Neural Image-Text Similarity for Image Selection

Matt Einhorn
Cornell University
me263@cornell.edu

Yen Huang
Cornell University
yh299@cornell.edu

James Stoyell
Cornell University
jms852@cornell.edu

Abstract

In this paper we explore the task of scoring the fit of an image to a caption. This task has many applications, particularly image search, but is also difficult because it combines two challenging fields in artificial intelligence: natural language understanding and image understanding. We propose a two-part model that uses an image encoder and sentence encoder to encode the image and caption into the same learned embedding space, then compute the cosine similarity between the two vectors to produce a final similarity score. The separate encoders allow for pre-computation of image or caption embeddings as applicable, allowing for very fast inference for applications like image search. We train our model on the 2017 COCO Dataset (Lin et al., 2014) and show promising results in experimentation.

1 Introduction

Image-text matching is the task of scoring how well a short piece of text describes an image. The score should be high if the caption describes the most salient portions of the image, and low if the caption includes information not in the image or describes things not salient to the image. This requires an understanding of both the vocabulary and semantics of a language, e.g. the caption "the dog is in the outdoor play" describes a different image than "the dog plays in the outdoors".

Thus, this task requires both natural language understanding and semantic image understanding, both difficult and sought after tasks. Recent trends in computer vision have been towards transfer learning, retraining an existing model such as ResNet or DenseNet for a different task. State of the art natural language systems mostly involve recurrent neural networks with a memory cell like an LSTM or GRU (Cho et al., 2014), though recent state of the art models have eschewed recurrent networks in favor of attention (Vaswani et al., 2017).

A closely related and well explored task is image captioning - generating a caption to describe an image. This is an inherently more difficult task due to the large possible output space, but shares many key similarities in that successful networks must possess a semantic understanding of the image space and language space. Significant advancements have also come in this space due to attention (Xu et al., 2015), which makes sense when pre-computation is impossible due to the one-to-one nature of the task of captioning. For a task like image search, pre-computation is extremely useful as recomputing an embedding for every image in the search space quickly becomes intractable as the number of images increases. Thus, a model that allows for pre-computation is desirable.

In this paper we introduce an approach to image-text similarity scoring that uses entirely separable encoders for encoding both the image and text. Our image encoder utilizes the DenseNet architecture, pre-trained on ImageNet with output layers to be retrained to fit the size of semantics of the embedding space.

We hypothesize that the separable image and text encoders will be somewhat less accurate but much faster than similar but connected models seen in literature. We also hypothesize that the recurrent text encoder will outperform that bag of words encoder.

We experiment with different text encoding models, one a recurrent network with LSTM state and a simpler bag of words encoder. We then discuss the relative performance of the models and discuss the potential applications of the work.

2 Related Work

There has been novel research in detecting semantic similarity between images and phrases to tackle several types of task. One task is called region-phrase matching, which is matching a phrase to a particular region in the image. Image-sentence matching is defined as given an image, return the relevant sentences. Wang et. al created two versions of a two-branch neural networks, in which one is an embedding network and the other is a similarity network. The embedding network learns a shared learning space, in which images and corresponding texts lie close to each other. The similarity network frames the problem as a binary classification problem by doing an element-wise product between the two branches and is trained to compute a similarity score between the image and the text. Their experiments indicated that similarity network performed well in region-phrase matching but not for image-sentence retrieval. Embedding network performed well in both region-phrase matching and image-sentence retrieval (Wang et al., 2017). Based on this finding, we model our architecture similar to the embedding layer.

It has been shown that CCA (Canonical Correlation Analysis) creates a powerful baseline model for image-text embedding (Klein et al., 2014). Klein et al. demonstrated that a CCA model can outperform more complex models. However, the downside to utilizing CCA is the memory cost. Based on this finding, we did not utilize CCA, as we did not have the resources to perform such heavy computational task.

