

An Enhanced Sketch2Scene using Natural Language

1. Motivation

Natural Language Processing and Computer vision are both the most actively developing machine learning research areas. Integrating these two fields has received a lot of attention recently. In this project, we would like to combine computer vision and Natural language Processing to transform an incomplete sketch into a scene. Translating a sketch into a scene is a challenging computer vision task. Realistic and meaningful scenes should contain multiple objects as well as a corresponding background, which is hard to extrapolate from a single input object. We propose a new approach to enhance the Sketch2Scene translation task using natural language processing techniques. Instead of directly transforming an image into another, we transform the sketch to a word and then use the word as a condition to create longer sentences and then use it to generate a scene. The first step is to classify the input sketch using a CNN based classifier. Once we have the class of the sketch, we find the most similar words related to that class. We generate a natural language scene description, conditioned on the class, and its similar words. Then we use a separate system that takes in the scene description and generates an image. Below is our scope of the project.

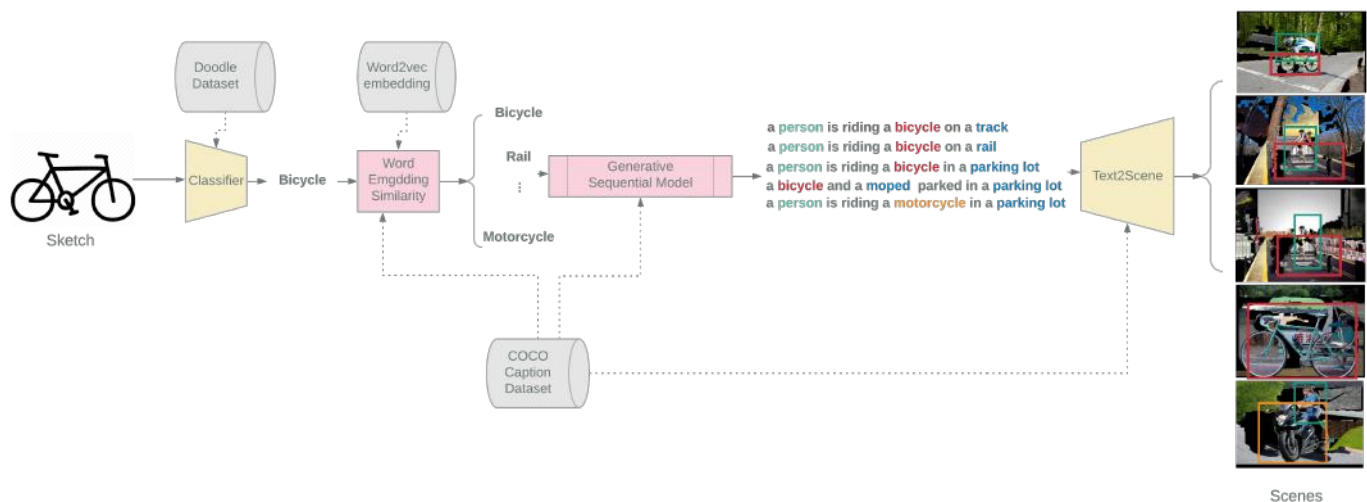
- Implement a new pipeline to generate realistic scenes from a single sketch
- Show how NLP can improve the Sketch2Scene task with respect to quality and diversity

2. Approach

The pipeline in this project contains three main steps to take in an incomplete sketch and transform it into a complete scene. For the first step, we have a pre-trained **MobileNetV2** (<https://arxiv.org/abs/1801.04381>) **sketch classifier** on ImageNet and fine-tuned it on 340 different object classes from the **Google QuickDraw dataset** (<https://www.kaggle.com/c/quickdraw-doodle-recognition>). Given the input sketch, the output class of the classifier is used for the next step, which is the **caption generator**, to find the similar words and feed them into an encoder-decoder based sequential model to encode those words and use them to generate the scene captions. In the last step we infer the scenes using a pre-trained **Caption2Scene Generator** model which learns to retrieve objects and arrange them in the scene using the semantic relationship of the objects in the captions. Our main contribution is to experiment with two different baselines for this task :

- **Baseline 1** - Get noun object from COCO captions and use it as the condition to generate captions
- **Baseline 2** - Get similar words of the predicted class of the sketch, using word2Vec embedding and use as the condition to generate captions.

Baseline 1 is used a sanity check to validate the conditional sequence generative model which will be discussed in section 2.2 . And, for the second baseline we will use different methods like retraining , beam search and train time data augmentations to improve the generated captions.



2.1 Sketch Classifier

Since our inputs are hand-drawn sketches, we decided to use a pre-trained model from [Google QuickDraw Doodle Recognition Challenge](https://www.kaggle.com/c/quickdraw-doodle-recognition) (<https://www.kaggle.com/c/quickdraw-doodle-recognition>). This competition was released as an experimental game on sketches, and their dataset contains 50M drawings encompassing 340 label categories. For the first stage of our pipeline, we chose one of the [Pytorch Implementations](https://github.com/adam9500370/Kaggle-QuickDraw) (<https://github.com/adam9500370/Kaggle-QuickDraw>) from this challenge to work with. We didn't use the pre-trained model from this implementation. Instead, we made changes to the data loader and the train code to fit it for fine-tuning a Pytorch pre-trained model on the doodle dataset:

- changed the input size to 256*256
- changed the last layer of the model
- changed the normalization
- fixed bugs in the original code

We first tried to fine-tune the pre-trained [DenseNet](https://arxiv.org/abs/1608.06993) (<https://arxiv.org/abs/1608.06993>) model, but training took more time than we expected, so we trained [MobileNetV2](https://arxiv.org/abs/1801.04381) (<https://arxiv.org/abs/1801.04381>). We chose MobileNet because it is faster than DenseNet and it also maintain the accuracy.

Hyperparameters

- epochs = 11
- batch_size = 64
- workers = 2
- l_rate = 1e-3
- weight_decay = 1e-4
- seed = 1234

2.2 Caption Generator

Based on [PyTorch Image Captioning Tutorial](https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning) (<https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning>) code, we discarded the image encoder and attention module, and then added our keywords embeddings that are obtained from the object name of user sketch. We trained the decoder based on the keywords-caption pairs which we constructed from COCO caption dataset. The main challenge of this part is how to obtain better quality and more diverse captions conditioned on keywords. To tackle this problem, we defined the baseline model and built up our model using some natural language technique that we have learned from the CMPT825 course. Also, we evaluated our outputs compared to the baseline using three metrics: BLEU4 score, Self-BLEU4 score, and Semantic Accuracy.

```
In [1]: import os
import sys
sys.path.append("./CaptionGenerator")
os.chdir("./CaptionGenerator")
import json

import utils
from create_inputs import create_doodle_vocab, create_input_files, create_input_embeddings
from train import Trainer
import evaluation

import inference
import sacrebleu
from nltk.translate.bleu_score import corpus_bleu
```

2.2.1 Baseline 1

- **[Step1] building keywords-caption pairs to generate Train/Valid/Test datasets** \ In COCO caption dataset, it has 5 captions per each image. So we sample **3 ~ 5 nouns of 5 captions as keywords** based on word frequency. For example, the baseline 1 model can output five different captions with respect to the single class name of "castle". As you can see, other keywords are different for each caption since we sample them using GloVe Embeddings.

['castle', 'cathedral', 'ruins'] a castle with a castle in the background \ ['castle', 'cave', 'tower'] a large castle with a castle in the background \ ['castle', 'hill', 'tower'] a castle with a castle on it \ ['castle', 'hill', 'kingdom'] a large hill with a castle in the background \ ['castle', 'fort', 'cathedral'] a large cathedral with a castle and a castle

- **[Step2] training and validation** \ We initialize the hidden state of the decoder with the keywords embeddings, and calculate Cross Entropy Loss between the generated caption output and the ground truth caption.

Hyperparameters For training we used the following hyperparameters:

- keyword_size=3 (5)
- epochs=10
- batch_size=64
- workers=1
- decoder_lr=4e-4
- embedding_dimension = 512

```

In [2]: #####
base_name = 'baseline'
keyword_size = 3 # 5
#####
#####takes more than 3 hours to complete below #####
****
'''

# [Step1] build Keywords-Caption pairs to generate Train/Valid/Test d
atasets
import nltk
nltk.download('averaged_perceptron_tagger')
create_input_files(base_name=base_name, keyword_size=keyword_size, ca
ption_json_path='./data/dataset_coco.json')

# [Step2] train and validate
trainer = Trainer(base_name=base_name, keyword_size=keyword_size,
                  epochs=20, batch_size=64, workers=1, decoder_lr=4e-
4,
                  checkpoint=None )
trainer.run()
'''

#####
****
# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_KEYWORDS_coco_{}_{}.j
son'.format(base_name, keyword_size)), 'r') as j:
    keywords = json.load(j)

# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_CAPTIONS_coco_{}_{}.j
son'.format(base_name, keyword_size)), 'r') as j:
    captions = json.load(j)

# Load word map (word2ix)
with open(os.path.join('data', base_name, 'WORDMAP_coco_{}_{}.json'.f
ormat(base_name, keyword_size)), 'r') as j:
    word_map = json.load(j)

# print noun keyowrds
for i in range(10,15):
    print(utils.convert_idx2word(word_map, keywords[i]), ' '.join(uti
ls.convert_idx2word(word_map, captions[i])))

```

```

['train', 'man', 'bicycle'] a person is riding a bicycle but there is
a train in the background
['train', 'man', 'bicycle'] a guy that is riding his bike next to a t
rain
['train', 'man', 'bicycle'] a red and white train and a man riding a
bicycle
['train', 'man', 'bicycle'] a man riding a bike past a train travelin
g along tracks
['train', 'man', 'bicycle'] a man on a bicycle riding next to a train

```

2.2.2 Baseline 2

- **[Step1] building Doodle Class Vocabulary** \ Since our model inference captions based on one of the Doodle class names, we require to build the training dataset related to Doodle class names. However, **Doodle has only 340 classes** so that we decide to use **GloVe Embeddings Similarity** to increase the size of vocabulary by **10 times**. Also, some class names that are not present in COCO caption vocabulary are discarded.
- **[Step2] building keywords-caption pairs to generate Train/Valid/Test datasets** \ In COCO caption dataset, it has 5 captions per each image. So we sample 3 ~ 5 keywords from **the intersection of Doodle Class Vocabulary and COCO caption Vocabulary** based on word frequency. Based on the keyword size, some captions that don't have any similar keywords are removed (for keyword_size 3, 7985 of total 122919 samples are excluded). Compared to Baseline 1 ('train', 'man', and 'bike'), it samples different keywords ('riding', 'train', and 'bicycle') given the same captions.

