Emoji-Language Image Captioning using Convolutional Neural Networks

Hernandez, Rayne
Department of Statistics
Stanford University
rayne@stanford.edu

Knight, Ian Scott
Symbolic Systems
Stanford University
isknight@stanford.edu

Abstract

Momentous progress has been made in recent years on deep learning techniques of image captioning. While effective generative models already exist for generating text descriptions of images, less attention has been given to models of more immediately symbolic description. Given that emojis are the de facto standard for social media expression, understanding how emojis are used in relation to visual information has the potential to lead to a more nuanced understanding of their use in the online world. In this work, we present a model for generating emoji-language image captioning by means of (1) an emoji2vec embedding of emojis into the same vector space as word2vec word represenations and (2) a convolutional neural network architecture for image classification.

1 Introduction

The rise of social media platforms in the past 5 years has caused a spectacular change in the way people communicate online. Emojis emerged as a compact means of expressing emotional context in everyday parlance. Considered a de-facto aspect of online English language, emojis have nevertheless received considerably little attention as an object of study within the fields of NLP and image captioning. Existing at the intersection of meaning and representation, emojis provide a linguistically universal means of understanding the emotional context of both text and image. In this sense, a natural language processing task that lends itself easily to emojis is image captioning. Specifically our problem was to capture the meaning of a given image using the emojis that best describe it. Solving this problem accomplishes two things. First of all, it opens up avenues for making visual media more universally accessible. Combined with a language model, emoji image captioning has ability to assist online users of many languages to understand online content found primarily in English. Second of all, emoji captioning produces a dense embedding of textual image captions to emojis. This opens ups the possibility of text to emoji translation, or emoji augmentation of textual information. Considering that emoji usage is the rule and not the exception when it comes to interaction on social media platforms, a means of augmenting textual responses/descriptions with emojis or predicting the emotional reaction to a text string or image as encoded by a series of emojis is paramount in understanding online language usage. In both of these senses, NLP stands to benefit greatly from a closer study of emojis.

2 Related Work

The most pertinent work to our project exists in two realms. The first of these realms is current work done on emoji embedding of textual data. Vastly popular textual embeddings exist for vector representations of textual information. These include word2vec [4] and GloVe [6]. Less studied are methods for embedding vector representations of text strings that include emojis as vectors. Barbieri in [1] provides a method based on the skip gram-model, used to embed word2vec vectors, to embed emojis. They trained a skip-gram model on Twitter text posts which included emojis in order to train dense word vectors. The embeddings produced were then tested by removing emojis from original tweets and attempting to predict these emojis as one would a normal word. These result were promising but Eisner in [2] provides an improved method called emoji2vec which attempts to produce emoji embeddings within the same dimensional space as typical word2vec vectors. Training on emoji definitions rather than training examples, emoji2vec attempts to embed emojis directly within the word2vec space. The results are accurate prediction of emojis in text with the original emojis removed. We utilize these emoji2vec embeddings in our project as a mediating process between typical image text captions and the space of embedded emojis that we can train on.

The second of these realms are efforts made in the last 4 years to engineer effective image captioning systems using Deep Learning techniques. We follow the successes of convolutional neural networks (CNN) for image classification and recurrent neural networks (RNN) for end to end machine translation of text of one language to another. The architecture put forward in [7] makes use of an encoder/decoder paradigm, whereby an encoder first processes the image into a dense representation that extracts key features from the visual information, and a decoder processes this dense representation to generate a text string describing these key features. In this context a CNN is chosen to encode the image because CNNs are very good at extracting hierarchal information from an image using stacked local convolutions and embedding this information into a dense representation vector as the final hidden layer of the CNN. In contrast, an RNN architecture, or more effectively an LSTM architecture, is chosen for a decoder since RNNs are very good at generating long strings of output of variable length given a dense embedding as input. LSTM is generally preferable to RNN since LSTMs avoid the problem of vanishing and exploding gradients, whereby the next word generated is only conditioned by the words generated locally. However, where LSTM has been found successful in text to text translation, LSTM is also successful in image to text translation provided the image is input in the form of a dense embedding produced by the CNN. The combination of developments in these two research areas provide us with the means for effective emoji image captioning. Our model differs from the above architecture in that our model has no need to generating emoji strings in response to images but rather attempts to provide a fixed number of emoji responses to images. In this sense the decoder aspect of the model is superfluous, and we only need an effective image classifier where the classes are emojis. Emoji2vec becomes important because to date there are no substantial training sets containing only emoji labels for images, and therefore we need to preprocess traditional captioned images to contain emoji labels.

