High Level Computer Vision Final Project Report

Personalized Image Captioning

Ayan Majumdar *(2571656)*, Rahul Mohideen Kaja Mohideen *(2571768)*
Group 9

***Abstract*—In this project, we are trying to personalize the task of image captioning. Our aim is to automatically generate a caption for the given image with the context of a specific user's vocabulary in previous posts. We test this network on a subset of an Instagram dataset which contains 155k posts from 500 users. We will be using a model called Context Sequence Memory Networks(CSMN) for our task.In this project, we take the network from the original paper [1] and we modify the parameters to see if it can give any improvements in performance.Also, there were no dropout layers in the original network. We have introduced dropout in various positions and compared the performance to the original network.**

I. Introduction

Captioning is the generation of a text that describes a given image automatically. The network has to not only understand what is in the image, but also has to generate a sentence. This project tries to work on the personalization part of the captioning task.

Personalization is the attribute of a captioning model to generate captions like a particular user would. In a sense, we are trying to mimic a user's writing style based on the user's active vocabulary. This is motivated by the fact that when people share photos on social media, writing a caption is usually harder and takes more time than taking the picture[1].

To achieve personalization, we use a model from the original paper [1] called the Context Sequence Memory Network(CSMN). This is a memory network, which incorporates memory cells from which the network uses data to give a long term dependence of the generated words. There are 3 main attributes to this network.

First, we use the memory to store context information from the prior knowledge we have from the dataset. This includes information from the previous posts that the users have written and also image descriptors. Second, we store the previously generated words as a context for generating new words. Consequently, this allows the network to selectively combine the previously generated words and other information. This network doesn't use RNNs because they often fail to take long term dependencies into account. Also, the gradients that were used to predict the current word is not propagated [1]. Third, we use CNN to combine nearby memory cells. This leads to a better understanding of the context.

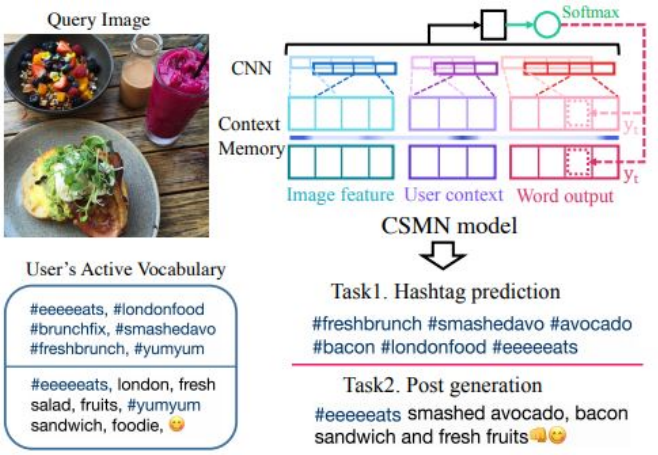


Fig. 1: For a given image, the network generates a personalized caption[1]

The experiments we ran were by tuning the parameters of the original network like the memory size and the number of channels. Also, the original network had no dropout layers. We experimented on adding the dropout layers at various positions in the network.

The original paper [1] used a dataset of 1.1 million Instagram posts from 6.3K users. But we decided to take a subset of the data, which consists of 155k posts from 500 users with the highest number of posts. This was done because the the original dataset was too large and training took too long to complete, so we deemed it appropriate

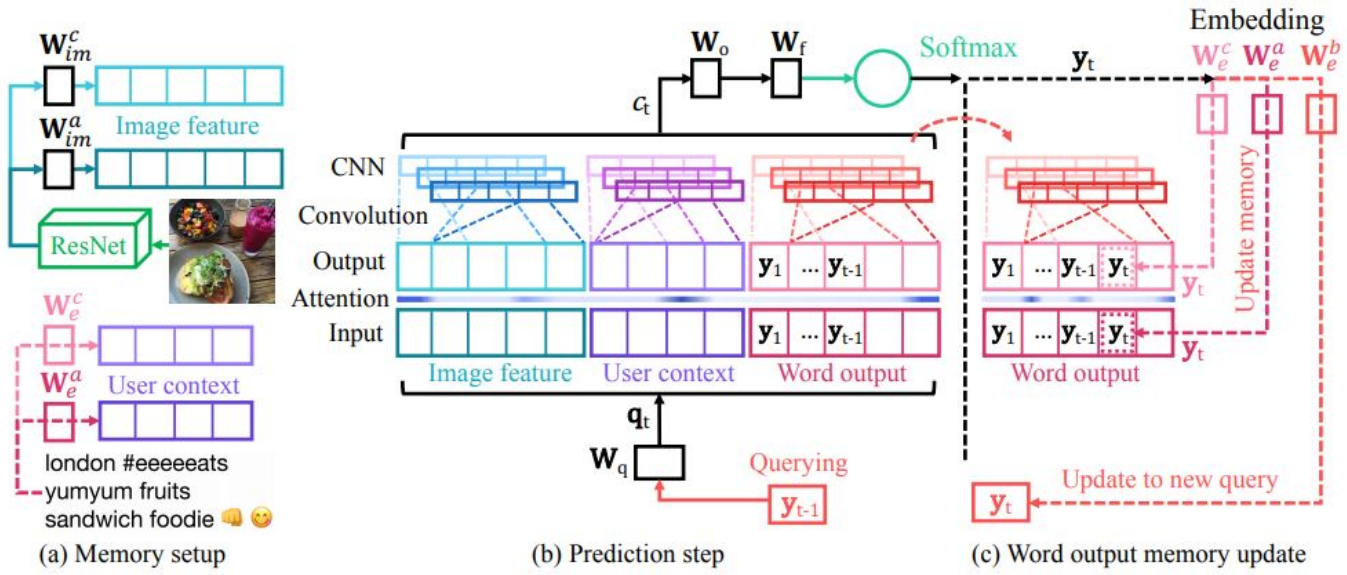


Fig. 2: The CSMN model from the original paper[1]

to just work with this subset. Our baseline is the original network's performance on our subset of the dataset. We do evaluation with metrics such as BLEU, CIDEr, METEOR and ROUGE.

II. Related Work

The network model, the code and the dataset is from the paper, Attend to You: Personalized Image Captioning with Context Sequence Memory Networks[1] by C.C.Park, B.Kim and G.Kim. We use the paper in [1], the corresponding code available on GitHub as our starting point. We also use the code used in [2] for understanding the use of dropout in a similar usage of CNNs on text data.

III. Context Sequence Memory Network

The model is based on a novel Context Sequence Memory Network as shown in [1]. It uses the image features as well the text vocabulary used by specific users, and use CNNs for the sequence generation.

A. Context memory

The network stores 3 types of context information, *image representation, the user's active vocabularies and the previously generated words*. The images are represented using ResNet101 trained on the ImageNet 2012 dataset. A 7x7 res5c feature map is used when the model considers spatial dependencies, otherwise it uses pool5 features[1]. Here, we use the pool5 features as mentioned in [1] they gave a better performance overall.

To build the user context memory, a TF-IDF score is used and the words with the D highest scores are selected for each user to build $\{u_i\}_{i=1}^{D}$. The TF-IDF scores takes care to only include words that are specific to users so that the network can use this active vocabulary . The user context memory $m_{us}^{a/c} \in \mathbb{R}^{1024}$ then becomes

$$u_j^a = W_e^a u_j, u_j^c = W_e^c u_j; y_j; j \in 1, ..., D \quad (1)$$

$$m_{us,j}^{a/c} = ReLU(W_h[u_j^{a/c}] + b_h) \quad (2)$$

where the superscript a/c denotes input/output. Here u_j is the j-th active word. W_h remains constant for both input and output memory, and $W_e^{a/c}$ is learned.

The word output memory stores all the previously generated words $y_{t-1}, ..., y_1$ and it is updated whenever a new word is generated. The word output memory vector is calculated as

$$o_j^a = W_e^a y_j, o_j^c = W_e^c y_j; j \in 1, ..., t-1 \quad (3)$$

$$m_{ot,j}^{a/c} = ReLU(W_h[o_j^{a/c}] + b_h) \quad (4)$$

where y_j is the j-th previous word. The same values are used for W_e^c from Eq.1 and the values of W_h and b_h from Eq.2, and we keep updating $m_{ot,j}^{a/c}$ when a new word is generated. In the end, we string together all these 3 contexts to give a final memory vector, whose size is the sum of the all the above mentioned contexts.

B. Sequence Generation

We predict the next words by storing previously generated words sequentially. We first generate an input vector q_t from the previous word y_{t-1} then see how much this vector matches with the final memory vector by doing a matrix multiplication and a softmax.

$$q_t = ReLU(W_q x_t + b_q), x_t = W_e^b y_{t-1} \quad (5)$$

The parameters $W_e^b \in \mathbb{R}^{512\times V}$ and $W_q \in \mathbb{R}^{1024\times 512}$ are learned. After giving q_t to the attention model, we get

$$p_t = softmax(M_t^a q_t), M_{o_t}(*, i) = p_t \circ M_t^c(*, i) \quad (6)$$

We can also see this as, the matrix multiplication tells which parts of the input vector are important in the context of the memory vector. Then we rescale the output memory vector according to the obtained weights with element-wise multiplication($\circ$) which also has 3 parts as mentioned in the previous subsection.

Next, a CNN is applied to the output memory vector. 3 separate filters are defined with window sizes of h=3,4 and 5 with depth 300. A convolutional layer and a max-pool layer is applied to each memory type.

$$c_{im,t}^h = maxpool(ReLU(w_i m^h * m_{im,1:49}^o + B_i m^h)) \quad (7)$$

where * is convolution. We give $b_i m^h \in \mathbb{R}^{49\times 300}$ and the filters $w_i m^h \in \mathbb{R}^{[3,4,5]\times 1024]\times 300}$ After the max-pool, we get a 300x1 vector for each window, which we combine together. This is repeated for all 3 memory types and the obtained vectors are concatenated. Finally, we get a 2700(3x3x300)x1 vector which we can call as c_t.

A hidden state is calculated by multiplying a weighting matrix with C_t and adding a bias. The output probability s_t is calculated over all the vocabularies by multiplying another weighting matrix followed by a softmax.

$$h_t = ReLU(W_o c_t + b_o) \quad (8)$$

$$s_t = softmax(W_f h_t) \quad (9)$$

where $W_o \in \mathbb{R}^{2700\times 2700}$ is the weight matrix and the bias is $b_o \in \mathbb{R}^{2700}$ Finally, we select the word that has the highest probability in s_t and present it as the output word. We repeat this process until we encounter the EOS symbol, and the obtained word is fed into the output memory for predicting the next word.

C. Training

For training, to provide the memory state for prediction, a teacher forced learning was used. For every time step, to estimate the loss, the softmax cross entropy loss was used and all the parameters were initialized randomly.

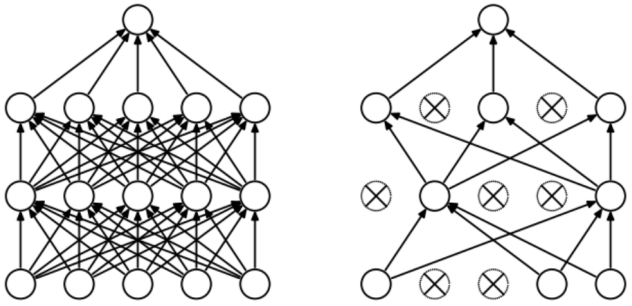
An Adam optimizer was used with $\beta_1 = 0.9$,$\beta_2 = 0.999$ and $\epsilon = 10^{-8}$ along with mini-batch stochastic gradient descent with batch size 200 for each GPU[1]. And the learning rate was set as 0.001. In all our experiments, we train the model for 70k steps.

IV. Proposed Modifications

In order to try and improve the current model, we attempted to increase the number of parameters, and as a result, the capacity of the model. But, as we will discuss later, it seemingly led to over-fitting on our smaller training set. Hence, we decided to add regularization to the CSMN model. The current model described by [1] does not have any regularization technique built-in. The choice of selecting dropout regularization was obvious. It is one of the most popular regularization techniques today, as it not only approximates an ensemble method, but also allows the network to have far more parameters, and let the network learn by itself how to make the neurons good against overfitting. We took inspiration from [3] and got the ideal location to add the dropout layer, as will be explained in Section VI. Also, according to [4] we tried different dropout rates, which is a hyperparameter to achieve a better score.

We had to take a subset of the training data, and our selection resulted in a lot more test data images than was there in the original test file, considering the fraction of the original training data we used.

After testing, as we will be discuss shortly, we saw that higher number of CNN channels, and higher memory dimensions can be used in the network along with dropout regularization, and that can indeed give a performance boost, at least when we have less training data, as in our case.



(a) Standard Neural Net (b) After applying dropout.

Fig. 3: Effect of dropout in a fully connected model

A. Number of CNN channels

The number of CNN channels describes the depth of each CNN layer, and the greater this is, the more complicated features it can learn. The initial model used 300 depth, and 3 different layers with 3 different filter sizes of 3,4, and 5. Increasing this number of channels should allow each CNN layer to learn more parameters about each of the image, user context text and output text memories. We tried experimenting with this value in this work.

B. Memory Dimension

This indicates the size of the memory cell that is used internally to represent the context data. If we increase the dimension of this memory cell, then the network should also be able to have a richer memory representation. We tried to increase this number and see the effect as well.

C. Dropout

We shortly introduce the idea of dropout in this subsection. As described in [5] dropout provides a method to regularize a broad *family* of models that is both effective and computationally efficient. With a base network, after we apply dropout, we stochastically turn off particular neurons with a probability of p. During training time, using mini-batch learning technique, each batch is fed into one of the subnetworks generated. In this way we train an ensemble of different models. During test time, we scale the neuron outputs where dropout was applied to approximate the fact that they were dropped out during training time. As observed in [6] experimenting with the dropout rate is important as the optimal value of the hyperparameter can vary depending on the data and model. Dropout became an important part of our experiment as we tried increasing the number of parameters of the network as mentioned above.

V. Evaluation Metrics

As with [1], we used different scores in order to provide a quantitative result of our experiments. Each scoring metric has its own unique advantages. The evaluation metrics used were:

- BLEU_1,2,3,4: The BLEU score [7] allows a modified precision computation of a machine generated text and a reference text at n-gram level. It has been a very popular metric for machine translation, and we use it here as well, using for unigram, bigram, trigram and 4-grams.
- CIDEr: This metric [8] allows for a consensus based image description score. It performs tf-idf weighing of the n-grams and then compares the occurrence of the same in reference sentences.
- ROUGE_L: The ROUGE_L score [9] helps to measure the longest matching sequence of words using the LCS or Longest Commmon Subsequence metric. This scoring allows to capture sentence level matches and word orderings, and also includes longest common sequence in the matches, and hence does not require a specific n-gram length to be predefined.
- METEOR: This score [10] is computed based on different alignments between the predicted text and reference text, and at the same time also allows the use of stemming and synonymy matching, along with exact word matching as found in the other scoring metrics.

VI. Experimental Results

A. Data selection

As part of our experiments we first had to decide on a subset of the dataset provided. The original dataset is comprehensive, but unfortunately too big to be trained on just one GPU till optimal parameters are learnt. Hence, we decided to take a subset of the dataset that could be trained in a reasonable time on a GPU. Hence, we decided to go through our entire dataset and select the top 500 users based on their total number of Instagram posts. Then, we made a 90-10 split of the number of images for

each user, 90% being used for training, and 10% for testing purposes. Hence, after the split, we get about 155K training images in total and 16.9K test images.

