Automated Image Captioning using CNN and RNN

REVIEW - 3

Course Code: CSE3013 – Artificial Intelligence

Slot: E2 + TE2

Professor: Dr. Anitha A

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

By,

Pranjal Karn	**18BCE0134**
Aniket Kumar	**18BCE0101**

Index

Topics Pg. No.

1. Abstract … 1
2. Keywords … 1
3. Introduction … 2
4. Literature Review … 3
5. H/W and S/W Requirements … 6
    5.1. Hardware Requirements … 6
    5.2. Software Requirements … 6
    5.3. Dataset used … 6
6. Architecture/ Working of system … 6
    6.1. CNN Encoder … 6
    6.2. RNN Decoder … 7
    6.3. Inception V3 … 8
7. Experimental Analysis … 8
    7.1. Code … 8
    7.2. Screenshot … 17
8. Result and Discussion … 21
9. Future Work … 21
10. Conclusion … 22
11. References … 22

1. Abstract

Captioning an image involves generating a human readable textual description given an image, such as a photograph. It is an easy problem for a human, but very challenging for a machine as it involves bothunderstanding the content of an image and how to translate this understanding into natural language. Recently, deep learning methods have displaced classical methods and are achieving state-of-the-art results for the problem of automatically generating descriptions, called "captions," for images. With the advancements in technology and ease of computation of extensive data has made it possible for us to easily apply deep learning in several projects using our personal workstation. Describing an image is the problem of generating a human- readable textual description of an image, such as a photograph of an object or scene. A solution requires both that the content of the image be understood and translated to meaning in the terms of words, and that the words must string together to be comprehensible. It combines both computer vision using deep learning and natural language processing and marks a true challenging problem in broader artificial intelligence.

In this project we create an automated image captioning model using Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) to produce a series of texts that best describe the image. This model is trained on MS- COSO dataset. Image captioning requires that you create a complex deep learning model with two components: a CNN that transforms an input image into a set of features, and an RNN that turns those features into rich, descriptive language. This project uses the encoder-decoder model, in which the CNN which performs the feature extraction and the output of the encoder is fed to the decoder which processes the classified features into appropriate sentences. The feature extraction will be done by the latest Inception V3 module-50 technology with means of transfer learning so that we can modify the project specific to our purpose. The language model uses natural language toolkit for simple natural language processing and the architecture used for recurrent neural network is long-short term memory.

2. Keywords: -

Image captioning, CNN, RNN, encoder- decoder, Deep learning, LSTM, NLTK, MS -COSO dataset.

3. Introduction

This project is aimed towards creating an auto captioning model for an image. This can be achieved by using Convolutional Neural Network for image feature classification and object localization. After that Recurrent Neural Networks can be used for text generation. We used Long Short-Term Memory (LSTM) a type of RNN model which can learn from sequential data like a series of words and characters. These networks use hidden layers that link the input and output layers. Which creates a memory loop so that the model can learn from previous outputs. So basically, CNN works with spatial features and RNN helps in solving sequential data.

This project uses advanced methods of computer vision using Deep Learning and natural language processing using a Recurrent Neural Network. Deep Learning is a machine learning technique with which we can program the computer to learn complex models and understand patterns in a large dataset.
The combination of increasing computation speed, wealth of research and the rapid growth of technology. Deep Learning and AI is experiencing massive growth worldwide and will perhaps be one of the world's biggest industries in the near future. The 21st century is the birth of AI revolution, and data becoming the new 'oil' for it. Every second in today's world large amounts of data is being generated. We need to build models that can study these datasets and come up with patterns or find solution for analysis and research. This can be achieved solely due to deep learning.

Computer Vision is a cutting-edge field of computer science that aims to enable computers to understand what is being seen in an image. Computers don't perceive the world like humans do. For them the perception is just sets of raw numbers and because of several limitations like type of camera, lighting conditions, clarity, scaling, viewpoint variation etc. make computer vision so hard to process as it is very tough to build a robust model that can work on every condition.

The neural network architectures normally we see were trained using the current inputs only. We did not consider previous inputs when generating the current output. In other words, our systems did not have any memory elements. RNNs address this very basic and important issue by using memory (i.e. past inputs to the network) when producing the current output. These are the things we will be learning ahead in this project.

4. Literature Review

4.1. A Survey on Automatic Image Caption Generation By: Shuang Bai*

Image captioning approach robotically producing a caption for a photo. As a lately emerged research place, its miles attracting more and more interest. To achieve the purpose of picture captioning, semantic records of pictures desires to be captured and expressed in natural languages. Connecting both studies communities of computer vision and herbal language processing, image captioning is a pretty difficult project. Various tactics have been proposed to remedy this trouble. In this paper, we gift a survey on advances in picture captioning research. Based on the method followed, we classify image captioning procedures into different classes. Representative strategies in each category are summarized, and their strengths and limitations are mentioned. In this paper, we first talk techniques utilized in early paintings which are in particular retrieval and template based totally. Then, we consciousness our essential attention on neural network primarily based techniques, which provide state of the artwork consequences. Neural community-based methods are further divided into subcategories primarily based on the specific framework they use.

Each subcategory of neural community based totally methods are mentioned in element. After that, state of the art strategies is as compared on benchmark datasets. Following that, discussions on destiny research instructions are provided.

