Everything Looks Like Chicken (and Waffles)

A Neural Image Captioning System, built on Yelp image data. A submission to the Yelp Dataset Challenge.

Tamir Bennatan
timibennatan@gmail.com

***Abstract*— This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.This is filler. Here goes the abstract.**

I. INTRODUCTION

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

Here is an introduction. I will detail what an NIC system is, why it's hard and useful. I talk about how other researchers use specialized datasets for image captioning, and how in this work I used a real life dataset, where the annotaters are not given any instructions. I then talk about the work that I did, briefly outline experiements, results, and mention that I do qualitative error checking.

II. RELATED WORK

In recent years, the deep learning community has have achieved great success in core computer vision tasks, such as image classification, object identification, and image extraction (Sinha et al. 2018). These successes, coupled with advancements in neural machine translation and language modeling technologies, have enabled researchers to develop end-to-end neural network models that can be used for image captioning systems.

Vinyals et al (2014). describe such a system, which utilizes the power of Convolutional Neural Networks (CNN) for extracting complex visual features from images, and Recurrent Neural Networks (RNN) for generating new sequences. Inspired by the encoder-decoder framework for machine translation, the authors use a deep CNN to encode images to fixed-length vector representation, and feed this representation to an RNN which is used to generate a caption (further details in section III).

In their 2017 papers, Tanti et al. describe a set of architectures that fall under the general encoder-decoder framework. The primary difference between these architectures concerns how to feed the image features to the RNN layers, if at all. Tanti et al. rigorously test the efficacy of image captioning models with varying architectures on canonical image captioning datasets, in an attempt to deduce favorable architectures for image captioning models, and to form a conjecture as to what broad classes of tasks RNNs are best suited for.

Xu et al. (2016) achieve state of the art performance on standard datasets by incorporating an attention machanism. The authors argue that attention proves useful for the image captioning problem, as a network can learn to focus on the important features of an image, and ignore noisy features that result from cluttered images. Attention also provides additional interpretability to an image captioning model, as one can visualize what section of an image the attention is focused on at each time step.

Previous work focuses on the use of specialized datasets that were collected for the purpose of image captioning and object segmentation tasks, such as Flickr-8k (Rashtchian et al.; 2010), Flickr-30k (Young et al.; 2014), and MS-COCO (Lin et al.; 2014). These datasets were collected by giving human annotators detailed instructions on how to annotate each image, resulting in relativly clean, albeit synthetic datasets. This work uses a much more difficult, real-world dataset collected by Yelp. As users are given no instructions on how to annotate each image before uploading it to the site, this dataset makes it much more difficult to learn an effective image captioning network. In this work, I discuss some of these difficulties, and undesireable behaviors that

arise when training an image captioning network on such data. I also experiment with several architectures proposed by Tanti et al., in an attempt to study which architectures perform best under such circumstances.

III. THEORETICAL BACKGROUND

In this section, I discuss several abstract perspectives regarding the problem of Neural Image Captioning (NIC) which motivated many of the design choices and experiments that follow. First, I discuss how one can fit the problem of generating a caption of an image into the probablistic framework. Then, I briefly describe encoder-decoder models, and how the models I built are inspired by the use of the encoder-decoder framework in neural machine translation. Finally, I highlight an important design choice made when specifying the architecture of an NIC model, and the conceptual implication of this choice.

A. Training as Maximizing Caption Likelihood

Although much of the previous work on image and video captioning involves stitching together several independent systems, researchers are achieving increasing success building single neural networks for image captioning that are fully trainable by back-propogation. Learning the parameters of such models has a natural probablistic interpretation; the objective is to directly maximize (with respect to the model weights, θ) the joint probability of the predicted captions given input images, I, and model weights. For a dataset of N image/caption pairs, this can be modeled as[1]:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{N} \log P(S^{(i)}|I^{(i)}; \theta \tag{1}$$

Where $(S^{(i)}, I^{(i)})$ is the i^{th} caption/image pair, and each caption $S^{(i)}$ is a sequence of tokens $(S_1^{(i)}, S_1^{(i)}, ...S_k^{(i)})$ for a fixed maximum sequence length, k.

We may simplify this by the chain rule of probability:

$$\theta_{\text{MLE}} = \arg\max_{\theta} \sum_{i=1}^{N} \sum_{t=1}^{k} \log P(S_t^{(i)}|S_{t-1}^{(i)}...S_1^{(i)}; \theta) \tag{2}$$

Equation (2) is the inspiration for the training schema I used to learn the weights, to be discussed further in section **SECTION HERE**.

