CPSC 393 Final Project - Image Captioning

Ethan Tarnarider, Daniel Boudagian, Spencer Au

Introduction

The problem that we're working on is image captioning. We used a subset of the MS-COCO 2017 dataset, which is a dataset of about 100,000 images, all with descriptive captions. Our goal for the project was to feed those pictures into a computer vision model with no head and have it learn the features of the images, then train a language model on the features of the images and their respective captions to be able to caption any image we give it. I think this project's goal is important because it essentially is an alt text creator for any given image. While some images may give it trouble, the model performs pretty well on most images and I think it can help make things more accessible for people with disabilities if it was more accurate and effectively deployed.

Analysis

Each image has 5 corresponding captions. The dataset is divided into 118,000 images for the training set, 40,700 images for the test set, and 5,000 images for the validation set. There are 80 object categories and 91 stuff categories, with a total of 1.5 million object instances. No missing values were found in the dataset. Additionally, there were no duplicate images and each image has 5 corresponding captions.. The dataset is already split into a directory containing the training, validation, and testing sets. In addition, the dataset contains json files for captions, instance segmentation masks, and object detection annotations for the training and validation sets.

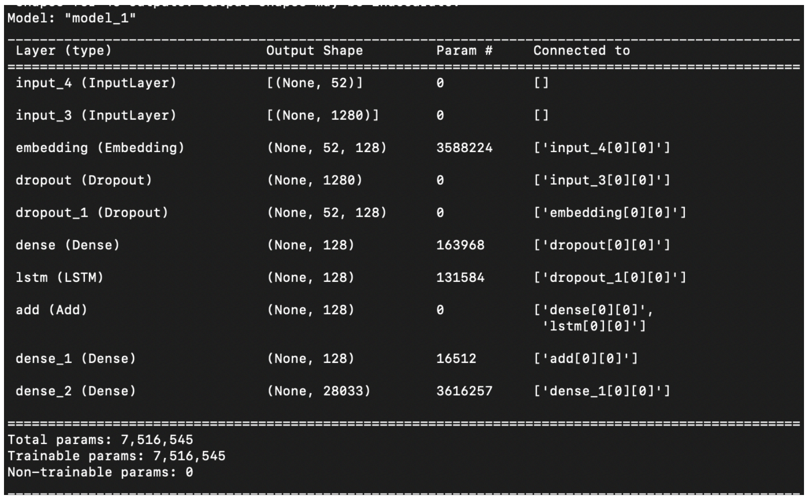
Methods

Pre-processing

In terms of pre-processing, we load in annotations from the JSON files for both the training and validation datasets. The image IDs are mapped to their filenames for quick lookup, and captions are associated with the corresponding image filenames. A dictionary is created where each key is an image filename and its value is a list of captions associated with that image. Each caption is wrapped with special tokens **start** and **end** to denote the beginning and end of a caption. EfficientNetB0, a pre-trained convolutional neural network, is used to extract features from images. Images are loaded, resized to 224x224 pixels, and preprocessed using the preprocess_input function from EfficientNetB0. Features are extracted by feeding the preprocessed images into EfficientNetB0, and the resulting feature vectors are saved for later use. Due to the nature of the image captioning task and the fact that we need to preserve both spatial positional context of objects, we opted to not go with image augmentation for regularization. All captions from both training and validation datasets are combined into a single list. A tokenizer is created and fitted on this list of captions. The tokenizer converts words to unique integer indices and builds a vocabulary of the most frequent words, up to a specified limit **(num_words=15000)**. The maximum length of any caption is also determined, which will be used for padding sequences during training. The tokenizer is saved in JSON format for future use. Features for validation images and their descriptions are also saved to disk.

