**Image Caption Generator**

By: Britney Wang, Anqi Wu, Fangzhou Yuan

### 1. Introduction

**Background**: Humans can easily look at an image and understand what it’s about; we want to teach computers to do the same. Our project is creating a model that turns image data into text descriptions, aiming to tackle significant challenges in accessibility and assistive technology. Our model will be particularly useful for visually impaired people by providing them with specific color details, aiding those who are color blind. In a broader context, this model could potentially be integrated into devices like glasses that can describe surroundings in real-time. We've used CNN models and combined with LSTM networks to process and generate accurate and contextual descriptions of visual scenes.

**Dataset:** The dataset we used is the [Flickr\_8k dataset](https://www.kaggle.com/datasets/adityajn105/flickr8k/data) from Kaggle. It contains a collection of 8,000 images, each paired with five unique captions. These captions, while distinct, are crafted to convey uniform and consistent meanings, capturing the essence of the images.

### 2. Data Preparation

**Caption Text Preprocessing:** In this phase, we processed all descriptions by performing several data cleaning steps. Initially, we removed punctuation, converted all text to lowercase, and eliminated words containing numbers. For example, a caption like "Dog jumps over 3 hurdles!" would be transformed into "dog jumps over hurdles". Next, we also employed tokenization that maps each unique word in our vocabulary to a distinct numeric value. Lastly, we appended '<start>' and '<end>' identifiers to each caption so that it helps our model to easily recognize the beginning and end of each caption.

**Image Data Preprocessing:** In this phase, we employed computer vision techniques to augment our images. We started by resizing and normalizing all images to a consistent format, because normalization can significantly stabilize and accelerate the learning process. Then, we applied image augmentation techniques such as rotation, zoom, and flipping to simulate various scenarios and angles, allowing us to prevent overfitting and improve the model's robustness. Additionally, we experimented with converting the image color spaces, such as from RGB to HSV or grayscale, to highlight features that are more prominent in certain color spaces. Lastly, we applied noise reduction techniques such as Gaussian blur and median filtering to clean up the images, smoothing out the images and ensuring that the model focuses on learning relevant features.

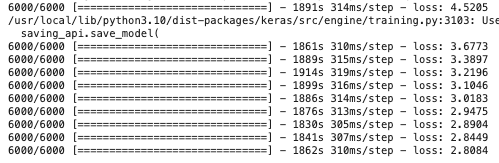
### 3. Pre-Modeling Data Transformation

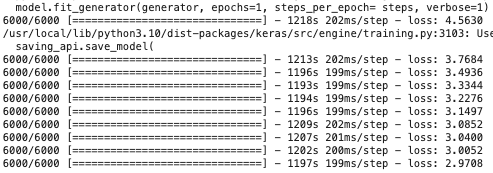
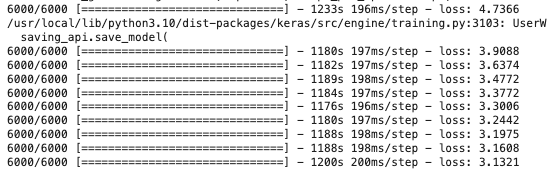
**Feature Extraction for Image:** For our project, the raw image data are transformed into feature vectors prior to being fed into the neural network. This feature vector represents a condensed version of the image's contents, capturing essential information about the visual patterns without tying it to specific object classifications. Therefore, we have experimented with the following feature extraction models:

***Xception:*** This model was pre-trained on the ImageNet dataset. In our application, we configured the Xception model to process inputs with the dimensions of 299 × 299 pixels, with 3 color channels. We removed the last fully connected layer to adapt the model for feature extraction. By removing this layer, instead of a classification output, we obtained a 2048-dimensional feature vector for each image.

***ResNet50:*** Just like Xception, it is a pre-trained model trained on ImageNet. It accepts input images with dimensions of 224 × 224 pixels and 3 color channels. Additionally, ResNet50 employs residual blocks to tackle the vanishing gradient problem, facilitating the training of significantly deeper neural networks. Notably, the architecture comprises 50 layers, which is 14 layers more than Xception.

