Generation of Captions from Images

Tianyuan Cai

This project aims to use neural network to generate sensible captions for images. By describing objects of images with text, an image captioning model can translate complex images into more compact data formats. This type of model can be applied in a wide range of fields such as voice over technologies, product recommendation, etc.

The training and validation data come from Common Objects in Context (COCO) dataset produced by Microsoft. COCO dataset provides a set of images that identifies common objects in context. The data include both images and captions, which describes the objects and their context.

I make use of the Inception V3 model and GloVe embeddings to create image and caption vectors as inputs. I then use LSTM model combined with fully connected layers to perform word-by-word prediction to generate sensible next words based on the training images and the portion of the caption that has already been generated. The model is run on the input sentence sequence and image until a full sentence is created, before continuing to create captions for the next image. The final model is able to produce sensible results, as shown in the example below:

A young man is jumping to catch a tennis ball.

A plain white restroom toilet appointment in corner.

Overall, the final model can produce sensible results after tuning the batch size hyperparameter. However, there are also several areas of improvements to address issues in the areas of prediction bias, performance measure, and hyperparameter tuning.

- The model suffers from bias due to the biased distribution of the data set. For instance, the model tends to predict all people as “man” possibly due to the frequent usage of the word “man” in the caption and the frequent presence of male subjects in the images. Data augmentations and more gender-aware captions can help reduce this type of bias in training and prediction.
- Model performance is measured by the categorical loss, BLEU score, as well as human judgment on how sensible the captions are. However, a systematic way to identify semantic similarities between the predicted and actual captions is needed to more accurately measure the overall model performance.
- Given the large amount of data and the limited computation resources, it was difficult to use the grid search function in sklearn to perform hyperparameter tuning. I used the for loop to tune the batch size parameter, but more computation resources are needed to perform grid search in combination with other parameters. If I had more time, I will write out a custom function for grid search.

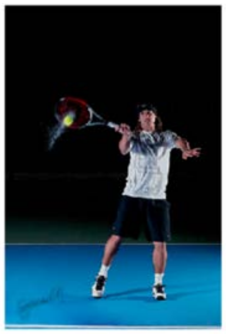
Link to YouTube Video: https://youtu.be/_IFIf4Gn4i0

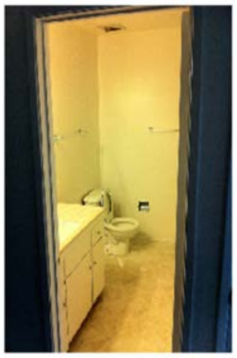
Overview

The goal of this project is to generate sensible captions for images. Image captioning has a large variety of applications. By understanding common objects in an image and transforming them into textual information, we can compress large photographic information into more compact data formats. The possibility of this type of transformation makes it easier to make inferences based on photographic data and gives rise to applications such as voice over technologies, recommender engine, etc.

In this project, I use the Inception-V3 and GloVe implementations to transform images and words into feature vectors. I then pass these data through LSTM hidden layer and produce sensible captions using word-by-word predictions. The training and validation data come from Common Objects in Context (COCO) dataset produced by Microsoft[1].

Here are some examples of the image captions predicted by the model:


A young man is jumping to catch a tennis ball.


A plain white restroom toilet appointment in corner.

In this document, I will go through the setup, data preparation, model training, and tuning. In the end, I will discuss the prediction results and the areas for improvements.

[1] The original data set is accessed from http://cocodataset.org. The data set is introduced in the research paper "Microsoft COCO: Common Objects in Context" in https://arxiv.org/abs/1405.0312.

Preparing Environment

I used Anaconda with Python 3.7 run on an Ubuntu 18.04 system to train and test this model. An Nvidia Titan Xp GPU was used to train the model, and 32 GB of RAM was available during training. The final data take up around 20 GB. All data are downloaded and extracted within the Jupyter Notebook, so the submission only requires the Jupyter Notebook itself to run. It helps, however, to run the notebook in an empty project folder to ensure the required data can be unpacked smoothly in the folder.

Despite the availability of relatively powerful graphics card, I still made use of methods such as the tqdm package, data generator class of Python, and fit generator function from Keras to accommodate less powerful systems. Batch size, number of training images and captions, sentence length, etc. can all be adjusted to accommodate a system of lower configuration. The notebook is tested on both remote Ubuntu server and Google Colab to make sure there is no error when running. Here are some system preparations needed before starting to run the Jupyter Notebook.