Architecture

The architecture of our image captioning model consists of two main components: an image feature extractor consisting of a pre-trained CNN and a caption generator utilizing an LSTM RNN.



```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_4 (InputLayer)           [(None, 52)]         0           []

 input_3 (InputLayer)           [(None, 1280)]       0           []

 embedding (Embedding)          (None, 52, 128)      3588224     ['input_4[0][0]']

 dropout (Dropout)              (None, 1280)         0           ['input_3[0][0]']

 dropout_1 (Dropout)            (None, 52, 128)      0           ['embedding[0][0]']

 dense (Dense)                  (None, 128)          163968      ['dropout[0][0]']

 lstm (LSTM)                    (None, 128)          131584      ['dropout_1[0][0]']

 add (Add)                      (None, 128)          0           ['dense[0][0]',
                                                                  'lstm[0][0]']

 dense_1 (Dense)                (None, 128)          16512       ['add[0][0]']

 dense_2 (Dense)                (None, 28033)        3616257     ['dense_1[0][0]']

==================================================================================================
Total params: 7,516,545
Trainable params: 7,516,545
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 1: Caption Model Architecture

Initially we planned to build our model using VGG16 without a head as the CNN component for feature extraction and using a pre-trained transformer like gpt2 for the image captioning, but this was before we really explored the problem or found the dataset. The MS-COCO dataset was essentially the dream dataset for a problem like ours because it allowed us to train a language model on the actual captions for the images due to the wide diversity and variety of the dataset. Once we found the dataset we realized we had a problem, that it took forever to extract the features from the images because it's a huge dataset of images. So we went looking for other pre-trained models that were more efficient but still had high accuracy. We settled on efficientNetB0 with no head for feature extraction. I don't remember how long training took when we were using VGG16 but with efficientNetb0, it takes about a minute on my (Ethan's) local PC and much shorter using the server or colab pro. The structure for LSTM has always stayed the same and we chose to use a LSTM model due to how good they are at text generation and we thought that that short term memory it has would be super helpful for us since we're captioning images and image captions usually have very high cohesiveness in their content. We used layers like embedding to make the words into vectors that keep the semantic mappings where words with similar meanings have similar mappings and layers like dropout in order to help with regularization. We used softmax activation on the output layer because in the case of word generation we are looking for a decimal number between 0 and 1 that correlates to a certain word in our vector mapping.