3 Data

The main dataset used for this task was the Coco 2017 dataset (Lin et al., 2014). This dataset consists of images and approximately five captions per image, where the captions are short descriptions of the most salient parts of the image. The images contain objects from a set of 80 possibilities. The possibilities are a number of everyday animals and objects. The training set contains 118287 images and 591753 captions with an average caption length of 11.34 words. The validation set contains 5000 images and 25014 captions with an average caption length of 11.32 words.

We also briefly trained on a dataset with longer paragraphs (Krause et al., 2017). The dataset contains 19,561 images from the Visual Genome dataset (Krishna et al., 2016) and each image contains one paragraph. The training/val/test sets contains 14,575/2487/2489 images, respectively.

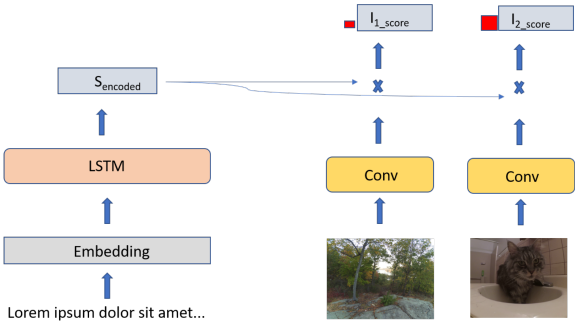

Figure 1: **Two-Branch Neural Network** — We encode the image and text separately, and we compute the cosine similarity between the encoded outputs.

Captions are fed into the text encodings using the 300 dimensional pre-trained GloVe word vectors (Pennington et al., 2014). The GloVe vector set consists of 400,000 words. The caption training set contains 27,356 unique words, 4,984 of which are out of vocabulary, whereas the caption validation set contains 7227 unique words, 346 of which are out of vocabulary.

During training and validation, each time a positive example is needed, an image is chosen and a random caption that corresponds to that image is chosen to accompany it. This means that a given image can be given a perfect similarity score to more than one caption. This matches the task; there is no single caption that perfectly describes an image, as many could describe it equally well.

4 Methods

4.1 Architecture

The goal of our system is to find the image, among a set of images, that most closely represents a text string. We use neural networks to separately embed the string and images, and we represent them each by a fixed length vector of size 1000. We then compute the cosine similarity between the embedded string and each embedded image and finally we return the image with the highest string similarity (Figure 1).

4.1.1 Text Embedding

For the text encoding, we used a bag of words or an LSTM network as follows. Given a paragraph, we embed each word into a 300 length vector using the pre-trained GloVe.6B embedding (Pen-

nington et al., 2014). For the bag of words model, we take a max pooling over the embedded words in the sentence and then pass the 300 length vector into a 1000 unit single layer neural network. For the LSTM model, we pass the embedded sentence into a bi-LSTM with various number of hidden units, and 3 stacks. The bi-LSTM output is concatenated and passed to 1000 unit single layer neural network. For each case, the output of the final neural network is the encode text.

4.1.2 Image Embedding

We used DenseNet (Huang et al., 2017) pretrained on ImageNet (Vision, 2018) to reduce the training time. Specifically, we used DenseNet-121, which is the smallest DenseNet trained on ImageNet. We removed the final softmax layer from DenseNet such that our encoding for each image is the DenseNet output of the final 1000 unit fully connected layer before the softmax layer. Before passing the image to the network, we randomly resize and crop the image to 224. During training, we augment the data by adding a random horizontal flip.

4.1.3 Similarity

Given an encoded text description and a set of encoded images, we compute the cosine similarity between the text and all images. During training, each batch contains a mix of positive and negative examples. The positive examples are pairs of a description and its associated image. Negative examples instead use a random image not associated with the description. The goal is for the loss, using the cosine similarity, to move positive example image-description pairs close, and negative examples apart.

