Image Captioning using an LSTM Network

Keshav Rungta
Dept. of Electrical & Computer Engineering
University of California, San Diego
San Diego, CA 92092
krungta@ucsd.edu

Geeling Chau
Dept. of Electrical & Computer Engineering
University of California, San Diego
San Diego, CA 92092
gchau@ucsd.edu

Anshuman Dewangan
Dept. of Computer Science
University of California, San Diego
San Diego, CA 92092
adewanga@ucsd.edu

Margot Wagner
Dept. of Bioengineering
University of California, San Diego
San Diego, CA 92092
mwagner@ucsd.edu

Jin-Long Huang
Dept. of Physics
University of California, San Diego
San Diego, CA 92092
jih002@ucsd.edu

Abstract

Image comprehension is becoming an increasingly important task in today's world as the need to use large sets of images to answer questions about them becomes more pertinent. A first step in that direction, in this paper we try and label a set of images in the COCO dataset using a pre-trained ResNet 50 model (trained on ImageNet) to extract and encode the features of the images and an LSTM network as a decoder to generate the caption one word as a time. Our baseline model using an LSTM with a learning rate of 5e-4, embedding size of 300, and hidden size of 512 achieved a test loss of 1.796, BLEU-1 score of 53.471, and BLEU-4 score of 13.988. We experimented with variations to the model, including using a Vanilla RNN, using stochastic caption generation with different temperatures, and varying the learning rate, embedding size, and hidden size. Our best model used an LSTM with learning rate of 5e-5, embedding size of 800, hidden size of 2048 and deterministic generation, and achieved a test loss of 1.729, a BLEU-1 score of 61.172, and a BLEU-4 score of 18.711, demonstrating that LSTMs can produce successful results for Automatic Image Annotation (AIA).

1 Introduction

With the explosion of data during the big data revolution, visual data in the form of images is no exception. A particular problem is image retrieval in the same way text documents are received. In order to approach this problem, images need to be annotated for their semantic contents. These semantic contents then allow us to gather and answer additional question about the images leading to better image comprehension. Typically, this is done manually, which is both impractical and subjective by nature necessitating an automatic alternative. Thus, significant research efforts have been focused on this issue of Automatic Image Annotation (AIA). Uses for AIA include image retrieval, scalable mobile image retrieval, facial recognition, facial landmark annotation and photo tourism as well as uses in numerous fields of study including urban management, biomedical engineering, social media services, and tourism [1]. The overall goal is to bridge the semantic gap between the low-level image features and a higher level understanding of contexts as well as providing useful keyword searching based on image content.

Here we focus on a deep learning AIA methodology, which exists as one of five general categories of models. The others being generative model-based methods, nearest neighbor model-based methods, discriminative model-based methods, and tag completion-based methods. Deep learning-based AIA is summarized by the combination of visual feature extraction or generation through convolution neural networks (CNN) for image annotation followed by further deep learning techniques for semantic generation often in the form of sequence models. Deep learning provides an effective AIA technique for large datasets where longer training times are acceptable [1]. That being said, there are challenges associated with deep learning techniques including local optimum and difficulty converging as well as lack of interpretability [2].

In this work, we create a deep learning network for automatic image captioning. The baseline model is built as an encoder-decoder system where the encoder is a pre-trained ResNet50 CNN which produces features that are then fed through a linear layer to a long short term memory (LSTM) recurrent neural network (RNN) that can handle data with a temporal structure. This model takes in an input images and produces a caption as a list of strings. We also compared the performance of the LSTM to a vanilla RNN model. For the different models we have we compare stochastic and deterministic methods of caption generation in order to see which one provides us with better BLEU scores. In order to train and evaluate our models, we use the COCO dataset, which contains manually annotated images. This allows us to tune the hyperparameters for our models including the embedding size and hidden size for generally better results.

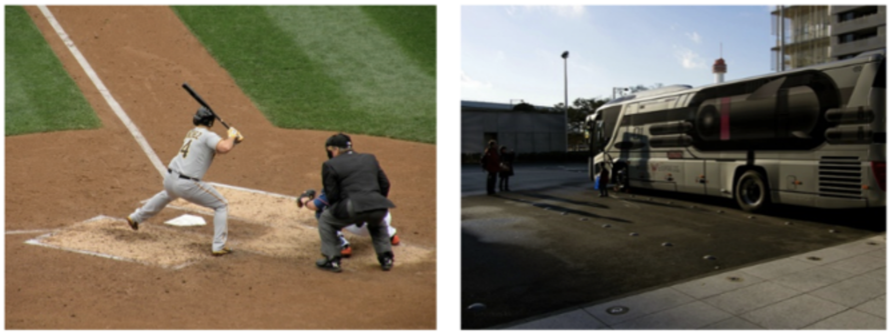
2 Related Works

Natural Language Processing as a genre of problems is forever growing to encompass a more varied set of applications ranging from text translation to text generation. One of the more recent set of solutions in the domain is the usage of Recurrent Neural Networks (RNNs) [3]. RNNs in the modern world have been adapted for numerous problems related to sequential data and generation of sequential output.

