Everything Looks Like Chicken (and Waffles)†

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

Tamir Bennatan
timibennatan@gmail.com

Code and link to live application available on GitHub:
https://github.com/tamirbennatan/Yelp-Image-Captioning

***Abstract*— Neural image captioning (NIC) is the task of developing a neural network which can describe the contents of a previously unseen image. In this technical report, I describe the inner workings of a NIC system I built, trained on photos from Yelp's website. I provide the theoretical background for this work, details of my methodology and experiments, and the lessons I've learned during the process of building this system and by analyzing the model's predictions. This report enables a reader to not only understand reproduce my work, but also illuminates some of the challenges one should be aware of when building a NIC using a real-world dataset, such as the one provided by Yelp.**

I. INTRODUCTION

The goal of image captioning systems is to automatically generate relevant descriptions of images. This task is substantially more difficult than more well-studied computer vision tasks - such as image classification and object detection - as an adequate image captioning system needs to not only identify the objects in an image and how they relate to one another, but also incorporate this perceptual information into a language model which can generate descriptions in coherent natural language.

Image captioning systems have many industrial applications, for example in helping visually impaired users understand a website's content, automatic subtitle generation, or identifying the semantic roles of objects within an image. Image captioning is also of theoretic value, as solving the task would represent a step towards complete scene understanding - where a machine can "see" and interpret perceptual information as a human does - which is a long-standing dream of computer vision and artificial intelligence practitioners.

Recent success of neural network approaches to statistical machine translation has inspired researchers to develop end-to-end neural models for image captioning that are fully trainable by backpropagation. Aided by large image classification datasets (Russakovsky et al.; 2015) and powerful networks trained to solve various computer vision tasks, it is possible to encode the contents of an image into a compact vector representation. This representation can then be combined with linguistic features, and used to generate (in other words, be *decoded* to) novel captions. Thus, image captioning systems broadly fall under the Encoder-Decoder neural network framework.

† To understand the title of this paper, read section 7.B.

Previous academic work of such systems typically utilize datasets that are specially crafted for the image classification task, such as Flickr-8k (Rashtchian et al.; 2010) and Flickr-30k (Young et al.; 2014). In contrast, my system is trained on a real-world image dataset collected by Yelp as part of the 11^{th} round of the Yelp Dataset Challenge . I discuss the challenges of working with datasets that are not specialized for the image captioning tasks, and analyze the root causes for some of the unexpected behaviors that stem from challenges.

My system often produces reasonable captions, and clearly take into account both the input images' content and the coherence of the output caption. Perhaps more importantly, by describing my methodology, detailing the experiments I ran, and performing both quantitative and qualitative analysis of my results, I reveal some of the important considerations one should be aware of when building an image captioning system on a real-life dataset, including:

- Different approaches to incorporating linguistic and perceptual features, and how to structure a dataset so that it can be used to train a neural image captioning system.
- How to formalize the tradeoff between caption relevance and conciseness during the generation process.
- Difficulties that arise from working with a dataset which was collected organically - i.e without prompting subjects to generate examples with detailed instructions.
- Common metrics for automatically evaluating an image captioning system, and their motivations.
- Unexpected behaviors that result form working with a sparse corpus with a large vocabulary - a challege that is common amongst many natural language datasets.

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 segmentation (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 language modeling

tasks. 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 cells, 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 tasks RNNs are best suited for.

Xu et al. (2016) achieve state of the art performance on standard datasets by incorporating an attention mechanism. The authors argue that attention is useful because it enables a network to focus on the important features of an image, and ignore noisy features that are common in 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 relatively clean datasets. This work uses a much more difficult, real-world dataset collected by Yelp. As Yelp 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 undesirable 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 concepts regarding the problem of Neural Image Captioning (NIC) which motivate many of the design choices and experiments that follow in later sections of this report. First, I discuss how one can fit the image captioning problem into the probabilistic 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 backpropagation. Learning the parameters of such models has a natural probabilistic interpretation; the objective is to directly maximize (with respect to the model parameters, θ) the joint probability of the predicted captions given input images, I. 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} (image,caption) 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 expand (1) 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 sections 5.C and 6.A.

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 inputs or outputs.

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 lengths.

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 for machine translation is called the "Encoder-Decoder" architecture. This architecture is comprised of two parts: an "encoder," which takes in a varying length input sequence, and encodes it into a fixed-length vector representation. A "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.

