IMAGE CAPTIONING

PROJECT REPORT

Submitted by

G.JAI SURYA GOWD – 17MIS1122

ARTIFICIAL INTELLIGENCE (SWE 4010)

M Tech Integrated Software Engineering

Submitted to

Prof. Rajesh Kumar (C1 Slot)

FALL SEMESTER 2020-21

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

IMAGE CAPTIONING

TABLE OF CONTENTS

OBJECTIVE

In the past few years, the problem of generating descriptive sentences automatically for images has garnered a rising interest in natural language processing and computer vision research. Image captioning is a fundamental task which requires semantic understanding of images and the ability of generating description sentences with proper and correct structure. In this study, the authors propose a hybrid system employing the use of multilayer Convolutional Neural Network (CNN) to generate vocabulary describing the images and a Long Short Term Memory (LSTM) to accurately structure meaningful sentences using the generated keywords. The convolutional neural network compares the target image to a large dataset of training images, then generates an accurate description using the trained captions. We showcase the efficiency of our proposed model using the Flickr8K and Flickr30K datasets and show that their model gives superior results compared with the state-of-the-art models utilizing the Bleu metric. The Bleu metric is an algorithm for evaluating the performance of a machine translation system by grading the quality of text translated from one natural language to another. The performance of the proposed model is evaluated using standard evaluation matrices, which outperform previous benchmark models.

INTIAL DESCRIPTION

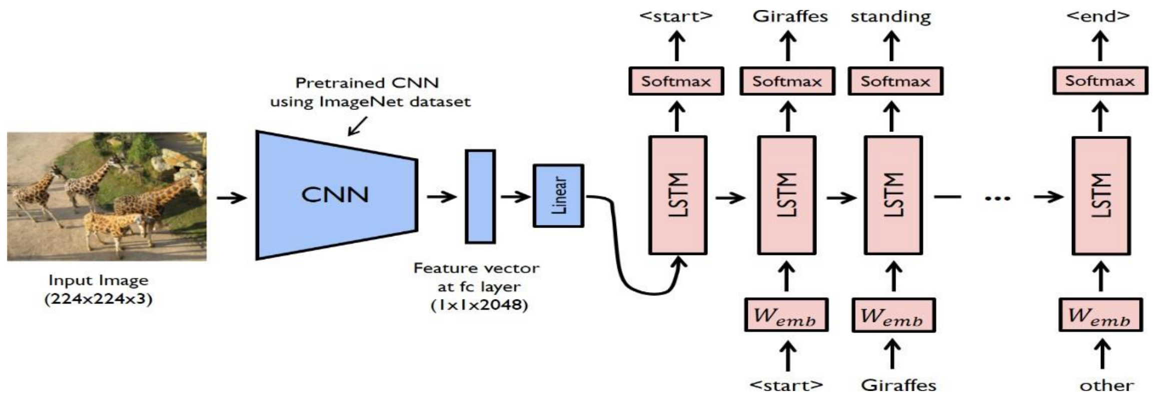
Caption generation is an interesting artificial intelligence problem where a descriptive sentence is generated for a given image. It involves the dual techniques from computer vision to understand the content of the image and a language model from the field of natural language processing to turn the understanding of the image into words in the right order. Image captioning has various applications such as recommendations in editing applications, usage in virtual assistants, for image indexing, for visually impaired persons, for social media, and several other natural language processing applications. Recently, deep learning methods have achieved state-of the-art results on examples of this problem.

It has been demonstrated that deep learning models are able to achieve optimum results in the field of caption generation problems. Instead of requiring complex data preparation or a pipeline of specifically designed models, a single end-to-end model can be defined to predict a caption, given a photo. In order to evaluate our model, we measure its performance on the Flickr8K dataset using the BLEU standard metric. These results show that our proposed model performs better than standard models regarding image captioning in performance evaluation.

