Information Technology Institute

Ministry of Communications and Information Technology

Scene Description

Graduation Project for ITI AI-Pro Program

Prepared by:

Habiba Mohamed

Mohamed Nasser

Mostafa Nafie

Salma Elmoghazy

Salma Hisham

Intake 42
2021 - 2022

Table of Content

1.Introduction

1.1 Problem definition

Being able to automatically describe the content of an image using properly formed English sentences is a very challenging task, but it could have a great impact, for instance by helping visually impaired people better understand the content of images on the web.

The problem introduces a captioning task; we planned to do image-to-sentence generation. This application bridges vision and natural language, which requires a computer vision system to both localize and describe salient regions in images in natural language. The image captioning task generalizes object detection when the descriptions consist of a single word. Given a set of images and prior knowledge about the content, find the correct semantic label for the entire image.

Input: an image.
Expected output: a natural language description of the input image.

Figure (1): *Image captioning model.*

A description must capture not only the objects contained in an image but also must express how these objects relate to each other, as well as their attributes and the activities they are involved in. Moreover, the above semantic knowledge has to be expressed in a natural language like English, which means that a language model is needed in addition to visual understanding.

1.2 Objective

Our main goal is to develop a technique that allows us to help and facilitate blind people's lives to see the world better by describing a scene and translating it to text/voice . The following is a list of our project objectives:

- Translate images to text and/or voice by taking a photo at any and hear the description of the object read back to you.
- Makes the user feel free and does not need a translator/interpreter. Since the user can use the application anytime anywhere.

1.3 Related work

1.3.1 TapTapSee

Is an app that allows the visually impaired and blind community to accurately identify objects they encounter in their daily lives without the need for sighted assistance. Using your iPhone camera, you can take a photo at any angle and hear the description of the object read back to you. The app also features an auto-focus notification and sharing options. In addition, you can have the last image identification repeated. Finally, you can upload photos from your camera roll for identification and even save them to your phone afterward with the provided definitions for easy reuse.

1.3.2 Lookout

Is an app that identifies important items in your environment and reports the information it believes is relevant. This might include things like exit signs, the location of a bathroom, people or objects nearby, and even text in a book. Lookout's spoken notifications are designed to be used with minimal interaction so that they don't distract you or get in the way.

The previously mentioned related work shows some advantages that we take under consideration, like:

- Specializes in automatic translation of images to text and/or voice content.
- Identify objects, text, and even people.
- A mobile application used to describe images to its equivalent text/audio in the form of an Multimedia Messaging Service (MMS) and Notifications .

But after a sufficient study of all aspects of these applications we discover some cons, that may hinder the ease of use of the service including:

- Machine-dependent.
- Expensive to buy.
- English Language Only.

1.4 Datasets

There are many open source datasets available for this problem, like Flickr 8k (containing 8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

Here we used **MSCOCO (Microsoft Common Objects in Context)**. MSCOCO dataset is maintained by a team of contributors and sponsored by Microsoft, Facebook, and other organizations. MSCOCO is composed of several datasets, created in different years; each one focuses on a different computer vision task, such as:

- Object detection.
- Segmentation.
- key-point detection.
- Image captioning.

The image captioning training dataset consists of more than 100,000 images with five independent human-generated captions being provided for each image.

2.System Architecture

2.1 Model Architecture Overview

Our image-captioning model follows the same architecture as the one proposed in the famous "Show, attend and tell" paper. It offers a neural network architecture that consists of three main components: encoder, decoder and attention module.

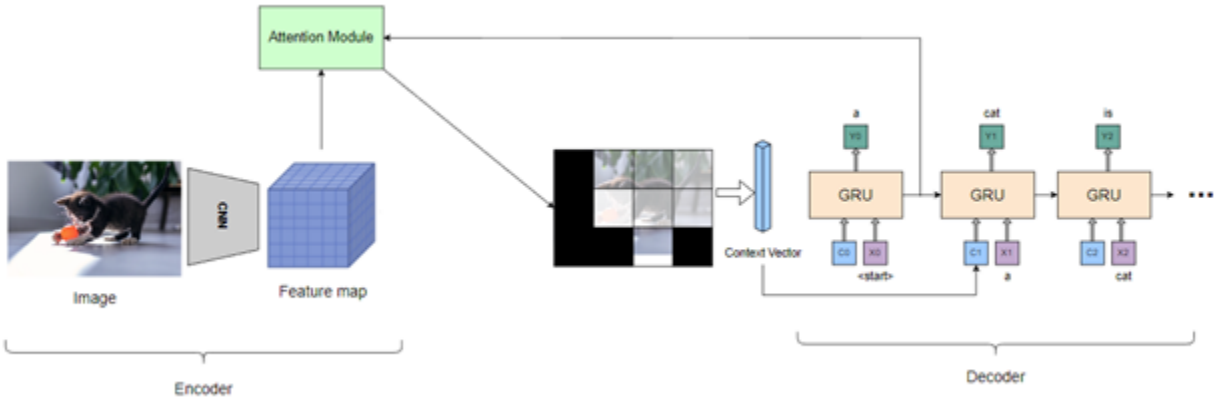


Figure (1): *System architecture.*

2.2 Encoder

It is a pretrained VGG16 convolutional neural network (CNN) that has been trained on the "imagenet" classification task, where only the convolutional layers are used and the fully connected layers are discarded.
This encoder takes in the image as its input and extracts the most important features and landmarks from it to produce multiple vectors, each of which corresponding to a particular part of the image.

Figure (2): *Image Encoder.*

What is VGG16 ?

It's a convolutional neural network, also known as a ConvNet, which is a kind of artificial neural network. A convolutional neural network has an input layer, an output layer, and various hidden layers. VGG16 is a type of CNN that is considered to be one of the best computer vision models. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3 × 3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16 weight layers making it approx — 138 trainable parameters.

VGG16 was trained on the ImageNet dataset. ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories.