4.2. Image Captioning with Convolutional NeuralNetworks By: Michal Najman

In this thesis, we difficult on picture captioning concerning specifically dense image captioning. We gift technical basics of a model striving to clear up any such challenge. Concretely, an in-depth shape of Dense Cap and Neural Image Caption is discussed. Experimentally, we have a look at effects of Dense Cap and analyses the model's weaknesses. We display that 92% of the generated captions are same to a caption within the education set even as the best of those and the radical ones remains the same. We propose a criterion that significantly reduces a fixed of captions addressing an image whilst SPICE rating of the set is maintained.

4.3. Improving Image Captioning by Leveraging Knowledge Graphs

By: Yimin Zhou, Yiwei Sun, Vasant Honavar Artificial Intelligent Research Laboratory the Pennsylvania State University

We explore the use of a knowledge graphs that capture general or commonsense knowledge,

to augment the information extracted from images by the state-of- the-art methods for image captioning. The results of our experiments, on several benchmark data sets such as MS COCO, as measured by CIDEr-D, a performance metric for image captioning, show that the variants of the state-of- the-art methods for image captioning that make use of the information extracted from knowledge graphs can substantially outperform those that rely solely on the information extracted from images.

CNet-NIC uses YOLO9000, a state-of-the-art general-purpose real-time object recognition module that is trained to recognize 9000 object categories. YOLO9000 takes an image as input and produces as output, a collection of terms that refer to objects in the scene. CNet-NIC use an external knowledge graph, specifically, Concept Net a labeled graph which connects words and phrases of natural language connected by edges that denote commonsense relationships between them, to infer two sets of terms related to the words that describe the objects found in the scene by the object recognition module. The first set of terms are retrieved based on the individual objects in the scene. The second set of terms are retrieved based on the entire collection of objects in the scene. The resulting terms are then provided to a pre-trained RNN to obtain the corresponding vector space embedding of the terms. A CNN is used to obtain vector space embedding of the image features. The two resulting vector space embeddings are used to specify the initial state of an LSTM-based RNN which is trained to produce the caption for the input image.

4.4. Image Captioning with Object Detection and Localization

By: Zhongliang Yang, Yu-Jin Zhang, Sadaqat ur Rehman, Yongfeng Huang,
Department of Electronic Engineering, Tsinghua University, Beijing

Automatically generating a natural language description of an image is a task close to the heart of image understanding. In this paper, we present a multi- model neural network method closely related to the human visual system that automatically learns to describe the content of images. Our model consists of two sub-models: an object detection and localization model, which extract the information of objects and their spatial relationship in images respectively.

Besides, a deep recurrent neural network (RNN) based on long short-term memory (LSTM) units with attention mechanism for sentences generation. Each word of the description will be automatically aligned to different objects of the input image when it is generated. This is

similar to the attention mechanism of the human visual system. Experimental results on the MS -COSO dataset showcase the merit of the proposed method, which outperform previous benchmark models.

4.5 Convolutional Image Captioning

By: Jyoti Aneja , Aditya Deshpande, Alexander G. Schwing University of Illinois at Urbana-Champaign

In recent years significant progress has been made in image captioning, using Recurrent Neural Networks powered by long-short-term-memory (LSTM) units. Despite mitigating the vanishing gradient problem, and despite their compelling ability to memorize dependencies, LSTM units are complex and inherently sequential across time. However, the complex addressing and overwriting mechanism combined with inherently sequential processing, and significant storage required due to back-propagation through time (BPTT), poses challenges during training. Despite the fact that the above RNNs based on LSTM/GRU deliver remarkable results, e.g., for image captioning, their training procedure is all but trivial. For instance, while the forward pass during training can be in parallel across samples, it is inherently sequential in time, limiting the parallelism. To address this issue, they proposed a Pixel CNN architecture for conditional image generation that approximates an RNN. This article demonstrates that convolutional architectures with attention achieve state-of the-art performance on machine translation tasks. In spirit similar is our approach for image captioning, which is convolutional but addresses a different task.

5. H/W AND S/W REQUIREMENTS

5.1. HARDWARE

- Processor: Minimum 1 GHz; Recommended 2GHz or more.
- Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)
- Hard Drive: Minimum 32 GB; Recommended 64 GB or more.
- Memory (RAM): Minimum 1 GB; Recommended 4 GB or above.

5.2. SOFTWARE

- Python
- OpenCV/NLTK
- Keras
- TensorFlow
- Atom editor
- Windows/Linux

5.3. DATASET/TOOL USED

MS-COCO dataset. The dataset contains over 82,000 images, each of which has at least 5 different caption annotations.

6. Architecture / Working of the System

Image captioning requires that you create a complex deep learning model with two modules:

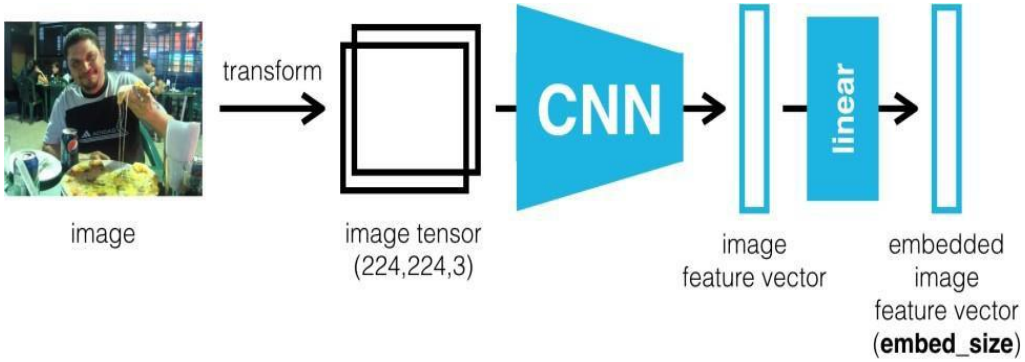
1) a CNN that transforms an input image into a set of features, and

2) an RNN that turns those features into rich, descriptive language.

