Image Captioning with LSTM and RNN Networks

Mitchell Zhang
miz134@ucsd.edu

Jun (Moses) Oh
j6oh@ucsd.edu

Nikhil Pathak
npathak@ucsd.edu

Amithab Arumugam
amarumug@ucsd.edu

Abstract

By storing memory, recurrent networks surpass feedforward networks at tasks involving sequences where dependencies occur in the time dimension for an arbitrary number of time steps. We compare different recurrent networks such as RNNs and LSTMs plus a variation of LSTM which we denote Architecture 2 on the COCO Image dataset. The models are trained to generate captions from images with ResNet50 as the encoder feeding its output to the decoder, one of the recurrent networks under analysis. We train and evaluate models with teacher forcing using cross-entropy loss on one-hot encoded outputs. When generating captions, we compare model performance using BLEU-1 and BLEU-4 scores at test time, which is a widely used metric for comparisons of predicted sentences to target sentences. In comparisons, test loss is not as effective of a metric because the loss is calculated with teacher forcing whereas BLEU scores are taken from independent generations by the network. As mentioned before, we explore baseline LSTM and RNN models with a default configuration provided, while also additionally tuning the network hyperparameters to achieve better BLEU performance. Our tuned baseline LSTM model produced BLEU-1 and BLEU-4 scores of 64.0168 and 8.1465, which outperformed compared to the best vanilla RNN model, which had BLEU scores of 63.0679 and 7.9217. Finally, we also explore a variation of the LSTM model that feeds the image embedding from the encoder into every time step LSTMCell input. While also exploring a default configuration of this model, we additionally tune this network to produce our best model performance with BLEU scores of 66.2410 and 9.2925. For the tuning, our hyperparameters included image size, batch size, number of epochs, learning rate, hidden size for LSTM cell, embedding size, image embedding size, and temperature for stochastic caption generation. We also explore choosing between using a deterministic approach or a stochastic (weighted softmax) approach for generating a caption word from the output of each cell. This investigation taught us the deterministic approach is more stable while the stochastic approach manages to reach and even surpass the former in performance within a certain range.

1 Introduction

In our project, we use a subset of the COCO (Common Objects in Context) 2015 Image Captioning Task dataset. This subset is randomly split into a training set of size 14900, validation set of size 1656, and test set of size 3000. Each dataset value comes with an image and 5 captions. This dataset split is provided to us by the CSE 151B staff. We use cross-entropy loss on one-hot-encoded output predictions to evaluate accuracy of our model on the validation set for hyperparameter tuning. This loss is generated using teacher-forcing–a concept used for training the model and evaluating losses in which we feed the ground-truth word of the caption from the previous time step (instead of the prediction word) as the next time step's cell input. This helps the model learn more efficiently! We then use BLEU-1 and BLEU-4 scores to evaluate our model's ability to generate captions on the test set data (Note: all BLEU scores are multiplied by 100 to better represent as percentages). The loss and BLEU scores are averaged across every sample. Prior to calculating BLEU scores, we convert all words to lowercase and remove punctuation marks. Then, we apply nltk's word_tokenize to split up the caption for both the generated and target captions [6]. In general across all our models with default configuration hyperparameters, we found validation and test loss in the range of 1.4-1.5. BLEU-1 scores are around 60-64, and BLEU-4 scores are around 6-8. We generally noticed validation loss begin to decrease much less at around epoch 4 stabilizing at a value of around 1.4. After tuning hyperparameters, we were able to increase BLEU-1 and BLEU-4 scores to 64.0168 and 8.1465 for the baseline LSTM model and 63.0679 and 7.9217 for the vanilla RNN model. We then tuned the Architecture 2 model (with image embeddings used at every time step) to achieve the highest BLEU-1 and BLEU-4 scores of 66.2410 and 9.2925. The most important hyperparameters we tune include learning rate, epoch size, hidden size, and temperature. Generally, we find decreasing learning rate, increasing epoch size, decreasing hidden size, and maintaining a low temperature are key in improving the model's performance overall and decreasing overfitting. We found that the LSTM model performed better than the RNN model overall, while the Architecture 2 model performed the best of all the models. In the report we explore more in detail some of these trends of hyperparameters in addition to explaining why some models performed better than others.

2 Baseline LSTM vs RNN

2.1 LSTM

An LSTM network attacks the problem of modeling sequences such as speech or image captions through mapping time into state via maintaining long-term information from previous time steps through the hidden and cell states. LSTMs are unique in their gate architecture in which there exists a forget, input, and output gate per LSTMCell which serves different purposes in learning the importance of some time steps and inputs.

For our baseline LSTM network, we use an encoder-decoder architecture. Since we apply teacher forcing during training, validation, and testing, the model takes in a batch of images and their target captions padded to fit the length of the longest caption. The model generates a sequence of one-hot vectors for each image representing the predicted caption. Trained on ImageNet, ResNet50 is a powerful feature extractor that serves as our encoder to output a feature vector for the input images. The output matches the embedding size that is input for the LSTMCell [1], [2] used in our decoder. The first input to the cell is the encoder's output with the hidden states [4], comprised of both the cell's hidden and cell state [5], initialized to 0. The output hidden state passes through a linear layer resulting in a one-hot vector the size of the training set's vocabulary. This becomes part of the final output of the forward pass. For subsequent passes through the cell and linear layer, teacher forcing is applied where the input to the LSTMCell is the embedded target caption of the previous time step and the hidden states are the previous time step's hidden states. Embedding is represented by PyTorch's Embedding layer [3], a mapping from our vocabulary to the embeddings, which through backpropagation learns appropriate embeddings to map words in the target captions to vectors the length of the embedding size.

Generation with this model is very similar with slight changes to the decoder to have it "run on its own" since captions may not necessarily accompany images at inference time. It is important to note that teacher forcing does not apply here since captions are not part of the input, and we want the model to generate captions on completely unseen images. When generating captions deterministically, after getting the one-hot vector from the linear layer from the previous time step, the word with the highest prediction value is chosen as input to the LSTMCell (after retrieving its embedding) for the current time step. For stochastic generation, a distribution based on a slightly modified softmax, parameterized by the temperature, is used to sample the next word.

The weighted softmax of the outputs follows the following formula:

$$y^j = \frac{e^{\frac{o^j}{\tau}}}{\sum_n e^{\frac{o^n}{\tau}}}$$

where o^j is the output from the last layer, n is the size of the vocabulary, and τ or temperature is defined as a hyperparameter (referenced PA instructions here).

2.2 RNN

Unlike conventional feedforward neural networks, recurrent neural networks consist of loops in order to model sequences such as speech or image captions. These loops can be thought of as feed-forward neural networks chained together where each layer represents a certain timestamp (mapping time into state). RNN forward propagation can be thought of as a feedforward neural network with shared weights. Each chained layer alters the same weight matrix and receives as input the previous layer's output. This allows the RNN to "remember" previous inputs - although not perfectly (vanishing gradients). RNN back propagation (Back Propagation Through Time) also works similarly except this time, all errors within the layers in each time step are accumulated to update the weights according to an averaged gradient.

