, "ŋNßßþÞSerial order: ŋNßßþÞA parallel distributed processing approach," in *Advances in psychology*, vol. 121, Elsevier, 1997, pp. 471–495.

[24] D. P. Kingma and J. Ba, *ŋNßßþÞAdam: ŋNßßþÞA method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[25] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "ŋNßßþÞMicrosoft coco: ŋNßßþÞCommon objects in context," in *European conference on computer vision*, Springer, pp. 740–755.

[26] H. Lu, R. Yang, Z. Deng, Y. Zhang, G. Gao, and R. Lan, "ŋNßßþÞChinese image captioning via fuzzy attention-based densenet-bilstm," *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 17, no. 1s, Mar. 2021, ISSN: 1551-6857. DOI: 10.1145/3422668. [Online]. Available: https://doi.org/10.1145/3422668.

[27] M.-T. Luong, H. Pham, and C. D. Manning, "ŋNßßþÞEffective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *ŋNßßþÞEfficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781 [cs.CL].

[29] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, *ŋNßßþÞDistributed representations of words and phrases and their compositionality*, 2013. arXiv: 1310.4546 [cs.CL].

[30] S. K. Mishra, R. Dhir, S. Saha, and P. Bhattacharyya, "ŋNßßþÞA hindi image caption generation framework using deep learning," *ACM Trans. Asian Low-Resour. Lang. Inf. Process.*, vol. 20, no. 2, Mar. 2021, ISSN: 2375-4699. DOI: 10.1145/3432246. [Online]. Available: https://doi.org/10.1145/3432246.

[31] H. A. Al-Muzaini, T. N. Al-Yahya, and H. Benhidour, "ŋNßßþÞAutomatic arabic image captioning using rnn-lst m-based language model and cnn," *International Journal of Advanced Computer Science and Applications*, vol. 9, no. 6, 2018.

[32] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "ŋNßßþÞBleu: ŋNßßþÞA method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pp. 311–318.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "ŋNßßþÞFaster r-cnn: ŋNßßþÞTowards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99.

[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "ŋNßßþÞLearning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, ISSN: 1476-4687. DOI: 10.1038/323533a0. [Online]. Available: https://doi.org/10.1038/323533a0.

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "ŋNßßþÞMobilenetv2: ŋNßßþÞInverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520.

[36] K. Simonyan and A. Zisserman, "ŋNßßþÞVery deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "ŋNßßþÞDropout: ŋNßßþÞA simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[38] I. Sutskever, O. Vinyals, and Q. V. Le, "ŋNßßþÞSequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112.

[39] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "ŋNßßþÞInception-v4, inception-resnet and the impact of residual connections on learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, ISBN: 2374-3468.

[40] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "ŋNßßþÞRethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826.

[41] M. Tan and Q. Le, "ŋNßßþÞEfficientnet: ŋNßßþÞRethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6105–6114.

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "ŋNßßþÞAttention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[43] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "ŋNßßþÞCider: ŋNßßþÞConsensus-based image description evaluation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4566–4575.

[44] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "ŋNʃßßþÞShow and tell: ŋNʃßßþÞA neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3156–3164.

[45] P. Werbos, "ŋNʃßßþÞBackpropagation through time: ŋNʃßßþÞWhat it does and how to do it," *Proceedings of the IEEE*, vol. 78, pp. 1550–1560, Nov. 1990. DOI: 10.1109/5.58337.

[46] J. Wu, H. Zheng, B. Zhao, Y. Li, B. Yan, R. Liang, W. Wang, S. Zhou, G. Lin, Y. Fu, Y. Wang, and Y. Wang, *ŋNʃßßþÞAi challenger : ŋNʃßßþÞA large-scale dataset for going deeper in image understanding*, 2017. arXiv: 1711.06475 [cs.CV].

[47] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "ŋNʃßßþÞShow, attend and tell: ŋNʃßßþÞNeural image caption generation with visual attention," in *International conference on machine learning*, pp. 2048–2057.

[48] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "ŋNʃßßþÞDive into deep learning," *arXiv preprint arXiv:2106.11342*, 2021.

APPENDIX

1. Models Definition

The code snippets below describes the model definitions using the sub-classing method in Keras [10]. The complete codebase is available at:
`https://github.com/SniperDW/Transformer-Based-Arabic-Image-Captioning`

