Text2Scene Generation
Discriminator & Brutal-force search

Abstract

1 Task

Generate a consistent picture given a query text, using learned discriminators as score functions

2 Dataset

The dataset contains clip art pictures and corresponding textual descriptions. A picture is composed of several layers, while a layer can be either one among the four basic types, including background, character, surrounding and decoration. A character or surrounding layer can be further classified into several categories. Each category is attached with exclusive identification code for the convenience of encoding. The detailed category table is shown in Table. 1 and 2.

Table 1: Nested categories that a surrounding layer can be divided into.

room				landscape				
object				abstraction	city			nature
furniture	appliance	wall	sundries	chart art	street	building	vehicle	park wild

Table 2: Nested categories that a character layer can be divided into.

interaction						person					
occupation	abstraction	emotion	sociable	travel	enjoyment	stand		sit	sport	show	gesture
						lean	back front				

For simplicity we will not consider the positions of objects, and the layout of a picture will only touch the layer level. Thus the dimensions of the layout are restricted to the specific categories of the layers, as well as the order of overlapping between layers. In addition, several rules need to be followed when composing a picture. Specifically, the background layer should lie below all other layers while the decoration layer should be on top of all other layers. Either a character layer or a surrounding layer should be contained in a picture, if not both.

There exists prominent biases in the dataset. For instance, the decoration layer rarely appears in the pictures. Occurrences of the categories are also strongly biased. Though at the same level, "chart" appears 19 times in all images, while "appliance" or "vehicle" only appears once. For character categories, "stand" appears 17 times, while "emotion" only appears once. As a reference, Fig. 1b shows the occurrences of category keywords across all pictures. The imbalanced distribution of categories will strongly impair the performance of the model. Other biases include that, in almost

all cases, the character layer is in front of the surrounding layer. To avoid it becoming too easy to discriminate real and fake pictures, here let us remove this dimension in the composition of the picture. The character layer is now always in front of the surrounding layer. The occlusion ambiguity is thus resolved and our task basically turns into an image retrieval problem. Given a description, we are trying to retrieve the most relevant character and surrounding, and at the same time the character and surrounding should be consistent with each other. Note that to retrieve a background or a decoration is trivial. Since the backgrounds and decorations do not have finer tags, the retrieval is entirely random.

As a reference, there are in total 668 possible combinations of layers in consideration of different categories. The number of combinations ever appear in the dataset is near a hundred. Thus there are quite a lot combinations that may not be reasonable.

At this moment 92 pairs of pictures and descriptions are available. The above investigation and the following training and evaluation are all based on these pairs.

3   Similarity

3.1   Knowledge base

3.1.1   Layer-based collocation knowledge

Let o be an object or subject, a be an action. $\mathcal{C}_L(o) = \{a_i\}$ represents the set of all actions that syntactically connected to object or subject o. In other words, for each action in the set, a_i and o co-occurs in some layer. Similarly, $\mathcal{C}_L(a) = \{o_i\}$ represents the set of all objects syntactically connected to action a.

3.1.2   Picture-based collocation knowledge

Let o be an object or subject, $\mathcal{C}_P(o) = \{o_i\}$ represents the set of objects or subjects that pictorially connected to o. In other words, for each element in the set, o_i and o co-occurs in some picture.

3.1.3   Picture-based subject collocation knowledge

Let $l = \{o_i\} \cup \{a_j\}$ be a layer that contains object or subject set $\{o_i\}$ and action set $\{a_j\}$. $\mathcal{C}_P(l) = \{l_k\}$ represents the set of all layers that co-occur with l across the picture dataset.

3.2   Keyword-Keyword similarity

3.2.1   WordNet

3.2.2   ConceptNet

relatedness

3.2.3   Layer-based

ConceptNet supports the measure of similarity between any two words. However, frequent collocations in the layer names may not be closely related to each other in terms of ConceptNet relatedness. For example, "present" and "chart" co-occur frequently in the layer names, but their ConceptNet relatedness is quite low, at least significantly lower than "present" and "box", which never co-occur in the layer names. Moreover, ConceptNet relatedness treats words with different POS tags indifferently. For example, the relatedness of "chart" with "present" as a verb and "present" as a noun is the same. This causes prominent ambiguity when it comes to measure the similarity of collocations.