- Make sure there are at least 40 GB of disk space and sufficient memory available before running the code.
- Install TensorFlow GPU in Anaconda by running the following command. It will install the package as well as all dependencies, such as Cuda and cuDNN.
  conda install -c anaconda tensorflow-gpu
- After installing TensorFlow, use pip to install the following packages. pycocotools is dependent on Cython. Therefore, Cython must be installed first. Note that this command works for Python 3. If pip does not point to pip3 by default, change that to pip3.
  - *pip install pillow numpy matplotlibpydotpylabscikit-image keras Cython pycocotools nltk seaborntqdm*
  - If pycocotools was not installed or was unable to import. Use the following command after installing Cython.
    pip3 install

"git+https://github.com/philferriere/cocoapi.git#egg=pycocotools&subdirectory=PythonAPI"

- Data set used by this code will all be downloaded to the project folder. Change the line *data_dir = "/home/tcai/Documents/nlp/final_project"* to your own project folder. For instance, if you are using Google Colab to run this code, you can change this line to *data_dir = "/content"*.
- Once the packages are installed and the data directory updated. Running the notebook does not require any additional setup. All the data will download into the project folder, and the model will be saved by each epoch as it trains with appropriate names created for them.

Data

Three data sets are downloaded and unpacked as the code runs:

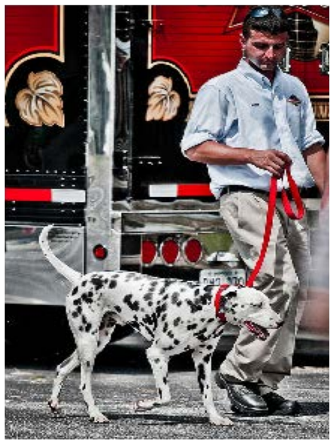
- Training image data set from Common Objects in Context (COCO) data set[2] produced by Microsoft. Due to the size of this data set, I obtain a subset of the downloaded data set and split it into train and validation set.
- Captions of the training images from COCO[3]. This data set contains captions that correspond to the images in the training images data set.

The COCO dataset provides a set of images of common objects in their natural context. The images are collected from a variety of sources, and the corresponding caption data provide multiple short captions to each image. The captions describe the objects in and the context of the image. Furthermore, the COCO API can be used to categorized by their super-categories, such as animal, furniture, sports, etc. for easy access. Here is an example of an image from the COCO data together with its corresponding captions.

[2] http://images.cocodataset.org/zips/train2017.zip
[3] http://images.cocodataset.org/annotations/annotations_trainval2017.zip

Captions:

- A man walking a dalmatian on a red leash.
- The man has a red leash on his Dalmatian dog.
- a man walks a dog with a leash.
- A guy is walking his dalmatian down the road.
- A man walking a dalmatian on leash in front of a firetruck.



The concise captions provide several layers of information including the objects, their relationships, and the context or the background of the image, each with some variety. For instance, note that in the image above, the person is said to be "a man", "a guy", and "the man"; The dog is said to be "Dalmatian dog", "dalmatian", and "a dog". Furthermore, the captions identify the relationship between the objects shown – the man is said to have a leash on his dog or walking the dog; the ownership of the dog is sometimes identified as well. Furthermore, the relationship between the objects and the context is also accurately reflected – the man and the dog are walking "in front of a firetruck".

However, the captions have their idiosyncrasies beyond semantic meanings. One caption has the first letter in lowercase, and others do not. One caption capitalized the first letter in "Dalmatian" while others do not. When processing the caption data, I transformed all words and removed special characters to ensure the captions are consistent. Similarly, the images in the data set also have inconsistent qualities. Therefore, additional preprocessing is performed on the images, such as converting images to 299 by 299 dimension before encoding them with the Inception V3 model.

When preparing the dataset, the captions, images paths, and encoded image features are all needed in order to effectively implemented the training. Rather than using train_test_split function, I choose to shuffle the data and subset to the desired number of train and test samples. Related data of each category share the same index throughout the analysis.

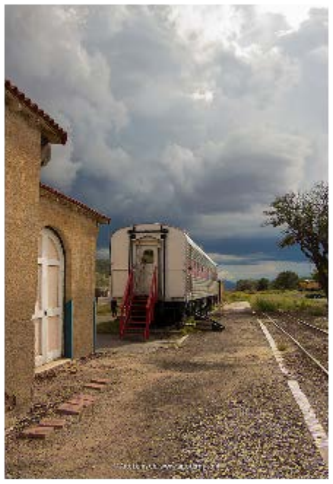
Again, note that the data set used by this code will all be created in the project folder. Look for the line data_dir = “/home/tcai/Documents/nlp/final_project” and change this directory into yours. Once the data set is downloaded and the images shuffled, I moved on to create image and text embeddings.

Image Embeddings

I use the Inception V3 model as an image encoder by removing the fully connected layers in the end. The lower-dimensional representation of the images reduces the amount of data needed by the model to perform the prediction of the captions. In addition, since the Inception V3 model is built to minimize the loss function for accurate object recognition, using the model as an encoder helps minimize this loss function for the training image data I used.

Below is an example of the prediction produced by the original Inception V3 model. The model outputs a score for each observed item, reflecting its confidence in the prediction.

