Visual-to-audio aid for visually Impaired – Training image captioning module

Matriculation Number: 00503319, 00616219

Progress Synopsis

First Results: Successfully trained an image captioning module with limited dataset
Major Stumbling Blocks: Minor infrastructure limitations
Time frame for completion: 3 more sessions
Will we meet the deadlines: Yes

Dataset

Name: MS COCO
Number of Images: >80,000
Number captions per image: 5
Size: 13GB

Roadblocks –

	Training Requirements / DataSet Issues	Infrastructure-at-hand
Pickled data and model checkpoints	Persistent Storage	Volatile Storage
InceptionV3 Input image size	299 X 299	Random Size
InceptionV3 Input image pixel size	-1 to 1: 3 channels	0 to 255: 3 channels
Caching preprocessed images in RAM	8 * 8 * 2048 floats per image	RAM: 12GB (not enough)
Caption data size	414113 captions,	Future processing not feasible
vocabulary on RAM (tokenizer size)	>25,000 words, ~1.8GB RAM	RAM: 12GB (reserved for training)
training epochs	15 Minutes per epoch	90-minute timeout

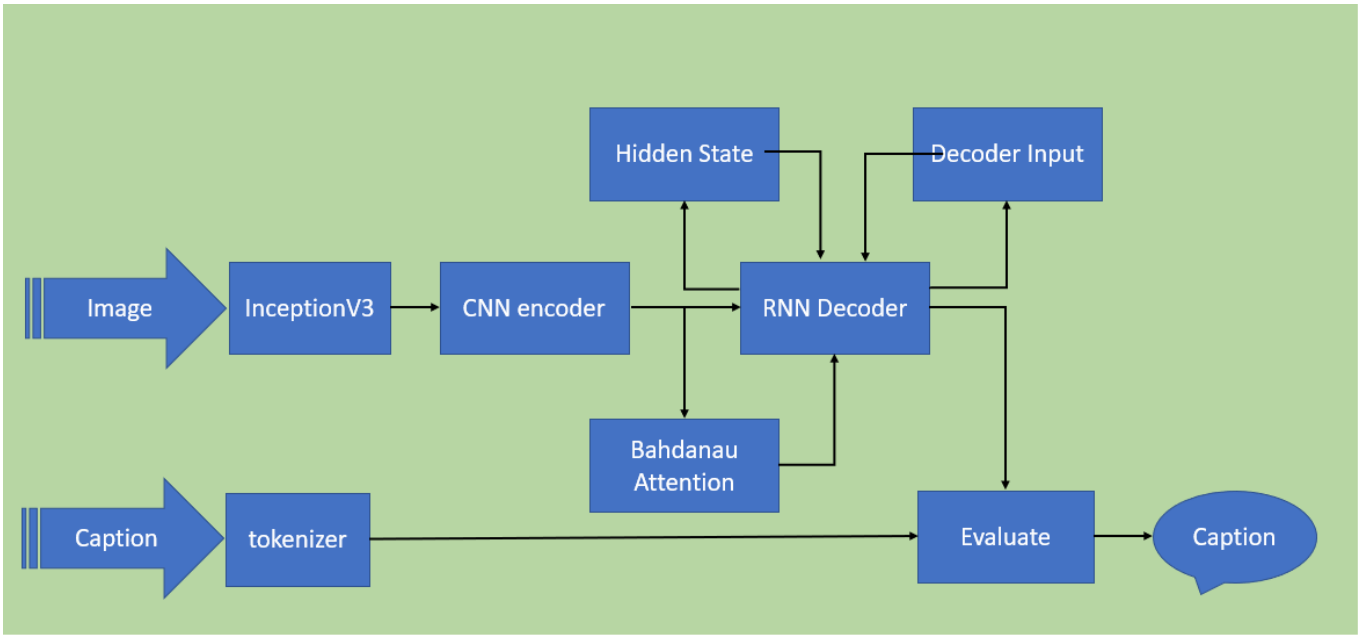
Corrective Measures

	Fullfillment	Resolution/Limitation
Pickled data and model checkpoints	fail	use google drive
InceptionV3 Input image size	fail	preprocess on loading
InceptionV3 Input image pixel size	fail	use preprocess_input method
Caching preprocessed images in RAM	fail	save to disk(longer time)
Caption data size	fail	limit to 50000 captions
vocabulary on RAM (tokenizer size)	fail	limit to top 5000 words
training epochs	fail	limit to 40 epochs

Components

- **Tokenizer** – It will tokenize the vocabulary and store it for easy mapping as we need to be working with numbers instead of words
- **InceptionV3** – it is pretrained with imagenet and will be used to preprocess the input images for our model, the feature matrix and corresponding images will be stored in a dictionary and given to the model
- **CNN encoder** – it will convolute the extracted feature matrix for each image by passing it through a fully connected layer
- **Bahdanau Attention model** – It will find the specific areas of interest in the given image, where we need to attend in order to remove unwanted things using softmax over scores of attention weights
- **RNN decoder** – It uses the attention model along with a GRU and a fully connected layer to predict the next word in the caption recurrently.
- **Adam Optimizer** – Adaptive moment estimation takes up on an adaptive learning rate as opposed to classical schotastic gradient to perform much better backpropagation
- **Sparse Categorical Cross Entropy loss function** – Computes the loss for our output which is a sparse categorical matrix data of the size same of our vocabulary

Architecture



Step 0: Pre-requisites

- We import tensorflow, matplotlib, numpy, os, time, pickle, PIL, glob and train_test_split, shuffle from sklearn.
- We mount our google drive through drive library in google.colab.
- We set up the pickles and checkpoints folder in google drive mount.

