Convolutional Image Captioning

Davin A. Hill[1§], Ankush Hore[2], Do H. Hwang[1], Krishna D. B. Anapindi[1,3]

[1]Department of Statistics, University of Illinois at Urbana Champaign, Champaign, Illinois, 61820
[2]Department of Mathematics, University of Illinois at Urbana Champaign, Urbana, Illinois, 61801
[3]Department of Chemistry, University of Illinois at Urbana Champaign, Urbana, Illinois, 61801

[§] corresponding author: Davin A. Hill
Email: davin2@illinois.edu
Sample Images: https://krishnaanapindi.wixsite.com/convcap
D.A.H(davin2), A.H (ahore2), D.H.H (hwang101) and K.D.B.A (anapind2) contributed equally towards this work.

Table of Contents

Abstract

Image captioning, which refers to being able to generate descriptive sentences from an image is one of the most interesting problems in Artificial Intelligence right now. A successful image captioning model not just needs to identify the objects in the image but also come up with an accurate description of the relationship between them. This poses a formidable challenge and the current state-of-the art models address the problem by combining the best of computer vision and natural language processing. In our current work, we discuss one such models, Convolutional Image Captioning, which uses convolutional neural networks for machine translation and conditional image generation. In addition to the existing experiments discussed in this manuscript, we further attempted to expand our knowledge on the performance of this model by trying various hyperparameters and critiqued on their influence on improving the results.

1. Introduction

Significant progress has been made in solving the problem of image captioning in recent years. Particularly, models based on deep recurrent neural networks (RNNs) powered by Long-Short-Term-Memory (LSTMs) and Gated-Recurrent Unit (GRUs) have shown promising results.[1] However, these RNN based approaches have a complex architecture that is sequential in nature and usually has a higher memory requirement. Due to these constrains, there has been a recent surge in the interest for alternate architectures that do not suffer from such similar drawbacks. Particularly, convolutional neural network (CNN) based models have shown great promise in recent years where they have been successfully implemented on tasks such as conditional image generation,[2] machine translation[3] and neural caption generation.[4] In the current work, Aneja et al.,[5] have implemented the CNN architecture based on the previous work of Gehring et al.,[3] on Convolutional sequence to sequence learning. Briefly, the model has three main components- 1) Input word embeddings, 2) Masked convolutions and 3) Output word embeddings. The masked convolutions are the equivalent of an LSTM or GRU unit in RNNs. However, instead of unrolling the model in time that can be described by $p_{i,w}(y_i \mid h_i, I) = g_w(y_i, h_i, I)$ and $h_i = f_w(h_{i-1}, y^*_{i-1}, I)$ (where p is the probability of predicting a word given the hidden state and input image and g is a non-linear function; a deep neural net that computes the probabilities from the given inputs), the CNN based approach uses a feed-forward deep net (f_w)that can be described by $p_w^{(i)}(y^{(i)} \mid y^{(<i)}, I) = f_w(y^{(i)}, y^{(<i)}, I)$ where the prediction (p) of a word only depends on the previous words. To ensure that the model only 'sees' the past data, a masking operation (referred to as masked convolutions) was performed on all the future data for a given time point.

2. Model Architecture

Learning

Given the dataset $D = \{(I^{(j)}, C^{(j)})\}_{j=0}^{m-1}$ consisting of images I and captions $C = \{y^{(i)}\}_{i=0}^{n-1}$ consisting of word tokens $y \in \mathcal{Y}$, we want to minimize the loss function:

$$L(\theta) = \min_{\theta} \frac{1}{nm} \sum_{I^{(j)} \in D} \sum_{y^{(i)} \in C} -\log(P(\hat{y}^{(i)} | \, y^{(i-1)}, y^{(i-2)}, \dots, y^{(1)}, I^{(j)}; \theta))$$

where k is a hyperparameter for the convolution kernel size, determining the number of previous words to consider, and $\hat{y}$ is the predicted word token. Note that during training, the predicted word $\hat{y}^{(i)}$ at each timestep is generated given the previous ground-truth words $y^{(<i)}$, which decouples the time sequence for word predictions and allows us to train each word in parallel