However, with the need for models to be able to save its state for longer periods of time, came the inception of Long Short Term Memory (LSTMs) [4]. These provided solutions to problems for numerous problems like recognising handwriting, generating music etc. In this paper, we use LSTMs to caption images and to that effect, we used Andrej Karpathy's blog on RNNs [5] and this blog post that gave us more insight into time series data and how to process them [6]. However, more modern techniques use Transformer networks but we do not use them in this paper.

In order to train and test this network we used the COCO dataset [7] which is one of the largest datasets of its kind. Although not with full captions for images, there are numerous other image based datasets that provide semantic information that can be used in numerous other applications like Kitti [8] which is one of the largest driving based dataset, or CityScapes [9] which contains semantic segmentations of the different classes, or even ImageNet [10] which was used to train our ResNet50 model [11], as it is the largest image based dataset that we know with over 1000 classes labelled on it. We used PyTorch's implementation for ResNet50, LSTMs, and Embedding to implement many of the critical parts for this paper [12].

In this paper, we used Cross Entropy Loss to train our model, and the BLEU metrics [13] to quantify the quality of our captions. BLEU is one of the first metrics to have high correlation with human judgement and remains to be one of the most popular metrics. For a brief introduction to BLEU, we followed the blog [14]. Natural Language Toolkit(nltk) python package was used to calculate BLEU [15]. They implemented both modified n-gram precision and brevity penalty factor. To make sure we don't get abnormal BLEU scores, we compare our results to two most influential papers [16, 17]. We achieved around 60 for BLEU-1 and around 20 for BLEU-4, which are just a few percentages lower than what they accomplished.



The man at bat readies to swing at the pitch while the umpire looks on.

A large bus sitting next to a very tall building.

Figure 1: Some sample images in the MS-COCO dataset and their corresponding labels

3 Methods

In this section we go into more details about the various aspects of our project: the dataset, the baseline model, a vanilla RNN model and the improvements we implemented to try and improve our metrics.

3.1 COCO Dataset

For this work, we are using the Common Objects in Context (COCO) **[7]** 2015 Image Captioning Task dataset. COCO datasets are presented by Microsoft for use in state-of-the-art object recognition tasks including object detection, segmentation, and captioning. The images are complex common daily scenes containing common objects in their natural context with the goal of better understanding scene context at depth. Specifically, we use the 2015 dataset oriented towards the task of image captioning. Thus, the dataset contains images as well as captions describing the primary contents of the images.

For this assignment, we use approximately 20% of the original dataset. The training set contains around 82k images and 410k caption, and the test set contains 3k images and nearly 15k captions. The train, test, validation split was also used as provided for hyper-parameter tuning.

We can see in the sample images and the labels in **Fig. 1**, both the image and the corresponding label is of size 1080×1920×3. The images were preprocessed to make our model more memory efficient which allowed us to increase the batch size. In the provided preprocessing code, the images were cropped to a size of 256×256×3 and the captions were padded and adjusted to include a start and end flag in order to signal to the model how a sequence started and ended.

3.2 Baseline Model

The simple idea to summarise this project is that our model would receive an image and should output a caption for that image after learning the semantic information about it. As such, this task is highly suited to a sequence model. Recurrent neural networks (RNNs) are a type of sequential model specialized to process, or in this case produce, a sequence of values $x(0), ..., x(t)$ [18]. Here, those values are words.

The baseline model for our image caption generation task utilizes an encoder-decoder architecture where the encoder receives image inputs and encodes them into feature vectors. These feature vectors are then passed through a linear layer before finally entering the decoder to generate the image caption. In the baseline, we use a pre-trained convolutional neural network (CNN) as the encoder and an LSTM model as the decoder. Both the linear layer in the encoder and the decoder are trained using back-propagation of error, more specifically the Cross Entropy Loss that we have been using all this while.

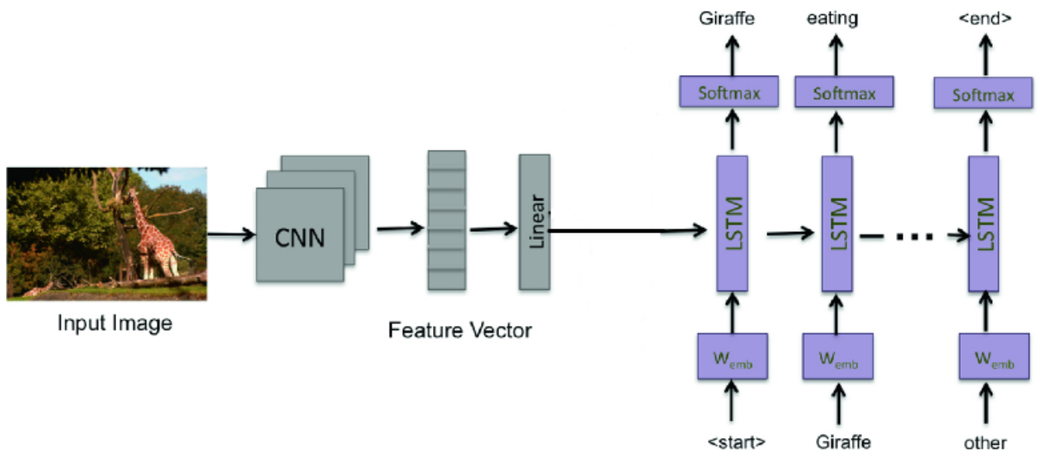


Figure 2: Baseline encoder-decoder architecture using a pretrained ResNet-50 convolutional neural network encoder and long short term memory model decoder