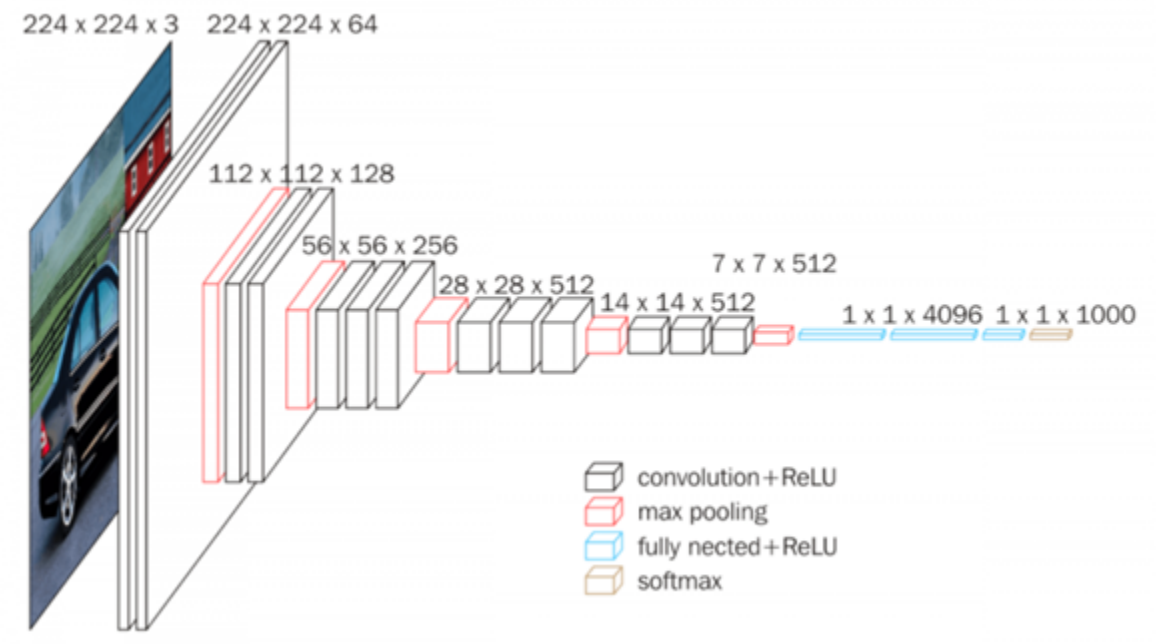


Figure (3): *VGG16 architecture.*

VGG16 architecture consists of an input layer that takes a fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3. The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2. All hidden layers are equipped with the ReLU non-linearity.

2.3 Decoder

It is a recurrent neural network (RNN) that consists of GRU memory cells. Each GRU cell takes in the embedding vector of the previous word in the caption concatenated with the context vector of the image as inputs.

The output of the GRU cell is passed through a couple of fully connected layers and a softmax layer to produce a probability distribution over the vocabulary words and then we choose the word with the highest probability to be the next word in the caption.

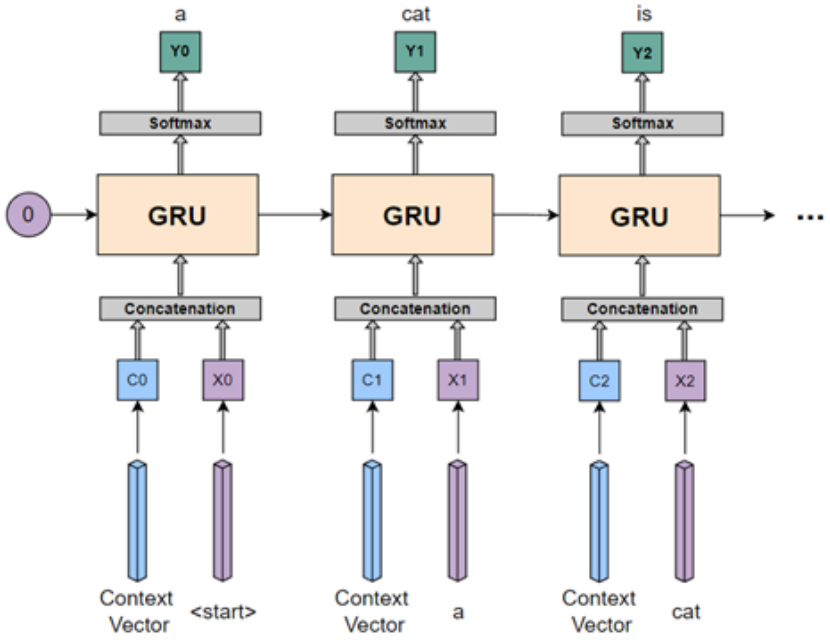


Figure (4): *Caption Decoder.*

So why did we use GRU?

GRUs address the vanishing gradient problem (values used to update network weights), from which vanilla recurrent neural networks suffer. If the grading shrinks over time as it back-propagates, it may become too small to affect learning, thus making the neural net untrainable.

If a layer in a neural net can't learn, RNN's can essentially "forget" longer sequences. GRUs solve this problem through the use of two gates, the update gate and the reset gate. These gates decide what information is allowed through to the output and can be trained to retain information from farther back. This allows it to pass relevant information down a chain of events to make better predictions, just like the LSTM, but it has advantages over long-term short-term memory

(LSTM). GRU uses less memory and is faster than LSTM. However, LSTM is more accurate when using datasets with longer sequences.

