Sentiment Based Image Captioning

Apra Gupta and David Zheng
Northeastern University
TA: Omkar Reddy Gojala
(Dated: April 27, 2021)

For our final project, we wanted to tackle image captioning. Image captioning combines two of the most common applications of machine learning: image and text processing. To add our own twist to this well-studied problem, we wanted to see if we could include sentiment in our generated captions. Since image captioning is a difficult problem, we deiced to break it down into two sub-problems.

1. *Sub-problem 1 (Classification)*: Classify the sentiment of an image.
2. *Sub-problem 2 (Generation)*: Generate a caption for an image with a given sentiment.

Here is all of the code and slides.

I. MOTIVATIONS

A. Sub-problem 1: Sentiment Based Image Classification

The motivation behind the first sub-problem is to apply the machine learning algorithms learned in class to a new area. The hope is to achieve a deeper understanding of these algorithms. Furthermore, since it is a classification problem, we will have known metrics to evaluate our model. This allows us to experiment with new topics like image feature extraction and sentiment analysis while working within frameworks we are familiar with.

B. Sub-problem 2: Sentiment Based Image Captioning

While the motivation behind the first sub-problem was to apply skills we already know, the motivation behind the second sub-problem is to learn new skills. Image captioning relies on many topics not covered in this class. Some of these topics are RNNs, GRUs, LSTM, and attention models. We wanted to learn how these neural networks work, and implement our own model architectures for this problem without relying on a tutorial.

II. DATA-SETS CREATION AND EXPLORATORY ANALYSIS

A. Google Conceptual Captions

1. Description

Google AI's Conceptual Caption dataset is the primary dataset we used for our project [1]. It contains +3 million image caption pairs. These image caption pairs are 1 to 1 and are scraped directly from the internet. This is different from MSCOCO, which has 5 human-curated captions per image. Furthermore, around 2 million of the images contain object labels with confidence thresholds. The dataset is structured as an image link with a caption. It also included other information like object confidence thresholds (for objects detected in the image using Google's API), and other miscellaneous details we discarded. We chose this dataset mainly because of its size: not other dataset comes close to having as many image-caption pairs as this one. We also chose it because the fact that the captions were somewhat subjective and not strictly objective descriptions of what was happening in the image: since they were scraped from various websites, the captions were very varied in terms of tone and purpose. FIG. 1. shows some sample image-caption pairs from our dataset.

2. Data processing pipeline

As mentioned above, the dataset contains links to images. So, we had to write a pipeline to download these images. Sadly, unlike MSCOCO, there is no way to batch download all the images (though if it existed, it would be ridiculously large). So, we wrote a pipeline to read from the dataset and download the images at the links. To start, the pipeline had to contend with multiple potential request errors, ranging from timeouts to 404s. We used the 'requests' python library for this. We also wrote the pipeline such that it can be stopped and started at any point and would resume where it left off. This allows us to get more unique data whenever we see fit. Finally, even after the images were saved, some were invalid in some way (could not be opened or was 0 by 0). Using another python library called PIL to verify an image was valid. The pipeline can process about 20k images in a day. The major bottleneck to speed was the HTML requests. We choose to download 30k images for our project.

This Jupyter notebook outlines our entire data acquisition and cleaning pipeline.

FIG. 1. Sample Image Caption Pairs

3. *Analysis*

We ran some data analysis on the Google Conceptual Captions to ensure there were no outliers or strange distributions. FIG. 2. shows that the captions had an exponential distribution of around 10 captions. FIG. 3. shows the number of objects in each image with high confidence had also an exponential distribution around 2 objects. So, both our images and captions are were very similar to each other in size and the distribution was concentrated to certain sizes for each, we did not have to worry about outliers too much. We also wanted to make sure that the image sizes were mostly the same, as they would need to be compressed for feature extraction. FIG. 4. and FIG. 5. shows that they were mostly the same. The images themselves were extremely diverse, however, with everything ranging from profile shots to clip art.

This Jupyter notebook outlines all the data exploration we conducted..

B. Google Conceptual Captions with Sentiment

1. *Description Creation*

Since we wanted to deal with sentiment in captions, we performed some feature extraction on the Google Conceptual Caption dataset. We wanted to use a pre-trained sentiment classifier to extract the sentiment of each caption. We tried two pre-trained models, one from nltk and another from hugging-face. We decided to hugging-face for the reason explained below. We hoped that the sentiment of the caption is a descriptor of the sentiment of the image itself.

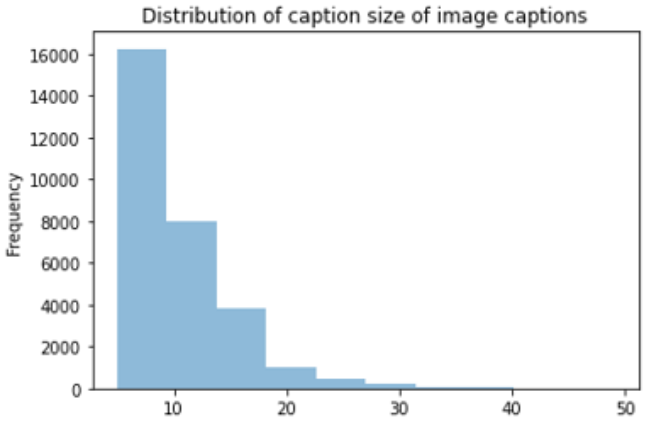

FIG. 2. Caption Size

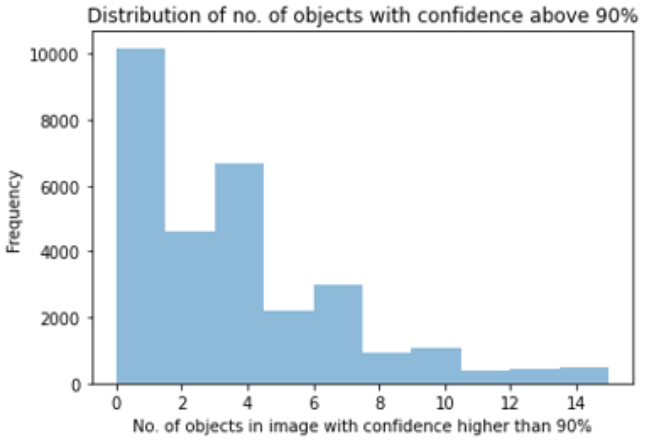

FIG. 3. Number of Objects with High Confidence

2. *Analysis*

Over the 30k data points we had, FIG. 6. describes the distribution of positive and negative captions. As we can see, about 70% of the captions were classified as positive and 30% negative. As you can see from FIG. 7., hugging-face, unlike nltk, had very high confidence in its sentiment, which led us to choose it over nltk. nltk classified most the captions as having neutral sentiment. Further, when we forced it to choose between positive and negative, over 90% of the captions were classified positive and the classification labels carried a much lower confidence.