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

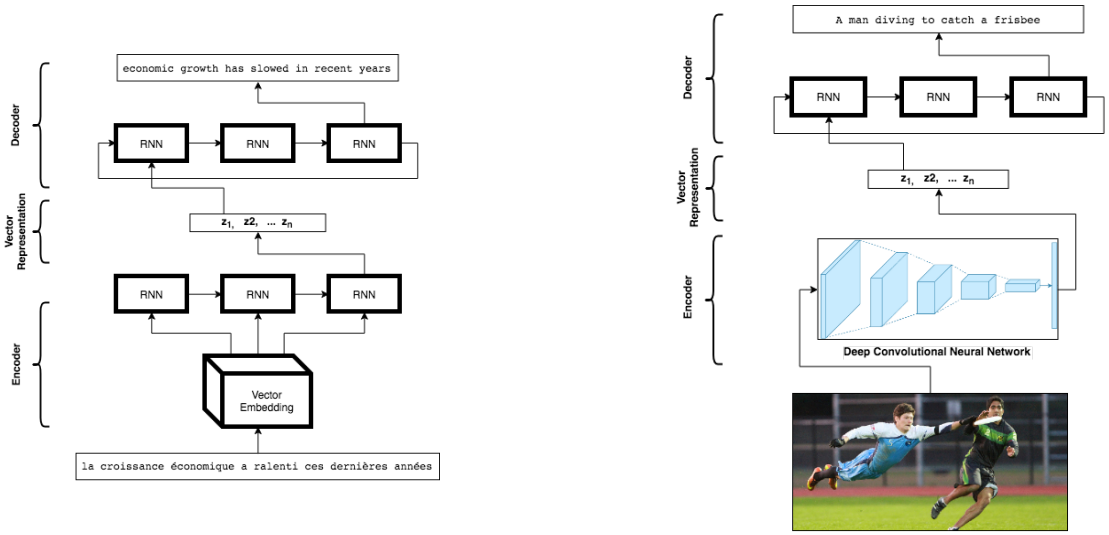


Fig. 1: The encoder-decoder framework in neural machine translation models (left) and image captioning models (right). In machine translation applications, an input sequence is encoded by an RNN to a vector represention, before decoded to an output sequence. Analogously, a neural image captioning system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

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 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. CNNs succeed in these tasks because they can extract meaningful and complex features from 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 to predict the next token in a sequence. There are many ways to condition this sequence prediction on the image 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 usually 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 perceptual 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 features in the Merge architecture. Instead, the RNN encodes an input sequence into a fixed length vector representation, and a later fully connected layer is responsible for generation of new tokens. Thus, in the Merge architectures, the RNN can be thought of as an encoder, rather than a generator.

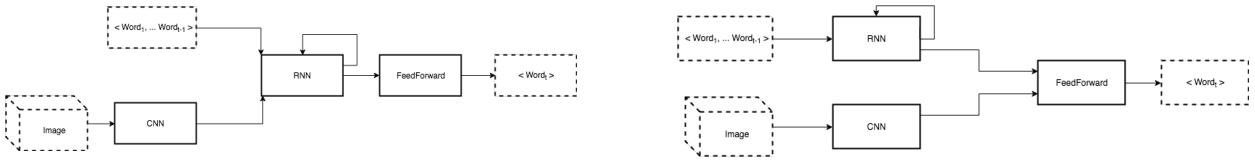


Fig. 2: A bare-bones diagram of the Inject architecture (left) and Merge architecture (right). The former conditions the RNN with both the image and linguistic features, and uses the RNN to generate new tokens. The uses the RNN to encode the previously predicted tokens into a vector representation, rather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

In the experiments described in section 6, I compare models using the Inject and Merge architectures. The idea that the choice of architecture dictates the 'role' of the RNN in a NIC adds adds a new perspective to these experiments. The relative efficacy of one architecture over another not only shows us which architecture is more appropriate for this task, but provides insights into what settings RNNs are best suited for 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 of which contain captions written by users upon uploading these images. This subset of images is what I used to train my image captioning system. Images are annotated with one of the categories *"food", "inside", "outside", "menu"* or *"drink"*.

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 disambiguate 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 captions for these photos. The result is a dataset with a much larger variability in caption style, content and relevance. 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. Figure 4 shows four images that exemplify why it is difficult to train a NIC on the Yelp image dataset.

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.

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

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.

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 backpropagation.

The problem with this approach is that powerful neural networks that effectively extract complex features from images often have many millions of parameters. Thus, a NIC that is trained to extract image features would be slow to train, and have a propensity to overfit.

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 localization and classification competitions, respectively. Thus, it is reasonable to assume that this network is capable of extracting complex features from diverse images.

The last 3 layers of the VGG16 architecture 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 dropped the last two layers because while the first layers of the network are responsible for feature extraction, 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 Processing (NLP) tasks is that corpus vocabularies are often sparse - meaning that few words occur many times, any many words occur few times[3]. Sparse vocabularies lead to models with many parameters, and makes it harder to generalize patterns learned during training.