To obtain a feature representation of I, we use a convolution neural network pretrained on the ImageNet dataset[6] and modify the final fully-connected layer to output a 512-dimensional image embedding vector. We then concatenate the image embedding with a 512-dimensional word embedding vector for each word, which is trained from scratch from a one-hot-encoded vocabulary. This forms an embedding matrix $E \in \mathbb{R}^{1024\,x\,n}$ which is fed into a series of 1-dimensional convolutional layers. In order to prevent the flow of future information, that is, $\hat{y}^{(i)} \mid y^{(\geq i)}$, we mask the convolutions using zero-padding of size $k-1$ and then truncate the output by $k-1$. The output $E^{(\ell)}$ of convolution operation $K^{(\ell)} * E^{(\ell-1)}$ at layer ℓ is therefore defined as:

$$E_{p,i}^{(\ell)} = \sum_{p\prime=0}^{1023} \sum_{r=0}^{k-1} K_r^{(\ell)} \, E'^{(\ell-1)}_{p',i+r}, \quad \text{s.t. } E'^{(\ell-1)} \in \mathbb{R}^{1024 \times [n+2(k-1)]}$$

After truncation, we apply a Gated Activation Unit (GLU)[7] activation function at every layer, along with dropout and residual connections. Following the convolution layers, we use a series of fully-connected layers to reshape the output such that it matches our predefined vocabulary size. Using a Softmax function, this output can then be converted to a probability distribution over our entire vocabulary for each word in the caption.

Attention

The use of attention provided an increase in test accuracy for,[8] therefore we similarly implemented attention layers in our model for testing. The attention layers add additional trainable parameters to our convolution captioning model by using the image feature output, represented by the last convolution layer from the image encoder model. We define the image encoder features as $C_{p,m,r}$, with p channels of $\mathbb{R}^{m \times r}$ image features. The raw attention scores for word y_i can then be defined as the following dot product:

$$a_{i,m,r}^{(\ell)} = \sum_{p} E_{i,p}^{(\ell)} \, C_{p,m,r}$$

We then transform the attention scores to a probability distribution by applying a Softmax function:

$$A_{i,m,r}^{(\ell)} = \frac{\exp(a_{i,m,r}^{(\ell)})}{\sum_{m,r} \exp(a_{i,m,r}^{(\ell)})}$$

We apply these attention scores to the convolution output using another dot product, resulting in a p-dimensional output vector for each word y_i:

$$output_{i,p}^{(\ell)} = \sum_{m,r} A_{i,m,r}^{(\ell)} \; C_{m,r,p}$$

Since $A_{i,m,r}^{(\ell)} \in [0,1]$, these attention scores act similarly to gate for the image encoder features, limiting the amount of information that each word receives from each p-dimensional feature vector.

Inference

During inference, each word in the sentence is predicted sequentially using a separate forward pass of the trained model. Mathematically, the probability distribution of each predicted word $\hat{y}^{(i)}$ is calculated as $P\left(\hat{y}^{(i)} \mid \hat{y}^{(i-1)}, \hat{y}^{(i-2)}, \ldots, \hat{y}^{(1)}, I; \theta\right)$, where k is the kernel size of the 1-d Convolution and I is the input image. At each timestep, we sample the most probable previous word with argmax, then feed all previous words through the model to generate a distribution for the subsequent word. This process is terminated once either the end-sentence token is sampled or the maximum caption length is reached.

Beam Search

As mentioned above, during inference we predict a word at time step i based on the previous predicted words. Therefore, we simply maximize $p(\hat{y}^{(i)} \mid \hat{y}^{(i-1)}, \ldots, \hat{y}^{(1)}, I; \theta)$. However, there may exist $\hat{y}^{(i)'}, \hat{y}^{(i-1)'}, \ldots, \hat{y}^{(1)'}$ such that

$p(\hat{y}^{(i)'}, \hat{y}^{(i-1)'}, \ldots, \hat{y}^{(1)'}) > \prod_i p(\hat{y}^{(i)} \mid \hat{y}^{(i-1)}, \ldots, \hat{y}^{(1)}, I; \theta)$. Hence, this is a greedy method for prediction. At the same time, finding the optimum $\hat{y}^{(i)'}, \hat{y}^{(i-1)'}, \ldots, \hat{y}^{(1)'}$ is an intractable problem. To overcome this problem, we use the beam search method.[1] It is a polynomial time greedy algorithm. Although this algorithm is also greedy, it explores more possibilities than the standard greedy inference. Although polynomial, its complexity increases with the hyper-parameter called beam size, with beam size of 1 being equivalent to the standard inference. In very brief, a beam size of n indicates that we maintain n possible combinations of words till the (i-1)th step. At the ith step, we feed in the n words from (i-1)th step, and get n new words for the ith step. Now, of the n^i choices of phrases of length i, we choose the top n of them using breadth-first search on the cumulation log probabilities.