B. Encoder-Decoder Framework for Image Captioning

Traditional sequence prediction tasks - such as time series forecasting and language modeling - involve using a varying length sequence of inputs (whose ordering usually have a temporal interpretation), and predicting a single output (usually at the next time step). Neural networks are well equipped to solve this type of "many-to-one" problem, as sequences of inputs may be coerced to a fixed number of time steps, so that the network does not need to account for varying length input or outputs.

[1]Under the assumption that the correct captions the images are mutually independent.

Another class of sequence prediction task are those with varying length sequences as both inputs and outputs. These so-called "many-to-many" or "sequence-to-sequence" problems are more difficult than many-to-one problems, as the network must learn to produce predictions of varying lenghts.

An important example of a sequence-to-sequence problem is that of statistical machine translation (SMT). SMT models attempt to take in a sequence of words in one language, and output a sequence of words in another language, all while preserving the meaning of the input sequence and the coherence of the output sequence.

An approach which has proven effective in sequence-to-sequence prediction problems, including machine translation, is called the "Encoder-Decoder" architecture. This architecture is comprised of two parts: the "encoder" takes in a varying length input sequence, and encodes it into a fixed-length vector representation. The "decoder" model decodes this vector representation into a varying-length sequence prediction.

Cho et al. (2014) use this framework to build a SMT system which translates sequences from French to English. They use LSTM recurrent neural networks to encode input sequences, and LSTMs to decode the encoded representations to form the predictions. Cho et al. show that the encoded representations of input sequences preserve syntactic and semantic structure - thus, this representation is often referred to as a "sequence embedding." Variants of encoder-decoder models have become the state of the art in machine translation; in fact, Google has adopted this approach in their Google Translate Service (Wu et al.; 2016).

Modern neural image captioning (NIC) systems, including those described in this paper, adopt this approach for the problem of image captioning. Image captioning is similar to machine translation in that the output is a varying length sequence of natural language, but differs in that the input is an image, rather than a sequence. Thus, instead of using RNN models such as LSTMs to encode the input, NIC systems typically use CNN models to form a vector representation from an image. Recurrent neural networks then use this representation to form an output sequence. Thus, NIC systems fall under the Encoder-Decoder framework, with CNNs acting as encoders and RNNs acting as decoders [figure 1].

The use of CNNs as encoders is justified by the success of CNNs in core computer vision tasks, such as image classification and segmentation. It is believed that CNNs succeed in these tasks becasue they can extract meaningful and complex features images effectively. Thus, just as LSTMs an encode the semantic and syntactic of an input sequence in SMT applications, it is believed that CNNs encode meaningful properties of an input image in NIC systems.

C. Merge and Inject Models

In a NIC system, CNN image features are combined with previously predicted tokens in a sequence to predict the next token in a sequence. The Encoder-Decoder framework includes a broad class of model architectures, and there are many ways to condition the sequence prediction on the image

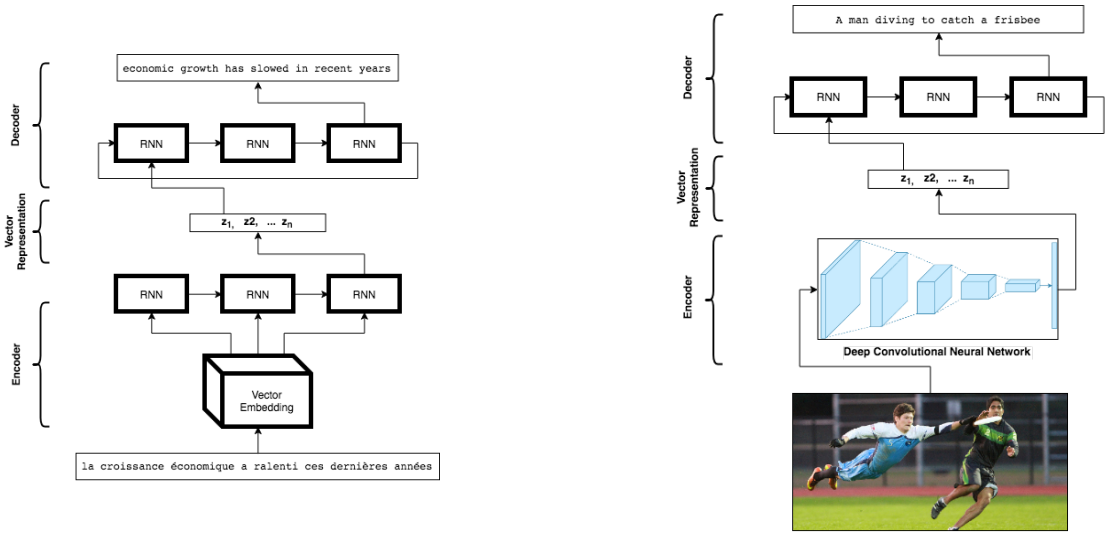


Fig. 1: The encoder-decoder framework in neural machine translation models (left) and image captioning models (right). In machine translation applications, an input sequence is encoded by an RNN to a vector represention, before decoded to an output sequence. Analogously, a neural image captioning system uses a CNN to encode an input image before generating an output sequence.

features, each of which uses the assigns a different role to the RNN component of a NIC system.