Image Feature Extractor CNN

Base Model: EfficientNetB0 is used as the base model for feature extraction. It is initialized with weights pre-trained on ImageNet and its top layer is removed. The model is set to non-trainable to retain the pre-trained weights during the initial training phase.
Input Layer: The input to the image feature extractor is a tensor of shape (224, 224, 3), representing the resized and preprocessed image.
Feature Extraction: The input image is passed through EfficientNetB0, and global average pooling is applied to the resulting feature maps, producing a feature vector of size 1280.
Dense Layer: This feature vector is passed through a dropout layer with a dropout rate of 0.5 to prevent overfitting, followed by a dense layer with 128 units and ReLU activation to reduce the dimensionality of the feature vector.

Caption Generator LSTM

Input Layer: The input to the caption generator is a sequence of word indices with a maximum length equal to the longest caption in the dataset.
Embedding Layer: The word indices are passed through an embedding layer that converts them into dense vectors of size 128. The embedding layer is initialized with a vocabulary size equal to the number of unique words in the tokenizer.
LSTM Layer: The embedded word sequences are passed through an LSTM layer with 128 units. The LSTM processes the sequence and returns the last hidden state, which captures the context of the entire caption.

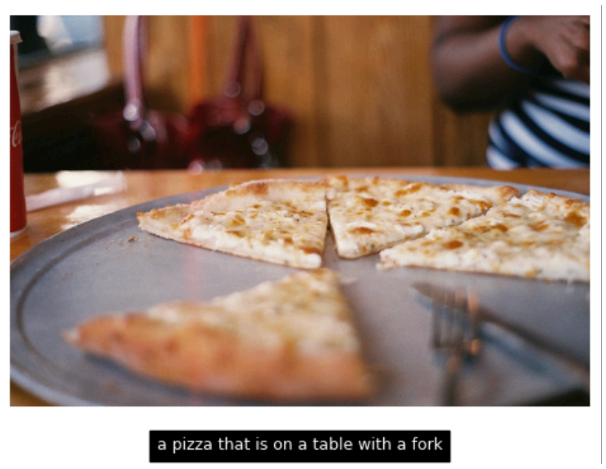
Add Layer: The image feature vector and the LSTM output are combined using an Add layer. This merged representation captures both visual and textual information.
Dense Layers: The combined features are passed through another dense layer with 128 units and ReLU activation to further integrate the information. Finally, a softmax layer is used to predict the next word in the sequence from the vocabulary.

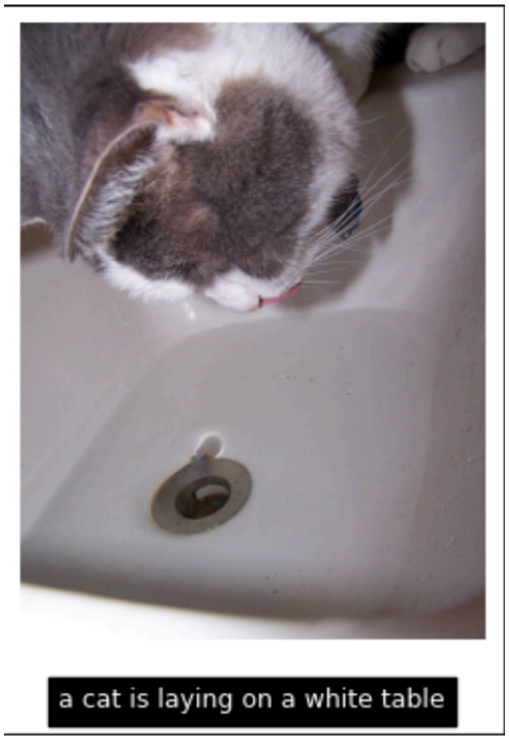
Compilation and Training

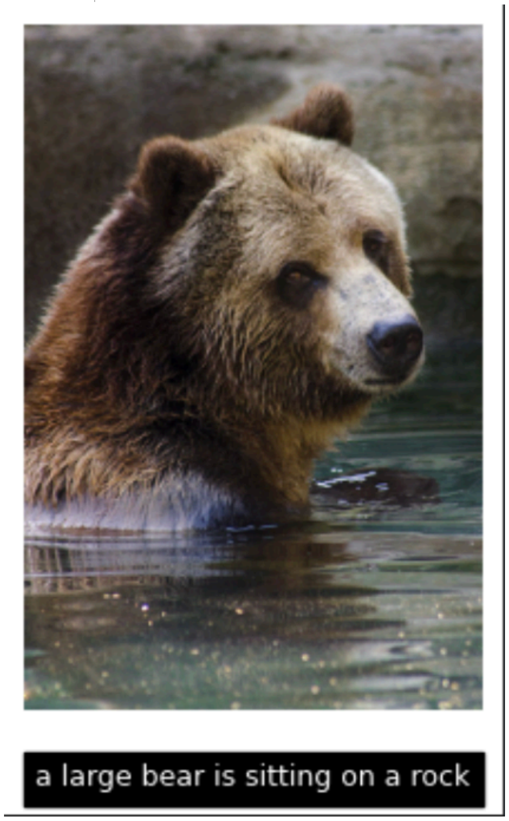