The limitations of neural networks are determined mostly by the amount of memory available on the GPUs used to train the network as well as the duration of training time it is allowed. Our network takes around seven days to train on GTX 1050 4GB and GTX 760 GPUs. According to our results, our results can be improved by utilizing faster and larger GPUs and more exhaustive datasets.

DESIGN

Our model to caption images are built on multimodal recurrent and convolutional neural networks. A Convolutional Neural Network is used to extract the features from an image which is then along with the captions is fed into a Recurrent Neural Network. The architecture of the image captioning model



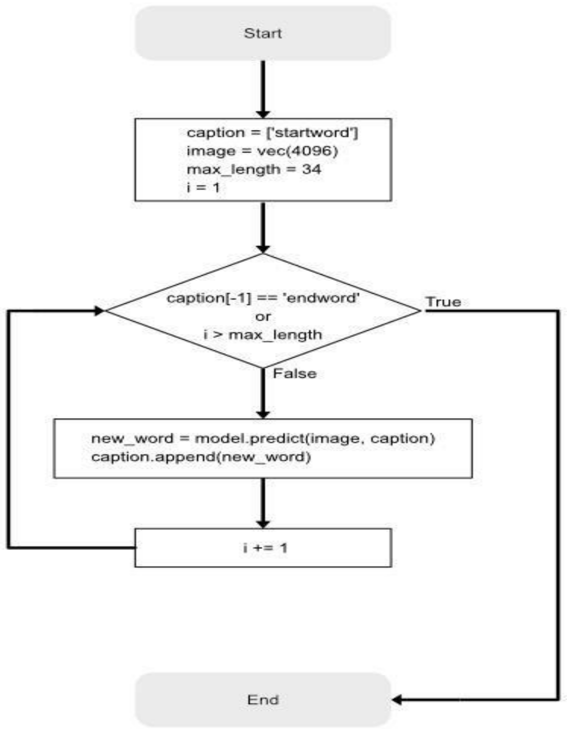
This architecture involves two main modules. The first one is image understanding module using CNN and the second one is text understanding module using LSTM. Each module is described in details in the following subsections.

Convolutional Neural Network (CNN): For image caption generation task, CNN is widely used because it has solved successfully for image annotation problems with high accuracy (Aditya et al., 2019). We have trained and tested two different models for feature extraction of images datasets. The two models have different capabilities in extracting features of images and the input image size of both models are 224× 224× 3 and the convolutional feature size of VGG is 4096.

VGG16: It is a pre-trained model on ImageNet dataset based on Visual Geometry Group (VGG) Oxford Net 16-layer CNN (Rahul and Aayush, 2018 ; Lakshminarasimhan et al., 2018). The VGG16 neural network is used for image classification. Output of VGG16 is probability of individual classes that the classification system has to classify. We remove the last layer of the VGG16 and use the output from second last layer as feature parameters for each image. We extract 4096 parameters for each image, which are further processed by a Dense layer to produce a 256 element representation of an image (Micah et al., 2013).