The CNN encoder uses the ResNet50 CNN pretrained on the ImageNet dataset with the last layer, the fully-connected classifer layer, replaced by a trainable linear layer that outputs a vector of features with a tunable fixed size for the input image. The linear layer transforms the image feature vector to a tunable embedding size the same size as the captions after they are embedded so that they can both be inputted into the decoder at the input.

In addition to obtaining the image as a feature vector of embedding size, we also need to obtain the caption words into a similar structure. This process of embedding the captions utilizes Pytorch's Embedding layer which is able to output a feature vector for each word in the caption initialized to our vocabulary data structure. These embeddings can then be fed into our decoder in addition to the feature vector from our image.

The baseline decoder utilizes an LSTM network which is trained using teacher forcing. Teacher forcing functions by using the input at a given step as the teaching signal from the training data at previous step, x(t) = target(t-1). Often, models function by using the output from the last time step, y(t-1), as the input at the current time step x(t), but this can lead to problems such as slow convergence, model instability, or poor performance which teacher forcing works to address [19].

In order to implement teacher forcing, the first timestep receives the encoded image as well as a '<start>' flag as input. The training caption is fed word by word until the '<end >' flag is reached. The words are first one-hot encoded based on the list of used vocabulary and convert to a lower dimensional embedding through a fully connected layer which is trained by backpropagation. Lastly, there is a fully connected layer from the hidden states to the one-hot encoded output.

To create the one-hot encoding, a vocabulary data structure is created mapping a word to a given index. This allows the model to convert between word captions and one-hot encodings with ease.

To implement the LSTM, we used PyTorch's built-in LSTM capability which has three inputs: the current feature, the cell state (c_t), and the hidden state (h_t). To begin, the <start>token embedding is input to the first LSTM cell as $x(0)$. The hidden states of the cell, both h_0 and c_0 are initialized from the image embedding produced by the encoder. The output is the next word in the caption in the training step. In the generation step, the model predicts the next word which is fed into the next LSTM cell as the input. In the baseline model, this is done deterministically where the value is taken as the output with the highest likelihood at each step.

For training, we used Cross Entropy Loss and the encoder and decoder were trained jointly. Training was conducted for 10 epochs (unless otherwise noted) and leveraged early stopping to evaluate the model in the lowest validation loss on the test set.

3.3 Model Variations

3.3.1 Vanilla RNN

To compare the function of the LSTM, we additionally implemented a basic RNN model (vanilla RNN). Compared to the LSTM, typical RNNs suffer from vanishing and exploding gradient problems. LSTMs have additional memory cells which are non-existent in the vanilla RNNs. The expectation is that the vanilla RNN should not achieve as high performance as the LSTM. In this paper, we used the default PyTorch implementation of RNN in order to do this. The rest of the set up is the exact same.

3.3.2 Stochastic Caption Generation

While generating captions, the network functions by predicting the next word in the sentence and using the prediction as the next input. In the baseline model, this was implemented deterministically where we used the highest likelihood of each word as the prediction for that step. Alternatively, the selection can be implemented stochastically. In our model, this was done by sampling from the probability distribution that is the weighted softmax of the output according to:

$$y^j = \frac{exp(o^j/\tau)}{\sum_n exp(o^n/\tau)} \tag{1}$$

In this equation, o^j is the last layer's output, n is the vocabulary size, and τ is the "temperature," which varies how stochastic the sampling strategy is with values approaching zero leading to a approximately deterministic solution compared to large temperature values resulting in a more uniform distribution. This stochastic method of prediction does not affect the training of the model, as this only changes the words that are selected but not the probability of the words which is what was used to train the model.

3.3.3 Hyperparameter Tuning

To tune our hyperparameters, we systematically began with experimenting with learning rate. High learning rates can cause instability in how the model learns as demonstrated by a validation loss curve with lots of variability. Too low learning rates result in models that train very slowly. With feedback on how the loss graphs for training and validation runs, we saw that 5e-4 began overfitting too quickly, suggesting that 5e-5 would have a better opportunity for training to a lower test loss. 5e-6 and smaller trained too slowly, so to restrict to 10 epochs, we selected 5e-5 as our best learning rate.

We also experimented with different embedding and hidden sizes. The embedding size determines the final number of features that the original vocabulary of text is mapped to. High embedding sizes can provide a richer representation of the textual data. The hidden size refers to the number of nodes in the hidden layer. A higher hidden size allows the model to learn more complex representations of the data. As we increased embedding and hidden size, our model performance improved; thus, our best model used the highest embedding and hidden sizes we experimented with (800 and 2048, respectively). More details can be found in **Section 4**.