B. Task

The original paper used the same network to perform two tasks, caption generation and hashtag prediction. But we concentrated on the sole task of caption generation as the captions can also contain hashtags inside, as the hashtags are also learnt into the vocabulary. Hence, while the network makes predictions for captions, it can also predict hashtags if it finds it relevant. Hence, given a user's image, it will try to use the user's vocabulary to predict the caption.

As mentioned earlier, we reduce the number of steps from 500K with batch size of 400 on 4 GPUs to 140K with batch size of 200 on 1 GPU. If we had to go for the same number of *effective steps*, we would have had to go through a lot more steps on a single GPU. Hence, this was done to reduce the training time, and to be able to perform more experiments.

C. Parameters

We chose certain parameters that we thought should be relevant to change and see the effect of the same on the caption generation. These include:

- Number of channels of the CNN layer
- Memory cell size

The network currently uses no specific regularization technique. The use of CNNs for text was inspired from the paper [3]. This work involved using dropout regularization, that allowed to increase the capacity of the actual network. So, along with the parameter variations, we also try to introduce the dropout regularization in the network.

But, it was important to experiment with the dropout position as well. We tried adding the dropout at two positions in the given network:

- After the layer in the network where they generate a fully-connected layer by multiplying with the weight matrix W_f.
- After the text features from the CNN layers is generated and concatenated as `h_pool_flat`.
- The image features are also flattened by passing them through a fully-connected layer with weight matrix W_{pool5}. After the concatenated text features `h_pool_flat` and the flattened image features are added and passed through another fully-connected layer, we add a dropout here. To implement dropout, we use the `tf.layers.dropout` module. We set the `training` flag accordingly to ensure that the dropout layers are not considered during evaluation task. We also set the rate of dropping a neuron at 40%, although for one experiment we tried to reduce it and see the effect. To understand how dropout was used in [3], we referred to an implementation of the project in TensorFlow, that was provided by [2].

D. Outcomes

Here we provide the results/outcomes of the experiments we attempted with the given model.
1) Varying number of channels: We try to see if we can get any improvement by increasing the number of channels in the CNN layers. The original model had 300, and we try to increase it to 400. As seen in Table I when we simply increased the number of CNN channels from 300 to 400, we did not see much difference in the scores. In fact the scores actually reduced by a very small amount. So just increasing the parameters in the CNN channels did not help improve the model.

Number of channels	Original(300)	400
Bleu_1	0.0447	0.043
Bleu_2	0.0157	0.015
Bleu_3	0.008	0.0078
Bleu_4	0.0048	0.0047
CIDEr	0.066	0.064
METEOR	0.0292	0.029
ROUGE_L	0.0679	0.0682

TABLE I: Varying number of CNN channels

Experiment	400 channels	400 channels +Dropout(40%)	400 channels +Dropout(30%)
Bleu_1	0.043	0.041	0.0367
Bleu_2	0.015	0.0146	0.0134
Bleu_3	0.0078	0.0075	0.007
Bleu_4	0.0047	0.0046	0.0045
CIDEr	0.064	0.0648	0.0625
METEOR	0.029	0.0296	0.028
ROUGE_L	0.0682	0.0763	0.0716

TABLE II: Adding dropout at last FC layer

2) Dropout before softmax layer: We see if adding dropout after the fully connected layer that is before the final softmax layer helps. Again as per Table II, we see when we added dropout in the FC layer just before the last softmax layer, the scores reduced despite keeping a dropout probability of 0.4. When we reduced it to 0.3, the scores reduced even further. This showed that we indeed needed to have dropout but we needed to find the optimal position in the network, and the optimal dropout rate.

