香　港　大　學

THE UNIVERSITY OF HONG KONG

Image Captioning for the Visually Challenged

COMP3359 - Final Report

Shivansh Mittal (3035347330)

Mridul Agarwal (3034453529)

May 11, 2020

Background

1.1 Project Objectives

The project aims to perform image captioning on real-time images captured through a mobile phone camera, with the ultimate aim of helping the visually challenged people to describe the image through verbal description. The process will be to first capture a picture from the camera, caption the significant image frames in an image using an AI model, convert text to speech, and return results to the user via a mobile application.

1.2 About the Dataset

The project uses the MS-COCO dataset [1] for model training. It is an image object detection, segmentation, and captioning dataset, containing over 330K images (>200K labelled), each of which has at least 5 different caption annotations. Since it was too expensive to use the complete dataset, a subset of 80K captions has been used.

MS-COCO datatset is apt because it contains captions that are less specific and provide more accurate and general descriptions of the images. Unlike 1k(Pascal) and 31k(Flikr), MS-COCO datatset's captions contain fewer static words and many verbs. Moreover, the descriptions rarely contain non-visual attributes and are not unnecessarily long [2].

1.3 System architecture

The project has two aspects: *Machine Learning* and *Mobile App Development*. The system architecture for both has been discussed below.

1. To carry out the machine learning model construction, training, and evaluation, Tensorflow (GPU version), Keras, along with many other Python modules have been used. Since it was understood that the model would be trained on images and that computationally, it would be extremely expensive, therefore initially, the ideas was to use a Google Cloud Platform's TPU instance [3]. However, due to the high cost, HKU CS GPU Farm server was used instead. A Jupyter Notebook kernel was hosted on the server.

2. **If time permits**, using React Native, a cross platform mobile application will be developed for the project. The mobile application will be connected to the machine learning model via Flask (Python) [4] server which will be hosted on a Google Cloud Platform's virtual machine. A text-to-speech API offered by Google [5] could be used to speak out the captions and inform the user.

Methodology

2.1 Preparing the Data

The following pre-processing steps were carried out to prepare the training and testing datasets, containing images and corresponding captions [6].

1. First, MS-COCO 2017 training dataset is downloaded and the image paths and corresponding captions are stored in separate lists. After shuffling the lists together, the first 80,000 samples are selected to be used for our model's training and testing.

2. Next, pre-trained weights of the InceptionV3 model on the Imagenet dataset are used to extract features from each image in our dataset. To match the InceptionV3 model input format, the images are converted into 299x299 pixel size and normalized to the range [-1, 1].

3. Lastly, the captions are tokenized i.e. separated by spaces and a dictionary (word-to-index) of all the unique words is created. To save memory, the top 5000 words (most frequently occurring) are selected to represent our vocabulary and the rest are replaced with a "<unk>" tag. Then, all the captions in our dataset are converted to sequences of words and a corresponding index-mapping is also maintained. Finally, size of the longest caption sequence is noted and all other sequences are padded with the tag "<pad>" to be of that same size.

2.2 Transfer Learning

A Keras model is created, which is essentially the same as the InceptionV3 model [7], but with the last layer of classification removed, since our project is not required to conduct classification of images, but rather to extract the features from the images which can be associated with certain words from the vocabulary. Each 299x299px image is passed through the model's multiple convolutions and an output tensor of dimensions 8x8x2048 is obtained. For a large dataset like ours, it is necessary to cache the resulting output tensors by mapping them to their image paths and storing the dictionary on the disk.

2.3 The AI Model

The project replicates the model architecture as described in [8]. The individual components are discussed below.

- **Bahdanau Attention Model-** In 2015, Bahdanau et al. proposed a model which searches for a set of positions in the encoder hidden states where the most relevant information is available [9] for sentence generation. Attention mechanism is an attempt to implement the action of selectively concentrating on a few relevant things, while ignoring others, in deep neural networks. Typically, Attention models were used to extract few important words and pass it to a decoder for machine translation. Later, this mechanism, or its variants, was used in other applications, including computer vision, speech processing, etc.