In their paper *What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?*, Tanti et al. describe two important classes of NIC models that differ in their method of incorporating image features.

The first, referred to as "Condition-by-Inject" (or "Inject" models for short), incorporate the CNN image features directly to the RNN component. Using both the image features and the previously predicted tokens to predict the next token of the sequence. This is ususally done by conditioning the internal state of the RNN cells with the image representation, or treating the image representation as the first 'word' in the input sequence. The second method, called "Condition-by-Merge" (or "Merge" models for short), never incorporates the image features into the RNN model. Instead, the RNN is used to encode the linguistic features from previously predicted tokens, independent of the perceptual encoding formed by the CNN. Then, the linguistic and pereceptual encodings are merged to a single representation, from which the output prediction is made [figure 2].

The Inject and Merge architectures assign different conceptual roles to the RNN in a NIC system. In the Inject model, both perceptual and linguistic features are available to the RNN, which is responsible for generating the next predicted token. Thus, the RNN acts as a "generator" in the Inject architecture. In contrast, the RNN has no access to the image freatures in the Merge architecture. Instead, the RNN encodes an input sequence into a fixed length vector represenation, and a later fully connected layer is responsible for generation of new tokens. Thus, in the Inject architectures, the RNN can be thought of as an encoder, rather than a generator.

In the experiments described in section **ENTER SECTION**, I compare models using the Inject and Merge architectures. The idea that the choice of an Inject/Merge architecture dictates the 'role' of the RNN in a NIC adds adds a new persepctive to these experiments. The relative efficacy of one architecture over another not only shows us which architecture is more appropriate for this taks, but provides insights into what settings RNNs best suited in general.

IV. The Yelp Dataset

Yelp maintains a public dataset for the purposes of research and education. Part of this dataset is a collection of over 200,000 images uploaded by users, 100,807 with captions written by users upon uploading these images. This subset of images is what I used to train my image captioning system.

Compared to standard benchmark datasets for image captioning, such as Flickr-8k and Flickr-30k, the Yelp dataset makes for a much more difficult task in training an effective NIC model.

Standard datasets typically have 4-8 training captions per image, which helps elucidate which words are related the an image's content, and which words are a result of more noisy phenomena, such as an author's writing style.

More importantly, datasets such as Flickr-8k and Flickr-30k were collected by giving human subjects detailed instructions on how to form a training caption [figure 3]. In contrast Yelp users are given no guidance on what photos to upload to the site, nor how to write corresponding captions. The result is a dataset wit a much larger variablility in caption style, content and relevence to the task of image captioning. More severely, since the users are not given a common set of instructions on how to construct training captions, the task of modeling the image-caption relationship is not well defined.

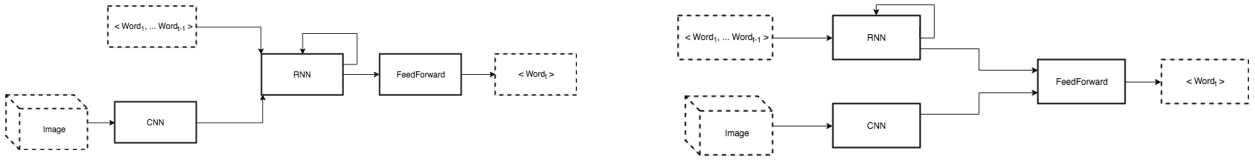


Fig. 2: A bare-bones diagram of the Inject architecture (left) and Merge architecture (right). The former conditions the RNN with both the image and linguistic features, and uses the RNN to generate new tokens. The uses the RNN to encode the previously predicted tokens into a vector representation, rather than to generate new tokens.

Fig. 3: Instructions presented to Amazon Turkers in the collection of the Flickr-8k dataset (Rashtchian et al., 2010). Image extracted directly from [7].

Figure 4 shows four images that make it difficult to train a NIC on the Yelp image dataset.

It's clear that for a NIC to perform well on the Yelp dataset, one would have to invest considerable effort in structuring and cleaning the training data. As I only did minimal preprocessing (see section **INSERT PREPROCESSING SECTION**, I do not expect my system to generalize well. Instead, my goal is to study the practical challenges of building an image captioning system on 'real-life' data. Perhaps the most pertinent challenge is collecting structured data which is designed for the task.