Thus, I performed a cursory cleanse of the captions provided in the Yelp dataset, with goal of reducing vocabulary size. This cleanse is by no means thorough, and is likely a 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 characters 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 foreign languages to English characters, further reducing vocabulary size.
- Remove all remaining punctuation.
- Prepend 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. Equation (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

[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] 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)

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

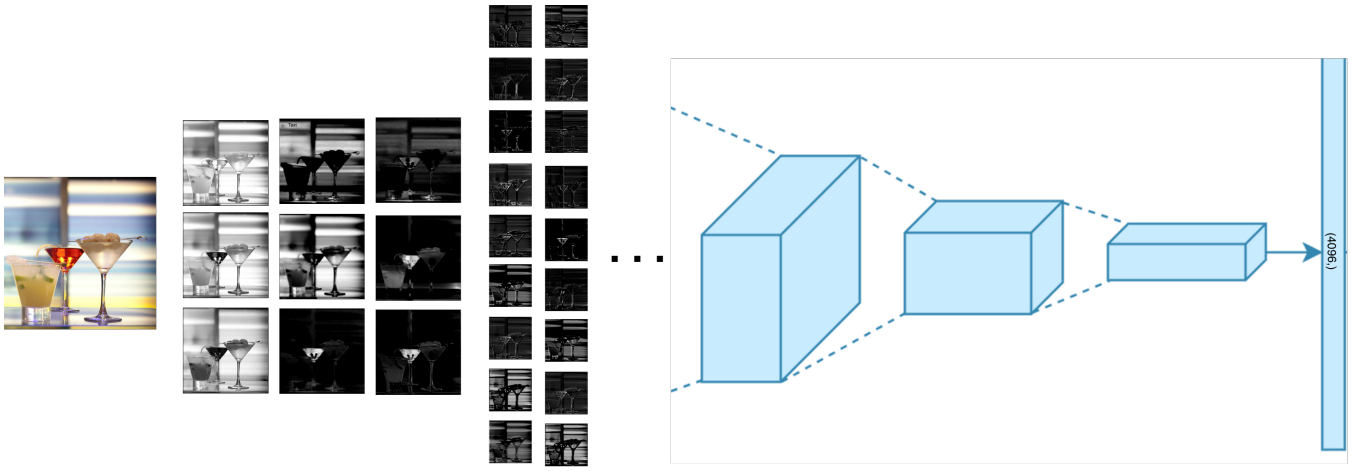


Fig. 5: Vector representations of images, or "image embeddings", are created by saving an intermediate activation of the VGG16 network, which is capable of detecting rich perceptual features, such as edges, blur and hue. These embeddings, combined with linguistic features, are used to generate captions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

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].

Image-Embedding	Image-Embedding
$(z_1, z_2, ..., z_{4096})$	<startseq> shanghai rainbow trout <endseq>

X-image	X-caption	y
$(z_1, z_2, ..., z_{4096})$	<startseq> - - -	shanghai
$(z_1, z_2, ..., z_{4096})$	<startseq> shanghai - -	rainbow
$(z_1, z_2, ..., z_{4096})$	<startseq> shanghai rainbow -	trout
$(z_1, z_2, ..., z_{4096})$	<startseq> shanghai rainbow trout	<endseq>

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.

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

D. Model Architectures

In this project, I experiment with three model architectures - two *merge models* and one *inject model*, using the terms 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 prepended to the word embeddings from the caption input, to form a matrix 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, though the way they are combined differs between the two merge models. In the first, the two activations are *concatenated*, yielding a single 512

[6]Embeddings I used can be downloaded here.

[7]Dense layers are fully connected layers.

