Image Captioning using Deep Neural Networks

Akarsh Balasubramanyam
Computer Science
University of Illinois at Chicago
Chicago, Illinois, United States of America
abalas26@uic.edu

ABSTRACT

Image captioning requires methods from both Computer Vision and Natural Language Processing. Computer vision is used to understand the content of the image and NLP is used to transform the learned content into words in the correct order. Deep Learning method have demonstrated state-of-the art results in Image captioning problems. A single end-to-end model is used to predict a caption given an image using transfer learning wherein we extract the features of the image using one model and generate word embedding for the captions using a separate sequence to sequence model, at the end we combine the inputs from both the models to generate captions for unseen images. Finally to evaluate how well our model has performed we will use BLEU (Bi-lingual Evaluation Understudy) scores.

INTRODUCTION

When you see an image your brain can easily tell what the image is about, but can a computer tell what an image is representing? Researchers are currently working on it, where provided an image, a computer is able to represent the features of it. Caption generation is a challenging task in the field of Artificial Intelligence where a textual description is generated for a given photograph. Image captioning has several advantages, it helps the visually impaired who find it hard to understand image features. It can used by the traffic controllers to recognize number plates on vehicles. It can serve as recommenders in editing applications. With the latest advancements in Deep Learning and access to variety of public datasets, computer scientists can build models that can recognize features in the images and generate captions. Computer Vision and Natural Language Processing techniques are extensively used to achieve state of the art results for image processing tasks.

BACKGROUND

Natural Language Processing

Natural Language Processing (NLP) a subfield of Linguistics Computer Science and Artificial Intelligence (AI) concerned with interaction between computer and human languages, in particular how to program computers to process and analyze large amounts of natural language data. With additional added features, NLP systems are trained for Part of Speech (POS) tagging, Identifying named entities and they are most widely used along with Machine Learning techniques to identify patterns and provide solutions to natural language data.

Computer Vision

Computer Vision is an interdisciplinary scientific field that deals with how computers can gain high level understanding from digital images and videos. From an engineering standpoint it deals with understanding and automating tasks that a human visual system can do. It includes methods for acquiring, processing and analyzing features from digital images. Computer vision aids in research areas such as 3D modelling, Object detection, Image analysis and restoration.

Machine Learning

Machine Learning is the study of computer algorithms that improve automatically through experience. It is a subfield of Artificial Intelligence. The idea of Machine Learning is to make computer program learn and predict data without explicitly programmed to do so.

Deep Learning

Deep Learning is a subfield of AI and Machine Learning family that has the capability of learning from unstructured and unsupervised data. It is made of layers of Artificial Neural Networks (ANN's) that mimic the functioning of a human brain. Deep Learning has been applied to fields of NLP, Computer Vision, Speech recognition etc. and they have provided results surpassing human expert performance.

EXISTING MODEL

There are certain existing methods like Object Detection within an image that are used to generate a caption. Based on the objects identified within the images, a suitable caption is generated. These methods can help identify the object and the image features but fail to generate captions that are semantically correct. The reason behind this is the model is not trained with any words or sentences. Having the image features trained with human annotated captions can help achieve state-of-the art results. This project aims to combine the features of both Computer Vision and Natural Language Processing to extract features from the image and map the captions to the generated features respectively. Having a sequential model trained to map captions with features, the model can now process new images and generate words for the features extracted. With the proposed methodology training the model can be expensive but it achieves better performances generating captions on unseen data.

CORPUS

The two most popular datasets available for Image captioning are the MS COCO (Common Objects in context) dataset and the Flickr dataset. MS COCO dataset contains images and 5 human annotated captions provided by the Microsoft Corporation and the data is divided into train, dev and validation sets. The only disadvantage of this dataset is that the captions are not provided for the testing set. This can just be used for predictions but the model cannot be evaluated for performance. The Flickr dataset is similar to the MS COCO dataset, 30k images are collected from Flickr and for each image 5 human annotated captions are provided. The dataset is further divided into train, dev and test sets. Because of limited computational power available, we will be using the abridged version of the Flickr dataset (Flickr8k). This dataset is a subset of the Flickr 30k data, which provides 6000 images of training and 1000 images for validation and testing respectively. The Flickr dataset provides captions to the testing set, so that we can evaluate the performance of the model using BLEU scores. With sufficient computational power and time, the same model can be experimented with the Flickr 30k dataset which helps the model to learn new features and provide better performance.

PROPOSED METHOD

In this method we will be using a sequential model built using Convolutional Neural Networks (CNN's) to extract the features from the images which will then be combined with the text features extracted from the Recurrent Neural Networks (RNN's). This combination of the model is used to predict the next word of the sentence. We will be using a startseq and endseq to identify the start and ending of a sentence. The captions generated by the model are then evaluated with the original human annotated captions using corpus_bleu for unigram and bigram overlaps.

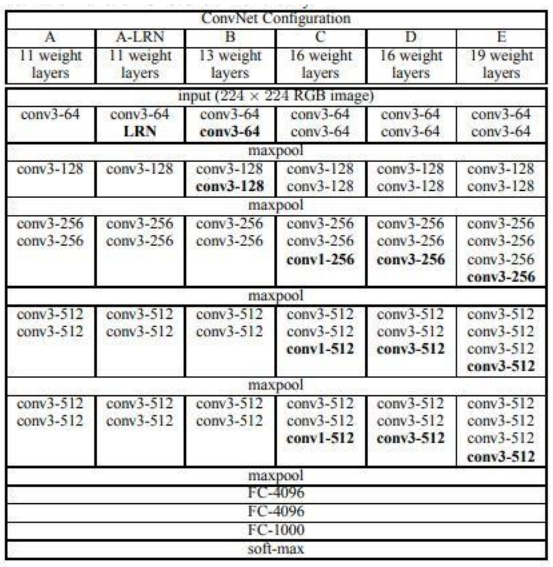
Image Pre-Processing

For extracting the image features, we will be using Convolutional Neural networks. There are a whole set open-source pre-trained models to choose from which are built using CNN and Pooling layers with Batch normalization and Dropouts, for this experiment we will be using only some of the models that are highly efficient and can be trained using lesser number of parameters keeping in mind the computational and time complexity. Some of the imageNet models that we will be using are VGG16, Inception V3, Xception and Resnets.

VGG16 (Visual Geometry Group Model - 16)

VGG16 is a CNN model proposed by the Visual Geometry Group at the University of Oxford in a paper " Very Deep Convolutional Networks for Large-Scale Image Recognition). The model consists of series of Convolutional and Pooling layers with ReLU activation functions. The architecture of the model is shown in figure 1.

The original VGG16 model was built for image classification. For the purpose of this experiment, we will be removing the final SoftMax layer which makes the predictions. The final Fully connected layer's output will provide us the image features. TensorFlow Keras provides the pre-trained model with the model weights which will be downloaded on model initialization. Extracting the image features will take some time depending on the hardware used. Instead of generating the features for every run of our model we will be extracting the features from the VGG model and dumping them in a pickle file which can then be loaded for further processing. Keras also provides functions for reshaping and pre-processing the image so that VGG model can process it. We will be re-sizing the image to 224 * 224 resolution with 3 channels one each for Red, Green and Blue. The extracted features from the final layer is of size 1 * 4096



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 1: Architecture of VGG16 model (image credits: https://arxiv.org/abs/1409.1556)

Inception V3

The Inception V3 architecture was introduced by Szegedy et al. in the paper "Going Deeper with Convolutions". Inception was a multi-level feature extraction model computing 1*1, 3*3 and 5*5 convolutions in the same module of the network. The original architecture of Inception was called GoogleNet. The model was then made available on Keras based on a separate paper " Re-thinking the Inception Architecture for Computer Vision"

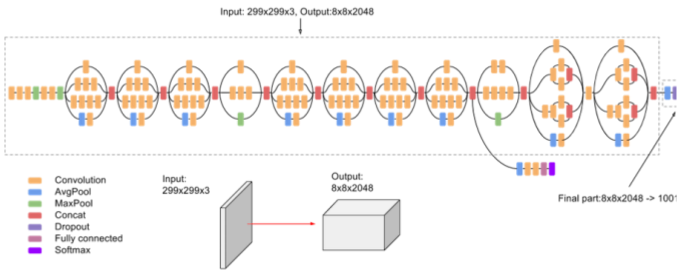


Figure 2: Architecture of Inception V3 model (image credits: https://cloud.google.com/tpu/docs/inception-v3-advanced)

Similar to VGG16, the final SoftMax layer is removed from the model and image features are extracted and dumped into a pickle file for later processing. Since the model has deep convolutional layers, this usually takes more time to load the weights and run the model compared to VGG16. The image is pre-processed using Keras by reshaping it to 299 * 299 resolution with 3 channels for Red, Green and Blue. The final extracted features are of size 1 * 2048.

Xception

Xception was proposed by Francois Chollet, the creator and maintainer of the Keras library. It is an extension of the Inception model which replaces the standard modules with depth wise separable convolutions. It was introduced in the paper "Deep Learning with Depth wise Separable Convolutions".

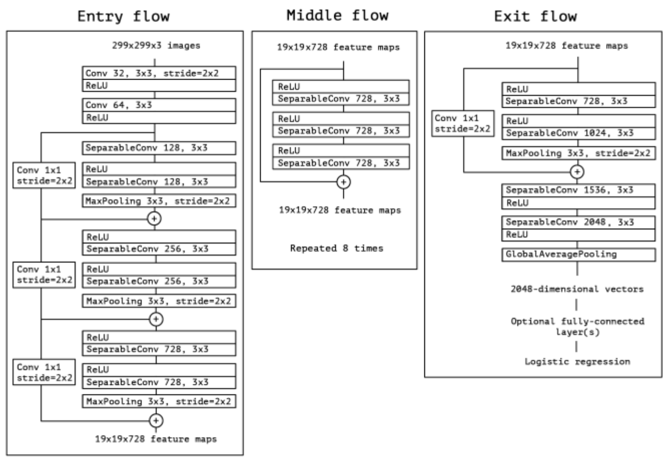
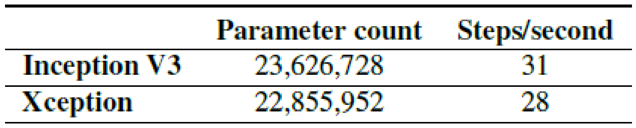


Figure 3: Architecture of Xception model (image credits: https://arxiv.org/abs/1610.02357)

As compared to the Inception model, introduction of separable convolutional layers reduces the number of parameters making the feature extraction faster.



	Parameter count	Steps/second
Inception V3	23,626,728	31
Xception	22,855,952	28

Figure 4: Parameter Comparison of Inception and Xception (image credits: https://arxiv.org/abs/1610.02357)

The final logistic regression layer is removed and the image features are extracted using the final fully connected layer. The extracted features are dumped to a pickle file. Using the Keras pre-processing libraries the image is resized to 299 * 299 resolution and 3 channels. The final output layer has a dimension of 1 * 2048.

Resnet50

Resnet short for Residual Neural Network are implemented with double or triple layer skips that contain non-linearities and batch normalizations in between the layers. Resnets relies on the micro architecture model (network-in-network architecture). This model was introduced by He et al. in the paper " Deep Residual Learning for Image Recognition" which contain 50 weight layers. Compared to VGG models Resets have a deeper layers but the model size is substantially smaller due to the usage of global average pooling rather than fully connected layers. The architecture of the Resnet50 is shown in Figure 4.

The input image is pre-processed using the Keras libraries and compressed to a 224 * 224 resolution. The extracted features are dumped into a pickle file. The output features has a dimension of 1 * 2048.

Text Pre-Processing

Each image in the dataset is associated with 5 human annotated captions. The descriptions must be cleaned and processed before passing it as an input. The input texts are loaded from the file and pre-processed by converting the characters to lower case and removing any punctuations in the text. To generate a robust vocabulary, any words that is neither an alphabet or a numeric is removed and words with word length less than 1 are removed.
To avoid pre-processing the same texts for every run of the model. The captions are pre-processed and dumped in a text file which is then used as a look-up with the image features which form the input to our model.

Splitting the data

Once the data is cleaned, the image features and captions are loaded from their respective files and are split into train, dev, test datasets. The training dataset now contains features of 6000 images with 5 cleaned captions for each image. The dev set and the test set each has 1000 images with 5 cleaned captions for each image respectively. In order for the model to recognize the start and end of the sentence, we append a “startseq” and “endseq” token. This will also help during prediction, i.e. generate the next word until an end of sentence token is encountered.

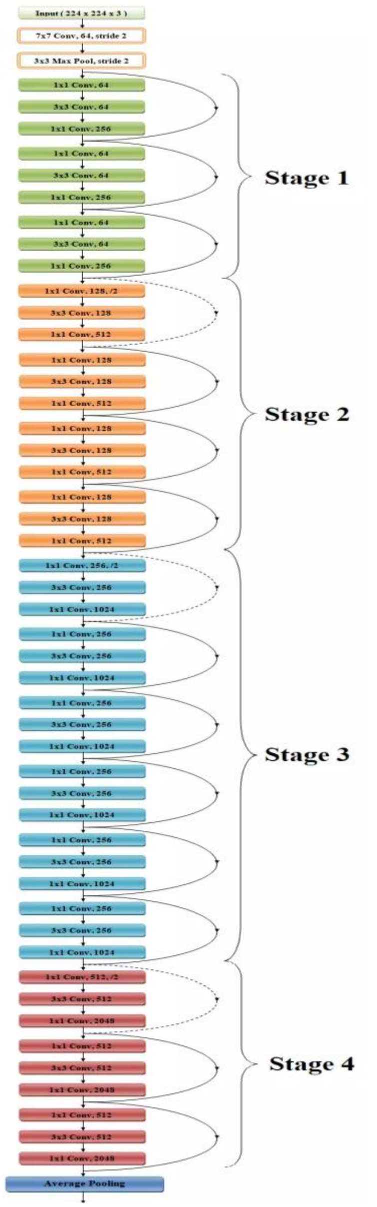


Figure 4: Architecture of Resnet50 model (image credits: https://arxiv.org/abs/1512.03385)

The pre-processed captions of the train data are tokenized using Keras Pre-Processing libraries and the words are converted to their respective index value from the vocabulary generated. The max length of the sentence generated is used and any other sentences with length lesser than that of the max length sentence is padding with 0’s at the end of the sentence. Any Unknown vocabulary tokens are marked with “<OOV>” keyword.

Word Embeddings

Word Embedding is a collective name for a set of language modelling and feature learning techniques in NLP. The words are mapped to vectors of real number. In this system the GloVe Vectors are used as word embeddings. Global Vectors (GloVe) model is an unsupervised learning algorithm for obtaining vector representation of the words. These words are downloadable from the Stanford NLP Repository. From the set of the words in this repository, we generate the vector form of the word.

For this project we will be using the 300 dimensional GloVe vectors. An embedding matrix is created with shape equal to size of vocabulary * 300. Each row in the matrix correspond to the word in the vocabulary and its 300D representation. While passing the words as inputs to the model, the words are transformed into its respective embeddings using this matrix.

Generating the Input Sequence

Since the model we use here is used to predict the next word in the caption. We will encode the input data to the model in the similar way. Firstly we will be passing the startseq as input along with the image features to the model and the next word in the caption forms the output, in the next time step we perform the same but the output of the previous word is concatenated to the current input and a new input is generated. This input along with the image features are provided as input to our model.

Consider an example of the caption : “man wearing red shirt”. The encoded input sequence is shown below

Input (Image)	Input(word)	Output(word)
image_features	startseq	man
image_features	startseq man	wearing
image_features	startseq man wearing	red
image_features	startseq man wearing red	shirt
image_features	startseq man wearing red shirt	endseq

Once the model is generated, we will be concatenating the tokens and recursively provide inputs to generate the output. At the end by removing the startseq and endseq tokens we generate the caption.

The model input are vectors which are passed to an embedding layer which is then combined with the image features separately. The output of the model will be a SoftMax over the probability distribution of all the words in the vocabulary. The output will be one-hot-encoded for each word with a value of 0 for all the positions except the position of the actual word which will have a value 1.

We now have the pre-processed captions and image features which form the input to our model. We will be generating a sequence model with these inputs and predicting the output.

MODEL SETUP AND LEARNING

The merge-model described by Marc Tanti et al. in the paper "Where to put the Image in an Image Caption Generator" is used as a reference to generate the model.

Hardware Requirements:
Since the image feature generation and pre-processing the captions are common for all the models, we will be processing them initially and dump them in a pickle files. Later the complete features are retrieved and merged with the captions to generate the output. A workstation with a minimum of 32 or 64GB RAM and GPU is recommended for the model to run. Google Cloud Platform with 16 vCPU's and 128GB ram is used to run our models with an average time of 30-40 mins per epoch. The model can also be run on workstations with limited computational power using progressive loading that generates input sequences on the go.

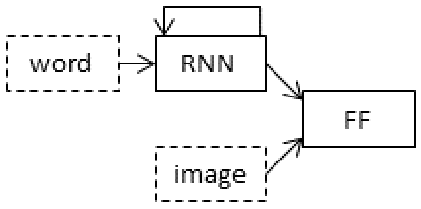


Figure 5: Block Diagram of the merge-model (image credits: https://arxiv.org/abs/1703.09137)

The image feature extractor in the model refers to the different imageNet model that are used like VGG16, Inception, Xception and Resnet50. The features extracted from these models are used as inputs. The sequence processor is the word embedding layer for handling the text input which is then followed by a Bi-Directional LSTM (Long Short Term Memory) layer.

The final decoding layer combines the input of both the images and sequences and uses Dense (Fully connected layer) layers to generate the next word in the caption.

Figure 6 shows the structure of the model. The first Input layer (Input 1) tokenizes the image captions and pads the sequences necessary to the max length of the captions which is followed by a embedding layer that uses the GloVe vector embeddings of 300D. We add a dropout after this layer with a dropout probability of 0.5. The inputs are then passed to a Bi-Directional LSTM with 256 memory units.

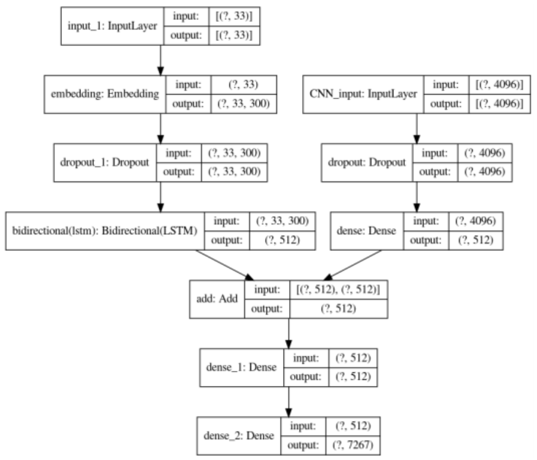

input_1: InputLayer | input: [(?, 33)] | output: [(?, 33)]
embedding: Embedding | input: (?, 33) | output: (?, 33, 300)
CNN_input: InputLayer | input: [(?, 4096)] | output: [(?, 4096)]
dropout_1: Dropout | input: (?, 33, 300) | output: (?, 33, 300)
dropout: Dropout | input: (?, 4096) | output: (?, 4096)
bidirectional(lstm): Bidirectional(LSTM) | input: (?, 33, 300) | output: (?, 512)
dense: Dense | input: (?, 4096) | output: (?, 512)
add: Add | input: [(?, 512), (?, 512)] | output: (?, 512)
dense_1: Dense | input: (?, 512) | output: (?, 512)
dense_2: Dense | input: (?, 512) | output: (?, 7267)

Figure 6: Plot of the model used for generating the captions (The shape of the input image features varies with the model used)

The CNN_input layer expects inputs with the dimensions equal to the extracted features. We add a dropout layer with a probability of 0.5 to avoid overfitting. These inputs are processed by a Dense layer of 512 memory units.

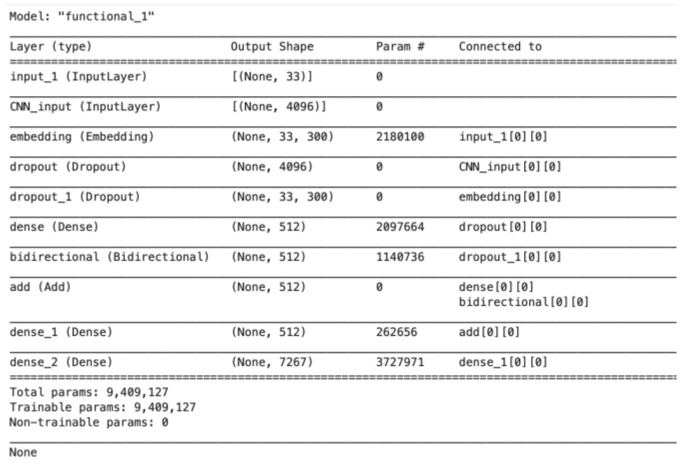
Both the input models produce a 512 element vector which is then merged and passed to a Dense layer with 512 memory units. This is followed by a final Dense layer that uses the SoftMax activation function over the entire output vocabulary to generate the next word in the sequence. Figure 7 shows the summary of the model used for training and the number of parameters generated to train the model.

The model is fit with the training data generated in the pre-processing step and the dev dataset is used for validation. Keras provides a model checkpoint functionality which can be used to save the model which has the least validation loss over all the epochs of learning. The model is run for a maximum of 10 epochs which takes around 30-40 mins per epoch on the Google Cloud Platform. Only the model with the best validation loss will be saved and this can be used to evaluate the performance of the model on the unseen test data or predict a caption for a new image.

MODEL EVALUATION AND RESULTS

Once the model is fit, we can evaluate the performance on the model on the unseen test dataset. The model is evaluated by generating captions for all the images in the test dataset and evaluating them with the ground truth.

Since the model always predicts the next word in the sequence, to predict a caption for an image, we will pass the startseq token to the model and the next word is predicted based on the image features and the captions learnt. Similar to training the model, the generated output sequences are used as input until the endseq token is encountered. After generating the complete sentence, the start and end sequence tokens are removed to generate the final image caption. The performance of the captions generated is evaluated using BLEU Scores.



```
Model: "functional_1"
________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
input_1 (InputLayer)            [(None, 33)]         0
________________________________________________________________________________________________
CNN_input (InputLayer)          [(None, 4096)]       0
________________________________________________________________________________________________
embedding (Embedding)           (None, 33, 300)      2180100     input_1[0][0]
________________________________________________________________________________________________
dropout (Dropout)               (None, 4096)         0           CNN_input[0][0]
________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 33, 300)      0           embedding[0][0]
________________________________________________________________________________________________
dense (Dense)                   (None, 512)          2097664     dropout[0][0]
________________________________________________________________________________________________
bidirectional (Bidirectional)   (None, 512)          1140736     dropout_1[0][0]
________________________________________________________________________________________________
add (Add)                       (None, 512)          0           dense[0][0]
                                                                 bidirectional[0][0]
________________________________________________________________________________________________
dense_1 (Dense)                 (None, 512)          262656      add[0][0]
________________________________________________________________________________________________
dense_2 (Dense)                 (None, 7267)         3727971     dense_1[0][0]
================================================================================================
Total params: 9,409,127
Trainable params: 9,409,127
Non-trainable params: 0
________________________________________________________________________________________________
None
```

Figure 7: Summary of the model used for generating the captions (The shape of the input image features varies with the model used)

BLEU Score

BLEU (Bi-Lingual Evaluation Understudy) is an algorithm for evaluating the quality of the text which has been machine translated. Quality is considered to be the correspondence between a machine's output and that of a human. BLUE was first proposed by Kishore Papineni et al. in the paper "BLEU: a Method for Automatic Evaluation of Machine Translation". BLEU's output is always a number between 0 and 1. A value closer to 1 represents similar texts.

NLTK's built in library corpus_bleu can be used to calculate the bleu scores for the generated captions and the captions that are human annotated. Unigram and Bigram overlap scores are calculated and the findings are discussed below.

Table 1 shows the individual Unigram, Bi-gram, Tri-gram and 4-gram BLEU scores for different imageNet model used on the validation dataset. Table 2 shows the bi-gram and tri-gram overlap for the same models. A score of 0 indicates that there were no common n-gram words in the corpus.

From this we can see that, VGG16 and Xception have performed better compared to other models. Resnet50 has better BLEU scores for the tri-gram and 4- grams but it fails to generalize well for Unigram and Bi-gram data compared to the other models.

Table 1: Individual Unigram, Bi-gram, Tri-gram and 4-gram BLEU Scores on Validation Dataset

Model	**1 gram**	**2 gram**	**3 gram**	**4 gram**
VGG16	0.6667	0.375	0.2857	0.1667
Xception	0.8889	0.625	0.4286	0.1667
Inception V3	0.5	0.0909	0	0
Resnet50	0.4286	0.1538	0.0833	0

Table 2: Cumulative Bi-gram and Tri-gram BLUE Scores on Validation Dataset

Model	**2 gram**	**3 gram**
VGG16	0.5	0.4186
Xception	0.7454	0.6228
Inception V3	0.2132	0
Resnet50	0.2568	0.1795

Looking at the results, we can make out that the Xception is the best performing model. We will use this model to evaluate the captions generated on the unseen test dataset and also show some examples of good and bad captions generated by the model.

To evaluate the performance of the test set, similar to the training data, the sequences are generated for the image captions and the features are extracted from the image using the best performing model. The model saved is used to evaluate the performance. Table 3 shows the performance of the model on the test dataset.

	Individual				**Cumulative**	
Model	**1 gram**	**2 gram**	**3 gram**	**4 gram**	**2 gram**	**3 gram**
Xception	0.4	0.1111	0	0	0.2108	0

We can see a small degradation in the BLEU scores for the test dataset because the images are not seen during training and the prediction is limited to the vocabulary seen during training the model.

Some of the examples for a good and bad captions are shown below. As discussed the model does not generate accurate results because it is trained on a limited dataset of just 6000 images and the test data vocabulary is not available. So some of the words in the test dataset captions are not available during prediction.

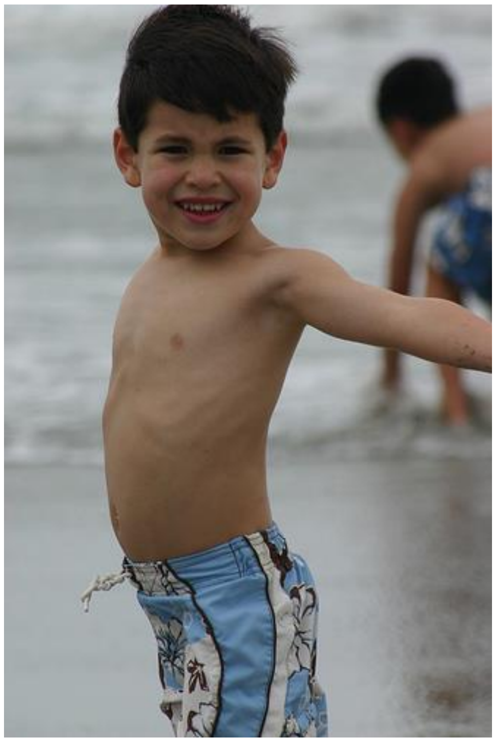


Image : 106490881_5a2dd9b7bd.jpg
Actual Caption : children playing on the beach
Predicted Caption : two children are playing in the water

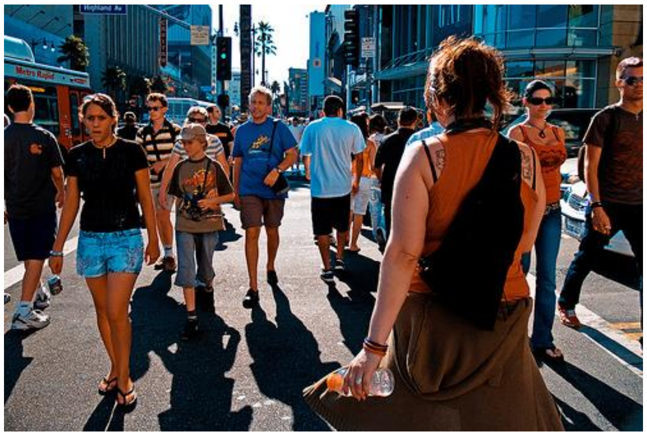


Image : 1174629344_a2e1a2bdbf.jpg
Actual Caption : group of different people are walking in all different directions in city
Predicted Caption: man in black shirt and black jacket is standing in front of building

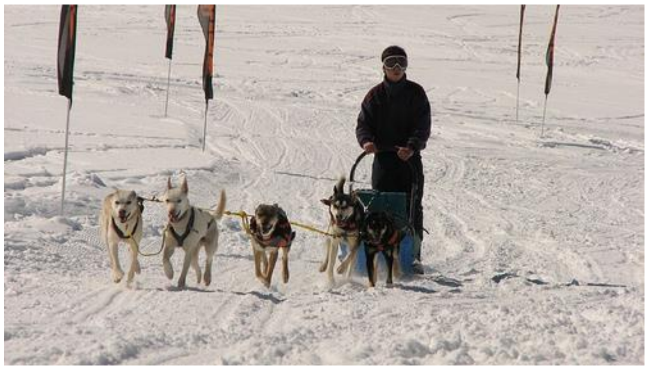


Image : 1287073593_f3d2a62455.jpg
Actual Caption : group of dogs pull person on dog sled uphill
Predicted Caption : two people are playing in the snow

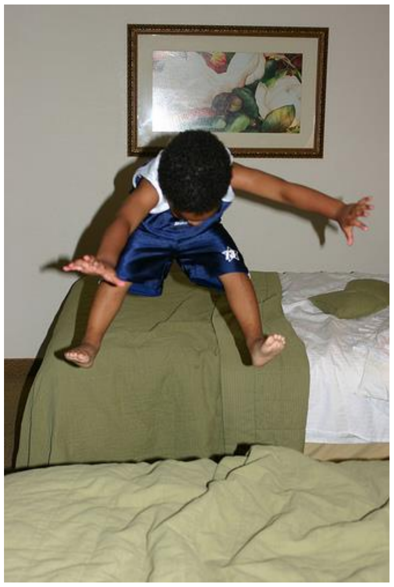


Image : 1131932671_c8d17751b3.jpg
Actual Caption : the boy in blue shorts is bouncing on the bed
Predicted Caption : the little girl is sitting on the grass

(The captions predicted for the above images are taken from the test dataset)

CONCLUSION

Using transfer learning, by extracting features from imageNet model and then combining them with a Sequence to Sequence models, we were able to generate better captions for the images. The features extracted and the sequences of text are learnt by our model and at every time step it generates a new word until the end of the sentence is reached. Some things to note about this project is our model is built on the set of vocabulary words seen during training, so any Out of Vocabulary words cannot be predicted. This is the reason we see lower BLEU scores for the test dataset. Due to limited computational ability, we have used a small dataset with just 8000 images. In production level models, several thousands of datasets are to be used to show accurate results.

FUTURE WORK

The current work considers image features and the word embeddings as input to the model which are then connected to a Fully Connected (Dense) layers to predict the next word in the sentence. The model does not consider any form of attention mechanisms. By having a attention mechanism in the decoder the words can be more aligned to the features in the context. This can be one of the best possible extensions to the current project.

Also, for the image pre-processing we have considered only a few imageNet models that reported higher performance with lesser parameters. Different models can be used to extract image features which can enhance the overall performance of the system.

Due to limited computational and time capabilities, we have considered an abridged version of the Flickr dataset. With the original Flickr 30k dataset, the model will have more visibility to the image features which can help achieve better performance.

The system can be trained with pre-trained transformers like BERT's, Hugging Face, etc. and the model performance can be evaluated.

REFERENCES

[1] Very Deep Convolutional Networks for Large-Scale Image Recognition, Karen Simonyan, Andrew Zisserman, arXiv:1409.1556

[2] Szegedy, Christian, et al. "Going Deeper with Convolutions." 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2015, pp. 1–9

[3] Szegedy, Christian, et al. "Rethinking the Inception Architecture for Computer Vision." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2016, pp. 2818–26

[4] Chollet, Francois. "Xception: Deep Learning with Depthwise Separable Convolutions." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 1800–07

[5] Tanti, Marc, et al. "Where to Put the Image in an Image Caption Generator." Natural Language Engineering, vol. 24, no. 3, May 2018, pp. 467–89

[6] Tanti, Marc, et al. "What Is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?" Proceedings of the 10th International Conference on Natural Language Generation, Association for Computational Linguistics, 2017, pp. 51–60

[7] Papineni, Kishore, et al. "BLEU: A Method for Automatic Evaluation of Machine Translation." Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02, Association for Computational Linguistics, 2001, p. 311

[8] Image Captioning: Transforming Objects into Words Simao Herdade, Armin Kappeler, Kofi Boakye, Joao Soares