***VGG16:*** It shares the same input image dimensions of 224 × 224 pixels as the other models. Its architecture is renowned for its simplicity and effectiveness in image classification tasks. VGG16 sequentially stacks convolutional layers interspersed with max-pooling layers. These convolutional layers use small 3 × 3 filters, which allows them to capture detailed spatial information in the input images.

**Image Feature Extraction Model Comparison:** By comparing the three feature extraction models, we found that Xception is slightly better than ResNet50 and VGG16. We ran 10 epochs and compared the loss using the extracted vectors from each model, maintaining identical settings but adjusting for variations in feature vector size:

Xception Performance

ResNet50 Performance VGG16 Performance

Upon examining the feature extraction times, we found that ResNet50 is the quickest while VGG16 is the slowest. This aligns with the expectation that ResNet50 might perform less effectively compared to the other two models, given its potentially less detailed feature extraction, as indicated by its shorter extraction duration. Additionally, considering the input size comparison, Xception necessitates a larger image size of 299 x 299 pixels compared to the other models. This larger input size enables Xception to capture more intricate details, likely resulting in enhanced performance.

### 4. CNN-RNN Model Structure

Here is the general structure of our CNN-RNN Model. The feature extractor uses Xception to extract features from the input image, the sequence processor generates the next word using an LSTM, and the decoder merges the outputs to predict the final word probabilities.

**Feature Extractor:** The feature extractor extracts relevant features from an input image using Xception. It outputs a vector embedding that captures the essential information. The extracted features are then passed through a dense layer, reducing the dimensionality to 256. This step compresses the image representation while retaining the most important aspects.

**Sequence Processor:** The sequence processor uses an embedding layer that converts the input text into dense vector representations. The image embeddings from the feature extractor are concatenated with the word embeddings. This combined representation is fed into an LSTM layer, which processes the sequence and generates the next word in the output sequence. The LSTM captures the sequential dependencies and generates coherent text based on the input image and previous words.

**Decoder:** The decoder merges the outputs from the feature extractor and the sequence processor using dense layers. The final layer has a number of nodes equal to the vocabulary size, allowing the model to predict the probability distribution over all possible words. By having a node for each word, the model generates the most likely word at each step of the output sequence.

### 5. Evaluation Metric and Results

**Incorporation of BLEU Score for Model Performance Evaluation:**

Originally, the training of our image captioning model was monitored solely through the loss metric, as conventional accuracy metrics are not suitable for the complexity of this task. To gain a more meaningful understanding of the model's performance in generating appropriate captions, we integrated BLEU score as an additional evaluation metric. It measures how the generated captions compare to a set of reference captions, which is particularly useful in natural language generation.

