Automatic Image Captioning With Keras

Carlos Ordonez[1]
[1]Department of Computer Science, Marist College, Poughkeepsie, New York, USA

Abstract— *Image Captioning can have profound effects for the future of visually impaired members of society as well as improving image searches through search engines such as google and video feed monitoring. In order to automate this process I have implemented a merged neural network which take a feature vector from a CNN and inputs it in to an RNN to produce a predicted string of words which act as a relevant caption to the input image. This was implemented through the python keras functional API.*

Keywords: CNN, RNN, InceptionV3, GloVe, BLEU, Image Captioning

1. Introduction

The idea of a merged machine learning algorithm is to take two separate machine learning algorithms and have them work together for a common goal. In order to caption an image we need two separate networks namely a Convolutional Neural Network (CNN) and a Recurrent Neural Network (RNN) to be merged together in order to process an image and predict a proper caption for that image.[1] This is done through the keras Functional api as opposed to its Sequential API.

2. Data

To train our models we will be using an open source data set provided by Flickr, an image sharing web site, called Flickr30k.[2] Our data consist of two separate files and file types. We have a data set of unique images with fixed size (300,300). Each image has a unique image names which will act as the primary keys for our data. In addition to our images we will also have a text file of captions with the corresponding image names acting as the referential key for each caption. Each image has approximately 5 captions.

2.1 Data Cleaning

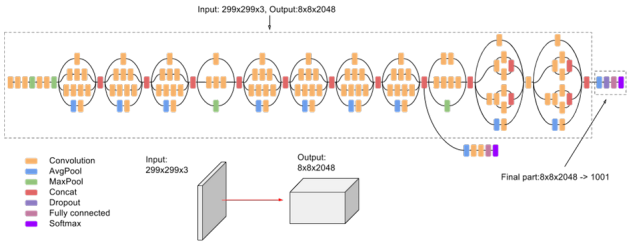
2.1.1 Captions

In order to work with the caption data set there is necessary cleaning that must be done. We must first drop the case of each letter to be lower case and then remove any punctuation or single letters such as an "s" after a word with its apostrophe removed or single character words like "i" or "a", as well as any words with numbers. The next step is our reason for the cleanup. We must build a dictionary of words from which our network can use to predict future image captions. We clean our captions to reduce any noise in our dictionary data set which can hinder our final predictions. We also need to remove any uncommon words from our final dictionary. To do this we filter out any words which occur less than 10 times. This works to reduce available words from which our RNN needs to choose from further reducing any noise present in our dictionary. Finally we need to place a strtseq and endseq at the beginning and end of each description respectively. This is to allow the model to learn where word fall in relation to the beginning and end of sentences. This allows the predictions to be in the form of a sentence and not just random words selected.

2.2 Preprocessing

2.2.1 Images

Due to the fact that our images will be an input in to our model and that input has to be in the form of a fixed length vector we must convert our images into a vector of fixed length 2048. Since our images are much larger than 2048 pixels we need to reduce our images to the proper size. If we just grab the first 2048 pixels we will get a sliver of the left side of the picture likewise if we randomly select 2048 pixels we will lose a lot of the feature in our image which are necessary for further classification. So instead we automatically generate features in the image which will be the most useful in further predictions. To do this we use the inception network from Google's image classification CNN.[3] Inception is a smaller CNN which does this feature generation for images.



It receives an image and produces a vector of length 2048 of important useful features and then uses those features to further classify its own images. We will not be using the classification abilities of inception instead we will use a technique called transfer learning where we use the information that inception learned to give our own network a head start in its own learning. This allows us to use the automatically generated rich feature vector created by inception and use that as input in to our own model instead of trying to learn the weights our self. To do this we remove the last softmax

layer of InceptionV3 which does the actual predictions in the inception network and simply use the output from its last dense layer as our X input in our (RNN)

2.2.2 Descriptions

Because our network is going to use the words in the dictionary as output and RNN do not work with strings or plain text the words need to be converted in to a usable form. For neural networks this is often a vector of set length. To do this we preform a one hot encoding on our data set and then feed this in to a pre-traind GloVe model.[4] This model converts each word in to a 200 length vector which can be used to extract relational information between words. These global vectors will be used in place of each word in our dictionary and our RNN will provide a vector of length 200 as output each time which will be converted from its vector format to word format to provide each concurrent word in the output caption.