For reference, FIG. 8 shows a image-caption pair whose caption is classified as positive by huggingface, while FIG.

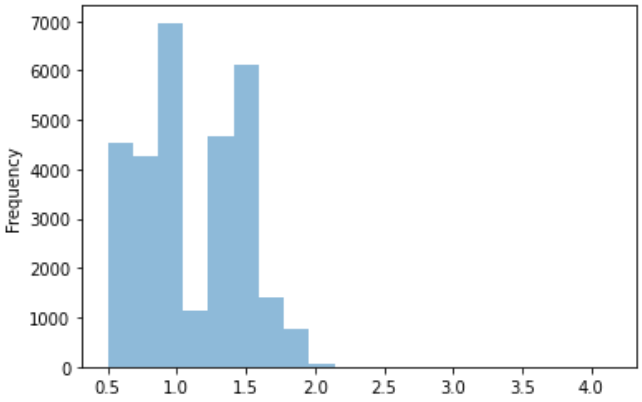

FIG. 4. Aspect Ratio Distribution

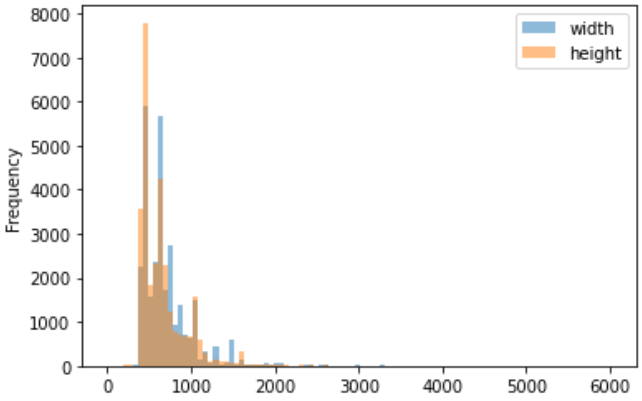

FIG. 5. Width and Height Distribution

9 shows a negatively classified caption. The labels look accurate.

III. FEATURE EXTRACTION

For feature extraction, we tried many different pretrained image feature extractors. We started with InceptionV3 which had an output size of 8x8x2048. This was too large for for our prediction models, so we used VGG16 for those, which has an output size of 1x4096. We evaluated our generation model on InceptionV3 and finally, as per the recommendation from the Google Conceptual Caption paper, on InceptionResnetv2, which had an output of size 8x8x1536. We used pretrained GloVe word embeddings to embed words for our generative model.

This Jupyter notebook outlines the feature extraction pipeline.

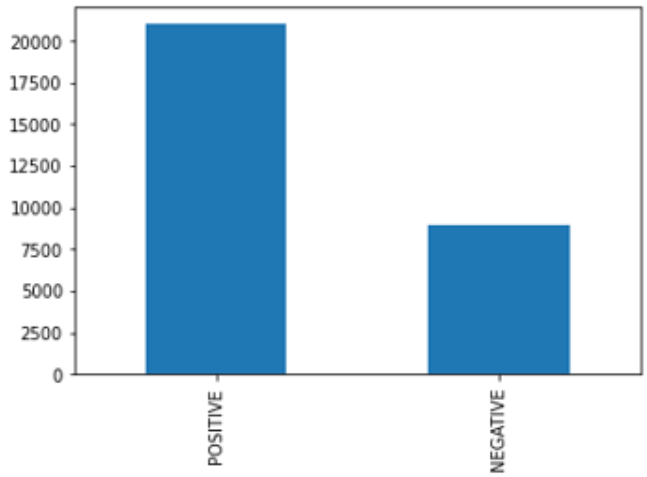

FIG. 6. Sentiment Distribution

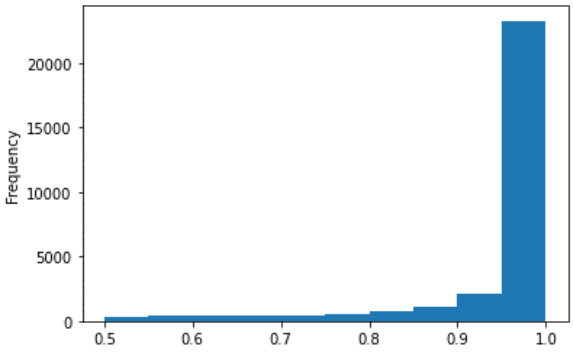

FIG. 7. Sentiment Confidence Distribution

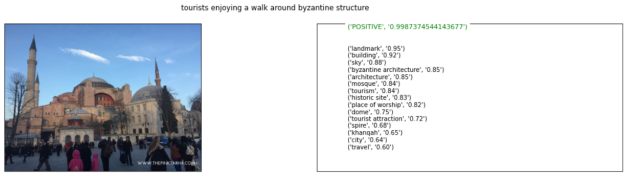

FIG. 8. Positive Labelled Caption: "tourists enjoying a walk around byzantine structure"

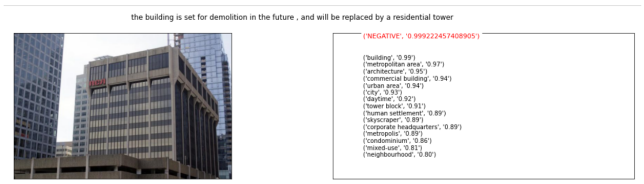

FIG. 9. Negative Labelled Caption: "the building is set for demolition in the future, and will be replaced by a residential tower"