3.4 Metrics

We saw in the last couple of assignments that proper initialisation of weights has a huge impact on the performance of the network. This is because poor initialization can cause the weights to either explode or the gradients to vanish both of which are detrimental. In our models, we utilized Xavier weight initialization [20]. This strategy initializes weights by sampling a random uniform distribution bounded by

$$\pm\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{2}$$

where n_i is the number of connections into a given layer and n_{i+1} is the number of connections outgoing from a layer. This is to avoid the problem of vanishing gradients for early layers when the fan-in size is large. When it was discovered, it was found to maintain gradient variances across all layers of the network leading to faster convergence and better accuracy.

In order to measure the caption equivalency, we used BLEU-1 and BLEU-4 metrics to evaluate our models in addition to Cross Entropy Loss. The BLEU metric compares a candidate translation against multiple reference translations. It is based on the previous *precision* measure, which is the ratio of number of words (unigrams) in prediction appeared in any reference sentences, to total number words in prediction. This simple precision measure can easily lead to generate too many “reasonable” words, like ”the the the the the the the”, while keeping high precision. To rectify this, BLEU first counts the maximal number of a word occurred in any single reference, and then clips total occurrence of this word in prediction by the maximal reference count, and divides it by the total (unclipped) number of prediction words. See the original paper [13] for some examples. For all n-gram precision, this modification can be made.

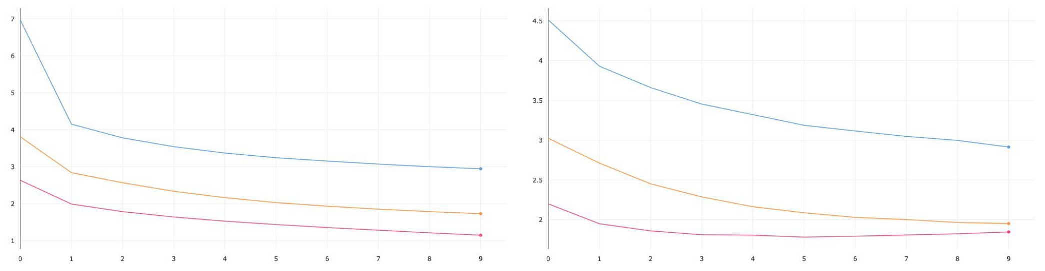
The shorter n-gram matches accounts for word *adequacy*, and the longer n-gram contributes to *fluency*. It’s been tested that using maximum n-gram of order 4 gives best results. Since we want both adequacy and fluency, BLUE-n uses equal weights of k-grams for $k = 1, 2, \cdots, n$. So in our implementation, we use weight vector (1, 0, 0, 0) for BLUE-1, and (0.25, 0.25, 0.25, 0.25) for BLEU-4.

The described way to calculate precision penalizes long sentences, but it doesn’t penalizes short sentences. For example, the model can learn to produce a single word sentence ”the” to get maximum BLEU-1=100%. To get a similar length as the target sentence, BLEU further multiplies a *brevity penalty* factor in front of weighted n-gram score. All of this was done in the nltk package we imported. In our calculation of BLEU scores for each image, we used all 5 actual captions, along with one predication caption.

4 Results & Discussion

Table. 1 records our test results for all the experiments we conducted. In each of the sections below, we further unpack the results and corresponding implications.

4.1 Baseline Model



(a) Training Loss

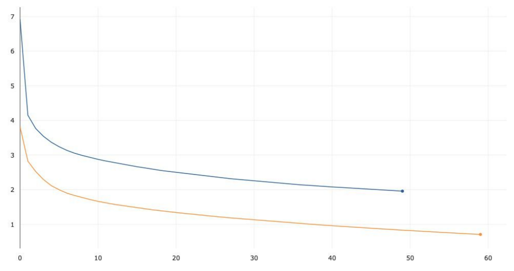
(b) Validation Loss

Figure 3: Varying learning rate for LSTM model with embedding size of 300 and hidden size of 512. (a) Training loss. (b) Validation loss. (Legend) Blue: 5e-6. Yellow: 5e-5. Red: 5e-4.

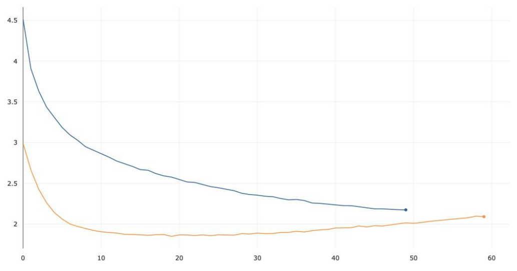
After varying the learning rate, our best baseline model use a learning rate of 5e-5, embedding size of 300, and hidden size of 512 to achieve a test loss of 1.923, BLEU-1 score of 58.824, and BLEU-4 score of 17.197. **Fig. 3** demonstrates that the training and validation loss curves across the three learning rates we experimented with (5e-4, 5e-5, 5e-6) are well-behaved, decreasing with relative stability over time. We recognized that the validation loss using learning rate of 5e-4 overfit the data within 5 epochs, while a learning rate of 5e-6 trained too slowly, which is why we chose a learning rate of 5e-5 to move forward with for the other experiments.