- **CNN-** A CNN encoder is a network that takes the image as input, and outputs a feature tensor. These feature tensors hold the information about the features that represent the input. The decoder is again

a network (usually the same network structure as encoder but in opposite orientation) that takes the feature vector from the encoder, and gives the best closest match to the actual input or intended output. Once trained, the encoder will give feature tensors for the input that can be used by the decoder to construct the input with the features that matter the most to make the reconstructed input recognizable as the actual input.

- **RNN Decoder (GRU)-** The RNN Decoder is a stack of several recurrent units where each unit predicts an output. Each recurrent unit accepts a hidden state from the previous unit and produces an output, as well as its own hidden state. The output sequence is a collection of all words from the generated image caption. We calculate the outputs using the hidden state at the current time step together with the respective weight. Softmax is used to create a probability vector which will help us determine the final output.

2.4 Training

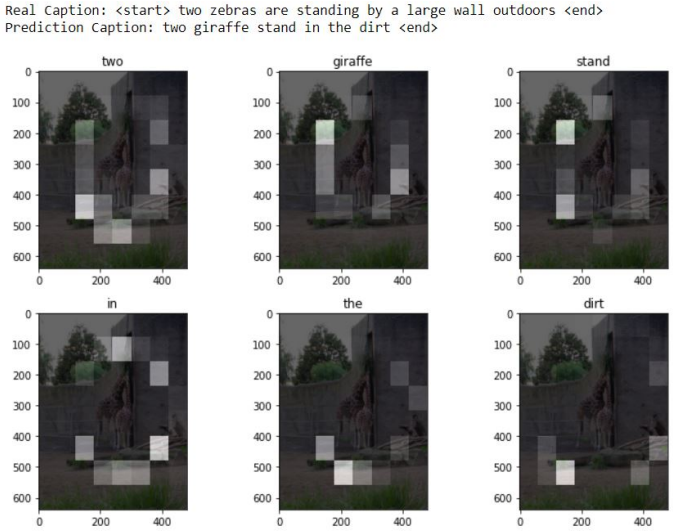
1. The dataset is split into training and validation sets using an 80-20 split. Therefore, the 80k images are separated into sets of 64k and 16k, respectively.
2. We define a class BahdanauAttention which is used to extract some important features from our images.
3. Then, we define a CNN encoder which takes the extracted features from pickle files and passes them through a fully connected layer.
4. We also define an RNN encoder which considers input from the previous state of decoder and present input from encoder followed by Attention.
5. The RNN generates embeddings of size 256, passes them to GRU through Glorot Uniform initializer and finally calls BahdanauAttention.
6. Next, we train the model using SparseCategoricalCrossentropy loss function and an Adam optimizer.
7. The CNN encoder's output, and the RNN decoder's input is passed to the decoder.
8. The decoder returns the predictions and the decoder's hidden state. The decoder's hidden state is then passed back into the model and the predictions are used to calculate the loss. The final step is to calculate the gradients and apply it to the optimizer and back-propagate.

Experiments and Results

3.1 Testing Base Model Captioning

We begin by having vocabulary size = 5k words and training the model for 20 epochs. In Figure 3.1, we observe that the model, even without any fine-tuning, is able to generate a decent caption. It is interesting to see that even though the real caption wrongly labels the giraffes in the image as zebras, our model gets it right. Moreover, it also gets the number of giraffes and the background right.

(a) OriginaliImage



(b) Predicted caption

Figure 3.1: Base Model Captioning

3.2 Increasing Number of Epochs from 20 to 50

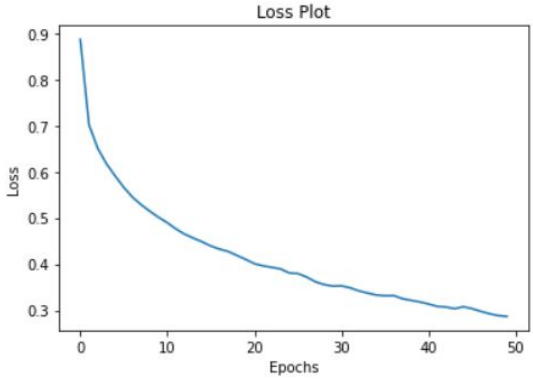
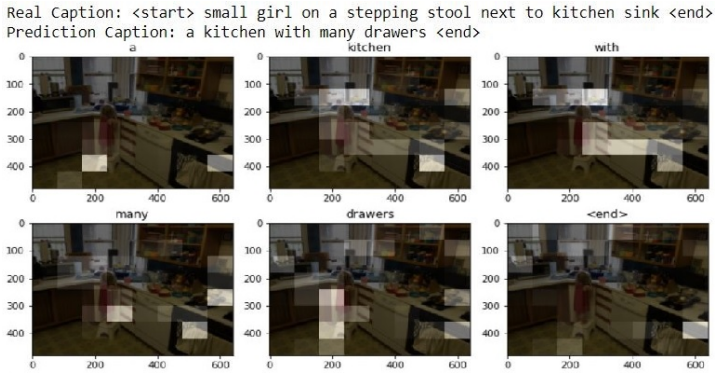