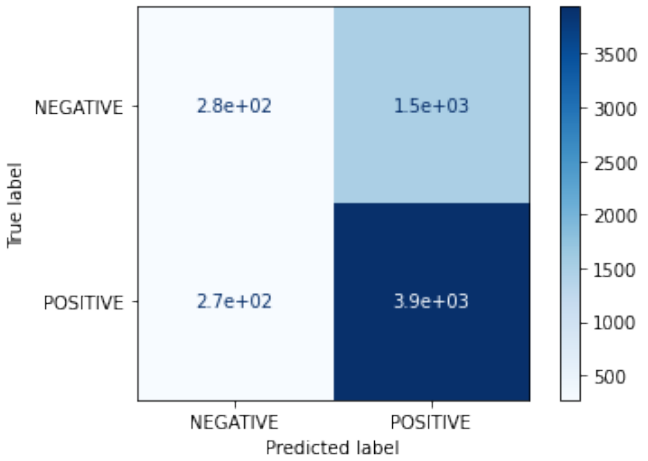

FIG. 10. Unbalanced Logistic Regression Confusion Matrix

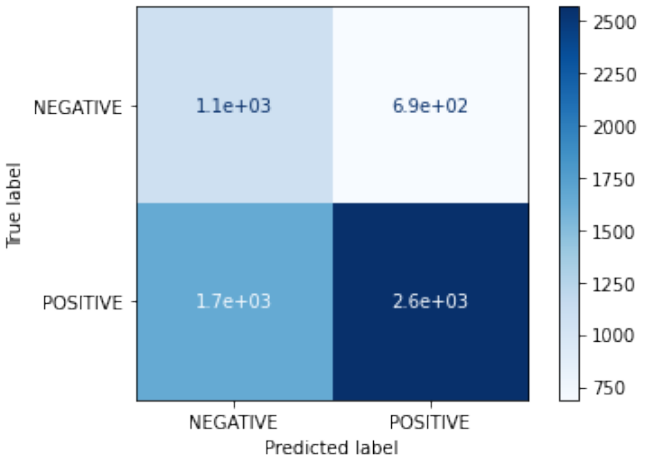

FIG. 11. Balanced Logistic Regression Confusion Matrix

IV. PREDICTIVE MODEL

The first suite of models we trained were focusing on sub-problem 1. We hoped to find a model that could predict the sentiment based on the image file alone. All confusion matrix displayed are for the testing set.

A. Feature Extraction and Selection

1. *Feature Extraction*

For feature extraction, we choose to use VGG-16. We tried other feature extraction methods as well. We started by attempting to use InceptionV3. However, the output from InceptionV3 was too large to form into one batch dataset, even when only considering 10k images. So, then we tried a pre-trained image extractor known for its small output, Mobile Net. The results were unpromising, so we choose to use an extraction model with more features like VGG-16. However, mentioned later, our results did not improve by much.

This Jupyter notebook contains the feature extraction pipeline for the predictive model.

2. *Feature Selection and Baseline models*

We choose to use logistic regression as our baseline classifier, seen in FIG. 10. It achieved an error of .3 and F1 of 0.83 on the testing set. To attempt to address the in balance of classes, we balanced the classes whenever possible. FIG. 11. shows a balanced logistic regression. It achieved an error of .4 and F1 of 0.68 on the testing set. This shall be our baseline classifier, as unbalanced logistic regression overwhelming predicts 'POSITIVE'.

Since VGG-16's output contains 4096 features, we started by applying some feature selection to stop potential future over-fitting. Feature selection was done and run on balanced logistic regression. However, as FIG. 12-16 show that feature selection did not improve the performance of the model on the testing set compared to the baseline, seen in FIG. 8. Furthermore, it did not reduce over-fitting.

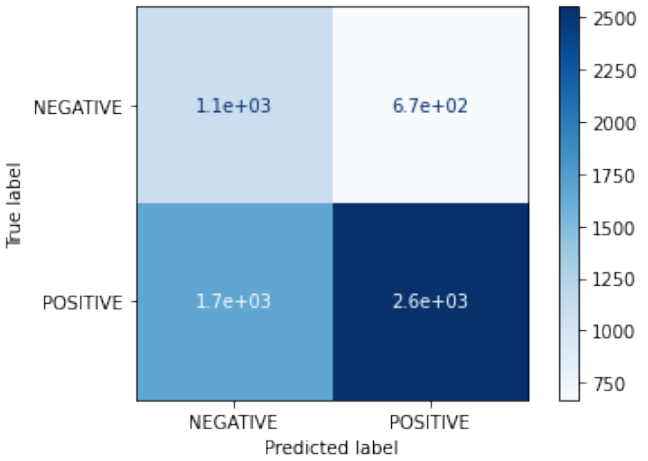

FIG. 12. ANOVA F Score Selection Confusion Matrix

B. Models Used

This Jupyter notebook contains all the predictive models.

1. *Decision Tree Classifier*

Our data appears to not be linearly separable. So, we decided to train some non-linear models. The first we attempted was a non-pruned balanced decision tree