6.1. CNN Encoder

The encoder is based on a Convolutional neural network that encodes an image into a compact representation. The CNN-Encoder is an Inception V3 module (Residual Network).

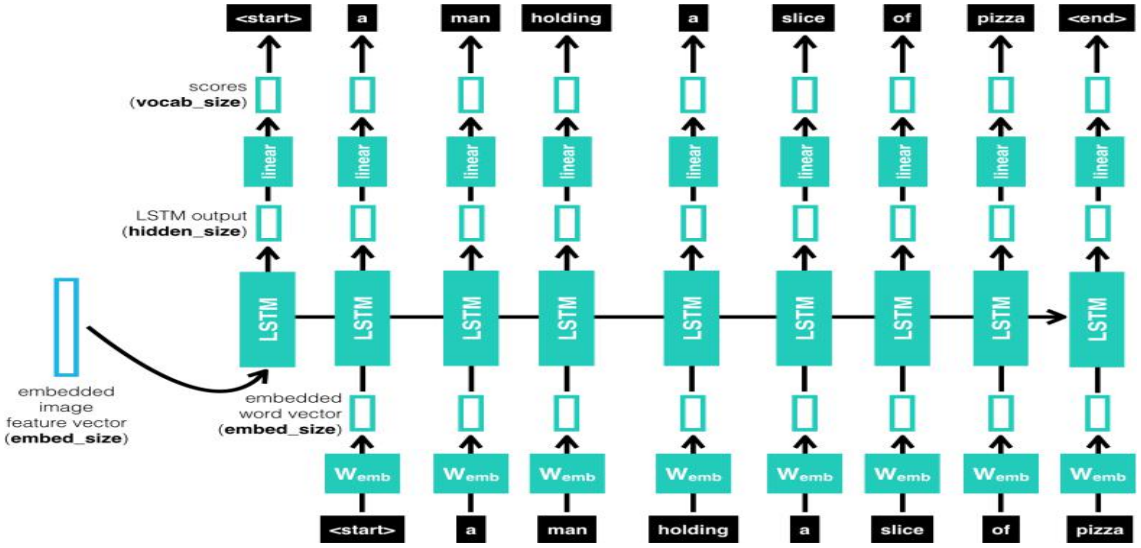
These kinds of network help regarding to the vanishing and exploding gradient type of problems. The main idea relies on the use of skip connections which allows to take the activations from one layer and suddenly feed it to another layer, even much deeper in the neural network and using that, we can build Inception V3 modules which enables to train very deep networks.



(Figure 1: DFD)

6.2. RNN Decoder

The CNN encoder is followed by a recurrent neural network that generates a corresponding sentence. The RNN-Decoder consists in a single LSTM layer followed by one fully-connected (linear) layer. The RNN network is trained on the MS -COSO dataset. It is used to predict the next word of a sentence based on previous words. The captions are presented as a list of tokenized words so that the RNN model can train and back propagate to reduce errors and generate better and more understandable texts describing the image.



(Figure 2: Working)

6.3. Inception V3 module:

There are 4 versions. The first Google Net must be the Inception-v1 , but there are numerous typos in Inception-v3 which lead to wrong descriptions about Inception versions. These maybe due to the intense ILSVRC competition at that moment. Consequently, there are many reviews in the internet mixing up between v2 and v3. Some of the reviews even think that v2 and v3 are the same with only some minor different settings.