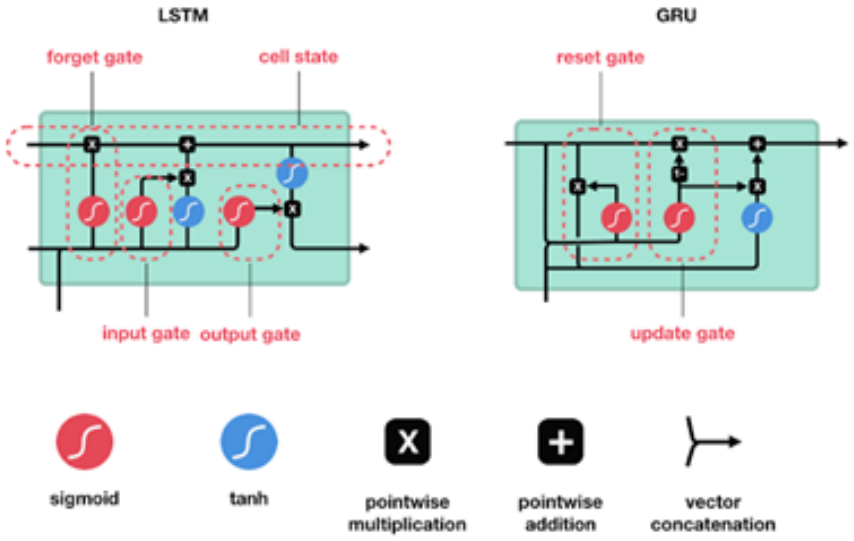
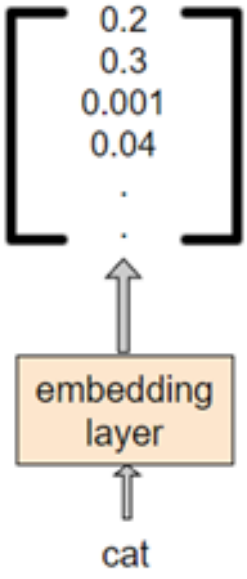


Figure (5): *LSTM and GRU cells.*

2.4 Word Embeddings

Since our decoder takes in a word at each time-step and produces a new word and neural networks can work only with numerical values and not strings, so here the embedding layer comes to the rescue.



The embedding layer replaces each word with its corresponding word embedding vector that represents that word.

The Word Embeddings try to capture the semantic, contextual, and syntactic meaning of each word in the corpus vocabulary based on the usage of these words in sentences. Words that have similar semantic and contextual meaning also have similar vector representations while at the same time each word in the vocabulary will have a unique set of vector representations. An important aspect of these representations is the ability to solve word analogies of the form "A is to B what C is to X" using simple arithmetic. This is generally simplified as "King - Man + Woman = Queen".

The word embeddings representation maps each word to N-Dimensional space where N is pre-defined so it solves the scalability issue. Each embedding vector is densely populated and

that solves the sparsity issue. These vectors are also learned in a way that captures the shared context and dependencies among the words. Therefore, word embeddings representation overcomes all the issues of BOW representation.

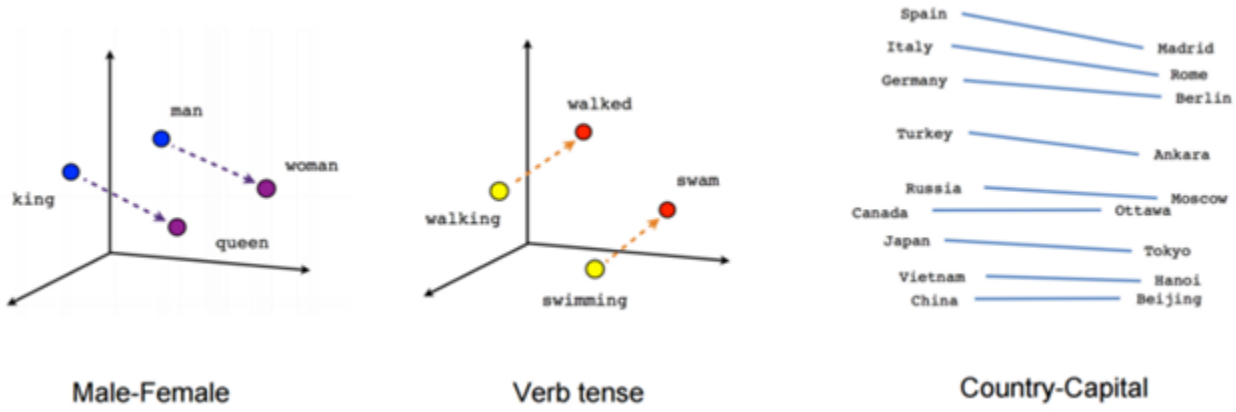


Figure (6): *Word embeddings.*

Since the dataset has a large corpus of captions, we trained the embedding layer using these captions.

2.5 Attention Module

The problem with a simple encoder – decoder model is that the entire feature representation of the image is used as a context vector at each time step when generating a new word in the caption, so no new information is introduced by the image, which is considered as a bottleneck.

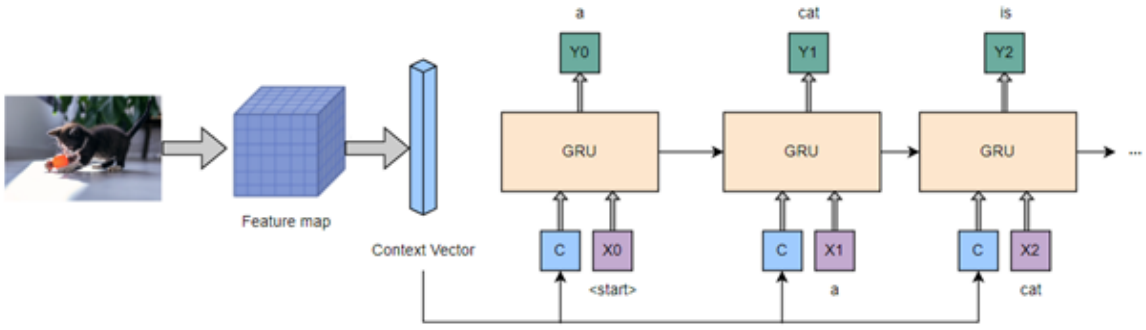


Figure (7): Simple Encoder-Decoder model.

The authors of the paper used the idea of attention that was inspired by the advances in machine translation models and they applied it to images.