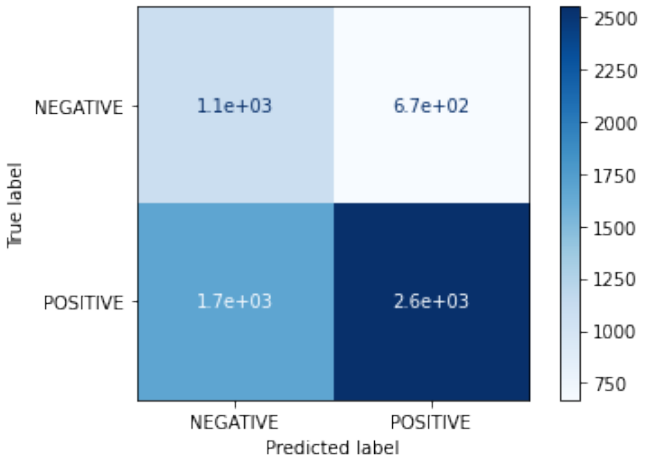

FIG. 13. Variance Threshold Selection Confusion Matrix

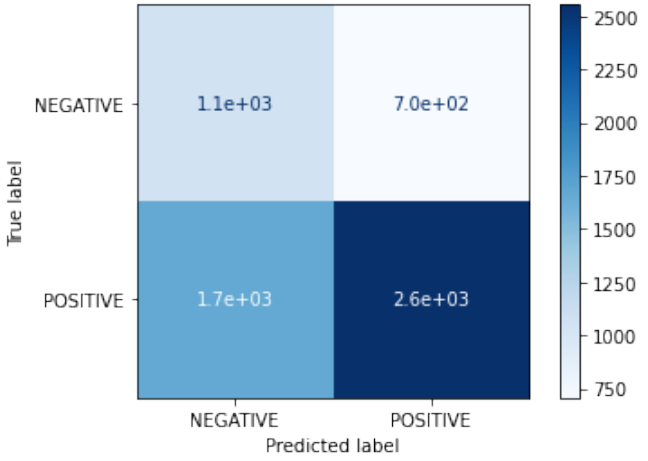

FIG. 14. Logistic Regression Wrapper Selection Confusion Matrix

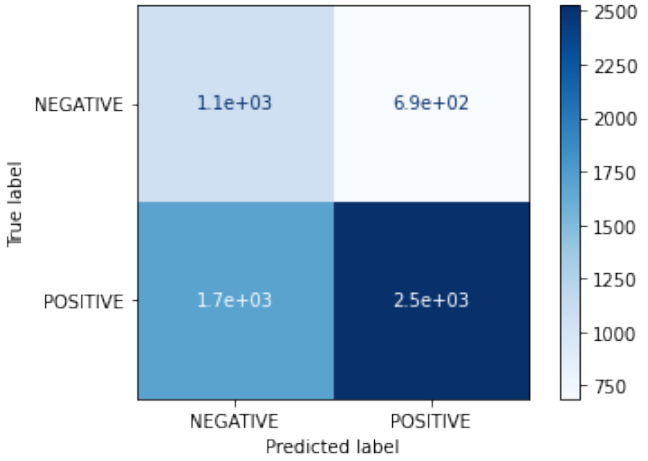

FIG. 15. Decision Tree Wrapper Selection Confusion Matrix

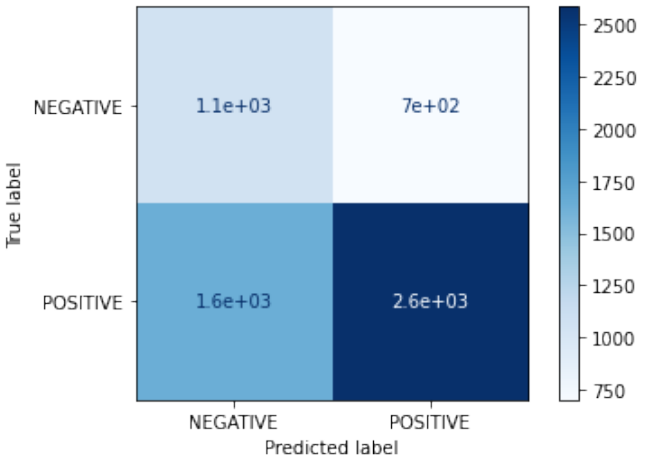

FIG. 16. Support Vector Wrapper Selection Confusion Matrix

classifier with balanced class weights.

2. *Random Forest Classifier*

Another ensemble method tried was a random forest classifier. Again, 100 estimators were used with balanced class weights.

3. *Adaboost*

As demonstrated above, a simple model like logistic regression fits poorly to the data. So, we decided to test Adaboost, for its ability to reduce bias. It was trained using 100 base estimators. The base estimator was a decision stump with balanced class weights.

4. *Multi-level perceptron*

Finally, we trained a multi-level perceptron with 2 hidden layers. The first layer had 1000 hidden nodes, and the second had 500 hidden nodes.

C. Results

1. *Decision Tree Classifier*

Our decision tree classifier achieved results very similar to the baseline, with an error of .39 and an F1 of .72. Since we used no pruning, there was extreme over-fitting. Its training error was .0004. Even though its classes were balanced, the resulting model still showed preference towards predicting 'POSITIVE'.

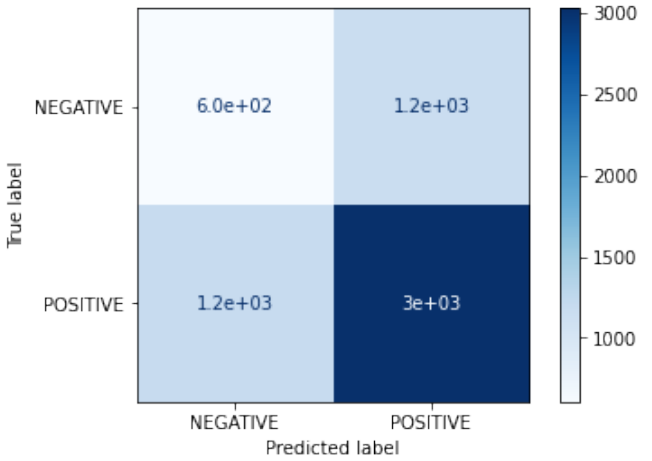

FIG. 17. Decision Tree Confusion Matrix

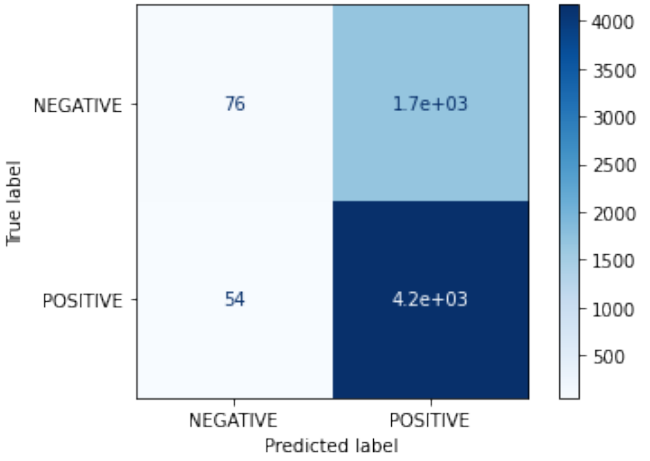

FIG. 18. Random Forest Confusion Matrix

2. *Random Forest Classifier*

Since our decision tree performed less than stellar, there was hope that ensembles could come to the rescue. However, even though our random forest classifier had a higher accuracy at .30 and an F1 at .82, it was not a good model. The confusion matrix clearly reveals how the model simply learned that there were more 'POSITIVE's than 'NEGATIVE's and thus defaulted to predicting 'POSITIVE'. It is interesting that it learned this bias in the data, despite using a balanced class weight. Furthermore, it heavily over-fits the data.

3. *Adaboost*

Adaboost is known for its ability to reduce both bias and variance. Given the poor performance of the other models, there was hope that Adaboost could produce a good model. However, it had the worst accuracy of all trained models. At least, it did not overfit the data.