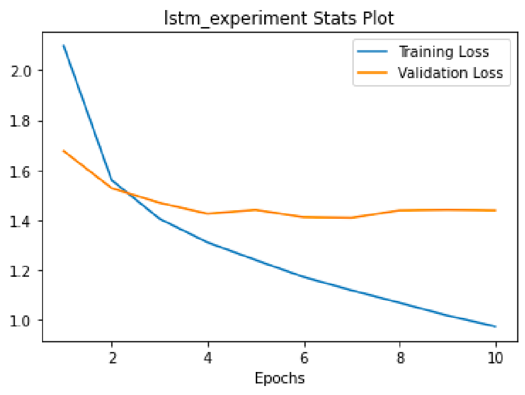
For our Vanilla RNN architecture, we follow the same implementation as our Baseline LSTM architecture, but PyTorch's RNNCell [7] replaces LSTMCell and the hidden states do not contain the cell state anymore. We first test with the same set of default hyperparameters as the baseline LSTM model which includes the hidden units, optimizer, and learning rate. We then tune our hyperparameters to discover a better result. In comparison to LSTM, Vanilla RNN suffers more from the vanishing gradient problem where the gradient of the loss function decays exponentially, as it propagates backwards in time. LSTM's cell state allows for the vanishing gradient problem to be more effectively handled than the RNN which only passes information from previous time steps through the hidden state activations. Because of this handling of the vanishing gradient problem, we expect a better performance in LSTM compared to RNN.

2.3 Default Comparison

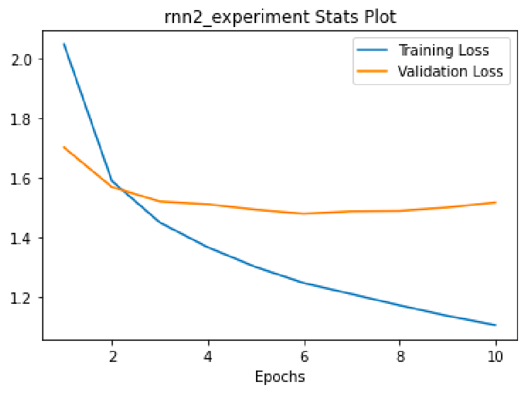
First, to properly compare the models, we use the same default hyperparameters for both LSTM and RNN which are:

$$\begin{aligned}
\text{image size} &= 256 \\
\text{batch size} &= 64 \\
\text{number of epochs} &= 10 \\
\text{learning rate} &= \text{5e-4} \\
\text{hidden size} &= 512 \\
\text{embedding size} &= 300 \\
\text{max length} &= 20 \\
\text{deterministic} &= false \\
\text{temperature} &= 0.1
\end{aligned}$$

2.3.1 Results



(a) LSTM

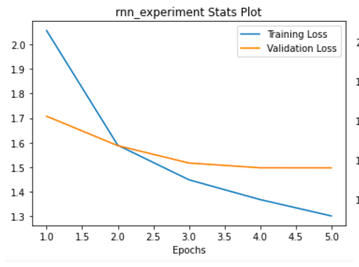


(b) RNN

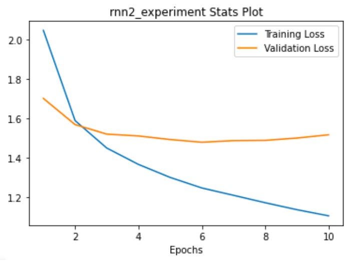
Figure 1: Training and Validation Loss Curves with Default Hyperparameters

Figure 1 shows the training and validation loss curves of our baseline models prior to tuning hyperparameters. This resulted in a test set loss of 1.4210, BLEU-1 score of 63.1867, and BLEU-4 score of 8.1538 for the baseline LSTM model. With the same default hyperparameters that we used for the baseline model, we saw the following test performance for RNN: Loss: 1.4871, Perplexity: 0, Bleu1: 61.9765, Bleu4: 7.5803. We can see a slight increase in test loss for the default RNN model (1.42 vs 1.49) compared to the our default baseline LSTM model. We also see the BLEU scores for the default LSTM are about 1 higher than the respective scores for the default vanilla RNN.

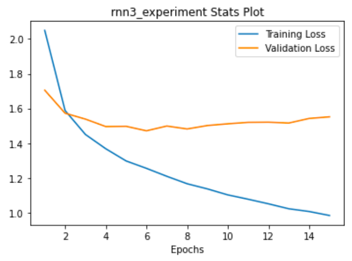
Next, we experiment with number of epochs for the RNN model to see if it has any effect on performance. The results are shown in Table 1. Figure 2 shows the loss curves for the three models trained for 5, 10, and 15 epochs respectively.



(a) RNN trained for 5 epochs



(b) RNN trained for 10 epochs



(c) RNN trained for 15 epochs

Figure 2: Training and Validation Loss Curves for RNN with Default Hyperparameters and Varying Epochs

Epochs	Test Loss	BLEU-1	BLEU-4
5	1.4947	61.4919	7.3996
10	1.4871	61.9765	7.5803
15	1.4844	62.7403	7.8973

Table 1: Performance across Models

2.3.2 Discussion

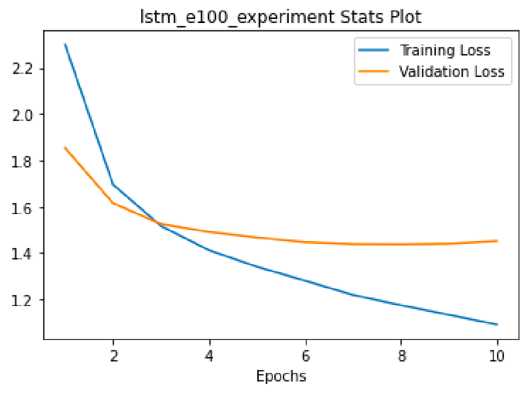
Both LSTM and RNN start off at the same loss but LSTM reaches a lower training and validation loss in the same number of epochs as seen in Figure 1. In terms of test loss, LSTM does better as well. With regards to the learning curves we see in the Figure 1, the two models have very identical curve shapes, especially with regards to the training curve. As mentioned, the training loss ends at a lower number for LSTM than RNN, but the shape of the curves are identical which tells us the models are learning at a very similar rate. The validation loss curves do differ more between the two models, as we see the default RNN begin to overfit and increase more in the validation loss towards the ending epochs. The LSTM validation loss curve plateaus around the same time as the RNN validation curve, but the LSTM curve does not begin to increase as drastically in the last few epochs. This tells us the LSTM model fights overfitting better than the RNN model does, and the reason for this can be attested to the different structures in an LSTMCell vs an RNNCell. It seems as though the RNNCell is more prone to overfitting than an LSTMCell which also has a cellstate to learn from in addition to just a hidden state.

Based on Table 1, increasing the number of epochs reduces test loss and improves both BLEU scores. With a test time BLEU-1 score of 61.97, RNN needs more epochs to achieve a similar performance level to LSTM which gets a BLEU-1 score of 63. The model running for 15 epochs gets much closer to this value with a BLEU-1 score of 62.7. LSTM's ability to remember long-term dependencies allows it to train faster than RNN and provides the reason for the behavior seen in these experiments. Therefore, RNN needs more epochs to achieve satisfactory performance. As mentioned before, it does appear that the more epochs allotted for RNN causes some more overfitting which we see with increasing validation loss towards the end of the 15 epoch graph. So there is a fine-line between the number of epochs helping or hurting the RNN model on unseen data, as more epochs may prove useless if they are simply overfitting.