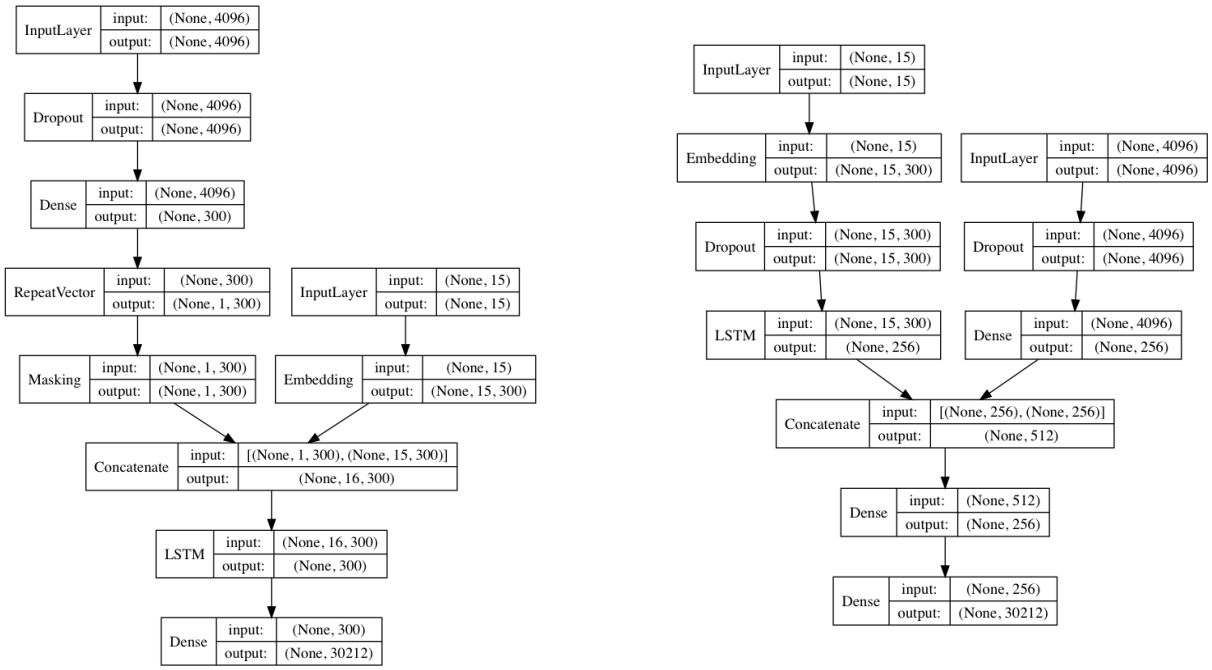


Fig. 7: Inject model (left) and merge-concat models (right). In the inject model, image features are treated as the first "word" in the caption, then passed to the LSTM cell. In the merge-concat model, image and caption features are processed separately, concatenated, then passed to a dense layer to 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, 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 dimensional vector - where each element represents the predicted probability of a certain word appearing next in the caption.

Given this setup, it is not obvious how to generate a new caption for an image previously unseen by the network, as partial captions are not available 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 enable 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, they are sorted according to some quality metric, and the top β candidates are move

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

Algorithm 1 Inference: Greedy Selection

procedure GREEDYSELECT(IMG-FEATURES, MODEL)
  caption ← [$< startseq >$] ▷ Initialize caption as start token.
  while Length(caption) < 15 **do** ▷ Repeat until generated caption is of maximum length.
    predictions ← `model.predict(img-features, caption)` ▷ Vector of predicted probabilities
    next-word ← 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 ← *cation*.append(*next-word*)
  return *caption*

Algorithm 2 Inference: Beam Search

procedure BEAMSEARCH(IMG-FEATURES, MODEL, β,κ, α)
  population ← [$< startseq >$] ▷ 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 ← `model.predict(img-features, caption)` ▷ Vector of predicted probabilities
      Add top κ predicted words to caption to create κ new candidates
    Truncate population top β candidates, according to quality metric.
    $i \leftarrow i + 1$
  return *population*

on to 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" 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. 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 $\beta = \kappa = \alpha = 1$.

VI. EXPERIMENTS

Before training, I randomly set aside 20% of the 100,807 images with captions to form a validation set; the remaining 80% form the training set. I kept this split consistent in all experiments, so that I could gauge the relative effectiveness the different models and hyperparameter configurations.

Although there are many hyperparameters that should be tuned in both the inject and merge models[11], I only experimented with different combinations of model architectures and α - discussed in section 5.E. For each of the inject, merge-concat and merge-add models, I generated and evaluated predictions using beam search with $\alpha \in \{.6, .7, .8\}$, leading to 9 total fits.

A. Training details

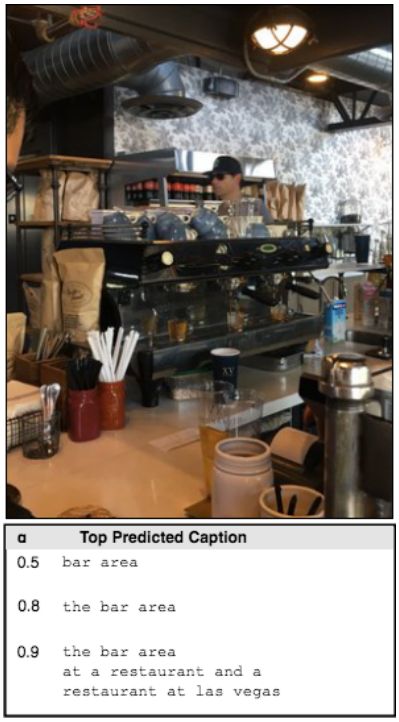
I trained all three model architectures using stochastic gradient descent with Nesterov momentum, with the learning rate $\eta = .01$, and the momentum term $\mu = .9$, and a learning rate decay rate[12] of 10^{-6}. I did not tune these hyperparameters, as I observed that the training loss converged steadily with this configuration [figure 9]. I used categorical-crossentropy as the objective function.

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

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

[11] See section 8 for a discussion of hyperparameters to tune in future work

[12] At iteration t, the learning rate η is updated: $\eta \leftarrow \frac{\eta}{1+t\cdot 10^{-6}}$.



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.

I trained each model for 30 epochs. I used an early stopping scheme - stopping training when validation loss does not decrease for two consecutive epochs. Each training epoch took around 8 minutes on an NVIDIA Kepler GK104 GPU, hosted on Amazon's AWS EC2 service.