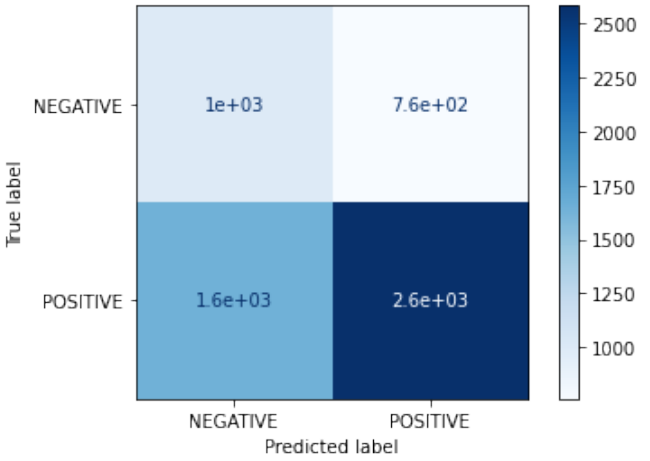

FIG. 19. Adaboost Confusion Matrix

4. *Multi-level perceptron*

Finally, our MLP had mediocre results, despite taking the longest to train. It had an accuracy of .39 with and an F1 of .72. This is very similar to the decision tree classifier. It also greatly over-fit the data, like the decision tree classifier.

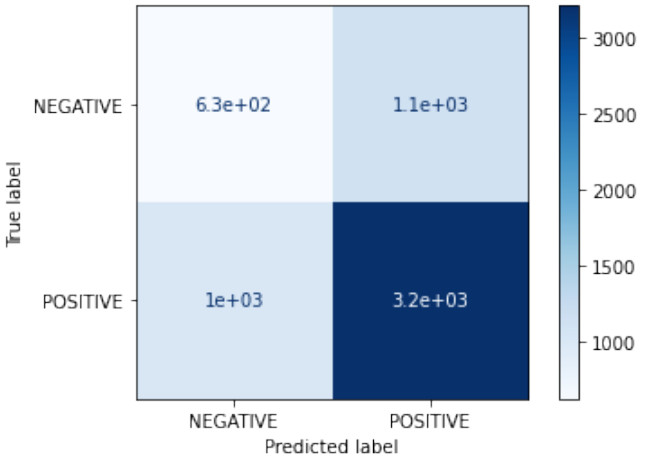

FIG. 20. MLP Confusion Matrix

V. GENERATIVE MODEL

For sub-problem 2, as stated we wanted to focus on training a generative model. Given an image, we hoped to be generate a caption that carries a certain sentiment: namely, positive or negative.

A. Feature Extraction

To extract features from images, we evaluated 2 pre-trained models:

1. InceptionV3: 8x8x2048
2. InceptionResnetV2: the method suggested by the paper released by Google with the dataset:8x8x1536

Both of these are pre-trained models trained on Google's imagenet dataset for the purpose of object detection. We extracted features from all the images in our data(train and test) by passing them through this pre-trained model and extracting the activation's from the lower convolutions layer of these models. The dimensions of the features extracted per image for each pre-trained model are given above. Each embedding is then squashed to size 64x1536 or 64x2048 (depending on the pretrained model).

To extract features from text, we used 300d GloVe word embeddings. Using GloVe's pre-trained model, we converted the words in our dataset into 300 dimensional vectors which carry some information about their meaning and usage. For example, if the GloVe embeddings of two words are very similar, it is very likely these words are similar: this allowed the model to learn some semantic information about captions. In the future, we hope to use BERT word embeddings, unlike GloVe embeddings, BERT embedding are able to capture the context in which words are used and are able to differentiate between homonyms.

B. Methodology and Training

1. Model architecture

We used a sequence-to-sequence model for this task inspired by the paper *Show, Attend and Tell* by Kelvin Xu [2]. Our trained model works as follows: it is supplied the 'start' token, signifying the start of a sentence and the pretrained embedding of an image. Given this, it will go on generating the next word in a caption and feeding that word back to itself and repeating the process until it generates an 'end' token or reaches a maximum number of words. The words generated between the start and end token are the final caption for that image. Architecture Parameters:

1. img-emb-dim: number of neurons in FC layer of Encoder that pre-trained image embeddings are passed through
2. units: size of the LSTM's hidden state
3. word-emb-dim: size of word embeddings: 300 in our case
4. vocab-size: number of words in our vocabulary, as per the training set: we tried keeping all words and also capping the maximum number of words

We used the following layers in our model:

1. ENCODER LAYER
   (a) Takes in pre-rained CNN embeddings with dims (64x2048 for InceptionV3 or 64x1536 for InceptionResnetV2): Since this is taken from lower convolutional layers, each of the 64 vectors represents a different spatial 'part' of the image according to the paper *Show, Attend and Tell* by Kelvin Xu
   (b) Passes them through an fully connected layer of output dims: (64ximg-emb-dim)
   (c) Applies Relu activation
2. ATTENTION LAYER
   (a) Sits inside the decoder model
   (b) Takes in output of encoder model and LSTM's hidden state at the previous step (discussed below).
   (c) Passes a concatenation of these into a Dense Layer with tanh activation with output dims:(64xunits)(Note, this is a deviation from the paper: the paper passes the encoder output and LSTM hidden state into two separate equal sized Dense Layers and adds their outputs before applying tanh)
   (d) Passes the output of the Dense Layer to a single neuron dense layer to get the attention weights for each of the 64 'parts' of the image. Uses softmax to normalize attention weights so that they sum up to one.
   (e) Finally, each of the 64 vectors in the CNN's output are weighted by their respective attention weight to calculate a 'context vector' that is the output of the Attention Layer.
3. DECODER LAYER: LSTM
   (a) Takes in a word and a context vector and outputs the next word in the caption (along with a hidden state)
   (b) Word is first embedded using GloVe
   (c) Then, encoder output along with the hidden state (initialized at 0 for start token) are passed through attention layer to get context vector
   (d) Context vector and word embedding are concatenated and passed through LSTM to get hidden state and output
   (e) Output is passed through 2 fully connected layer, last one with vocab-size neurons and softmax activation representing the probability of each word in the vocabulary

(f) Word with highest probability is the next word generated

In order to generate captions with a certain 'sentiment' attached to them, we simply divided our training data into 2 parts based hugging face's sentiment label on the captions. Then, in order to obtain a model generating positive sentiment captions, we simply only trained the model on the image-caption pairs whose captions were labelled 'positive' by huggingface. We did the same for our negative caption model. This would allow the LSTM to learn a 'positive language model' for generating captions, such that the sentences generated were more likely to be labelled 'positive' by huggingface.