- Freight car: 40.83%
- Passenger car: 29.35%
- Electric locomotive: 5.97%
- Steam locomotive: 1.17%
- Mobile home: 0.59%



To encode the image using Inception V3, I first transform images into 299 by 299 dimension, convert them to arrays, and then use the preprocessing function in Keras to preprocess the images. I then transfer the weight learned by Inception V3 on ImageNet data onto predicting the training data set. The output of the predict function is the encoded image features, and I store the output in a pickle file, indexed by image IDs so that I can use them for future uses. The 100,000 images in the training and validation data set took around 2,100 seconds to encode. The composition of the final data set is as follows:

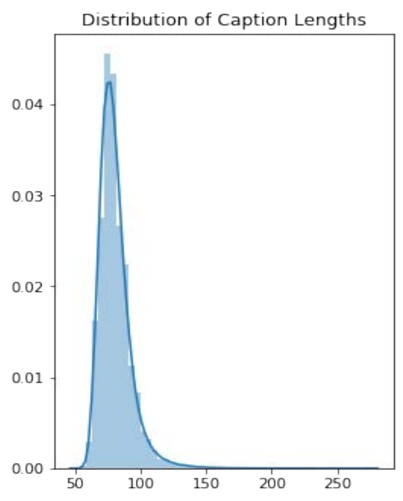
- Training: 80,000 distinct images, 400204 captions.
- Validation: 20,000 distinct images, 100057 captions.

Note that because each image is paired with multiple captions, the number of captions is larger than that of the images. When training images, I choose distinct images to ensure the same image is not encoded multiple times.

Word Embeddings

Before using the captions in the training and prediction, the data are first processed to remove special characters and tokenized. Later on, as the sentence is separated by words and supplied to the model word-by-word as training and validation, the sentence is then padded to the maximum length. In addition, sentence starting tag “start_sentence” and ending tag “end_sentence” are added to the tokenizer and the training captions to help identify the end of a caption.

After removing special characters and tokenize the words, the lengths of the sentences have the following distribution. I limit the maximum sentence length to 100, and the maximum number of words to 6,000. When setting the maximum sentence length, I referred to the sentence length distribution as follows. The value on the x-axis is the number of words in a sentence, and the values on the y-axis are the proportions of all the captions. The overall area covered by the following histogram equals one. The line characterizes the kernel density estimation. Most of the captions are less than 100 words.