['riding', 'train', 'bicycle'] a man on a bicycle riding next to a train \ ['riding', 'train', 'bicycle'] a man riding a bike past a train traveling along tracks \ ['riding', 'train', 'bicycle'] a person is riding a bicycle but there is a train in the background \ ['riding', 'train', 'bicycle'] a red and white train and a man riding a bicycle \ ['riding', 'train', 'bicycle'] a guy that is riding his bike next to a train

- **[Step3] training and validation** \ We initialize the hidden state of the decoder with the keywords embeddings, and calculate Cross Entropy Loss between the generated caption output and the ground truth caption.

Hyperparameters For training we used the following hyperparameters:

- keyword_size=3 (5)
- epochs=10
- batch_size=64
- workers=1
- decoder_lr=4e-4
- embedding_dimension = 512

```

In [3]: #####
base_name = 'baseline2'
keyword_size = 3 # 5
#####
*****takes more than 3 hours to complete *****
'''

# [Step1] build Doodle vocabulary by generating each 10 similar words
per a class name
create_doodle_vocab(w2v_magnitdue_path='data/glove.840B.300d.magnitud
e',
                    wordmap_path='data/baseline/WORDMAP_coco_baseline
_3.json',
                    out_doodle_path='data/doodle_map.json',
                    topn=10)
# [Step2] build Keywords-Caption pairs to generate Train/Valid/Test d
atasets
create_input_embeddings(base_name=base_name, keyword_size = keyword_s
ize,
                        caption_json_path='data/dataset_coco.json',
                        doodle_json_path='data/doodle_map.json',
                        w2v_magnitdue_path = 'data/glove.42B.300d.magnitud
e')
# [Step3] train and validate
trainer = Trainer(base_name=base_name, keyword_size=keyword_size,
                  epochs=20, batch_size=64, workers=1, decoder_lr=4e-
4,
                  checkpoint=None )

trainer.run()
'''

*****
****
# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_KEYWORDS_coco_{}_{}.j
son'.format(base_name, keyword_size)), 'r') as j:
    keywords = json.load(j)

# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_CAPTIONS_coco_{}_{}.j
son'.format(base_name, keyword_size)), 'r') as j:
    captions = json.load(j)

# Load word map (word2ix)
with open(os.path.join('data', base_name, 'WORDMAP_coco_{}_{}.json'.f
ormat(base_name, keyword_size)), 'r') as j:
    word_map = json.load(j)

# print noun keyowrds
for i in range(10,15):
    print(utils.convert_idx2word(word_map, keywords[i]), ' '.join(uti
ls.convert_idx2word(word_map, captions[i])))

```

['riding', 'train', 'bicycle'] a man on a bicycle riding next to a train
['riding', 'train', 'bicycle'] a man riding a bike past a train traveling along tracks
['riding', 'train', 'bicycle'] a person is riding a bicycle but there is a train in the background
['riding', 'train', 'bicycle'] a red and white train and a man riding a bicycle
['riding', 'train', 'bicycle'] a guy that is riding his bike next to a train

2.2.3 Ours 1: Retrofitting

- **[Step1] retrofitting GloVe Embeddings using COCO captions** \ The generated keywords from the baseline 2 might not be found in the vocabulary of COCO dataset. So our main idea is using retrofitting technique to improve the relations between keywords and captions for both training and inference. For this, we need to extract **noun object relationships from COCO dataset**, and then **retrofit GloVe embeddings** to find Q by minimizing the distance between word vectors in the relationship E (COCO Caption Dataset) as below equation.

$$L(Q) = \sum_{i=1}^n \left[\alpha_i ||q_i - \hat{q}_i||^2 + \sum_{(i,j) \in E} \beta_{ij} ||q_i - q_j||^2 \right]$$

We put $\alpha = 1$ and $\beta = 1$ and use the code from the HW2 with some modifications. We utilize this retrofitted embeddings instead of direct GloVe embeddings.

- **[Step2] building Doodle Class Vocabulary** \ Since our model inference captions based on one of the Doodle class names, we require to build the training dataset related to Doodle class names. However, Doodle has only 340 classes so that we decide to use **Retrofitted GloVe Embeddings Similarity** to increase the size of vocabulary by **10 times**. Also, some class names that are not present in COCO caption vocabulary are discarded.
- **[Step3] building keywords-caption pairs to generate Train/Valid/Test datasets** \ In COCO caption dataset, it has 5 captions per each image. So we sample 3 ~ 5 keywords from the intersection of Doodle Class Vocabulary and COCO caption Vocabulary based on word frequency. **Compared to Baselines, it generates similar to Baseline 1** which has keywords directly extracted from COCO captions rather than Baseline 2. Comparison is below.

Baseline 1 : ['train', 'man', 'bicycle'] \ Baseline 2 : ['train', 'riding', 'bicycle'] \ Retrofit : ['train', 'man', 'bicycle']

- **[Step4] training and validation** \ We initialize the hidden state of the decoder with the keywords embeddings, and calculate Cross Entropy Loss between the generated caption output and the ground truth caption.

Hyperparameters For training we used the following hyperparameters:

- keyword_size=3 (5)
- epochs=10
- batch_size=64
- workers=1
- decoder_lr=4e-4
- embedding_dimension = 512

```

In [4]: #####
base_name = 'retrofit'
keyword_size = 3 # 5
#####
*****takes more than 3 hours to complete *****
'''

# [Step1] retrofit
# create COCO Word Relation files
create_relation(input_json='data/dataset_coco.json',
                output_txt='data/coco-retrofitting.txt',
                max_len=5)
# conver w2v into magnitude file
wordvecfile = os.path.join('data', 'glove.42B.300d')
subprocess.run(["python3", "-m", "pymagnitude.converter",
                "-i", wordvecfile+".txt", "-o", wordvecfile+".magnitud
e"], check=True)
# retrofitting
new_retro_file = os.path.join("data", "glove.42B.300d.retrofit.magnit
ude")
if not os.path.exists(new_retro_file):
    # initialize retrofitting class
    retro = Retrofitting(wv_magnitude_file=wordvecfile+".magnitude")
    # read ontology files
    coco = readLexicon(os.path.join("data", "coco-retrofitting.txt"))
    retro.retrofitting(coco, alpha=1, beta=1)

    # write the final output into Magnitude format
    retro.writeMagnitude_reduced(new_retro_file, 'data/coco/WORDMAP_c
oco_baseline_3.json')
    subprocess.run(["python3", "-m", "pymagnitude.converter",
                    "-i", new_retro_file+".txt", "-o", new_retro_file], check=
True)

# [Step2] build Doodle vocabulary by generating each 10 similar words
per a class name
create_doodle_vocab(w2v_magnitdue_path='data/glove.840B.300d.retrofi
t.magnitude',
                    wordmap_path='data/baseline/WORDMAP_coco_baseline
_3.json',
                    out_doodle_path='data/doodle_map_retro.json',
                    topn=10)
# [Step3] build Keywords-Caption pairs to generate Train/Valid/Test d
atasets
create_input_embeddings(base_name=base_name, keyword_size = keyword_s
ize,
                        caption_json_path='data/dataset_coco.json',
                        doodle_json_path='data/doodle_map_retro.json',
                        w2v_magnitdue_path ='data/glove.42B.300d.retrofit.
magnitude')
# [Step4] train and validate
trainer = Trainer(base_name=base_name, keyword_size=keyword_size,
                  epochs=20, batch_size=64, workers=1, decoder_lr=4e-
4,
                  checkpoint=None )
trainer.run()
'''

```

```

# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_KEYWORDS_coco_{}_{}.json'.format(base_name, keyword_size)), 'r') as j:
    keywords = json.load(j)

# Load encoded captions
with open(os.path.join('data', base_name, 'TEST_CAPTIONS_coco_{}_{}.json'.format(base_name, keyword_size)), 'r') as j:
    captions = json.load(j)

# Load word map (word2ix)
with open(os.path.join('data', base_name, 'WORDMAP_coco_{}_{}.json'.format(base_name, keyword_size)), 'r') as j:
    word_map = json.load(j)

# print noun keywords
for i in range(10,15):
    print(utils.convert_idx2word(word_map, keywords[i]), ' '.join(utils.convert_idx2word(word_map, captions[i])))

```

```

['train', 'man', 'bicycle'] a man on a bicycle riding next to a train
['train', 'man', 'bicycle'] a guy that is riding his bike next to a train
['train', 'man', 'bicycle'] a red and white train and a man riding a bicycle
['train', 'man', 'bicycle'] a person is riding a bicycle but there is a train in the background
['train', 'man', 'bicycle'] a man riding a bike past a train traveling along tracks

```

2.2.4 Ours 2: Retrofitting + Data Augmentation

To increase the diversity of the model, we try to apply **online data augmentation technique** by randomly shuffling keywords among "keywords+2" pairs at each step during training.

```
In [7]: #####
base_name = 'augment'
keyword_size = 3 # 5
#####
#*****takes more than 3 hours to complete *****
'''
create_input_embeddings(base_name=base_name, keyword_size = keyword_s
ize+2,
                        caption_json_path='data/dataset_coco.json',
                        doodle_json_path='data/doodle_map_retro.json',
                        w2v_magnitdue_path = 'data/glove.42B.300d.retrofit.
magnitude')

trainer = Trainer(base_name=base_name, keyword_size=keyword_size,
                  epochs=20, batch_size=64, workers=1, decoder_lr=4e-4,
                  checkpoint=None )

trainer.run()
'''
print()
```

2.2.5 Ours 1+ or 2+: Retrofitting | Data Augmentation + Beam Search

To improve the performance of our model, we add Beam Search, and **find the optimal beam size as 3** for both keyword size 3 and 5. We use **Normalized Log Probability** to count scores and skip the caption when its length is less than 5. Since our HW4 didn't give us better score, instead we implemented based on [PyTorch Image Captiioning Tutorial \(https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning\)](https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning).