At test time, we compute the similarity between an encoded description and all the encoded images in the batch or set. Ideally, the image associated with the description will have the highest similarity. We measure the goodness of the training by looking at the index, or rank of the associated image in the set of images sorted by decreasing similarity. Ideally, the associated image will have rank one. We compute the mean top-1, top-3, top-5, and top-10 ranks, where top-n indicates the percent positive examples with descriptions whose associated images were in the top n images when sorted by decreasing similarity with the description.

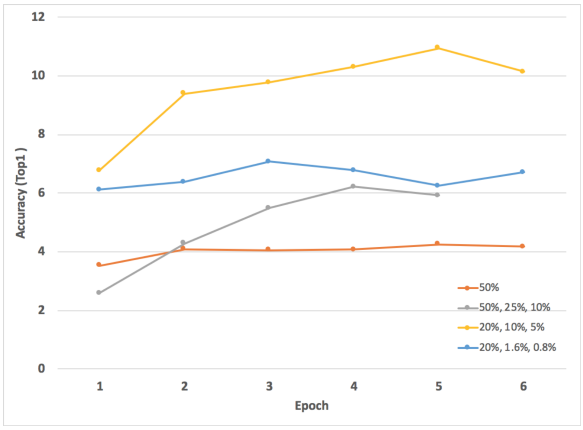

Figure 2: **% positive examples in each batch at each epoch** — We trained our LSTM with different variations of positive and negative examples over epochs split over the range: Epoch 1, Epoch 2, Epoch ≥ 3. The percent in the legend, indicates the percent positive examples for the corresponding epoch, where epochs are indicated by comma separation

4.1.4 Implementation Details

All our models were implemented in Pytorch (Paszke et al., 2017) and Pytorch Vision (Vision, 2018). We trained using SGD with momentum of 0.9 and weight decay of 1e-4. We trained the models on a Quadro M4000 and a GTX Titan. We used the full Coco training set for training. For testing, we split the test set into 2,500 for validation and 2,500 for test.

5 Experiments

5.1 Percent Positive Batch Examples

We first focused on positive batch examples in order to calibrate our model so that our LSTM model learns what is a good and bad image for a given sentence. We experimented with reducing the number of positive examples per batch (Figure 2). Our baseline LSTM was 50% positive and 50% negative for every epoch from 1 to 6. We then fine-tuned it to several variations of batch positives and batch negatives. The variations were (in terms of batch positives):

1. 50% for epoch 1, 25% for epoch 2, 10% for epoch 3-6
2. 20% for epoch 1, 10% for epoch 2, 5% for epoch 3-6
3. 20% for epoch 1, 1.6% for epoch 2, 0.8% for epoch 3-6

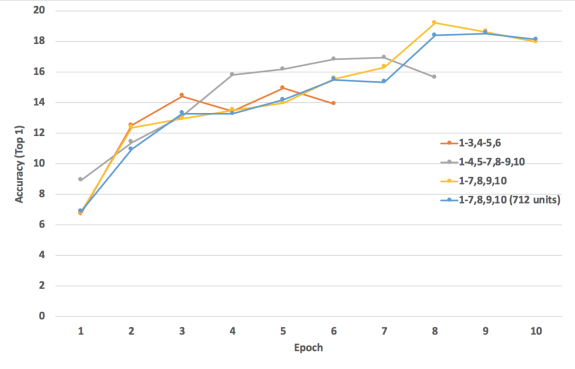

Figure 3: **Learning Rate Schedule over Epochs** — We trained our LSTM text encoder with different schedules when to drop the learning rate. Commas in the legend indicate the epoch where we dropped the LR by a factor of 10. The one with 712 units, had 712 hidden units in the LSTM rather than 512.

To evaluate the performance of these variations, we used the top-1 ranking described earlier. From Figure 2, the 3 variations outperformed the baseline batch positive of 50% after 6 epochs. The optimal variation of batch positive was 20% for epoch 1, 10% for epoch 2, 5% for epoch 3-6. The results suggest that too many positive examples in a batch mis-calibrates the model with the final task that has only one associated image for a description.

