Generating Image Captions Using Deep Learning

Matt Mammarelli

***Abstract*—Automatically generating image captions for new images is a prime application for deep learning. In order to accomplish this, two datasets were used, the Flickr8kDataset which contains 8000 images and the Flickr8ktext dataset which contains corresponding descriptions for the images. First, the photo feature data was extracted using a pre-trained model like VGG16 or InceptionV3. Then some preprocessing was done on the text data before it was fed into the model. The photo and text data was then be loaded and fed into a model that was created. The model consisted of a recurrent neural network (RNN) with LSTM layers which provided the functionality of creating an intelligible sentence description. The model was then evaluated using bilingual evaluation understudy (BLEU) scores which measure how natural the language structure of the generated description is. Finally, this model was used to generate a caption for a photo the network had never seen. The loss while training the model was evaluated using both the VGG16 and InceptionV3 models in order to try to find if there was a measurable difference between the two. In addition, several techniques were applied to reduce validation loss including changing the LSTM unit size to 512, lowering the learning rate for the adam optimizer, changing the adam optimizer to Stochastic Gradient Descent (SGD), and changing the Dropout size. It was generally found that these alterations did not have a measurable impact on the validation loss.**

I. Introduction

Deep Learning is an application of machine learning where there is an emphasis on using a large amount of data to train and test a system. Usually deep learning is defined as a system that uses over 5,000 examples to train a system. Neural Networks are a non-linear arrangement of nodes or neurons that allow a system to classify a given example. They work through a system of using weights and inputs, where these the dot product of these two elements is calculated and compared to a threshold. The weights are updated through a process known as gradient descent where the network is back-propagated through and partial derivatives are calculated and distributed throughout the network. Recurrent neural networks (RNN) are a type of neural networks where there is a sort of feedback where previous outputs are used as inputs in sequential steps. Recurrent neural networks usually do not perform well on their own because the gradients usually explode or vanish. A common solution to this is using Long short-term memory (LSTM) units in the RNN. A very popular application for RNNs with LSTM layers is the automatic generation of text. The problem of trying to generate a naturally sounding caption to an image is a two part problem that is ideal for deep learning. The part where objects in the image are recognized is handled by computer vision and the part where text is generated is handled by the LSTM. The computer vision aspect can be handled by a pre-trained model like VGG-16 or InceptionV3. These models took an immense amount of computing power to train and have been tested in machine learning competitions like the ImageNet competition in 2014. VGG-16 is an older network but still performs well in computer vision tasks. InceptionV3 is newer and offers more accuracy in applications like this. Utilizing the weights from these networks instead of trying to generate new weights is essential in trying to expand on previous work rather than just trying to replicate it. This project is built on a working system that already has the ability to extract features from an image using VGG-16 and generate a caption from it. However, the loss and BLEU accuracy leaves room for improvement. The objective is to improve the loss by attempting various techniques including measuring the difference between VGG-16 and InceptionV3, testing different network architectures, and tuning hyperparameters including the learning rate and dropout. [1]

II. Methods

The first step to confronting this problem was getting the development environment properly setup. The development environment consisted of utilizing Python 3 along with the Python SciPy environment. In order to utilize deep learning for this application, Keras (2.1.5) with the Tensorflow backend was installed. Additional libraries that were installed are scikit-learn, Pandas, NumPy, Matplotlib, and pillow. A Google Cloud Compute virtual machine was created for this purpose and consisted of 4 vCPUs, 44GB of RAM, a 20GB HDD, and one Nvidia Tesla P100 GPU. Utilizing multiple GPUs for this problem was not necessary because this project relies heavily on LSTMs which are sequential in nature and would not benefit from the performance of multiple GPUs which specialize in parallel processing.

In order to get access to the photo dataset Flicker8kDataset and the text descriptions dataset Flicker8ktext, a request to the Department of Computer Science at the University of Illinois was sent. These datasets can be used for non-commercial and for educational purposes, but cannot be distributed. Once the datasets are downloaded, the process of working with the data was begun. The Flicker8kDataset was split so that 6000 were used for training, 1000 used for validation, and 1000 used for testing.