As a consequence, we need a definition of similarity specific to our dataset. This similarity needs only to support inter-POS words because the semantic relatednesses between words with same POS tags are explicit. Here let us define the keyword-keyword similarity with layer-based knowledge sim_L, which utilizes the existing collocations in the layer name repository. Specifically,

$$\text{sim}_L(o, a) = \frac{1}{2}\left[\max_{o_i \in \mathcal{C}_L(a)} \text{sim}(o, o_i) + \max_{a_i \in \mathcal{C}_L(o)} \text{sim}(a, a_i)\right],$$

where $\text{sim}(o_1, o_2)$ and $\text{sim}(a_1, a_2)$ can either be WordNet *lch* similarity or ConceptNet relatedness.

(They should be similar on both sides then can say similar, may be harmonic mean?)

(use frequency as weights? let's do this)

(layer level reality? otherwise what's the point. we will compare the similarities between true layers anyway. Within a layer the action-object always fit because it's true.)

3.2.4 Picture-based

We have endorsed collocations that are uncorrelated in external repositories but specifically correlated in our dataset by utilizing knowledge at layer level. But this is still insufficient to reflect the reality of our dataset. For instance, there are a number of pictures in our dataset that refer to a man or woman stands next to an abstracted chart, where the layers will include the character part, i.e. a man or woman stands somewhere, and the surrounding part, i.e. a chart which has some specific elements such as lists or graphs. A comparison between these two layers with mere WordNet or ConceptNet keyword similarities will easily lead to the conclusion that these two layers do not fit with each other, because keywords in these two layers are indeed uncorrelated. To support the reasonability of the collocation of such two layers, let us define the keyword-keyword similarity with picture-based knowledge sim_P, specifically

$$\text{sim}_P(o_1, o_2) = \frac{1}{2}\left[\max_{o_i \in \mathcal{C}_P(o_2)} \text{sim}(o_i, o_2) + \max_{o_j \in \mathcal{C}_P(o_1)} \text{sim}(o_1, o_j)\right].$$

We only want to encode the relatedness of two entities that co-occur in a picture, but not the relatedness of an action in one layer with an entity in another layer, though in the same picture. Therefore, the definition here only applies to entities, namely objects or subjects.

3.2.5 Aggregate

think should concatenate, such that some special conditions will not gain excessive weights

$$\text{sim}(w_1, w_2) = \text{sim}_R(w_1, w_2) + \lambda \text{sim}_L(w_1, w_2) + \eta \text{sim}_P(w_1, w_2)$$

4 Feature

4.1 Layer-layer similarity

5 Feature engineering

We decide to build the discriminator with either a simple logistic regression or boosting model to . Thus feature engineering is of great importance in the pipeline. Basically we have to deal with three parts of features in this problem, namely those encoding the textual description, those encoding the picture layout, and the additional ones that jointly encode the relations between the description and the picture.

To deal with the text, we first introduce a tokenizer that transform a sentence into a series of keywords. Specifically, the sentence will be processed into tokens by the *nltk* standard tokenizer, which are then fed into the *nltk* standard POS tagger to identify their associated POS tags. Punctuations and stop words will be removed afterwards. A lemmatizer introduced by WordNet is used to lemmatize the tokens based on their POS tags. And the lemmas are further transformed into WordNet synsets, in aid of the calculations of similarities. Word sense disambiguation will not be mentioned here. Obtained synsets now serve as the keywords in our vocabulary. Unseen tokens will be discarded directly, which is obviously not reasonable and will be fixed later. Once we manage to represent a sentence using keywords included in the vocabulary, a simple Tf-idf vectorizer can be introduced to encode it. Both unigrams and bigrams will be used. The idfs of all keywords in the vocabulary will

Table 3: Features introduced to encode text, picture and them jointly. The bottom row indicates the size of a particular type of feature. Idf means this particular feature is reweighted by the idf of the associated token or category.

text			picture				joint
token frequency (idf)		# layers	layer occurrence (idf)	category occurrence (idf)		category-category normalized similarity (idf)	token-category normalized similarity (idf)
unigram	bigram			character	surrounding		
135	163	1	4	16	17	272	9834

be scaled linearly such that they range from 0 to 1. As a reference, there will be in total 298 keywords in the vocabulary, including bigrams. The occurrences of keywords are biased prominently. In Fig. 1a we list the keywords that occurs more than once. In fact, there are only 39 such keywords in the entire description corpus.