7. Experimental Analysis

7.1. Coding

```
!pip install tqdm
import tensorflow as tf
import matplotlib.pyplot as plt
from tqdm import tqdm
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import re
import numpy as np
import os
import time
import json
from glob import glob
from PIL import Image
import pickle



# Download caption annotation files
annotation_folder = '/annotations/'
if not os.path.exists(os.path.abspath('.') + annotation_folder):
  annotation_zip = tf.keras.utils.get_file('captions.zip',
                                           cache_subdir=os.path.abspath('.'),
                                           origin = 'http://images.cocodataset.org/a
nnotations/annotations_trainval2014.zip',
                                           extract = True)
  annotation_file = os.path.dirname(annotation_zip)+'/annotations/captions_train201
4.json'


  os.remove(annotation_zip)



# Download image files
```

```
image_folder = '/train2014/'
if not os.path.exists(os.path.abspath('.') + image_folder):
  image_zip = tf.keras.utils.get_file('train2014.zip',
                                      cache_subdir=os.path.abspath('.'),
                                      origin = 'http://images.cocodataset.org/zips/
train2014.zip',
                                      extract = True)
  PATH = os.path.dirname(image_zip) + image_folder
  os.remove(image_zip)
else:
  PATH = os.path.abspath('.') + image_folder



# Read the json file
with open(annotation_file, 'r') as f:
    annotations = json.load(f)




# Store captions and image names in vectors
all_captions = []
all_img_name_vector = []

for annot in annotations['annotations']:
    caption = '<start> ' + annot['caption'] + ' <end>'
    image_id = annot['image_id']
    full_coco_image_path = PATH + 'COCO_train2014_' + '%012d.jpg' % (image_id)

    all_img_name_vector.append(full_coco_image_path)
    all_captions.append(caption)



# Shuffle captions and image_names together
# Set a random state
train_captions, img_name_vector = shuffle(all_captions,
                                          all_img_name_vector,
                                          random_state=1)



# Select the first 30000 captions from the shuffled set
num_examples = 30000
train_captions = train_captions[:num_examples]
img_name_vector = img_name_vector[:num_examples]
len(train_captions), len(all_captions)
```

```
(30000, 414113)

def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
image_model = tf.keras.applications.InceptionV3(include_top=False,
                                                 weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)



# Get unique images
encode_train = sorted(set(img_name_vector))



# Feel free to change batch_size according to your system configuration
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(
  load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(16)

for img, path in tqdm(image_dataset):
  batch_features = image_features_extract_model(img)
  batch_features = tf.reshape(batch_features,
                              (batch_features.shape[0], -1, batch_features.shape[3]))
)

  for bf, p in zip(batch_features, path):
    path_of_feature = p.numpy().decode("utf-8")
    np.save(path_of_feature, bf.numpy())



# Find the maximum length of any caption in our dataset
def calc_max_length(tensor):
    return max(len(t) for t in tensor)



# Choose the top 5000 words from the vocabulary
top_k = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
```

```
                                                  oov_token="<unk>",
                                                  filters='!"#$%&()*+.,-/:;=?@[\]^_
`{|}~ ')
tokenizer.fit_on_texts(train_captions)
train_seqs = tokenizer.texts_to_sequences(train_captions)
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'



# Create the tokenized vectors
train_seqs = tokenizer.texts_to_sequences(train_captions)
cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding='pos
t')



# Calculates the max_length, which is used to store the attention weights
max_length = calc_max_length(train_seqs)



# Create training and validation sets using an 80-20 split
img_name_train, img_name_val, cap_train, cap_val = train_test_split(img_name_vector
,
                                                                    cap_vector,
                                                                    test_size=0.2,
                                                                    random_state=0)
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
(24000, 24000, 6000, 6000)
BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = top_k + 1
num_steps = len(img_name_train) // BATCH_SIZE


# Shape of the vector extracted from InceptionV3 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 64



# Load the numpy files
def map_func(img_name, cap):
  img_tensor = np.load(img_name.decode('utf-8')+'.npy')
```

```
  return img_tensor, cap
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))



# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)



# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.W1 = tf.keras.layers.Dense(units)
    self.W2 = tf.keras.layers.Dense(units)
    self.V = tf.keras.layers.Dense(1)

  def call(self, features, hidden):
    # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)
    # hidden shape == (batch_size, hidden_size)
    # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
    hidden_with_time_axis = tf.expand_dims(hidden, 1)



    # score shape == (batch_size, 64, hidden_size)
    score = tf.nn.tanh(self.W1(features) + self.W2(hidden_with_time_axis))



    # attention_weights shape == (batch_size, 64, 1)
    # you get 1 at the last axis because you are applying score to self.V
    attention_weights = tf.nn.softmax(self.V(score), axis=1)



    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * features
    conext_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights
class CNN_Encoder(tf.keras.Model):
```

```
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
class RNN_Decoder(tf.keras.Model):
  def __init__(self, embedding_dim, units, vocab_size):
    super(RNN_Decoder, self).__init__()
    self.units = units

    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(self.units,
                                   return_sequences=True,
                                   return_state=True,
                                   recurrent_initializer='glorot_uniform')
    self.fc1 = tf.keras.layers.Dense(self.units)
    self.fc2 = tf.keras.layers.Dense(vocab_size)

    self.attention = BahdanauAttention(self.units)

  def call(self, x, features, hidden):
    # defining attention as a separate model
    context_vector, attention_weights = self.attention(features, hidden)

    # x shape after passing through embedding == (batch_size, 1, embedding_dim)
    x = self.embedding(x)

    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
    x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # shape == (batch_size, max_length, hidden_size)
    x = self.fc1(output)

    # x shape == (batch_size * max_length, hidden_size)
    x = tf.reshape(x, (-1, x.shape[2]))

    # output shape == (batch_size * max_length, vocab)
    x = self.fc2(x)
```