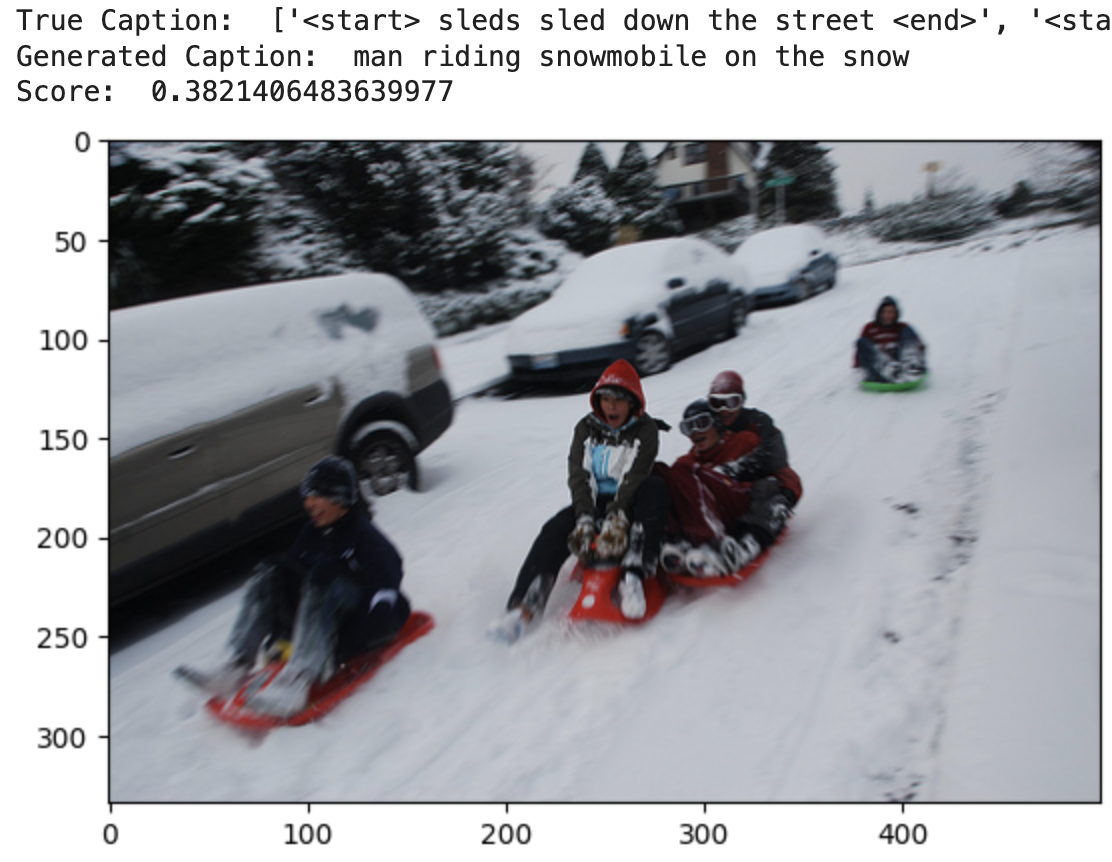
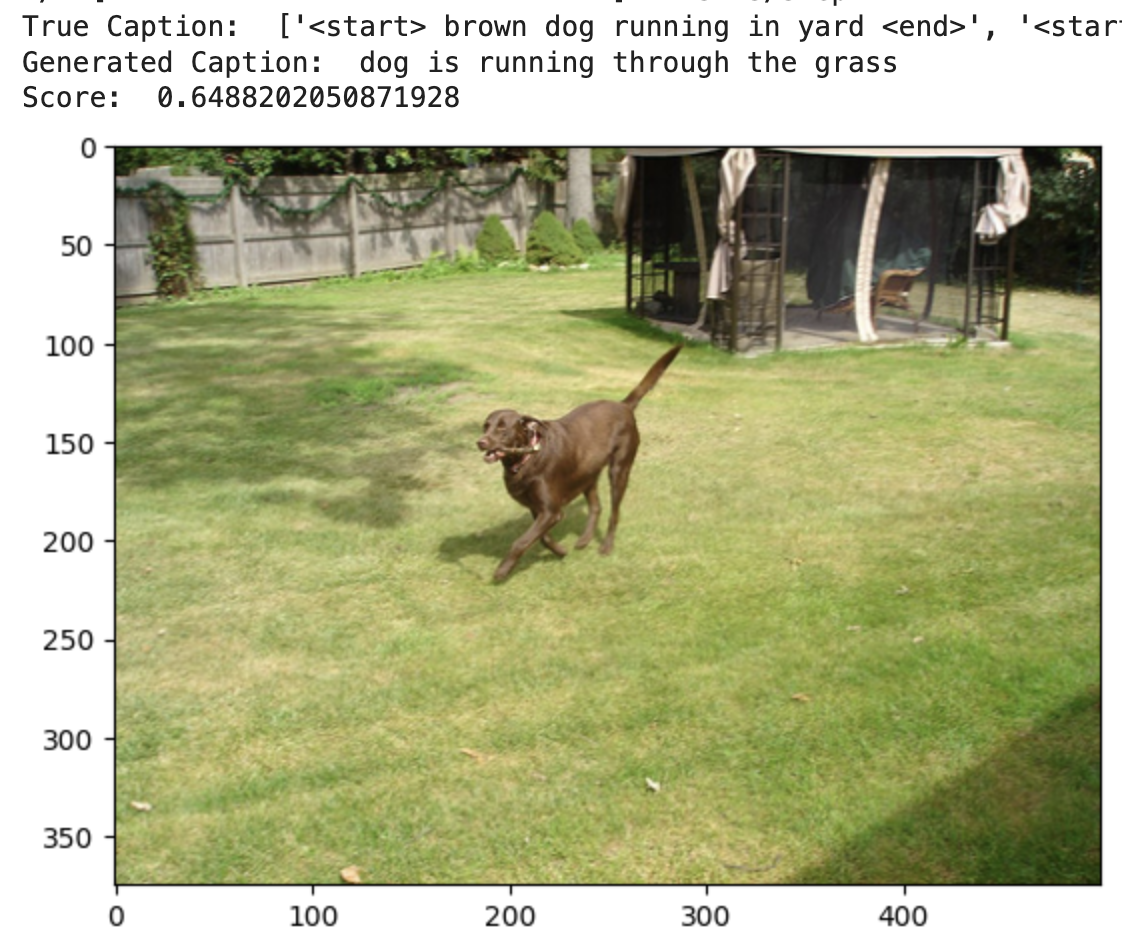
Therefore, we implemented the BLEU score as a callback metric within our training process, allowing for the evaluation of generated captions at the end of each epoch. This setup enabled us to monitor the progression of caption quality over time, aiming to directly observe improvements in linguistic accuracy and relevance as the model's parameters were optimized.