(a) Keywords that occurs more than once in the description corpus

(b) Occurrences of category keywords across all pictures

Figure 1: Histograms of keyword occurrences

Next we encode the pictures, or as a matter of fact, only the nested categories a picture belongs to, since we will not touch pixel-wise information, nor any position or attribute information. As mentioned before, there's no finer category to differ the backgrounds or the decorations. Therefore these two layers can be simply encoded to binaries, telling if the layer exists in the picture or not. The character and surrounding layer can be encoded similarly, with two binaries telling if they exists, and two additional binary vectors where each entry indicates whether this specific category gets mentioned or not. There will be a binary vector of length 16 to encode the categorical tags of a character layer, and a vector of length 17 to encode the tags of a surrounding layer. There will also be a feature telling how many layers there are in the picture. Finally, to facilitate the evaluation of the reality of a picture, similarities between each category name associated with character and that associated with surrounding will be counted as features. Here we will use the WordNet *lch* similarity, and the category names will be transformed into most relevant synsets manually. Those similarities will be z-scored such that they have zero mean and unit standard deviation. Consequently, there will be in total 310 features to encode a picture.

Now let us consider the encoding in a different view. Since the pictures only differ from each other in the layer names, as well as the category names each layer associated with, we can actually treat these tags in whole as the image caption. Each picture is then simply a bag of words. And our task becomes to find the most relevant keywords given a description, subject to some constraints that describe the nesting relations. Following this point, we can introduce the tf-idf vectorizer here as well, to suppress the contributions of some most frequent categories. Scaled Idfs can take place of the binary vectors that indicate the occurrence of the layers and categories. They can also reweight the pairwise similarities as mentioned before. The effect of this trick will be discussed later.

In the end, we need to jointly encode the pictures and the textual descriptions such that the consistency between them can be interpreted. We can simply encode this information by constructing a binary table with the row containing all the keywords from the text, and the column containing all the keywords from the picture categories. Each entry then indicates whether a textual keyword and a picture keyword cooccur in a picture-description pair. Similar as before, we can also introduce the *lch* similarity between keywords. As shown in Fig. 2, these similarities should be z-scored, otherwise dissimilar keyword pairs will take little effects in inferring the discrepancy between pictures and descriptions. Note that here we will not count the bigrams in the vocabulary. Moreover, *idf* reweighting can be introduced here as well. Idfs of both textual keywords or categorical keywords can be multiplied with pairwise similarities to suppress highly frequent keywords.

Figure 2: The histogram of similarities between text keywords and picture keywords.

Figure 3: Noticeable features from pictures, descriptions and them jointly

As a reference, Table. 4 lists the main features types and their size introduced by the above three encoders. Fig. 3 shows some noticeable features in the three encoders respectively. Values of textual features are related to specific documents in terms of term frequency. Therefore here we show the top features that are penalized by *idf*.

6 Model and evaluation

In order to fulfill the text to picture generation task, our idea is to train two discriminators, one to tell consistent description-picture pairs from inconsistent ones, and the other one to tell if a picture is fake or not. These two discriminators can be properly combined into a score function. Then given a piece of query text, we only have to find the one that is scored the highest among all the possible patterns that represent a picture.

6.1 Consistency discriminator

To train a consistency discriminator, we need to build a dataset containing both consistent and inconsistent picture-description pairs. A consistent description is simply the one manually generated to describe a picture, while an inconsistent description can be picked from all the descriptions in the corpus randomly, as long as it is different from the original one. Similarly, given a description, we can also retrieve an inconsistent picture from all the pictures we have. Here we do not consider the document-wise similarities between descriptions, nor the picture-wise similarities. Thus we may accidentally retrieve a description very similar to the original one, and to simply label this description as inconsistent may not be appropriate, or at least it may not be as inconsistent as other pairs. A more promising method is to sort all other descriptions with respect to their similarities with the original one. During training, highly dissimilar descriptions are first picked to form inconsistent (negative) pairs, then the less dissimilar ones. This is often known as curriculum learning.

