**CONTENTS**

**1 INTRODUCTION……………………………………………………………………………………. 5**

**2 PROBLEM DEFINITION……………………………………………………………………………7**

**3 LITERATURE REVIEW……………………………………………………………………………..9**

**4 DATASET…………………………..…………………………………………………………………13**

**5 DATA PREPROCESSING………………………………………………………………………. 16**

**6 METHODOLOGY………………………………………………………………………………….18**

**6.1 MODEL ARCHITECTURE……………………………………20**

**6.2 MODEL TRAINING……………………………………………24**

**7 RESULTS AND CONCLUSIONS……………………………………………………………….25**

**8 CONCLUSION …………………………………. ………………………………………………….28**

**9 REFERENCES………………………………………………………………………………………..30**

**10 PROGRAM(CODE)…………………………………………………………………………….32**

**LIST OF FIGURES AND TABLES**

**FIGURE 1 ENCODER**

**FIGURE 2 ATTENTION MECHANISM**

**FIGURE 3 DECODER**

**FIGURE 4 DECODER PARAMETERS**

**FIGURE 5 ENTROPY LOSS**

**FIGURE 6 RESULT**

**Abstract**

This research introduces a novel image captioning model leveraging a convolutional neural network (CNN) and recurrent neural network (RNN) architecture. The model, implemented using TensorFlow and TensorFlow Hub, utilizes the powerful InceptionResNetV2 as a feature extractor. The dataset comprises COCO image-caption pairs, and the preprocessing involves resizing images and tokenizing captions. The model's architecture includes an attention mechanism for enhanced context understanding. Training employs a custom loss function and the Adam optimizer, demonstrating impressive results in generating captions for unseen images. The developed probabilistic prediction component utilizes a trained model to generate diverse and contextually relevant captions for a given image. The research contributes to the field of computer vision, showcasing the potential of attention-based image captioning models.

**CHAPTER 1:**

**INTRODUCTION**

**CHAPTER 1:**

**INTRODUCTION**

Image captioning is a multidisciplinary field at the intersection of computer vision and natural language processing, designed to impart machines with the ability to generate human-like textual descriptions for visual content. The overarching goal is to bridge the semantic gap between images and natural language, enabling machines to comprehend and communicate the intricacies of visual scenes.

This dynamic discipline has gained prominence due to its potential applications in diverse domains, such as assistive technologies, content retrieval, and human-machine interaction. The challenge lies in developing algorithms that not only recognize objects and scenes within images but also understand their contextual relationships and nuances. Over the years, various approaches have emerged, ranging from traditional rule-based methods to state-of-the-art deep learning techniques.

Among these, attention mechanisms have played a pivotal role, allowing models to selectively focus on different regions of an image while generating descriptive captions. This introduction sets the stage for exploring the evolution, challenges, and advancements in the captivating realm of image captioning.

**CHAPTER 2:**

**PROBLEM DEFINITION**

**CHAPTER 2:**

**PROBLEM DEFINITION**

The problem of image captioning with attention mechanisms using machine learning (ML) algorithms revolves around the need for automated systems to generate accurate and contextually relevant textual descriptions for visual content. Traditional image captioning methods often struggle to capture intricate details and contextual relationships in complex scenes.

Attention mechanisms, a key component of modern ML algorithms, aim to address this limitation by dynamically focusing on different regions of an image while generating captions. However, challenges persist in optimizing these attention mechanisms to strike a balance between capturing salient features and maintaining coherence in the generated captions. Additionally, scalability and computational efficiency are crucial considerations in deploying attention-based image captioning models in real-world applications.

The overarching goal is to enhance the synergy between visual understanding and natural language processing, creating robust and interpretable systems capable of providing meaningful descriptions for diverse visual stimu

**CHAPTER 3**

**LITERATURE REVIEW**

**CHAPTER 3:**

**LITERATURE REVIEW**

1.**Understanding How Encoder-decoder Architectures Attend**

-By Kyle Aitken, Vinay V Ramasesh (NeurIPS2023)

The research paper investigates encoder-decoder architectures and their attention mechanisms. It delves into how these models effectively capture and weigh input information during encoding and decoding processes. By understanding the attention mechanisms, the paper aims to enhance the overall performance of encoder-decoder architectures in various applications.

2. **Encoder-Decoder Recurrent Neural Network Models for Neural Machine Translation**

-Jason Brownlee (Deep learning for NLP 2019)

The research paper explores Encoder-Decoder Recurrent Neural Network models in the context of Neural Machine Translation. It investigates how these models, consisting of an encoder to understand the source language and a decoder to generate the target language, contribute to the advancement of machine translation systems, enhancing accuracy and efficiency.