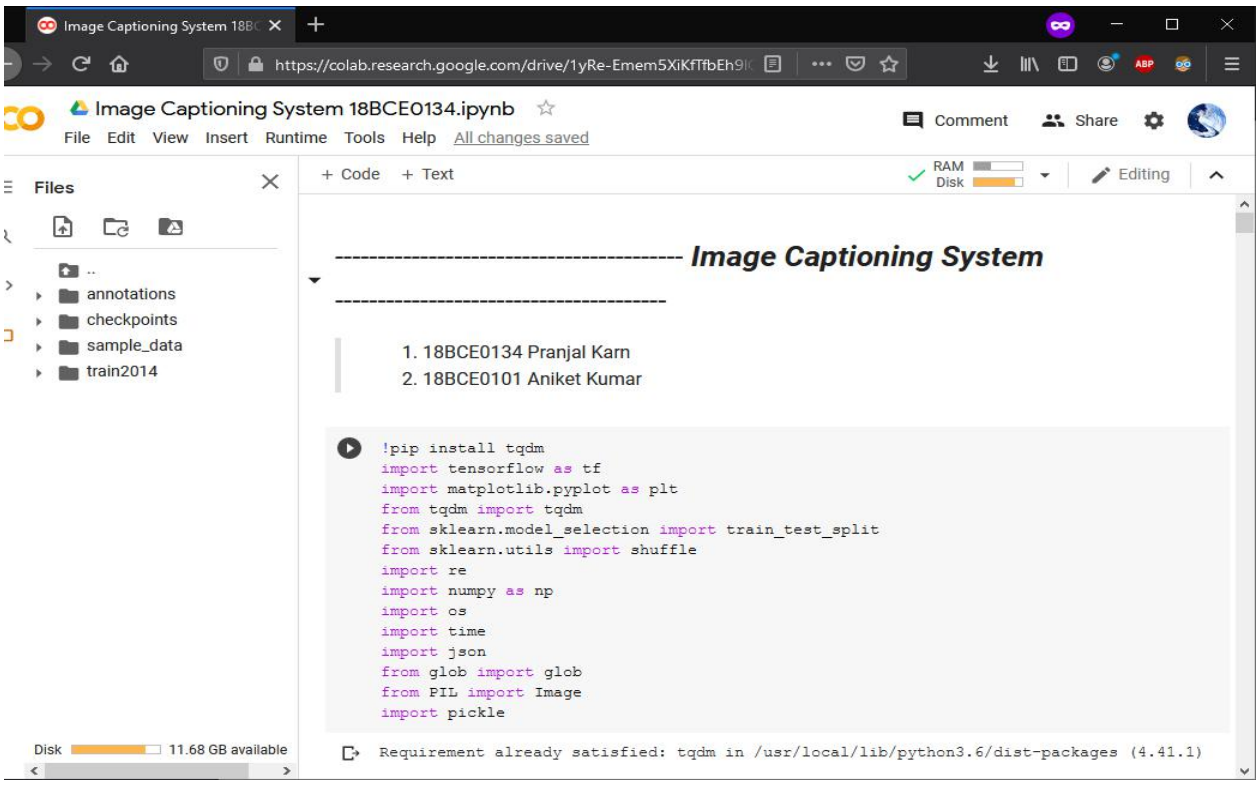
```
    return x, state, attention_weights

  def reset_state(self, batch_size):
    return tf.zeros((batch_size, self.units))
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)

  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask

  return tf.reduce_mean(loss_)
Step 10: Checkpoint
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer = optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
start_epoch = 0
if ckpt_manager.latest_checkpoint:
  start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])

  # restoring the latest checkpoint in checkpoint_path
  ckpt.restore(ckpt_manager.latest_checkpoint)

# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
@tf.function
def train_step(img_tensor, target):
  loss = 0

  # initializing the hidden state for each batch
  # because the captions are not related from image to image
  hidden = decoder.reset_state(batch_size=target.shape[0])

  dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.shape[0], 1
)

  with tf.GradientTape() as tape:
      features = encoder(img_tensor)
```

```
      for i in range(1, target.shape[1]):
          # passing the features through the decoder
          predictions, hidden, _ = decoder(dec_input, features, hidden)

          loss += loss_function(target[:, i], predictions)

          # using teacher forcing
          dec_input = tf.expand_dims(target[:, i], 1)

  total_loss = (loss / int(target.shape[1]))

  trainable_variables = encoder.trainable_variables + decoder.trainable_variables

  gradients = tape.gradient(loss, trainable_variables)

  optimizer.apply_gradients(zip(gradients, trainable_variables))

  return loss, total_loss
EPOCHS = 5

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(dataset):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            print ('Epoch {} Batch {} Loss {:.4f}'.format(
              epoch + 1, batch, batch_loss.numpy() / int(target.shape[1])))

    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if epoch % 5 == 0:
      ckpt_manager.save()

    print ('Epoch {} Loss {:.6f}'.format(epoch + 1,
                                        total_loss/num_steps))
    print ('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()

def evaluate(image):
```