First the photo data was extracted using the VGG16 and the InceptionV3 Models. The VGG16 model won the 2014 ImageNet competition while the InceptionV3 model is newer and offers better performance. The process of extracting features was performed using both of these models and evaluated later to determine if there was less loss when training and validating the model. Keras provides both of the models as imports and can automatically reshape the images into 224x224x3 channel

images. The features were extracted for every photo utilizing preparephotodata.py and saved to the features.pkl file using pandas. This was tested on both CPU and on GPU and was verified that running on GPU ran significantly faster.

Some preprocessing of the text dataset was necessary in order to properly parse the descriptions. The text was cleaned by converting all words to lowercase, removing all punctuation, removing all words containing numbers, and removing any word that was one character or less. This description cleaning helped to reduce the amount of vocabulary that would need to be processed later in the LSTM layers. A mapping was created between each description and its corresponding identifier and then saved to the file descriptions.txt utilizing the script preparetextdata.py. The vocabulary size after this process was 8,763 in length.

Utilizing the loaddata.py script, the features.pkl and the descriptions.txt files generated from the photo and text preprocessing, respectively, are then loaded into memory. In this application, in order to generate a caption for a photo, every word of the caption would be generated one step at a time in sequential order. This caption would be wrapped by start and end tokens in the string which were named ”startseq” and ”endseq”. During the loading process of the clean text descriptions, these tokens were added for each description to make parsing easier later.

In the defmodel.py script, some encoding had to be done on the text data and a model was defined. The text data loaded from descriptions.txt had to be encoded into numbers in order to be fed into a model. In order to accomplish this, the Tokenizer class provided by Keras was utilized to generate a mapping between the words and unique integers. All of the photos are then loaded into memory. In order for this to be possible, a machine with an abundance of memory was needed which is why 44GB of RAM was chosen in the configuration of the Google Cloud Compute virtual machine. Another technique to avoid this is one known as progressive loading; however, for this application there was limited information at the time on utilizing this technique so the first method was chosen. In order to train the model, every description was split into separate words. The first word along with a photo is fed into the model and the next word is generated from that. Then the first two words will be fed into the model and then the next word is generated. Once the ”endseq” token is reached, all of the generated words were then concatenated into a complete caption. The result of creating the sequence will be a one-hot encoding for each word.

The model that was created is a recurrent neural network that utilizes LSTM layers so the gradients don’t vanish or explode. This model will take in input1 from the photo features in features.pkl and input2 from the clean text descriptions in descriptions.txt. Input1 is fed into a 50 percent dropout layer to prevent overfitting during training. Then the data is fed into a Dense layer so the output size is 256. Input2 is fed into an embedding layer which is a more efficient alternative to one hot encoding. That leads to a 50 percent dropout layer which is used for regularization so the model doesn’t overfit during training. This is then fed into an LSTM layer with 256 memory units. Then the output of the LSTM and the output of the Dense layer for the feature extractor are added together with an output size 256. This output is fed into a Dense 256 layer. This is then fed into a final softmax Dense layer used for classification.