However, the practical application of this metric during training presented significant challenges. Given the substantial size of our dataset, which comprises 6,000 images, the task of generating a caption for each image and subsequently computing the BLEU score proved to be exceedingly time-consuming. This extensive processing requirement made the epoch-wise evaluation a tedious task. In our trails, we only successfully ran one epoch and obtained a BLEU score which proves that such an approach did work. This early result provided a reference on how the model could improve its BLEU score through training.



**One-time Evaluation with BLEU score:**

Given the challenges mentioned earlier, we adjusted our strategy for employing the BLEU score. Instead of applying this metric continuously during training, we use it to evaluate the performance of our model through specific demonstrations. This approach involved selecting a random sample of five images from either the training or test set. For each selected image, our model generated a caption, which was then assessed against the true reference captions using the BLEU score. This method allowed us to effectively gauge the quality of our model's output in a more controlled and less time-consuming manner, providing a focused snapshot of its capability to generate relevant and accurate captions.



### 

### 6. Lessons Learned and Next Steps

Throughout the project, we gained insights into the utilization of pretrained models. These models provided us with a strong performance baseline. We also discovered the importance of the data preprocessing phase. The quality of data proved to be as significant as its quantity, highlighting the need for meticulous data cleaning and preparation to ensure the model receives accurate and relevant information for training.

Additionally, we learned the importance of striking an optimal balance between model complexity and training efficiency. An overly complex model can lead to prolonged training times and require more computational resources, while a too-simplified model might not capture the necessary nuances of the data.

Looking forward, if given more time, we aim to further improve our model’s performance in several ways. First, we plan to enhance our data preprocessing strategies by incorporating additional computer vision techniques, which could refine our model’s input data quality. Secondly, experimenting with more feature extraction models could provide insights into achieving better accuracy and adaptability of our model. Lastly, expanding the project to include video analysis presents an exciting opportunity. Since a video is essentially a sequence of image frames, our current model could potentially be adapted to interpret and analyze video content, opening up broader applications and challenges in dynamic scene understanding.