The model is compiled using categorical cross-entropy loss, which is suitable for multi-class classification problems like word prediction. The Adam optimizer with the default initial learning rate is used to train the model, and accuracy is used as the metric to evaluate the model's performance during training. We use a batch size of 10 for training and validation. Data generators are used to yield batches of image features and corresponding text sequences for training and validation. Each batch consists of image features, input sequences (partial captions), and output sequences (next words). We also use Early Stopping Callback to monitor the validation accuracy and stop training if the model does not improve for 5 epochs in order to prevent overfitting. The model is trained for 100 epochs.

Results

Example Captioned Images







From a cursory look at the captions generated for the images, the model seems to be able to generate captions that are somewhat relevant to the content of the images. For example, the first image is pretty spot on, as the model can indicate that the image is of a pizza and it can even recognize the blurred out fork. The second image is somewhat accurate, as the model still recognizes that the main object is a cat, but mistakenly things it is laying on a white table instead of presumably drinking from a white faucet. The third image is also somewhat accurate, as the model recognizes a bear as the main subject of the photo, but seems to think the bear is stiting on a rock instead of in the water.

Metrics

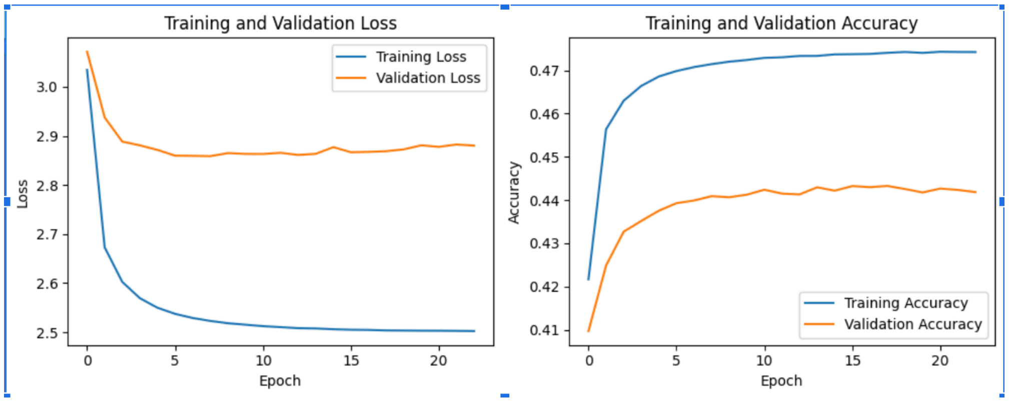


Figure 2: Train and Val Metrics

Both the loss and accuracy seem to convert and have minimal gains around 5 epochs, and the early stopping callback kicks in around 22-23 epochs. Given the better performance from the training set vs the validation set, there is some indication of slight overfitting. Accuracy however is not an ideal metric when considering image captioning, as the model can still generate captions that are semantically correct but not exactly the same as the ground truth captions. For that reason we have also utilized BLEU and CIDEr scores to evaluate the model's performance, two common metrics to evaluate the quality of generated captions.

Metric	Score
BLEU Score	0.1148
CIDEr Score	0.3420

The BLEU score measures how close the captions generated by your model are to human-written captions. It does this by comparing short sequences of words (called n-grams) in the generated captions to those in the reference captions. The score ranges from 0 to 1, with higher scores indicating better quality. The CIDEr score evaluates how well the generated captions match human-written captions by checking the importance and frequency of words in the context of the image. It also ranges from 0 to 1, with higher scores indicating better captions. In terms of the actual results, for the BLEU Score, the generated captions have some similarities to the reference captions but often differ significantly. There is still a lot of room to make the captions more accurate. The CIDEr score seems to be a bit better, with the generated captions somewhat matching the reference captions, capturing some correct aspects but still needing improvement to better reflect the human-written captions.