Both the merge-concat and merge-add models converged faster than the inject model; after 18 epochs the early stopping scheme suspended training. On the other hand, training loss decreased much more slowly for the inject model, and training continued for the full 30 epochs. The inject model achieved the lowest validation loss at the end of training - evidence that although it is slower to train, the inject model does a better job of generalizing to new examples.

VII. Results

In this section, I explore the efficacy of each of the nine NIC system configurations I discuss in section VI. I discuss two metrics for quantitatively evaluating the performance of a NIC system - *BLEU* and *ROUGE*, and compute these metrics on each configuration's predictions. I then perform a brief qualitative analysis - where I discuss some unexpected behaviors of my system, and their potential causes.

A. Quantitative Analysis

Defining paradigms for evaluating the quality of generated captions remains a challenge is a topic of active research (for example: Vedantam et al.; 2014). Perhaps the most widely adopted metric for evaluating predictions of machine

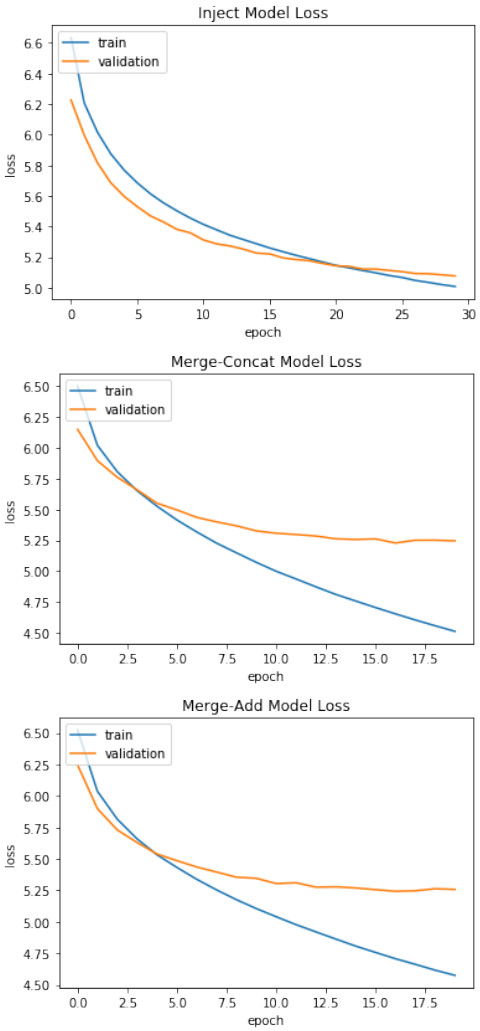


Fig. 9: Training and validation losses for the inject, merge-concat, and merge-add models.

translation and image captioning systems is *BLEU* (Papineni et al.; 2002). BLEU-N works by counting n-grams that appear in both the predicted text and the "true" reference text. Since it is a precision-based metric, short predictions are naturally favored by the most basic version of BLEU. Thus, most implementations (including the one I used[13]) penalize short predictions, and predictions that use only frequently occurring terms.

BLEU is agnostic to n-gram ordering, and thus cannot measure the coherence of a predicted caption. Although it is not perfect, the score is widely adopted partly because it is inexpensive to compute, and it has been shown to correlate highly with human evaluation. I compute BLEU-1, BLEU-2, BLEU-3, and BLEU-4 for each model configuration, as a proxy for measuring how relevant the predictions are to the images' content.

A robust NIC system should not only form predictions that are related to the image content; it should also generate

[13] I used the BLEU implementation from python's `nltk` library.

TABLE I: Validation set prediction quality scores.

Model	α	BLEU-1	BLEU-2	BLEU-3	BLEU-4	ROUGE-L	Terms Generated	Average Caption Token Length
Inject	**.6**	.0687	.0295	.0137	.0095	.0885	501	3.17
	.7	.0693	.029	.0129	.0083	.0962	559	4.31
	.8	.0711	.0283	.0121	.0082	.1172	644	7.57
Merge-Concat	**.6**	**.084**	**.0384**	**.0207**	**.0128**	*.1133*	*1191*	6.41
	.7	*.0832*	.0367	*.0198*	*.0125*	*.1229*	*1294*	8.1
	.8	*.0781*	*.033*	*.018*	*.0122*	**.139**	**1406**	10.49
Merge-Add	**.6**	.0812	.0373	.0192	.0125	.1096	961	4.96
	.7	.0782	*.0338*	.0171	.011	.1197	1058	7.31
	.8	.0735	.0304	.0152	.001	.1311	1147	9.98

BLEU-N, ROUGE-L and number of terms generated for each model configuration. Boldface represents the highest score achieved amongst any model configuration for that metric. Underscore represents the best score achieved for all models with the same architecture (inject, merge-concat or merge-add), and italics represent the best score achieved amongst configurations with the same value of α.