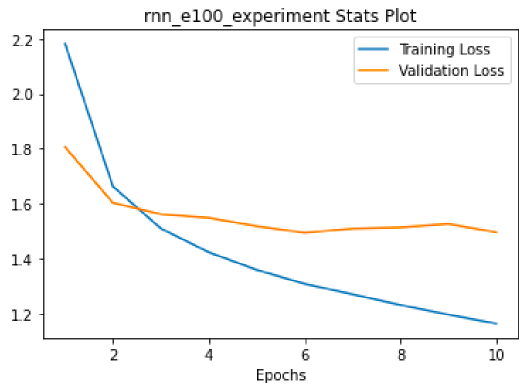
2.4 Best Model Comparison

Now, we present the best models for LSTM and RNN after tuning our hyperparameters.

2.4.1 Results



(a) LSTM



(b) RNN

Figure 3: Training and Validation Loss Curves with the Best Hyperparameters

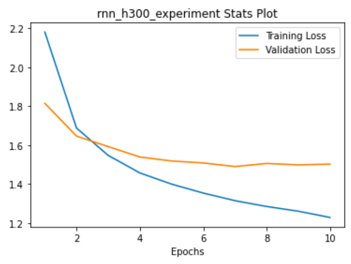
Interestingly, our best tuned models do not require much changing to the default configuration. For both these best tuned models, we change the embedding size to 100 while the rest of the hyperparameters remain the same as the defaults which allows for a more robust comparison. This resulted in a test set loss of 1.4375, a BLEU-1 of 64.0168 and BLEU-4 of 8.1465 for LSTM. The tuned RNN model resulted in a test loss of 1.5169, Bleu1 = 63.0679, Bleu4 = 7.9217.

On the way to the best models, some of the main tuned hyperparameters included different embedding and hidden sizes. For example, we show some of our experiments with just the RNN model in Figure 4 and 5.

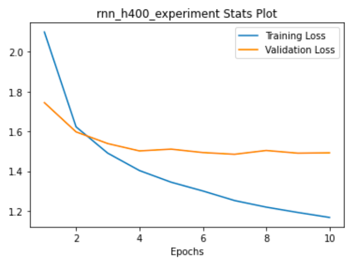
Hidden Size	Test Loss	BLEU-1	BLEU-4
300	1.5039	61.1682	7.0474
400	1.5138	62.0677	7.4979
600	1.4918	61.4218	8.1278
700	1.4934	62.2050	7.8065
1024	1.5065	61.6296	6.9269

Embedding Size	Test Loss	BLEU-1	BLEU-4
50	1.5321	62.2936	8.0562
100	1.5169	63.0679	7.9217
200	1.4926	62.1954	7.9219
400	1.4824	61.5460	7.5863
500	1.4896	61.5455	7.1716
600	1.4755	60.9027	7.2001

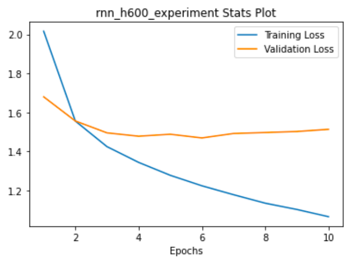
Table 2: Performance across RNN Models with different hidden and embedding sizes



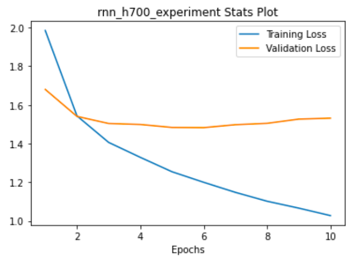
(a) RNN - 300 Hidden Units



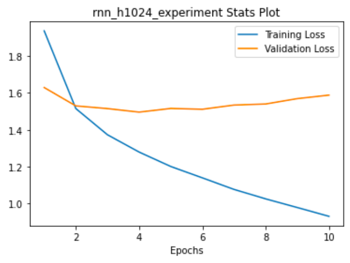
(b) RNN - 400 Hidden Units



(c) RNN - 600 Hidden Units



(d) RNN - 700 Hidden Units



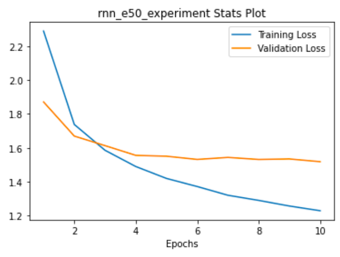
(e) RNN - 1024 Hidden Units

Figure 4: Training and Validation Loss Curves for RNN with Default Hyperparameters and Varying Hidden Sizes

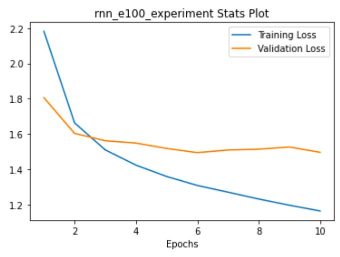
2.4.2 Discussion

The tuned RNN model has similar training and validation losses with the validation loss being higher than that for the tuned LSTM model as seen in Figure 3. Similar to the default models, there is more overfitting with the RNN since it cannot remember the long-term dependencies as well as the LSTM model. Compared to the models with the default hyperparameters, both of our best models have validation curves much closer to the training loss curves meaning that they are learning to generalize much better with the reduced embedding size. The validation loss curve for RNN crosses the training curve an epoch earlier than for LSTM. Overall, LSTM performs better with higher BLEU scores ($64.02 > 63.07$ and $8.15 > 7.92$) and lower loss ($1.44 < 1.52$). This may be because of the vanishing gradient problem in the RNN model, which in turn may cause less accurate captions to be generated as well, causing a lower BLEU score. This means that the LSTM model generates better captions based on n-grams of size one and four–potentially explaining industry preference for LSTMs over RNNs.

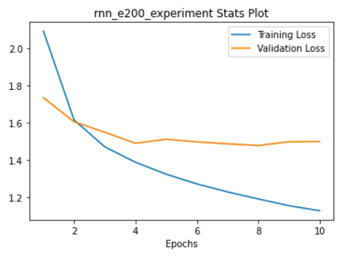
When generating captions deterministically, the best LSTM model has a test loss of 1.441 with a BLEU-1 and BLEU-4 score of 64.134 and 8.114 respectively. The best RNN model has a test loss of 1.524 with a BLEU-1 and BLEU-4 score of 63.305 and 7.896 respectively. Again, LSTM outperforms RNN, but more importantly, the deterministic approach boosts BLEU-1 scores for both models than when the temperature is set to its default of 0.1. However, both the BLEU-4 and test loss scores get worse which indicates that removing randomness helps generate n-grams of size 1 correctly while a bit of randomness is necessary when sampling to generate accurate n-grams of size



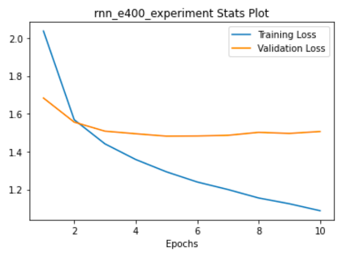
(a) RNN - 50 Embedding Size



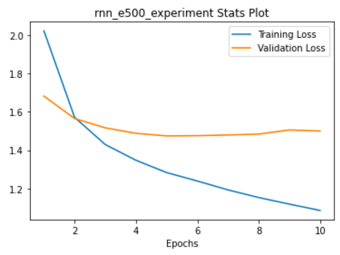
(b) RNN - 100 Embedding Size



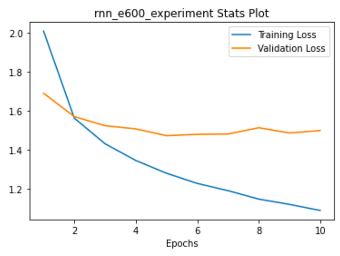
(c) RNN - 200 Embedding Size



(d) RNN - 400 Embedding Size



(e) RNN - 500 Embedding Size



(f) RNN - 600 Embedding Size

Figure 5: Training and Validation Loss Curves for RNN with Default Hyperparameters and Varying Embedding Sizes

4. The LSTM model outperforming the RNN model is consistent with what we have discussed so far in the report–the vanishing gradient problem is an issue handled more effectively by the LSTM model which is why we see stronger performance from it. The LSTM model also has a cell state that provides longer-interactions with previous time steps.