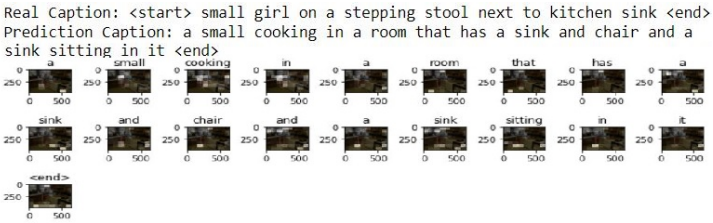
Figure 3.2: Loss plot- Increasing number of epochs from 20 to 50

Keeping the same vocabulary size and training dataset size, we trained the model for 50 epochs instead of 20. The comparison of the loss values for both cases can be drawn from Figure 3.2. A lower loss value indicates that the model is learning well and converging towards zero loss. Since the plot is smooth, we know that the learning rate of the model is justified.

(a) Original Image



(b) Epochs = 20



(c) Epochs = 50

Figure 3.3: Captioning comparison

(a) Simple image

(b) Complex image

Figure 3.4: Caption comparison between Simple and Complex images

Let's compare the captions generated for the image in Figure 3.3a, as shown in Figure 3.3. Figure 3.3b shows that 20 epochs are sufficient to form a correct, short, and crisp caption. Figure 3.3c, in contrast, shows that 50 epochs return a much detailed caption, describing individual items in the kitchen. It is worth nothing that both captions have no mention of the girl, as can be seen from the attention plots of the two cases. This suggests that either increasing the number of epochs for training will help in captioning the image more accurately, or the model has already overfitted the training dataset. To answer this question, we will analyze various evaluation metrics in Section 3.4.

Moreover, simple images are captioned quite well after 50 epochs, as in Figure 3.4a. Figure 3.4b shows that the grammar of the caption suffers due to increased complexity of the picture. This issue can perhaps be addressed by training on a dataset larger than 64k images or increasing vocabulary size.

3.3 Increasing Vocabulary Size from 5k to 8k

Starting from number of epochs = 20 again and keeping the same training dataset size, we increased the size of the vocabulary from top 5k words to 8k.

(a) Coined "skateboard"

(b) Coined "fire hydrant"