3. Model

As mentioned earlier our model will be a merge of two separate models. Our CNN uses inceptionV3 which produces a feature image vector of length 2048 this will get fed as a rich feature input into our CNN which will produce an output of features. These feature get fed into our RNN model as our X value. This RNN will then use these X values to predict a GloVe of length 200. These two vectors, the image vector and word vector will get fed back into the RNN as input again and a second word vector will be produced this will repeat iteratively until a stop caption is predicted. The output of each RNN iteration will be single additional words built upon each other to create a relevant caption.

3.1 CNN

The architecture of our convolutional neural network will be as follows

inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

We have an initial shape of 2048 from google inceptionV3 this gets fed in to a dropout layer which randomly freezes 50 percent of our nodes. This layer gets fed into a dense layer of 256 nodes with a relu activation function. This is the ultimate output for our Cnn and the X value which gets fed into our RNN

3.2 RNN

Our Reccurant Neural Network Model archictecture is as follows:

inputs2 = Input(shape=(max$_l ength,)$)
$se1 = Embedding(vocab_s ize,$
$embeddingdim, maskzero = True)(inputs2)$
$se2 = Dropout(0.5)(se1)$
$se3 = LSTM(256)(se2)$

This model takes in the vectors of partial image captions which is the length of the longest caption in our training image description data set. These vectors then get input into an embedding node which preforms the GloVe embedding and produces an encoded version of the length 34 vector input. These then get fed in to a dropout node where we preform a dropout of 10 percent of the nodes. These vectors get fed into a LSTM node. This node gives our network the ability to retain more of a memory of prior weights for a long time without having issues of losing memory of prior states due to new states or weights taking over. This layer has 256 nodes so its output will be a 256 length vector as well. Each iteration takes in the prior prediction of the network and builds upon it so captions get created step wise

Table 1: An example of Caption Prediction

Item	
Itteration	Caption Prediction
1	Man
2	Man standing
3	Man standing in
4	Man standing in front
5	Man standing in front of
6	Man standing in front of tree

3.3 Merge

The output to our RNN and CNN are both the same size of 256 length vector.Because of this we are able to add our two vectors into one this is where we merge the two models we do so as follows:

decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab$_s ize, activation = '$
$softmax')(decoder2)$

Here we add both vectors together using vector addition and feed that into a dense layer of 256 nodes which will feed in to a second dense layer this time with as many nodes as we have words in our dictionary and a relu activation. This is to produce a final output vector giving us a prediction for how likely each word is to come next in the caption. We use an activation function of soft max on this layer.

4. Generator

Due to the size of our dataset if we simply feed in our full dataset to our model all at once it will overload the

working memory of the model. This is due to the large size of the image vector plue the size of the word vector which is a 200 length vector for each available word within the max length of the captions. This leads to a exponential increase in array size which if managed to get into working memory would slow down our model training considerably and make convergence take a long time. When Stochastic Gradient Decent is used for an optimization such as adam not all the data points are used to generate the gradient only the small points around the current location of the model. Due to this we don't have to load all of our data all at once to train our model we can incrementally supply data continually maintaining the current state and as we progress down our optimization path we just continually feed more relevant data. This is done through a generator which is built on an infinite loop to continually encode and build very small versions of our data set which it can then feed into the network to allow for the gradient decent to occur and more images are fed in again a few at a time so as not to overload the network with data.[5]

5. Hyper-parameters

When building this model various hyper parameters were tried initially on a training set of size 500 in order to reduce the training time. I then applied these model states to the full 27k data set.