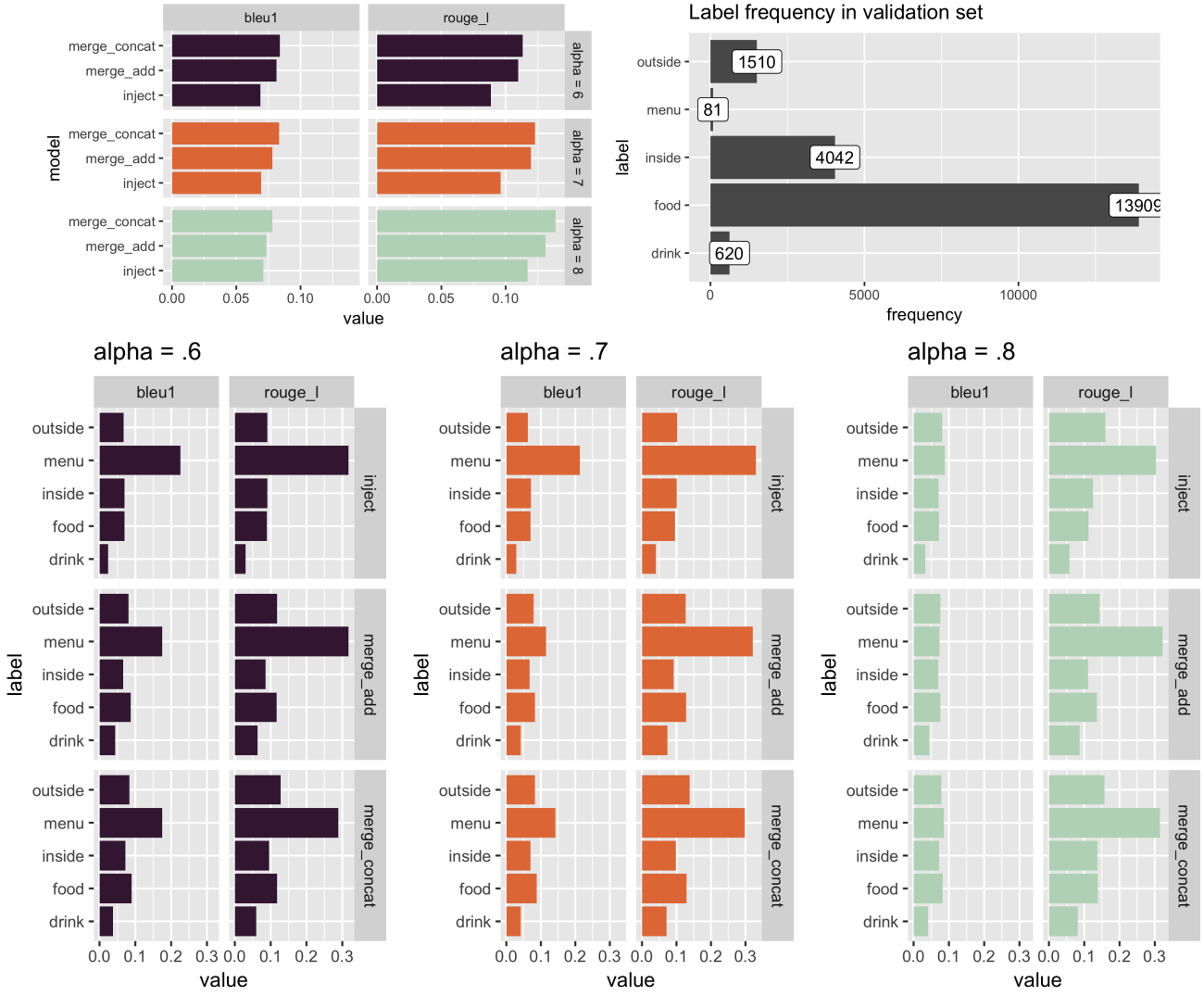


Fig. 10: **Top-left:** BLEU-1 and ROUGE-L scores grouped by α, to highlight differences between model architectures. **Top-right:** Label frequency in validation set. **Bottom:** BLEU-1 and ROUGE-L achieved by each model on validation examples grouped by label, split by value of α.

diverse captions and utilize a substantial vocabulary, and describe multiple objects/relationships in images. Thus, a metric which complements the precision-based BLEU is the recall-based *ROUGE* (Lin; 2004). I report ROUGE-L as a proxy of the diversity of the captions generated by each model configuration. I also report the number of unique terms generated by each configuration, as a measure of the vocabulary size utilized by each model [table 1].

In general, large values of α yield high ROUGE scores, and low values of α yield high BLEU scores, holding the

model architecture fixed. This is as expected, as high values of α yield more wordy predictions, which leads to higher recall, but lower precision.

Interestingly, the merge-concat and merge-add models yield a much larger vocabulary than the inject model, and higher ROUGE scores. In fact, for any given α, the merge-concat yields a vocabulary that is more than double the size of that of the inject model. Notice, however, that the average length of captions generated by the merge models are much larger than those of the inject model. Thus, the fact that the merge models yield superior ROUGE scores and larger vocabulary sizes may suggest that these models indeed tend to utilize richer vocabularies. It may also suggest that merge models are more prone to generating "wordy" captions, and perhaps more strict regularization (smaller α) would lead to predictions more similar to those generated by the inject models.