Fig. 4: Sample images from the Yelp image dataset, chosen to exemplify the varying sources of noise in the dataset.

- *Top-left:* Rarely encountered noun-phrases such as 'Artery clogger' and apocryphal words like 'Head-which' make it difficult to identify words tied to an image's content.
- *Top-right:* Captions which are unrelated to the image content.
- *bottom-left:* Wordy captions, which are likely to have content that is particular to an image, and does not generalize to images with similar content.
- *bottom-right:* Over simplified captions. This training example may cause a neural network to learn a link between the features of strawberry shortcake to the word *dessert*, which describes a much larger class of dishes.

V. METHODOLOGY

In this section, I describe the preprocessing and feature extraction steps I took to prepare the Yelp dataset for training. I then outline the different model architectures I tested, and a custom scheme for generating caption from unseen images using these models.

A. Extracting Image Embeddings

A NIC system must incorporate features extracted from both an input image and a partial caption [2] in order to predict the next word of the caption. This requires that one compress an input image into a fixed-length vector which encodes the visual features of the image - thus forming a sort of *image embedding*.

One approach to doing so is to construct a model architecture in which input images pass through many layers - responsible for extracting informative features from the images - before incorporating these image features with the rest of the NIC model - and then training the entire model using back-propogation.

The problem with this approach is that powerful neural networks that prove effective in various computer vision tasks often have many millions of parameters. Thus, a NIC that is trained to extract image features from scratch would likely be very deep and have many parameters[3], which leads to slow training, and a propensity to overfit.

[2]Initially, the 'partial caption' is a special token that signifies the beginning of the predicted sequence, `<startseq>`. Then, one predicts the next word of the sequence and feeds it back to the network, until the token `<endseq>` is predicted by the network, which marks the end of the predicted sequence.

[3]This argument uses the intuition that networks that achieve state of the art performance in classic tasks like image classification do a good job of extracting interesting features from images. Therefore, a network trained to form powerful image embeddings from scratch will likely also need to have many parameters.

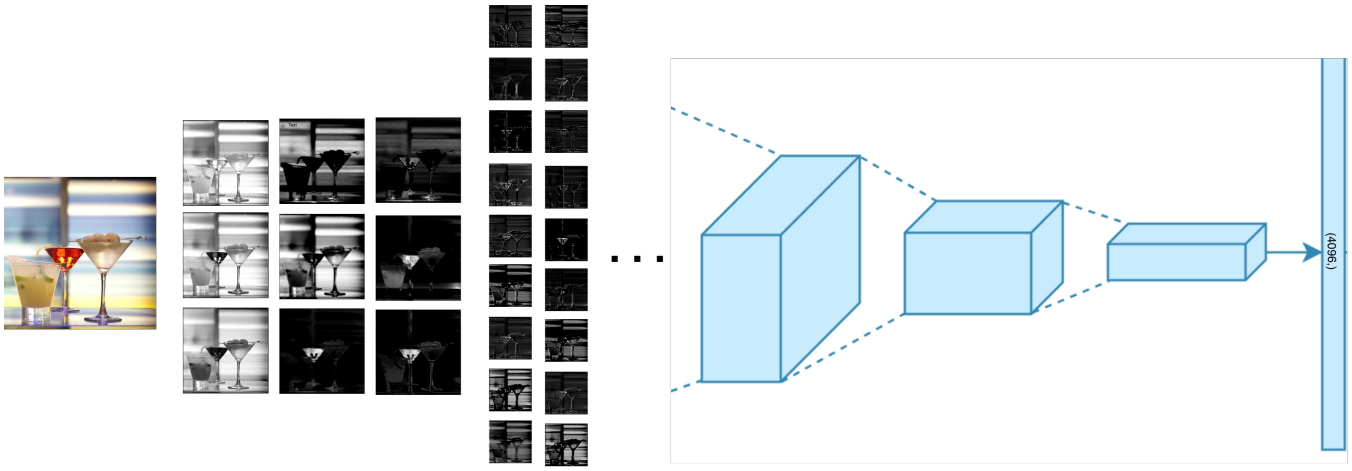


Fig. 5: Vector representations of images, or "image embeddings", are created by saving an intermediate activation of the VGG16 network, which is capable of detecting rich perceptual features, such as edges, blur and hue. These embeddings, combined with linguistic features, are used to generate captions for input images. Thus, the abridged VGG16 network may be thought of as the *encoder* in the *Encoder-Decoder* framework.