To evaluate how well a model was working at predicting our captions I used a standard BLEU score. This system uses the given caption and compares them to the predicted captions and gives a score on how well they match where the exact same sentence receives a score of 1 and a completely different sentences receive a score of 0. This system has flaws due to the fact that we have a small set of training captions per image and these are what each sentence is scored upon. So while some captions have a BLEU score which is relatively low the predicted caption though is contextually relevant and quite good sometimes so these values do seem to be unrepresentative of how well the model is actually preforming and instead just give us a relative value for which we can compare separate model parameters to evaluate how well each network is preforming.

I initially trained for 20 epochs using a batch size of 3 and a learning rate equal to the length of our description set divided by the batch size to get an equal stepping for each batch.Then I preformed an optimization step where I decreased the epochs to 10 and decreased the learning rate way down to .0001 as we need to slow down as we get closer and closer to convergence so we don't pass over it. I also increased the batch size to 6 to allow less time for memory loss over all the batches. This set up with each initial dropout set to .5 and trained on 500 pictures got a BLEU score of .44 when the number of pictures was increased to the full 27k set of images in data set this score was raised to .54 which is slightly better as expected due to a larger amount of data being trained on. I then changed my batch size to 100 for the first 20 epochs with the same training rate and increased the batch size for the optimization set of epochs to 150. This seemed to help out my model as it raised the BLEU score when training on the 500 length data set to .51 which is at least 5 percent better when this model was expanded to the full 27k data set the blue score was increased to the highest value of .57. This is the ultimate model weights set up that was used.

6. Results

Table 2: Optimal Model Results with varying Training Set Size

Item	
Training set size	Test Set BLEU Score
500	.45
6000	0.5108587361155279
27000	0.5742436309710937

Below are some results of the predictions made for input images from the test set

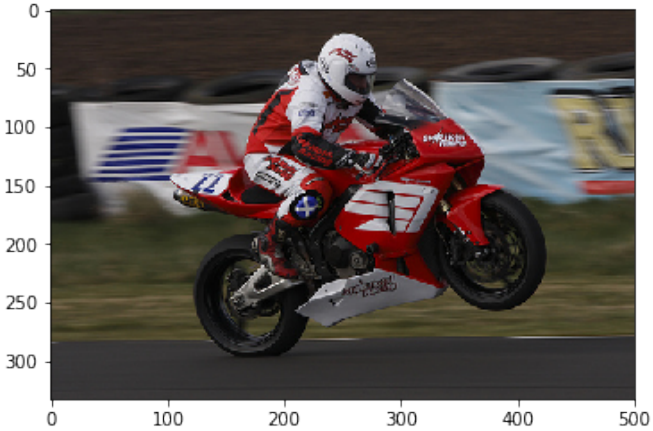
Fig. 1: 2513260012.jpg
Greedy: two dogs are running through the snow
Real Captions: black dog is running after white dog in the snow
BLEU: 0.871846861591009

Fig. 2: 2537806838.jpg

Greedy: man in blue shirt and jeans is cleaning up some pipes
Real Captions: men in orange vests and white hard hats stand on scaffolds working
BLEU: 0.3261271338801469

Fig. 3: 2496236371.jpg

Greedy: motorcyclist is driving on track
Real Captions: motorcycle rider doing wheelie
BLEU Score: 0.8579992277790618



References

[1] Y. Li, Two layers LSTM with attention for multi-choice question answering in exams, IOP Conference Series: Materials Science and Engineering, vol. 323, p. 012023, 2018.
[2] http://shannon.cs.illinois.edu/DenotationGraph/
[3] SJ. Yim, D. Joo, J. Bae, and J. Kim, A Gift from Knowledge Distillation: Fast Optimization, Network Minimization and Transfer Learning, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017.
[4] J. Pennington, R. Socher, and C. Manning, Glove: Global Vectors for Word Representation, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014.
[5] B. Korel, Automated software test data generation, IEEE Transactions on Software Engineering, vol. 16, no. 8, pp. 870âĂŞ879, 1990.