The best BLEU scores are also achieved by the merge models, which suggests that these models are more likely to predict words that are present in the ground truth.

It's also interesting to note the differences in BLEU and ROUGE scores achieved by the models on validation examples from different categories [figure 10]. For $\alpha \in \{.6, .7\}$, all three model architectures achieved substantially higher BLEU-1 and ROUGE-L scores for images in the *menu* category, though this does not hold true for $\alpha = .8$. This result is unreliable, however, as only 81 images (0.4%) of the images in the validation set are labeled *menu.* Of the remaining 4 categories which are better populated, the models achieve the smallest BLEU and ROUGE scores in for examples in the *drink* section, which may suggest that images with this label are harder to predict accurately.

B. Qualitative Analysis

Although the study of BLEU and ROUGE scores [table 1, figure 10] would lead one to believe that the merge models perform better than the inject models, manually reviewing these models' predictions reveals that the inject model performs much better in practice.

The merge models' predictions are often long and insensical, and repeat common phrases like *"special offer, limited time only"* and *"with a side of mac n cheese."* My theory for why this occurs is that the merge models are severely overfit, as evidenced by the training trajectories[14] in figure 9. Thus, subtle features in an image may cause a merge model to regurgitate a sequence of words which are present in the training data with high confidence, even though those words do not relate to the image at hand.

I found that the system generated the most relevant and appropriately lengthed predictions when using the inject architecture, with $\alpha = .6$. This is the configuration used in the final system.

Although the inject architecture is more well-behaved than the merge models, it too exhibits bizarre behavior - most notably its eagerness to label images as **chicken and waffles.** Of the 20,162 validation images, the final system predicts that 3,866 of them (19.2%) contain the string "chicken and waffles" [figure 11].

This is not a result of the images' content, but rather of the sparseness of the training captions, and the method with which I train the network.

Consider a naive agent which completely ignores the image at hand when making a prediction, and behaves as an n-gram based language model. Starting with no prior image nor caption information, the most reasonable[15] prediction for the first word in a sequence is the word that leads the training captions most frequently - `the`, followed by the second most common word to lead the training captions - `chicken` [figure 12]. If one were to use this agent to generate captions using beam search with a beam width $\beta \geq 2$ (as I have), then the population after the first iteration would contain the partial caption `[chicken]`.

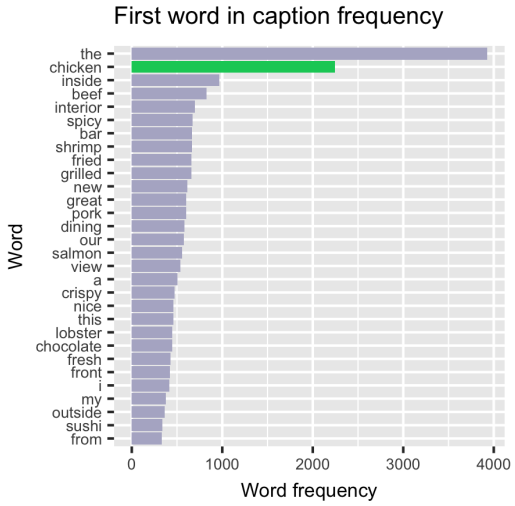


Fig. 12: Words that lead training captions most frequenlty.

To select the second word to add to this caption, the agent would find the word that is preceded by the word `chicken` most frequently in the training captions, which is `and` [figure 13]. Similarly, the next iteration of the beam search would lead to the word `waffles` being added to the caption, as it is the second most frequently word to follow the bigram `chicken and` [figure 14]. Thus, the string `chicken and waffles` is generated by the naive agent.

[14]For the merge models, the training loss is much lower than the validation loss - a clear sign of overfitting. This phenonmenon is not as severe for the inject model.

[15]If we assume the first word of a caption does not depend on subsequent words, this is the maximum likelihood estimate.

Fig. 11: Sample images the final system predicts are "chicken and waffles." The system predicts that around one in five images - often times or images that look nothing like the American dish. This behavior resembles a policy of *"when in doubt: chicken and waffles."*

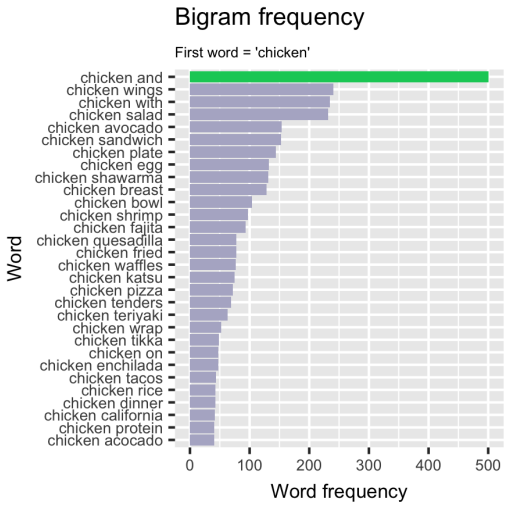


Fig. 13: Most common bigrams - given that the first word is `"chicken"`.