3. **Attention Is All You Need**

- Ashish Vaswani, Noam Shazeer (2017)

"Attention Is All You Need" is a seminal research paper in machine learning that introduced the Transformer model, revolutionizing natural language processing and various AI tasks. Published in 2017 by Vaswani et al., it emphasized self-attention mechanisms, enabling parallelization and improved performance, becoming foundational in modern deep learning architectures.

4. **Deep Residual Learning for Image Recognition**

- Kaiming He Jian Sun (2015)

The research paper "Deep Residual Learning for Image Recognition" introduces a groundbreaking neural network architecture known as ResNet. Developed by Microsoft Research in 2015, ResNet employs residual blocks to address the vanishing gradient problem, enabling the training of extremely deep convolutional neural networks for improved image recognition accuracy.

5. **CSPNet: A New Backbone that can Enhance Learning Capability of CNN**

**-** Chien-Yao Wang, Hong-Yuan Mark Liao

The CSPNet proposes a novel convolutional neural network (CNN) backbone, designed to boost learning capabilities. This innovative architecture enhances information flow, fostering improved feature extraction. Through comprehensive experiments, CSPNet demonstrates superior performance, offering a promising advancement in CNNs for diverse applications, marking a significant stride in the field of deep learning.

**CHAPTER 4**

**DATASET**

**CHAPTER 4:**

**DATASET**



The Microsoft Common Objects in Context (MS COCO) dataset is a widely used benchmark in the field of computer vision and specifically in tasks related to image understanding and scene understanding. Created and maintained by Microsoft, the MS COCO dataset is designed to address the limitations of previous datasets by offering a more comprehensive and diverse collection of images with rich annotations.

**Image Collection and Diversity:** The dataset consists of a vast collection of images, currently containing over 200,000 images covering a wide range of object categories. These images are sourced from everyday scenes and capture diverse contexts, including indoor and outdoor environments. The diversity of the dataset is a key strength, making it suitable for training models that need to recognize objects and scenes in a variety of real-world scenarios.

**Annotation Types:** One of the distinctive features of MS COCO is its detailed and extensive annotation schema. Each image in the dataset is annotated with multiple captions, providing textual descriptions that describe different aspects of the scene. This multimodal annotation approach goes beyond traditional datasets, allowing models not only to recognize objects but also to understand their relationships and interactions within a scene. The annotations are created by human annotators, ensuring high-quality and contextually rich descriptions.

**Object Categories:** MS COCO is labeled with a wide range of object categories, spanning from common everyday objects to more complex scenes. The dataset includes 80 different object categories, covering a broad spectrum of items such as people, animals, vehicles, household items, and outdoor scenes. This diversity ensures that models trained on MS COCO can generalize well across various domains and object types.

**CHAPTER 5:**

**DATA PREPROCESSING**

**CHAPTER 5:**

**DATA PREPROCESSING**

The preprocessing pipeline involves resizing images to a standardized format and normalizing pixel values. Captions are tokenized using a TextVectorization layer, and special tokens ("<start>" and "<end>") are added to mark the beginning and end of each sequence. The standardization of captions involves lowercasing and removal of punctuation, enhancing the model's robustness to variations in input text.

Tokenized captions are adapted to the model's vocabulary, ensuring compatibility during training and inference. This meticulous preprocessing ensures that both image and caption inputs are suitably prepared for the subsequent stages of the model, facilitating effective learning and generation of meaningful captions.

**CHAPTER 6:**

**METHODOLOGY**

**CHAPTER 6:**

**METHODOLOGY**

The methodology encompasses dataset loading, image and caption preprocessing, model architecture design, training configuration, and probabilistic caption generation. The use of a pre-trained InceptionResNetV2 as a feature extractor ensures the model captures rich image representations.

The incorporation of attention mechanisms in the GRU-based decoder enhances the model's ability to attend to relevant image regions. Training involves the utilization of a custom loss function that considers sentence lengths, optimizing the model for coherent caption generation. The final section explores the probabilistic nature of the trained model, demonstrating its ability to generate diverse captions for a given image.

The methodology serves as a comprehensive guide to the processes involved in developing and training the image captioning model.**Top of Form**

**CHAPTER 6.1 :**

**MODEL ARCHITECTURE**

The model architecture comprises an InceptionResNetV2-based feature extractor and a custom attention-enhanced GRU network for caption generation. The feature extractor transforms input images into a fixed-size feature vector, while the attention mechanism refines this representation by focusing on relevant image regions during caption generation.