This attention mechanism allows the model to produce a different context vector at each time step. That context vector is a weighted combination of the image features the model thinks that they matter the most when generating the next word in the caption.

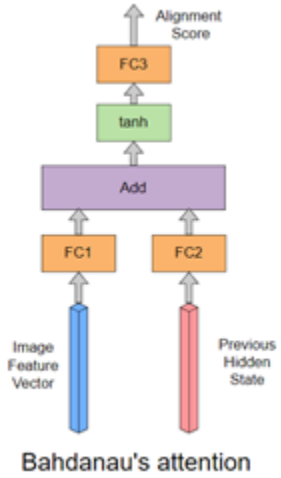
To produce the context vector, the model goes through four steps:

1. The model takes in the image feature map and the decoder's previous hidden state and computes alignment score between each vector in the image's feature map that represents a particular part of the image and the hidden state that represents the generated caption up till that point. This alignment score is considered as a similarity or relationship measure and can be computed using two ways:

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1} \boldsymbol{h}_t + \boldsymbol{W_2} \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases}$$

   a. Bahdanau's additive attention:

      It uses a one-hidden layer feed-forward network to calculate the alignment score, where each of the feature vector and the hidden state pass through a fully connected layer, added together and then the result goes through a tanh activation function and the result finally goes through a single neuron to generate the score.



   b. Luong's multiplicative attention:

      It simply passes the image's feature vector through a fully connected layer to reshape it and then computes the dot product between the resulting vector and the decoder's previous hidden state.

      By training the fully connected layers in both methods, the model learns where to look.

2. The alignment scores go through a softmax layer to normalize them into a probability distribution that represents the attention weights.
3. Each vector in the feature map of the image is multiplied by the corresponding attention weight.
4. The context vector is generated by taking a weighted sum of the feature vectors.

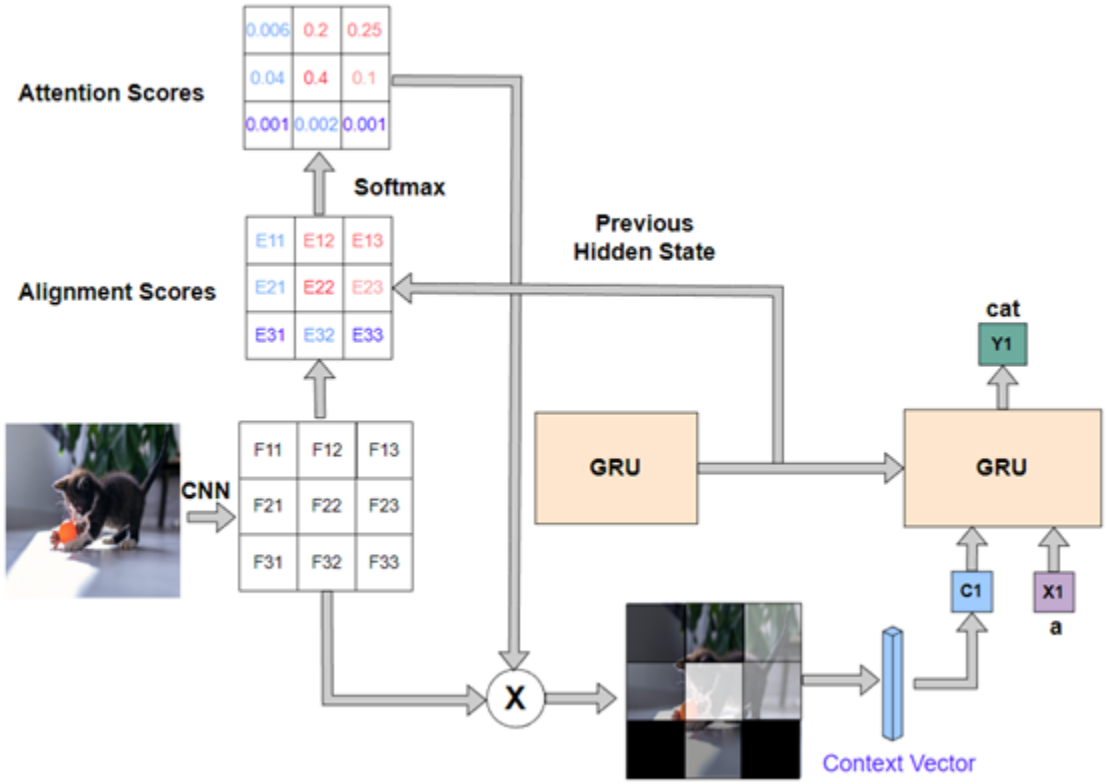


Figure (8): *Attention Mechanism.*

After training, the model learns which parts of the image to focus on at each time step and gives them higher weights to contribute more in the context vector.

Examples of the attention weights of our model after training, where white color means a higher weight:

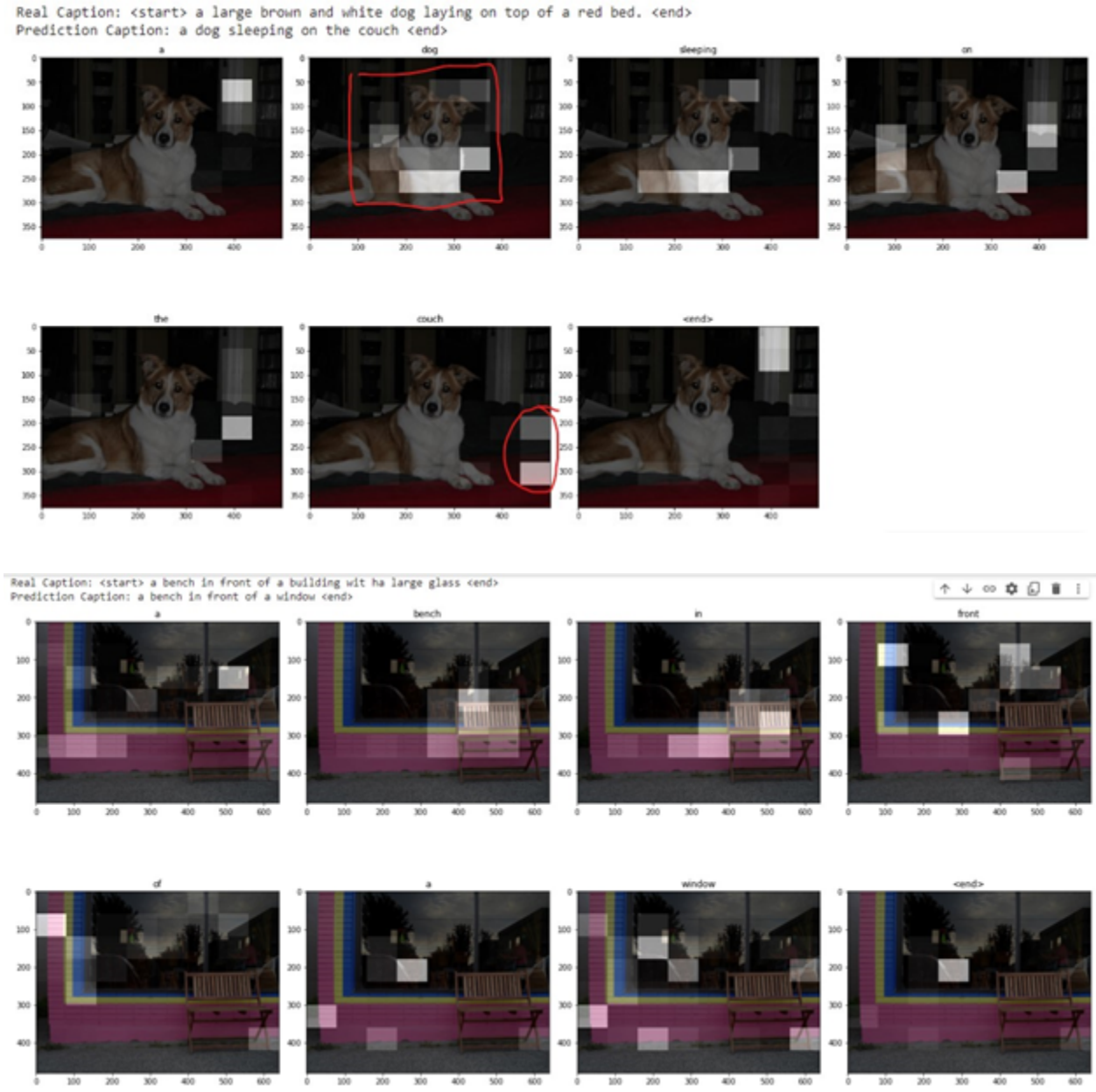


Figure (9): *Examples of Model's Generated Captions and Attention Weights.*

3.Implementation

3.1 Preprocessing

The first step in any machine learning problem is to prepare the raw data and make it suitable for the given machine learning model. In our proposed model, data preprocessing is applied to:

1. Images
2. Captions/ text

3.1.1 Images Preprocessing

Convolutional neural networks (CNNs) are powerful for image processing. We used the VGG16 convolutional neural network, pre-trained on ImageNet without fine tuning, to extract features from the input images. For this purpose, images are preprocessed to match the characteristics of images on which VGG16 was pre-trained as following:

a. Converting images to tensor.
b. Images are resized to (224 x 224 x 3).
c. Passing images to vgg16.preprocess_input method in TensorFlow to convert images from RGB to BGR. Each color channel is zero-centered with respect to the ImageNet dataset, without scaling, so that the network converges faster.

Images are then fed to the VGG16 convolutional neural network to perform feature extraction. We excluded the top (Fully connected) layer and used the result of Conv5 block, as in ***figure (3)***, as our feature map. The shape of the output of this layer is 7 x 7 x 512. Later, our decoder operates on the flattened 49 x 512 encoding. Those features are then saved as .npy files to persist NumPy-array features on a disk, to reuse them without repeating feature extraction steps.

3.1.2. Captions Preprocessing

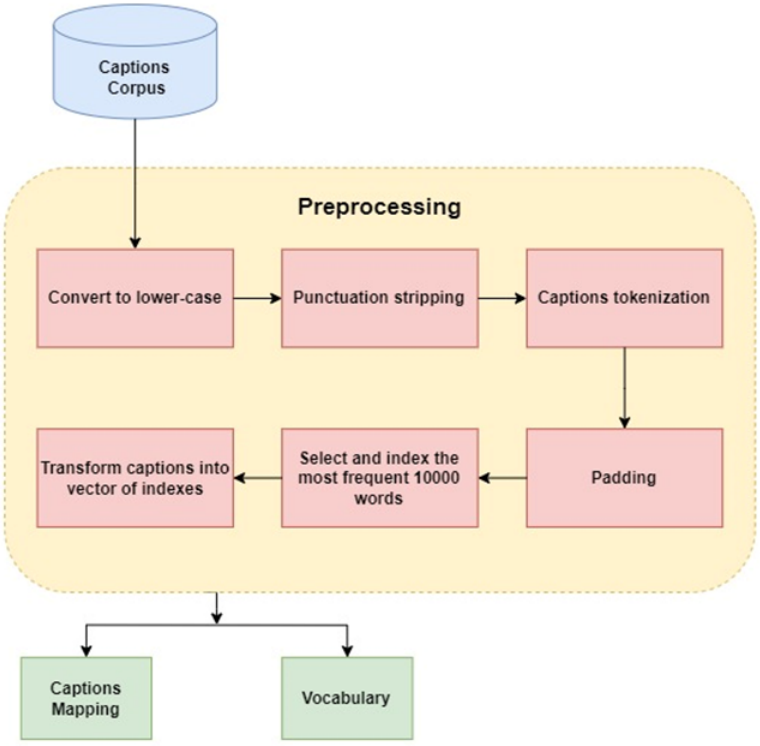


Figure (10): *Captions preprocessing workflow.*

To prepare the text data for the model building we perform text preprocessing in the steps shown in ***figure (10)***.