This Jupyter notebook contains all the code we wrote to build and train all our generative models.

2. *Model Training*

The model was trained using the following steps: For each image:

1. Pass the image's squashed pre-trained embedding into the encoder
2. Pass the 'start' token into decoder (whose hidden state is initialized to 0 when a new image is passed in)
3. Obtain a prediction for the next word in the caption (as outlined in the architecture)
4. Calculate Categorical Cross Entropy loss using the prediction and the actual word after the one fed into the model.
5. Use teacher forcing: instead of feeding the predicted word back into our decoder, we feed in the actual word in the caption to generate the next word
6. Feed the decoder's hidden state back into the decoder in order to calculate attention weights for the each image vector with respect to the next word.
7. Repeat 3-4 until an 'end' token is predicted or we reach the maximum caption length, then move on to the next image and repeat.

As mentioned above, we trained positive, negative and general sentiment models. Because we had fewer negative sentiment image-caption pairs, the training data for the negative caption models was much less that for positive sentiment models, leading to worse metrics and caption quality in the negative sentiment models.

This Jupyter notebook shows our pipeline for preprocessing and tokenizing the captions in our training data.

3. *Evaluation Methodology*

For evaluation, we extracted an unseen set of 5000 image caption pairs and used it as our test set. We also used a random sample of 5000 images from our training set to calculate some metrics on our training set because evaluating on the entire training set was too time consuming. The following metrics were calculated to evaluate our model, and averaged across all train/test samples.

1. BLEU-4 score: BLEU or bilingual evaluation understudy is a metric developed to compare a generated translation to the real translation. It ranges from 0 to 1, where 1 is a perfect match and 0 a perfect mismatch. It is calculated by comparing n-grams. Since we use BLEU-4, it compares unigrams, bigrams, trigrams, and 4-grams.
2. METEOR score: METEOR or metric for evaluation of translation with explicit order is another common metric for language models. It uses unigrams from the generated text and attempts to match it to a unigram in the true text. It was created to address shortcomings in BLEU.
3. Rouge1 score: Rouge1 or recall-oriented understudy for gisting evaluation compares the overlap of unigrams from the generated to true text.
4. RougeL score: RougeL compares the overlap of the longest common sub-sequence from the generated to true text to account for sentence elve structure similarity.
5. % generated captions classified negative/positive by huggingface's classifier: We use to to see if sentiment is either being maintained or shifted by the different models.

This Jupyter notebook outlines our pipeline for evaluating our model on training and unseen testing using the various metrics mentioned above.

C. Model Variations and Evaluation

We evaluated 2 slightly different variations of the generative model. For each variation we trained and evaluated positive, negative and general caption generation models. The following models were trained

1. *Initial Model: InceptionV3 model + Full vocabulary*

The initial model we trained used pre-trained embeddings from InceptionV3 and did not limit the size of the vocabulary. We trained the model on 40 epochs for the general model and 20 epochs each. Initially, we were only able to save model weights from the last 20 epochs

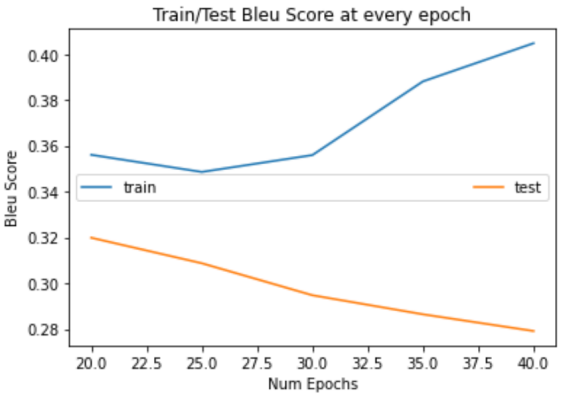

FIG. 21. Train/Test BLEU-4 Plot on Initial Generative Model: No sentiment

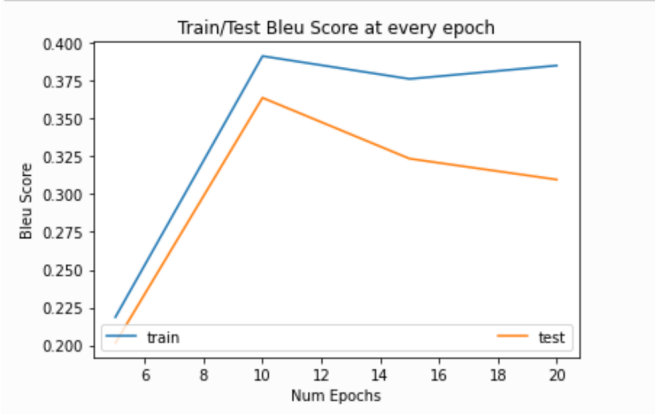

FIG. 22. Train/Test BLEU-4 Plot on Initial Generative Model: Positive sentiment

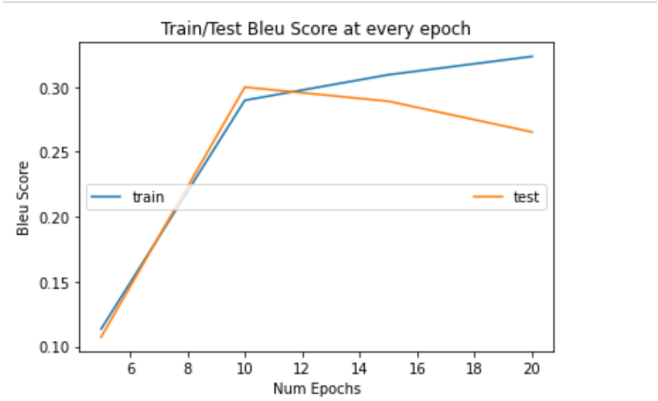

FIG. 23. Train/Test BLEU-4 Plot on Initial Generative Model: Negative sentiment

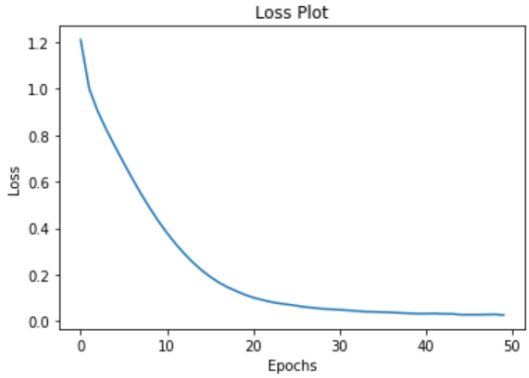

FIG. 24. Loss per epoch: Second Generative Model: No sentiment

of each model. FIG. 21-23 shows the BLEU-4 score of each type of model a sample of 5000 images from train and test across various epochs. We can see that the general model seems to be over fitting a lot after 20 epochs. This is why we decided to train the rest of the models for less that 20 epochs.