Instead, to create image embeddings for each of the training images, I utilized the pre-trained weights of the VGG16 network (Simonyan & Zisserman; 2014), available in the `keras` deep learning library (Chollet et al.; 2015). This network, with 138,357,544 parameters and trained on over 1.3 million images from the ImageNet dataset (Russakovsky et al.; 2015), secured the first and the second places in the ImageNet ILSVRC-2014 localisation and classification competitions, respectively. Thus, it is reasonable to assume that this network is capable of extracting complex feautres from diverse images.

The last 3 layers of the VGG16 architecuture are fully connected layers, with dimensions 4096, 4096 and 1000, respectively. To generate an image embedding for each image in the Yelp dataset, I first removed the last two fully connected layers from VGG16, and used the predictions of this abridged model as the image embeddings [figure 5]. I justification for dropping the last two layers of VGG16 is that while shallow layers of the network are responsible for extracting revealing features, the final layers are specialized for the task they were trained for - namely ImageNet classification. Thus, omitting the final two layers allows me to encode each image as a $4,096$ dimensional vector which is both informative and generalizes to the image captioning task.

B. Caption Text Preprocessing

A widespread challenge in Natural Language Processsing (NLP) tasks is that corpus vocabularies are very sparse - meaning that few words occur many times, any many words occur few times. Empirical studies repeatedly show that the frequency of a word in a corpus is inversely proportional to its frequency rank - a result commonly referred to as *Zipf's Law* (Wyllys, R; 1981). Sparse vocabularies lead to models with many parameters, and makes it harder to generalize patterns learned during training.

Thus, I performed a cursory clense of the captions provided in the Yelp dataset, with goal of reducing vocabulary size. This clense is by no means thorough, and is likely the bottleneck in my final captioning system.

The steps I took - in order - are:

- Lowercase all characters.
- Remove newline characters.
- Replace the character `&` with the word `and`. This is so the system will identify phrases like `sweet & spicy` and `sweet and spicy` to be identical.
- Convert strings which match a simple regex pattern of a website domain[4] to the token `website`. This is because two images are in the Yelp dataset are unlikely to refer to the same website domain, leading to tokens of low frequency and a larger vocabulary.
- For the same reasons, I replaced all tokens with numeric characthers with the token `num`.
- Using the python `unidecode` library, I re-encoded all characters to ascii characters. This way, phrases with/without accents like *gruyère soufflé* and *gruyere souffle* are consolidated to the same tokens. The `unidecode` library also implements functions for transliterating tokens from forign languages to english characters, further reducing vocabulary size.
- Remove all remaining punctuation.
- Prepent all captions with the token `<startseq>`, and postpend with the token `<endseq>`, to mark the beginning and end of each caption.

After these steps, the processed captions contain 30,012 unique terms.

C. Formatting Data for Training

Section 3.A discusses how one can conceptually fit the image captioning task into the probabilistic framework. Equa-

[4]The regex pattern used to detect webite domains is `[a-z\:\/\.0-9]+\.(org|com|net)`, using the python `re` library.

tion (2) shows that under mild assumptions, the probability of a caption being correct, given an image, can be decomposed into the probability of each token given the previous tokens and the image:

$$\log P(S_1, ...S_k|I;\theta) = \sum_{t=1}^{k} \log P(S_t|S_1...S_{t-1}, I;\theta)$$

Typically, NIC systems model $P(S_t|S_1...S_{t-1}, I;\theta)$ with a recurrent neural network (RNN) sub-architecture. That is, the RNN is responsible for predicting the next token, given the previous tokens in the caption and the image; this way, the RNN plays the role of a language model.

Inspired by (2), I reformat the training captions to be compatible with a language modeling task. First I extract an image embedding for each image in the dataset (section 5.A), and preprocess the text of the corresponding caption (section 5.B). Then, for each embedding/caption pair, I create a new training example for each token in the caption, where each token plays the role of the response, and the covariates[5] are the tokens that occur previously in the original caption, as well as a copy of the corresponding image embedding [figure 6].

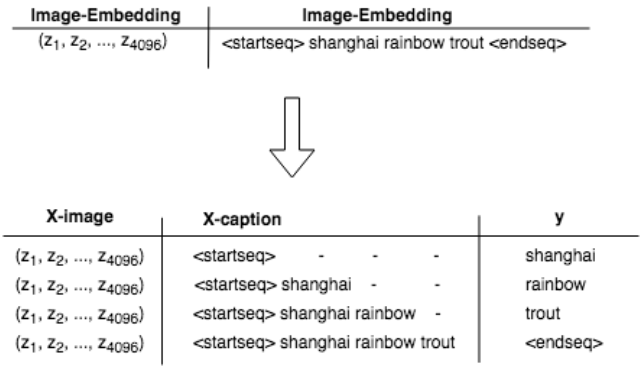


Fig. 6: An example of how an image-embedding/caption pair is reformatted to suit a language modeling task. Each token (except for the first `<startseq>` token is treated as a response, where the covariates are the previous tokens in the sequence, and a copy of the image embedding.