3) Dropout after `h_pool_5`*:* Now, we see if adding dropout after the text features are extracted and flattened through a fully connected layer improves the scores. As described before, this was influenced by [3] and [2]. However as we see from Table III the scores reduce drastically if we try to add dropout here. This shows that by just dropping out text features we are reducing the textual features too much. Hence, again adding dropout here is not the optimal choice. Also, it might be that we do not have enough memory channel parameters, and adding dropout is losing out too many features, especially given the smaller dataset we are working on.

Experiment	1024:400	1024:400+Dropout(40%)
Bleu_1	0.043	0.0364
Bleu_2	0.015	0.0136
Bleu_3	0.0078	0.0072
Bleu_4	0.0047	0.0045
CIDEr	0.064	0.0631
METEOR	0.029	0.0278
ROUGE_L	0.0682	0.0713

TABLE III: Adding dropout after `h_pool_5`

4) Varying the memory cell dimension: We try to find out the influence of the memory cell size. For this experiment we remove the dropout layers, and keep the number of CNN channels as 400, and try to increase the memory dimension from 1024 to 2048 and try to see the result. As seen from the results in Table IV, simply increasing the memory dimension again reduced the performance of the model. Scores across the board dropped by a small bit.

5) Dropout after image feature and text feature addition: Finally, we see if adding a dropout after the image features are actually added to the text features changes the results. For this experiment, we maintained the higher number of parameters, so the number of CNN channels were 400, the memory dimension was 2048. From the results of Table V we clearly see that adding dropout at this position increased the model performance vastly. Also, dropout rate of 0.5 gave the better results. At that dropout rate, with respect to the original configurations on the smaller dataset, we see BLEU_2 scores increase by 42%, BLEU_3 by 38.75%, BLEU_4 37.5%. Other scores like METEOR increased by 25%, ROUGE_L by 27.8%.

Experiment	Mem_Dim=1024	Mem_Dim=2048
Bleu_1	0.043	0.0416
Bleu_2	0.015	0.0144
Bleu_3	0.0078	0.0073
Bleu_4	0.0047	0.0043
CIDEr	0.064	0.0607
METEOR	0.029	0.0281
ROUGE_L	0.0682	0.0664

TABLE IV: Varying the memory cell dimension

E. Caption Prediction

Based on certain query images and the corresponding users who had posted them, we try to see what the network predicts, and how it is compared to the ground truth. Figure 4 shows the network predictions and the corresponding ground truth captions for four query images. As we can see, for the images 1,2 and 4 the network is able to predict many of the words the corresponding users had used actually. This shows that the network is able to learn the image features and able to correspond those to the user vocabulary and context words. Even in the third image, even though the network failed to actually predict any of the exact words, it did understand the context and provided a prediction accordingly. This result also shows an interesting property of the network: it is able to predict hashtags while generating captions. This is because hashtags are also stored as part of the vocabulary, and if the network is able to relate any of those hashtag features to the image, it will predict them in the context.

We also attempt to show the personalization aspect. Although it is a bit difficult to show outright, we try to show it by selecting the same query images, and instead of the actual user, feed the network other users' context words. Hence, the network should be able to relate the image features to

Experiment	Original(300:1024)	400:2048	(+) Dropout@40%	(+)Dropout@50%
Bleu_1	0.0447	0.0416	0.0553	**0.0614**
Bleu_2	0.0157	0.0144	0.02	**0.0223**
Bleu_3	0.0080	0.0073	0.01	**0.0111**
Bleu_4	0.0048	0.0043	0.006	**0.0066**
CIDEr	0.0660	0.0607	0.0793	**0.0823**
METEOR	0.0292	0.0281	0.035	**0.0365**
ROUGE_L	0.0679	0.0664	**0.0882**	0.0868

TABLE V: Adding dropout after combination of memory image features with text features

Fig. 4: Caption Prediction vs Ground Truth (num_channel:400, memory_size:2048, dropout:0.5)