Value	Test Loss	BLEU-1	BLEU-4
Learning Rate			
5e-4	**1.796**	53.471	13.988
5e-5	1.923	**58.824**	**17.197**
5e-6	2.922	36.696	4.964
Longer Training			
5e-5 (60 epoch)	**1.850**	**59.313**	**17.195**
5e-6 (50 epoch)	2.172	55.355	14.092
Vanilla RNN			
Vanilla + 5e-5	1.958	55.298	14.475
Temperature (Stochastic)			
0.01	NA	58.813	17.194
0.05	NA	58.819	**17.202**
0.1	NA	**58.859**	17.190
0.2	NA	58.581	16.739
0.7	NA	51.815	11.237
1	NA	40.465	6.457
1.5	NA	14.835	1.898
2	NA	5.322	0.765
Embedding Size			
100 + 5e-5	2.025	57.603	16.127
500 + 5e-5	1.889	**58.920**	16.734
800 + 5e-5	**1.844**	58.626	**16.991**
Hidden Size			
256 + 5e-5	2.124	53.478	12.789
1024 + 5e-5	1.808	**60.869**	**18.428**
2048 + 5e-5	**1.780**	60.487	18.046
Best Model			
2048H + 800em + 5e-5	**1.729**	**61.172**	**18.711**

Table 1: Model results on 10 epochs per experiment. First we experimented with learning rate to find that 5e-5 had the most potential since 5e-4 quickly began overfitting within 10 epochs. Testing our 5e-5 baseline model with a stochastic caption choice, we achieved highest BLEU results with lower temperatures. Increasing embedding size also gave us a lower test loss, as well as increased hidden size. Thus, we hypothesized that a large hidden and embedding size will produce the best model, which was indeed the case.



(a) Training Loss



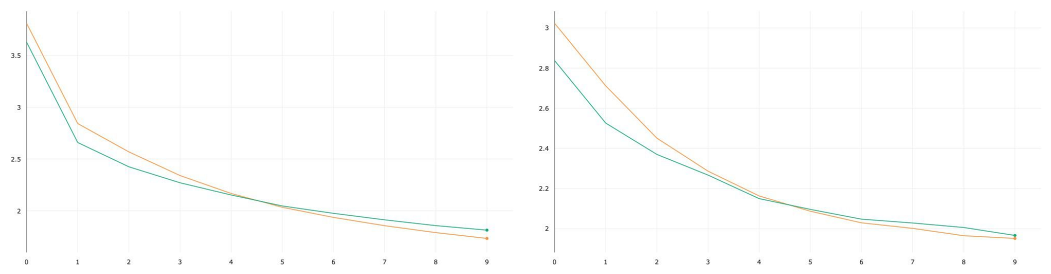
(b) Validation Loss

Figure 4: Varying learning rate for LSTM model with embedding size of 300 and hidden size of 512, trained for 50+ epochs. (a) Training loss. (b) Validation loss. (Legend) Blue: 5e-6. Yellow: 5e-5.

Since 10 epochs did not seem to be long enough for learning rates of 5e-5 and 5e-6 to achieve minimal validation loss, we trained the models on a separate run for 60 and 50 epochs, respectively.

Fig. 4 demonstrates that a learning rate of 5e-5 begins overfitting around 20 epochs, while a learning rate of 5e-6 still underfits after 50 epochs. This suggests that a learning rate of 5e-6 is impractical to train due to its long training times.

4.2 LSTM vs. Vanilla RNN



(a) Training Loss (b) Validation Loss

Figure 5: Varying model type with learning rate of 5e-5, embedding size of 300, and hidden size of 512. (a) Training loss. (b) Validation loss. (Legend) Yellow: LSTM. Green: Vanilla RNN.

Using the learning rate of 5e-5 that we found above, we next compared the performance of the LSTM with a vanilla RNN, leaving the other parameters unchanged. The Vanilla RNN achieved a test loss of 1.958 (higher than LSTM, 1.923), BLEU-1 score of 55.298 (lower than LSTM, 58.824), and BLEU-4 score of 14.475 (lower than LSTM, 17.197). Overall, the performance of the two models were relatively comparable. From the loss curves, we notice that the LSTM loss starts higher than that of Vanilla RNN, but after 5 epochs becomes lower than the Vanilla RNN loss. We also notice that the BLEU scores for the Vanilla RNN model are lower than those of the LSTM.

In summary, these results suggest that the Vanilla RNN needs less epochs to achieve satisfactory performance, but the LSTM model would outperform in the long run. This aligns with our intuition; the LSTM model builds upon the RNN model by introducing a cell state that can encapsulate longer term information. While this allows the model to perform better in the long run once it has learned the right representations to "remember," it performs worse than the simpler RNN for early training epochs.