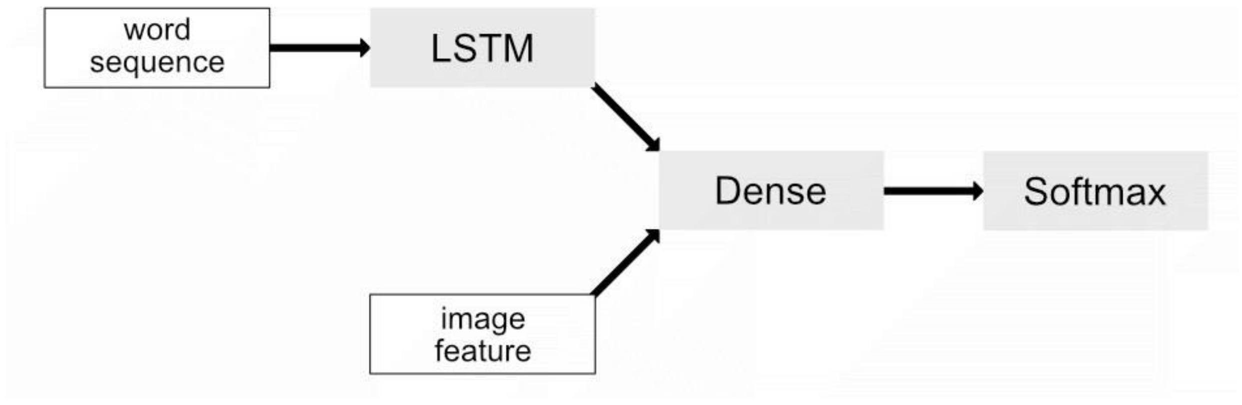
Long-Short Term Memory (LSTM): LSTM can maintain information in memory for long periods of time and retrieve sequential information through time (Yang et al., 2018). The text understanding part produces words or phrases based on the word embedding vector of previous part. The language generation model is trained to predict each word in the caption after it has seen both image and all previous words. For any given sentence in Myanmar corpus we add two extra symbols for start word and stop word which designates the start and end of the sentence. Whenever stop word is found it halts generating caption and it denotes end of the sentence. Sequence Processor is a word embedding layer to handle the input text and then followed by a Long Short-Term Memory (LSTM) recurrent neural network layer (Shiru et al., 2017). The proposed model is defined by the input sequences length (21 words) which are fed into an Embedding layer and then uses a mask to ignore padded values and followed by an LSTM layer with 256 memory units (Parth et al., 2017). Both input models produced a 256 element vector and used regularization of 50% dropout to reduce over fitting during the training. In decoding, the model combined the vectors from both input models by using an addition operation and then fed to a Dense 256 neuron layer to make a **softmax** prediction over the whole output vocabulary for the next word in the sentence. Loss function for both models are evaluated as, L **(I, S) = - ∑ log (□□) □ □=1 (1)** . Where I is input image and S is generated sentence, N is the length of generated caption. pt and St are probability and predict word at time t respectively. During the training process we have tried to reduce this loss function

let’s think about how the caption generation process should work in the end. As our goal is not to map an image to a specific caption but rather learn the relationship between image features and word sequences and between word sequences and single words our target is not a complete caption in the dataset but a single word. In other words, the model generates a new caption word by word based on a given image feature and a caption prefix. In the beginning, each caption only contains the artificial start sequence. We also need the image feature of the image the model should describe in the end. Both, the caption and the image feature vector will be passed to the model. The model will predict the word from the vocabulary which has the highest probability to follow the given caption prefix in combination with the image feature. The predicted word will be appended to the caption and will pass to the model again. As you noticed this is an iterative process, which will terminate in two ways. The first stopping criteria would be, that the model predicts the artificial end sequence as the next word. The second one is given by an upper bound of the caption length. We have to specify this bound in advance, which will also have an influence on the model’s structure later. Either way, the model will terminate and output the generated caption. We illustrated the process in the flow-chart below.



Model Structure

This may be one of the most important questions whenever developing a neural net. This question is not an easy one and can not easily be answered. A model will always be developed for a specific task, therefore no general answer can be given. However, there is some research about how to come up with a good model structure for a given task. In the field of image captioning Marc Tanti, et al. compared 16 different model architectures and determined the best one. In this tutorial we will follow their merge-add architecture, which is depicted in Figure below:



This model architecture takes tow vectors as inputs:

1. image feature vector
2. word sequence vector

As you can see, the vectors get injected into two different parts in the model. Marc Tantie, et al. came to the conclusion, that models, where the text data and the image information are handled exclusively perform better. Later in the model, both vectors get merged. Since models with an LSTM layer performed slightly better than models with RNN layer in their experiments we will go with this approach.