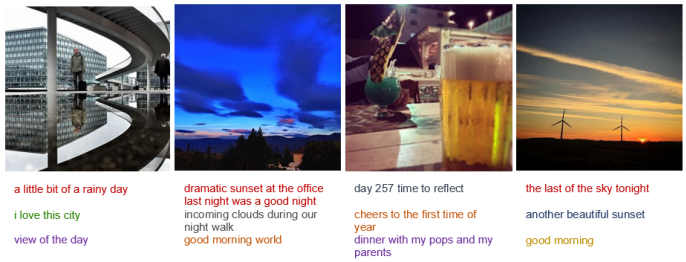


Fig. 5: Predictions based on Users (num_channel:400, memory_size:2048, dropout:0.5)

those context words that are closest to those features and use those words of the particular user. We show the output in Figure 5. We represent different users with different colors. As seen, when using a user's vocabulary, we get different predictions. If we compare these to the actual network predictions, or, in fact the ground truth given in Figure 4, we see that they are indeed different. This captures the fact that different users would in fact describe each image differently, given each of their individual ordered list of commonly used words. Even then, if we compare the different user predictions, we can see that they are still relevant to the actual query image.

VII. Conclusion and Future Work

As part of our project we have shown that the CSMN network is indeed able to provide personalized captions provided query images, by using the particular user's context words. We also performed several experiments, and concluded that simply attempting to increase the number of parameters in the network did not help in giving better performance. Hence, using dropout regularization was quite important. But at the same time, we see that adding dropout at the correct position is also very important, along with setting the dropout rate appropriately. We found that using dropout after the image memory features were added with the textual memory features and passed through a FC layer provided the best results, along with dropout rate of 0.5 or 50%. Although our evaluation scores were not comparable to the actual scores reported by the authors of [1], after we applied the same network and parameters as in [1] on our smaller dataset, we can see that indeed using dropout along with increased number of parameters lead to a much better score and performance. Hence, dropout does allow the network to work with a lot more parameters, but at the same time not over-fit.

However, there is a lot of further work that can be done. We, of course, have to see if this increased number of parameters and dropout works positively when working on the full-size dataset of 800K images. We can perform more experiments on applying dropout in other places, including in the CNN layers and observe the performance. We also have to experiment more with multiple dropout layers, and with the dropout rates, and the other network parameter values along with this dropout regularization.

We should also try and see if the same goal can be achieved by using a more traditional sequential model like an LSTM-RNN model instead of using CNNs. However, integrating the vocabulary system in the sequential model is a great challenge. We could try to see if we can try to do it using some adversarial method.

We can also try to add more dimensions of personalization into the model, e.g. using the particular user's social media profile information, face-tagging, location tagging, temporal tagging using the time and date information, etc. Simply looking through the users' images and the context will not give us that information, and using these additional dimensions can indeed help the model in providing more relevant captions.

Finally, we can try to tackle the personalized captioning task as a style transfer task, and compare the results of that to using the current CSMN model.

REFERENCES

[1] C. C. Park, B. Kim, and G. Kim, "Attend to you: Personalized image captioning with context sequence memory networks," 2017. [Online]. Available: https://github.com/cesc-park/attend2u

[2] Britz, "Convolutional neural network for text classification in tensorflow." [Online]. Available: https://github.com/dennybritz/cnn-text-classification-tf

[3] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[6] A. Budhiraja, "Dropout in (deep) machine learning," Medium Blog Link.

[7] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ser. ACL '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 311–318. [Online]. Available: https://doi.org/10.3115/1073083.1073135

[8] R. Vedantam, C. L. Zitnick, and D. Parikh, "Cider: Consensus-based image description evaluation," *CoRR*, vol. abs/1411.5726, 2014. [Online]. Available: http://arxiv.org/abs/1411.5726

[9] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004. [Online]. Available: http://www.aclweb.org/anthology/W04-1013

[10] A. Lavie and A. Agarwal, "Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments," in *Proceedings of the Second Workshop on Statistical Machine Translation*, ser. StatMT '07. Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, pp. 228–231. [Online]. Available: http://dl.acm.org/citation.cfm?id=1626355.1626389