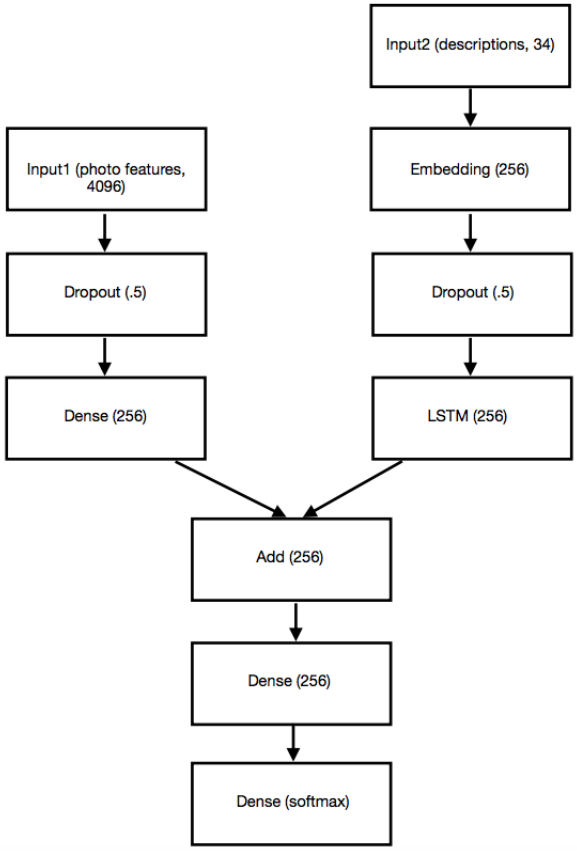

Fig. 1. RNN LSTM Model

After the model was defined, the adam optimizer was used for categorical cross entropy. The model was then fit with the training and validation data for 20 epochs where every model with decreasing loss was saved as checkpoint h5 files. This is the file where modifications were performed in an effort to try to reduce the loss in the network. The alterations that were performed include increasing the LSTM to 512 memory units, lowering the adam optimizer learning rate, changing the adam optimizer to the stochastic gradient descent (SGD) optimizer, and changing the dropout.

After the model was trained, the model was then evaluated using evalmodel.py. This file will use the best trained epoch with the lowest loss to generate descriptions for photos in the validation dataset. When the caption is generated, it will be evaluated with a BLEU score which represents the cost function. The bilingual evaluation understudy (BLEU) is a way to measure the accuracy of translated text when comparing to reference material. There are four BLEU scores

considered which are BLEU-1, BLEU-2, BLEU-3, and BLEU-4 depending on the cumulative n-grams and range from 0 to 1 where 1 is the highest accuracy. For reference, ranges for skillful models based on a 2017 paper are as follows: BLEU-1: 0.401 to 0.578, BLEU-2: 0.176 to 0.390, BLEU-3: 0.099 to 0.260, BLEU-4: 0.059 to 0.170. [2]

After evaluation of the model was performed, the model was utilized to create a tokenizer in the gencaptions.py file. This tokenizer.pkl file contained the encoding for the descriptions in the descriptions.txt file. In the gendescnewphoto.py file, this tokenizer.pkl file and the model with the lowest loss was loaded. The VGG-16 and InceptionV3 models were then used to extract the features from a new photo unseen to the network. Utilizing the trained h5 model and the tokenizer.pkl file, a description was generated for an image of a dog in some water.