Captions who exceed 15 tokens in length are truncated to 15 tokens. Captions that are shorter than 15 tokens are padded by a special `null` token. This way, all captions have a uniform length of 15 tokens.

After performing this expansion, the original 100,807 images with captions become 675,289 image-embedding/sub-caption pairs. When splitting the dataset into training/validation splits (section 6), I split the data *before* performing the expansion detailed above. This is to ensure that the models are trained on complete captions.

D. Model Architectures

In this project, I experiment with three model architectures - two *merge models* and one *inject model*, in the language defined in section 3.C. See figure 7 for architecture details.

For all three models, the inputs are the same. Each training example is comprised of two input vectors - one for the image embeddings of dimension $(4096,)$, and an integer encoding of the input caption of dimension $(15,)$. The caption vectors are then passed to an embedding layer, where each token is represented as a 300 dimensional word embedding. I used pre-trained Word2Vec embeddings (Mikolov et al.; 2013), trained on the Google News corpus[6]. I configured these embedding weights to be fixed (non-trainable) in all three models.

For the inject model, image embeddings are fed to a dense[7] layer of output dimension 300. The activation of this layer is then prependended to the word embeddings from the caption input, to form a volume of dimension $(16, 300)$, which is then fed to an LSTM layer, with a hidden state of 300 dimensions. In other words, the image features are treated as the first "word" in the sequence, before being processed by the RNN language model. The output of this LSTM is fed to a dense layer with a softmax activation to predict the next word in the sequence. I refer to this architecture as simply the "inject model."

In both merge models, the captions - after being transformed to word embeddings - are passed to an LSTM cell with a hidden state of 256 dimensions. Image embeddings are passed to a dense layer of output dimension 256.

The activation of this dense layer is combined with the output of the final LSTM cell, thought he way they are combined differs between the two merge models. In the first, the two activations are *concatenated*, yielding a single 512 dimensional vector, which is fed to a dense layer to form predictions. This model is referred to as the *"merge-concat"* model. In the second, the two activations are *added*, yielding a vector of dimension 256, which is passed to a dense layer to form predictions. This model is referred to as the *"merge-add"* model.

The justification for experimenting with both the merge-concat and merge-add model is that the final dense layer of the merge-concat model has many parameters, as a 512 dimensional layer is fully connected to the final 30012 dimensional layer. The merge-add model halves the number of parameters in the final weight matrix, as a 256 dimensional layer is fully connected to the final 30012 dimensional layer.

E. Inference as Beam Search

The models detailed in section 5.D each take in two vectors as input - an image generated by passing an image through the VGG16 network, and the integer representation of a (partial) caption. The output of the models is a 30012

[5] *Covariates*, as in the features which are trained upon. Also referred to as *independend variables* or *predictors*, and typically denoted as $\mathbb{X}$ in the statistics literature.

[6] Embeddings I used can be downloaded here.

[7] Dense layers are fully connected layers.

