# Image Captioning with Encoder-Decoder Implementation and Comparison to Other Methods

by

Johannah Cushing

Dr. Piotre Sczcurek, faculty advisor

Masters project submitted in partial fulfillment of the

requirements for the Master of Science in Data Science degree

in the College of Arts and Sciences of

Lewis University

**Abstract** ***Automatic image captioning is a rapidly developing field which combines advances in computer vision processing and natural language processing. Most image captioning systems today are capable of producing a caption for a novel "unseen" image. There are several models types which have been developed in recent years which have shown to be successful. Here a common type of model is implemented, an encoder-decoder, and trained on two common and freely available datasets: the Flickr8k and Flickr30k datasets. A common metric called the BLEU score is used to evaluate and compare the results to other models; it is generally reported as a set of four scores. The BLEU scores on the Flickr8k are 0.552, 0.365, 0.241, and 0.130, while the scores on the Flickr30K dataset are 0.575, 0.381, 0.262, and 0.150. The model demonstrates the capabilities of a relatively simple system and demonstrates the need for modifications for improved/more accurate results. The particular requirements of the captioner as well as time and system constraints should dictate modifications made for improvements.***

**Index Terms— image captioning, Flickr dataset, neural networks, encoder-decoder, convolutional neural networks, long-short-term memory neural networks, recurrent neural networks**

## I. INTRODUCTION

Automatic image captioning is the task of producing a natural language description (sentence) to accurately reflect the visual content of an image. This field has seen wide growth in recent years with advances in deep learning architectures, computer vision, and natural language processing. Practical applications include image indexing or retrieval. There are applications for context-based image retrieval in many areas such as biomedicine, commerce, education, and web searching. Aiding the visually-impaired by transforming visual signals into information which can be communicated via text-to-speech technology is another important application of image captioning**.**

The development of new annotated datasets as well as new models themselves add to the development of the field. Datasets vary in terms of number of images, size of images, number of captions per image, and how broad a scope the dataset covers.

Modern image captioning generally uses a deep learning framework to produce novel captions. Current research has focused both on improving language models and modifications to various types of architectures, such as the popular CNN-RNN (convolutional neural network – recurrent neural network) or CNN-LSTM (convolutional neural network – long short-term memory) architecture. However, newer architectures are also being used such as GAN (generative adversarial networks). Dense captioning, stylized captions, and especially attention-based mechanisms have improved captioning results on several metrics. Attention based methods focus on a certain part of the *important* concepts, i.e. objects or actions which are integral to constructing an accurate image caption.

The goal of this research is to create an encoder-decoder model, test the model on the Flickr8K and Flickr30K datasets and compare results to current modern methods. This will show the capabilities of a common type of model and help assess how much advancement has been made in the field in the past few years.

## II. LITERATURE SURVEY

### *A. Datasets*

**Flickr8k and Flickr30k datasets** [1]- The Flickr datasets are a collection of images from the Flickr website. The Flickr8k dataset has 8,000 images, divided into train/test/development sets, with five captions for each image annotated by humans. The

30K has 30,000 images and the set does not provide any fixed split. There are detectors for common objects, a color classifier, and a bias towards selecting larger objects.

**MS COCO dataset** [2]- The MS-COCO dataset was designed to advance object recognition by placing the question of object recognition in the context of the broader question of scene understanding. Images of complex everyday scenes containing common objects in their natural context are used. Objects are labeled using per-instance segmentations to aid in precise object localization. The dataset contains photos of 91 object types that would be easily recognizable by a 4-year-old. There is a total of 2.5 million labeled instances in 328,000 images.

**ImageNet dataset [3]-** The ImageNet dataset was built upon the backbone of the WordNet structure. ImageNet aims to populate the majority of the 80,000 synsets (synonyms) of WordNet (a lexical database) with an average of 500-1000 clean and full resolution images. There are 3.2 images in total.

**Visual Genome dataset** [4]- Visual Genome is a dataset and knowledge base created in an effort to connect structured image concepts to language. The database features 108,077 captioned images, where each image has an average of 35 objects, 26 attributes, and 21 pairwise relationships between objects. The objects, attributes, relationships, and noun phrases in region descriptions and questions-answer pairs are canonicalized to WordNet synsets. The dataset was designed to better understand the interactions and relationships between objects in an image.

**Conceptual Captions** [5]- Conceptual Captions is a new dataset consisting of approximately 3.3 million image-caption pairs that are created by automatically extracting and filtering image-caption annotations from billions of web pages. The dataset contains an order of magnitude more images than MS COCO, with a wider variety of images and image captioning styles.

The Instagram Dataset, IAPR TC-12 Dataset, Stock3M, MIT-Adobe FiveK, and FlickrStyle 10K, are also used for image captioning.

### *B. Image Captioning Models*

There are many different methods used for image captioning. Older methods include Local Binary Patterns, Scale-Invariant Feature Transform, and Histogram of Oriented Gradients where features are extracted from input data and then passed to a classifier such as a support vector machine [6]. However, research of the past several years has focused largely on deep machine learning techniques.

Some deep learning methods of image captioning are template-based. These can generate grammatically correct captions, but templates are predefined and captions are fixed in length [6]. Retrieval based methods find the visually similar images with their captions from the training data set. The captions for the query image are selected from the caption pool. Although retrieval based methods produce syntactically correct captions, they are general, not image specific, and cannot create semantically correct captions [6]. Novel captions methods use deep learning techniques and are semantically more accurate than previous approaches. This is the focus of most modern research.

Most captioning methods use visual space for generating captions. However, some use a multimodal space. A typical multimodal space based method contains a language encoder part, a vision part, a multimodal space part, and a language decoder part [6]. In these methods, deep neural networks and a multimodal neural language model are used to learn both image and text *jointly* in multimodal space. Then, the generator part generates captions. The vision part typically uses a CNN as a feature extractor. The language encoder extracts word features and learns dense feature embedding for each word, forwarding

the semantic temporal context to the recurrent layers [6]. The multimodal space maps image features and word features into a common space. The map is then fed to the language decoder and captions are generated.

A popular multimodal model was developed by Mao et al. [7]. They use a multimodal Recurrent Neural Networks (m-RNN) model. The model contains a language model part, a vision part and a multimodal part. The language model learns a dense feature embedding for each word in the dictionary and stores the semantic temporal context in recurrent layers. The vision part contains a deep CNN which generates the image representation. The multimodal part connects the language model and the deep CNN by a one-layer representation. The model is learned using a log-likelihood cost function. The temporal context being stored in a recurrent architecture allows for arbitrary context length. This model has a two-layer word embedding system in the m-RNN network structure which learns the word representation more efficiently than a single-layer word embedding. Another improvement over past models is that the recurrent layer is not used to store the visual information. The image representation is inputted to the m-RNN model along with every word in the sentence description. The capacity of the RNN is used more efficiently and thus a relatively small dimensional layer can be used. It utilizes the capacity of the recurrent layer more efficiently. A comparison of the m-RNN model to the standard RNN is given in Fig. 1.

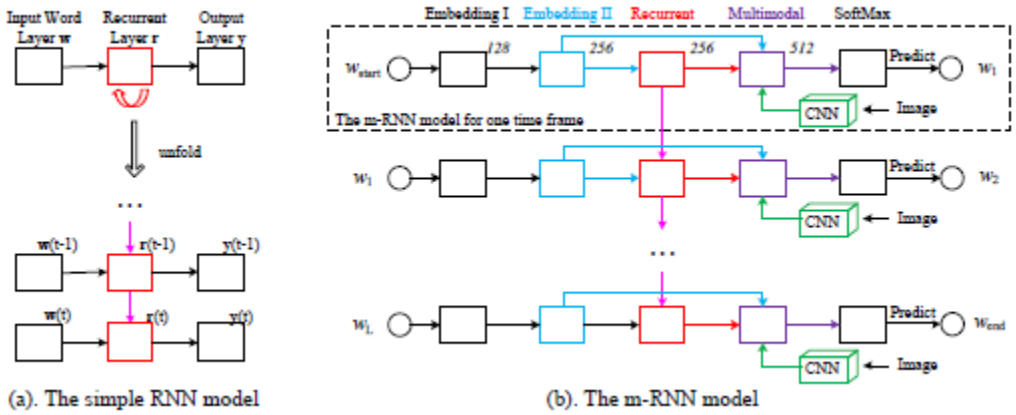


**Fig. 1. RNN and mRRN models [7]. a) A simple RNN model as compared to b) a m-RNN model used by Mao et al. The inputs of the model are an image and its corresponding sentence descriptions. w1, w2, ..., wL represents the words in a sentence. a start sign wstart and an end sign wend are added to all the training sentences. The model estimates the probability distribution of the next word given previous words and the image. It consists of five layers (i.e. two word embedding layers, a recurrent layer, a multimodal layer, a softmax layer) and a deep CNN in each time frame. The number above each layer indicates the dimension of the layer. The weights are shared among all the time frames.**

While most models use the whole scene, dense captioning methods generate captions for *each region* of the scene. DenseCap, proposed by Johnson et al. [8] uses a CNN, a dense localization layer, and an LSTM language model, as seen in Fig 2. It uses a differential, spatial soft attention mechanism and bilinear interpolation to help backpropagate through the network and smoothly select active regions. A problem with dense captioning is that there may be problems recognizing overlapping regions of interest for a particular object.

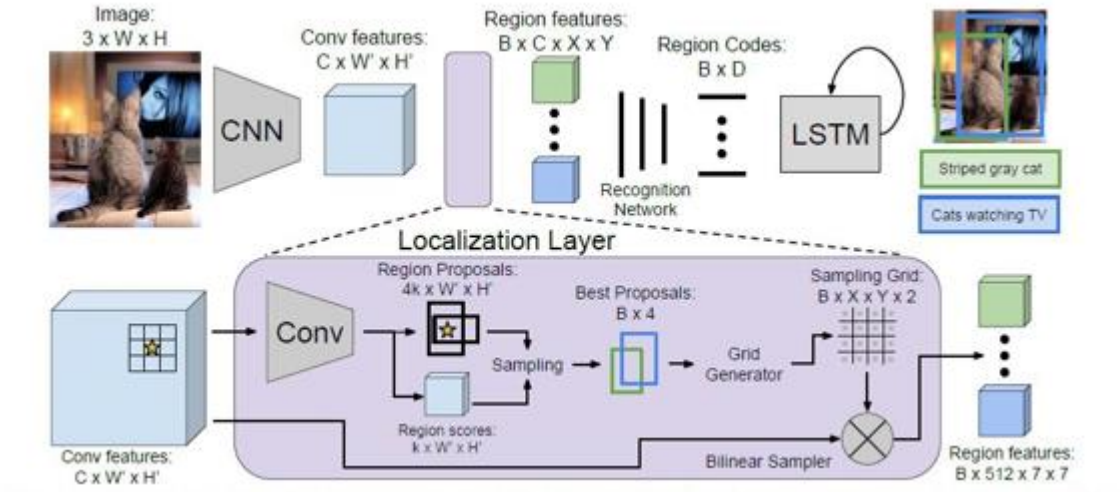


**Fig. 2. DenseCap model [8].**

Captioning models can also be divided into supervised versus other deep learning models. Supervised models have been used in tasks such as image classification, object detection and attribute learning. There are also many supervised learning-based image captioning methods- many of the ones which are commonly heard of—encoder-decoder, compositional architecture, attention-based, semantic concept based, stylized captions, novel object based, and dense image captioning [6]. Other deep learning models include reinforcement learning approaches and GANs.

A common framework is an encoder-decoder architecture. A typical method uses a CNN to obtain the scene type and detect the objects. The output from the CNN is then used by a language model, such as an LSTM to produce captions. Image information is included to the initial state of an LSTM. The next words are generated based on the current time step and the previous hidden state [6], continuing until the end token of a sentence. Recurrent Neural Networks (RNNs) are also commonly used as decoders to convert this representation word-by-word into natural language description of the image. These methods are unable to analyze the image over time while generating the image, and do not consider the spatial aspects of the image that is relevant to the parts of the image captions. [6].

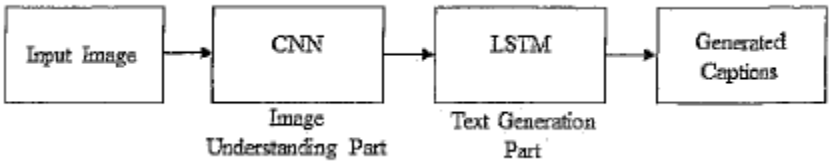


**Fig. 3. Generic encoder-decoder architecture [6].**

LSTM may also face challenges generating longer sentences, since the role of words generated at the beginning becomes weaker and weaker. Jia et al. [9] proposed a model called gLSTM (guided LSTM) which is an extension of the LSTM model. In particular, it adds semantic information extracted from the image as extra input to each unit of the LSTM block, with the aim of guiding the model towards solutions that are more tightly coupled to the image content. Unidirectional LSTM cannot generate contextually well-formed captions [6]. Wang et al. proposed a deep bidirectional LSTM based method for image captioning which can produce contextually and semantically richer captions [10]. The architecture consist of a CNN and two

separate LSTM networks which can utilize both past and future context information to learn long term visual language interactions.

In contrast to the encoder-decoder, there are compositional architecture-based methods [6]. These are composed of several independent building blocks. Here, a CNN is used to obtain image features. Then visual concepts or attributes are obtained from visual features. Multiple captions are generated by a language model. To generate a final caption, these candidate captions are re-ranked using a multimodal similarity model. The overall architecture is illustrated in Fig. 4. CNN-based approaches use the top layer of CNN for extracting information of the salient objects from the image. These techniques may lose information which is useful to generate more detailed captions. This can be overcome by using features from the lower convolutional layer instead of fully connected layer [6].

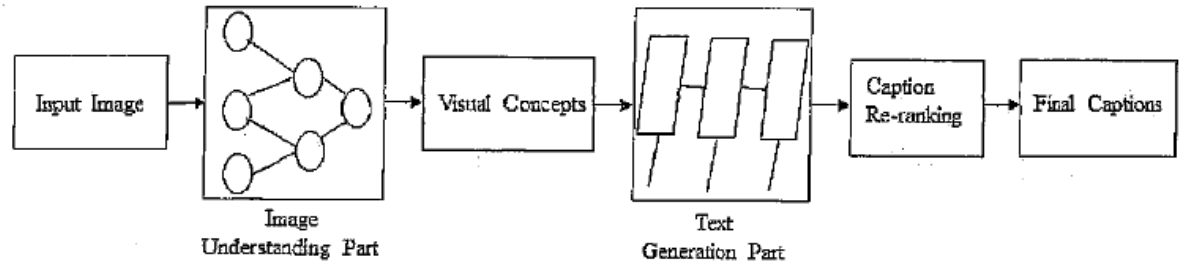


**Fig. 4. Generic Compositional architecture-based model [6].**

**A**ttention based mechanisms are becoming popular in deep learning image captioning. They focus on the various parts of the input image while the output sequences are being produced. A typical method follows these steps [6]: 1) Image information is obtained based on the whole scene with a CNN. 2) The language generation phase generates words or phrases based on the output from 1**.** 3) Salient regions of the image are focused in each time step of the language generation model based on generated words or phrases**.** 4) Captions are generated dynamically until the end state of the language generated model is reached. The main difference between attention methods from others is that they can concentrate on the salient parts of the image and generate corresponding words at the same time. This uses stochastic hard attention and deterministic soft attention to generate attentions [6].

Anderson implemented bottom up-attention using Faster R-CNN [11]. Faster R-CNN (region-CNN) detects objects in two stages. Faster R-CNN is used in conjunction with ResNet-101 CNN to generate an output set of image features for use in image captioning. Faster R-CNN detects objects in two stages. First, a Region Proposal Network (RPN) predicts object proposals. In the second stage, Region of Interest Pooling (RoI) is used to extract a small feature map for each box proposal. The feature maps are then boxed together as input for the final layers of the CNN.

Semantic concept-based methods have become more popular. These methods selectively attend to a set of semantic concept proposals extracted from the image. The concepts are combined into hidden states and the outputs of recurrent neural networks. These methods follow these steps [1]: A CNN-based encoder is used to encode the image features and semantic concepts. The image features are fed into the input of language generation model. Semantic concepts are added to the hidden states of the language model. Then, the language generation part produces captions with semantic concepts.

You et al [12] proposed an image captioning model with semantic attention. It is novel in that it combined bottom up and top down approaches. Top-down approaches start from the gist of an image and convert into words, and bottom-up come up with words describing various aspects of an image and then combine them. This model learns to selectively attend to semantic concept proposals and fuses the7m into hidden states and outputs of RNNs. This forms a feedback connecting the top-down and bottom-up computations. Bottom-up and Ttp-down approaches are also being used in other captioning models and for other tasks such as visual question answering [11]. Top-down visual attention mechanisms enable deeper image understanding through fine-grained analysis. These are what most conventional visual attention mechanisms are [11]. These methods are typically trained to selectively attend to the output of one or more layers of a CNN. This corresponds to a uniform grid of neural receptive fields regardless of the content of the image.

General Adversarial Networks (GANs) is a method previously used in other image tasks. These are unsupervised learning methods. They aim to improve the naturalness and diversity of captions. The first to use CGANs (conditional GANs) to generate image descriptions were Dai et al. [13]. The generator is G and there is an evaluator E. Given an image I, the generator G takes two inputs: an image feature f(I) derived from a CNN and a random vector z. The random vector allows the generator to produce different descriptions given an image. The diversity of the cations can be controlled by altering the variance of z. The generated uses an LSTM net as a decoder to generate a sentence word by word. The evaluator E is another neural network. It embeds the image I and words of the sentence into vectors of the same dimensions, the image with a CNN and the sentence with an LSTM net. The quality of the description is measured by the dot product of the embedded vectors. The model used by Dai et al.[13] is illustrated in Fig. 5.

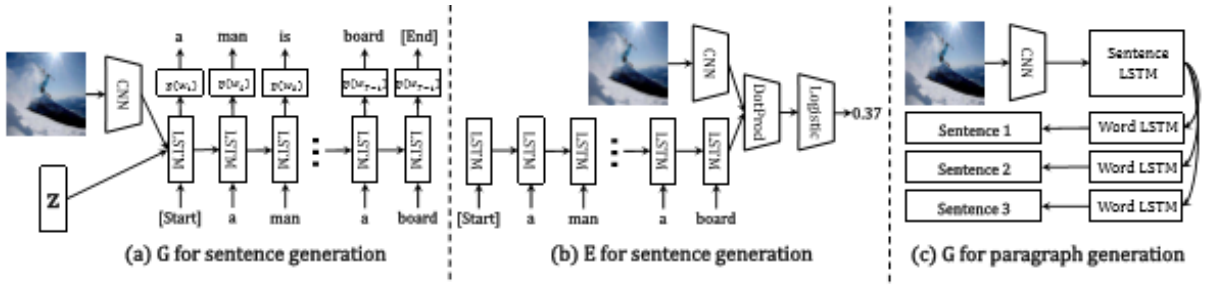


**Fig. 5. The structures of the generator G for both single sentences and paragraphs, and the evaluator E for single sentences. [13]**

Wang et al [14] proposed a parallel-fusion RNN-LSTM architecture for image caption generation. It combines the advantages of a simple RNN and LSTM. The proposed approach divides the hidden units of RNN into several same-size parts, and lets them work in parallel. Then, the outputs are merged with corresponding ratios to generate the final results. Moreover, these units can be different types of units, for instance, a simple RNN and a LSTM.

There are models which stand out from others in that they can generate captions on unseen objects. These novel object caption generation methods can generation descriptions of objects which are not present in paired image-caption datasets. Generally, these methods follow these steps [6]: A separate lexical classifier and a language model are trained on unpaired image data and unpaired text data. Then, a deep caption model is trained on paired image caption data. Both models are combined together to train jointly in order to generate captions for novel objects. Hendricks et al. [15] proposed one of the first models in this class. It explicitly transfers the knowledge of semantically related objects to compose the descriptions about novel objects in the proposed Deep Compositional Captioner (DCC).

Yao et al. [16] proposed a new architecture which uses a Long Short-Term Memory with Copying Mechanism (LSTM-C). It incorporates copying into the CNN plus RNN image captioning framework, for describing novel objects in captions (see Fig. 6). Specifically, freely available object recognition datasets are leveraged to develop classifiers for novel objects.

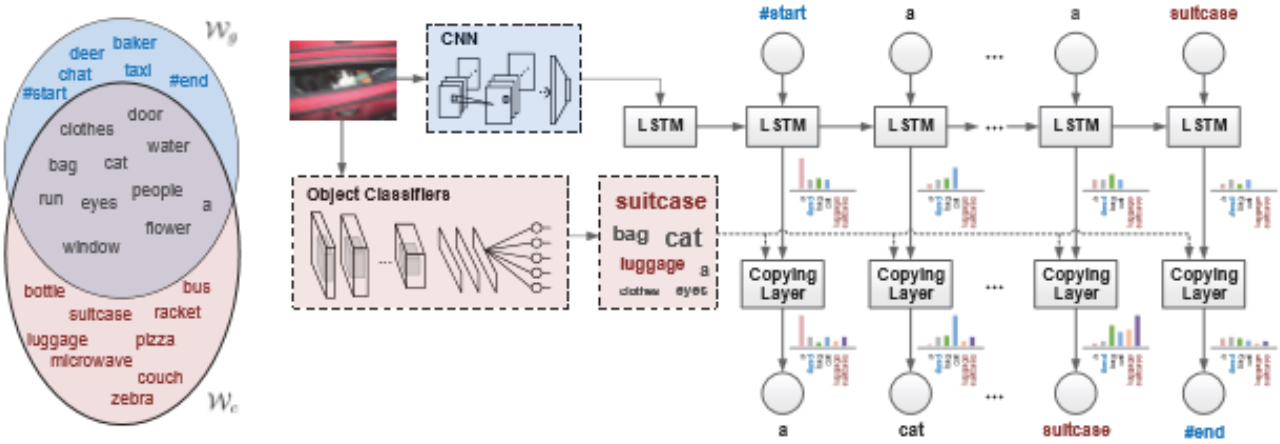


**Fig 6. LSTM-C model. Wg and Wc are the vocabularies on paired image-sentence dataset and unpaired object recognition dataset, respectively. The image representation extracted by CNN is injected into LSTM at the initial time for standard word-by-word sentence generation. The object classifiers learnt on unpaired object recognition dataset are used to detect the object candidates which are incorporated into LSTM, enabling captioning of novel objects. The copying mechanism and end-to-end trainable architecture leverages the standard word-to-word sentence generation mechanism as well as the copying mechanism.**

One area of research with image captioning involves making captions more engaging. This is the area referred to as "stylized caption generation". A model developed by Shuster et al [17] aims to define a new task "Personality-Captions" where the goal is to be as engaging as possible to humans by incorporating controllable style and personality traits. The researchers collected a dataset of over 240 thousand captions conditioned over 215 possible traits. They developed an overall TransResNet model (ResNeXt-IG-3.5B), shown below (Fig. 7). The researchers designed different models—generative and retrieval. The generative model gives a new state of the art on MS-COCO caption generation, and the retrieval architecture, TransResNet, yields the highest known R@1 score in the Flickr30k dataset. Another method that has worked to make captions more engaging included using puns [18].

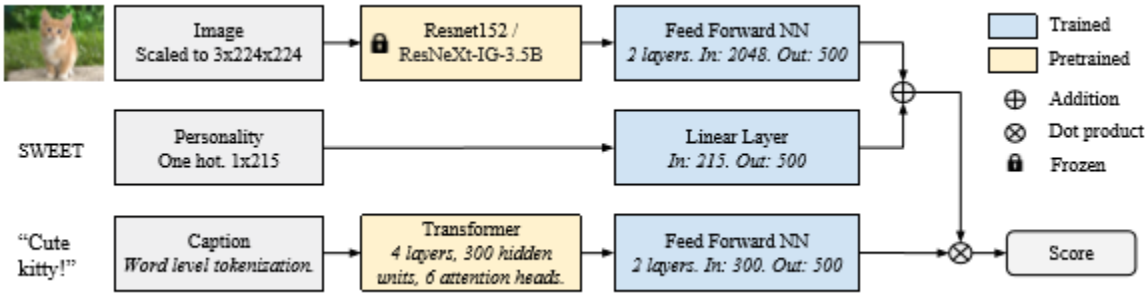


**Fig. 7. TransResNet architecture, used for our retrieval models [17].**

## *C. Evaluation Metrics*

**BLEU-** The BLEU (Bilingual evaluation understudy) Score, originally described in 2002, is a common method for automatic evaluation of machine translation [19]. Individual text segments are compared with a set of reference tests which are human generated and scores are computed; the overall score is the average of the scores computed. The score should be between 0 and 1, one being a perfect match. BLEU is a popular measure but is better used for short texts [20]. BLEU scores are generally reported as a set of BLEU-1, BLEU-2, BLEU-3, and BLEU-4. BLEU computes the geometric mean of n-grams precision with a positive weighting.

**ROUGE-** (Recall-Oriented Understudy for Gisting Evaluation) [21]- This metric measures the quality of a text comparing word sequences, word pairs, and n-grams with a set of reference summaries created by humans. ROUGE-2, ROUGE-L, ROUGE-W, and ROUGE-S worked well in single document summarization tasks, while ROUGE-1, ROUGE-L, ROUGE-W, ROUGE-SU4, and ROUGE-SU9 work better for evaluating very short summaries. Multi-document evaluation is not well evaluated with ROUGE.

**METEOR** [22]- METEOR was designed to address weaknesses identified with BLEU. Namely, lack of recall, using higher order n-grams as an indirect measure of how well a translation is grammatically formed, lack of explicit word-matching between translation and reference, and the use of geometric averaging of N-grams. It evaluates a translation by computing a score based on explicit word-to-word matches between the translation and a reference translation. If more than one reference translation is available, the given translation is scored against each reference independently, and the best score is reported.

**SPICE** (Semantic Propositional Image Captioning Evaluation) [23] - is a newer evaluation metric, developed in 2016. It is based on a graph-based semantic representation called scene-graph. The newly proposed SPICE correlates well with human judgments, but fails to capture the syntactic structure of a sentence [23].

**CIDEr (Consensus Based Image Caption Evaluation)** [24]- This method introduces a novel consensus-based evaluation protocol, which measures the similarity of a sentence to the majority, or consensus of how most people describe an image.

**Metric developed by Cui et al. [25]**- Cui et al proposed a novel learning-based discriminative evaluation metric directly trained to distinguish between human and machine-generated captions. It correlated with human judgements while also capturing the syntactic structure of a sentence.

## III. METHODS

An encoder-decoder model is developed to generate novel captions from images. It is developed in python 3.6 and uses keras 2.2.

The overall model has three parts: a photo feature extractor, a sequence processor, and decoder. Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a dense layer to make a final prediction.

The first part of the model is an image encoder (feature extractor) and for this a VGG16 model trained on the ImageNet dataset is used (Fig. 9). The model is a type of CNN and was originally described by K. Simonyan and A Zisserman in 2014 [26]. The input to Convolution1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional layers, where the filters are used with a small receptive field of size 3×3, which is the smallest size capable of capturing the notion of left/right, up/down and center. The convolution stride is fixed to 1 pixel so that the spatial resolution is preserved after convolution. Spatial pooling is carried out by five max-pooling layers. Max-pooling is performed over a 2×2

pixel window, with stride 2. Three Fully-Connected (FC) layers follow a stack of convolutional layers. The first two have 4096 channels each, the third performs classification and contains 1000 channels (one for each class in ImageNet classification). From the original VGG16 model this last FC layer for classification will is removed for this model. The final layer is the soft-max layer. The hidden layers are equipped with the rectification (ReLU) non-linearity.

The VGG model has some advances over previous methods [26]. First, it uses ReLU which cuts training time, and the increased number of RELU units makes the decision function more discriminative. For example, as compared to a previous model AlexNet, the 2012 ImageNet Competition winner, there are fewer parameters per channel [26]. VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields.

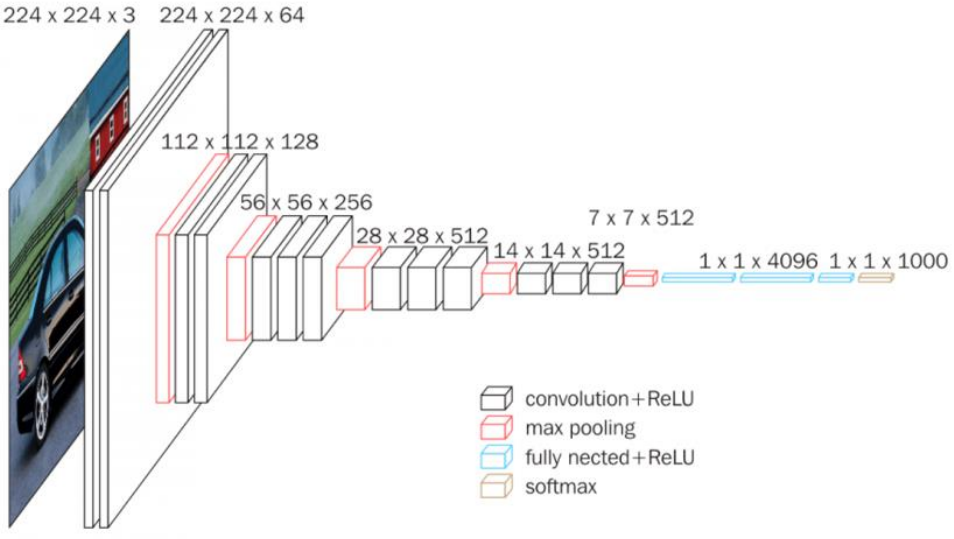


**Fig. 9 [27] VGG16 architecture**


**Fig. 10 [27] Illustration of ConvNet configuration with pooling**

A preprocess function is used to convert the images to the format required. Features are stored in a dictionary. The extracted features from the photo feature extractor are used as input.

To prepare the text data, the descriptions are loaded into memory and a mapping of photos to descriptions is created. Descriptions are human generated captions read from a sample dataset, along with an image identifier tag, associating it with a specific image. Descriptions are cleaned by converting to all lower case and removing one letter words. The loaded descriptions are converted into a vocabulary of words. Extraneous words which add little meaning will unnecessarily add to the training time of the model. Loaded descriptions are converted to a vocabulary of words and a list of description strings is made.

While training, the performance of the model on the development dataset is monitored. This is used to determine which model (after which epoch) to save.

The caption will be generated one word at a time. The sequence of previously generated words will be provided as input. a '*first word*' is thus needed to start the generation process and a '*last word*' to signal the end of the caption. The strings '*startseq*' and '*ends*' are used for this purpose.

The description text is encoded to numbers. The first step in encoding the data is to create a consistent mapping from words to unique integer values. The dictionary of descriptions is converted into a list of strings and another function will fit a tokenizer given the loaded photo description text.

For training, descriptions are split into words. The model is fed a word and the photo to generate the next word. Then the first two words of the description will be provided to the model as input with the image to generate the next word. There are two input arrays to the model: one for photo features and one for the encoded text. There is one output for the model which is the encoded next word in the text sequence.

The model will output a prediction, which will be a probability distribution over all words in the vocabulary. The output data will be a one-hot encoded version of each word, representing an idealized probability distribution with 0 values at all word positions except the actual word position, which has a value of 1.

The feature extractor and sequence processor both output a fixed-length vector. These are merged together and processed by a dense layer to make a final prediction. The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a dense layer to produce a 256 element representation of the photo.

The Sequence Processor model expects input sequences with a pre-defined length (no more than the maximum length computed) which are fed into an embedding layer which ignores padded values. This is followed by an LSTM layer with 256 memory units.

Both the input models produce a 256 element vector. Further, both input models use regularization with 50% dropout. This is to reduce overfitting the training dataset. The decoder model merges the vectors from both input models using an addition operation. This is then fed to a dense 256 neuron layer, followed by an output dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.

When the results of the model on the development dataset improves at the end of an epoch, the model is saved. The saved model with the best skill on the training dataset will be used as the final model.

This model is evaluated on both the Flickr8K and Flickr30K datasets. The Flickr8K was pre-split. The Flickr 30K was split 75% train, 12.5% development, 12.5% test, to match the proportions used in Flickr8K. BLEU scores are computed on the test datasets as a means of evaluating the captioning model. New images which are "unseen" to the captioner are also tested to examine any possible patterns; these are selected images from the Flickr website and stock images available from the internet.

## IV. RESULTS AND DISCUSSION

Figure 12. Example of an image captioned with this model. Caption generated when trained with Flickr8k: *Black dog running through the water* Caption when trained with Flickr30K: *A dog is running through the water*

Figure 13. Example of an image captioned with this model. Caption generated when trained with Flickr8K: *Man in red shirt is riding bike* Caption generated when trained with Flickr30K: *Boy is riding bike on dirt path*

**Fig. 14. Example of an image captioned with this model. Caption generated when trained with Flickr8K: Woman pours a glass of wine. Caption generated when trained with Flickr30K: Woman pours a glass of red wine**

**Fig. 15. Example of an imaged captioned with this model. Caption generated when trained with Flickr8K: Boys in blue shirts are playing soccer Caption generated when trained with Flickr30K: Boys in blue shirts run after soccer ball**

| | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 |
|---|---|---|---|---|
| **Flickr8K** | 0.552 | 0.365 | 0.241 | 0.130 |
| **Flickr30K** | 0.575 | 0.381 | 0.262 | 0.150 |

**Table 1. BLEU scores for the model when trained on the Flickr8K and Flickr30K datasets**

| | BLEU-1 | BLEU-2 | BLEU-3 | BLEU-4 |
|---|---|---|---|---|
| **Dai et al - gGAN generator [13]** | --- | --- | 0.153 | 0.088 |
| **Chen & Vitnick – MSR/CMU [29]** | --- | --- | --- | 0.126 |
| **Cushing (this model)** | 0.552 | 0.365 | 0.241 | 0.150 |
| **Donahue et al – LRCN [30]** | 0.587 | 0.390 | 0.250 | 0.165 |

| | | | | |
|---|---|---|---|---|
| **Vinyals et al – CNN-LSTM [31]** | 0.663 | 0.423 | 0.277 | 0.183 |
| **Mao et al –m-RNN [6]** | 0.600 | 0.410 | 0.280 | 0.190 |
| **Xu et al – hard attention [33]** | 0.667 | 0.434 | 0.288 | 0.191 |
| **Jia et al –gLSTM [9]** | 0.646 | 0.466 | 0.305 | 0.206 |
| **Chen et al – SCN-CNN-VGG [32]** | 0.646 | 0.453 | 0.317 | 0.218 |
| **Chen et al – SCA-CNN-ResNet [32]** | 0.662 | 0.468 | 0.325 | 0.223 |
| **You et al – ATTN-FCN [12]** | 0.647 | 0.460 | 0.324 | 0.230 |
| **Dai et al – gMLE generator [13]** | --- | --- | 0.372 | 0.305 |

**Table 2. Comparison of BLEU scores for various models. Values are in order of increasing BLEU-4 scores.**

As shown in Fig. 1-4, captions obtained for a given image are different when the model is trained on different datasets. The captions obtained when trained on the Flickr30K dataset are perhaps a little more detailed or in some cases accurate. For instance, in Fig. 12, Flickr8K training yields a caption with "black dog" when the dog appears brown or spotted. When trained with Flickr30K the model leaves out a descriptive adjective for the dog entirely. In Fig. 13, both captions are faulty. With the 8K dataset, the child in the photo is described as a "man in red shirt". With the Flickr30K training, the child riding a bike is identified correctly, which is the main element of the photo, but the street or path, which appears to be pavement, is identified as dirt. In Fig. 14, both captions are accurate, with the caption created from Flickr30K training being slightly more detailed, adding in an extra adjective, "red". Fig. 15 was well-captioned as well, neither caption here seems obviously better than the other. Overall, the captions tend to be short in length. It is common for LSTMs to have challenges in generating long length sentences [7]. Here an encoder-decoder, which is a common type of model, is implemented. It is questionable whether these types of models can deal with the two types of modalities—the structure of the visual information is very different from the structure of the description to be generated [9]. Jia et al. note that CNN-LSTM encoder-decoders sometimes generate sentences that seemed to "drift away" or "lose track" of the original image content generated a description that is common in the dataset but only weakly coupled to the input image [9]. This may be a problem with this model implemented here.

BLEU scores can range from 0 to 1, higher numbers being better, or indicating a more similar match to reference translations. Adding more training data increases the scores here, as can be seen in the better scores on the Flickr30k set compared to the Flickr8k set. Better scores would be expected on another dataset such as MS COCO. There are images on which the model performs very well, i.e. giving "correct" images. However, there are images where the model does not perform well. It should be noted, however, that this is true for this model and these two datasets. Data comparing results on different datasets from [6, Table 3] including Flickr8 and Flickr30k, shows that for some other models the BLEU scores actually were slightly lower when trained on the Flickr30k as compared to the Flickr8k. However, the four models examined in [6, Table 3] did show improved scores when trained on the much larger MS COCO dataset, suggesting some dependency on training set size.

Due to its depth and number of fully-connected nodes, the VGG16 model is over 533MB as it has many weight parameters and thus a long inference time, a smaller network architecture, or when with greater accuracy may yield better results [27]. A different choice of photo encoder may be chosen to meet accuracy and/or inference time expectations. ResNets take lesser memory, have faster inference time, and allow deeper networks to be trained [27]. Many of the models listed in Table 3 are also available pretrained. Everything else being the same, the more parameters the slower the network, complicated activations functions are slower than simple ones such as ReLU [28]. In general, other things being equal, deeper networks will be slower

than shallow networks, and the type of connections/units is important as well. More recurrent units will make for a slower networks, and a convolutional network will be faster than one which is fully connected [28].

| Model | Size | Top-1 Accuracy | Top-5 Accuracy | Parameters | Depth |
|---|---|---|---|---|---|
| Xception | 88 MB | 0.790 | 0.945 | 22,910,480 | 126 |
| VGG16 | 528 MB | 0.713 | 0.901 | 138,357,544 | 23 |
| VGG19 | 549 MB | 0.713 | 0.900 | 143,667,240 | 26 |
| ResNet50 | 99 MB | 0.749 | 0.921 | 25,636,712 | 168 |
| InceptionV3 | 92 MB | 0.779 | 0.937 | 23,851,784 | 159 |
| InceptionResNetV2 | 215 MB | 0.803 | 0.953 | 55,873,736 | 572 |
| MobileNet | 16 MB | 0.704 | 0.895 | 4,253,864 | 88 |
| MobileNetV2 | 14 MB | 0.713 | 0.901 | 3,538,984 | 88 |
| DenseNet121 | 33 MB | 0.750 | 0.923 | 8,062,504 | 121 |
| DenseNet169 | 57 MB | 0.762 | 0.932 | 14,307,880 | 169 |
| DenseNet201 | 80 MB | 0.773 | 0.936 | 20,242,984 | 201 |
| NASNetMobile | 23 MB | 0.744 | 0.919 | 5,326,716 | - |
| NASNetLarge | 343 MB | 0.825 | 0.960 | 88,949,818 | - |

**Table 3. [28] Comparison of models commonly used in computer vision tasks.**

It may be possible to improve the model through improving the preprocessing of the text corpus. This would require a more detailed analysis of the corpus including investigating which words occur most and least frequently. Particularly for Flickr30 or if a different larger dataset is used, it may be possible to change the frequency threshold at which words are removed from the corpus (i.e. they must appear at least three times instead of twice).

This model could be improved most by adding an attention mechanism. Attention mechanisms consider spatial aspects of an image relevant to the parts of the image captions, they can concentrate on the salient parts of the image and generate the corresponding words at the same time. A possibility would be similar to the method of Jia et al [9]. Jia et al. alter the LSTM model by adding semantic information as an extra input to the gate of each LSTM memory cell. This input can take different forms as long as they build a semantic connection between the image and its description, a semantic embedding, a classification or a retrieval result. Jia et al. found that current methodologies are heavily biased towards short sentences and this hurts the quality, thus they propose sentence normalization. The three different normalization methods found most useful are report in their results: polynomial, min-hinge, and Gaussian. These normalization methods when couples with a gLSTM brings much improvement in the performance of the captioner. Their model performs on par with more complicated and expensive attention mechanisms and other methods which use an ensemble of multiple LSTM models [31].

Another method which adds attention to a model of this basic encoder-decoder which could be adopted here is described by Lu et al. [34]. The basis of the model is that some words can be predicted easily from the language model while others cannot. At each time step, the model decides whether to attend to the image (and if so, to which regions) or to the visual sentinel. This helps extract meaningful information for sequential word generation.

Although as seen in Table 2, the implementation of a gMLE generator by Dai et al. [13] achieved the best results, this model is completely different than the one implemented here so making adjustments to achieve similar results are not feasible.

## V. CONCLUSION

The captioner model developed was able to make accurate captions in many instances when trained on either the Flickr8K or Flickr30K dataset. The model does not, however, always produce a completely accurate caption. It is unclear what percentage of the time this happens. However, the BLEU scores when trained on both datasets are not far below that of some other models with which it is compared.

There are some important ethical considerations in automatic image captioning. For some images humans may produce sample captions which are very similar. However, many images are more prone to interpretation and with that ambiguity comes the room for bias or stereotypes. The example captions and images should be as diverse as possible. If objects that are only found in certain cultures are not present in the dataset this will impact the effectiveness of the captioner. If the subjects producing captions for datasets on which captioners are trained have any sort of bias this can be imparted into the captions generated. This can have an effect on how people respond to the images captioned, but could be especially harmful if the automatic captioning is used for a task such as text-to-speech software for the visually impaired, where the caption is all the information that is received by the user. This may be a particular issue when captioning images from diverse cultures, recognizing emotions which may be displayed differently across cultures, or for images which are specifically designed to provoke an emotional response.

Here we provide a captioner model which works well in some situations but still has problems with captioning certain images. The syntax of the captions is universally acceptable, however, the content varies. This demonstrates the complexity of the image captioning task. There are many different types of models for image captioning, and no one model is a sufficient representation of a class as whole. There are promising advances in many subfields of image captioning.

The field of image captioning is developing models which can more accurately caption more complicated images, including those with numerous features and less common configurations of elements. An extension of this work would aim to add an attention mechanism to the model, test on the MS COCO dataset, and evaluate results using other metrics. Future work may also aim to compare captions created from existing models, and compare the requirements of each model to determine the “best” model for different scenarios.

## VI. REFERENCES

[1] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," In Transactions of the Association for Computational Linguistics, Dec 2014; 2: pp. 67-78.

[2] T Lin, M. Maire, S. Belongie, J. Hays J, P. Perona , D. Ramanan, P. Dollár , C. Zitnick. "Microsoft coco: Common objects in context," In *European Conference on Computer Vision*, Sep 6 2014, pp. 740-755.

[3] J. Deng, W. Dong, R. Socher, L. Li ,K. Li , and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," In *2009 IEEE conference on computer vision and pattern recognition*, Jun 2009, pp. 248-255.

[4] R Krishna, et al, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," *International Journal of Computer Vision*, vol. 123, no. 1, May 2017, pp. 32-73.

[5] P. Sharma, N. Ding, S. Goodman, and R Soricut. "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning," In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, Vol. 1: Long Papers, Jul 2018, pp. 2556-2565.

[6] Z. Hossain, MD, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Comput. Surv*, vol. 51, no. 6, Feb 2019, 36 pages, art no. 18.

[7] J. Mao, W. Xu, Y. Yang, J Wang , Z Huang, and A.Yuille, "Deep captioning with multimodal recurrent neural networks (m-rnn)," arXiv preprint arXiv:1412.6632. 2014 Dec 20.

[8] J. Johnson, A. Karpathy, and L. Fei-Fei, "Densecap: Fully convolutional localization networks for dense captioning," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2016*, pp. 4565-4574.

[9] X. Jia, E. Gavves, B. Fernando, T. Tuytelaars, "Guiding the long-short term memory model for image caption generation," In *Proceedings of the IEEE international conference on computer vision 2015*, pp. 2407-2415.

[10] H Wang, et al, "From captions to visual concepts and back" In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1473-1482.

[11] P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould, L. Zhang. "Bottom-Up and top-down attention for image captioning and visual question answering," In the *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6077-6086.

[12] Q. You, H. Jin, Z. Wang, C. Fang, and J. Luo, "Image captioning with semantic attention," In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2016, pp. 4651-4659.

[13] Dai Z, Almahairi A, Bachman P, Hovy E, Courville A. "Calibrating energy-based generative adversarial networks," arXiv preprint arXiv:1702.01691. Feb 2017.

[14] M. Wang, L. Song, X. Yang, and C. Lu, "A parallel-fusion RNN-LSTM architecture for image caption generation," In *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016, pp. 4448-4452.

[15] A. Hendricks L, S. Venugopalan, M. Rohrbach, R. Mooney R, K. Saenko, T. Darrell. "Deep compositional captioning: Describing novel object categories without paired training data," In *Proceedings of the IEEE conference on computer vision and pattern recognition* 2016, pp. 1-10.

[16] Yao T, Pan Y, Li Y, Qiu Z, Mei T. "Boosting image captioning with attributes," In *Proceedings of the IEEE International Conference on Computer Vision* 2017, pp. 4894-4902.

[17] K. Shuster, S. Humeau, H. Hu, A. Bordes, J. Weston. "Engaging image captioning via personality," The *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR), 2019, pp. 12516-12526.

[18] A. Chandrasekaran A, D. Parikh, M. Bansal, "Punny captions: Witty wordplay in image descriptions," arXiv preprint arXiv:1704.08224. 2017 Apr 26.

[19] K Papineni, S Roukos, T Ward, and W Zhu, "BLEU: a method for automatic evaluation of machine translation," In *Proceedings of the 40th annual meeting on association for computational linguistics*, Jul 2002, pp. 311-318.

[20] C. Callison-Burch, M. Osborne, and P. Koehn, "Re-evaluation the Role of Bleu in Machine Translation Research," In *EACL*, Vol. 6, pp. 249–256.

[21] C. Lin. "Rouge: A package for automatic evaluation of summaries," In *Text summarization branches out: Proceedings of the ACL-04 workshop*, 2004, Vol. 8. Barcelona, Spain.

[22] S. Banerjee and A. Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments," In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization,* Jun 2005, pp. 65-72.

[23] P. Anderson, B. Fernando, M. Johnson, S. Gould. Spice: "Semantic propositional image caption evaluation," In *European Conference on Computer Vision,* Oct 8 2016, pp. 382-398.

[24] R. Vedantam, L. Zitnick C, D. Parikh. "Cider: Consensus-based image description evaluation," In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 4566-4575.

[25] Y. Cui, G. Yang, A. Veit, X. Huang, and S. Belongie, "Learning to evaluate image captioning," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 5804-5812.

[26] Simonyan K, Zisserman A. "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014 Sep 4.

[27] "VGG-Convolution Network." Neurohive. Nov 2018. Accessed from https://neurohive.io/en/popular-networks/vgg16/

[28] "Data Science: Which is the fastest image pretrained model?" Accessed November 27, 2019 from https://datascience.stackexchange.com/questions/39177/which-is-the-fastest-image-pretrained-model

[29] X. Chen and C. L. Zitnick. "Mind's eye: A recurrent visual representation for image caption generation," In *CVPR*, 2015, pp. 2422–2431.

[30] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. "Long-term recurrent convolutional networks for visual recognition and description," In *CVPR*, 2015, pp. 2626–2634.

[31] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. "Show and tell: A neural image caption generator," In *CVPR*, 2015, pp. 3156–3164.

[32] Chen L, Zhang H, Xiao J, Nie L, Shao J, Liu W, Chua TS. "SCA-CNN: Spatial and channel-wise attention in convolutional networks for image captioning," In Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5659-5667.

[33] K. Xu, et al. "Show, attend and tell: Neural image caption generation with visual attention," *International conference on machine learning*, June 1 2015, pp. 2048-2057.

[34] J. Lu, C. Xiong, D. Parikh, and R. Socher. "Knowing when to look: Adaptive attention via a visual sentinel for image captioning," In *Proceedings of the IEEE conference on computer vision and pattern recognition,* 2017, pp. 375-383.

# APPENDIX: PYTHON CODE

- The data file names/paths are hard-coded in— these need to be changed for different datasets.
- This method also loads the data and runs the model in one file. Loading the data can take a long time. If run multiple times, load the data separately and alter the code accordingly.
- Python version 3.6 was used.

## *A. File to Create Model, Train and Store Best Model to .h5 file*

```
import string
from numpy import array
from pickle import load, dump
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_ing
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from os import listdir

###PREPARE PHOTOS

#function returns dictionary of image identifier to image features
def extract_features(directory):
        # load the model, this is a pretrained model available form keras.  Another model could be used
        model = VGG16()
        #remove last layer- not classifying
        model.layers.pop()
        model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
```

```
	# extract features photos
	features = dict()
	for name in listdir(directory):
		# load an image from file
		filename = directory + '/' + name
		image = load_img(filename, target_size=(224, 224))
		image = img_to_array(image)
		# reshape data for the model
		image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
		# prepare the image for the VGG model
		image = preprocess_input(image)
		feature = model.predict(image, verbose=0)
		image_id = name.split('.')[0]
		features[image_id] = feature
		print('>%s' % name)
	return features

# extract features from images
directory = '/home/jcushing1/Flicker8k_Dataset/'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
dump(features, open('features.pkl', 'wb'))


###PROCESS TEXT DATA

# load into memory
def load_doc(filename):
	file = open(filename, 'r')
	text = file.read()
	file.close()
	return text

filename = '/home/jcushing1/Flickr8k_text/Flickr8k.token.txt'
doc = load_doc(filename)

# extract descriptions for images

def load_descriptions(doc):
	mapping = dict()
```

```
        # process lines
        for line in doc.split('\n'):
                # split line by white space
                tokens = line.split()
                if len(line) < 2:
                        continue
                # first token is image id, the rest is the description
                image_id, image_desc = tokens[0], tokens[1:]
                # remove filename from image id
                image_id = image_id.split('.')[0]
                #put description back into string
                image_desc = ' '.join(image_desc)
                # create the list if needed
                if image_id not in mapping:
                        mapping[image_id] = list()
                # store description
                mapping[image_id].append(image_desc)
        return mapping

# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))


def clean_descriptions(descriptions):
        # prepare translation table for removing punctuation
        table = str.maketrans('', '', string.punctuation)
        for key, desc_list in descriptions.items():
                for i in range(len(desc_list)):
                        desc = desc_list[i]
                        # tokenize
                        desc = desc.split()
                        # convert to lower case
                        desc = [word.lower() for word in desc]
                        # remove punctuation from each token
                        desc = [w.translate(table) for w in desc]
                        # remove hanging 's' and 'a'
                        desc = [word for word in desc if len(word)>1]
                        # store as string
                        desc_list[i] =  ' '.join(desc)
```

```
# clean descriptions
clean_descriptions(descriptions)


# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
        # build a list of  description strings
        all_desc = set()
        for key in descriptions.keys():
                [all_desc.update(d.split()) for d in descriptions[key]]
        return all_desc

# size of vocal
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))


# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
        lines = list()
        for key, desc_list in descriptions.items():
                for desc in desc_list:
                        lines.append(key + ' ' + desc)
        data = '\n'.join(lines)
        file = open(filename, 'w')
        file.write(data)
        file.close()

# save descriptions
save_descriptions(descriptions, 'descriptions.txt')


filename = '/home/jcushing1/Flickr8k_text/Flickr8k.token.txt'
# load, parse, clean descriptions
doc = load_doc(filename)
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
clean_descriptions(descriptions)
# summarize vocabulary
```

```
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
# save to file
save_descriptions(descriptions, 'descriptions.txt')

####BUILDING THE  MODEL

# load a list of photo identifiers
def load_set(filename):
	doc = load_doc(filename)
	dataset = list()
	# process line by line
	for line in doc.split('\n'):
		# skip empty lines
		if len(line) < 1:
			continue
		# get the image identifier
		identifier = line.split('.')[0]
		dataset.append(identifier)
	return set(dataset)



# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
	# load document
	doc = load_doc(filename)
	descriptions = dict()
	for line in doc.split('\n'):
		# split line by white space
		tokens = line.split()
		# split id from description
		image_id, image_desc = tokens[0], tokens[1:]
		if image_id in dataset:
			# create list
			if image_id not in descriptions:
				descriptions[image_id] = list()
			# wrap description in tokens
			desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
			descriptions[image_id].append(desc)
	return descriptions
```

```
# load photo features
def load_photo_features(filename, dataset):
	# load all features
	all_features = load(open(filename, 'rb'))
	# filter features
	features = {k: all_features[k] for k in dataset}
	return features

# load training dataset
filename = '/home/jcushing1/Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
train_descriptions = load_clean_descriptions('descriptions.txt', train)

# photo features
train_features = load_photo_features('features.pkl', train)

# convert a dictionary of clean descriptions to a list
def to_lines(descriptions):
	all_desc = list()
	for key in descriptions.keys():
		[all_desc.append(d) for d in descriptions[key]]
	return all_desc

# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
	lines = to_lines(descriptions)
	tokenizer = Tokenizer()
	tokenizer.fit_on_texts(lines)
	return tokenizer

# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)



# create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, descriptions, photos, vocab_size):
	X1, X2, y = list(), list(), list()
```

```
        # walk through each image identifier
        for key, desc_list in descriptions.items():
                # walk through each description for the image
                for desc in desc_list:
                        # encode the sequence
                        seq = tokenizer.texts_to_sequences([desc])[0]
                        # split one sequence into multiple X,y pairs
                        for i in range(1, len(seq)):
                                # split into input and output pair
                                in_seq, out_seq = seq[:i], seq[i]
                                # pad input sequence
                                in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                                # encode output sequence
                                out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                                # store
                                X1.append(photos[key][0])
                                X2.append(in_seq)
                                y.append(out_seq)
        return array(X1), array(X2), array(y)

# calculate the longest description (most words)
def max_length(descriptions):
        lines = to_lines(descriptions)
        return max(len(d.split()) for d in lines)

###DEFINE MODEL

# define the captioning model
def define_model(vocab_size, max_length):
        # feature extractor model
        inputs1 = Input(shape=(4096,))
        fe1 = Dropout(0.5)(inputs1)
        fe2 = Dense(256, activation='relu')(fe1)
        # sequence model
        inputs2 = Input(shape=(max_length,))
        se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
        se2 = Dropout(0.5)(se1)
        se3 = LSTM(256)(se2)
        # decoder model
        decoder1 = add([fe2, se3])
```

```
        decoder2 = Dense(256, activation='relu')(decoder1)
        outputs = Dense(vocab_size, activation='softmax')(decoder2)
        # tie it together [image, seq] [word]
        model = Model(inputs=[inputs1, inputs2], outputs=outputs)
        model.compile(loss='categorical_crossentropy', optimizer='adam')
        # summarize model
        print(model.summary())
        plot_model(model, to_file='model.png', show_shapes=True)
        return model

# train dataset

# load training dataset (6K)
filename = '/home/jcushing1/Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
# photo features
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# prepare sequences
X1train, X2train, ytrain = create_sequences(tokenizer, max_length, train_descriptions, train_features, vocab_size)


# load dev set
filename = '/home/jcushing1/Flickr8k_text/Flickr_8k.devImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
```

```
# prepare sequences
X1test, X2test, ytest = create_sequences(tokenizer, max_length, test_descriptions, test_features, vocab_size)

model = define_model(vocab_size, max_length)
# define checkpoint callback
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
# fit model
model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint],  validation_data=([X1test, X2test], ytest))

filepath = 'model-ep{epoch:03d}-loss{loss:.3f}-val_loss{val_loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode="min")

# fit model
model.fit([X1train, X2train], ytrain, epochs=20, verbose=2, callbacks=[checkpoint], validation_data=([X1test, X2test], ytest))# fit model
```

B. Calculation of BLEU Scores

```
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
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 pickle import dump

# load doc into memory
def load_doc(filename):
	# open the file as read only
	file = open(filename, 'r')
	# read all text
	text = file.read()
	# close the file
	file.close()
```

```
    return text


# load a pre-defined list of photo identifiers
def load_set(filename):
    doc = load_doc(filename)
    dataset = list()
    # process line by line
    for line in doc.split('\n'):
        # skip empty lines
        if len(line) < 1:
            continue
        # get the image identifier
        identifier = line.split('.')[0]
        dataset.append(identifier)
    return set(dataset)

# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    # load document
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        # split id from description
        image_id, image_desc = tokens[0], tokens[1:]
        # skip images not in the set
        if image_id in dataset:
            # create list
            if image_id not in descriptions:
                descriptions[image_id] = list()
            # wrap description in tokens
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            # store
            descriptions[image_id].append(desc)
    return descriptions

# load photo features
def load_photo_features(filename, dataset):
```

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

# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)

# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    # seed the generation process
    in_text = 'startseq'
    # iterate over the whole length of the sequence
    for i in range(max_length):
        # integer encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad input
        sequence = pad_sequences([sequence], maxlen=max_length)
```

```
            # predict next word
            yhat = model.predict([photo,sequence], verbose=0)
            # convert probability to integer
            yhat = argmax(yhat)
            # map integer to word
            word = word_for_id(yhat, tokenizer)
            # stop if we cannot map the word
            if word is None:
                  break
            # append as input for generating the next word
            # append as input for generating the next word
            in_text += ' ' + word
            # stop if we predict the end of the sequence
            if word == 'endseq':
                  break
      return in_text

# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
      actual, predicted = list(), list()
      #over whole set
      for key, desc_list in descriptions.items():
            # generate description
            yhat = generate_desc(model, tokenizer, photos[key], max_length)
            # store actual and predicted
            references = [d.split() for d in desc_list]
            actual.append(references)
            predicted.append(yhat.split())
      # calculate BLEU score
      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=(0.33, 0.33, .33, 0)
      print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, .25, .25)

# prepare tokenizer on train set

# load training dataset (6K)
filename = '/home/jcushing1/Flickr8k_text/Flickr_8k.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
```

```
# descriptions
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
# prepare tokenizer
tokenizer = create_tokenizer(train_descriptions)
vocab_size = len(tokenizer.word_index) + 1
# determine the maximum sequence length
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)

# prepare test set

# load test set
filename = 'Flickr8k_text/Flickr_8k.testImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))

# load and evaluate model
filename2 = '/home/jcushing1/model-ep003-loss3.658-val_loss3.881.h5'   #model saved previously
load_model(filename2)
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

C.  Create Caption for a New Photo File

```
#"max_length" value would need to be changed for different dataset
#image file input is near the bottom, change file name accordingly

from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
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
from keras.models import load_model

# extract features from each photo in the directory
def extract_features(filename):
	# load the model and remove final layer (for classifying)
	model = VGG16()
	model.layers.pop()
	model = Model(inputs=model.inputs, outputs=model.layers[-1].output)
	# load the photo
	image = load_img(filename, target_size=(224, 224))
	image = img_to_array(image)   # convert the image pixels to a numpy array
	image = img_to_array(image)
	# reshape data for the model
	image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2$
	# prepare the image for the VGG model
	image = preprocess_input(image)
	# get features
	feature = model.predict(image, verbose=0)
	return feature

# map an integer to a word
def word_for_id(integer, tokenizer):
  for word, index in tokenizer.word_index.items():
		if index == integer:
			return word
	return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
	#add token to start the generation
	in_text = 'startseq'
	# iterate over length of the sequence
for i in range(max_length):
		# integer encode input sequence
		sequence = tokenizer.texts_to_sequences([in_text])[0]
		# pad input
		sequence = pad_sequences([sequence], maxlen=max_length)
		# predict next word
		yhat = model.predict([photo,sequence], verbose=0)
```

```
            # convert probability to integer
            yhat = argmax(yhat)
            # map integer to word
    word = word_for_id(yhat, tokenizer)
            # stop if we cannot map the word
            if word is None:
                break
            # append as input for generating the next word
            in_text += ' ' + word
            # stop if end of the sequence
            if word == 'endseq':
                break
        return in_text # load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
# the max sequence length (from training)
max_length = 34
# load the model
model = load_model(model-ep002-loss4.262-val_loss4.184.h5)
# load and prepare the photograph
photo = extract_features(‘/home/jcushing1/example_image10.jpg')
#generate and print description for new image
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```