After the LSTM layer processed the word sequence (in the following caption prefix) its output will be merged with the image feature vector. This merge step can be realized in the ways:

1. *merge-concat*: Concatenate the image feature vector and the caption prefix to one vector. The resulting vector has the length of the sum of the length of the input vectors.
2. *merge-add*: Add elementwise image feature vector and caption prefix vector together. The resulting vector has the same dimensions as the input vectors.
3. *merge-mult*: Multiply elementwise the image feature vector and caption prefix together. The resulting vector has the same dimensions as the input vectors.

A clear downside of the concatenate approach is the increase of dimensions of the resulting vector, which results in a more complex layer and requires more computational power in the end. In the following, we will go with the *merge-add* approach.

The last layer of the model, a softmax layer, finally outputs a probability distribution for all words in the vocabulary. In order to get the next word in the caption prefix, we only need to choose the word with the highest probability.

IMPLEMENTATION DETAILS

The model consists of 3 phases:

Image Feature Extraction

The features of the images from the Flickr 8K dataset is extracted using the VGG 16 model due to the performance of the model in object identification. The VGG is a convolutional neural network which consists of consists of 16 layer which has a pattern of 2 convolution layers followed by 1 dropout layers until the fully connected layer at the end. The dropout layers are present to reduce overfitting the training dataset, as this model configuration learns very fast. These are processed by a Dense layer to produce a 4096 vector element representation of the photo and passed on to the LSTM layer.

Sequence processor

The function of a sequence processor is for handling the text input by acting as a word embedding layer. The embedded layer consists of rules to extract the required features of the text and consists of a mask to ignore padded values. The network is then connected to a LSTM for the final phase of the image captioning.

Decoder: The final phase of the model combines the input from the Image extractor phase and the sequence processor phase using an additional operation then fed to a 256 neuron layer and then to a final output Dense layer that produces a softmax prediction of the next word in the caption over the entire vocabulary which was formed from the text data that was processed in the sequence processor phase. The structure of the network to understand the flow of images and text.

For task of image captioning there are several annotated images dataset are available. Flickr 8K Image captioning dataset is used in the proposed model. Flickr 8K is a dataset consisting of 8,092 images from the Flickr.com website. This dataset contains collection of day-to-day activity with their related captions. First each object in image is labeled and after that description is added based on objects in an image. We split 8,000 images from this corpus into three disjoint sets. The training data (D Train) has 6000 images whereas the development and test dataset consist of 1000 images each. In order to evaluate the image-caption pairs, we need to evaluate their ability to associate previously unseen images and captions with each other.

Evaluation

In order to evaluate the performance of our model we test it by means of the BLEU score. The BLEU score was developed to automatically evaluate the quality of machine translations [5]. The closer a automatically translated candidate sentence is to a human translated reference sentence the better it is. The score is calculated by comparing the matching n-grams. As the closeness of a candidate translation to a reference translation is measured this metric can also be used to evaluate the generation of image captions.

RESULTS

Extract Features from the images using vgg16 model

By calling extract_feature() we can generate an image feature using the modified VGG16 model. To do so we have to pass the image to the model calling the load_img() function and make some image pre-processing which the VGG16 model requires. In the end, the model returns a 4,096-element vector.

The get_features() function combines both previously defined methods and offers a handy way to generate features either for multiple images, by passing a directory or for a single image. However, the function returns a dictionary which maps the image identifier and its feature.

Since it takes some time to generate all features for the dataset, it makes sense to save the dictionary for later use. You can use the save_features()

```
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model

def extract_features(directory):

  model = VGG16()
  model.layers.pop()
  model = Model(inputs=model.inputs,outputs=model.layers[-1].output)

  print(model.summary())

  features = dict()
  for name in listdir(directory):

    filename = directory + '/' + name
    image = load_img(filename, target_size=(224,224))
    image = img_to_array(image)
    image = image.reshape((1,image.shape[0], image.shape[1], image.shape[2
]))
    image = preprocess_input(image)
    feature = model.predict(image, verbose=0)
```

function from our helper package to do so. Just pass the dictionary and a path and the features will be saved as a .pkl file.

```
    img_id = name.split('.')[0]
    features[img_id] = feature
    print('>%s' % name)
  return features

directory = '/content/drive/My Drive/Colab Notebooks/Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
dump(features,open('/content/drive/My Drive/image_captioning_project/feature.pkl', 'wb'))
```