III. Results

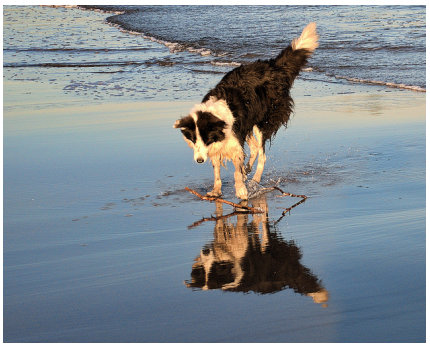

Fig. 2. Tested Image

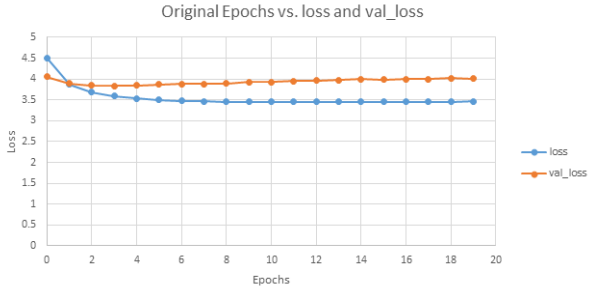

Fig. 3. Original Epochs vs. Loss LSTM 256, alpha = .001, VGG-16

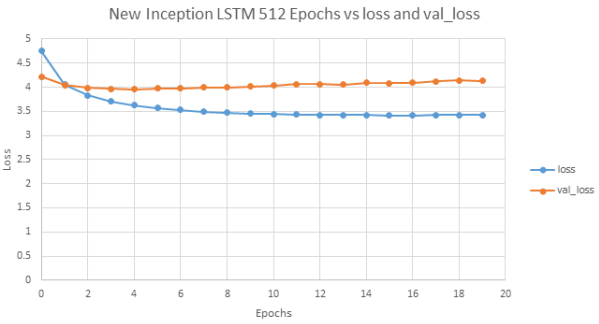

Fig. 4. New Inception Epochs vs. Loss LSTM 512, alpha = .001, InceptionV3

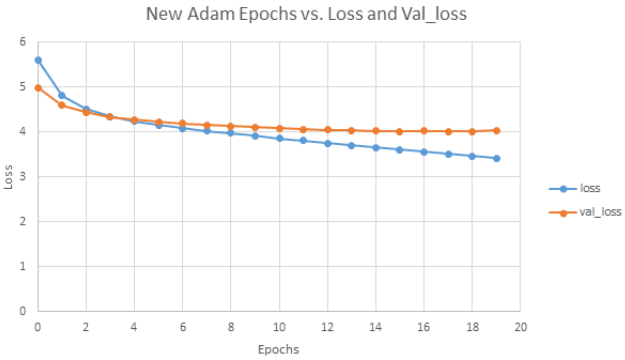

Fig. 5. New Epochs vs. Loss LSTM 512, alpha = .0001, InceptionV3

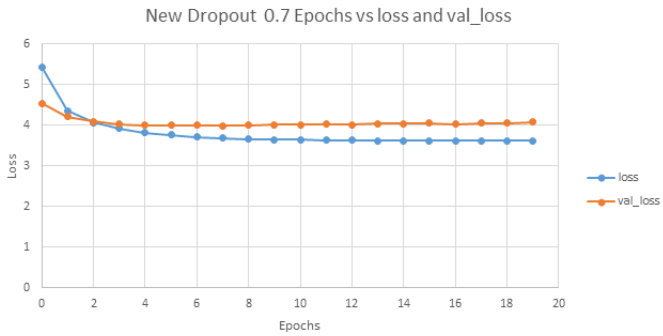

Fig. 6. New Dropout 0.7 Epochs vs. Loss LSTM 512, alpha = .001, InceptionV3

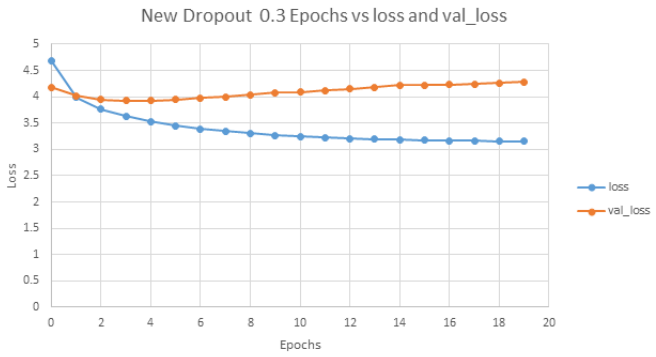

Fig. 7. New Dropout 0.3 Epochs vs. Loss LSTM 512, alpha = .001, InceptionV3

Original BLEU Scores	
BLEU-1	0.520737
BLEU-2	0.2586
BLEU-3	0.167577
BLEU-4	0.066349

Fig. 8. Original BLEU Scores

New Inception Scores	
BLEU-1	0.51068
BLEU-2	0.248021
BLEU-3	0.1694783
BLEU-4	0.066455

Fig. 9. New InceptionV3 LSTM 512 BLEU Scores

New Adam BLEU Scores	
BLEU-1	0.455759
BLEU-2	0.200561
BLEU-3	0.125791
BLEU-4	0.049523