First, all captions are converted to lower-case and punctuation is stripped. Then we add *<start>* and *<end>* tokens to the start and end of each caption. Captions are then tokenized, while setting maximum length to 30, and applying padding to the captions with smaller length. We constructed a vocabulary of fixed size of 10000, using the most frequent words. Each word in the vocabulary is associated with a unique index. Finally, we transformed each example/ caption using these indexes, into vectors of integers.

The final outputs of the preprocessing steps are the vocabulary and the captions vectors, which are later used in word embedding.

3.2 Model Training

3.2.1 Training step

To train our model, we used *TensorFlow* framework to build it, as described in the previous sections our model consists of two parts encoder and decoder, for the encoder part we used the features extracted from a pre-trained CNN *vgg16* this part does not need that much work we did not unfreeze any layer to train it on our data. The big part that involved in the training step was the decoder part, this part was a time sequence which means that at each time step we have different input and different predictions for that we build our custom training loop. In training we had two different loops one for training the model while the other one for validating our model in the validation set.

Starting with training step:

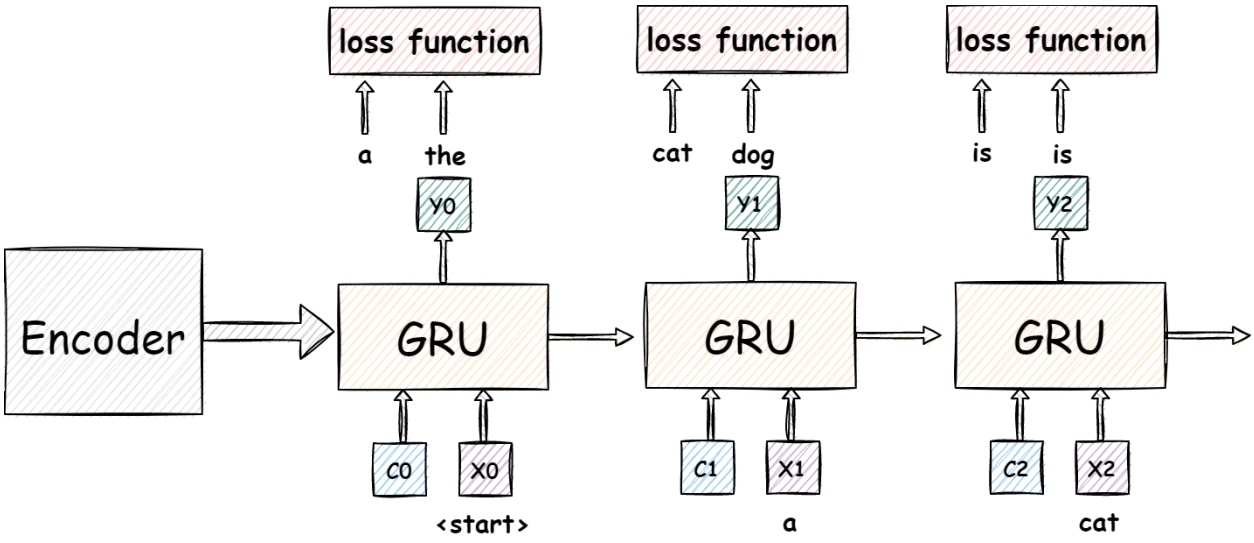


Figure (11): *Training step decoder.*

As we can see we feed the model at each time step with the target word, the corresponding context vector, and the features extracted from the image as an input to the GRU unit, then the output for this unit is the predicted word from our model we compare it with the next actual token in the real caption, then compute the loss -*categorical cross entropy*- between them, the next time step we also feed the GRU unit the next actual word in our real caption as we used *teacher forcing* technique to train our model, then we calculate the cumulative sum of losses, as previous we

compute the loss between the predicted word and the actual next token and add it to the loss.

3.2.2 Validation Step

For the validation step it is not a big difference but as we can see below the only difference is that the predicted word at the current time step is the input word for the next time step this predicted word is the highest probability in the calculated probability distribution among all words in our vocabulary.