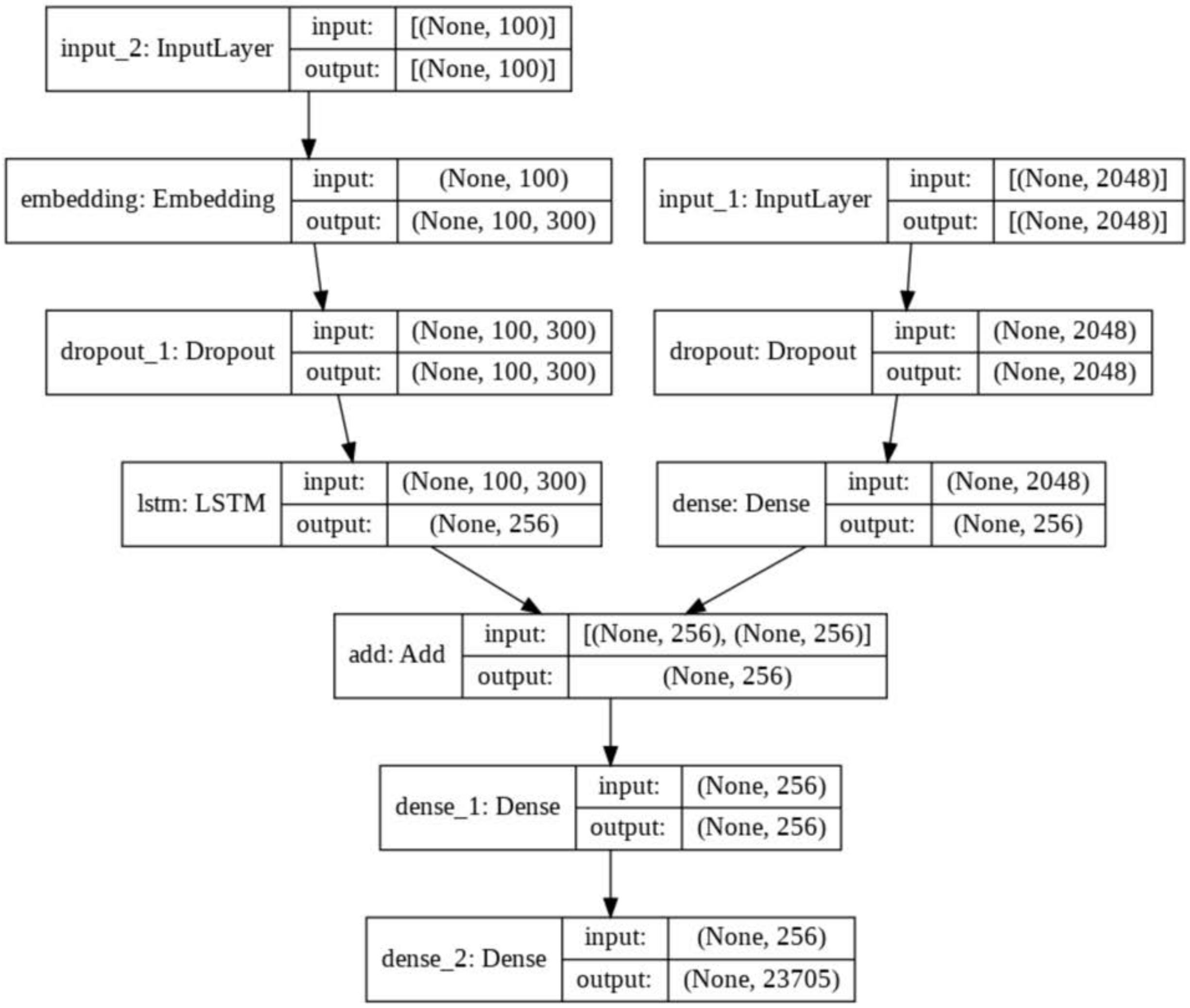


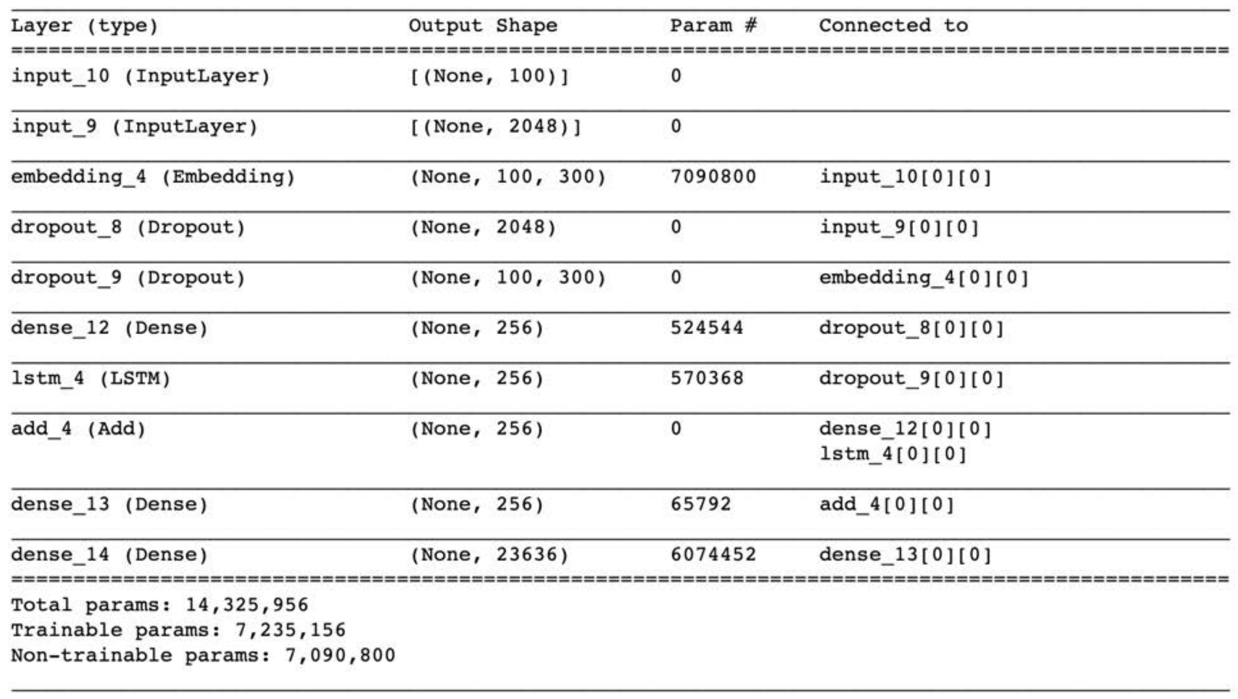
After preprocessing, I use Global Vectors for Word Representation (GloVe) to obtain embedding vectors for words in the captions. The position of a word within the vector space is learned based on the words that surround the word in the training data, in this case, on the Wikipedia 2014 and Gigaword 5 data (glove.6B.300d.txt)[4]. And I supplied the vector weights to be used in the embedding layer of the model.

Model

The original model framework used here is inspired by various online blog posts and tutorials on similar projects, but the code is largely my own. The use of Inception-V3 and for image encoding is inspired by the paper Show, Attend and Tell: Neural Image Caption Generation with Visual Attention.