For our other experiments, we trained models on a variety of different hyperparameters. When reducing the learning rate, it produced smoother plots but required more epochs to achieve similar scores. When tuning the hidden size, Table 2 shows how the loss decreases then increases while the BLEU scores reflect the inverse. Figure 4 reveals how much overfitting occurs as the hidden size increases. The training loss gets lower and lower while the validation loss monotonically increases. Increasing the hidden size allows the model to memorize its inputs to a higher degree leading to overfitting and poor test performance. Similarly, Figure 5 shows the same overfitting problem when the embedding size increases. For our metrics in this case, all three decrease after a certain point as the embedding size increases as displayed in Table 2 meaning that the model may not be powerful enough to learn all the correct signals as the input which is the embedding from the previous layer or the image increases in size. We briefly experimented with batch size where larger sizes improved test loss but maintained or decreased bleu scores. Several data augmentation techniques were applied with the other hyperparameters fixed to the defaults. Color jitter is seen as a reasonable transformation since affecting certain aspects of the color such as saturation, contrast, hue, and brightness should not change the caption for the image. This resulted in a slightly better BLEU-1 score of 62.2 over the default RNN model, but it did not work as well when the embedding size was decreased to 100. Random erasing was applied but produced even lower gains due to the reduced information from the original image that made it to the network. Random horizontal flips had minimal gains that could be attributed to randomness, while different random crops reduced performance due to the mismatch between the focus of the picture that resulted in the image and the area actually cropped. We used the Adam optimizer throughout which performed higher than the SGD optimizer.

3 Baseline LSTM: Temperature

To convert captions to be allowed as inputs for our network, we created a vocabulary of all the words from the train set captions. We also added the custom words ”<pad>”, ”<start>”, ”<end>”, and ”<unk>”. Each word is then one-hot encoded when used as input and decoded from the same vocabulary/dictionary set. Captions were added with "<pad>" if they contained less words than the maximum length, so all captions were of equal length.

In deterministic approach, we would take the maximum output prediction as our caption word for each step. Since this lacks randomized sampling, each run would produce the same caption on the same input data.

In stochastic approach, we would take the weighted softmax. The formula we used for this was:

$$\begin{aligned} y^j &= \frac{e^{\frac{o^j}{\tau}}}{\sum_n e^{\frac{o^n}{\tau}}} \\ o^j &= \text{output from the last layer} \\ n &= \text{size of the vocabulary} \\ \tau &= \text{temperature} \end{aligned}$$

Temperature	BLEU-1	BLEU-4
Deterministic (0)	63.3789	8.1541
0.01	63.3905	8.1875
0.05	63.2730	8.2361
0.1	63.2218	8.2346
0.2	62.9937	7.9593
0.3	62.1144	7.4436
0.4	61.0358	6.9679
0.5	59.8698	6.3310
0.7	55.4127	4.8129
1	44.6519	2.6515
1.5	20.3076	1.1743
2	7.3293	0.5838

Table 3: BLEU scores on Default LSTM model with different temperatures

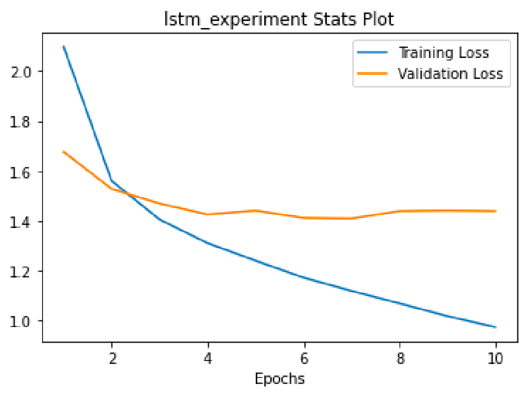
The temperature refers to the stochasticity of the weighted softmax sampling, with lower values (0) as more deterministic and higher values (∞) as more uniform. In general as seen by Table 3, we did not find that one temperature produced noticeably better results. The deterministic model seemed better and more consistent. However, a low temperature stochastic model could produce slightly better results, particularly BLEU-4 scores, than the deterministic model. As we increased the temperature, both BLEU-1 and BLEU-4 scores dropped dramatically. We concluded that this was because increasing the temperature picked words to be generated as captions that had a low confidence produced by the model. It would weight the high probability/more confident words less, and those would therefore be picked less. Since the deterministic approach does not have this problem of weighting high probability words less and less probability words more, since it would simply take the highest probability word, it was more consistent and produced better or similar BLEU-scores. The range that performed similarly to or better than the deterministic approach (0.01-0.1) may be because there was not much influence in the random sampling, and would often still pick the highest probability word with a bit of randomness to pick the second highest which may be more correct.

4 Baseline LSTM vs Best Model

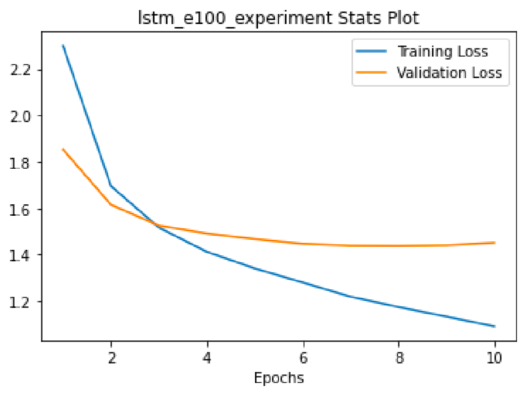
Now, we compare our best tuned LSTM model against the default baseline LSTM model.

The default LSTM uses the default hyperparameters defined previously which again is:

$$\begin{aligned}
\text{image size} &= 256 \\
\text{batch size} &= 64 \\
\text{number of epochs} &= 10 \\
\text{learning rate} &= \text{5e-4} \\
\text{hidden size} &= 512 \\
\text{embedding size} &= 300 \\
\text{max length} &= 20 \\
\text{deterministic} &= false \\
\text{temperature} &= 0.1
\end{aligned}$$



(a) LSTM



(b) Tuned LSTM

Figure 6: Training and Validation Loss Curves comparing Default LSTM vs Tuned LSTM

With the tuned LSTM, we just changed the embedding size to 100. The loss curves are plotted in Figure 6. Changing the hidden size did not help our model performance when we investigated it.