Figure 3.5: Predicted captions with niche objects

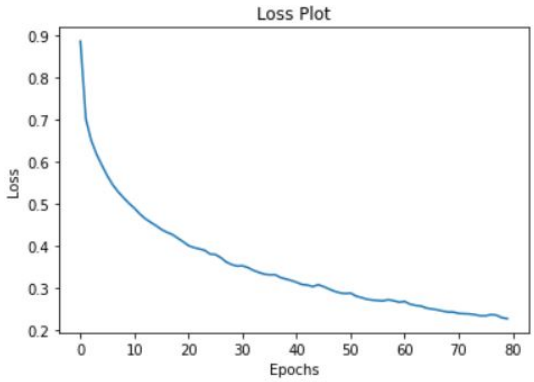


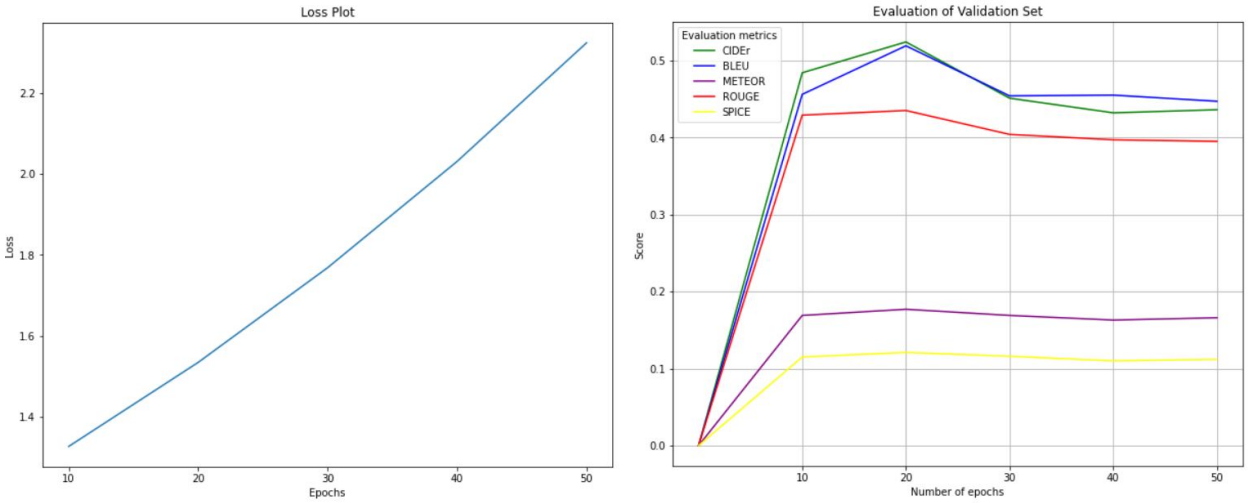
Figure 3.6: Loss plot- Training up to 80 epochs

As can be seen in Figure 3.5, the model is now able to include niche words in the generated captions,

thereby, describing the overall images more accurately. Previously, these words would be replaced by the ”<unk>” tag. We decided to train the model for up to 80 epochs to bring down the loss value to a region even lower that that shown in Figure 3.2. The resulting loss plot is shown in Figure 3.6

3.4 Evaluation

Thus far, we have only assessed the captions generated for randomly selected images from the validation set of 16k images. Now, we will introduce some evaluation metrics which specifically cater to image captioning models, to quantify the performance of our model and understand overfitting.



(a) Loss plot- Validation set

(b) Performance evaluation on validation set using different metrics

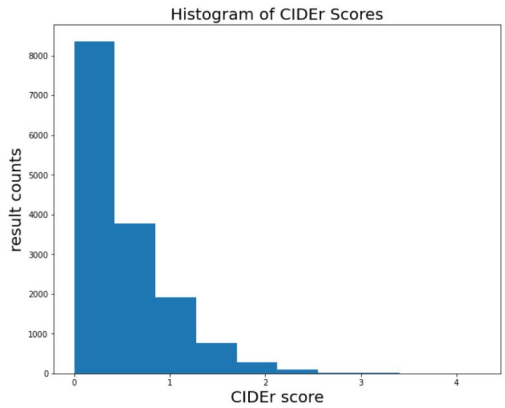
Figure 3.7: Understanding overfitting through plots

Before we look at the metrics, we can compare the loss values on the validation set to get an initial idea about overfitting. The loss plot in Figure 3.7a suggests that with every 10 epochs of training, our model's generalisation performance worsens. Since the plot is nearly linear, we can assume that the performance will be even worse for the model trained for 80 epochs, as in Section 3.3.

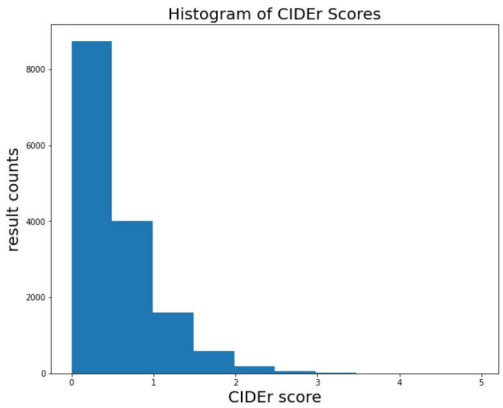
To compute performance evaluation, we have used the following metrics: CIDEr, BLEU, ROUGE, METEOR, and SPICE. These are all popular metrics used for performance evaluation purposes in various NLP domains. Figure 3.7b shows the trend of scores for different evaluation metrics on the validation set against the number of epochs the model is trained for. It can be observed that for all metrics, the score rises from 0 epochs to 20 epochs. The scores are highest for all metrics at 20 epochs, suggesting that epochs = 20 is the appropriate cut-off for training the model. Thereafter, the scores for all metrics fall. We can see that the rate of change of scores with increasing number of epochs is the most drastic for CIDEr and BLEU. Due to this sensitivity, they are also regarded as the two most popular image captioning performance evaluation metrics.