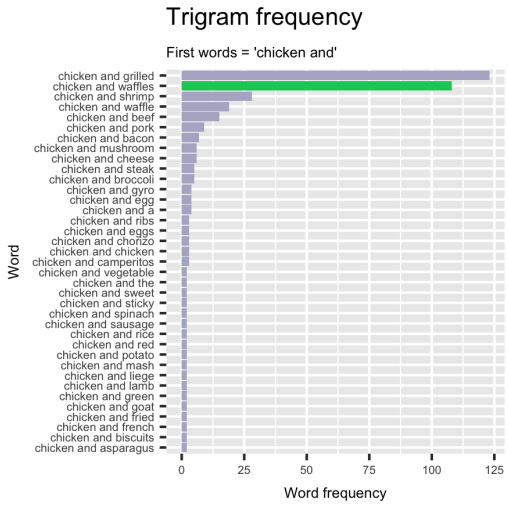


Fig. 14: Most common trigrams - given that the first two words are `"chicken and"`.

Thus, an intuitive explanation as to why my NIC system emits the string *"chicken and waffles"* so frequently isn't that it sees chicken and waffles in the image. Instead, *it has no idea what is in the image*, and so it predicts the string *"chicken and waffles"* because it behaves as a naive n-gram based language model.

This problem is exacerbated by the sparseness of the token frequency table the training captions generate. Token frequencies follow a power rule with a very long tail, which motivates the language model component of a NIC to predict the few frequently occurring n-grams, regardless of an image's content.

VIII. CONCLUSION

In this technical report, I've discussed the inner workings of a simple neural image captioning system. After a short time playing with the application, it becomes clear that the system has the capacity to detect objects in an input image, and form a sequence of natural language that is conditioned on the image content.

The system I've built is far from perfect, however, mainly due to the lack of due dilligence in my methodology. In my experiments, I compare three NIC model architectures, and conclude that the "inject" architecture (section 5.D) generates the most relevent and succinct predictions. This result is not legitimate, however, as I did not thoroughly tune the hyperparameters associated with each model. In future work, some of the important hyperparameters I would like to tune are:

- Hyperparameters associated with the optimization scheme, such as the learning rate η, batch size, momentum ter μ, etc.
- Model architecture and the complexity of each model, for example by tuning the number of hidden layers, the output dimension of each dense layer, the hidden state size of the LSTM layer, etc. This is especially critical for the merge models, who display clear evidence of overfitting [figure 9].
- Different methods of transfer learning - for example by experimenting with networks other than VGG16 for encoding images, and using word embedding models other than Word2Vec. It may prove useful to allow the network to modify pre-trained word embeddings, or to learn word embeddings from scratch.
- Use ofs regularization to control overfitting.
- Ignoring terms that have a corpus frequency lower than a given threshold, as a means of reducing the vocabulary size of the training captions.

Besides tuning these hyperparameters, a more important and difficult task I would like to undertake in future work is to process the Yelp dataset so that it is more consistent and "well defined." Currently, the captions of the image captions are irregular; a caption may describe the content of an image, a user's experience at an establishment, or advertise a restuarant's impressive menu and social media presense. One cannot expect a system behave consistently after it is trained on inconsistent data, hence the phrase *"garbage in, garbage out."* Refining the Yelp dataset for the purpose of building an image captioning system, for example by categorizing images by the semantic content of their captions, is an undertaking that warrants its own separate - and likely very beneficial - research project.

Due to the rise of user friendly deep learning frameworks, public access to pre-trained neural network models and word embeddings, and inexpensive cloud computing services, anyone can build applications that would have been considered science fiction a decade ago in a few hundred lines of code and a modest time/price budget. This democratization of artificial intelligence will no doubt bring a great deal of value to companies like Yelp who embrace the public's creativity, and enable them to build useful systems by sharing their data. I hope that the work I've presented in this paper will inspire novice machine learning practitioners to take on unconventional machine learning tasks that are not yet well-understood, such as neural image captioning. Furthermore, I hope that this work will motivate internet companies to share their data, and invite the public to share their work and express their creativity. I believe that in this new age of democratized artificial intelligence, companies who take on this transparent and open mindset will enjoy substantial value generated from the masses of passionate learners.

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.

[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

[16] K. Papineni, S. Roukos, T. Ward, and W. jing Zhu, "Bleu: a method for automatic evaluation of machine translation," 2002, pp. 311–318.

[17] C.-Y. Lin, "Rouge: a package for automatic evaluation of summaries," July 2004. [Online]. Available: https://www.microsoft.com/en-us/research/publication/rouge-a-package-for-automatic-evaluation-of-summaries/

[18] 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