[4] http://nlp.stanford.edu/data/glove.6B.zip


input_2: InputLayer | input: [(None, 100)] | output: [(None, 100)]
embedding: Embedding | input: (None, 100) | output: (None, 100, 300)
input_1: InputLayer | input: [(None, 2048)] | output: [(None, 2048)]
dropout_1: Dropout | input: (None, 100, 300) | output: (None, 100, 300)
dropout: Dropout | input: (None, 2048) | output: (None, 2048)
lstm: LSTM | input: (None, 100, 300) | output: (None, 256)
dense: Dense | input: (None, 2048) | output: (None, 256)
add: Add | input: [(None, 256), (None, 256)] | output: (None, 256)
dense_1: Dense | input: (None, 256) | output: (None, 256)
dense_2: Dense | input: (None, 256) | output: (None, 23705)



```
___________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
===================================================================================================
input_10 (InputLayer)           [(None, 100)]        0
___________________________________________________________________________________________________
input_9 (InputLayer)            [(None, 2048)]       0
___________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 100, 300)     7090800     input_10[0][0]
___________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 2048)         0           input_9[0][0]
___________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 100, 300)     0           embedding_4[0][0]
___________________________________________________________________________________________________
dense_12 (Dense)                (None, 256)          524544      dropout_8[0][0]
___________________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 256)          570368      dropout_9[0][0]
___________________________________________________________________________________________________
add_4 (Add)                     (None, 256)          0           dense_12[0][0]
                                                                 lstm_4[0][0]
___________________________________________________________________________________________________
dense_13 (Dense)                (None, 256)          65792       add_4[0][0]
___________________________________________________________________________________________________
dense_14 (Dense)                (None, 23636)        6074452     dense_13[0][0]
===================================================================================================
Total params: 14,325,956
Trainable params: 7,235,156
Non-trainable params: 7,090,800
___________________________________________________________________________________________________
```

In this model, input layers from images and captions vectors are passed into the model, dropout layers are applied to the respective inputs, and then the text input is passed through an LSTM layer while the image input through a fully connected layer to ensure the output has the same dimension as the LSTM output. The input tensors are then added together and passed through fully connected layers for next word prediction. To train and predict the caption word-by-word, the model starts by using the current image and the starting tag “start_sentence” to predict the first actual word of the caption. After this iteration, two words are in the input. Based on the “start_sentence” and the last word predicted, the model is then trained to predict the second actual word of the caption. This cycle continues until the model has predicted the “end_sentence” word. I will now explain why dropout layers and LSTM layer are used.

Dropout layers are used because they can reduce overfitting (regularizing), therefore improving the generalizability of the model. The dropout layer achieves this effect by randomly sampling and dropping outputs from the image and text input layers. Due to the limited number of images used, I decided to use a dropout layer because the large network used to train the model on this small data set has the potential of over-learning the signals in the images and text, picking up noise as signal as a result. A commonly used dropout value is 0.5, which means each output has 50% of the chance to be retained. I use a dropout value of 0.5.

After passing the image and text features through the dropout layer respectively, I passed the text input through a Long Short Term Memory (LSTM) layer, and the image feature input through a dense layer, before adding the inputs together. When generating a sentence, creating a sensible prediction of the next word requires an understanding of the word prior to it. Therefore, a recurrent neural network is used when the understanding of the prior data is required to produce more accurate prediction. While RNN is not competent at learning long-term dependencies, LSTM model is able to remember information for a long period of time. This is essential to the word-by-word prediction tasks because the last word of a caption might bear a strong connection to the first word. For instance, in the sentence “a plane fly through the sky”, the last word “sky” ties back to the second word “a plane”, because the subject is a “plane”, it can only fly through the “sky” rather fly through, say, the “road”. In this case, an understanding of the long-term dependencies is needed to successfully make an accurate prediction of the last word. On the other hand, the image data is passed through a fully connected layer to shrink the image features to the same size as text input.

After processing image and text by themselves, the Add layer combines the image and text tensors and return a single tensor before passing them through the last two fully connected layers for prediction. Based on the input sequence and the image features, the final model predicts the output of the next word. I then append the new word predicted by the model onto the existing list of words, which then serve as the new input sequence to be supplied to the model along with the same image. This prediction cycle repeats until the model has predicted the keyword “end_sentence” to let us know that it believes this is the appropriate place to end the sentence.

The model is trained on 80,000 image features together with their respective captions. The captions are each broken down to a sequence of input and output with each segment of the sentence as input. A sequence generator is used to take the first word as input to predict the second word, and then first and second words as input to predict the third word, and so on. The sequence generator is then placed inside the data generator to link the sequence inputs with the image inputs and yield these results to the fit generator of the model. The series of generators make it possible for the model to run through the word-by-word prediction on this large number of image vectors without running into memory issues.