5.2 Learning Rate Over Epochs

For the LSTM, we experimented with optimizing the epochs when we reduced the learning rate in order to ensure that a local minimum in the loss function is not missed. We dropped the learning rate from $1e^{-1}$ to $1e^{-2}$ to $1e^{-3}$ to $1e^{-4}$ during scheduled epochs. We experimented with different variations for when each of these learning rates should be used.

The variations were:

1. $1e^{-1}$ for epoch 1-3, $1e^{-2}$ for epoch 4-5, $1e^{-3}$ for epoch 6
2. $1e^{-1}$ for epoch 1-4, $1e^{-2}$ for epoch 5-7, $1e^{-3}$ for epoch 8-9, $1e^{-4}$ for epoch 10
3. $1e^{-1}$ for epoch 1-7, $1e^{-2}$ for epoch 8, $1e^{-3}$ for epoch 9, $1e^{-4}$ for epoch 10

From Figure 3, the best performance was $1e^{-1}$ for epoch 1-7, $1e^{-2}$ for epoch 8, $1e^{-3}$ for epoch 9, $1e^{-4}$ for epoch 10. This performance indicates that more epochs are needed before decreasing the first learning rate.

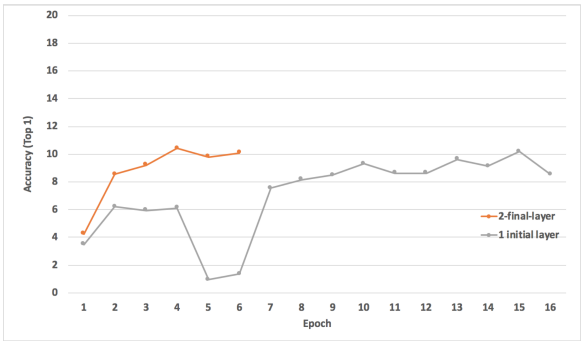

Figure 4: **LSTM Modification Architecture** — We experimented with adding 2 final fully connected layers after the LSTM or an initial layer before the LSTM.

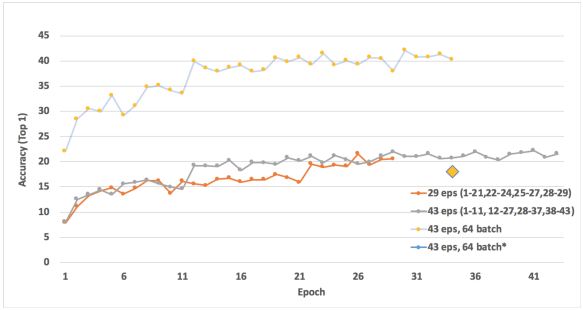

Figure 5: **Final LSTM Model top-1 rank** — We trained our LSTM with longer epochs and a more spread out schedule of dropping learning rate. We also trained the model with a batch size of 64 rather than 256. "43 eps, 64 batch*" is the model trained on batch size of 64 and tested with a batch size of 256.

The original number of LSTM hidden units was 512 when running these experiments. We also increased the hidden units to 712 when running the most optimal learning rate schedule, and it did approximately the same as the 512 hidden units.

5.3 LSTM Architecture Modification

We experimented with adding a second fully connected layer after the LSTM as well as an initial fully connected layer before the LSTM. From Figure 4, adding another final layer or an initial layer was not an effective optimization because it significantly reduced performance compared to Figure 3.