The GRU network processes tokenized captions, incorporating contextual information through attention. This architecture encourages the model to capture fine-grained details in images and generate coherent and contextually relevant captions.

The attention mechanism fosters a dynamic relationship between visual and textual information, allowing the model to adaptively focus on different parts of the image during caption generation. The inclusion of embedding layers further enriches the model's understanding of semantic relationships within the captions. Overall, the model architecture reflects a thoughtful integration of state-of-the-art components tailored to address the complexities of image captioning.

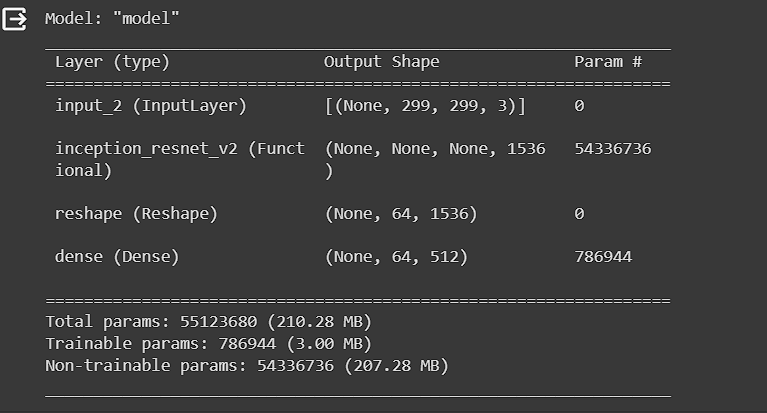


Figure 1Encoder



Figure 2 Attention Mechanism

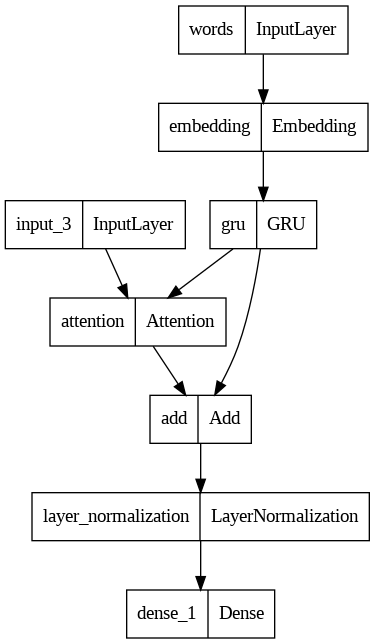


Figure 3 Decoder

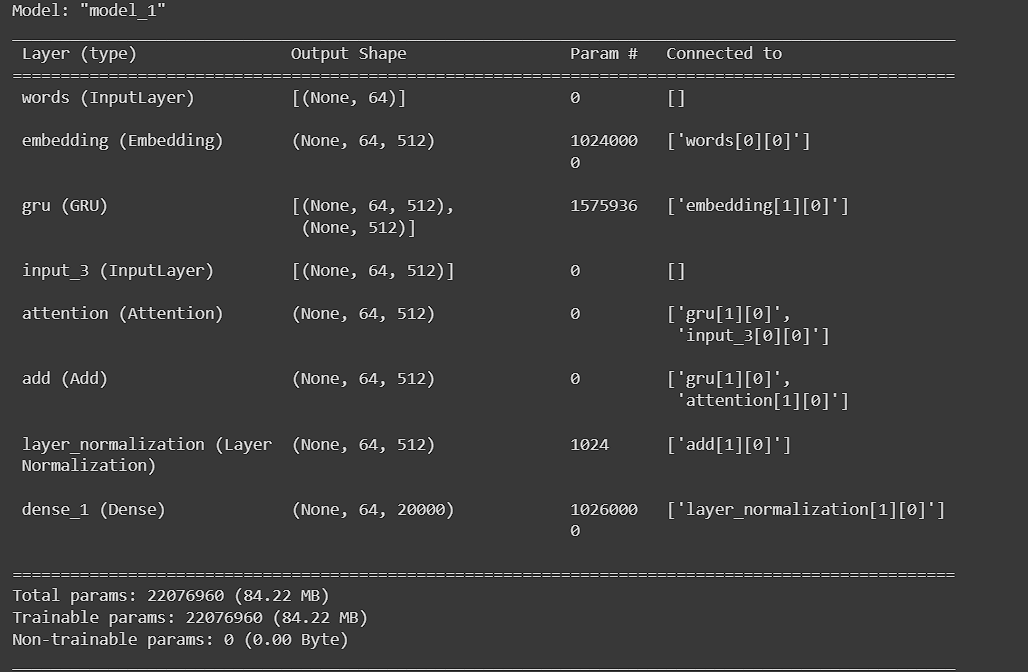


Figure 4 Decoder Parameters

**CHAPTER 6.2:**

**MODEL TRAINING**

The model is trained using an Adam optimizer and sparse categorical cross-entropy loss function. The training process involves optimizing the model's parameters to minimize the discrepancy between predicted and actual captions.

The research emphasizes the importance of custom loss functions and sequence-aware padding to handle variable-length captions. The training loop iterates through the dataset, updating the model weights to enhance its ability to generate accurate and contextually relevant captions.

By leveraging GPU acceleration and batching techniques, the code achieves an efficient training process. The model's performance is evaluated using a probabilistic prediction mechanism, demonstrating its capability to generate diverse and meaningful captions for input images.

**CHAPTER 7:**

**RESULTS AND ANALYSIS**

**CHAPTER 7:**

**RESULTS AND ANALYSIS**Top of Form

The model's efficacy is demonstrated through the generation of captions for sample images. The probabilistic prediction mechanism allows for diverse and contextually rich captions, showcasing the model's versatility. By utilizing attention mechanisms, the model excels in capturing fine-grained details in images, producing captions that align with human-like understanding.

The analysis highlights the model's potential for real-world applications, such as content retrieval and assistive technologies. The efficient training process and integration of state-of-the-art components contribute to the model's robustness, paving the way for further advancements in image captioning research.

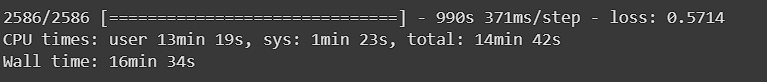


Figure 5 cross entropy loss

The model was able to give a cross entropy loss of 0.5714.

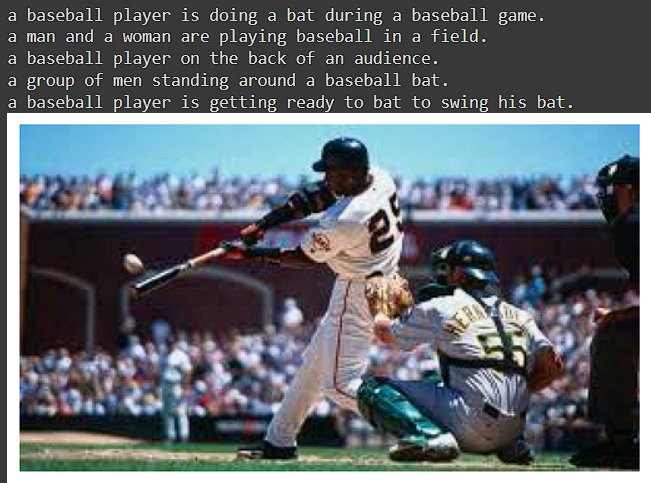


Figure 6 Result

Top of Form

**CHAPTER 8:**

**CONCLUSIONS**

**CHAPTER 8:**

**CONCLUSION**

In conclusion, the presented image captioning model showcases the fusion of cutting-edge computer vision and natural language processing techniques. The custom attention-enhanced GRU architecture, coupled with InceptionResNetV2 for feature extraction, contributes to the model's ability to generate descriptive and contextually relevant captions for diverse images.

The research underscores the significance of attention mechanisms in refining feature representations, emphasizing their role in capturing intricate visual relationships. The code's modular design and integration of pre-processing steps make it a valuable resource for researchers and practitioners interested in advancing image captioning capability

**CHAPTER 9:**

**REFERENCES**

**CHAPTER 9:**

**REFERENCES**

* **https://Machinelearningmastery.Com/The-attention-mechanism-from-scratch/**
* **https://Www.Cloudskillsboost.Google/Course\_templates/543**
* **https://Medium.Datadriveninvestor.Com/Attention-in-rnns-321fbcd64f05**
* **https://Arxiv.Org/Abs/1706.03762**
* **https://Towardsdatascience.Com/Image-captioning-in-deep-learning-9cd23fb4d8d2**

**CHAPTER 10:**

**PROGRAM (CODE)**

**CHAPTER 10:**

**PROGRAM(CODE)**

import time

from textwrap import wrap

import matplotlib.pylab as plt

import numpy as np

import tensorflow as tf

import tensorflow\_datasets as tfds

import tensorflow\_hub as hub

from tensorflow.keras import Input

from tensorflow.keras.layers import (

    GRU,

    Add,

    AdditiveAttention,

    Attention,

    Concatenate,

    Dense,

    Embedding,

    LayerNormalization,

    Reshape,

    StringLookup,

    TextVectorization,

)

print(tf.version.VERSION)

"""## Read and prepare dataset

We will use the TensorFlow datasets capability to read the [COCO captions](https://www.tensorflow.org/datasets/catalog/coco\_captions) dataset.

This version contains images, bounding boxes, labels, and captions from COCO 2014, split into the subsets defined by Karpathy and Li (2015) and takes

care of some data quality issues with the original dataset (for example, some

of the images in the original dataset did not have captions)

First, let's define some constants.<br>

In this lab, we will use a pretrained [InceptionResNetV2](https://www.tensorflow.org/api\_docs/python/tf/keras/applications/inception\_resnet\_v2/InceptionResNetV2) model from `tf.keras.applications` as a feature extractor, so some constants are comming from the InceptionResNetV2 model definition.<br>

So if you want to use other type of base model, please make sure to change these constants as well.

`tf.keras.applications` is a pretrained model repository like [TensorFlow Hub](https://tfhub.dev), but while Tensorflow Hub hosts models for different modalities including image, text, audio, and so on, `tf.keras.application` only hosts popular and stable models for images.<br>

However, `tf.keras.applications` is more flexible as it contains model metadata and it allow us to access and control the model behavior, while most of the TensorFlow Hub based models that only contains compiled SavedModels.<br>

So, for example, we can get output not only from the final layer of the model (e.g. flattend 1D Tensor output of CNN models), but also from intermediate layers (e.g. intermediate 3D Tensor) by accessing layer metadata.

"""

*# Change these to control the accuracy/speed*

VOCAB\_SIZE = 20000  *# use fewer words to speed up convergence*

ATTENTION\_DIM = 512  *# size of dense layer in Attention*

WORD\_EMBEDDING\_DIM = 128

*# InceptionResNetV2 takes (299, 299, 3) image as inputs*

*# and return features in (8, 8, 1536) shape*

FEATURE\_EXTRACTOR = tf.keras.applications.inception\_resnet\_v2.InceptionResNetV2(

    include\_top=False, weights="imagenet"

)

IMG\_HEIGHT = 299

IMG\_WIDTH = 299

IMG\_CHANNELS = 3

FEATURES\_SHAPE = (8, 8, 1536)

"""### Filter and Preprocess

Here we preprocess the dataset. The function below:

- resize image to (`IMG\_HEIGHT`, `IMG\_WIDTH`) shape

- rescale pixel values from [0, 255] to [0, 1]

- return image(`image\_tensor`) and captions(`captions`) dictionary.

\*\*Note\*\*: This dataset is too large to store in an local environment. Therefore, It is stored in a public GCS bucket located in us-central1.

So if you access it from a Notebook outside the US, it will be (a) slow and (b) subject to a network charge.

"""

GCS\_DIR = "gs://asl-public/data/tensorflow\_datasets/"

BUFFER\_SIZE = 1000

**def** get\_image\_label(example):

    caption = example["captions"]["text"][0]  *# only the first caption per image*

    img = example["image"]

    img = tf.image.resize(img, (IMG\_HEIGHT, IMG\_WIDTH))

    img = img / 255

    return {"image\_tensor": img, "caption": caption}

trainds = tfds.load("coco\_captions", split="train", data\_dir=GCS\_DIR)

trainds = trainds.map(

    get\_image\_label, num\_parallel\_calls=tf.data.AUTOTUNE

).shuffle(BUFFER\_SIZE)

trainds = trainds.prefetch(buffer\_size=tf.data.AUTOTUNE)

"""### Visualize

Let's take a look at images and sample captions in the dataset.

"""

f, ax = plt.subplots(1, 4, figsize=(20, 5))

for idx, data in enumerate(trainds.take(4)):

    ax[idx].imshow(data["image\_tensor"].numpy())

    caption = "\n".join(wrap(data["caption"].numpy().decode("utf-8"), 30))

    ax[idx].set\_title(caption)

    ax[idx].axis("off")

"""## Text Preprocessing

We add special tokens to represent the starts (`<start>`) and the ends (`<end>`) of sentences.<br>

Start and end tokens are added here because we are using an encoder-decoder model and during prediction, to get the captioning started we use `<start>` and since captions are of variable length, we terminate the prediction when we see the `<end>` token.

Then create a full list of the captions for further preprocessing.

"""

**def** add\_start\_end\_token(data):

    start = tf.convert\_to\_tensor("<start>")

    end = tf.convert\_to\_tensor("<end>")

    data["caption"] = tf.strings.join(

        [start, data["caption"], end], separator=" "

    )

    return data

trainds = trainds.map(add\_start\_end\_token)

"""## Preprocess and tokenize the captions

You will transform the text captions into integer sequences using the [TextVectorization](https://www.tensorflow.org/api\_docs/python/tf/keras/layers/TextVectorization) layer, with the following steps:

\* Use [adapt](https://www.tensorflow.org/api\_docs/python/tf/keras/layers/TextVectorization#adapt) to iterate over all captions, split the captions into words, and compute a vocabulary of the top `VOCAB\_SIZE` words.

\* Tokenize all captions by mapping each word to its index in the vocabulary. All output sequences will be padded to the length `MAX\_CAPTION\_LEN`. Here we directly specify `64` number which is sufficient for this dataset, but please note that this value should be computed by processing the entire dataset if you don't want to cut down very long sentense in a dataset.

\*\*Note\*\*: This process takes around 5 minutes.

"""

MAX\_CAPTION\_LEN = 64

*# We will override the default standardization of TextVectorization to preserve*

*# "<>" characters, so we preserve the tokens for the <start> and <end>.*

**def** standardize(inputs):

    inputs = tf.strings.lower(inputs)

    return tf.strings.regex\_replace(

        inputs, **r**"[!\"#$%&\(\)\\*\+.,-/:;=?@\[\\\]^\_`{|}~]?", ""

    )

*# Choose the most frequent words from the vocabulary & remove punctuation etc.*

tokenizer = TextVectorization(

    max\_tokens=VOCAB\_SIZE,

    standardize=standardize,

    output\_sequence\_length=MAX\_CAPTION\_LEN,

)

tokenizer.adapt(trainds.map(**lambda** x: x["caption"]))

"""

Let's try to tokenize a sample text"""

tokenizer(["<start> This is a sentence <end>"])

sample\_captions = []

for d in trainds.take(5):

    sample\_captions.append(d["caption"].numpy())

sample\_captions

print(tokenizer(sample\_captions))

"""Please note that all the sentenses starts and ends with the same token (e.g. '3' and '4'). These values represent start tokens and end tokens respectively.

You can also convert ids to original text.

"""

for wordid in tokenizer([sample\_captions[0]])[0]:

    print(tokenizer.get\_vocabulary()[wordid], end=" ")

"""Also, we can create Word <-> Index converters using `StringLookup` layer."""

*# Lookup table: Word -> Index*

word\_to\_index = StringLookup(

    mask\_token="", vocabulary=tokenizer.get\_vocabulary()

)

*# Lookup table: Index -> Word*

index\_to\_word = StringLookup(

    mask\_token="", vocabulary=tokenizer.get\_vocabulary(), invert=True

)

"""### Create a tf.data dataset for training

Now Let's apply the adapted tokenization to all the examples and create tf.data Dataset for training.

Here note that we are also creating labels by shifting texts from feature captions.<br>

If we have an input caption `"<start> I love cats <end>"`, its label should be `"I love cats <end> <padding>"`.<br>

With that, our model can try to learn to predict `I` from `<start>`.

The dataset should return tuples, where the first elements are features (`image\_tensor` and `caption`) and the second elements are labels (target).

"""

BATCH\_SIZE = 32

**def** create\_ds\_fn(data):

    img\_tensor = data["image\_tensor"]

    caption = tokenizer(data["caption"])

    target = tf.roll(caption, -1, 0)

    zeros = tf.zeros([1], dtype=tf.int64)

    target = tf.concat((target[:-1], zeros), axis=-1)

    return (img\_tensor, caption), target

batched\_ds = (

    trainds.map(create\_ds\_fn)

    .batch(BATCH\_SIZE, drop\_remainder=True)

    .prefetch(buffer\_size=tf.data.AUTOTUNE)

)

"""Let's take a look at some examples."""

for (img, caption), label in batched\_ds.take(2):

    print(**f**"Image shape: {img.shape}")

    print(**f**"Caption shape: {caption.shape}")

    print(**f**"Label shape: {label.shape}")

    print(caption[0])

    print(label[0])

"""## Model

Now let's design an image captioning model.<br>

It consists of an image encoder, followed by a caption decoder.

### Image Encoder

The image encoder model is very simple. It extracts features through a pre-trained model and passes them to a fully connected layer.

1. In this example, we extract the features from convolutional layers of InceptionResNetV2 which gives us a vector of (Batch Size, 8, 8, 1536).

1. We reshape the vector to (Batch Size, 64, 1536)

1. We squash it to a length of `ATTENTION\_DIM` with a Dense Layer and return (Batch Size, 64, ATTENTION\_DIM)

1. Later, the Attention layer attends over the image to predict the next word.

"""

FEATURE\_EXTRACTOR.trainable = False

image\_input = Input(shape=(IMG\_HEIGHT, IMG\_WIDTH, IMG\_CHANNELS))

image\_features = FEATURE\_EXTRACTOR(image\_input)

x = Reshape((FEATURES\_SHAPE[0] \* FEATURES\_SHAPE[1], FEATURES\_SHAPE[2]))(

    image\_features

)

encoder\_output = Dense(ATTENTION\_DIM, activation="relu")(x)

encoder = tf.keras.Model(inputs=image\_input, outputs=encoder\_output)

encoder.summary()

"""### Caption Decoder

The caption decoder incorporates an attention mechanism that focuses on different parts of the input image.

#### The attention head

The decoder uses attention to selectively focus on parts of the input sequence.

The attention takes a sequence of vectors as input for each example and returns an "attention" vector for each example.

Let's look at how this works:

<img src="https://user-images.githubusercontent.com/6895245/173408554-d4b6387b-248b-421e-8911-550d0561d001.png" alt="attention equation 1" width="800">

<img src="https://user-images.githubusercontent.com/6895245/173408648-38c6b582-a68b-4697-982a-1d885b83dd0b.png" alt="attention equation 2" width="800">

Where:

\* $s$ is the encoder index.

\* $t$ is the decoder index.

\* $\alpha\_{ts}$ is the attention weights.

\* $h\_s$ is the sequence of encoder outputs being attended to (the attention "key" and "value" in transformer terminology).

\* $h\_t$ is the decoder state attending to the sequence (the attention "query" in transformer terminology).

\* $c\_t$ is the resulting context vector.

\* $a\_t$ is the final output combining the "context" and "query".

The equations:

1. Calculates the attention weights, $\alpha\_{ts}$, as a softmax across the encoder's output sequence.

2. Calculates the context vector as the weighted sum of the encoder outputs.

Last is the $score$ function. Its job is to calculate a scalar logit-score for each key-query pair. There are two common approaches:

<img src="https://user-images.githubusercontent.com/6895245/173408773-3781cacc-de00-49c6-9909-f6cd65a0501b.png" alt="attention equation 4" width="800">

This notebook implement Luong-style attention using pre-defined `layers.Attention`.

#### Decoder Steps

The decoder's job is to generate predictions for the next output token.

1. The decoder receives current word tokens as a batch.

1. It embeds the word tokens to `ATTENTION\_DIM` dimension.

1. GRU layer keeps track of the word embeddings, and returns GRU outputs and states.

1. Bahdanau-style attention attends over the encoder's output feature by using GRU outputs as a query.

1. The attention outputs and GRU outputs are added (skip connection), and normalized in a layer normalization layer.

1. It generates logit predictions for the next token based on the GRU output.

We can define all the steps in Keras Functional API, but please note that here we instantiate layers that have trainable parameters so that we reuse the layers and the weights in inference phase.

"""

word\_input = Input(shape=(MAX\_CAPTION\_LEN), name="words")

embed\_x = Embedding(VOCAB\_SIZE, ATTENTION\_DIM)(word\_input)

decoder\_gru = GRU(

    ATTENTION\_DIM,

    return\_sequences=True,

    return\_state=True,

)

gru\_output, gru\_state = decoder\_gru(embed\_x)

decoder\_attention = Attention()

context\_vector = decoder\_attention([gru\_output, encoder\_output])

addition = Add()([gru\_output, context\_vector])

layer\_norm = LayerNormalization(axis=-1)

layer\_norm\_out = layer\_norm(addition)

decoder\_output\_dense = Dense(VOCAB\_SIZE)

decoder\_output = decoder\_output\_dense(layer\_norm\_out)

decoder = tf.keras.Model(

    inputs=[word\_input, encoder\_output], outputs=decoder\_output

)

tf.keras.utils.plot\_model(decoder)

decoder.summary()

"""### Training Model

Now we defined the encoder and the decoder. Let's combine them into an image model for training.<br>

It has two inputs (`image\_input` and `word\_input`, and an output (`decoder\_output`). This definition should correspond to the definition of the dataset pipeline.

"""

image\_caption\_train\_model = tf.keras.Model(

    inputs=[image\_input, word\_input], outputs=decoder\_output

)

"""### Loss Function

The loss function is a simple cross-entropy, but we need to remove padding (`0`) when calculating it.<br>

So here we extract the length of the sentence (non-0 part), and compute the average of the loss only over the valid sentence part.

"""

loss\_object = tf.keras.losses.SparseCategoricalCrossentropy(

    from\_logits=True, reduction="none"

)

**def** loss\_function(real, pred):

    loss\_ = loss\_object(real, pred)

*# returns 1 to word index and 0 to padding (e.g. [1,1,1,1,1,0,0,0,0,...,0])*

    mask = tf.math.logical\_not(tf.math.equal(real, 0))

    mask = tf.cast(mask, dtype=tf.int32)

    sentence\_len = tf.reduce\_sum(mask)

    loss\_ = loss\_[:sentence\_len]

    return tf.reduce\_mean(loss\_, 1)

image\_caption\_train\_model.compile(

    optimizer="adam",

    loss=loss\_function,

)

"""## Training loop

Now we can train the model using the standard `model.fit` API.<br>

It takes around 15-20 minutes with NVIDIA T4 GPU to train 1 epoch.

"""

*# Commented out IPython magic to ensure Python compatibility.*

*# %%time*

*# history = image\_caption\_train\_model.fit(batched\_ds, epochs=1)*

"""## Caption!

The predict step is different from the training, since we need to keep track of the GRU state during the caption generation, and pass a predicted word to the decoder as an input at the next time step.

In order to do so, let's define another model for prediction while using the trained weights, so that it can keep and update the GRU state during the caption generation.

"""

gru\_state\_input = Input(shape=(ATTENTION\_DIM), name="gru\_state\_input")

*# Reuse trained GRU, but update it so that it can receive states.*

gru\_output, gru\_state = decoder\_gru(embed\_x, initial\_state=gru\_state\_input)

*# Reuse other layers as well*

context\_vector = decoder\_attention([gru\_output, encoder\_output])

addition\_output = Add()([gru\_output, context\_vector])

layer\_norm\_output = layer\_norm(addition\_output)

decoder\_output = decoder\_output\_dense(layer\_norm\_output)

*# Define prediction Model with state input and output*

decoder\_pred\_model = tf.keras.Model(

    inputs=[word\_input, gru\_state\_input, encoder\_output],

    outputs=[decoder\_output, gru\_state],

)

"""

1. Initialize the GRU states as zero vectors.

1. Preprocess an input image, pass it to the encoder, and extract image features.

1. Setup word tokens of `<start>` to start captioning.

1. In the for loop, we

    - pass word tokens (`dec\_input`), GRU states (`gru\_state`) and image features (`features`) to the prediction decoder and get predictions (`predictions`), and the updated GRU states.

    - select Top-K words from logits, and choose a word probabilistically so that we avoid computing softmax over VOCAB\_SIZE-sized vector.

    - stop predicting when the model predicts the `<end>` token.

    - replace the input word token with the predicted word token for the next step."""

MINIMUM\_SENTENCE\_LENGTH = 5

*## Probabilistic prediction using the trained model*

**def** predict\_caption(filename):

    gru\_state = tf.zeros((1, ATTENTION\_DIM))

    img = tf.image.decode\_jpeg(tf.io.read\_file(filename), channels=IMG\_CHANNELS)

    img = tf.image.resize(img, (IMG\_HEIGHT, IMG\_WIDTH))

    img = img / 255

    features = encoder(tf.expand\_dims(img, axis=0))

    dec\_input = tf.expand\_dims([word\_to\_index("<start>")], 1)

    result = []

    for i in range(MAX\_CAPTION\_LEN):

        predictions, gru\_state = decoder\_pred\_model(

            [dec\_input, gru\_state, features]

        )

*# draws from log distribution given by predictions*

        top\_probs, top\_idxs = tf.math.top\_k(

            input=predictions[0][0], k=10, sorted=False

        )

        chosen\_id = tf.random.categorical([top\_probs], 1)[0].numpy()

        predicted\_id = top\_idxs.numpy()[chosen\_id][0]

        result.append(tokenizer.get\_vocabulary()[predicted\_id])

        if predicted\_id == word\_to\_index("<end>"):

            return img, result

        dec\_input = tf.expand\_dims([predicted\_id], 1)

    return img, result

"""Let's caption!"""

filename ="/content/y.jpeg" *# you can also try surf.jpeg*

for i in range(5):

    image, caption = predict\_caption(filename)

    print(" ".join(caption[:-1]) + ".")

img = tf.image.decode\_jpeg(tf.io.read\_file(filename), channels=IMG\_CHANNELS)

plt.imshow(img)

plt.axis("off");