1.1 LSTM/GRU Model

```
import tensorflow as tf

embedding_dims = 256
units = 512
vocab_size = 8000 # 5000 for AraBERT
rnn_type = "lstm" # lstm | gru

class Encoder(keras.Model):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc = tf.keras.layers.Dense(embedding_dim, activation='relu')

    def call(self, x):
        return self.fc(x)

class Deocder(tf.keras.Model):
  def __init__(self):
    super(Deocder, self).__init__()

    self.units = units

    # Bahadanu Style Attention Mechanism
    self.u_attn = tf.keras.layers.Dense(units)
    self.w_attn = tf.keras.layers.Dense(units)
    self.v_attn = tf.keras.layers.Dense(1)

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

    self.rnn = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True) if rnn_type == "gru"
    else tf.keras.layers.LSTM(self.units,
                                   return_sequences=True,
                                   return_state=True)

    self.fc1 = tf.keras.layers.Dense(self.units)
    self.dropout = tf.keras.layers.Dropout(0.5)
    self.fc2 = tf.keras.layers.Dense(vocab_size)
```

```
  def call(self, x, features, hidden):

    hidden_with_time_axis = tf.expand_dims(hidden, 1)

    score = self.v_attn(tf.nn.tanh(self.u_attn(features) + self.w_attn(
   hidden_with_time_axis)))
    context_vector = attention_weights * features
    context_vector = tf.reduce_sum(context_vector, axis=1)

    x = self.embedding(x)

    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    output, state = self.rnn(x)
    x = self.fc1(output)
    x = tf.reshape(x, (-1, x.shape[2]))
    x = self.dropout(x)
    x = self.fc2(x)
    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))
```

1.2 Transformers Model

```
import tensorflow as tf

sequence_length = 20
vocab_size = 8000 # 5000 for AraBERT
embed_dim = 512
num_heads = 8
units = 512

class PositionalEmbedding(tf.keras.layers.Layer):
    def __init__(self, sequence_length, vocab_size, embed_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = tf.keras.layers.Embedding(
            input_dim=vocab_size, output_dim=embed_dim
        )
        self.position_embeddings = tf.keras.layers.Embedding(
            input_dim=sequence_length, output_dim=embed_dim
        )
        self.sequence_length = sequence_length
        self.vocab_size = vocab_size
        self.embed_dim = embed_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
```

```
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)


class TransformerEncoderBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, units, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.units = units
        self.num_heads = num_heads
        self.attention = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.dense_proj = tf.keras.layers.Dense(embed_dim, activation="relu")
        self.layernorm_1 = tf.keras.layers.LayerNormalization()

    def call(self, inputs, training, mask=None):
        inputs = self.dense_proj(inputs)
        attention_output = self.attention(
            query=inputs, value=inputs, key=inputs, attention_mask=None
        )
        proj_input = self.layernorm_1(inputs + attention_output)
        return proj_input

class TransformerDecoderBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, units, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.units = units
        self.num_heads = num_heads
        self.attention_1 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.attention_2 = tf.keras.layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim
        )
        self.dense_proj = keras.Sequential(
            [tf.keras.layers.Dense(units, activation="relu"), tf.keras.layers.
    Dense(embed_dim)]
        )
        self.layernorm_1 = tf.keras.layers.LayerNormalization()
        self.layernorm_2 = tf.keras.layers.LayerNormalization()
        self.layernorm_3 = tf.keras.layers.LayerNormalization()

        self.embedding = PositionalEmbedding(
            embed_dim=EMBED_DIM, sequence_length=SEQ_LENGTH, vocab_size=
    VOCAB_SIZE
        )
        self.out = tf.keras.layers.Dense(VOCAB_SIZE)
        self.dropout_1 = tf.keras.layers.Dropout(0.1)
        self.dropout_2 = tf.keras.layers.Dropout(0.5)
```

```
        self.supports_masking = True

    def call(self, inputs, encoder_outputs, training, mask=None):
        inputs = self.embedding(inputs)
        causal_mask = self.get_causal_attention_mask(inputs)
        inputs = self.dropout_1(inputs, training=training)

        if mask is not None:
            padding_mask = tf.cast(mask[:, :, tf.newaxis], dtype=tf.int32)
            combined_mask = tf.cast(mask[:, tf.newaxis, :], dtype=tf.int32)
            combined_mask = tf.minimum(combined_mask, causal_mask)

        attention_output_1 = self.attention_1(
            query=inputs, value=inputs, key=inputs, attention_mask=
    combined_mask
        )
        out_1 = self.layernorm_1(inputs + attention_output_1)

        attention_output_2 = self.attention_2(
            query=out_1,
            value=encoder_outputs,
            key=encoder_outputs,
            attention_mask=padding_mask,
        )
        out_2 = self.layernorm_2(out_1 + attention_output_2)

        proj_output = self.dense_proj(out_2)
        proj_out = self.layernorm_3(out_2 + proj_output)
        proj_out = self.dropout_2(proj_out, training=training)

        preds = self.out(proj_out)
        return preds

    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -1), tf.constant([1, 1], dtype=tf.
    int32)],
            axis=0,
        )
        return tf.tile(mask, mult)
```

2. Adam Optimizer

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 5.1: The algorithm of the Adam optimizer as it appears the paper by Kingma et al. [24]. The Adam optimizer uses an adaptive learning rate for each parameter and maintains an exponentially decaying average of past gradients. These techniques allow models using the Adam optimizer to converge more quickly during training.