At the first few epochs, the model exhibits underfitting behavior where no distinct captions are generated, but after around 60 epochs, the model starts to exhibit overfitting behavior where convoluted vocabularies are used to form captions that do not make much sense. While training the model, I spent additional time to evaluate sample output in person rather than relying solely on metrics such as categorical loss and BLEU scores, because sometimes the captions produced by the model with the low loss and high BLEU scores do not always make sense to a human reader, and the appeal to human readers is the best judge of the model performance.

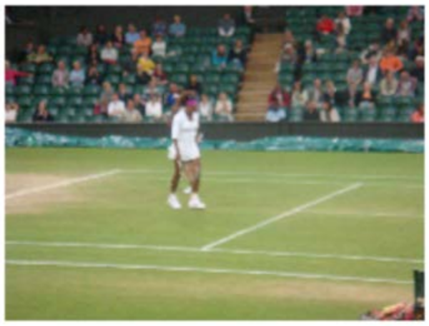
A wide range of parameters in the model can be tuned. Due to the limited computation resources and methods for tuning due to the use of the fit generator, I tuned the batch size of 500, 1000, 2000, and 3000. The model with a batch size of 500 produced the most sensible results overall among the 30 images I sampled. If I have more time, I will also tune the dropout rate, maximum sentence length, word frequency cut-off values (currently none), etc.

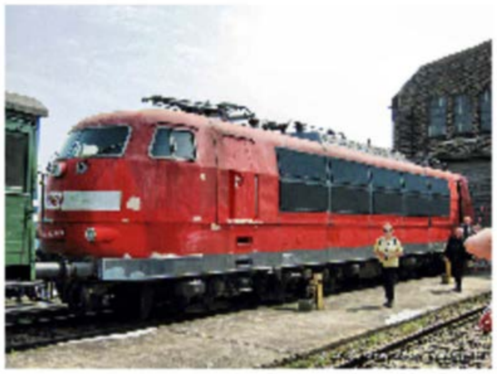
Discussion

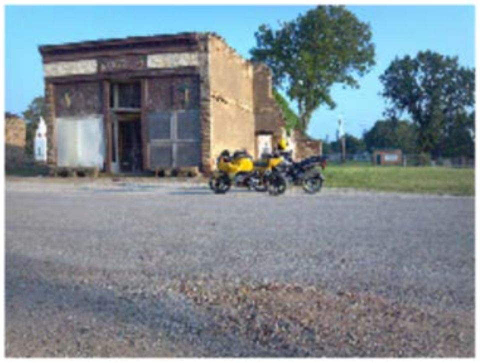
Results

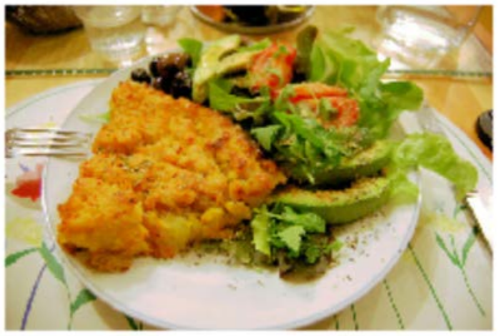
Here are some examples of the output from the model trained with a batch size of 500 after 50 epochs. I show four images with good captions, four with sensible ones, and four with surprising ones. However, overall, the “surprising” ones are hard to come by.

Good results

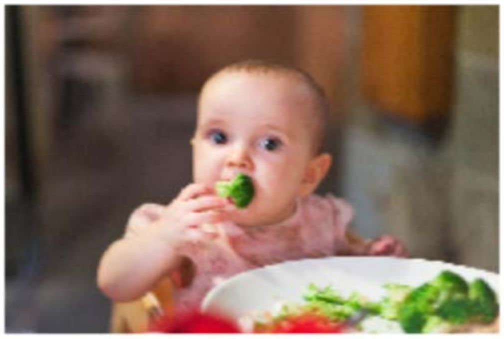

A professional tennis game with a lot of spectators.


A train comes down the tracks and enters the tunnel

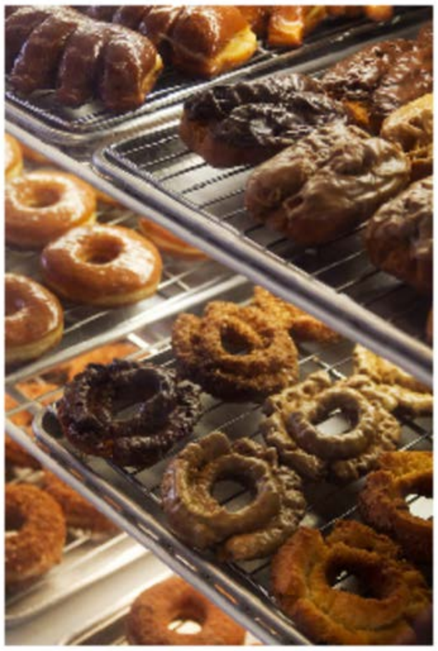

A green motorcycle parked in a parking space.


A buffet with lots of clutter and vegetables on a table.