Compared to other evaluation metrics for image captioning, CIDEr gives more weightage to important n-grams and has a higher correlation with human consensus scores [10]. Thus, we perform further analysis using the CIDEr evaluation metric. Figure 3.8 shows the different histograms conveying information about the distribution of validation set data on the basis of its CIDEr score. We can see that the result counts corre-

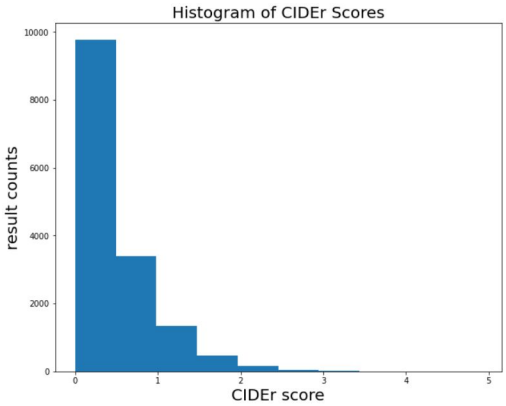
sponding to the score bucket of 0 - 0.5 keeps increasing for every 10 epochs. The result counts for the score bucket 0.5 - 1, which is a reasonable score region, are the highest for epochs = 20. As the number of epochs increase, the results counts from the high score buckets keep shifting towards the low score buckets. The few samples corresponding to the unreasonably high score buckets can be explained by the fact that they probably belong to the same distribution of images as the majority in the training dataset.