Specifically, to build the dataset, when given a positive example, namely the true picture-description pairs, we can generate two negative examples correspondingly, which are incorporated into a triplet. Each pair is encoded as mentioned above. Textual, pictorial and joint features are concatenated, yielding in total a sparse feature vector of length 10,442. We can shuffle the true pairs before training. But to maintain the triplet structure is crucial to the performance of the discriminator.

Now given the dataset, we can train a simple binary classification model. Logistic regression will be used here because of its interpretability. We will introduce the L1 regularizer to encourage a sparse representation. A negative example will be given a weight only the tenth of that of a positive one, to force the learner to focus on the true pairs.

We randomly choose about 90% examples to train the discriminator, and test on the rest 10% ones. The overall accuracy is 0.4, while the accuracy of positive examples, or equivalently the recall, is 0.9. The latter one is more important in judging the performance, but we still needs a more robust metric to evaluate the model. We can either try the precision-recall curve, or the receiver operating characteristic. It turns out the precision-recall curve is more preferred, since we are handling an imbalanced dataset. We show both two metrics in Fig. 4. The associated average precision (AP) and the area-under-curve (AUC) are annotated on the top. We also show the maximum F1 score obtained given different thresholds. Due to the insufficient data, the performance of the discriminator varies prominently as the test set varies. In all, the average precision is about 0.5, which is acceptable. We should use K-folds cross-validation to obtain a more robust evaluation, which will be tried later.

Figure 4: Precision-recall curve and ROC of the logistic regressor obtained on the test set in discriminating the consistency between a picture and a description

We count the coefficient associated with each feature in the learned logistic regression model as its importance. In Fig. 5a, we list the most important features. There are quite a lot interesting things we can discuss in this bar graph. The most contributive features are often those pairs composed of

(a) Most important features in discriminating the con-
sistency between a picture and a description

(b) Most important features in discriminating the rea-
sonability of a picture.

Figure 6: Precision-recall curve and ROC of the logistic regressor obtained on the test set in dis-
criminating the reasonability of a picture.

exactly same keywords, for instance, "stand" as a category name, paired with "stand" as a keyword
in the vocabulary. Dissimilar keyword pairs can also contribute to the discrimination. For example,
"show" and "play" as a paired feature helps ruling out the character of showing things when seeing
"play" in the description. One can notice that the names of finer categories contribute more, obvi-
ously because they carry more concrete meaning than the general ones. One can also notice that
paired features dominate in the contributive list, which shows the importance of joint encoding. As
an additional note, in Chang 2014, a similar discriminator is trained to obtain the most important
keyword pairs, which are then used to map keywords in the description to categories in the picture.
In this way, we should avoid z-scoring the pairwise similarity features, such that dissimilar pairs will
not appear in the list important features, and the model can focus on leveraging the similar keyword
pairs.

Now let us discuss some specific examples that the discriminator made mistakes on. In one example,
the description is "A woman holds a toy.", while the corresponding picture contains three layers,
including a character, a surrounding and a background. The categories associated with the character
layer are "person" and "sit", while the categories associated with the surrounding layer are "indoor",
"object" and "sundries". The discriminator mistakenly predicts this pair as inconsistent. Among the
non-zero features encoding this particular example, there are quite a few that strongly support the
consistency. "object" and "plaything", the synset name of "toy", for instance, are highly similar and
at the same time quite distinct among the corpus. However, the learned coefficient corresponding
to this feature is 0. In fact, the only two features having non-zero coefficients in this example are
the number of layers and the keyword "woman". Therefore there's no way for the discriminator to
determine the consistency. The problem lie in the shortage of recurrent data.

6.2   Reasonability discriminator

Now let us build a dataset to train the reasonability discriminator. The real pictures serve as positive
example, while negative examples, namely fake pictures must be generated properly. As mentioned
before, there are in total 668 possible combinations of layers that can form a picture. We can
simply remove the combinations that ever appeared in the real pictures from all the combinations.
The rest can serve as fake pictures. These fake pictures may not be entirely irrational, because
first the available pictures are obviously not enough to cover all reasonable combinations, and more
importantly, the reasonability of a picture may not be that easy to tell. After all, the reasonability can
only depend on the consistencies between the character and surrounding layers, and a few category
tags are insufficient to characterize them.

when building the dataset, for each real picture, we randomly choose a fake picture from the fake
pool, and jointly feed them to a logistic regressor. Here only the 330 pictorial features are involved.
Similarly we introduce the L1 regularizer, but the classes will share equal weights since the dataset
is balanced. Note that here we must ensure that the data used to train this discriminator are same as
the training data used above.

In Fig. 6 we show the precision-recall curve and receiver operating characteristic of this discrim-
inator. The model does well, obtaining an average precision of 0.86. In Fig. 5b we list the most
important features. Some are reasonable. For example, "object" as a type of surrounding is often
related to people in "social", and "art" is often related to people "sitting". Nevertheless, there are
features contributing to the decision totally due to bias. For instance, in real pictures the decora-
tion layer does not appear very often, and on the contrast, "wild" surrounding appears very often.
But these should not contribute to the discrimination because the occurrences of particular layers or
categories have nothing to do with the reasonability of a picture.

Figure 7: The performance metric of the generator mF1, as the discriminator weight λ varies

6.3 Generator

Now that we have the two discriminators, to build a generator is trivial. Given a description, we pair it with all possible pictures (category combinations), then use the consistency discriminator to give the probability that a pair is consistent, and the reasonability discriminator to give the probability that the picture is reasonable. The two probabilities are weighted using a coefficient λ. The objective is then to find the picture yielding the maximum probability, namely

$$c^* = \arg\max_{c \in \mathcal{C}} \left[\lambda P_{cons}(c, t) + (1 - \lambda) P_{reas}(c) \right],$$

where c represent a possible category combination and t is the query description. This task can be simply performed by a brutal-force search.

One important thing left is to find a metric to evaluate the generated picture. This is basically same as to quantitate the similarity between pictures. Here we introduce the category-level precision. We first encode the layers and associated categories into binary vectors, which is performed in the picture encoding mentioned above, but leaving the number of layers and pairwise similarities. Then we can measure the similarity of the generated picture with respect to the true picture by the precision and recall of its binary vector compared to the true one. We take the F1 score as a balanced metric, and can be averaged if evaluated on multiple test examples. Our generator achieves a maximum average F1 about 0.58 on the test set, where $\lambda = 0.8$. Again, the choice of the test set strongly affects the performance and the optimal hyper-parameters as well. In Fig. 7 we show how the performance of the generator varies with different values of λ. One can see that the peak locates in between and near 1, which generally means that the consistency discriminator helps more than the other one, but will be deficient if alone.

Now let us peer into some specific examples that the generator yields good and poor results. To help with the diagnosis, for each discriminator, we sort the non-zero features learned to non-zero coefficients according to the products between feature values and coefficient values. This is basically a ranking of contribution since we are dealing with a linear model. Between the two discriminators we rank their respective features as well, but according to the same product discounted by the associated weights, which is λ for the consistency discriminator and $1 - \lambda$ for the reasonability discriminator. This ranking does not reflect the relative contribution realistically, but is rather acceptable under the first-order approximation of the sigmoid function.

Now first, we present an example reported a high F1 score (> 0.82) and a high confidence[1] (> 0.99) as well, stating "A woman is showing two charts.", with the true picture including a surrounding layer of "chart", a character layer of "gesture", and an additional decoration layer. This is a rather easy example since there are quite a lot pictures in the dataset related to "charts showing". The generator thus grabs a "chart" because of the high contribution of the pair between "chart" in the text vocabulary and "chart" in the category keywords, a credit to the consistency discriminator. It also grabs a "show" because of the feature pair 'show-show'", due to the same reason. Here we have some ambiguity between "gesture" and "show". But they share a common hypernym, thanks to the nested tagging. In general this is a typical example that the paired similarities between text and pictures are well leveraged.