To reiterate, the default baseline LSTM model obtains a test set loss of 1.4210, BLEU-1 score of 63.1867, and BLEU-4 score of 8.1538. The change in embedding size for the tuned LSTM resulted in a test set loss of 1.4375, a BLEU-1 of 64.0168, and BLEU-4 of 8.1465. Although BLEU-4 and test loss degraded very slightly which we attribute to the randomness of the sampling, the BLEU-1 score rose significantly by almost a full point. The training loss for the default model reaches a lower point than for the tuned model meaning the extra embedding size allows it to memorize more of the inputs. The validation loss curve is higher relative to the training loss curve than the corresponding curves for the tuned model indicating that the default model is more prone to overfitting. The reduced embedding size allows the tuned model to generalize better. The loss curve for the tuned model is also smoother showing the stability of training the model and gradient descent.

Another point to notice is that the validation loss decreases until around epoch 4 where it starts to hover near 1.4-1.5. We tuned various hyperparameters to improve this loss. Our hyperparameters consisted of image size, batch size, number of epochs, learning rate, hidden size for LSTM cell, embedding size, maximum length of caption, and temperature for stochastic caption generation. Although a lower image size would make training faster, we maintained an image size of 256 x 256 to prevent loss of data. We found that a lower batch size decreased performance, while a higher batch size slightly improved performance, and ended up keeping the batch size of 64 since the improvement was not stable. We lowered learning rate to 2.5e-4 and increased the number of epochs to 35 to train for longer and prevent more overfitting, but this only improved loss while the BLEU scores did not change much. We found that a larger hidden size noticeably increased overfitting in our training and loss validation curves, so we kept the hidden size to 512. As explained earlier, the embedding size of 100 produced the highest value out of the embedding sizes we tried. We noticed better performance with a lower embedding size, which we set to 100. We maintained a caption length of 20 after analyzing the dataset captions. After testing with deterministic and stochastic approaches with different temperatures, we found that a small stochastic temperature of 0.1 was most consistent and worked best.

5 Baseline LSTM vs Architecture 2

Architecture 2 is similar to the baseline LSTM model in that the core of the architecture is in the LSTMCell that holds long-term information in the cell state and also passes hidden states from one time step to the next. However, in the baseline LSTM and RNN models, the input for the first LSTM step was the image encoded as an embedding that is produced by the ResNet50 pre-trained model. Then all subsequent time steps would feed the word embedding to the LSTM cell of the previous time steps output prediction (or the ground truth output for the previous step in training/evaluating with use of teacher forcing). This is consistent with the idea that the LSTMCell has one set embedding input size that is shared with both the image and word embedding. The twist in Architecture 2 is that each time step LSTMCell receives the encoded image embedding in addition to the the word embedding of the previous output/ground truth word. (It is important to note that the first time step receives <pad> in the architecture2 model so it should learn to output the <start> token). So at each time step, the input passed into the LSTMCell is the image embedding concatenated to the word embedding, producing an overall input size to the cell: word_embedding_size + image_embedding_size. A unique property of this Architecture 2 model is that the image embedding size is able to differ from the word embedding size so these two parameters are independently adjustable for hyperparameter tuning.

First, we tested the Architecture 2 model on the same set of hyperparameters used by the default baseline model:

$$\begin{aligned}
\text{image size} &= 256\\
\text{batch size} &= 64\\
\text{number of epochs} &= 10\\
\text{learning rate} &= \text{5e-4}\\
\text{hidden size} &= 512\\
\text{word embedding size} &= 300\\
\text{image embedding size} &= 300\\
\text{max length} &= 20\\
\text{temperature} &= 0.1
\end{aligned}$$

This produced the following loss curves:

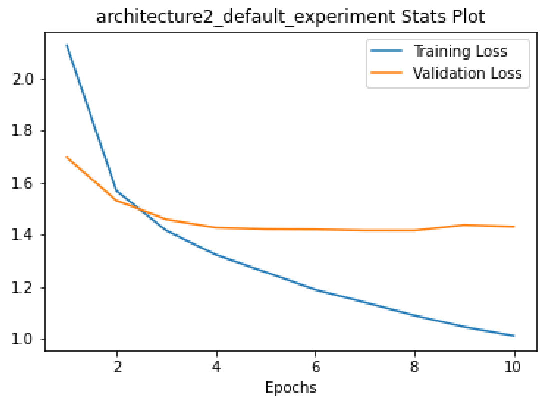


Figure 7: Loss for Default Architecture 2 Model

This model performed quite well on the test set, especially better on BLEU scores compared to the other models seen so far:

$$\text{loss} = 1.4247$$
$$\text{BLEU-1} = 64.5642$$
$$\text{BLEU-4} = 8.7532$$

While this model was clearly better than the previous RNN and LSTM models, we further tuned the hyperparameters, testing different learning rates, embedding sizes (for both images and words), and batch sizes. Tuning on validation loss, we find our best model across all the models in this assignment with the following hyperparameters:

$$\text{image size} = 256$$
$$\text{batch size} = 64$$
$$\text{number of epochs} = 20$$
$$\text{learning rate} = \text{1e-4}$$
$$\text{hidden size} = 512$$
$$\text{word embedding size} = 300$$
$$\text{image embedding size} = 300$$
$$\text{max length} = 20$$
$$\text{temperature} = 0.1$$

We attempted varying the batch sizes, the learning rates, and the embedding sizes independently, but a lot of the best hyperparameters actually overlapped with the default config!

This produced the following loss curves:

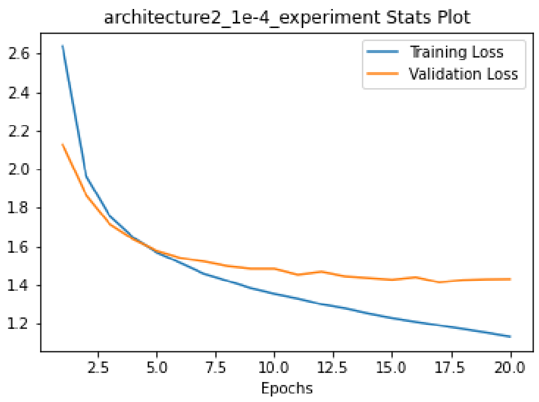


Figure 8: Loss for Best Architecture 2 Model (and Best Model in Assignment)

This model performed the best on the test set, with very high BLEU scores compared to the other models seen so far:

$$\text{loss} = 1.4317$$
$$\text{BLEU-1} = 66.2410$$
$$\text{BLEU-4} = 9.2925$$

Compared to the baseline model, we see an interesting trend where the loss is around the same for this model, but the BLEU scores are much higher for this model. We can first understand this by realizing that the loss is calculated using teacher forcing, so that loss is not necessarily an accurate measure of the overall quality of the model generating captions since generation time does not use teacher forcing. This then reconciles why the loss and BLEU scores are not necessarily moving in the way we expect them to together. So while the test loss may be the same, the model performance can still be better at generating captions.