Step 1: Download and limit dataset for training

- We modify the captions/annotations to include the <start> and <stop> keywords.
- We create the vectors of random 50,000 captions and respective images.
- We define the load_image function to facilitate the loading and resizing the image as per InceptionV3 standards. The function also calls preprocess_input function from InceptionV3 in Keras library to normalize pixels in the range of -1 to 1 which is also required for InceptionV3.

Step 2: Pre-process the Images – extract features and cache the dictionary to disk

- We load the InceptionV3 application from tensorflow.keras library with imagenet weights into a model.
- We find out unique images in our training images vector using a sorted set and use dataset.map function to create a map of image and path for the unique images. We do this to avoid working multiple times on same Image.
- We use the model to preprocess and extract the feature vector for each image and store it to the disk as numpy files

Step 3: Pre-process the captions – tokenize and create the vocabulary

- We create a tokenizer object from keras library, initialize it with the 5000 top words limitation and <unk> for the rest of words.
- We fit the tokenizer on the training captions vector and convert the sentences into numeric sequences to be used later.
- We find out the max length and pad all sequences to the same length for ease of use with RNN.
- We pickle the max length and tokenizer objects for use in isolated testing environment.

Step 4: Data Preparation and Hyperparameter setting

- We divide the dataset into 80% training and 20% validation datasets
- We set the Hyperparameters for training as below,

```
BATCH_SIZE = 64                    // for shuffling and prefetching
BUFFER_SIZE = 1000
embedding_dim = 256                // Input dimensions
units = 512                        // number of GRU units in RNN
vocab_size = top_k + 1             // +1 for <pad> token
num_steps = len(img_name_train)    // to calculate the loss
# Shape of the vector extracted from InceptionV3 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 64
```

- We load the numpy files into a dataset, then shuffle and prefetch the dataset into buffer.

Step 5: Define Model

- We extract the features from the lower convolutional layer of InceptionV3 giving us a vector of shape (8, 8, 2048).
- We squash that to a shape of (64, 2048).
- This vector is then passed through the CNN Encoder, which consists of a single Fully connected layer activated with ReLU non linearity for some degree of learning.
- We define a Bahdanau Attention Model with two Dense layers of ‘units’ as mentioned in parameters and a dense layer of 1 unit to calculate the score. Finally, we use softmax to select features requiring attention according to score.
- The RNN (here GRU) attends over the image using the Bahdanau Attention model created above.
- The RNN then passes the selected features(context_vector) through a GRU layer of ‘units’ as mentioned in parameters and then through two additional dense layers, one with ‘units’ number of nodes and one with Vocab_Size number of nodes.
- The score for the whole vocab size is compared to select the next word.

Step 6: Define Optimizer, loss function and checkpoint manager

- We define our Adam optimizer and Sparse Categorical cross entropy loss functions with the help of respective components from tensorflow.
- We define the checkpoint manager with the encoder, decoder, optimizer and google drive checkpoint path.
- to facilitate multiple runs, we check if any checkpoints are existing using checkpointmanager.latest_checkpoint object and if so they the weights and starting epoch are loaded instead of a fresh start with epoch = 0 and weights as zero vectors.

Step 7: We define the training step function

- We extract the features stored in the respective '.npy' files and then pass those features through the encoder.
- The encoder output, hidden state (initialized to 0) and the decoder input (which is the start token) is passed to the decoder.
- The decoder returns the predictions and the decoder hidden state.
- The decoder hidden state is then passed back into the model and the predictions are used to calculate the loss.
- We use teacher forcing to decide the next input to the decoder. Teacher forcing is the technique where the target word is passed as the next input to the decoder.
- We calculate the gradients with respect to loss and trainable variables and apply it to the optimizer and backpropagate to optimize the losses.

Step 8: Training

- We define the number of epochs we want to train it for.
- We start a timer and initialize batch_loss to zero.
- We enumerate the dataset and pass the batch of image tensor and target vector for each caption to train step.
- With each batch iteration the train_step continually updates the encoder, decoder and attention model according to the Adam optimizer.
- For every 5 epochs completed the checkpoint manager is called to save a checkpoint to be used in isolated testing environment or in case of discontinuity.
- The epoch loss and time taken to complete the epoch is then shown before moving on to next epoch.
- The epoch vs loss is plotted for inspection for convergence of the overall model (In case of discontinuity the plot is not saved).

Step 8: Testing environment preparation

- We repeat Step 1 in separate environment.

- Then we Initialize InceptionV3 with step 3 but do not attempt to pre-process any dataset as there is none for now.
- The Pickled Tokenizer and max_length objects are loaded into memory from pickles folder
- We then repeat steps 4-6 or rather copy the code from training environment without any changes.
- The checkpoint manager is defined and called upon to load weights from checkpoints folder on drive.

Step 9: Define evaluate, plot_attention function and run evaluation

- We reset the decoder hidden state as we will process a new image.
- We then preprocess the new test image provided through InceptionV3 to get the image tensor. We do not store it in disk this time as it can be accommodated in RAM and will be faster.
- The image tensor extracted in previous step is then passed through our encoder and similar to the training environment, the features are extracted.
- The decoder input is set to <start> token.
- For the max_length of caption we pickled earlier we run a loo to predict the next word till <end> token is received.
- In the loop we pass the decoder input and features (encoder output) through our decoder and receive the prediction, hidden state and attention weights.
- The tokenizer object loaded from pickle is used to get the word according to the prediction from decoder and is then appended to the result.
- The attention plot function is then defined to plot which areas were focused during prediction.
- The test image is then passed through evaluate function and attention plot along with results are shown.

Example Test Sample