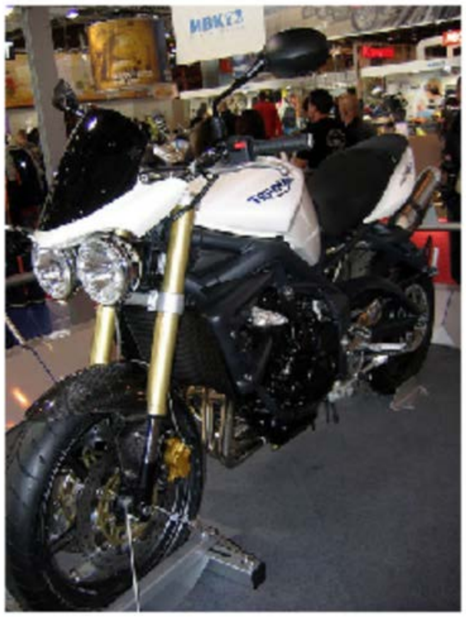
Sensible results



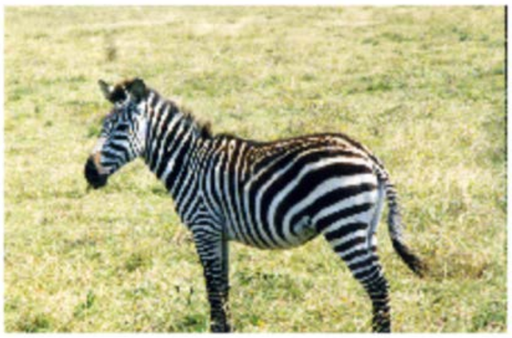
A lot of buckets of fruits including red and green apples.



Grouped fruits in boxes with handwritten price signs.

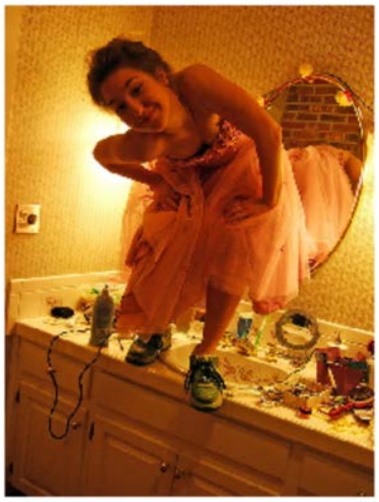


A golden motorcycle driving along a street near a truck space.

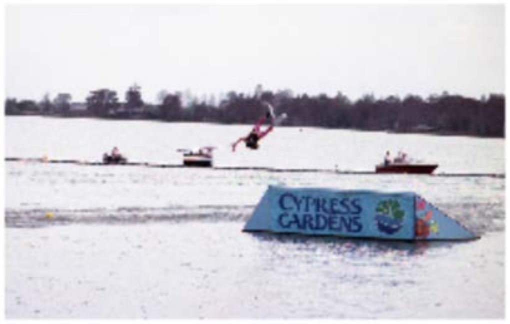


This is a image of an zoo outdoor.

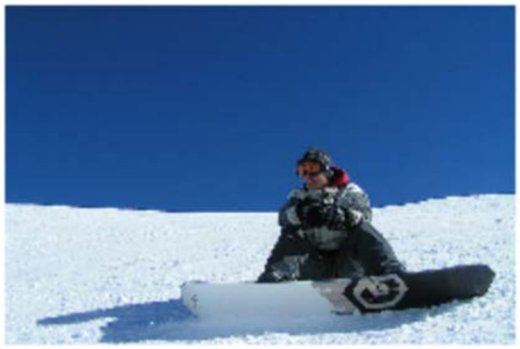
Surprising results



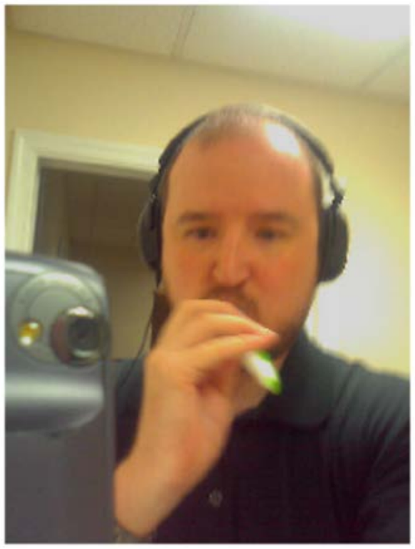
A cookie and orange on a table next to a tablet computer.



A man is standing next to others talking on a cell phone.



A man flies a kite on the beach.



A bearded man is wearing a tank top tie and a hat.