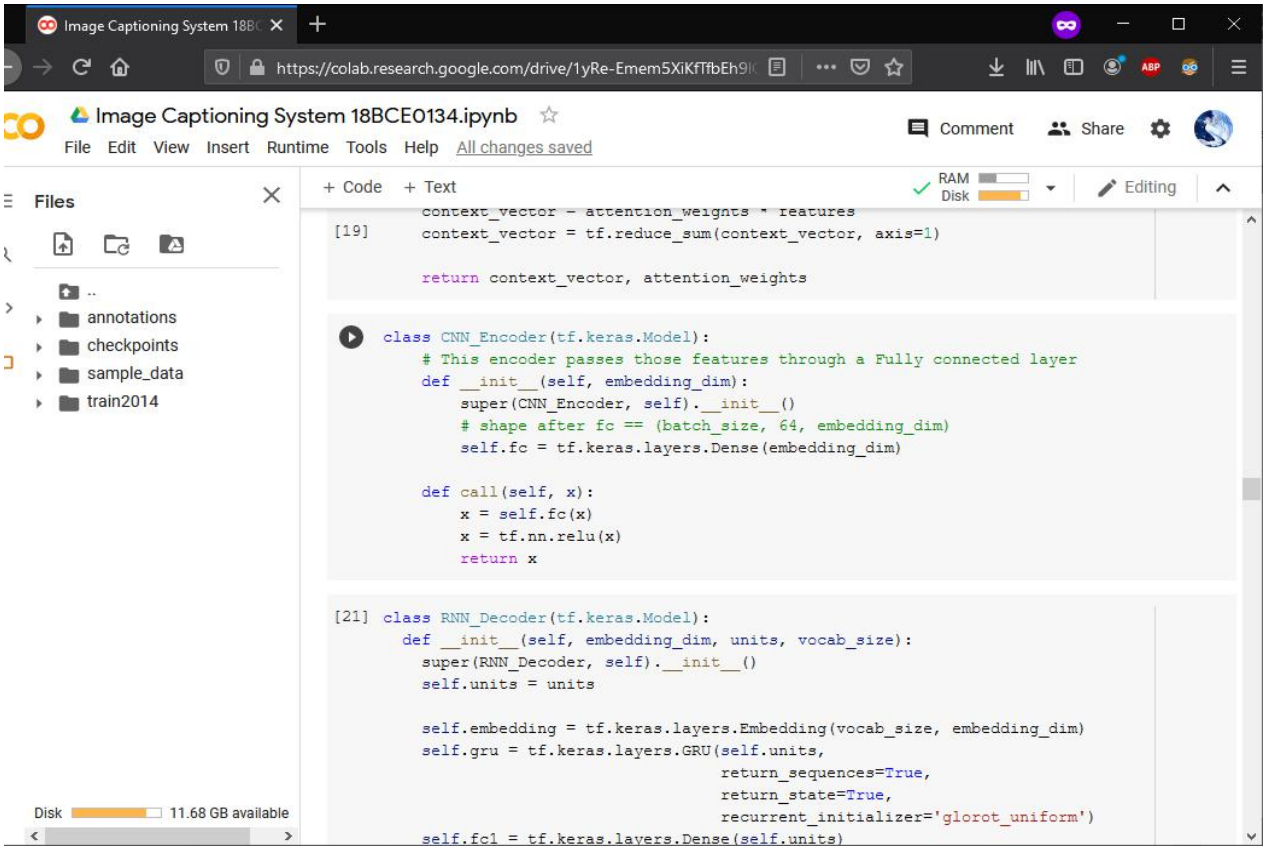
```
    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_t
ensor_val.shape[3]))

    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 0)
    result = []
    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input, features, hidde
n)


        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

        predicted_id = tf.random.categorical(predictions, 1)[0][0].numpy()
        result.append(tokenizer.index_word[predicted_id])

        if tokenizer.index_word[predicted_id] == '<end>':
            return result, attention_plot

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot
def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result//2, len_result//2, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

    plt.tight_layout()
    plt.show()

# captions on the validation set
rid = np.random.randint(0, len(img_name_val))
image = img_name_val[rid]
```

```
real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid] if i not in
[0]])
result, attention_plot = evaluate(image)