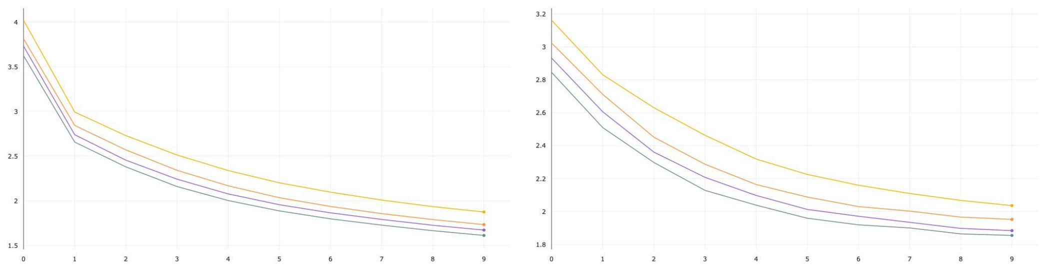
4.3 Deterministic vs. Stochastic Caption Generation

Temperature (Stochastic)		
Value	BLEU-1	BLEU-4
0.01	58.813	17.194
0.05	58.819	17.202
0.1	58.859	17.190
0.2	58.581	16.739
0.7	51.815	11.237
1	40.465	6.457
1.5	14.835	1.898
2	5.322	0.765

Table 2: BLEU-1 and BLEU-4 scores when varying temperature for stochastic generation using LSTM model with learning rate of 5e-5, embedding size of 300, and hidden size of 512.

For our next experiment, we compared deterministic caption generation with stochastic caption generation at various temperatures. We observed that stochastic caption generation with a temperature of 0.1 performed marginally better than deterministic caption generation, with a BLEU-1 score of 58.859 (higher than deterministic, 58.824) and a BLEU-4 score of 17.190 (slightly lower than deterministic, 17.197). At temperatures 0.2 and lower, stochastic generation performance is comparable to that of deterministic generation. However, at higher temperatures, performance steeply drops off. This aligns with our intuition; higher temperatures represent a more uniform distribution of the selection of words. Lower temperatures resemble the deterministic case. Consequently, as the temperature grows vastly, the captions become non-sensical and the model performance decreases drastically.

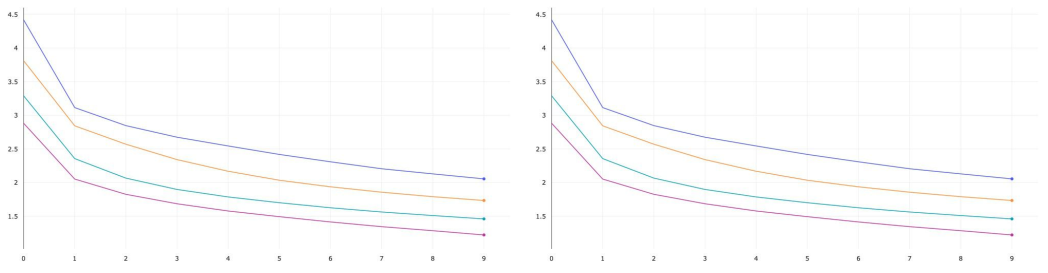
4.4 Varying Embedding & Hidden Size



(a) Training Loss (b) Validation Loss

Figure 6: Varying embedding size of LSTM model with learning rate of 5e-5 and hidden size of 512. (a) Training loss. (b) Validation loss. (Legend) Yellow: 100. Orange: 300. Purple: 500. Blue: 800.

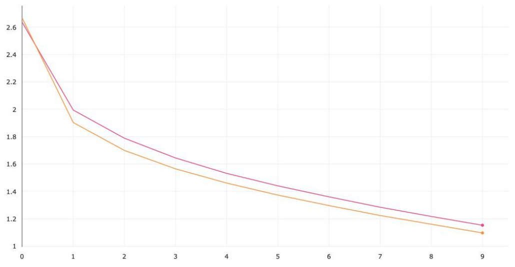
We then experimented with different embedding sizes. The best performing model used an embedding size of 800, with a test loss of 1.844 (lower than baseline, 1.923), BLEU-1 score of 58.626 (higher than baseline, 58.824), and BLEU-4 score of 16.991 (higher than baseline, 17.197). In general, as the embedding size increased, the model performance increased as well. This is because higher embedding sizes allow for a more rich representation of the textual data, which improves performance while our model is underfitting, demonstrated by the still-decreasing validation loss curves.



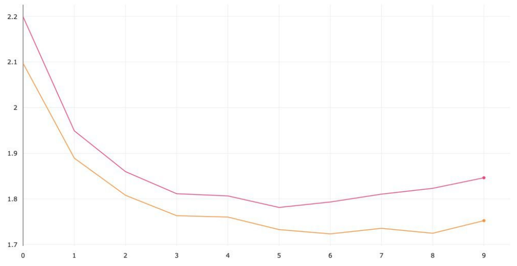
(a) Training Loss (b) Validation Loss

Figure 7: Varying hidden size of LSTM model with learning rate of 5e-5 and embedding size of 300. (a) Training loss. (b) Validation loss. (Legend) Blue: 256. Orange: 512. Green: 1024. Red: 2048.