3 Data

Our dataset consists of the 2017 COCO dataset, which includes images (118K train, 5K validation, 20K test) and their corresponding English-language captions (5 per image) for a total of approximately 715K (image + caption) data points.

Since our ultimate goal is to produce semantically accurate emojis for an image given only the image, and our raw dataset contains only images with English text captions, we transformed the text captions into emoji class representations. For the sake of simplicity, we decided to classify each image with exactly 5 emoji labels.

To accomplish preprocessing we leverage emoji2vec in order to translate text caption to emojis classes for each image such that the dataset became a set of 715K (image + emoji caption) data points. The images are in .jpg format.

4 Methods

4.1 Preprocessing

4.1.1 Text-to-Emoji Translation

Since our ultimate goal is to produce semantically relevant emojis for an image given only the image, and our raw dataset contains only images with English text captions, we needed to devise a method for translating English text captions into emojis in a semantically accurate manner. For the sake of simplicity, we decided to classify each image with exactly 5 emoji labels.

For this task, we made use of emoji2vec, a vector space model of emojis (instead of words) within the same vector space as the famous word2vec model. This vector space model comes from work done by Eisner et al. [2, and it provided us an extremely convenient and semantically accurate tool for relating English words to emoji characters. By obtaining the emojis with the smallest cosine distances from the words in text captions, we invented an algorithm for translating English text into emojis class labels. The algorithm is as follows.

1. Part-of-speech tag each of the 5 captions for a given image (e.g. noun, adjective, verb, etc.)

2. Remove certain kinds of part-of-speech that dont carry valuable information. After some experimentation, we found the best results to follow from only keeping nouns, adjectives, and verbs. Moreover, we found the best results to follow from weighting nouns 3x, adjectives 2x, and verbs 1x.

3. Vectorize the remaining words in each caption using word2vec.

4. For every word remaining in the caption text, obtain a certain number of emojis that are closest to the word in word2vec space. After some experimentation, we found the best results to follow obtaining the 3 nearest emojis for each word. Add the cosine similarity scores of these emojis to a score list. Once all 5 captions linked to a given image have been processed, output the 5 emojis with the highest scores. These are the final emojis assigned to the image.

4.1.2 Emoji Class Elimination

Initial analysis of our resultant emoji captions found that there were 1049 different emojis represented. However, of these only 421 emojis had at least 100 instances while 628 classes had fewer than 100 instances. Since there were 118K images, we determined that fewer than 100 instances was not substantial enough to warrant class representation within our model. Therefore, we removed all instances of said 628 classes with fewer than 100 instances.

4.1.3 Fixing Number of Labels

Due to the class elimination process, we were left with many images with fewer than 5 unique emojis. We corrected this by simply randomly cloning enough of the remaining emojis for that image until it numbered 5 again.

4.1.4 Class Condensation

Due to the fact that many emojis carry essentially the same meaning (e.g. ", ", " all mean car), the models understanding of classes would be improved by condensing these emojis into the same class, which is designated by only one emoji (just "). We therefore investigated which of the remaining 421 emojis shared meanings and manually placed them into groups. This led to the creation of 69 groups, each containing at least two emojis. The instances of each groups emojis were then condensed to just one of that groups emojis. This led to a reduction of classes for a total of 208 emoji classes.