Our implementation of beam search uses the class **beamsearch** from the following repository:
https://github.com/aditya12agd5/convcap

3. Data Preprocessing

Our experiments were conducted on the MSCOCO 2014 dataset,[9] using training and validation splits of 117287 & 5000 images, respectively. Each image is associated with 5 ground truth captions. During preprocessing we used the NLTK package to tokenize each caption in order to generate our one-hot-encoded vocabulary. This process resulted in 29560 unique tokens, with a median caption length of 11 tokens. During training we truncated this vocabulary to 9221 tokens, including punctuation, and added 4 tokens for start/stop/unknown/mask indicators.

For each batch of data during training, the ground-truth captions are tokenized using NLTK and the start/end tokens are prepended/appended to the caption vector. To manage the variable lengths of the captions, we truncate each caption at a predefined maximum caption length and use zero-padding on captions that do not meet than this threshold.

The MSCOCO images were scaled and normalized to match the expected input of our image encoding models, which were pretrained on the ImageNet dataset. No other image augmentations were performed.

4. Experiment Results

Overview

The goal of our experiments was to 1) establish a baseline performance for comparison to Aneja et al., and 2) evaluate different hyperparameters based on their impact on accuracy and runtime. Unless otherwise mentioned, all models use default parameters identical to those used in Aneja et al. Specific parameters of note:

- Maximum caption length is set to 15
- Vocabulary size set to 9221 words
- Models are trained for 30 epochs
- Image Encoder starts training at epoch 8
- Learning rate:$5e^{-5}$ at start and decreased by 0.1 at epoch 15
- Convolution kernel size of 5, with 3 layers of convolutions

All tests were conducted on Blue Waters XK nodes. Each node uses a single AMD 6276 Interlagos Processor with 32GB memory and a NVIDIA GK110 (K20X) processer with 6GB memory. Total computation time for all experiments, including training and accuracy calculations, was approximately 900 hours.

Baseline

Test Variant	B1	B2	B3	B4	R	C	Train Loss	Time
Baseline Model	0.709	0.508	0.359	0.254	0.532	0.811	**1.21**	**182.3**
Baseline Model + Attn	**0.712**	**0.514**	**0.368**	**0.263**	**0.537**	**0.829**	1.79	187.5

Table 1: Comparison of the baseline model with and without attention layers. Test metrics used: Bleu-1 (B1), Bleu-2 (B2), Bleu-3 (B3), Bleu-4 (B4), ROUGE-L (R), CIDEr (C). The metrics shown are the highest-achieved scores on the validation set. 'Train Loss' is the lowest average epoch loss achieved on the training set. 'Time' indicates the average training time (excluding test accuracy calculations) in minutes per epoch.

In addition to following the same hyperparameters, we also implemented weight normalization, dropout, and residual connections for each convolution layer. Enabling the attention layers added a consistent, albeit minor, boost in accuracy.

Image Encoders

Test Variant	B1	B2	B3	B4	R	C	Train Loss	Time
VGG19 + Attn	0.712	0.514	0.368	0.263	0.537	0.829	**1.79**	187.5
Resnet18 + Attn	0.706	0.507	0.360	0.258	0.530	0.811	2.29	**67.3**
Densenet161 + Attn	**0.730**	**0.539**	**0.390**	**0.282**	**0.549**	**0.928**	2.15	154.5

Table 2: Comparison of the baseline model using a VGG19 image encoder with two different models using Resnet18 and Densenet161 image encoders

We implemented two alternative image encoder models for comparison with the VGG19 baseline. Resnet18 was selected to recreate a similar accuracy with lower computational expense, and Densenet161 was selected to increase test accuracy. Implementation of both these models was similar to that of VGG19: the convolution output was sent to the Attention modules and the final fully-connected layer was reshaped to output the 512-dimensional image features. In the Densenet161 model, we down sampled the 2208 channel convolution output to 512 channels using a 1x1 convolution kernel to match the Attention model requirements.