2. *Second Model: InceptionResnetV2 model + Limited vocabulary*

For the second model we trained used pre-trained embeddings from InceptionResnetV2 and limited the size of our vocabulary to the top 10000 words for the general and positive model and top 8000 words for the negative model (since there were only around 8200 words in the negative set). We trained the model on 20 epochs for the general model and and positive models each but for 50 epochs on the negative model because we had lesser data and the model loss was still decreasing at a high rate at 20 epochs. This time, we were able to save the weights from all the epochs we trained on. FIG. 24-26 shows the the loss of each each type of model across the epochs. The BLEU-4 score distribution on train/test across epochs for this model looked quiet similar to the previous model. FIG. 27 shows the average BLEU-4 score of the negative of model a sample of 5000 images from train and test across various epochs.

D. Results

In this section, we include the various metrics mentioned above by training each model for the number of epochs with the best BLEU-4 score on the test set (as seen in the plots above) and then running those models on the test set. We display the mean metric across all test samples.

1. *Initial Model: InceptionV3 model + Full vocabulary*

1. General model: trained on all positive and negative captions: best (known) BLEU-4 score was achieved

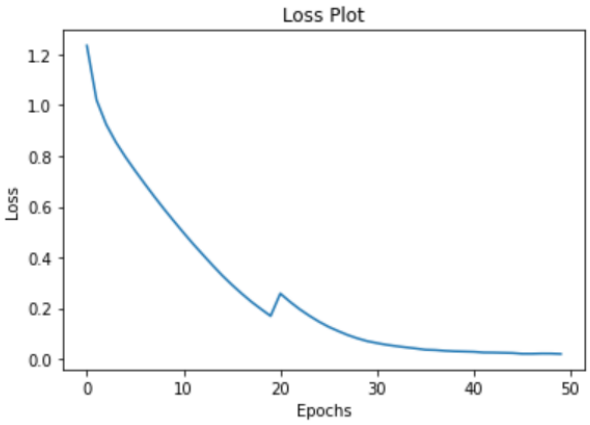

FIG. 25. Loss per epoch: Second Generative Model: Positive sentiment

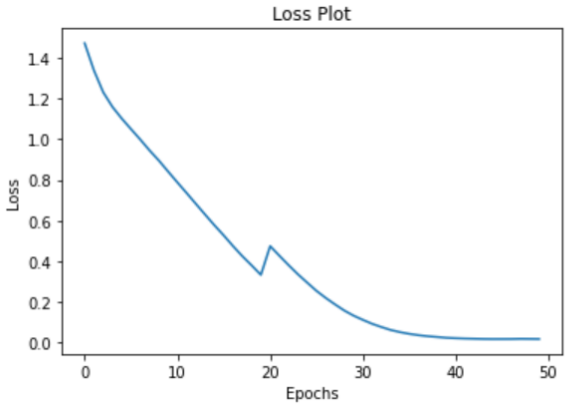

FIG. 26. Loss per epoch: Second Generative Model: Negative sentiment

at 20 epochs. FIG. 28 shows some a random sample of the captions generated on the test set.

(a) BLEU-4 score: 0.32

(b) METEOR score: 0.13

(c) Rouge1 score: 0.19

(d) RougeL score: 0.18

2. Positive model: trained on all positive captions : best (known) BLEU-4 score was achieved at 10 epochs. FIG. 29-30 shows some of the captions generated on the test set.

(a) BLEU-4 score: 0.36

(b) METEOR score: 0.13

(c) Rouge1 score: 0.2

(d) RougeL score: 0.18

(e) % generated captions classified positive: about 70% of the captions in the test set were classified as 'positive' by huggingface's classifier: this was not a major shift in the % of actual captions that were classified positive by hunggingface in the test set.

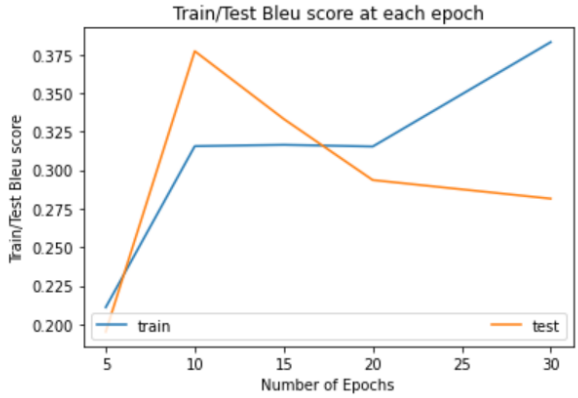

FIG. 27. Test Bleu Plot on Second Generative Model: Negative sentiment

3. Negative model: trained on all negative captions : best (known) BLEU-4 score was achieved at around 10 epochs. FIG. 31-32 shows some of the captions generated on the test set.

(a) BLEU-4 score: 0.3

(b) METEOR score: 0.1

(c) Rouge1 score: 0.14

(d) RougeL score: 0.13

(e) % generated captions classified negative: 85% of the generated captions were classified as negative by huggingface's classifier: this was a clear shift from the original distribution of 30% negative label captions of the test set!

2. Second Model: InceptionResnetV2 model + Limited vocabulary

1. General model: trained on all positive and negative captions: best (known) BLEU-4 score was achieved at around 10 epochs. The quality of the captions generated was similar to the initial model with the difference of some captions having the 'unk' token (since vocab size limited). Examples are not included for the sake of brevity.

(a) BLEU-4 score: 0.35

(b) METEOR score: 0.13

(c) Rouge1 score: 0.2

(d) RougeL score: 0.18

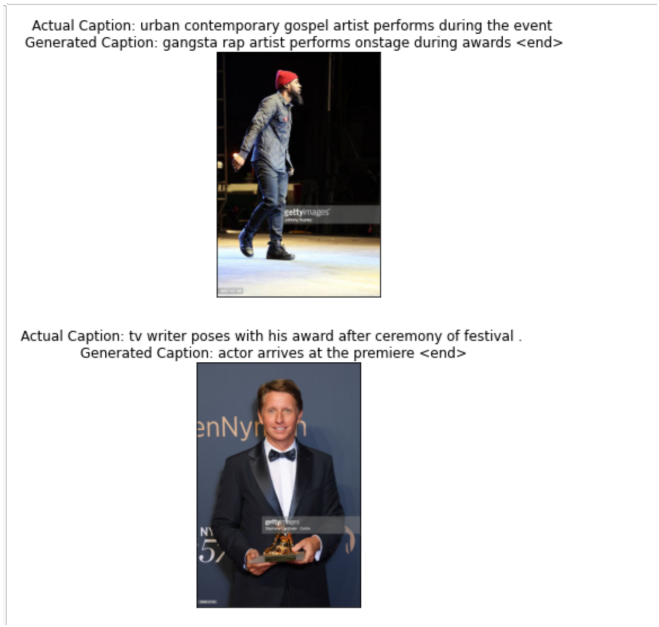

FIG. 28. Sample Captions: Initial Model: No sentiment

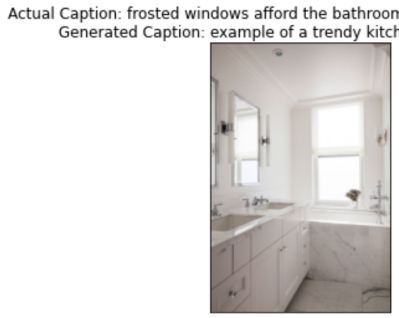

FIG. 29. Sample Captions: Initial Model: Positive sentiment

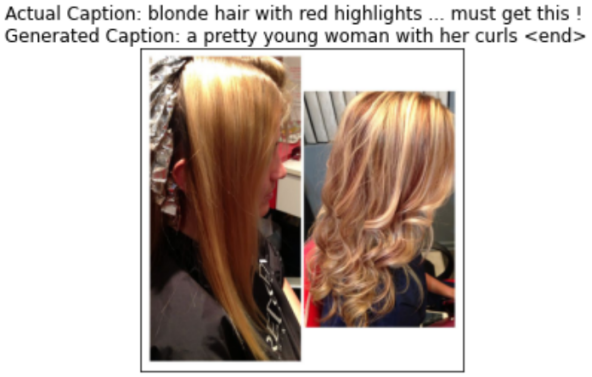

FIG. 30. Sample Captions: Initial Model: Positive sentiment

FIG. 31. Sample Captions: Initial Model: Negative sentiment

FIG. 32. Sample Captions: Initial Model: Negative sentiment

2. Positive model: trained on all positive captions : best (known) BLEU-4 score was achieved at 10 epochs.
   (a) BLEU-4 score: 0.36
   (b) METEOR score: 0.13
   (c) Rouge1 score: 0.2
   (d) RougeL score: 0.19
3. Negative model: trained on all negative captions : best (known) BLEU-4 score was achieved at 10 epochs.
   (a) BLEU-4 score: 0.38
   (b) METEOR score: 0.11
   (c) Rouge1 score: 0.19
   (d) RougeL score: 0.18

VI. DISCUSSIONS/CONCLUSIONS

1. *Sub-Problem 1*

As our results clearly show, we did not produce a good model that could classify the sentiment of an image. It is hard to tell whether is a failure of our model or our data. We wished to do more analysis on our extracted features,

but they are very abstract. If we were to continue working on this project in the future, we would like to trying using the InceptionV3 output with a TensorFlow neural network. The hope is that the large feature space reveals more about the image. Furthermore, writing a neural network from scratch would allow us to more fine-tune it.

2. Sub-Problem 2

Looking at subproblem 2, we also get some questionable results. As seen in our results, the quality of the generated captions varies greatly in quality. Overall, our models obtained better than random metrics, and we could see that they were clearly able to learn something from the images and caption them somewhat intelligently, if not always accurately.

For the models trained on less data, and with a smaller vocabulary like the negative sentiment models, our metrics also show unexpected results, and we are unsure about their accuracy. For example, the quality of the caption on the test set, using human judgment, appeared to increased when epochs increased. This seems expected. However, the BLEU-4 score decreased as epochs increased. This may be because BLEU-4 score is not the best metric to judge such a model.

Our model seemed to perform really good on the type of images that were very common in our dataset like those of celebrities in award shows, fashion shows etc- but not so well on rarer images.

We can attribute the shorting coming of our model to either its architecture or the data-set. The architecture was not rigorously tuned. The data-set also had a wide range of images and captions, unlike MSCOCO, and may require a much larger dataset (and perhaps a more complicated model) to properly learn the task well.

As seen in the results, the model clearly performed better when images had fewer objects in them. We acknowledge we do not have an expanse interpretation of our results as we spent much of our effort learning and implementing the architecture. Further, training and evaluation in and of itself was very time consuming, and it took hours to obtain meaningful metrics.

The project allowed us a very in-depth understanding of the working of sequence to sequence models as well as attention mechanisms. Furthermore, we gained practical skills in training and debugging complex models in Keras, which we think will be really valuable for us in the future.

Just like our predictive model, there are many ways we extend our generative model. The first thing we would like to do to extend our project is simply training our model on more data. 30k images is a relatively small amount of data, especially when the Google Conceptual Caption data-set has 3 million. Another area we can extend the project is by changing the embeddings we used. We have read that BERT has great embeddings for text generation, which are able to learn contextual embeddings of a word. Finally, we could have added a Gaussian blur to our image to prevent over-fitting.

VII. REFERENCES

1. Slides: https://docs.google.com/presentation/d/1py3B492MF
   xcBG5srXQWG4/edit?usp=sharing
2. Code: https://github.com/apragupta/ML_FP_New
3. https://ai.google.com/research/ConceptualCaptions/
4. https://arxiv.org/abs/1502.03044/
5. https://www.coursera.org/learn/nlp-sequence-models
6. https://arxiv.org/pdf/1510.01431.pdf
7. https://www.tensorflow.org/tutorials/text/image_captioning
8. https://www.hindawi.com/journals/cin/2020/3062706/

VIII. CONTRIBUTIONS

Apra Gupta: Generative model (sub-problem 2), data-set analysis

David Zheng: Predictive model (sub-problem 1), data processing pipeline

---

[1] Https://ai.google.com/research/ConceptualCaptions/.

[2] https://arxiv.org/abs/1502.03044/.