4.1.5 Image Resizing

Due to the irregularity of dimensionality of images in our data set, it was necessary to resize every image to conform to the same size and dimensions. We therefore resized every image to 256x256 pixels. This

4.2 Training

4.2.1 Network Architecture

Our training algorithm centers on using a convolutional neural network (CNN) to train a multi-class, multi-label image classifier. This approach is based on successful existing implementations [3] [7] of neural network-based approaches to the task of image captioning. As for software, we chose to use PyTorch to build the CNN.

The bulk of the network is made up of stacked instances of specialized sequential layers, where each sequential layer contains (in order) a 2D-convolutional layer, a batch normalization layer, a ReLU activation layer, and a max pooling layer (with the exception of the final sequential layer of the stack, which uses an average pooling layer instead of max pooling). Moreover, the final convolutional layer uses dropout of 0.5.

Following this main convolutional portion of the network, there is a 3-layer fully connected neural network where the last layer contains the same number of nodes as the number of emoji classes. We do not need to opt for a more complex structure for multi-label classification since our desired label quantity is fixed at five. That is, for each input image we simply output the five emoji classes with the highest logits.

4.2.2 Loss and Optimization

For loss calculation, we simply add the criterion of the cross-entropy loss for each of the five true emoji labels of each image. It is important to note that the true labels for each image can have more than one instance of the same emoji class(es); it is even possible for all five true labels to be instances of the same emoji class.

For optimization, we used the Adam optimizer. After trying various learning rates, a value of 0.0001 was found to work best.

5 Experiments

The primary variable of experimentation for this project involved variation in the architecture of the convolutional neural network. We implemented and test four different architectures, each differing in the depth and number of outputs of each convolutional layer. These are the results for 40 epochs, batch size 100, and learning rate of 0.0001.

5.1 2-convolutional layer

Recall: 0.091

only a minority of emoji classes were present in output, mostly very elements of images in the training set (e.g. people, airplanes, cars, etc.). Very inaccurate output overall.

5.2 5-convolutional layer

Recall: 0.394

At this point, most emoji classes started being present in output, but they were fairly inaccurate and seemed to come at the expense of correctly labeling the more common emoji classes previously mentioned (e.g. people, airplanes, cars, etc.).

5.3 7-convolutional layer

Recall: 0.526

This architecture provided the best results. At this point, all emoji classes were present in output. Additionally, the output predictions quite accurate over all classes.