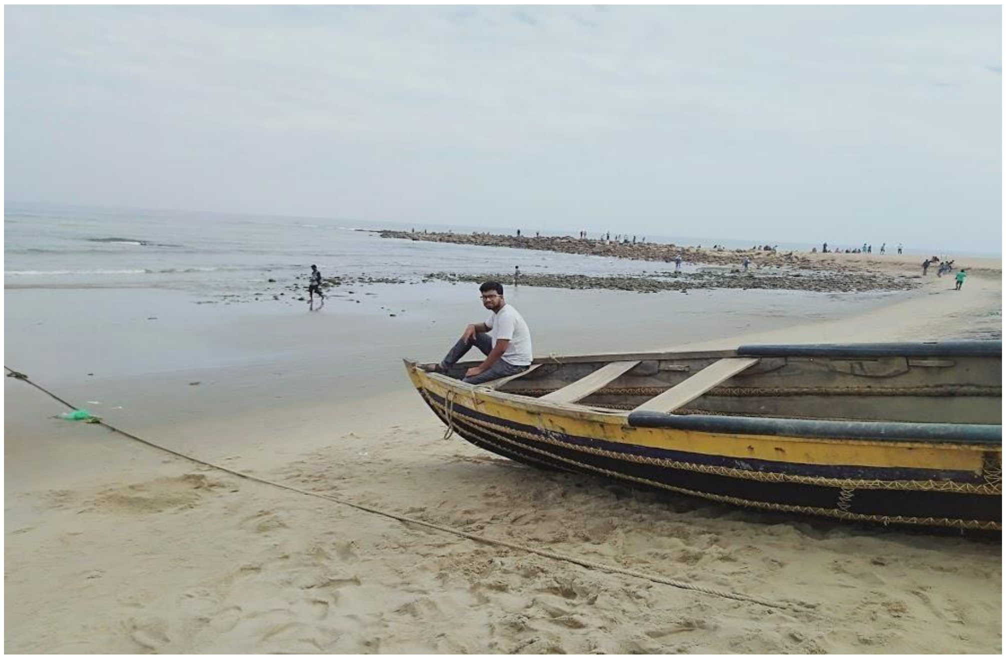
BLEU Scores for the selected model :

```
def load_doc(filename):

    file = open(filename, 'r')

    text = file.read()

    file.close()
    return text


def load_set(filename):
    doc = load_doc(filename)
    dataset = list()

    for line in doc.split('\n'):

        if len(line) < 1:
            continue

        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)


def load_clean_descriptions(filename, dataset):

    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):

        tokens = line.split()

        image_id, image_desc = tokens[0], tokens[1:]

        if image_id in dataset:

            if image_id not in descriptions:
                descriptions[image_id] = list()

            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'

            descriptions[image_id].append(desc)
    return descriptions


def load_photo_features(filename, dataset):

    all_features = load(open(filename, 'rb'))
    features = {k: all_features[k] for k in dataset}
    return features


def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc


def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer


def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)


def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

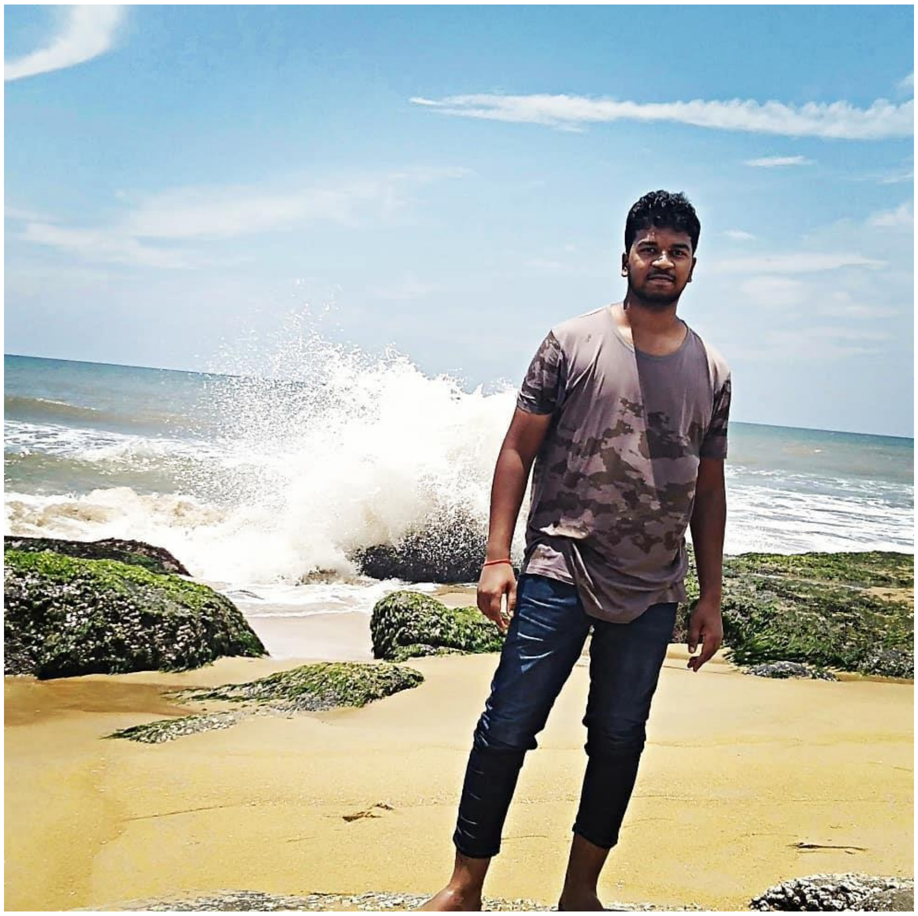
```
def generate_desc(model, tokenizer, photo, max_length):