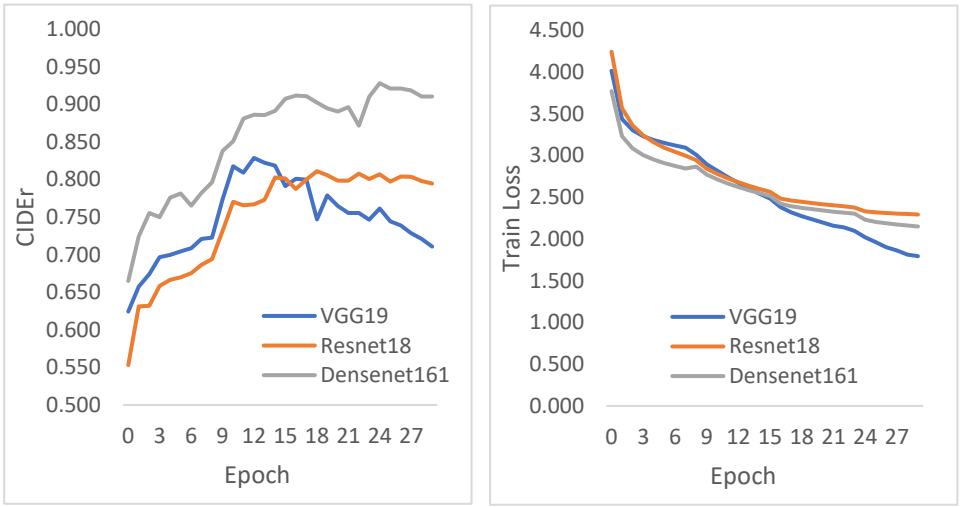

Figure 1: Test accuracy and training loss comparison for different image encoders

We see from the results that Resnet18 training took 64% less time than the VGG19 model at a 3% lower CIDEr score. The Densenet161 model attained a 14% CIDEr score while still using 18% less training time. The results of both models suggest that using more recent image encoders have a positive effect on performance.

Temperature

Test Variant	Model	B1	B2	B3	B4	R	C	Train Loss	Time
T = 0.05	Resnet18 + Attn	0.702	0.503	0.358	0.255	0.529	0.801	2.12	67.8
T = 0.10	Resnet18 + Attn	**0.711**	0.505	0.356	0.253	0.529	0.810	**2.09**	**67.0**
T = 0.25	Resnet18 + Attn	0.706	0.503	0.356	0.253	0.528	0.809	2.11	67.2
T = 0.50	Resnet18 + Attn	0.709	0.505	0.358	0.254	0.527	0.808	2.18	67.4
T = 0.75	Resnet18 + Attn	**0.711**	**0.508**	0.358	0.254	0.529	0.803	2.25	67.2
T = 1.00	Resnet18 + Attn	0.706	0.507	**0.360**	**0.258**	0.530	**0.811**	2.29	67.3
T = 1.25	Resnet18 + Attn	0.710	**0.508**	0.359	0.255	**0.532**	**0.811**	2.33	67.3

Table 3: Comparison of different temperatures when using a temperature-softmax activation function during training.

Neural network outputs logit, $z_i = w_i^T h_{i-1}$, in the fully connected layer. $z_i \in (-\infty, \infty)$ is then converted into class probability by using the Softmax Function:

$$p_i = f(z_i) = \frac{\exp(z_i/T)}{\sum_j \exp(z_i/T)}$$

Note that temperature T is usually set to 1. Using higher temperature value produces softer class probabilities where the inter-class probability differences are low in comparison. Consequently, it leads to softer probability distribution over all classes.

We implemented various temperature values and the performance for each temperature value is displayed above. Note that if temperature $T < 1$, then the model will exhibit overly confident class-probabilities.