Here is another typical example that the generator is doing right, but not so confident. The description states "A woman puts up a lantern.", while the picture includes a surrounding layer of "wild", and a character layer of "enjoyment". In fact, the predicted picture is correct entirely. But how the decision is made is not that reasonable. In this case, the mostly contributed feature is actually the number of layers. And no similarity feature appears in the contribution list, probably because the tokens in the description are hard to be connected with the category keywords in terms of similarity. It turns out that the generator retrieves "wild" and "enjoyment" merely because these two are weighted high by the learners, with the former in the reasonability discriminator and the latter in the consistency discriminator. And that they are weighted is simply due to their high occurrences in the dataset, with "wild" the fifth most frequent in the character categories and "enjoyment" the

---

[1]namely the sum of the weighted probabilities of the two discriminators

fifth most frequent in the surrounding categories. The most frequent keywords like "stand" and "abstraction" are suppressed because of the idf reweighting. That's how the second class stands out. In all, this example points out the deficiency of our feature engineering. We may try to remove the single tokens in our feature list, and at the same time figure out a better way to jointly encode the similarities between text and picture. This deficiency is also magnified by the sparsity of our dataset. The discriminator may be able to learn the connection between "lantern" and "enjoyment" spontaneously if multiple examples mention them both. We may replace rare words by their hypernyms to encourage recurrence, but cannot ensure that they all agree on the associated categories.

Now let us look at some examples causing poor performance. There is one description stating "A woman sat to rest". The corresponding picture includes a surrounding layer of abstract arts, a character layer of a seated woman, and an additional background layer. The predicted picture is right about the character, which benefits from the contribution from the paired feature "sit-sit", but mistakenly presents a wild surrounding, which is due to its high occurrence as the case in the above example. Here, there is no meaningful connection between the description and abstract arts. But an ideal reasonability discriminator should at least capture the consistency between "sit" and "indoor", thus guiding the generator to retrieve an indoor surrounding. Unfortunately, "sit" is a verb while "indoor" is an adjective, causing an error when using *lch* similarity. And our model simply returns a zero. This indicates that we may either modify the keywords of the categories, or explore a more compatible algorithm to measure similarity.

The next example includes a description saying "A man and a woman are transferring money", and a picture includes only a character layer of "sociable", and an additional background layer. The predicted picture mistakenly includes a redundant surrounding layer. The redundancy of layers is quite common among the predictions. In fact, for the reasonability discriminator, an additional feature with positive contribution, for instance, "wild", will always increase the probability, not to mention the positive contribution of the feature associated with the number of layers. Again we should try removing single-token features, or introducing features that can capture the connection between the description and the number of layers. Another interesting thing in this example is that the predicted picture mistakenly grabs a character layer of "enjoyment", but not "sociable", again due to the occurrence divergence. But in fact the social activity can be inferred from the existences of both man and woman, which probably requires a feature encoding the similarity between the bigram "man woman" and the keyword "sociable". We may improve our engineering to incorporate this part into the features.

The final example is a rather difficult one. The description states "A woman looks in the window", and the corresponding picture includes a character layer of "lean" and a surrounding layer of "street". The generator retrieves a "wall" for this query because of the high similarity between "wall" and "window", leading to an "indoor" surrounding. And it also retrieves a "gesture" because of its contribution in the reasonability discriminator. In this particular case it would be difficult even for human to infer from the sentence it is an indoor or outdoor environment. But it should be fair to infer the person is standing. This surely requires a deeper understanding of semantics.

7   Ablation test

if remove non-joint features, significantly impair the performance

8   runtime

Since we are using the exhaustive search, one textual query requires to traverse a search space of in total 668 possible pictures, which typically takes 3-6 seconds. The majority of the runtime is spent on encoding the query text and pictures. Our encoder takes a pair of description and picture as input, and is thus called 668 times, which means the description is repetitively encoded. Nevertheless, the time to encode the joint features, i.e. the pair between textual tokens and category keywords, accounts for the main part, mostly due to a redundant for-loop through all the feature pairs. The following table shows the most time-consuming functions.

Table 4: Runtime profile in a hierarchy. The number below each module indicates the cumulative time spent on it, on seconds. Each module here is called 668 times during a query. The number after for_loop indicates the number of pairs this loop traversed.

query 4.092							
encoder (consistency discriminator) 3.508						encoder (reality discriminator) 0.344	...
encoder (joint) 1.972		encoder (text) 0.730		encoder (picture) 0.344	...	encoder (picture) 0.344	
for_loop(33*298) 1.643	tokenizer 0.180 ...	tokenizer 0.427	vectorizer 0.303	for_loop(16*17) 0.285	...	for_loop(16*17) 0.180	...

9 future work

- remove non-joint features to make a baseline
- synonym replacement for unseen words
- extra similarities: cosine, jaccard, lch, wikipedia esa, word2vec