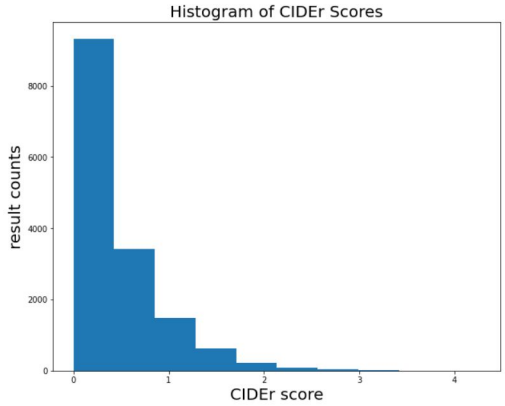
(a) Epochs = 10



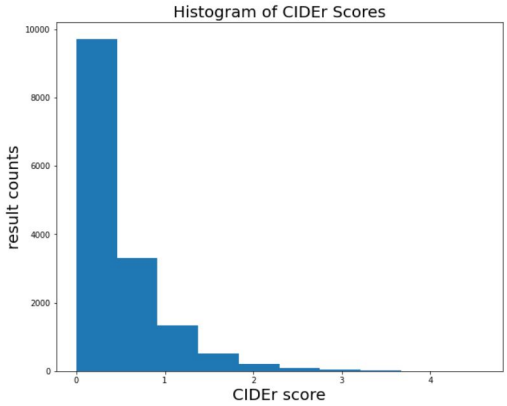
(b) Epochs = 20



(c) Epochs = 30



(d) Epochs = 40



(e) Epochs = 50

Figure 3.8: CIDEr Histograms

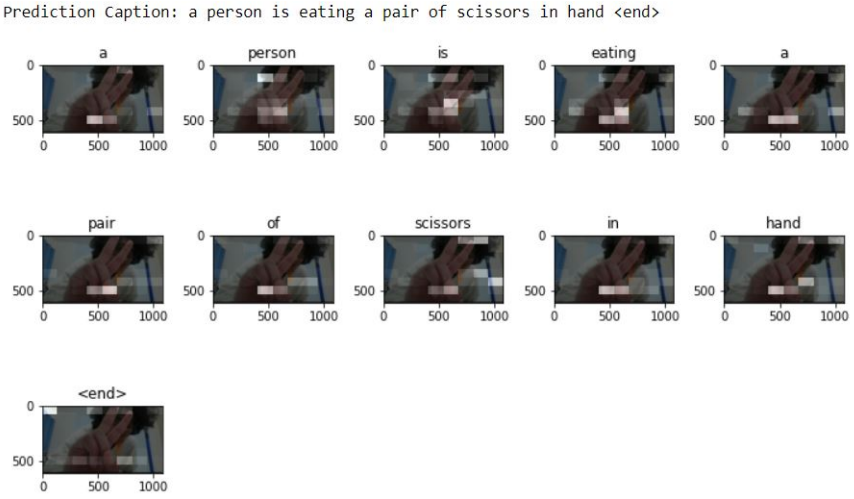
Taking all the analysis into consideration, we decide that the model trained using a vocabulary of 8k words for 20 epochs on a dataset of 80k total images, having train-test split of 80-20, works best for our image captioning application.

The straightforward way to understand the extent of overfitting is by comparing the values of evaluation metrics for different models on the training dataset and the validation dataset. However, all evaluation metrics mentioned above require the formulation and caching of the generated captions and then calculating the scores after comparing with ground truth captions. Doing this for 64k images of our training dataset is very resource exhaustive and time consuming. For similar reasons, the comparison between 5k vocabulary and 8k vocabulary models cannot be assessed. Due to these limitations, we perform an indirect way of evaluation to understand overfitting and choose the appropriate number of epochs to train the model.

3.5 Captioning a Real-Time Image

The biggest test for the model is captioning a real-time image. This is where using a vast and diverse dataset having images from different distributions, like MS-COCO, can help in obtaining good results. Figure 3.9 shows the captioning result on a screenshot taken from a real-time video call. While the sentence structure of the caption is not perfect, the image has been successfully described to a large extent.

(a) Original image



(b) Predicted caption

Figure 3.9: Screenshot from a video call

Ideas for Improving the Model

Following are some points that can be considered to improve the captioning results of the model [11].

- Due to RAM constraints, we were restricted to using a training datatset having only 64k images. A larger training datatset will retrieve better results.
- Further hyperparameter tuning, for example, trying different values for units, embedding dimensions, batch size, etc. can result in an improved model.
- Using LSTM instead of GRU for the RNN decoder.
- Build the mobile application to make the model useful in real-life.

References

[1] COCO. (). Common objects in context, [Online]. Available: http://cocodataset.org/#home. [Accessed: March 29, 2020].

[2] S. Ganju. (2016). Exploring image captioning datasets, [Online]. Available: http://sidgan.me/technical/2016/01/09/Exploring-Datasets. [Accessed: April 16, 2020].

[3] G. C. Platform. (). Cloud tpu, [Online]. Available: https://cloud.google.com/tpu. [Accessed: March 30, 2020].

[4] T. P. Projects. (). Flask, [Online]. Available: https://palletsprojects.com/p/flask/. [Accessed: March 30, 2020].

[5] G. Cloud. (). Text to speech, [Online]. Available: https://cloud.google.com/text-to-speech. [Accessed: March 30, 2020].

[6] T. Core. (). Image captioning with visual attention, [Online]. Available: https://www.tensorflow.org/tutorials/text/image_captioning. [Accessed: March 29, 2020].

[7] K. Documentation. (). Inceptionv3, [Online]. Available: https://keras.io/applications/#inceptionv3. [Accessed: April 16, 2020].

[8] X. et al. (2016). Show, attend and tell: Neural image caption generation with visual attention, [Online]. Available: https://arxiv.org/pdf/1502.03044.pdf. [Accessed: March 28, 2020].

[9] B. et al. (2014). Neural machine translation by jointly learning to align and translate, [Online]. Available: https://arxiv.org/abs/1409.0473. [Accessed: April 16, 2020].

[10] K. Kundu. (). Csc2539 - datasets and metrics for image caption generation, [Online]. Available: http://www.cs.toronto.edu/~fidler/slides/2017/CSC2539/Kaustav_slides.pdf. [Accessed: April 16, 2020].

[11] H. Lamba. (2018). Image captioning with keras, [Online]. Available: https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8. [Accessed: March 29, 2020].