When keyword_size = 3, \ Test BLEU Score (beam size: 1): 0.36034294511596937 \ **Test BLEU Score (beam size: 3): 0.3659191816566126** \ Test BLEU Score (beam size: 5): 0.36262749823432455 \ Test BLEU Score (beam size: 7): 0.358677671380397

When keyword_size = 5, \ Test BLEU Score (beam size: 1): 0.40214758432644454 \ **Test BLEU Score (beam size: 3): 0.42702580300893117** \ Test BLEU Score (beam size: 5): 0.4237561271804396 \ Test BLEU Score (beam size: 7): 0.4180983371130172

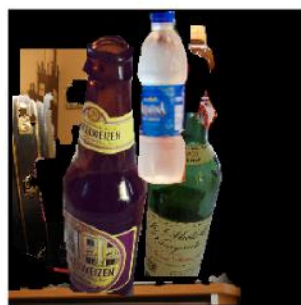
```
In [ ]: *****takes more than 10 min to complete *****
'''
base_name = 'retrofit'
keyword_size = 3 # 5
for b in [1, 3, 5, 7]:
    # Evaluation on Test split
    checkpoint = os.path.join('pretrained', 'BEST_checkpoint_coco_{}_
    {}.pth.tar'.format(base_name, keyword_size))
    references, hypotheses, ref_inwords, hyp_inwords, key_inwords = e
    valuation.evaluate(
        base_name, keyword_size, checkpoint, beam_size = b)
    print("Test BLEU Score (beam size: {}): {}".format(b, corpus_bleu
    (references, hypotheses)))
'''
print()
```

2.3 Caption2Scene Generator

For this part , we used a pretrained model proposed by the paper [Text2Scene: Generating Compositional Scenes from Textual Descriptions](https://arxiv.org/pdf/1809.01110.pdf) (<https://arxiv.org/pdf/1809.01110.pdf>). Text2Scene is a model that generates various forms of compositional scene representations from natural language descriptions. Unlike many recent works , they do not use Generative Adversial Network (GAN) for this task. Text2Scene instead learns to sequentially generate objects and their attributes (location, size, appearance, etc) at every time step by attending to different parts of the input text and the current status of the generated scene . They show that under minor modifications, the proposed framework can handle the generation of different forms of scene representations, including cartoon-like scenes, object layouts corresponding to real images, and synthetic images . That is the reason why we chose this model to convert our captions to scene which will satisfy our motivation to generate realistic looking images which can be used as a dataset to improve object detection and image captioning tasks. We used the pretrained model that generates composite scenes using COCO dataset , as we trained our caption generator based on the same dataset.

As you can see from the below image , this model uses the caption to retrieve objects and position them using the relationship between objects in the caption recursively to generate the final scene.

a bottle of wine sitting next to a bottle of wine.



3. Data

- [Google QuickDraw dataset \(https://www.kaggle.com/c/quickdraw-doodle-recognition\)](https://www.kaggle.com/c/quickdraw-doodle-recognition) Used for sketch classifier
 - 4,960,414 train sketches
 - 102,000 valiation sketches
 - 112,199 test sketches
- [COCO Dataset \(http://cocodataset.org/#home\)](http://cocodataset.org/#home)
 - Captioning 2015 dataset for caption generation
 - Detection 2017 dataset to biuld database for Caption2Scene inference
- [Word2vec embedding \(https://nlp.stanford.edu/projects/glove/\)](https://nlp.stanford.edu/projects/glove/)
 - Common Crawl (42B tokens, 1.9M vocab, uncased, 300d vectors, 1.75 GB download): glove.42B.300d.zip used for searching keywords to generate captions

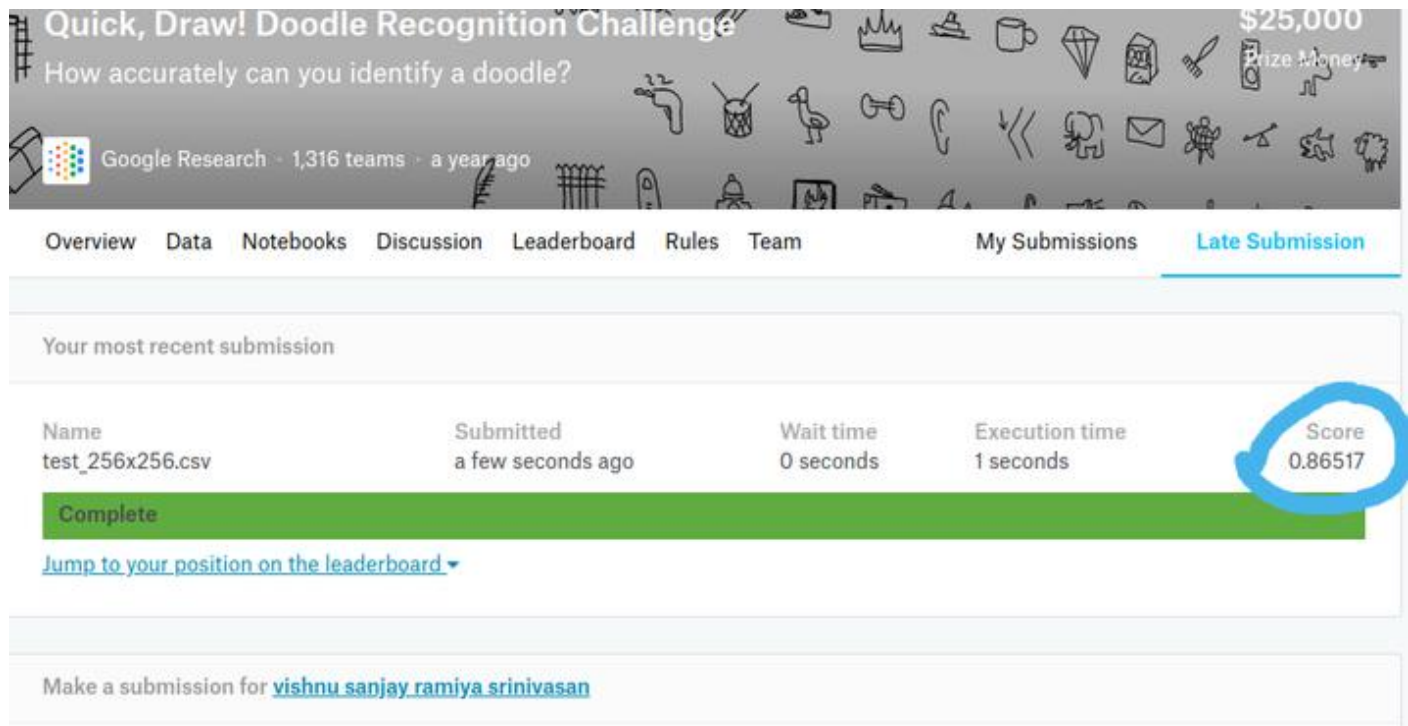
4 Experimental Setup

4.1 Classifier Accuracy

After two days training for 11 epochs on 4,960,414 train images with batch size of 64 we got the following accuracy on 102,000 validation images for top 1,2 and 3 output classes by comparing it with the ground truth:

- top1_accuracy = 74.989
- top2_accuracy = 86.136
- **top3_accuracy = 89.997**

Since for the test file , we dont have the ground truth , we uploaded the outputs to the kaggle competition to get a unbiased evaluation of our model Test accuracy - **.86517** in the competition :



Quick, Draw! Doodle Recognition Challenge

How accurately can you identify a doodle?

Google Research · 1,316 teams · a year ago

Overview Data Notebooks Discussion Leaderboard Rules Team My Submissions **Late Submission**

Your most recent submission

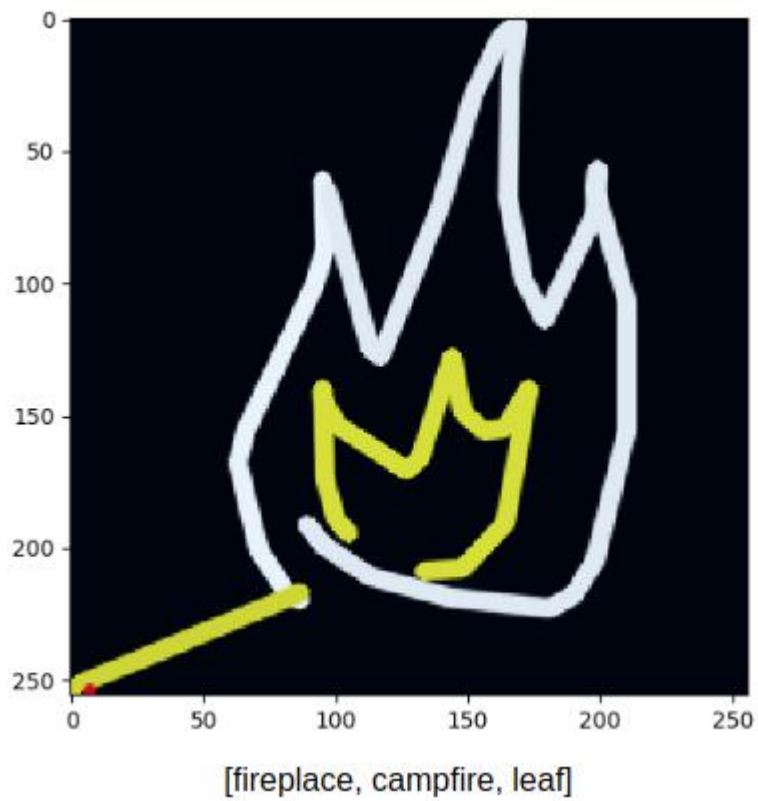
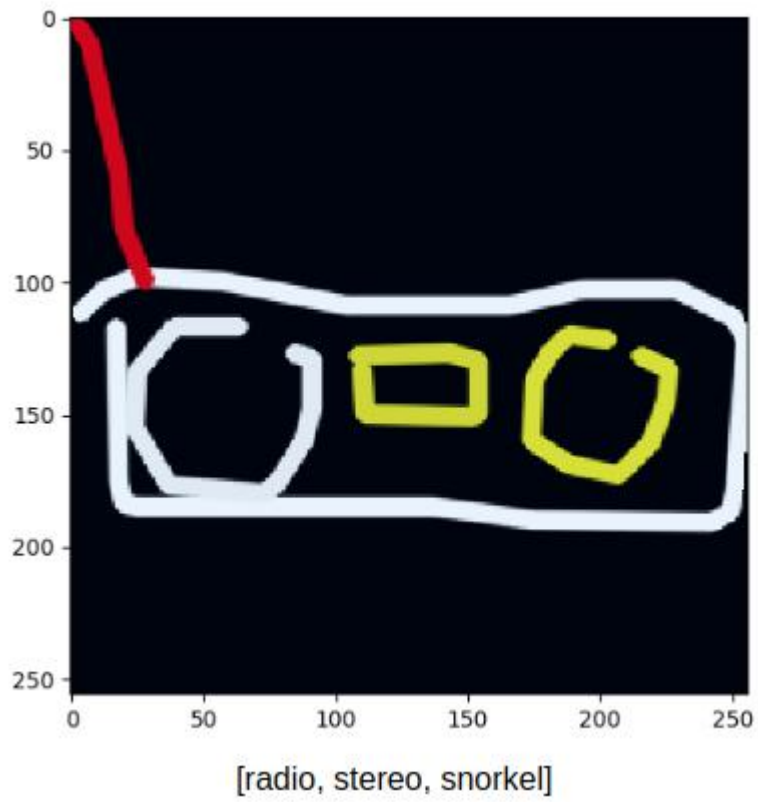
Name	Submitted	Wait time	Execution time	Score
test_256x256.csv	a few seconds ago	0 seconds	1 seconds	0.86517

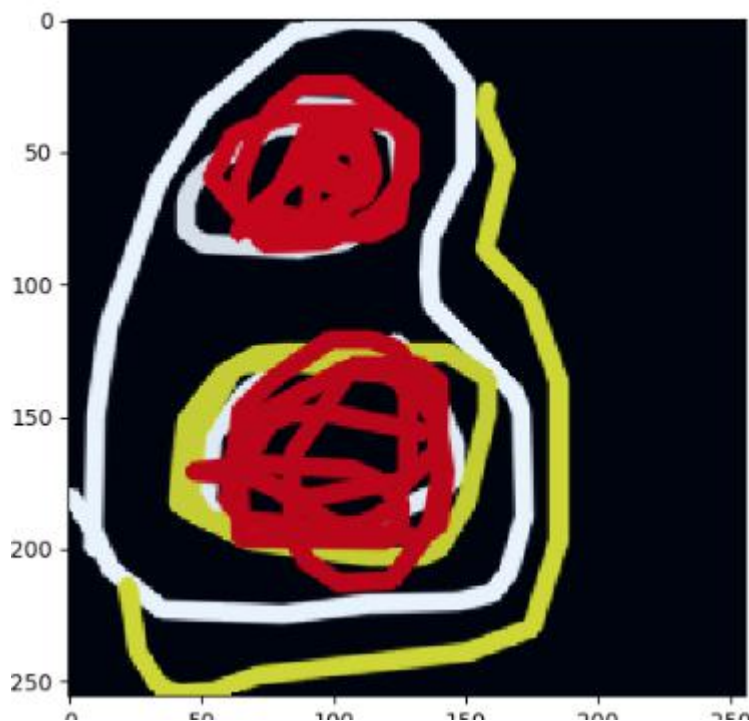
Complete

[Jump to your position on the leaderboard](#)

Make a submission for [vishnu sanjay ramiya srinivasan](#)

Below are some samples from our validation images and their top 3 output classes:





4.2 Caption Generation

We evaluate Our Models(1-3) compared to two baselines **Qualitatively** and **Quantitatively**.

4.2.1 Qualitative Evaluation

Since our goal is to generate diverse captions given a single doodle class name, we write the inference code to produce five different captions by sampling keywords everytime (but we fix the first keyword as Doodle class name). We compare all the outputs from four different models (Baseline 1 and 2, and Ours 1 and 2) for the 'castle' Doodle class input.

- Both **Our methods produces more diverse keywords and captions** compared to both Baselines which generates duplicate keywords and captions.
- Both **Our methods** used the COCO retrofitted word embeddings, so the generated keywords showed **more connection to the generated captions**.
- **Ours 2 showed more diversity than Ours 1** since Ours 2 is trained with the data augmentation.
- Since the keywords of **Baseline 1 and 2** are sampled using GloVe embeddings, **most keywords shows very similar semantic meanings** of input class name such as 'palace'.
- **Baseline 1** is trained on nouns-caption pairs so that it only takes 'castle' keywords to generate captions, resulting in generating **more duplicated captions**.

Baseline 1 (nouns of COCO)

['castle', 'cathedral', 'ruins'] a castle with a castle in the background \ ['castle', 'mansion', 'palace'] a castle with a castle in the background \ ['castle', 'medieval', 'hill'] a castle with a castle in the background \ ['castle', 'hill', 'ruins'] a castle with a mountain in the background \ ['castle', 'hill', 'ruins'] a castle with a mountain in the background

Baseline 2 (GloVe embedding)

['castle', 'medieval', 'mansion'] a castle with a castle in the background \ ['castle', 'ruins', 'mansion'] a castle with a castle in the background \ ['castle', 'ruins', 'palace'] a large castle like building with a clock on it \ ['castle', 'palace', 'cathedral'] a large castle like structure with a castle in the background \ ['castle', 'cathedral', 'tower'] a castle with a castle in the background

Ours 1 (retrofitting with COCO)

['castle', 'gate', 'forest'] a large castle with a sign on it \ ['castle', 'park', 'town'] a group of people in a town square \ ['castle', 'park', 'street'] a street sign in front of a building \ ['castle', 'forest', 'clock'] a large castle with a clock on it \ ['castle', 'inside', 'park'] a group of people inside of a building

Ours 2 (retro + data augmentation)

['castle', 'view', 'cave'] a view of a castle with a castle in the background \ ['castle', 'courtyard', 'tower'] a castle like building with a clock tower in the background \ ['castle', 'door', 'city'] a castle with a clock on the side of it \ ['castle', 'park', 'hill'] a castle with a large clock on the side of it \ ['castle', 'battle', 'inside'] a castle with a large castle in the background \

```
In [8]: #####
# Inference outputs
#####
doodle_class = "castle"

# base_name, keyword_size = 'baseline', 3 # baseline2
# w2v_magnitdue_path = 'data/glove.42B.300d.magnitude'

base_name, keyword_size = 'augment', 3
w2v_magnitdue_path = 'data/glove.42B.300d.retrofit.magnitude'

# Sketch2Caption Inference
keys, sentences, failure = inference.sketch2caption(doodle_class=doodle_class,
                                                  checkpoint='pretrained/BEST_checkpoint_coco_{}_{}.pt'.format(base_name, keyword_size),
                                                  word_map_path='data/{}/WORDMAP_coco_{}_{}.json'.format(base_name, base_name, keyword_size),
                                                  w2v_magnitdue_path=w2v_magnitdue_path,
                                                  beam_size=3, num_sen=5)

for i in range(len(keys)):
    print(keys[i], ' '.join(sentences[i]))

['castle', 'cave', 'tower'] a castle like building with a clock on it
['castle', 'cave', 'close'] a close up of a castle with a castle in the background
['castle', 'wall', 'forest'] a castle with a clock on the side of it
['castle', 'part', 'front'] a castle with a castle in the background
['castle', 'inside', 'gate'] a large castle with a clock on the side of it
```

4.2.2 Quantitative Evaluation

Our task is not only generating good captions but also diverse outputs given single class name. Also, the output should provide meaningful captions related to the keywords including class name. Thus, we evaluate our results using the below metrics based on the goals.

- Quality: BLEU4
- Diversity: [Self-BLEU4 \(https://arxiv.org/pdf/1802.01886.pdf\)](https://arxiv.org/pdf/1802.01886.pdf)
- Semantic Accuracy: check if keywords are present in the output captions or not

4.2.2.1 Quality: BLEU4 score

We evaluate the quality of the generated captions based on the ground truths in Test Dataset. We average BLEU1-4 with the same weights using `NLTK corpus_bleu()` (https://www.nltk.org/_modules/nltk/translate/bleu_score.html) method.

- **Ours 1 showed the best quality of output captions for both keyword size**
- Ours 2 gave the lowest accuracy. We assume that data augmentation of keywords might disturb the model to learn how to generate captions based on the ground truth.

When Keyword size is 3, \ Baseline 1 : 0.35447955769733747 \ Baseline 2 : 0.329850170252854 \ **Ours 1 : 0.36034294511596937** \ Ours 2 : 0.3328501542643379

When Keyword size is 5, \ Baseline 1 : 0.3872948174606526 \ Baseline 2 : 0.3584256187411019 \ **Ours 1 : 0.40214758432644454** \ Ours 2 : 0.22522079844728965

```
In [2]: #####
# BLEU4 Evaluation
#####
# base_name, keyword_size = 'baseline', 3 # 5
# base_name, keyword_size = 'baseline2', 3 # 5
base_name, keyword_size = 'retrofit', 3 # 5
# base_name, keyword_size = 'augment', 3 # 5

# Evaluation on Test split
checkpoint = os.path.join('pretrained', 'BEST_checkpoint_coco_{}_{}.pt
h.tar'.format(base_name, keyword_size))
references, hypotheses, ref_inwords, hyp_inwords, key_inwords = evalu
ation.evaluate(base_name, keyword_size, checkpoint)
print("Test BLEU Score: ", corpus_bleu(references, hypotheses))
```

4.2.2.2 Diversity: Self-BLEU4 score

To evaluate how diverse the caption is, we use the diversity metric, called as [Self-BLEU4](https://arxiv.org/pdf/1802.01886.pdf) (<https://arxiv.org/pdf/1802.01886.pdf>). Since our model inference five captions, we implemented Self-BLEU4 by calculating BLEU4 scores between each caption and the corresponding four captions. We add all scores for all Doodle class names and normalize them. Since the Self-BLEU4 score represents diversity, **lower score is better**.

- **Ours 1 showed the best diversity of output captions for both keyword size** (lowest score)
- We expected that the Ours 2 would show better diversity but it gave the worst diversity compared to other methods. As we mentioned in the quality evaluation, we believe that data augmentation doesn't help to guide the network to learn.

When Keyword size is 3, \ Baseline 1 : 0.3989175388159808 \ Baseline 2 : 0.37840215178057673 \ **Ours 1 : 0.2909877677336257** \ Ours 2 : 0.44158844645520334

When Keyword size is 5, \ Baseline 1 : 0.2599512809602099 \ Baseline 2 : 0.4420841167665019 \ **Ours 1 : 0.2460393290983978** \ Ours 2 : 0.5398446699047991

```
In [12]: #####
# Self-BLEU4 Evaluation
#####
evaluation.selfbleu('baseline', 3)
evaluation.selfbleu('baseline2', 3)
evaluation.selfbleu('retrofit', 3)
evaluation.selfbleu('augment', 3)

evaluation.selfbleu('baseline', 5)
evaluation.selfbleu('baseline2', 5)
evaluation.selfbleu('retrofit', 5)
evaluation.selfbleu('augment', 5)
```

```
Self-Bleu 0.3989175388159808
Self-Bleu 0.37840215178057673
Self-Bleu 0.2909877677336257
Self-Bleu 0.44158844645520334
Self-Bleu 0.2599512809602099
Self-Bleu 0.4420841167665019
Self-Bleu 0.2460393290983978
Self-Bleu 0.5398446699047991
```

```
Out[12]: 0.5398446699047991
```

4.2.2.3 Semantic Accuracy

Our method propose the caption generation based on keywords, so it is important to evaluate if keywords are present in the corresponding output captions. We calculate semantic accuracy against each input keywords for all Doodle class names, and normalize them.

- **Ours 1 showed the best semantic accuracy of output captions for both keyword size**
- Baseline 1 showed similar accuracy with Ours 2 because it is trained on COCO caption dataset only without any Word2Vec embeddings

When Keyword size is 3, \ Baseline 1 : 0.6058997050147493 \ Baseline 2 : 0.56379821958457 \ **Ours 1 : 0.7125382262996944** \ Ours 2 : 0.6515337423312887

When Keyword size is 5, \ Baseline 1 : 0.7525525525525532 \ Baseline 2 : 0.5384146341463412 \ **Ours 1 : 0.7576687116564423** \ Ours 2 : 0.22585034013605473

```
In [13]: #####
# Self-BLEU4 Evaluation
#####
evaluation.semantic_acc_keys('baseline', 3)
evaluation.semantic_acc_keys('baseline2', 3)
evaluation.semantic_acc_keys('retrofit', 3)
evaluation.semantic_acc_keys('augment', 3)

evaluation.semantic_acc_keys('baseline', 5)
evaluation.semantic_acc_keys('baseline2', 5)
evaluation.semantic_acc_keys('retrofit', 5)
evaluation.semantic_acc_keys('augment', 5)

Semantic accuracy 0.6058997050147493
Semantic accuracy 0.56379821958457
Semantic accuracy 0.7125382262996944
Semantic accuracy 0.6515337423312887
Semantic accuracy 0.7525525525525532
Semantic accuracy 0.5384146341463412
Semantic accuracy 0.7576687116564423
Semantic accuracy 0.22585034013605473
```

```
Out[13]: 0.22585034013605473
```

4.3 Caption2Scene Generator

4.3.1 Inception score

Inception score was first introduced by [Salimans et al \(https://arxiv.org/pdf/1606.03498.pdf\)](https://arxiv.org/pdf/1606.03498.pdf) to evaluate the images generated by a generative model. In this method, we apply the [inception model \(https://arxiv.org/abs/1409.4842\)](https://arxiv.org/abs/1409.4842) to every generated image to get the conditional label distribution $p(y|x)$, where y is the target data and x is generated data. Images that contain meaningful objects should have a conditional label distribution $p(y|x)$ with low entropy. Moreover, the model is also expected to generate varied images, so the marginal $\int p(y|x = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(E_x KL(p(y|x)||p(y)))$, where the exponentiated results makes the values comparable. This inception score is directly provided by pytorch library and we used this score to evaluate the quality of our generated scenes.

```
In [1]: import sys
import os
sys.path.append("./Text2Scene")
print(os.getcwd())
os.chdir("./Text2Scene")
from evaluate_images import *
```

/scratch/nlpclass-1197-g-thefinalproject/project


```
In [2]: print("Inception Score evaluation for image quality")
print("=====")

print("Evaluating Images generated using caption generated by the baseline method")
print("=====")
inc_score = evaluate_inception(images_path = "./logs/baseline/composites_samples")
print("Inception Score for Baseline images ",inc_score)

print("Evaluating Images generated using caption generated by our method")
print("=====")
inc_score = evaluate_inception(images_path = "./logs/retrofit/composites_samples")
print("Inception Score for Retrofitted images ",inc_score)
```

```
Inception Score evaluation for image quality
=====
Evaluating Images generated using caption generated by the baseline method
=====
=====

/scratch/nlpclass-1197-g-thefinalproject/project/Text2Scene/evaluate_images.py:181: UserWarning: Implicit dimension choice for softmax has been deprecated. Change the call to include dim=X as an argument.
  return F.softmax(x).data.cpu().numpy()

Inception Score for Baseline images  47.763093829189415
=====
Evaluating Images generated using caption generated by our method
=====
Inception Score for Retrofitted images  49.08888457356971
```

Inception Score evaluation for image quality

Evaluating Images generated using caption generated by the **baseline method** Inception Score for Baseline images 47.763093829189415

Evaluating Images generated using caption generated by **our method** Inception Score for Retrofitted images 49.08888457356971

4.3.2 Semantic Object Accuracy

"Generative networks conditioned on simple textual image descriptions are capable of generating realistic-looking images. However, current methods still struggle to generate images based on complex image captions from a heterogeneous domain. Furthermore, quantitatively evaluating these text-to-image synthesis models is still challenging, as most evaluation metrics only judge image quality but not the conformity between the image and its caption. To address the aforementioned challenges, [Semantic Object Accuracy for Generative Text-to-Image Synthesis \(https://arxiv.org/pdf/1910.13321.pdf\)](https://arxiv.org/pdf/1910.13321.pdf) introduces both a new model that explicitly models individual objects within an image and a new evaluation metric called Semantic Object Accuracy (SOA) that specifically evaluates images given an image caption."

We used this Semantic Object Accuracy (SOA) to evaluate the accuracy of our generated images. We used a pretrained YOLOv3 network to infer the objects in the scene. Since we don't have the ground truth object for our generated captions, we used NLTK.pos_tag to get noun words in our captions. We noticed that some objects in the captions are not a defined class for detection, so we used wordnet corpus to find the hypernyms of every object in the class to get all parents of the objects in the captions. Then we intersected these hypernyms and objects with predefined detection classes to generate ground truth for our captions. To calculate accuracy, we intersected these generated ground truth and yolo3 predictions sets to get count of caption2scene detections and divided it by total caption groundtruth to all generated images and averaged it.

```
In [2]: print("Semantic Object Accuracy (SOA) evaluation for caption to image accuracy")
print("Evaluating Images generated using captions generated by the baseline method")

pred_dict = yolo_predictions(images = "./logs/baseline/composites_samples")
soa_accuracy = cal_soa_accuracy(images_path="./logs/baseline/composites_samples",caption_path='./examples/composites_samples_baseline.json',yolo_pred_dict=pred_dict)
print("SOA Score for Baseline images ",soa_accuracy)

print("Evaluating Images generated using captions generated by the retrofitted method")
pred_dict_retro = yolo_predictions(images="./logs/retrofit/composites_samples")
soa_accuracy = cal_soa_accuracy(images_path="./logs/retrofit/composites_samples",
                                caption_path='./examples/composites_samples_retorfit.json', yolo_pred_dict=pred_dict_retro)
print("SOA Score for Baseline images ", soa_accuracy)
```

Semantic Object Accuracy (SOA) evaluation for caption to image accuracy

Evaluating Images generated using captions generated by the baseline method

Loading network.....

Network successfully loaded

Using GPU: 0

100%|██████████| 1405/1405 [00:19<00:00, 72.18it/s]

SOA Score for Baseline images 0.48320413436692505

Evaluating Images generated using captions generated by the retrofitted method

Loading network.....

Network successfully loaded

Using GPU: 0

100%|██████████| 1430/1430 [00:19<00:00, 74.16it/s]

SOA Score for Baseline images 0.22536287242169595

Semantic Object Accuracy (SOA) evaluation for caption to image accuracy

Evaluating Images generated using captions generated by the baseline method

Loading network.....

Network successfully loaded

Using GPU: 0

SOA Score for Baseline images 0.48320413436692505

Evaluating Images generated using captions generated by the retrofitted method

Loading network.....

Network successfully loaded

Using GPU: 0

SOA Score for Baseline images 0.22536287242169595

5 .Code

5.1 Sketch Classifier:

we chose one of the [Pytorch Implementations \(https://github.com/adam9500370/Kaggle-QuickDraw\)](https://github.com/adam9500370/Kaggle-QuickDraw) from this challenge to work with. We didn't use the pre-trained model from this implementation. Instead, we made changes to the data loader and the train code to fit it for fine-tuning a Pytorch pre-trained model on the doodle dataset:

- changed the input size to 256*256
- changed the last layer of the model
- changed the normalization
- fixed bugs in the original code

We first tried to fine-tune the pre-trained [DenseNet \(https://arxiv.org/abs/1608.06993\)](https://arxiv.org/abs/1608.06993) model, but training took more time than we expected, so we trained [MobileNetV2 \(https://arxiv.org/abs/1801.04381\)](https://arxiv.org/abs/1801.04381). We chose MobileNet because it is faster than DenseNet and it also maintain the accuracy.

5.2 Caption Generator:

- For the generative sequence model we used [pytorch image captioning tutorial](https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning) (<https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning>) as reference.
- For retrofitting , we used the code that we wrote from HW2.
- All other code to preprocess , inference and evaluation are written by us .

5.3 Caption2Scene Generator:

- The code and pretrained model are downloaded from the [text2scene](https://github.com/uvavision/Text2Scene) ([git@github.com:uvavision/Text2Scene.git](https://github.com/uvavision/Text2Scene)) official pytorch repository and made some minor modifications to work with it.
- The evaluation code for inception score and Semantic object accuracy were written by us.

6. Results

Some generated captions from Baseline 1, Baseline 2, retrofitting and retrofitting + data augmentation

Baseline 1 (nouns of COCO)

['castle', 'cathedral', 'ruins'] a castle with a castle in the background \ ['castle', 'mansion', 'palace'] a castle with a castle in the background \ ['castle', 'medieval', 'hill'] a castle with a castle in the background \ ['castle', 'hill', 'ruins'] a castle with a mountain in the background \ ['castle', 'hill', 'ruins'] a castle with a mountain in the background

Baseline 2 (GloVe embedding)

['castle', 'medieval', 'mansion'] a castle with a castle in the background \ ['castle', 'ruins', 'mansion'] a castle with a castle in the background \ ['castle', 'ruins', 'palace'] a large castle like building with a clock on it \ ['castle', 'palace', 'cathedral'] a large castle like structure with a castle in the background \ ['castle', 'cathedral', 'tower'] a castle with a castle in the background

Ours 1 (retrofitting with COCO)

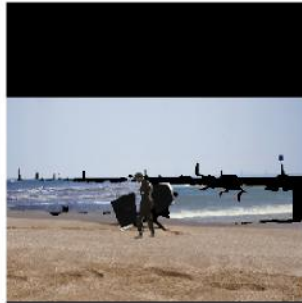
['castle', 'gate', 'forest'] a large castle with a sign on it \ ['castle', 'park', 'town'] a group of people in a town square \ ['castle', 'park', 'street'] a street sign in front of a building \ ['castle', 'forest', 'clock'] a large castle with a clock on it \ ['castle', 'inside', 'park'] a group of people inside of a building

Ours 2 (retro + data augmentation)

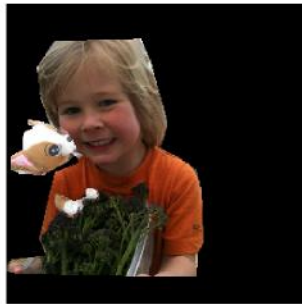
['castle', 'view', 'cave'] a view of a castle with a castle in the background \ ['castle', 'courtyard', 'tower'] a castle like building with a clock tower in the background \ ['castle', 'door', 'city'] a castle with a clock on the side of it \ ['castle', 'park', 'hill'] a castle with a large clock on the side of it \ ['castle', 'battle', 'inside'] a castle with a large castle in the background \

Final good looking images from captions generated by Retrofitting(2.2.3):

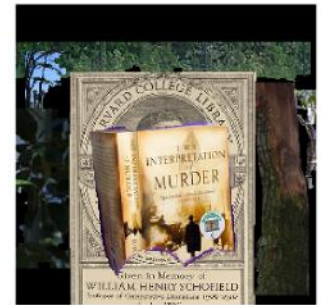
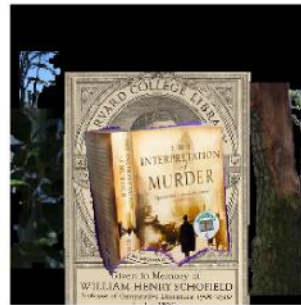
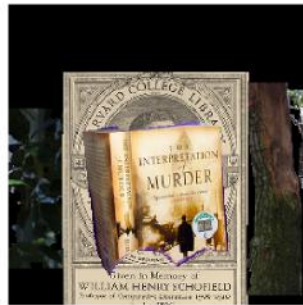
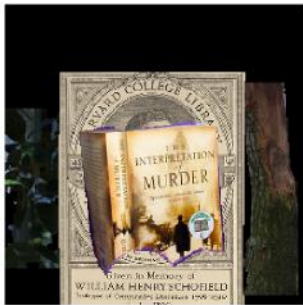
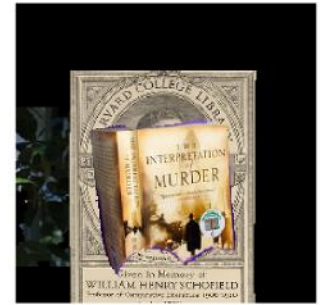
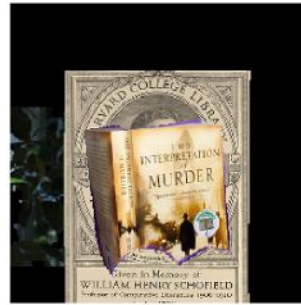
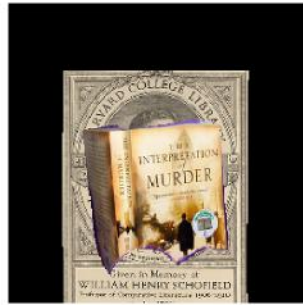
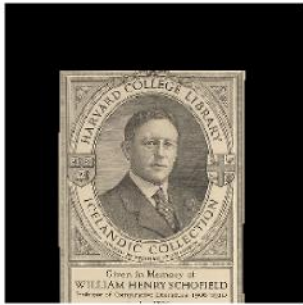
a person holding a surf board on the beach.



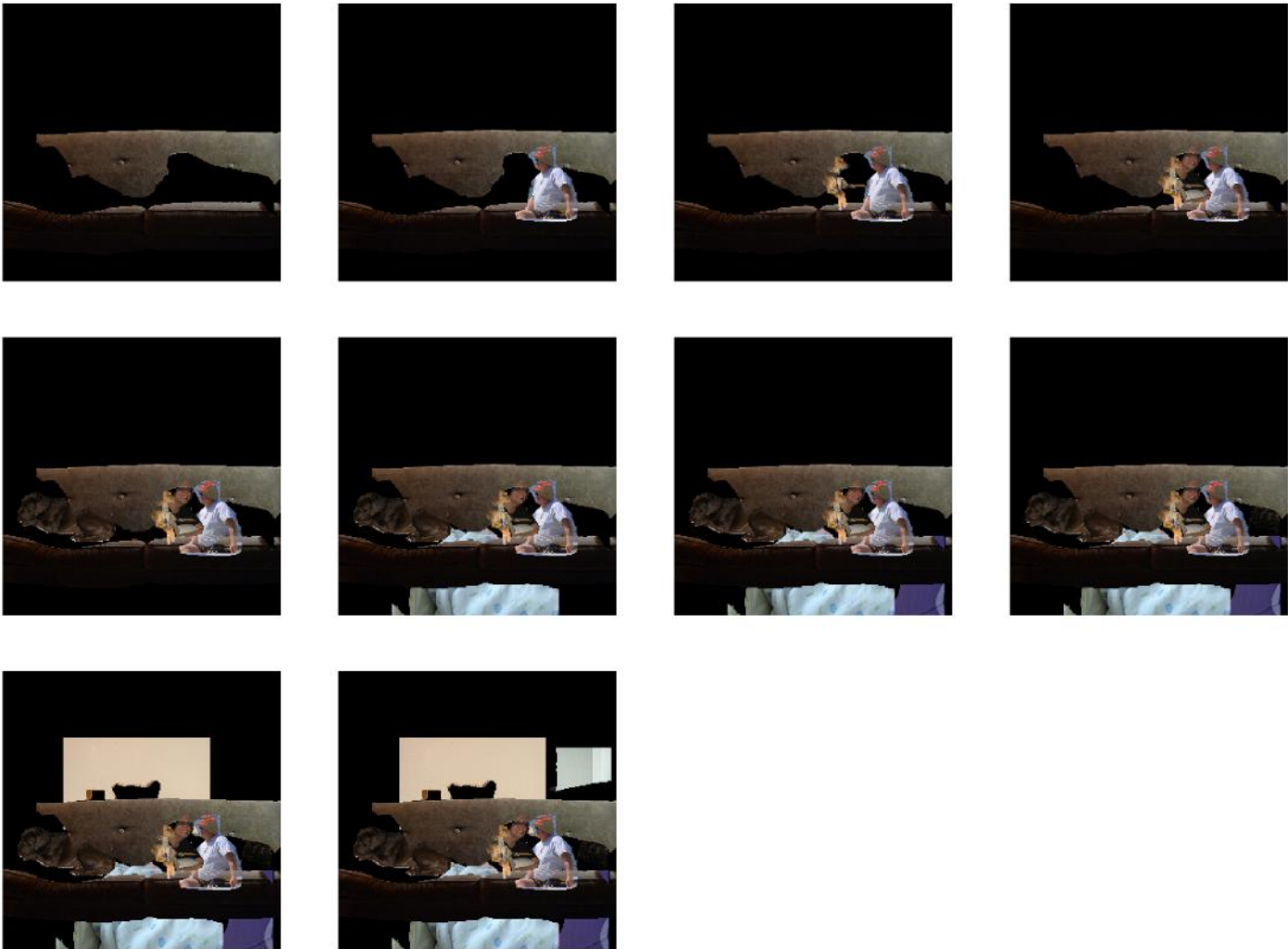
a young boy is holding a stuffed animal.



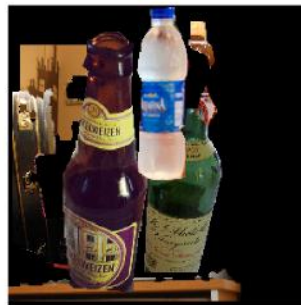
a book that is laying on the ground.



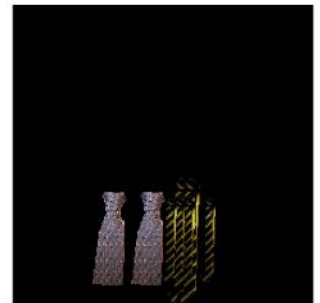
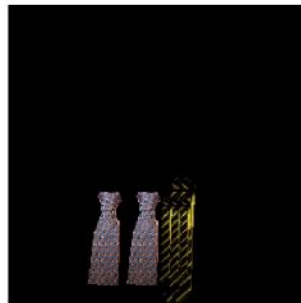
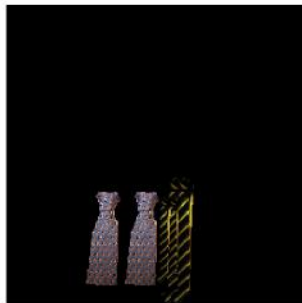
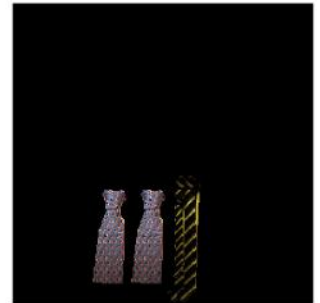
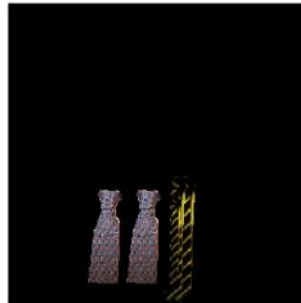
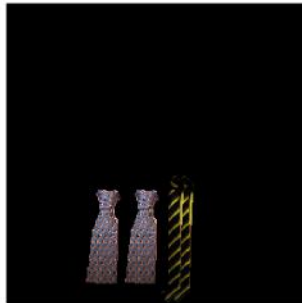
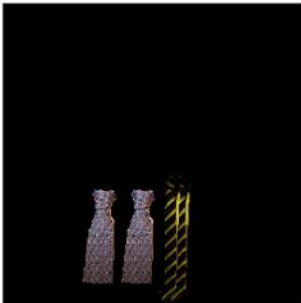
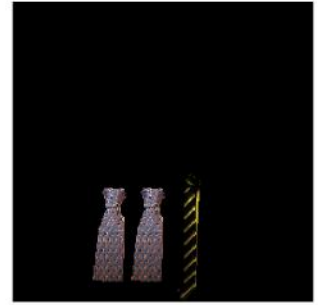
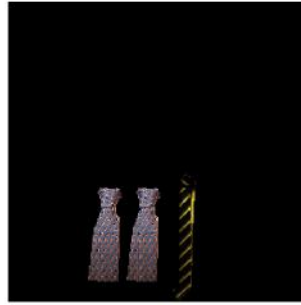
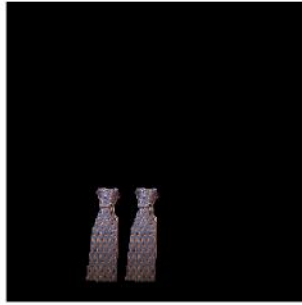
a person sitting on a couch with a small dog.



a bottle of wine sitting next to a bottle of wine.



a couple of ties that are sitting next to each other.



a close up of the side of a parking meter.



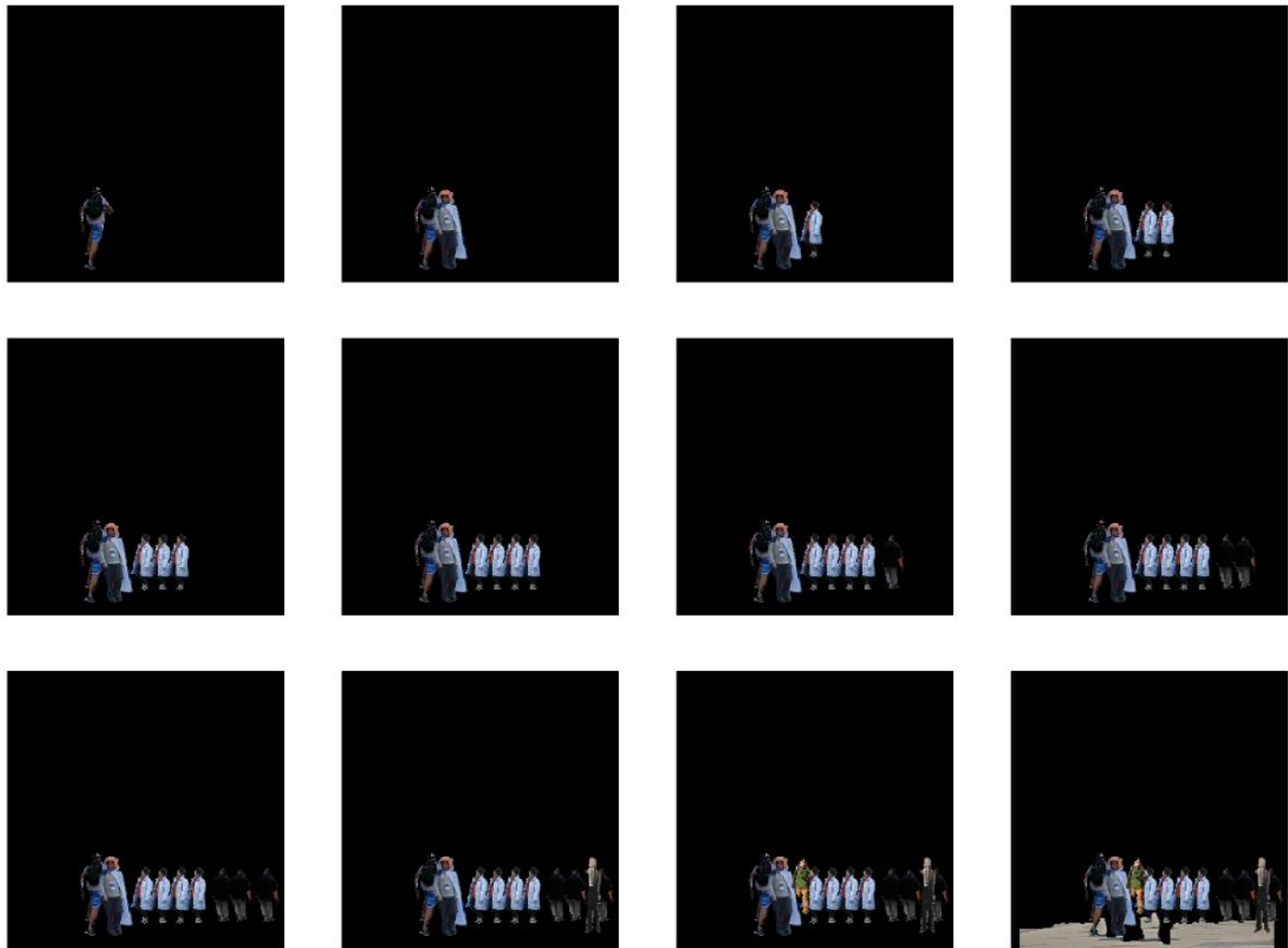
a bus that is parked next to a car.



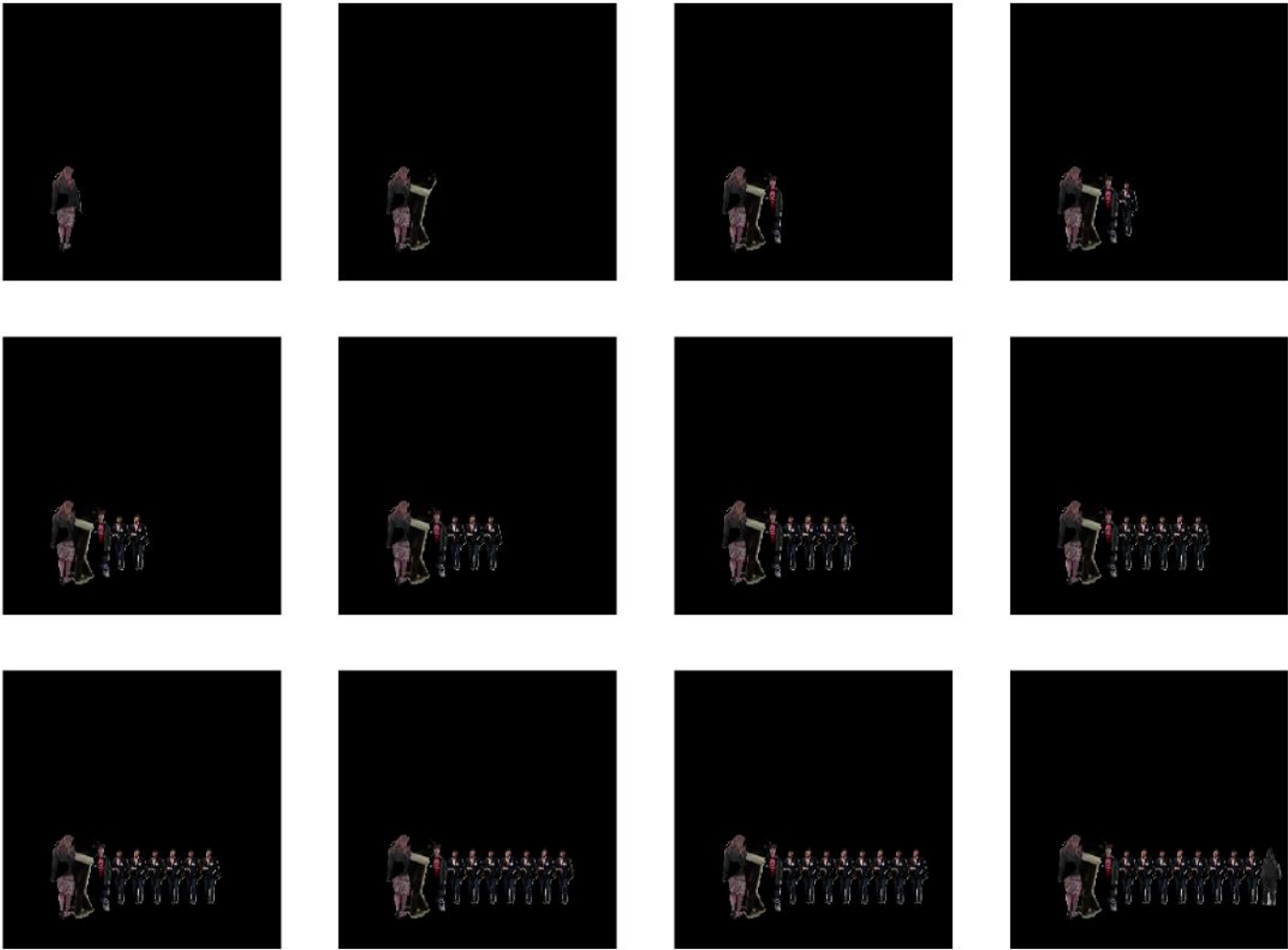
a public transit bus on a city street.



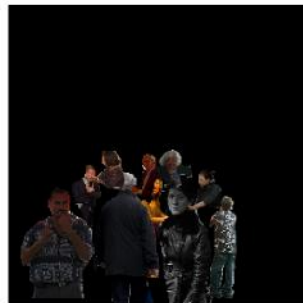
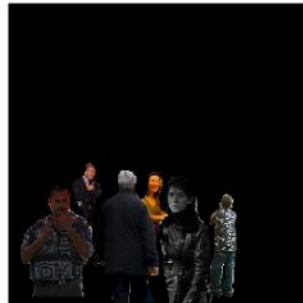
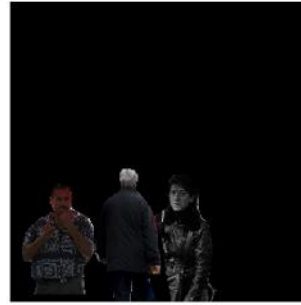
Failure cases from captions generated by Retrofitting(2.2.3):
a person is standing in front of the eiffel tower.



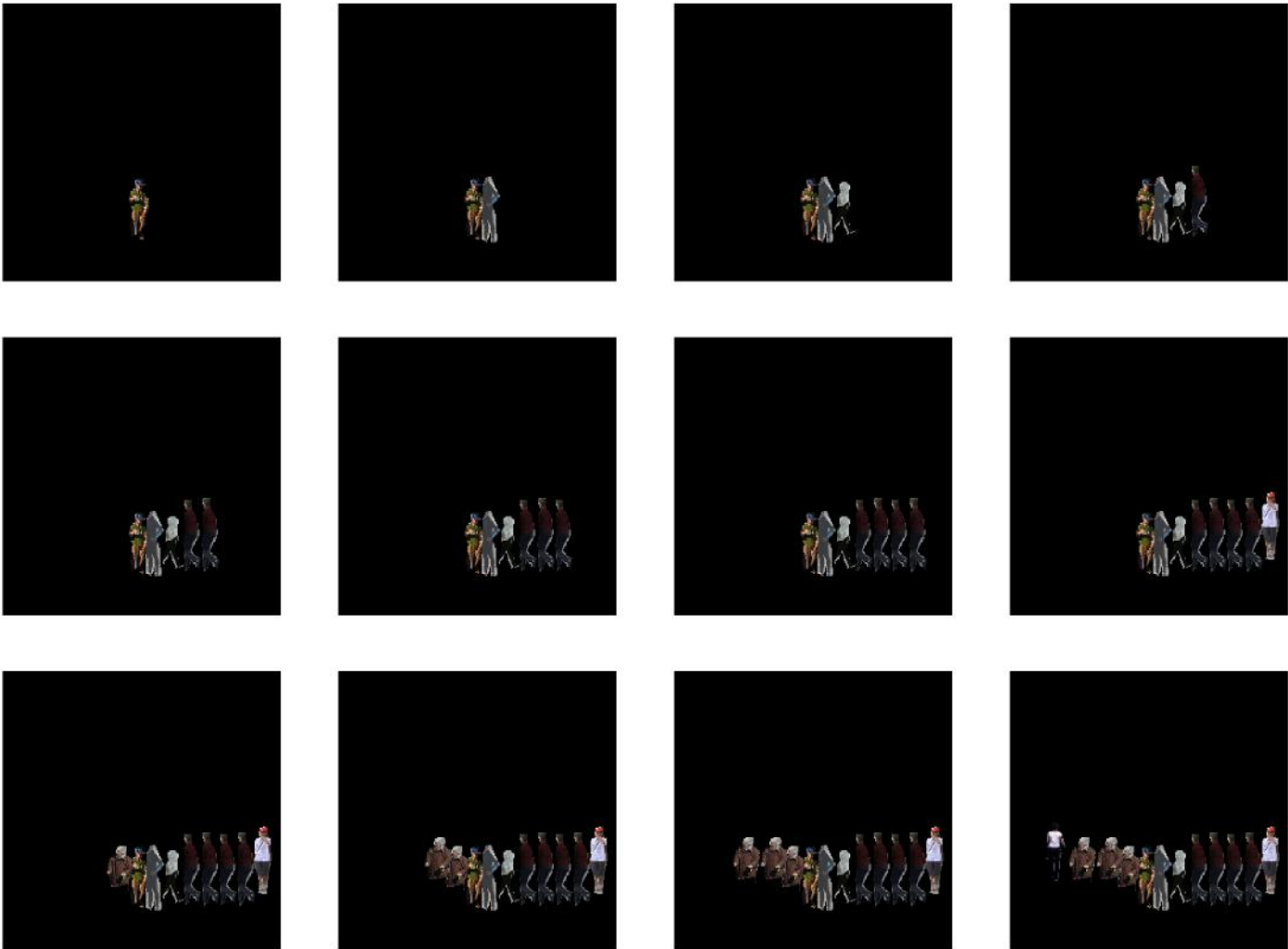
a group of people standing on the side of the road with a clock tower in the background.



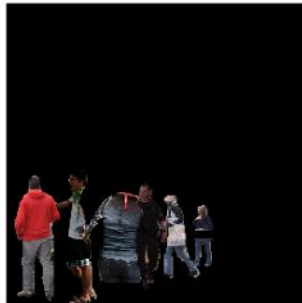
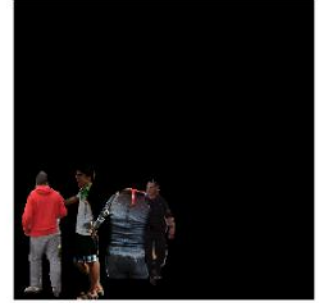
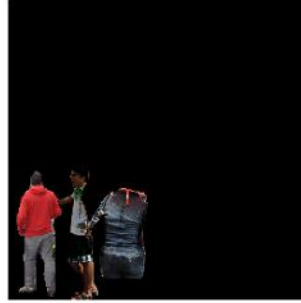
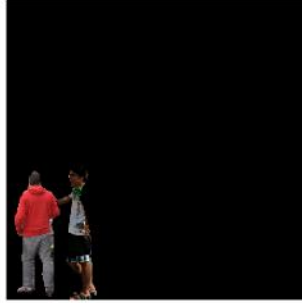
a group of people standing in front of a hotel.



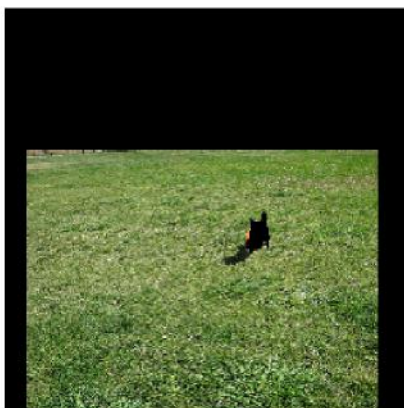
a group of people standing on a pier.



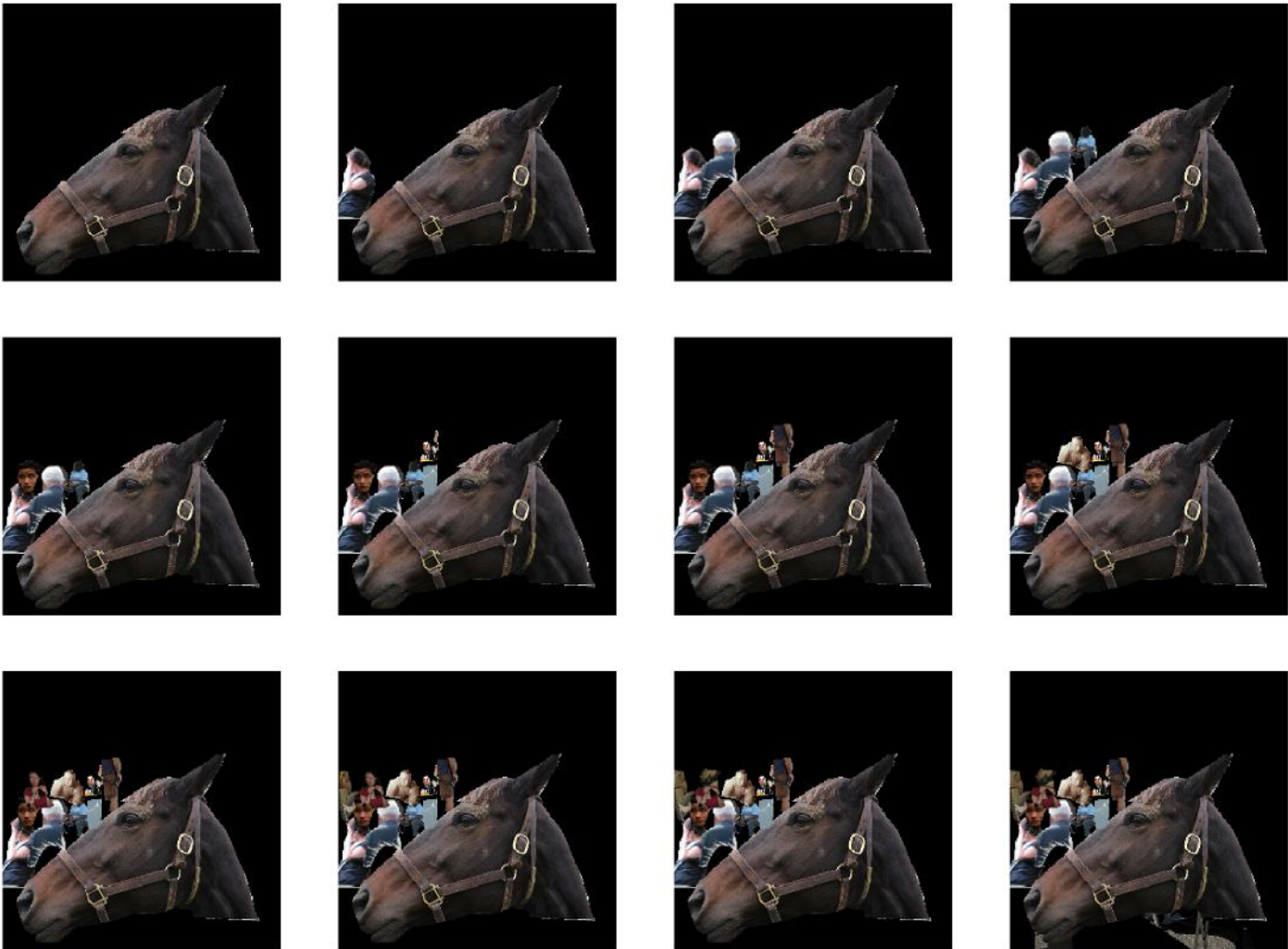
a group of people are standing next to a bus.



a black and white photo of a cactus on a field.

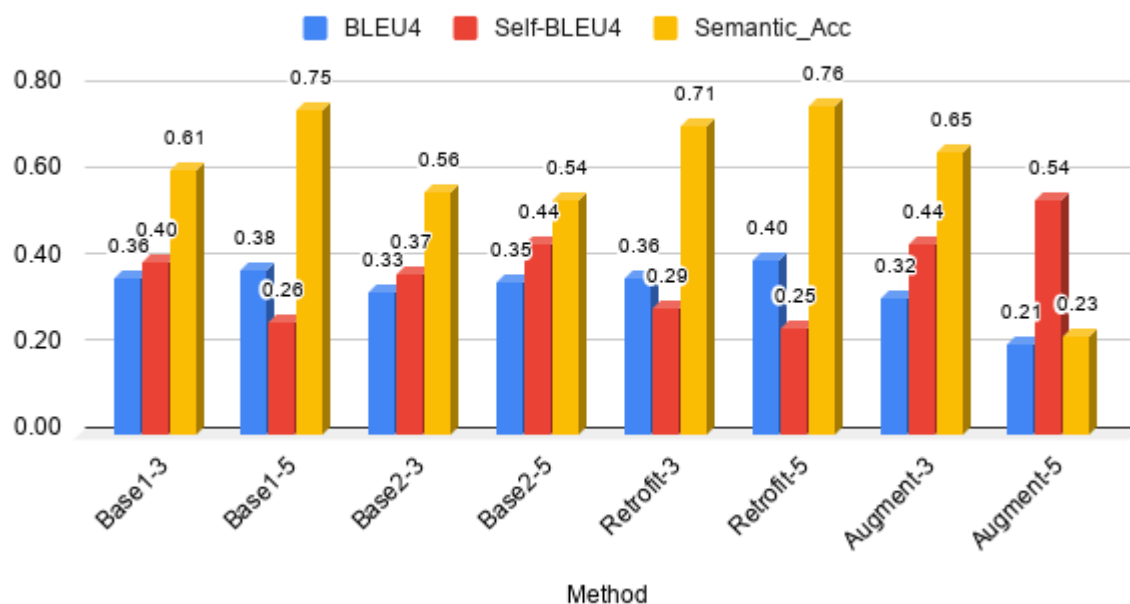


a camel with a camel on it and a camel on it.



6. Analysis of the Results (Conclusion)

Quantitative Evaluation



Our method (Retrofit-3 and 5) produces high quality (**High BLEU4 score**) and more diverse (**Lower Self-BLEU4 score**) captions given sketch input. Our method generates captions which have better correspondence with input sketch (**Higher Semantic Accuracy**)

As explained in the experimental setup section , baseline -1 is trained completely using COCO caption and so it is not fair to compare baseline-1 with other methods, but the results are displayed for your information.

We tried to improve diversity by randomly selecting similar words to input words as a data augmentation during training . Although it showed better results for training data , it couldn't generalise well for test data as you can see from the above results. We suspect that since we replace the actual word with similar word , the link between input and output are broken , which led to reduced score . For future work , we want to try replacing the word in both input and output to get better results .

Caption2Scene Generator		
Method	Inception Score	Semantic Object Accuracy
Base3	47.76	48.3 %
Ours3	49.10	22.5 %

Our method generates better captions resulting in better scenes (**higher Inception score**). Caption2Scene model generates the scenes with more diverse objects and relationships, which are unseen by YOLOv3 detector (**Lower SOA score**)

6.1 Limitations

- As you can see from the failure cases from the output images ,whenever "**a group of people**" is present in the caption , the text2scene focuses only on them and adds more and more people and discards other objects and relationships in the caption .
- We can also see from the SOA score of scenes generated using retrofitted model , the captions generated are too complex and diverse for a retrieval network to handle . Maybe using a GAN model to generate images may help in these cases but it may drop the quality of generated images.
- Our pipeline depends on the class relationships between doodle classes(340) and coco classes (91) which add more unseen objects in the captions. As the Text2Scene model is trained using COCO dataset , it cannot identify these missing classes like Eiffel tower , Monalisa , Camel etc.

7. Future Work

- We can extend this model to produce additional data to improve object detection and image captioning task
- The end-to-end architecture will help to produce more corresponding scenes with input sketch with respect to pose and style of objects
- Conditional GAN can replace our conditional sequence generator to improve the diversity of captions
- input sketch classifier can be trained using RNN to learn the sketching style which is available in the google quick draw dataset and improve the classifier accuracy.