print ('Real Caption:', real_caption)
print ('Prediction Caption:', ' '.join(result))
plot_attention(image, result, attention_plot)
Real Caption: <start> some horses pull a wagon up a hill on to a road <end>
Prediction Caption: a black and white horses horses down the horses <end>
Finally!! Play Time, We can add the url of any freely available picture on internet,
 and wait for the model to work.

image_url = "Enter your preferred lib"
image_extension = image_url[image_url.rindex('.'):]
image_path = tf.keras.utils.get_file('image'+image_extension,
                                     origin=image_url)

result, attention_plot = evaluate(image_path)
print ('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

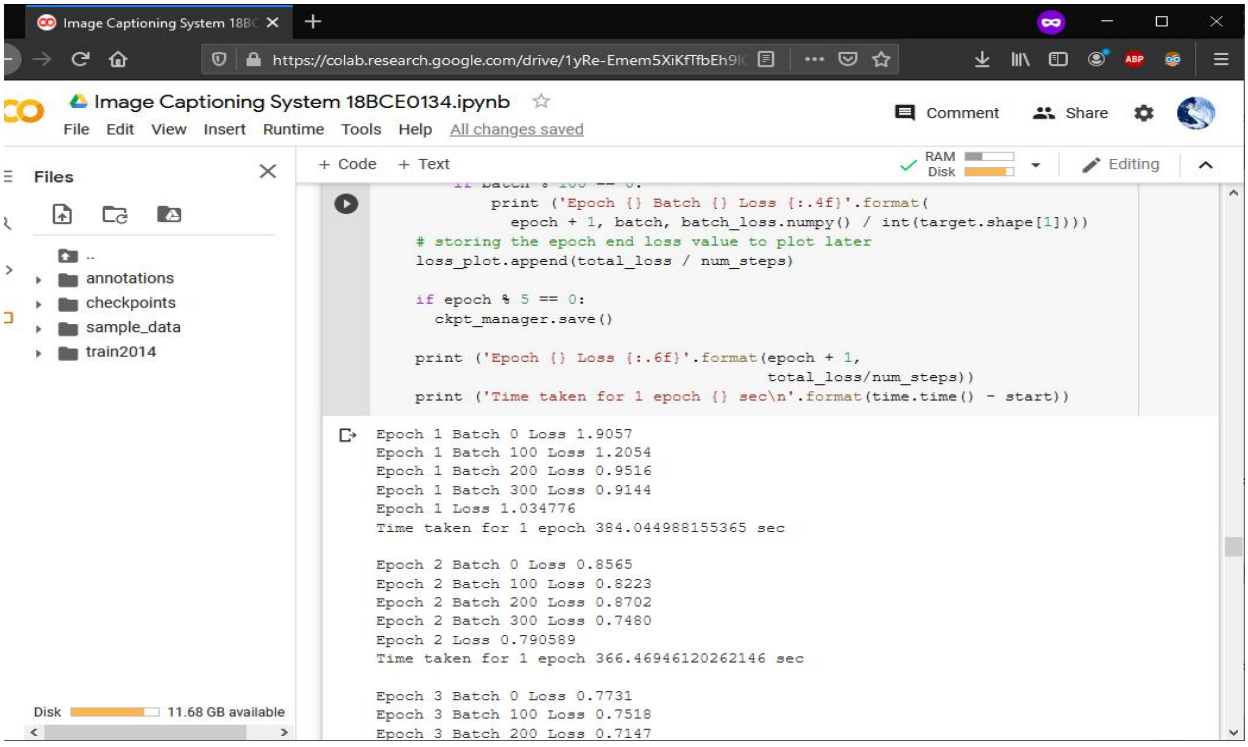
7.2. Screenshots



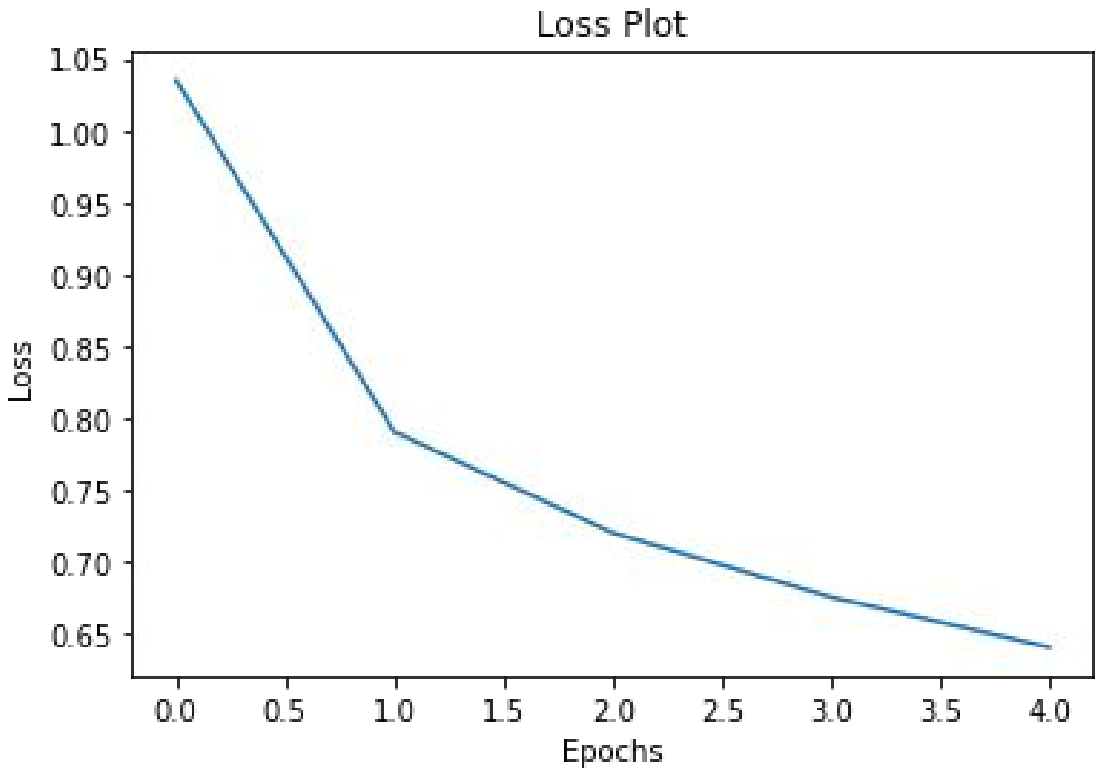
(Figure 3: Libraries used)



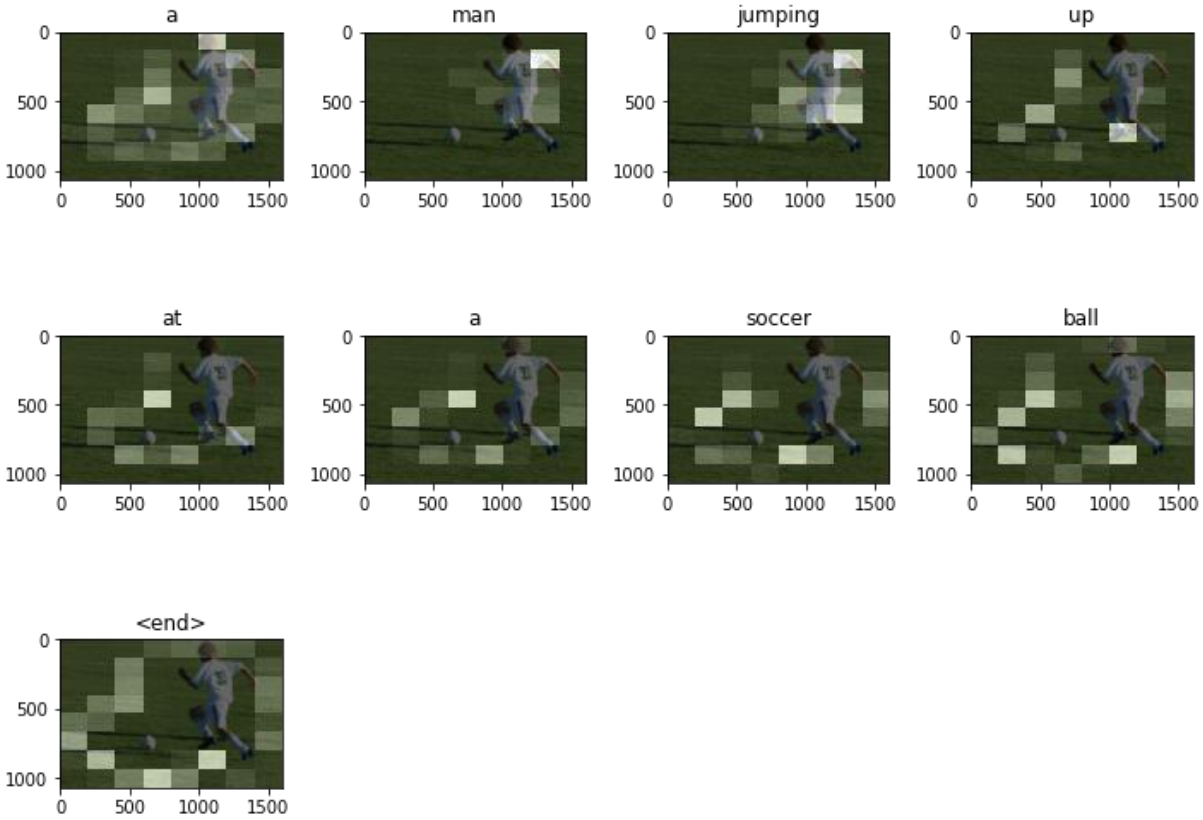
(Figure 4: CNN encoder and RNN decoder function)



(Figure 5: Model being trained)



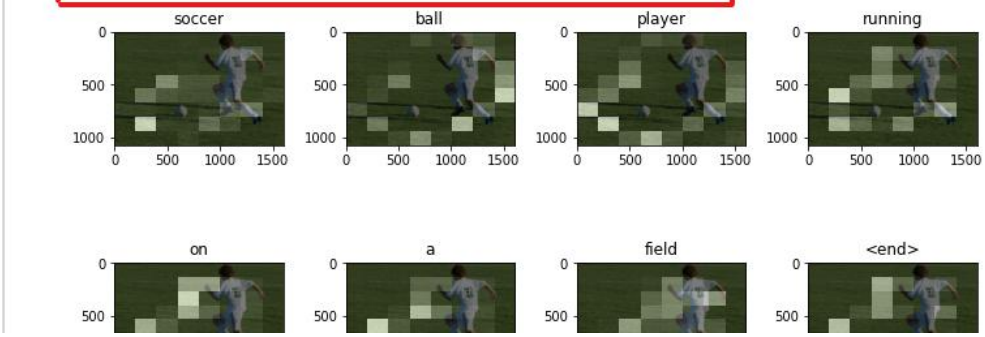
(Figure 6: Loss Plot)



(Figure 7: Attention Graph)