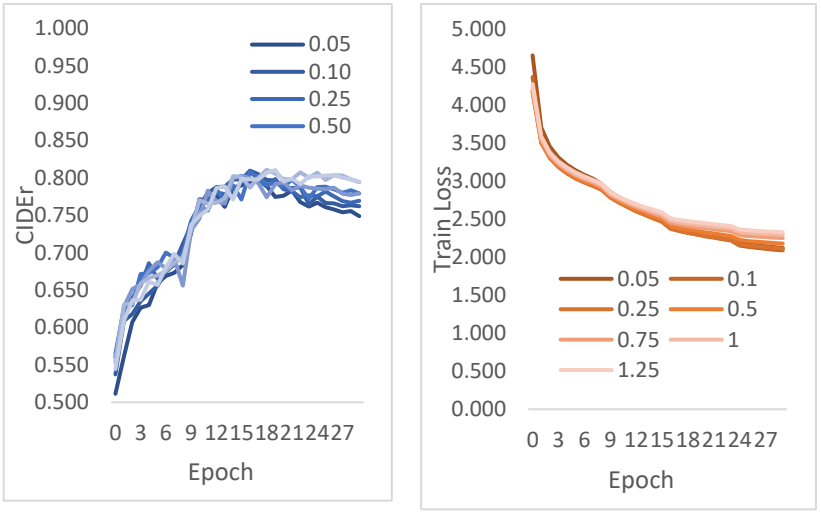

Figure 2: Test accuracy and training loss comparison for different temperatures using the Resnet18 image encoder

Beam Search

Test Variant	Model	B1	B2	B3	B4	R	C
Beam Size 1	Resnet18 + Attn	0.710	0.508	0.359	0.255	0.532	0.811
Beam Size 3	Resnet18 + Attn	**0.715**	**0.519**	0.378	0.279	**0.536**	0.845
Beam Size 5	Resnet18 + Attn	0.713	0.518	0.378	0.281	**0.536**	0.844
Beam Size 7	Resnet18 + Attn	0.712	0.518	**0.379**	**0.283**	**0.536**	**0.847**

Table 4: Comparison of different beam sizes and their effect on test scores for the Beam Search algorithm

Beam size $\in \{1,3,5,7\}$ were used to generate captions for each image. BLEU1, BLEU2, BLEU3, BLEU4, ROUGE-L, and CIDEr were calculated for each parameter. The results are listed in *Table 4*. Note that there is a noticeable jump in all metrics when we use Beam Size greater than 1. Interestingly, the performance increased due to difference in beam size is negligible for any beam size greater than 1. Additionally, we note that the number of unique words at each word position is relatively smaller as we increase Beam Size. Case in point, the *Figure 3* shows the number of unique words between beam size of 1 (blue bar) to beam size of 5 (orange bar)

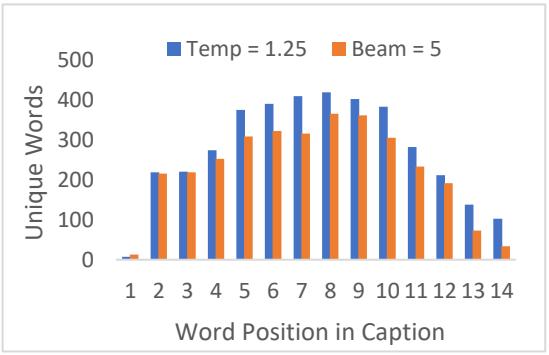

Figure 3: Comparison of word diversity: the number of unique words at each word position.

Glove Features

Test Variant	Model	B1	B2	B3	B4	R	C	Train Loss	Time
No Pretrained Embedding	Resnet18 + Attn	0.706	0.507	0.360	0.258	0.530	0.811	2.29	67.3
GloVe Embedding	Resnet18 + Attn	0.715	**0.515**	**0.365**	**0.260**	0.534	0.823	**2.28**	66.8
GloVe Embedding (Frozen)	Resnet18 + Attn	**0.716**	0.514	**0.365**	0.259	**0.535**	**0.824**	2.31	**66.3**

Table 5: Test results for using a pretrained word embedding

We tested using pretrained GloVe[10] word embeddings in place of the default word embedding trained from scratch on the training data. The GloVe word vectors were trained on 840B tokens extracted from the Common Crawl web archive dataset. In order to maintain comparability, we limited the GloVe vocabulary to match the baseline 9221 word set. This resulted in 639-word tokens not available in the GloVe vocabulary, which we replaced with 300-dimensional vectors of Uniform random variables. During the implementation of the pretrained word embedding, we also experimented with freezing the embedding layer rather than allowing the model to finetune the parameters.