5.4 Best LSTM Results

For our best models, we trained for 29 and 43 epochs, respectively (Figure 5). For the 29 epochs model, we spent more time at a higher learning rate, but looking at the 43 epochs model, reducing the time spent at a higher learning rate improved the model more quickly. Therefore, we pick the

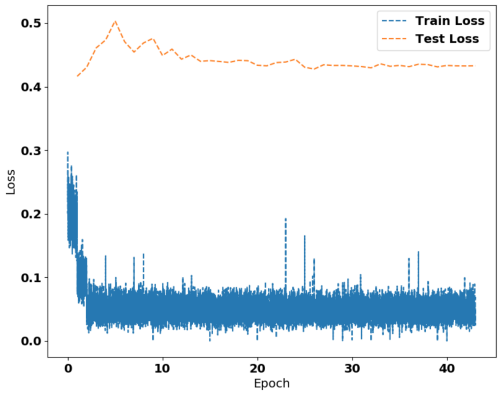

Figure 6: **Final LSTM loss** — Train and test loss for the best LSTM model.

Figure 7: **Rankings** — A histogram of the indices of the image associated with a description for our best and other models. Values close to 1 is desired, indicating the associated image was very similar to the original description.

model trained for 43 epochs with a more even LR schedule as our best model.

We also experimented with a batch size of 64, as opposed to our standard batch size of 256. Figure 5 shows that on a batch size of 64, performance improved. However, this is expected because by random chance, a smaller batch size is expected to improve performance because being ranked 1 is now likely by 1/64 vs 1/256. The "43 eps, 64 batch*" line in Figure 5 shows the single data point when testing the 64 batch sized trained model on a 256 batch size.

Indeed, looking at that data point, the error is worse than the one trained one a 256 sized batch, indicating that the apparent performance increase was indeed due to random chance. This also suggests that training on larger batch size improves performance.

Following is some further analysis of the best LSTM model. Figure 6 shows the training and test loss. Although training loss quickly converges, test loss converges more slowly. Further, there's a interesting increase in loss for the test set in the beginning of training, even though the rank improves. This indicates that perhaps there's re-training from some initial pre-trained minima due to ImageNet, which ultimately allows us to improve even further.

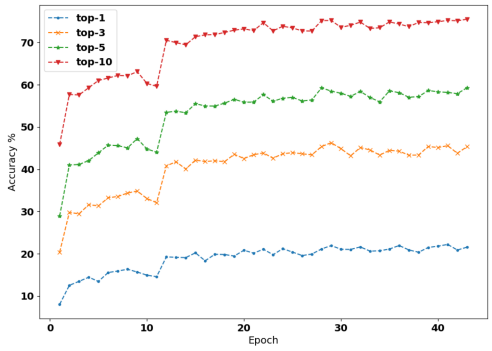

Figure 8: **Mean Accuracy** — Accuracy for top-1, top-3, top-5, and top-10 for the best LSTM model.

Figure 9: **Example Image output** — From our best LSTM model, these were the most similar images associated with the given text "the parents cried with happiness when their child started school". Also shows the histogram of the all the similarity values for the text for all the validation and test images.

Figure 7 shows the histogram of the rank of positive examples. As expected, our best model ranks the specific image associated with a description close to 1, indicating that most of the test pairs listed the correct associated image among its top results in terms of similarity.

Finally, Figure 8 shows the mean accuracy for the top-1, top-3, top-5, and top-10 ranks as described in the methods. It shows that out of a batch of 256, the correct image was in the top-1 22.21% of the time, in the top-3 46.27%, in the top-5 59.29%, and in the top-10 75.52% of the time.

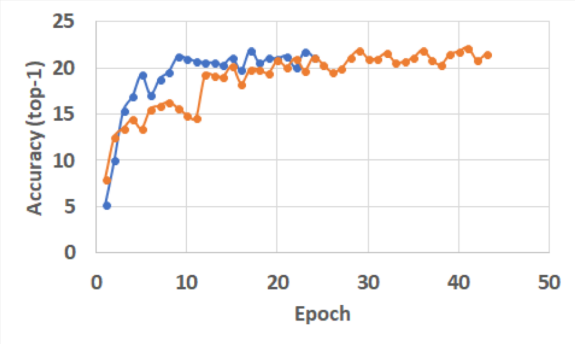

Figure 10: **Bag of Words Encoder vs the best LSTM Model** — Top-1 error. The blue line is the BOW model, and the orange line is the best LSTM model.