GradCAM Visualization

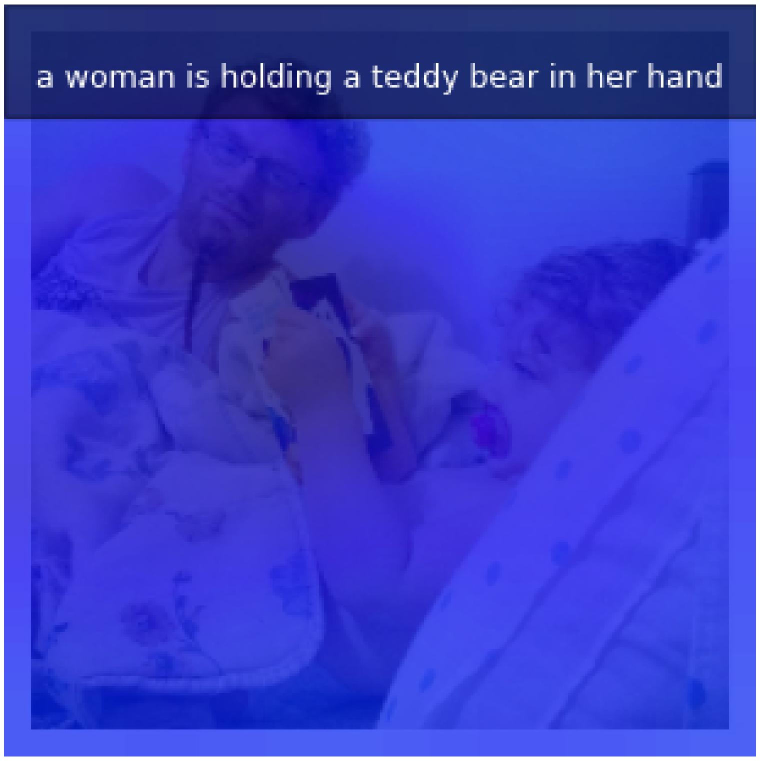


Figure 3: GradCAM Visualization 1

In the GradCAM visualization above, we can see that the model is focusing on the hands of the person in the image, which is relevant to the caption “A woman holding a teddy bear in her hand”. The model seems to be able to identify that the focus should be on the hands, but misidentifies the subject as a woman isntead of a baby and the fact that the baby seems to be reading something.

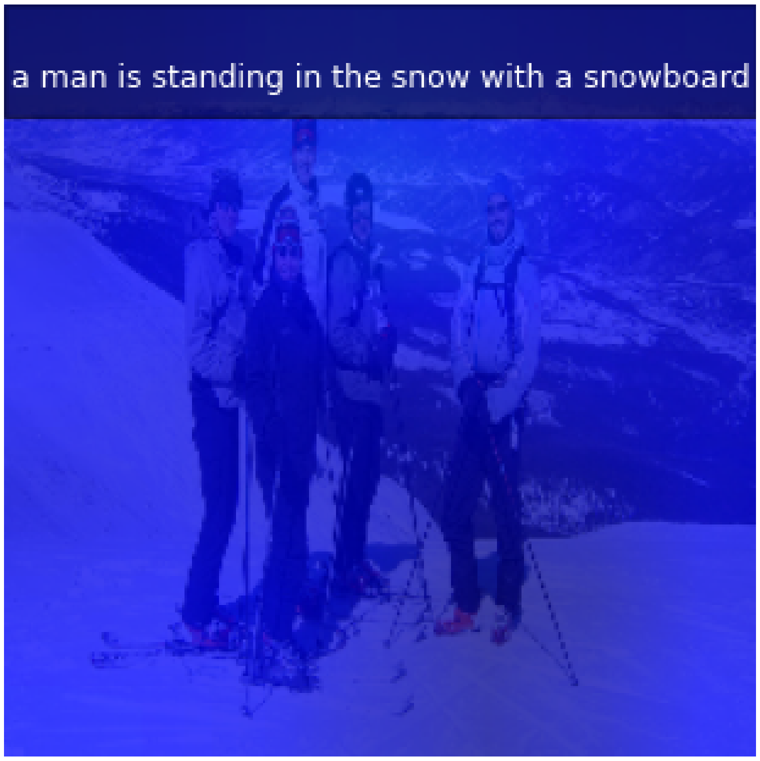


Figure 4: GradCAM Visualization 2

In the second GradCAM visualization, the model is focusing mainly on the shoes in the image, as well as the surround snow with a mild orange hue, which is somehwat relevant to the caption “A man is standing in the snow with a snowboard”. However, the model misidentifies the group as snowboarding when it seems like they all have ski sets. It does seemt to corectly recongize that the man is standing in snow participating in a winter sport.

Reflection

What we learned is the power that neural networks have that if you make your model complex enough it can solve a lot of different problems. What we also learned and discovered in the process of doing this assignment is how long it takes to train a model with such a big dataset and using a retrained model which is very complex. So we had to find more efficient pre-trained models to be faster at extracting features from the dataset. We also learned that training neural networks is an iterative process that is not a one shoe fit all. It requires a lot of fine tuning experimenting with different architectures, loss functions, and different regularization techniques to get the results you are looking for. One thing in the future we would like to try is to either use a RNN or even a transformer to do image captioning for us instead of an LSTM and see if it could do better. In the future we would also like to experiment with different regularization methods for slight overfitting. We would also like to try adding object detection to avoid false mentions of objects. Also we would want to figure out why certain words seem to show up in almost every second or third caption. Words such as scissors or clocks/clock tower seemed to appear a lot despite being irrelevant in most images. Overall we had a good experience doing this project and learning how to use neural networks to train a model to generate captions from an image.