Learning

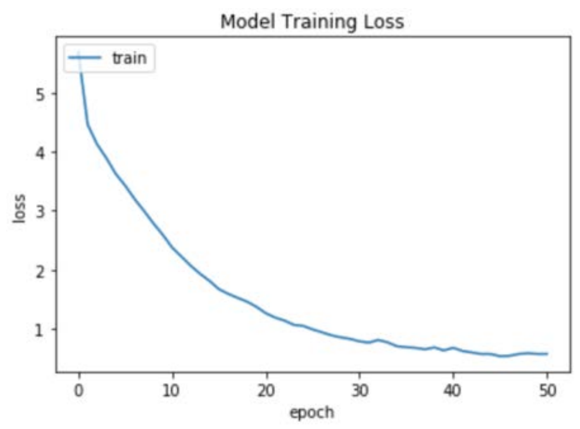


Image captioning requires the implementation of deep learning methods in both computer vision and natural language processing. While doing this project, I learned to use transfer learning by using image and word embeddings. Drop-out layers are included to prevent the model from overfitting, and model tuning is performed to ensure the model achieve reasonable performance.

Model tuning helps identify that the model tends to perform better with the batch size of 500 among the list of the values tested. Increasing the number of epochs is shown to help the model generate more complex vocabularies but doing that also cause the model to overfit easily. The final model is able to produce sensible captions for the images supplied. However, there remains potential for improvements.

Areas of Improvement

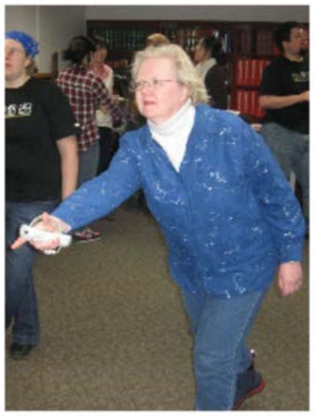
The model is able to provide reasonable captions given the images supplied, but the model appears to have difficulty distinguishing objects with similar property, and the model also has a bias towards objects and captions that appear more often in the data set. Last but not least, a better metric should be used to systematically measure model performance.

The model seems to have trouble distinguishing between objects that have subtle differences, such as sky and ocean, snowboards and surfboard, etc. For instance, the model mistakes a man performing a snowboard jump as someone surfing in the ocean, presumably because the blue sky in the background bears resemblance to the color ocean, and the skis appearing in the image looks similar to a surfboard. Data augmentation using data generators may help model better distinguish similar objects.

Furthermore, there appears to be bias in captioning due to the class imbalance in the training data set. When describing people in images, the model is more likely to predict a person as a "man" even when the subject is, in fact, a woman. This is likely caused by the nature of the original training captions. As evidence, in the "Word Distribution" section, you will notice that the "man" is the most popular word used after removing stop words, while "woman" is ranked 5th. Here is the top-five words with their respective frequencies in the data.

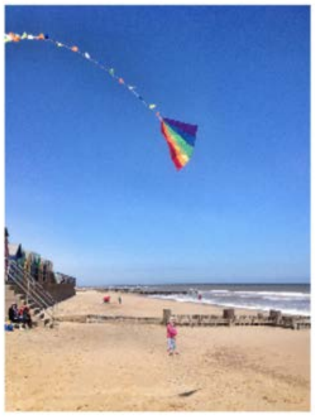
Word	Frequency
man	48873
sitting	35724
two	32401
standing	28438
people	27452
woman	26082

In this example, the model confuses man and woman. As shown in the list above, "man" is the more popular word.



```
The actual caption is:
an older woman playing nintendo wii near other people

The predicted captions is:
a man about a <unk> while hit a skateboard
```



```
The actual caption is:
a child flying a kite on the beach.

The predicted captions is:
a man about a kite on the air
```

One future improvement is to up-sample the female images and captions in the data set to improve the model's ability to recognize female. In addition, in the process of image labeling, the generic use of “man” when both genders appear should be avoided to help model better understand the gender differences.

The categorical loss measure for model accuracy is not a good measure of the model performance. While the model performance improves with categorical accuracy, at a low loss value, the model has a tendency of overfitting, producing vocabularies that do not fit well together as a sentence. The categorical loss fails to account for the readability of the content. I have tested out the use of BLEU score to measure caption similarity to the original caption, but BLEU score fails to account for the semantic similarity of the predicted caption and actual caption. In addition, due to the difficulty with the caption learning task in itself, the scale BLEU score provides fails to provide meaningful comparison across models. For instance, one might argue that the predicted caption “a baby giraffe eating leaves on a meadow” is a good approximation of the original one “A giraffe that is eating some leaves off of a tree”. However, a BLEU score of $1.39e{-}231$ is given in this case.

Lastly, I was unable to identify a way to perform grid search for hyperparameter tuning when fit_generator is used. In addition, my computation resource is limited. Therefore, I used a for loops for testing the model with different batch sizes. If time and resource permit, I will also tune dropout rate, maximum sentence length, word frequency cut-off values (currently none), etc.

Link to YouTube Video: https://youtu.be/_IFIf4Gn4i0