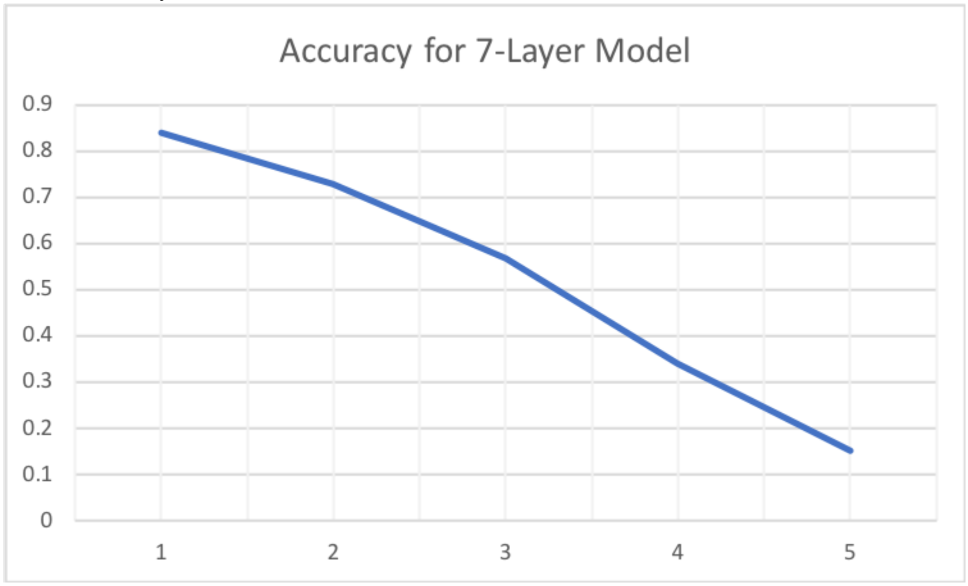
5.4 8-convolutional layer

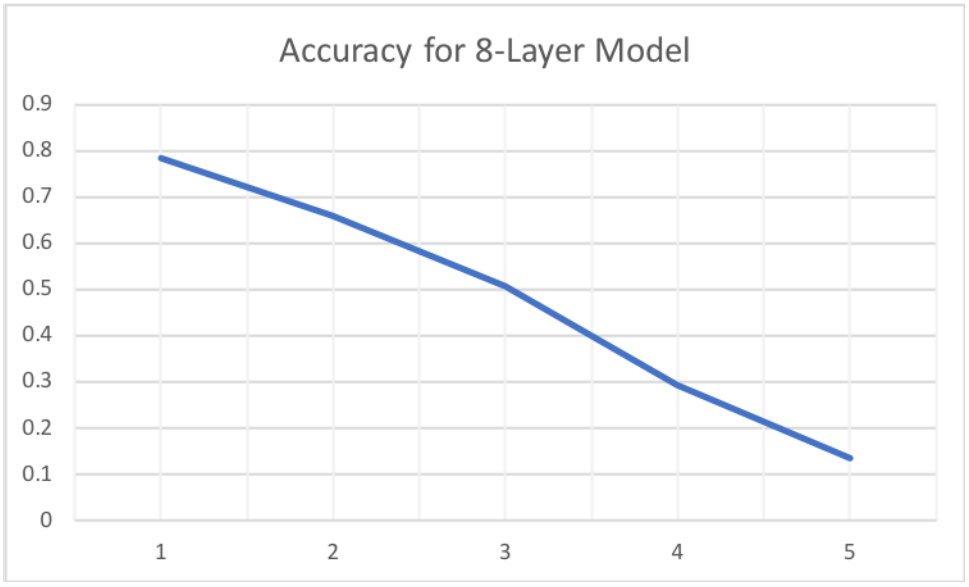
Recall: 0.475

While this architecture provided good results, they were not as quantitatively good as those of the 7-layer architecture. However, it could be the case that the results are subjectively better. After some manual comparison of the subjective quality between the 7-layer and 8-layer models, they were found to be roughly the same, but the question remains unexplored due to the inability of the researchers to check a massive amount of emoji predictions for subjective quality.

5.5 Comparison Between 7-Layer and 8-Layer

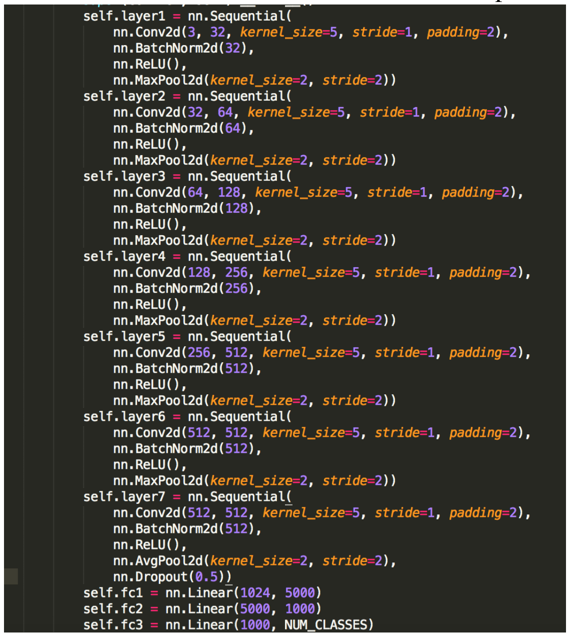
The following demonstrates the difference in accuracies between the two models for n=1, n=2, ..., n=5, where n is the number of true labels that are present in the predicted classes for an image to be considered correctly classified.