Fig. 10. New Adam BLEU Scores

New Dropout 0.7 Scores	
BLEU-1	0.525182
BLEU-2	0.224304
BLEU-3	0.130761
BLEU-4	0.044643

Fig. 11. New Dropout 0.7 BLEU Scores

New Dropout 0.3 Scores	
BLEU-1	0.439
BLEU-2	0.16883
BLEU-3	0.096961
BLEU-4	0.027336

Fig. 12. New Dropout 0.3 BLEU Scores

Configurations	Generated Sequences
Original VGG-16 LSTM 256	startseq dog is running through the water endseq
New InceptionV3 LSTM 512	startseq dog is running through the water endseq
New Adam alpha = .0001, LSTM 512	startseq two dogs are running through the grass endseq
New Dropout 0.7 LSTM 512	startseq man in blue shirt is standing on the street endseq
New Dropout 0.3 LSTM 512	startseq man in red shirt is standing on the beach with his arms open endseq

Fig. 13. Configurations and Generated Sequences

IV. Discussion

The original results for the unmodified network led to a validation loss of around 4 and BLEU scores falling within a level that is in a defined intelligent network. However, during training the model would always stop learning after only 5 epochs when ran for 20. There were many different methods that were attempted to try to reduce the loss; however, for the most part the loss was always around 4. First VGG-16 was changed to InceptionV3 with LSTM 512. This led to almost identical loss and validation losses. Next, the adam optimizer was changed to stochastic gradient descent (SGD), LSTM unit size was 512, and InceptionV3 was used; however, this led to the validation loss holding at around 15.7787 and did not improve over 20 epochs. The adam optimizer learning rate was then lowered to 0.0001, LSTM unit size was 512, and InceptionV3 was used; however, this still led to the same level of validation loss by 20 epochs although it could be visually seen that it learned slower. The BLEU scores for this configuration were also consistently lower than the original and the generated text was not as accurate. In the next configuration, the dropout was changed to 0.7 with LSTM 512, learning rate for adam .001, and InceptionV3 used led to around the same validation loss of 4 but it did improve up to 8 epochs. The BLEU-1 score was actually higher than the original, but the other values were lower. The generated text, unfortunately was not even close to the image though. In the final configuration, the dropout was lowered to 0.3 with LSTM 512, learning rate for adam .001, and InceptionV3 was used. The network stopped learning after five epochs and the validation loss was again around 4. The BLEU scores were lower than that of the original and the generated sentence wasn't descriptive of the image. It seems that the dropout had the largest affect on how accurate in context the sentence would be. The biggest challenge for decreasing the validation loss was that there wasn't any real way to predict how altering the hyperparameters would affect the network without actually retraining the network. Training the system would take several hours for every configuration so this wasn't an ideal situation.

V. Conclusion

Overall, the methods that were applied to lower the loss of the original training were unsuccessful and there wasn't a significant difference between using VGG-16 and InceptionV3. However, these methods did offer insight into how the hyperparameters affected the network. One particular aspect that stood out was changing the dropout. The dropout controls

the regularization of the network and it definitely had the greatest affect on the generated text output. It was interesting that the BLEU scores for the new Dropout 0.7 were pretty close to the original yet the output was so different. This might be due to the LSTM being chosen to be 512 instead of 256, but this was a limitation of migrating from a VGG-16 network to the InceptionV3 network. In the original VGG-16 preparephotodata.py, there was a preprocessinput method imported from the VGG-16 network that set the output to 4096 in length. In the conversion to InceptionV3, this method had to be removed and it led to an output of 2048 which limited the network to have LSTM units of 512. In the Dropout 0.3 there was a larger gap between the training and validation loss which probably indicates overfitting. It might be useful to explore in the future fine tuning the dropout near 0.5 since it seems like it might have the best outcome on reducing validation loss. However, doing this would require a lot of computational power as accurately tuning this model requires retraining it every time.

VI. References

[1]https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/
[2]https://arxiv.org/abs/1703.09137