As expected, using the GloVe embeddings improved performance over the baseline Resnet18 model. Freezing the embeddings did not significantly affect test accuracy; our hypothesis is that subsequent fully-connected layer adjusted to "absorb" the lack of parameter backpropagation in the embedding layer.

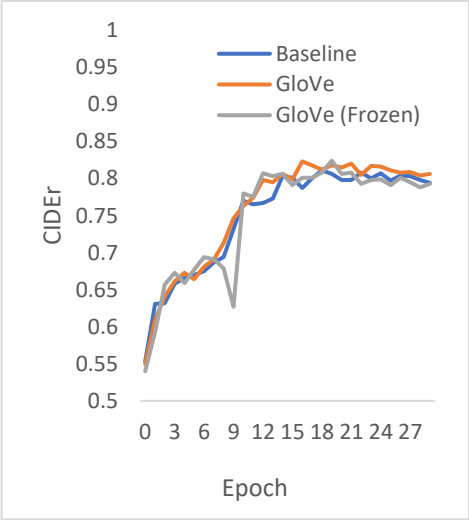

Figure 4: Comparison of word embedding variants

Kernel Size

Test Variant	Model	B1	B2	B3	B4	R	C	Train Loss	Time
K = 3	Resnet18 + Attn	**0.712**	**0.512**	**0.364**	**0.260**	**0.533**	**0.811**	2.38	**66.7**
K = 5	Resnet18 + Attn	0.706	0.507	0.360	0.258	0.530	**0.811**	2.29	66.8
K = 8	Resnet18 + Attn	0.702	0.504	0.359	0.257	0.528	0.810	2.26	70.0
K = 12	Resnet18 + Attn	0.702	0.502	0.357	0.254	0.527	0.800	**2.24**	71.6

Table 6: Comparison of different kernel sizes during the 1-dimensional convolution operation

The kernel size parameter for the 1-dimensional convolution determines the number of historical words to consider for the current word prediction. Test results suggest that a lower kernel size (K=3) results in higher accuracy, however this may, in part, be due to differences in the test metrics. Simpler metrics such as Bleu-1 and Bleu-2 rank the K=3 variant higher, whereas more recent metrics such as CIDEr and ROUGE indicate comparable results. One potential explanation is

that lower kernel sizes tend to produce captions with lower word diversity; specifically, that they repeat certain words. This repetition would not have as measurable an impact on the Bleu-1 and Bleu-2 metrics due to their calculation methods.

5. Code Details

Our implemented code is available on github: https://github.com/davinhill/Convolution_Captioning. Please visit the github repository for more details regarding individual files and implementation.

All files were coded by our team, except for the following functions:

- **Beam Search**: Implemented from https://github.com/aditya12agd5/convcap. We modified the code to work with our Convolution Captioning implementation.
- **PycocoTools and PycocoEvalCap**: Available here: https://github.com/tylin/coco-caption. We modified the code to work with Python 3 and Blue Waters.
- **Download of Split file**: To match the train/validation data splits from https://github.com/aditya12agd5/convcap, we reused the code to extract the splits from the downloaded file.

6. References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems*, 2014, vol. 4, no. January, pp. 3104–3112.

[2] A. Van Den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with PixelCNN decoders," in *Advances in Neural Information Processing Systems*, 2016, pp. 4797–4805.

[3] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *34th International Conference on Machine Learning, ICML 2017*, 2017, vol. 3, pp. 2029–2042.

[4] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June, pp. 3156–3164.

[5] J. Aneja, A. Deshpande, and A. G. Schwing, "Convolutional Image Captioning," 2018.

[6] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009, pp. 248–255.

[7] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *34th International Conference on Machine Learning, ICML 2017*, 2017, vol. 2, pp. 1551–1559.

[8] A. Vaswani *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, 2017, vol. 2017-Decem, pp. 5999–6009.

[9] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8693 LNCS, no. PART 5, pp. 740–755.

[10] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014, pp. 1532–1543.