5.5 Sample Output

We also looked at the highest ranked images in the validation and test set for sample descriptions passed to the best LSTM model. Figure 9 shows the highest ranked images for the input sentence "the parents cried with happiness when their child started school". It also shows a histogram of the similarity values for all the images in the validation and test set for the given description.

It is clear that non of the images truly express the sentence - although there's some relationship between the the description and the images. After running with several inputs of text, our LSTM model rarely outputs a similarity score more than 0.2. This is likely because the COCO dataset has 80 object categories and 91 stuff categories. As a result, the categories are very limited and is not likely to be able to exhibit complex situations.

5.6 Baseline BOW

We wanted to compare our LSTM model to a more basic bag of words (BOW) model. We trained the BOW encoder similarly to the LSTM model, except for the reduction in the number of epochs due to the reduced BOW model size (no LSTM layers). From Figure 10, it can be observed that the BOW encoder does as well as the LSTM encoder. This may be because there are only so few objects within a sentence and in an image. As a result, it is easy to detect a particular object within an image using a simple BOW model.

5.7 Paragraph Image Matching

We decided to see if the LSTM model would do better on a longer paragraph dataset. We used our best LSTM model trained on COCO and then fine-tuned for 10 epochs on a larger paragraph dataset (Krause et al., 2017). After testing on this dataset, there was no significant improvement in top-K accuracy. Given that our LSTM does not improve with longer paragraphs, it suggests to us that our simple model has already exhausted what it can learn from the data. We need more complex techniques to further improve performance (i.e adding POS tags and including object relations within an image).

6 Conclusion

Our results suggest that our hypothesis is not correct because the bag of words encoder did just as well as the LSTM encoder. Despite this, it is important to note that the separable image and text encoders allow us to pre-encode the images and quickly compare them to any encoded sentence.

Through our experiments we also discovered that for this task, an optimal mix of batch positive and negative examples are important, and that dropping the number of positive examples in the later epochs improves performance. In addition, we found that an optimized learning rate schedule is important for performance, and that spending a lot of time at higher learning rates is not helpful for performance.

7 Future Work

For future work, we would focus on including a POS tagging in order to capture the object relations within a sentence. More importantly, we would focus on including relationships and regions within an image.

We would also consider modifying our image architecture network to an image encoder decoder network so that the hidden layers can learn the rich information of the image and then enforce structure on the hidden layer such that we could use it when computing similarity to an encoded description..

8 Contributions

James implemented the word embedding model and the bag of words encoder, and helped with the implementation of the LSTM encoder. He also integrated Tensorboard and set up the initial demo. For the paper, he wrote the abstract, introduction, and data sections.

Yen implemented the lstm encoder. Yen worked on creating the final presentation slide and generated the graphs. For the final paper, Yen worked

on Related Work, Experiments, Conclusion, and Future Work.

Matt implemented the image network (DensetNet) as well as the python training skeleton code and some optimizations for the demo. Matt also trained the networks. For the final paper, Matt worked on creating the methods and the helped on the experiments and proof-reading.

References

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Benjamin Klein, Guy Lev, Gil Sadeh, and Lior Wolf. 2014. Fisher vectors derived from hybrid gaussian-laplacian mixture models for image annotation. *CoRR*, abs/1411.7399.

Jonathan Krause, Justin Johnson, Ranjay Krishna, and Li Fei-Fei. 2017. A hierarchical approach for generating descriptive image paragraphs. In *Computer Vision and Patterm Recognition (CVPR)*.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. 2016. Visual genome: Connecting language and vision using crowdsourced dense image annotations.

Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In *NIPS-W*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

PyTorch Vision. 2018. Pytorch vision. https://github.com/pytorch/vision.

Liwei Wang, Yin Li, and Svetlana Lazebnik. 2017. Learning two-branch neural networks for image-text matching tasks. *CoRR*, abs/1704.03470.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044.