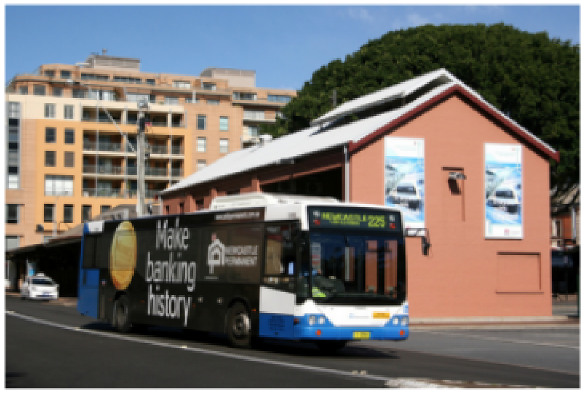
As for the BLEU score performance increases with this best tuned Architecture 2 model, we see the Architecture 2 model perform better than the RNN and LSTM models because this model preserves the long range interaction with the input image much better than the baseline models. By this, we simply mean that the presence of the image embedding present in all inputs of the LSTMCell at all time steps then allows the model to not ”forget” about the input image in a long generation of captions. Through our baseline models, the input image is only used at the first time step input to the LSTM or RNN Cell, so its information passed to the as hidden state or cell state can only be preserved well for short term interactions in the model. As the time steps increase, the model will not remember the original input image features because they are drowned out by additions of multiple hidden states or learned forget gate factors in the LSTM. Then, by concatenating the input image embedding at every LSTMCell entry, we ensure that the model is learning from both past activations, long term interactions, and the original input image, which can then explain this boost in BLEU score performance. A more numerical analysis lends us an increase of about 2-3 in BLEU-1 score and 1-2 in BLEU-4 score when comparing the best Architecture2 model with the best LSTM model. As we can see through these generally accepted metrics for such an image captioning model, our best Architecture 2 model outperforms the best tuned baseline models of the RNN and LSTM.

6 Caption Results

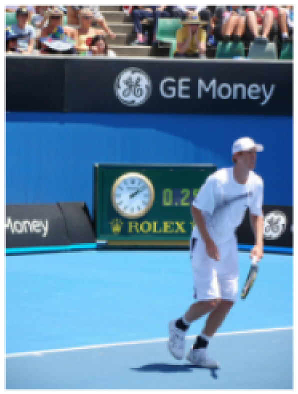
Using the best model described above (tuned Architecture 2 in Figure 8), we can generate some captions for images of the test set. Below, we will show images and their predicted captions that we classify as good and making sense. Then, we will show images that our model performed poorly on.

6.1 Good Captions

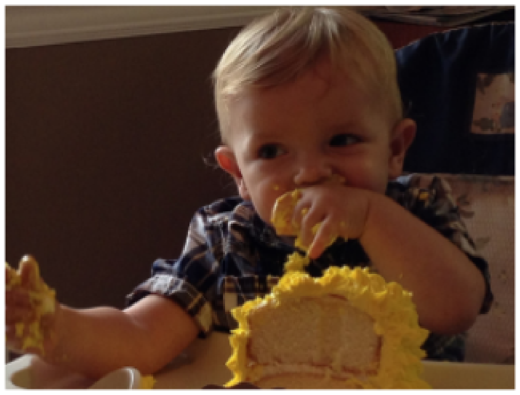
For these good captions, we see that there are some imperfect ones that do not perfectly capture everything happening in the image (missing some details), but they still do a good job to describe the image to the point where we are confident that they make sense and match the image.



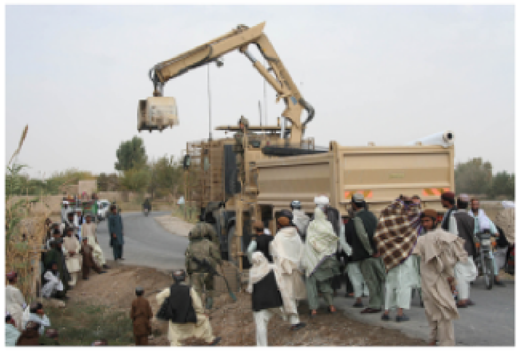
```
Actual Captions:
a large long bus on a city street.
a city bus on the street in front of buildings.
a blue bus traveling down an incline of a busy street.
a city bus with full side advertisement in front of a building.
a public transit bus on a city street

Predicted caption: a bus is parked in front of a building
```



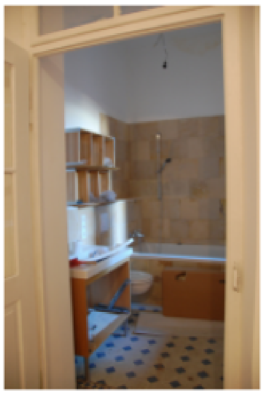
```
Actual Captions:
man with tennis racket preparing to hit ball
a tennis player with racket and a rolex clock behind him
a male tennis player wearing white waits for the ball.
a man is out on a court with a tennis racket
a person on a court with a tennis racket.

Predicted caption: a man is standing on a tennis court holding a racquet
```



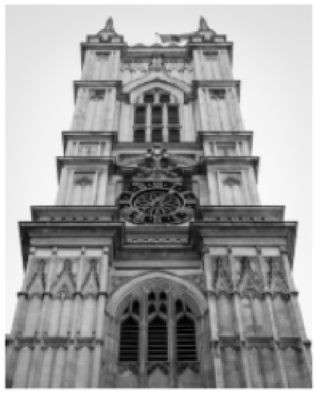
```
Actual Captions:
a baby in plaid shirt eating a frosted cake.
the young is enjoying his cake so much its all over him.
a toddler eats cake with his hands in his high chair.
a toddler is getting messy while eating his cake.
baby boy at the table eating cake frosting off his hand.

Predicted caption: a young boy is eating a piece of cake
```



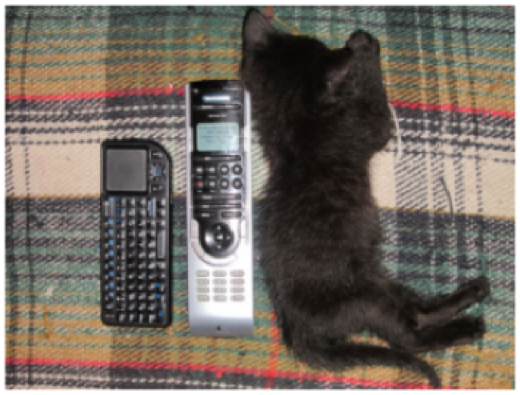
```
Actual Captions:
there is a garbage truck surrounded by people on a road
a machine is digging as a group of people look on.
a crowd of men stand beside heavy equipment on a road.
many men in turbans stand around a dump truck as it works on the side of the road.
people gathered around a construction truck in the street.

Predicted caption: a group of people standing around a large truck
```



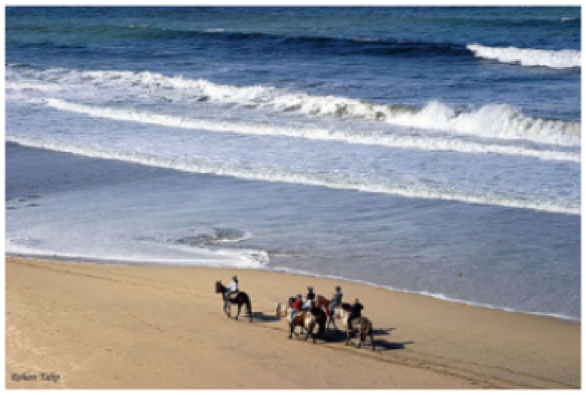
```
Actual Captions:
a bathroom with a tiled floor and a sink.
a bathroom has a sink, toilet, and bathtub in it.
a bathroom with a sink, toilet, and bath tub
a bathroom with tiled walls is being re-modeled.
a look into a bathroom that has a blue and white floor.

Predicted caption: a bathroom with a toilet and a sink
```