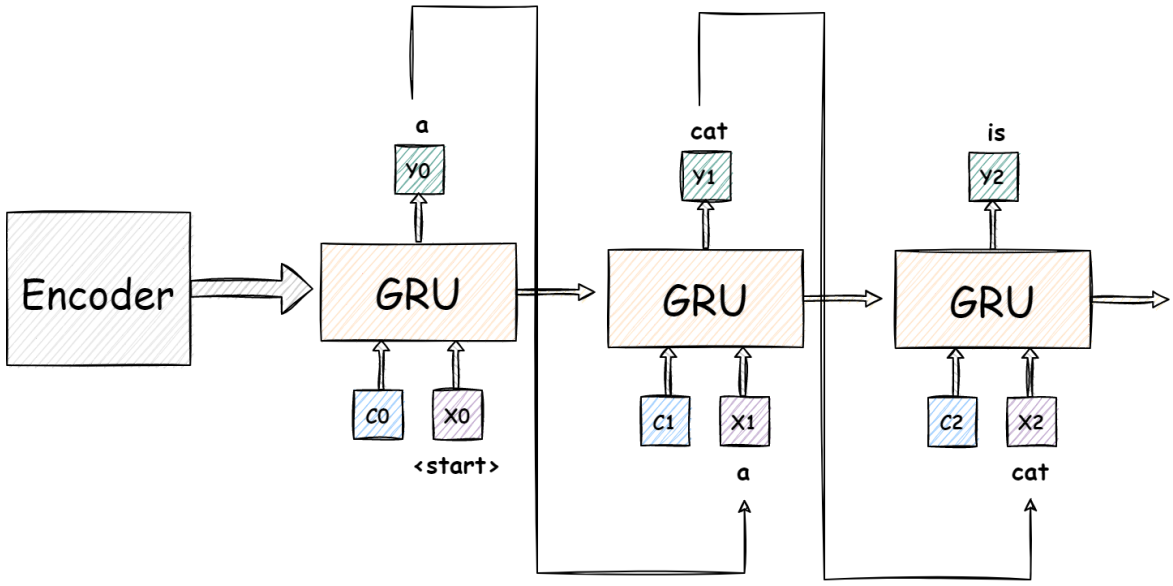


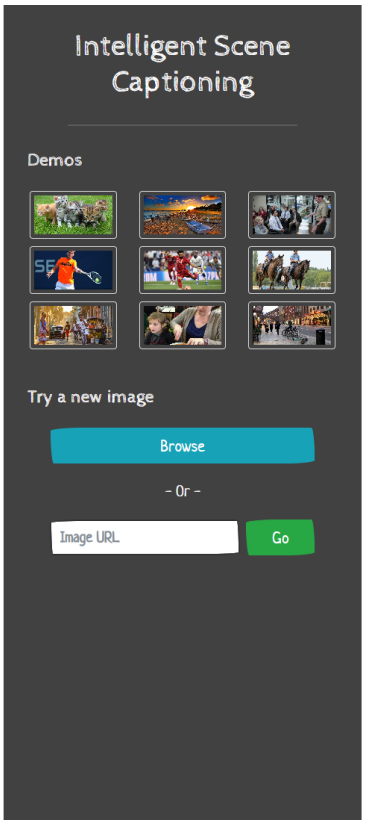
Figure (12): *Validation/Testing step decoder.*

3.3 Deployment

We used Dash to build our interactive web application. Dash is an Open Source Python library for creating reactive, Web-based applications, released under the permissive MIT license. It's written in plotly.js and reasct.js, which are used for creating interactive web applications, and also Ideal for building and deploying data apps with customized interfaces. It's suitable for any data application. It makes it dead simple to build a GUI around the data analysis code because we don't have to write any Javascript or HTML. Dash provides a Python interface to a rich set of interactive web-based components. It's rendered in the web browser and you can deploy the app to VMs or Kubernetes clusters and then share them through URLs. Since Dash apps are viewed in the web browser, Dash is inherently cross-platform and mobile-ready. We developed our web application from 2 pages.

3.3.1 Main Page

It contains the demo of our model, where you can test it using diff facilities. You can either upload, drag and drop or use the image link and get the caption within the attention plot of it in just a sec.

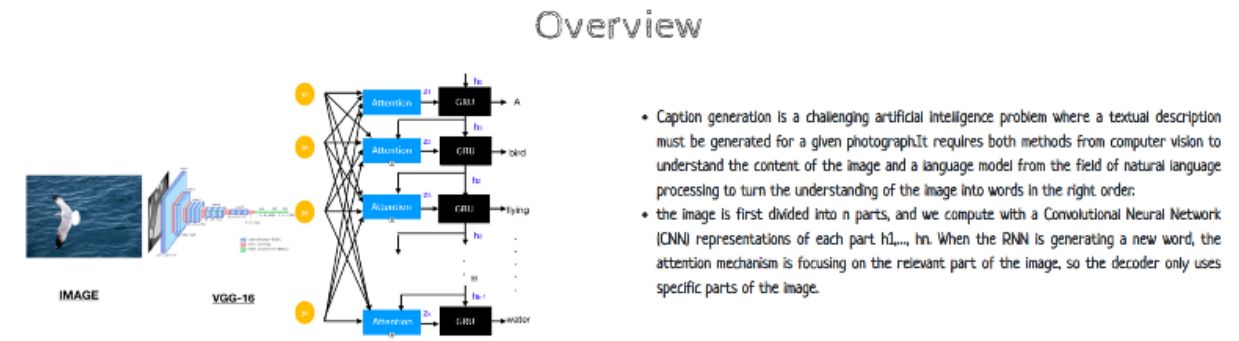


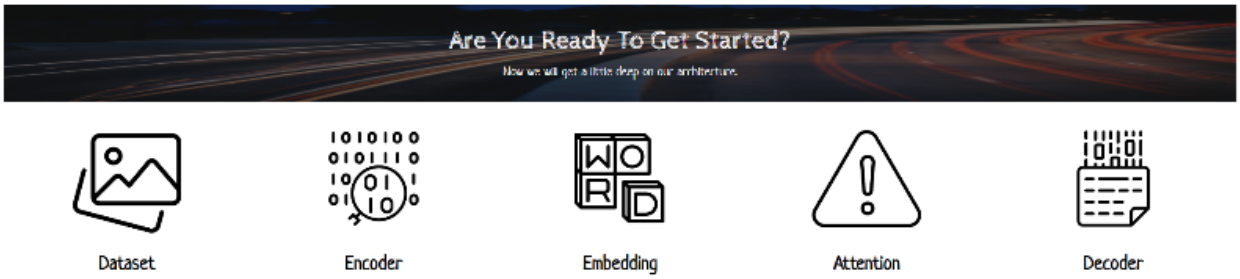
group of men and women standing next to each other in front of a stone bench.

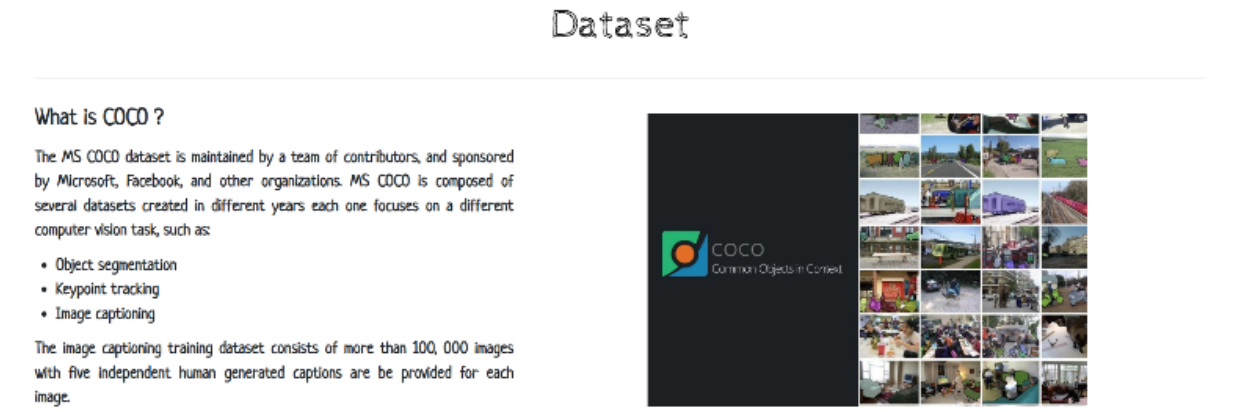
Show Attention Plot

3.3.2 Landing page

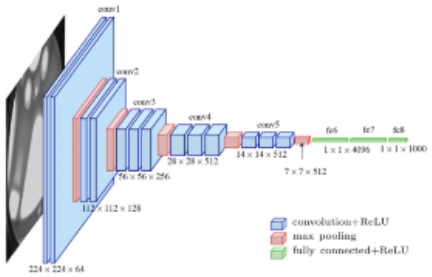
It's for explaining our model architecture a little bit and giving a short brief about its components.







Encoder (CNN)



Encoder ?

It's a convolutional neural network that takes in an image and extracts the most important features and landmarks from it. Later, this feature map is passed to the decoder, which relies on it to generate a caption that fits that image.

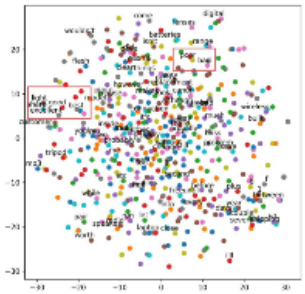
VGG ?

It's an innovative object-recognition model that supports up to 19 layers. Built as a deep CNN Here we used VGG16 which is one of the most popular algorithms for image classification and is easy to use with transfer learning.

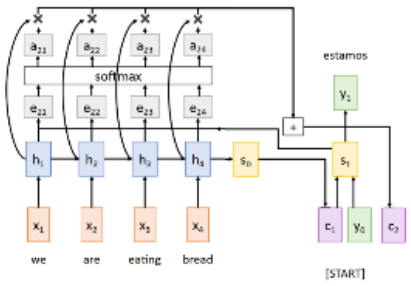
Word Embedding

Why do we need word embedding ?

- It's a layer that studies the captions' vocabulary words and gives them a numerical representation. This numerical representation encodes the meaning of the word such that the words that are closer in the vector space are said to be similar in meaning.
- Word embeddings are basically a form of word representation that bridges the human understanding of language to that of a machine. Word embeddings are distributed representations of text in an n-dimensional space.



Attention



What is the attention mechanism ?

The attention mechanism was introduced to address the bottleneck problem that arises with the use of a fixed-length encoding vector, where the decoder would have limited access to the information provided by the input.

This module takes in the feature map of the image along with the previous decoder's hidden state and tries to learn the relationship between them using a fully connected network. Later, this relationship will tell the decoder which parts of the image to focus on in order to generate the next word.

Decoder (RNN)

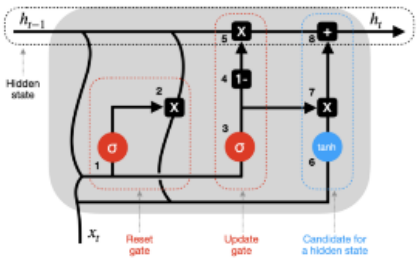
Decoder ?

It's a recurrent neural network that consists of GRU cells that generate a word at each time step. These GRU cells help the model learn long sequences at a low computational cost. At each time step, the decoder uses the attention module to focus on specific parts of the image and generates a new word.

GRU ?

It's a gating mechanism in recurrent neural networks utilises different ways of gating information to prevent vanishing gradient problems.

- Controls the flow of information without having to use a memory unit.
- Just exposes the full hidden content without any control.
- Computationally efficient.



4.Future Work

We can divide our future work into two steps:

1. Model Architecture
2. Model Deployment

First, we should say that *image captioning* is a challenging multimodal task. Significant improvements could be obtained by deep learning. Yet, captions generated by humans are still considered better, which makes it an interesting application for interactive machine learning and explainable artificial intelligence methods. In this work, we aim at improving the performance and explainability of the state-of-the-art method Show, Attend and Tell by using *Transformers* in our architecture. The transformer network contains a similar architecture like we used *(encoder-decoder).* The main difference is that the transformer will give us the opportunity to receive the input sequence in parallel which is suitable for parallel computing.

Second, we used *plotly-dash* to deploy our model while a successful and useful application requires continuous updates and improvements, our application is not excluded from this rule. To make our application useful considering our main goal that to provide real aid to the visually impaired so, we should make it accessible for that we are planning to make an android application that can take voice commands to take a picture of the surrounding scene then generate the text caption, finally we can use a text to speech API to tell the user the generated caption.

Finally, we were planning to add some features to our project so that it could be *a video captioning* also not just *an image captioning*, but we know that it might be difficult using the same model with the same architecture so it might be our new project!

5.References

[1] Vinyals, Oriol, Alexander Toshev, Samy Bengio, and Dumitru Erhan. "Show and tell: A neural image caption generator." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3156- 3164. [2015].

[2] Xu, Kelvin, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio."Show, attend and tell: Neural image caption generation with visual attention." arXiv preprint arXiv:1502.03044 [2015].

[3] Karen Simonyan, Andrew Zisserman: *Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR* [2015].

[4] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling. NIPS* [2014].

[5] Karpathy, Andrej, and Li Fei-Fei. "Deep visual semantic alignments for generating image descriptions" In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3128-3137. [2015].