```
image_url = 'https://images.freeimages.com/images/large-previews/a17/soccer-dribble-1436311.jpg'
image_extension = image_url[image_url.rindex('.'):]
image_path = tf.keras.utils.get_file('image'+image_extension,
                                     origin=image_url)

result, attention_plot = evaluate(image_path)
print ('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: soccer ball player running on a field <end>
```



(Figure 8: Prediction)

(Figure 9: Actual image)

8. Result and Discussion

As we can see that our model predicts near to perfect Captions with some errors in the grammatical part of the statement. Such errors can be removed by training another model of english conversations and sentences. Such models can be generated by using movie subtitles (English movie) for a vast variety of sentences and various forms of sentences.

Other than the errors found in the grammar, rest of our model is working fine. Further, due to the memory intensive job we shifted onto the Google colab Platform to show the demonstration, but it kind of restricts further usage of our trained model and future enhancements. So to counter that we would like to generate the model locally in order to get more of a in-hand experience of our model.

9. Future Work

- Aid to the blind—We can create a product for the blind which will guide them travelling on the roads without the support of anyone else. We can do this by first converting the scene into text and then the text to voice. Both are now famous applications of Deep Learning.
- Automatic Captioning can help, make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.
- CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.

10. Conclusion

We combine "Image Labeling" and "Automatic Machine Translation" into an end-to-end hybrid neural network system. The developed model is capable to autonomously view an image and generate a reasonable description in natural language with reasonable accuracy and naturalness. Further extension of the present model can be in regard to increasing additional CNN layers or increasing/implementing pre-training, which could improve the accuracy of the predictions.

11. References

P. Anderson, B. Fernando, M. Johnson, and S. Gould. Spice: Semantic propositional image caption evaluation. In ECCV, 2016

S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual Question Answering. In International Conference on Computer Vision (ICCV), 2015

M. Denkowski and A. Lavie. Meteor universal: Language specific translation evaluation for any target language. In Proceedings of the EACL 2014 Workshop on Statistical Machine Translation, 2014

M. Hodosh, P. Young, and J. Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. J. Artif. Int. Res., 47(1), May 2013.

J. Mao, W. Xu, Y. Yang, J. Wang, and A. L. Yuille. Deep captioning with multimodal recurrent neural networks (mrnn). CoRR, abs/1412.6632, 2014.

M. Mostajabi, P. Yadollahpour, and G. Shakhnarovich. Feedforward semantic segmentation with zoom-out features. In 2015 IEEEConference on Computer Vision and Pattern Recognition (CVPR), pages 3376–3385, June 2015.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, pages III– 1310–III–1318. JMLR.org, 2013.