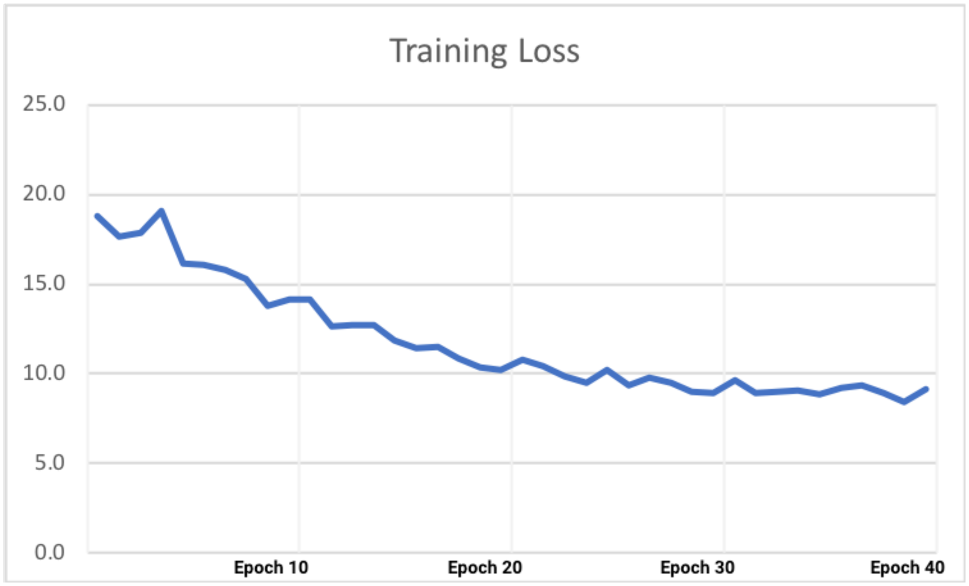
5.5.1 Best Model

The 7-layer architecture was found to be the best model. Here is the specific architecture:



```
        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(64),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(128),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer4 = nn.Sequential(
            nn.Conv2d(128, 256, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer5 = nn.Sequential(
            nn.Conv2d(256, 512, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer6 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.layer7 = nn.Sequential(
            nn.Conv2d(512, 512, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.AvgPool2d(kernel_size=2, stride=2),
            nn.Dropout(0.5))
        self.fc1 = nn.Linear(1024, 5000)
        self.fc2 = nn.Linear(5000, 1000)
        self.fc3 = nn.Linear(1000, NUM_CLASSES)
```

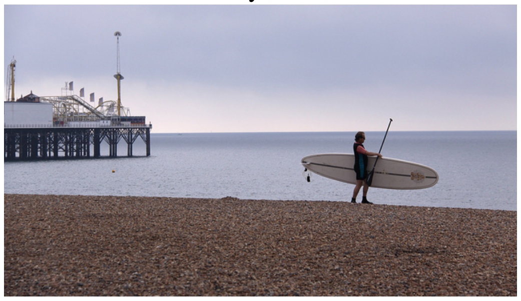
Loss:



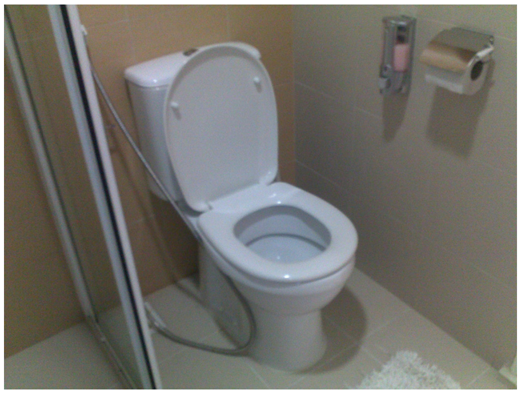
Recall: 0.526

5.5.2 Example Results

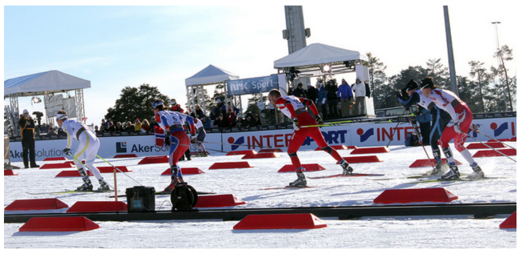
The following results come from the trained 7-layer model.



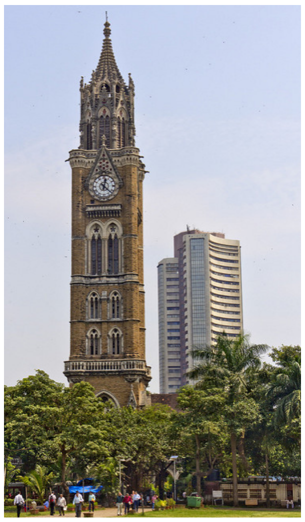
```
Predicted: ['🌊', '👨', '🐚', '🏄', '👙']
Actual: ['🌊', '🏄', '🐚', '👙', '👨']
```



```
Predicted: ['🚽', '🚿', '🛋', '🛁', '🚪']
Actual: ['🚽', '🚿', '🚽', '🛋', '🚽']
```



```
Predicted: ['🎿', '🏂', '🚙', '🗻', '⛄']
Actual: ['🎿', '👪', '👪', '🏂', '🎿']
```



```
Predicted: ['⏰', '🏢', '🏗', '🗼', '🏛']
Actual: ['🗼', '🏢', '⏰', '⏰', '⏰']
```

6 Conclusion

For evaluation, it was not initially clear what sort of metrics we should utilize to quantify our success. We settled on recall as our primary metric: the ratio of true labels that were produced in the corresponding predicted labels. We decided not to use precision for the reason that the predicted labels are forced to be different while the true labels are allowed to be duplicates of the same emoji class(es).

Further, we developed a compromised accuracy metric where we counted a predicted output as correct so long as at least n of the true labels were produced in the predicted output. We calculated this metric for n=1, n=2, , n=5 and graphed the results.

Moreover, there should be an element of subjectivity in our evaluation technique, for two reasons:

(1) There is a tolerable degree of inaccuracy in our true emoji labels for each image, arising from the lack of emoji representations for all ideas in the English language. For example, an image of a giraffe is assigned a emoji labels of a rhinoceros and an elephant since there is no giraffe emoji.

(2) The nature of our model is such that multiple unrelated elements of an image could produce multiple unrelated predicted emoji labels, while the true labels for that image only focus on the major elements of the image while ignoring the minor ones. For example, our trained classifier output the taxi emoji for an image with a taxi in the background (not very noticeable), but the true labels for that image did not include the taxi emoji because the human caption-makers did not find the presence of a taxi in the background to be significant compared to the foreground of the image. This represents a positive sign of generalization in our model that is not reflected by the quantifiable metrics previously

discussed, prompting us to consider the subjective perception of accuracy that is only knowable through human judgment of predicted outputs.

References

[1] Francesco Barbieri, Francesco Ronzano, and Horacio Saggion. What does this emoji mean? a vector space skip-gram model for twitter emojis.

[2] Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel. emoji2vec: Learning emoji representations from their description. *CoRR*, abs/1609.08359, 2016.

[3] J. Gu, G. Wang, J. Cai, and T. Chen. An empirical study of language cnn for image captioning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1231–1240, Oct 2017.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[5] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

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

[7] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):652–663, April 2017.