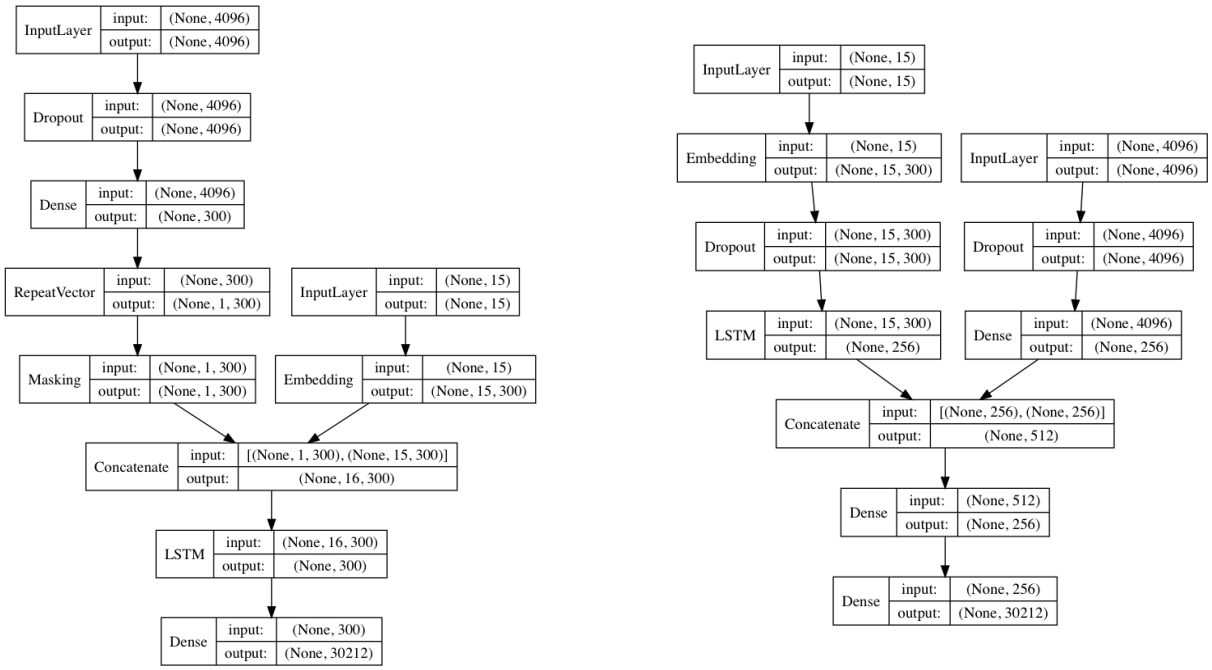


Fig. 7: Inject model (left) and merge-concat models (right). In the inject model, image features are treated as the first "word" in the caption, then passed to the LSTM cell. In the merge-concat model, image and caption features are processed seperately, concatenated, then passed to a dense layer to form predictions.
The merge-add layer (not shown above) differs from the merge-concat model only in that image and caption features are *added* instead of *concatenated*.
Layer names above refer to `keras` layer classes.

dimensional vector - where each element represents the predicted probability of a certain word appearing next in the caption.

Given this setup, it is not trivial how to generate a new caption for an image previously unseen by the network, as partial captions are not availble during the inference stage.

A naive method for generating new captions is **greedy selection**. Starting with an image features and a caption with only the start token `<startseq>`, one repeatedly uses a trained model to predict the probability of each word being next in the caption. Then, the word with the highest predicted probability is added to the caption. This process is repeated until the maximum sequence length is reached[8], or the special token `<endseq` is selected, marking the end of a sequence [algorithm 1].

In each iteration, greedy selection selects the word it finds most probable and adds it to the generated caption. It does not allow for adding words that may be "suboptimal" at a given iteration, but then enables the model to make a prediction it is very confident of in a later iteration. Since our goal is to generate a sequence of tokens $\mathbf{S}$ which well approximates the *joint* probability of each of the tokens, $\mathbf{S}^* = \arg\max_{\mathbf{S}} P(S_1, S_2, ..., S_k|I)$, it seems that this scheme is too rigid.

A more flexible method of inference is **Beam Search**. Instead building up a single caption as greedy selection does, one maintains β candidate solutions at all times, where β is called the *beam width*. The set of candidate solutions is often refered to as the *population* of candidates. At each iteration, each candidate solution in the population is expanded into κ candidates by adding the words with the top κ predicted probabilities to the candidate, where κ is called the *neighborhood size*. After each candidate in the population is expanded into κ new candidates, only the top β candidates maintained (according to some metric of candidate quality), and move on the next iteration.

The metric of candidate solution quality is yet to be defined. Previous work uses standard metrics designed for machine translation and image captioning tasks, such as BLEU (Xu et al.; 2015) and the MSCOCO evaluation code[9] (Tanti et al.; 2017).

In this work, I define a custom quality metric inspired by personal model of what a "good" predicted caption looks like. A good prediction should should be descriptive, and be relevant to an image's content. It should also be succinct, and avoid adding predicted tokens if the model is not "confident" in these predictions.

The quality metric is calculated as follows: for each candidate caption $S \equiv (S_1, S_2, ...S_k)$, keep track of the probabilities of each token $\Omega_S = (\omega_1, \omega_2, ...\omega_k)$[10], predicted by the model during earlier iterations of the beam search.

[8] In my models, I used a maximum sequence length of 15 tokens.

[9] https://github.com/tylin/coco-caption

[10] In this notation, ω_t is the predicted probability of the token S_t.