Similarly, we experimented with different hidden sizes. The best performing model used an hidden size of 2048, with a test loss of 1.780 (lower than baseline, 1.923), BLEU-1 score of 60.487 (higher than baseline, 58.824), and BLEU-4 score of 18.046 (higher than baseline, 17.197). Similar to embedding size, as hidden size increased, the model performance increased as well. Higher hidden sizes allow for more complex learned representations and improves performance while the model is underfitting.



(a) Training Loss



(b) Validation Loss

Figure 8: Performance of best model (learning rate of 5e-5, embedding size of 800, hidden size of 2048) vs. absolute baseline (learning rate of 5e-4, embedding size of 300, hidden size of 512). (a) Training loss. (b) Validation loss. (Legend) Pink: Baseline. Orange: Best model.

4.5 Best Model

Based on the experiments above, we defined our best model by picking the parameters that resulted in the lowest validation loss while maintaining high BLEU scores. With a learning rate of 5e-5, embedding size of 800, and hidden size of 2048 with deterministic generation, the best model achieved a test loss of 1.729 (lower than absolute baseline, 1.796), a BLEU-1 score of 61.172 (higher than absolute baseline, 53.471), and a BLEU-4 score of 18.711 (higher than the absolute baseline, 13.988). With deeper representations of the data and a well-tuned learning rate that optimizes training speed against loss variability, our best model vastly outperforms the absolute baseline **Figure 8**.

Of this best mode, we have plotted 5 good predicted caption examples in **Figure 9** and 5 bad prediction caption examples in **Figure 10**.

We can see that the model does pretty well when the things that need to be identified are easily generalizable, e.g. identifying everyday items and the concept of a person. In the good sample, we clearly see that certain salient objects or people can be identified quite accurately. In identifying objects, common context for the types of objects seem to help it be more accurate. In identifying people, salient features of people such as faces being in the image or the people are easily identified. Even with image distortion, if the object is easily generalizable, the model is able to identify the salient objects. These results are consistent with the fact that we used a pretrained ResNet model which was trained to identify objects in an image to encode our image features.

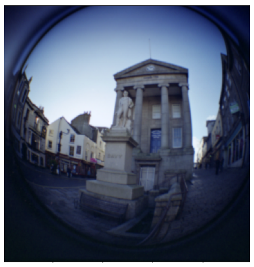
However, when there are more minute details that the model needs to pick up on, it is not able to perform as well. The model mistakes a running dog for a horse, is not able to see the precise contents of sandwiches, is not able to distinguish a face of a woman from a child, and cannot identify cats simply from their tail. In **Figure 10c**, we can see that the model attempts to piece together the context but assembles something that is not consistent with the image, saying that there is a couple on the table rather than a couple of cups of ice cream. In most of these mistakes, we can see how the model might be conflating contextual clues with actual details in the image. With the dog and sandwich, it appears that the grass and sandwich existence is enough for the model to think that the animal is a horse or the sandwich comes with pickles, when it in fact does not. Perhaps in the training set, the ResNet encoder primarily saw running horses and sandwiches with pickles. Similarly, with the people mistakes, the harry potter child may look like a woman facially, but due to the existence of a tie, the model gives the person a beard as well. With the cut off cat image, we can see that the pillow makes it likely the model thinks it is a bed, and since it is more likely for a bed to have a blanket than a cat, it predicts that there is a blanket rather than a cat. These mistakes can primarily be explained by a bias in the original ResNet50 training dataset where these contexts will cause some predictions to be more likely.

(a) Actual captions:
- a desk with a laptop and extra keyboard on it
- a laptop computer on a cluttered desk connected to an external mouse and keyboard.
- the laptop is connected to a full size keyboard to make an effective work station.
- a laptop is set up at an office station.
- a laptop sits precariously on a desk, with a second keyboard in front of it, and windows behind it.
Predicted caption:
- a desk with a laptop computer and a keyboard

(b) Actual captions:
- a bathroom with a toilet and a sink and a bath tub
- a bathroom with sink, tub and toilet black and white
- a bath room with a bath tub a sink and a toilet
this bathroom has a sink, toilet, bathtub, and shower.
- a small bathroom has a white tub, sink and toilet with a wooden lid.
Predicted caption:
- a bathroom with a sink and a toilet in it

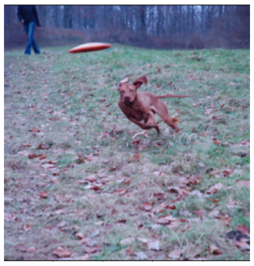
(c) Actual captions:
- a happy woman about to eat a slice of pizza.
- a woman smiles as she holds a plate of food
- an asian woman smiles as she received pizza
- woman smiling receiving a plate with a slice of pizza.
- the older woman smiles as she holds a plate with a slice of pizza.
Predicted caption:
- a woman is eating a piece of pizza.

(d) Actual captions:
- a man and a woman are in the snow with helmets.
- a man with his arm around a woman in front of several skiers.
- two people posing for a photo with young people in the background wearing skis
- a man and a woman wearing eye protection and coats stands in the snow near a couple of skiers.
- a couple embracing on a snow covered ski slope.
Predicted caption:
- a couple of people that are standing in the snow.

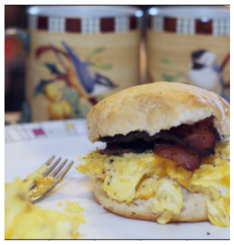


(e) Actual captions:
- a tall building with a large white statue.
- a statue of a man standing in front of a building
- a fish-eye view of a statue in front of a columned building.
- a stylized photo of a statue in front of a building.
- a statue standing in front of a building shot with a fish-eye lens.
Predicted caption:
- a building with a clock on the side of it.

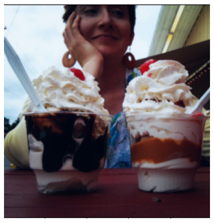
Figure 9: Best model: Good caption predictions vs actual



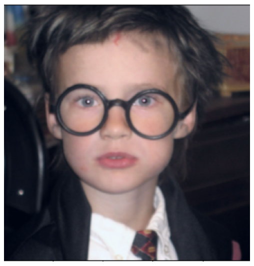
(a) Actual captions:
- a dog running towards a frisbee in the air
- a dog running towards a frisbee on some grass and leaves.
- a tan dog running in the grass after a frisbee
- a person is in the distance while a brown dog is in midair and is running after a frisbee.
- a brown dog chasing a frisbee in a field
Predicted caption:
- a person is riding a horse in the grass.



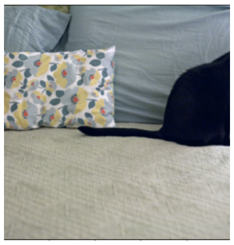
(b) Actual captions:
- a plate with a breakfast sandwich, eggs and a fork next to two coffee mugs.
- a sandwich with scrambles eggs on a white plate.
- a plate with a breakfast sandwhich and some other food along with 2 mugs
- a breakfast sandwich sits on a plate with a fork in front of two cups
-scrambled egg and bacon on a biscuit at home
Predicted caption:
- a sandwich with a pickle and a pickle on it.



(c) Actual captions:
- two cups of ice cream sitting on a table in front of a woman with whipped cream and a cherry on top.
- the woman sits in front of two ice cream sundaes.
- a lady sitting at a table with two sundaes in front of her.
- a woman sitting next to a table with desserts in front of her face
- a woman is sitting in front of some ice cream on a table
Predicted caption:
- a couple of people that are sitting on a table.



(d) Actual captions:
- a little boy dressed up in a ”harry potter” costume.
- a young boy wearing glasses and a neck tie.
- a boy dressed in a harry potter type outfit in a room.
- a little boy in a suit wear eye glasses
- a young boy in harry potter round glasses and tie.
Predicted caption:
- a woman with a beard and a tie.



(e) Actual captions:
- a cat looks off the edge of a made up bed that has blue pillows and a floral pillow.
- there is a black cat sitting on the edge of the bed
- a bed has a brown cover, blue pillows and a floral decorative pillow and a black cat sits on the edge.
- a black cat seated on the edge of a neatly made bed
- a cat sitting on a neatly made bed
Predicted caption:
- a bed with a blanket on the floor and a blanket on it.

Figure 10: Best model: Bad caption predictions vs actual

5 Conclusion

We were successful in implementing an LSTM to solve the problem of Automatic Image Annotation (AIA), with a test loss of 1.729, a BLEU-1 score of 61.172, and a BLEU-4 score of 18.711. Fine-tuning our learning rate and increasing the embedding and hidden sizes improved the performance of our model. As expected, the LSTM outerperformed the simpler Vanilla RNN model and stochastic caption generation resulted in similar performance to deterministic caption generation for low temperature values. For future work, we can consider to add attention mechanisms and model the textual data using transformers.

6 References

[1] Qimin Cheng, Qian Zhang, Peng Fu, Conghuan Tu, and Sen Li. A survey and analysis on automatic image annotation. *Pattern Recognition*, 79:242–259, 2018.

[2] Dengsheng Zhang, Md. Monirul Islam, and Guojun Lu. A review on automatic image annotation techniques. *Pattern Recognition*, 45(1):346–362, 2012.

[3] Ilya Sutskever and Geoffrey Hinton. Temporal-kernel recurrent neural networks. *Neural Networks*, 23(2):239–243, Mar 2010.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[5] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks, May 2015.

[6] Roman Orac. Lstm for time series prediction, Sep 2019.

[7] 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. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[8] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[9] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[12] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[13] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[14] Jason Brownlee. A gentle introduction to calculating the bleu score for text in python, 2017.

[15] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[16] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks

for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[17] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] Jason Brownlee. What is teacher forcing for recurrent neural networks?, 2017.

[20] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.