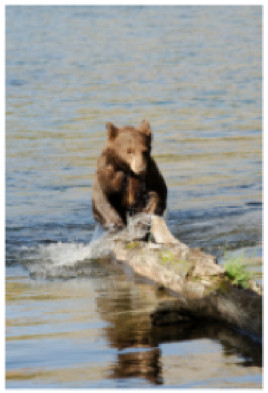
```
Actual Captions:
a historical church has an ornate clock face.
a black and white image of a unique looking building.
a tower that has a clock on it.
a large tall tower with a clock on the top.
black and white photograph of clock tower with windows.

Predicted caption: a large building with a clock on it
```



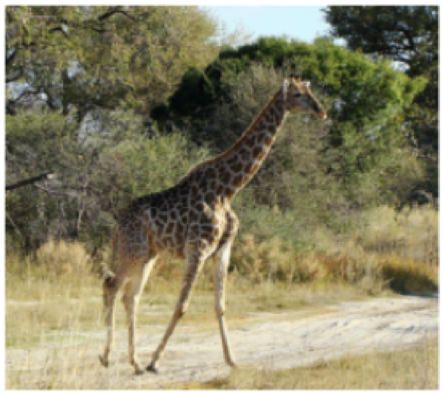
```
Actual Captions:
a black kitten lays on her side beside remote controls.
a black kitten laying down next to two remote controls.
a black cat is lying next to a remote control.
a couple of phones are on a blanket next to a black kitten.
a kitten laying on a bed next to some phones

Predicted caption: a black cat is sitting on a couch with a remote control
```



```
Actual Captions:
a group of people walking across a sandy beach.
people are taking a horseback ride along the beach as the waves roll in.
distance shot of people riding horses on a beach.
a group of people riding horses on the beach.
five people wearing hats horseback riding on a beach.

Predicted caption: a group of people riding horses along the beach
```



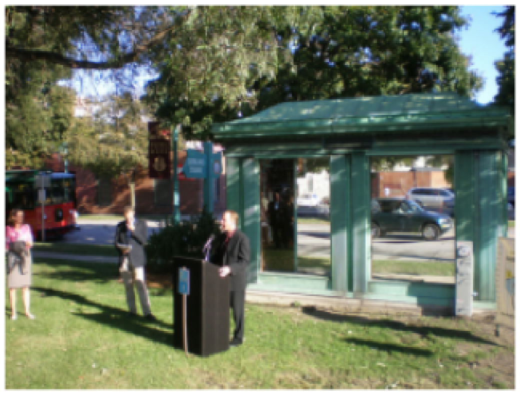
```
Actual Captions:
a bear cub is playing with a log in the water
here is an image of an outdoor place.

a bear that is standing in the water on a log.
a bear standing near a log in the water
a brown bear walking out of a lake on top of a log.

Predicted caption: a brown bear is walking in the water
```



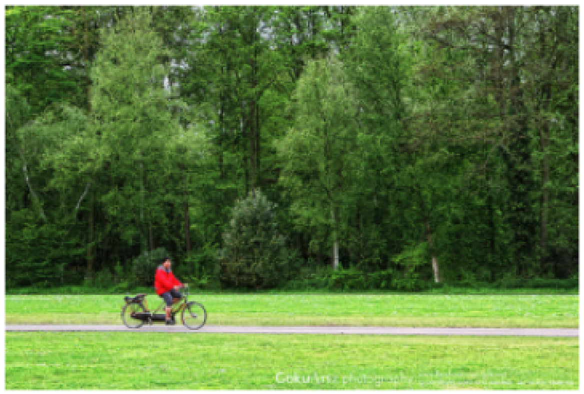
```
Actual Captions:
a giraffe is crossing a dirt road and walking away from the trees.
a giraffe waling in the open. the ground looks dry and arid.
a giraffe strolls through a grassy area between trees.
a giraffe walking across the middle of a trail by itself
a very pretty giraffe crossing a dirt road.

Predicted caption: a giraffe standing in a field next to a tree
```

6.2 Bad Captions

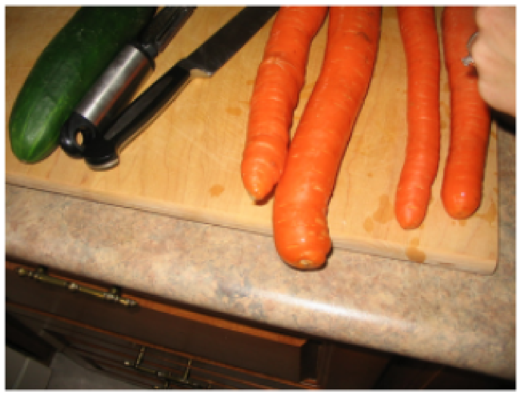
For these bad captions, we see a lot of errors and sentences that do not make much sense. There are cases of repetition in the caption that are clear blunders in the model predictions. We also show cases where the model does a decent job of describing some of the details in the image, but then makes a critical error on a crucial detail like elephant vs. horse. Finally, there are some captions that are just plain wrong with little correlation to the image or major mistakes in the description.



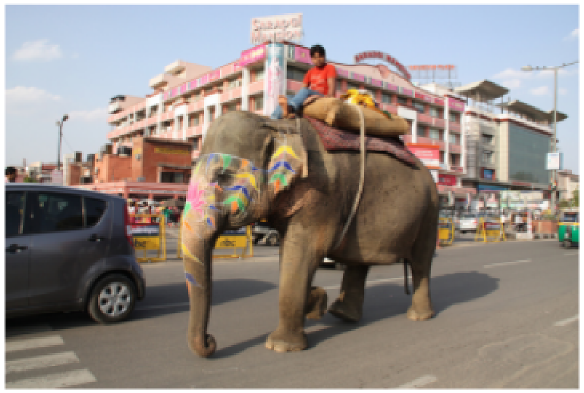
```
Actual Captions:
man speaking from lectern at outdoor gathering in urban area.
a man in a black suit talking at a podium outside.
a man is speaking at a podium in a park.
a man at a podium while a man a woman listen to him.
a man is at a podium and giving a speech.

Predicted caption: a group of people sitting on a bench near a fence
```



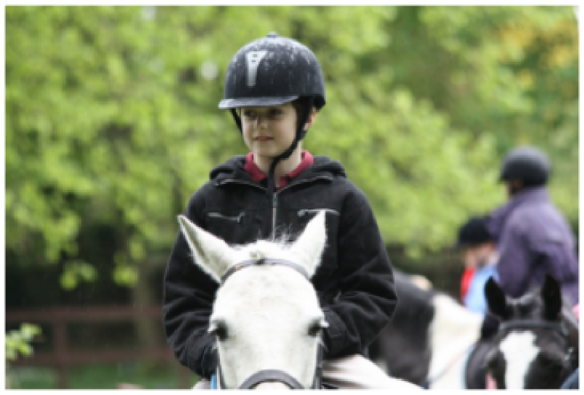
```
Actual Captions:
a man riding a bike through a lush green park.
a young male riding a bicycle near a forest area
single person in red shirt riding a bicycle made for two.
a man riding a bicycle on a path in a park.
one person riding a two seater bike on a path.

Predicted caption: a bike parked in a grassy field with a bike in the background
```



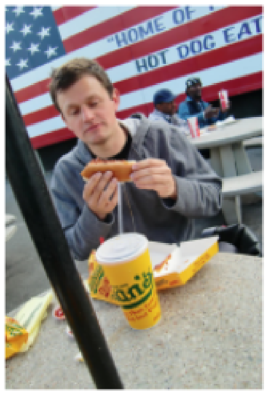
```
Actual Captions:
a group of four carrots on top of a wooden cutting board.
carrots and cucumber on wooden cutting board near knives.
a cutting board with carrots and a cucumber on top.
several carrots and a zucchini on a cutting board with two knives.
a close up of many different vegetables on a cutting board.

Predicted caption: a pair of scissors are sitting on a cutting board
```



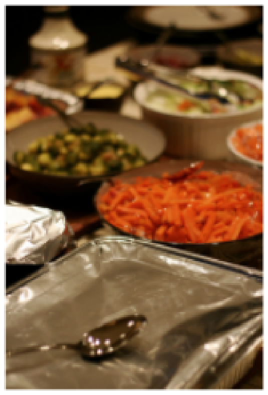
```
Actual Captions:
a man ridding on top of an elephant in a city.
an elephant walking down the street, during the day.
a decorated elephant is ridden through the streets by a man in an orange shirt.
painted elephant walking on roadway with rider in city .
a man is riding on top of an elephant.

Predicted caption: a man riding a horse in the middle of a street
```



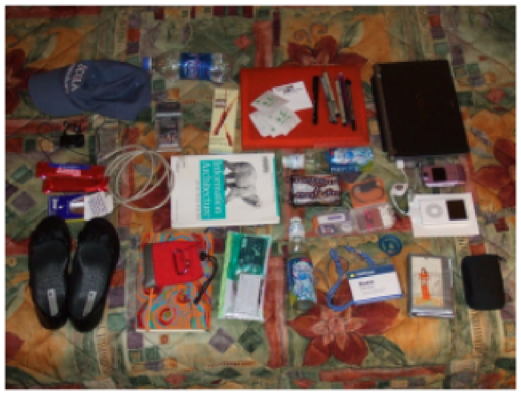
```
Actual Captions:
a little child riding on top of a white horse.
a young helmeted boy rides a white horse.
a young boy in a black jacket riding a white horse.
a young girl riding a white horse in front of other people
a young boy in riding clothes rides a horse.

Predicted caption: a man in a hat and a hat
```



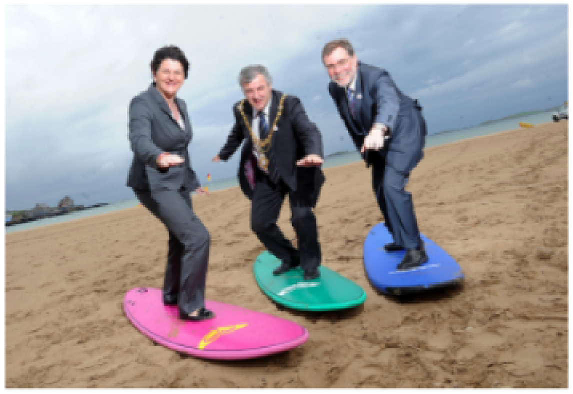
```
Actual Captions:
a man sitting at a table eating a hot dog near an american flag.
a man is holding a large hot dog at a table.
a man at a table with a drink and hot dog.
a man holding a hotdog at a picnic table.
the man is eating a hot dog at the table.

Predicted caption: a woman is eating a doughnut in front of a box of food
```



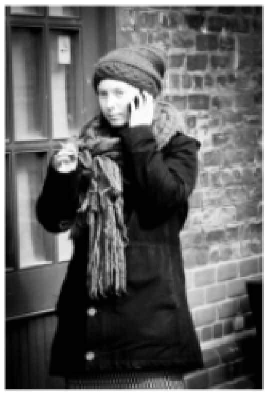
```
Actual Captions:
large spread of many different kinds of food.
some cooked carrots and a spoon on a aluminum pan
many dishes of food, some including, carrots, brussels sprout and more.
various bowls and vegetables on a table ready to be served.
bowls and carrots and broccoli are on a table with a tray and spoon.

Predicted caption: a pizza with a lot of food on it
```



```
Actual Captions:
a floor filled with the contents of a woman's bag.
assorted items sitting neatly on a bed
gathering items to pack for an work seminar.
a assortment of things lined up on a bed including writing utensils.
shoes, a cap, an ipod, a name tag and other items

Predicted caption: a pile of books and a bag on a table
```



```
Actual Captions:
there are there men standing on top of surf boards
three people in suits are standing on top of surf boards, in the sand.
three businesspeople stand on surfboards on the beach
two men and a woman wearing suits on surf boards in sand
three people are posing on surf boards on a beach.

Predicted caption: a man and a woman standing on a skateboard
```



```
Actual Captions:
a young woman holds a cell phone to her ear.
a woman walking down the street talking on a cell phone.
female in coat holding cigarette and talking on cellphone
a smoking women in a scarf makes a phone call.
the woman is walking while talking on her phone.

Predicted caption: a woman wearing a black and white photo of a black and white photo
```

7 Best Model

8 Author's Contributions

8.1 Mitchell Zhang

Implemented testing and worked on testing and tuning the models. Helped with the writeup including abstract/introduction/LSTM/RNN and best models/deterministic/stochastic sections.

8.2 Jun (Moses) Oh

Helped test and tune LSTM/RNN/Arch2 models to find the best hyperparameters. Wrote RNN description and architecture as well as the plots for default hyperparameters + scores vs tuned hyperparameters + scores.

8.3 Nikhil Pathak

In this PA, I wrote, tested, and tuned the Architecture2 Model which included lots of model training time. I then wrote the corresponding section of the report for the Architecture2 Model. I found the best model for the assignment, and wrote the visualizing generated caption section of the report which included ciphering through images to pick good and bad captions. I also conducted the last proofread of the report for final submission.

8.4 Amithab Arumugam

For this project, I wrote the encoder, Baseline, and RNN classes. I implemented get_model and the Config class. I wrote the generate and forward functions plus the helper functions, next_word and generate_caption. I wrote most of the train and val loops. My contribution to training included different epochs, embedding sizes, and hidden sizes for RNN plus a couple Architecture 2 and Baseline models. I also experimented with various data augmentation techniques including random erasing, color jitter, flips, and crops. With regard to the report, I set up the ordering of the sections and revised the abstract, introduction, and related works. I adding some of the plots, tables, and analysis of the baseline vs RNN section.

References/Related Works R4

Below are some pytorch documentation pages that helped implement the PA. Also cited is a paper on BLEU scores which helped us understand how to maintain consistency in reporting the metric and achieve slight improvements to our model evaluation through tokenization choices.

[1] https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html

[2] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

[3] https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html

[4] https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

[5] https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

[6] https://arxiv.org/pdf/1804.08771.pdf

[7] https://pytorch.org/docs/stable/generated/torch.nn.RNNCell.html