    in_text = 'startseq'

    for i in range(max_length):

        sequence = tokenizer.texts_to_sequences([in_text])[0]

        sequence = pad_sequences([sequence], maxlen=max_length)

        yhat = model.predict([photo,sequence], verbose=0)

        yhat = argmax(yhat)

        word = word_for_id(yhat, tokenizer)

        if word is None:
            break

        in_text += ' ' + word

        if word == 'endseq':
            break
    return in_text


def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()

    for key, desc_list in descriptions.items():
        yhat = generate_desc(model, tokenizer, photos[key], max_length)

        references = [d.split() for d in desc_list]
        actual.append(references)
        predicted.append(yhat.split())

    print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
    print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
    print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

filename = 'model_4.h5'
model = load_model(filename)

evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

```
BLEU-1: 0.535445
BLEU-2: 0.288257
BLEU-3: 0.195596
BLEU-4: 0.089424
```

IMAGES AND CAPTIONS



startseq man in red shirt is riding bike on the side of the water endseq



```
startseq man in red shirt is riding bike on the street endseq
```

CONCLUSION

We combine "Image Labelling" 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

REFERENCES

[1] Michael Grubinger, Paul Clough, Henning Müller, and Thomas Deselaers. The IAPR TC-12 Benchmark – a New Evaluation Resource for Visual Information Systems. 2006.

[2] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. Journal of Artificial Intelligence Research, 47:853– 899, 2013

[3] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[4] Marc Tanti, Albert Gatt, and Kenneth P. Camilleri. Where to put the Image in an Image Caption Generator. arXiv preprint arXiv:1703.09137, 2017.

[5] Kishore Papineni, Salim Roukos,Todd Ward and Wei-Jing Zhu. BLEU: A method for automatic evaluation of machine translation. In Proceedings of the 40th Annual Meeting on Association for Computational Linguistics, pages 311–318. Association for Computational Linguistics, 2002.