Algorithm 1 Inference: Greedy Selection

procedure GREEDYSELECT(IMG-FEATURES, MODEL)
    caption $\leftarrow$ [$< startseq >$] ▷ Initialize caption as start token.
    while Length(caption) < 15 **do** ▷ Repeat until generated caption is of maximum length.
        predictions $\leftarrow$ `model.predict(img-features, caption)` ▷ Vector of predicted probabilities
        next-word $\leftarrow$ argmax(*predictions*) ▷ Predicted next word is the one with the highest predicted probability.
        if *next-word* == `<endseq>` **then**
            break ▷ If end token is predicted, return caption as-is.
        else
            cation $\leftarrow$ *cation*.append(*next-word*)
    return *caption*

Algorithm 2 Inference: Beam Search

procedure BEAMSEARCH(IMG-FEATURES, MODEL, β,κ, α)
    population $\leftarrow$ [$< startseq >$] ▷ Initialize population as a single 'starter' caption.
    $i \leftarrow 0$ ▷ Iteration Number
    for $i < 15$ **do**
        for each candidate caption $S \in$ *population* **do**
            if Last token in candidate == `<endseq>` **then**
                break
            predictions $\leftarrow$ `model.predict(img-features, caption)` ▷ Vector of predicted probabilities
            Add top κ predicted words to caption to create κ new candidates
        Truncate poopulate top β candidates, according to quality metric.
        $i \leftarrow i + 1$
    return *population*

Then, the quality score of the sequence S is a weighted sum of the predicted probabilities Ω_S, where the weight multiplied by ω_t is α^t, for some $\alpha \leq 1$:

$$\text{Score}\big((\omega_1, \omega_2, ...\omega_k)\big) = \sum_{t=1}^{k} \omega_t \cdot \alpha^t \tag{3}$$

This measure rewards candidates whose tokens are predicted with high probability, but discounts later tokens due to the geometric decrease in weights, α^t. Thus, α is a sort of regularization hyperparameter - small values of α leads to terse captions, while large values of α leads to captions at or near the maximum token length [figure 8].

Note that greedy selection is a special case of beam search, where $\beta = \kappa = \alpha = 1$.

REFERENCES

[1] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CoRR*, vol. abs/1411.4555, 2014. [Online]. Available: http://arxiv.org/abs/1411.4555

[2] M. Tanti, A. Gatt, and K. P. Camilleri, "Where to put the image in an image caption generator," *CoRR*, vol. abs/1703.09137, 2017. [Online]. Available: http://arxiv.org/abs/1703.09137

[3] ——, "What is the role of recurrent neural networks (rnns) in an image caption generator?" *CoRR*, vol. abs/1708.02043, 2017. [Online]. Available: http://arxiv.org/abs/1708.02043

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: http://arxiv.org/abs/1406.1078

[5] R. K. Sinha, R. Pandey, and R. Pattnaik, "Deep learning for computer vision tasks: A review," *CoRR*, vol. abs/1804.03928, 2018. [Online]. Available: http://arxiv.org/abs/1804.03928

[6] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *CoRR*, vol. abs/1502.03044, 2015. [Online]. Available: http://arxiv.org/abs/1502.03044

[7] C. Rashtchian, P. Young, M. Hodosh, and J. Hockenmaier, "Collecting image annotations using amazon's mechanical turk," in *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, ser. CSLDAMT '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 139–147. [Online]. Available: http://dl.acm.org/citation.cfm?id=1866696.1866717

[8] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 67–78, 2014. [Online]. Available: https://transacl.org/ojs/index.php/tacl/article/view/229

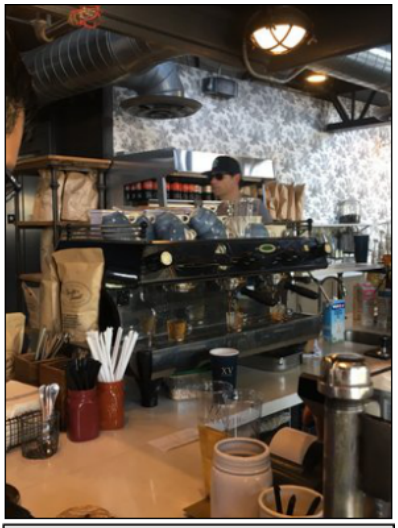
[9] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. [Online]. Available: http://arxiv.org/abs/1405.0312

[10] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," *CoRR*, vol. abs/1609.08144, 2016. [Online]. Available: http://arxiv.org/abs/1609.08144

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[12] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[13] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.



α	Top Predicted Caption
0.5	bar area
0.8	the bar area
0.9	the bar area at a restaurant and a restaurant at las vegas

Fig. 8: Top caption predictions using beam search and the inject model, for different values of α. Small values of α lead to overly terse captions, while large values of α lead to wordy and repetitive captions.

[14] R. E. Wylly, "empirical and theoretical bases of zipf's law," *empirical and theoretical bases of Zipf's law*, vol. 30, p. 53–64, 1981. [Online]. Available: https://www.ideals.illinois.edu/bitstream/handle/2142/7182/librarytrendsv30i1g_opt.pdf?sequence=1

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf