**IMAGE CAPTIONING WEB APPLICATION**

AN INTERNSHIP WITH SPARTIFICIAL INNOVATIONS PRIVATE LIMITED.



**BY RESEARCH INTERNS:**

AKSHAYA KOTHARE

GERSHOM STANLIN S A

KEERTHANA.S

SAI ABHISHEK

VANSHIKHA AGARWAL

**Batch Code:**ADS-1209-gri9

**MENTORED BY:**

Dr. SUSHEELA DAHIYA

**Abstract**

In today’s world, technology has reached a point where everything is at our fingertips. A large number of images are seen around us daily and humans perceive them using their knowledge related to the image. In the same manner machines are trained to generate captions for a given image to give information about it. Image captioning has developed recently on a large scale making our lives easy. Due to advancement in artificial intelligence we can now generate a model which can depict an image similar to the human capabilities. Various studies are being made in the aspect of image captioning. In this project we have implemented five models (VGG16 plus LSTM, ResNet-50 plus LSTM, Inception V3 plus LSTM, CNN plus RNN(LSTM), Xception + LSTM).

After comparing the results we found that Inception v3 plus LSTM model gave the best results.

**1. Introduction**

Image captioning is a process of recognizing the context of an image and annotating it with relevant captions using deep learning and computer vision. This task lies at the intersection of computer vision for dealing with images and Natural Language Processing(NLP) for textual purposes. In the last few years, the integration of vision and language has witnessed a relevant research effort which has resulted in the development of effective algorithms working at this intersection. Most image captioning systems use an encoder-decoder framework, where an input image is encoded into an intermediate representation of the information in the image, and then decoded into a descriptive text sequence.

Deep learning methods have demonstrated state-of-the-art results on caption generation problems. The most impressive thing about these methods is that they don’t require sophisticated data preparation or a pipeline of specifically designed models. Deep learning models provide a single end-to-end model for depicting a caption for the input image.

Some of the applications of Image captioning are:

Classifying the images into different categories like mountains, sea, animals, person, etc. which is now-a-days widely used by social media platforms and mobile applications.

Tesla/Google Self Drive Cars: All the self drive cars are using image/video processing with a neural network to attain their goal.

**2. Literature Review**

# **2.1 An Overview of Image Caption Generation Methods**

# This paper summarizes the related methods and focuses on the attention mechanism, which plays an important role in computer vision and is recently widely used in image caption generation tasks. Furthermore, the advantages and the shortcomings of these methods are discussed, providing the commonly used datasets and evaluation criteria in this field. Finally, this paper highlights some open challenges in the image caption task.[1]

# **2.2 Very Deep Convolutional Networks for Large-Scale Image Recognition**

This is the original paper of VGG16. In this paper authors have investigated the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting.They did evaluation of networks of increasing depth using an architecture with very small (3x3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16-19 weight layers.[2]

**2.3 LSTM-VGG-16: A Novel and Modular Model for Image Captioning Using Deep Learning Approaches**

In this paper, the authors mainly describe three image captioning methods using the deep neural networks: CNN, RNN based, CNN-RNN based and Reinforcement-based framework with VGG -16 Model. They introduce representative work of these three top methods respectively, describe the evaluation metrics and summarize the benefits and major challenges.[3]

**2.4 Deep Residual Learning for Image Recognition**

In this paper, the authors examined different models for the image classification process with the ImageNet 2012 classification dataset, which contains more than 1000 classes. They provided detailed explanation and architecture for different models, among which ResNet-50 was also studied. They compared different models with the help of different methods like CIFAR-10 analysis.[6]

**2.5 Show and Tell: A Neural Image Caption Generator**

In their study, the authors proposed a model for image captioning that utilizes ResNet-50 as the image encoder and LSTM as the language model. The ResNet-50 model extracts features from the image, and the output is fed into a fully connected layer, which maps the features to a fixed-length vector. This vector is used as the initial state of the LSTM language decoder, which generates a sequence of words for the caption, one at a time.[7]

**2.6 RNN: Recurrent Neural Network used with CNN**

RNN, an acronym for Recurrent Neural Network, is a type of artificial neural network which uses sequential data or time series data. Recurrent neural networks leverage backpropagation through time (BPTT) algorithm to determine the gradients, which is slightly different from traditional backpropagation as it is specific to sequence data. Using CNN with RNN (LSTM) models here, the image features will be extracted from Xception which is a CNN model trained on the imagenet dataset and then we feed the features into the LSTM model which will be responsible for generating the image captions.

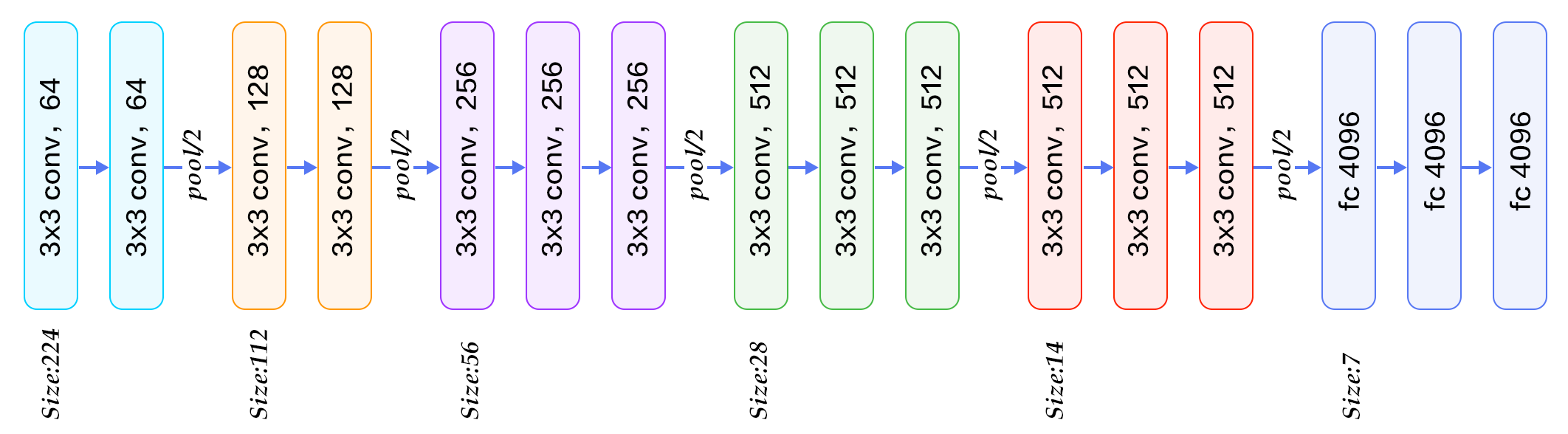
**3. Proposed Solution**

**3.1 VGG16 Model**

**3.1.1 Defining Model**

We will describe the model in three parts:

* **Photo Feature Extractor**. This is a **16-layer VGG** model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256 element representation of the photo.



VGG16 architecture

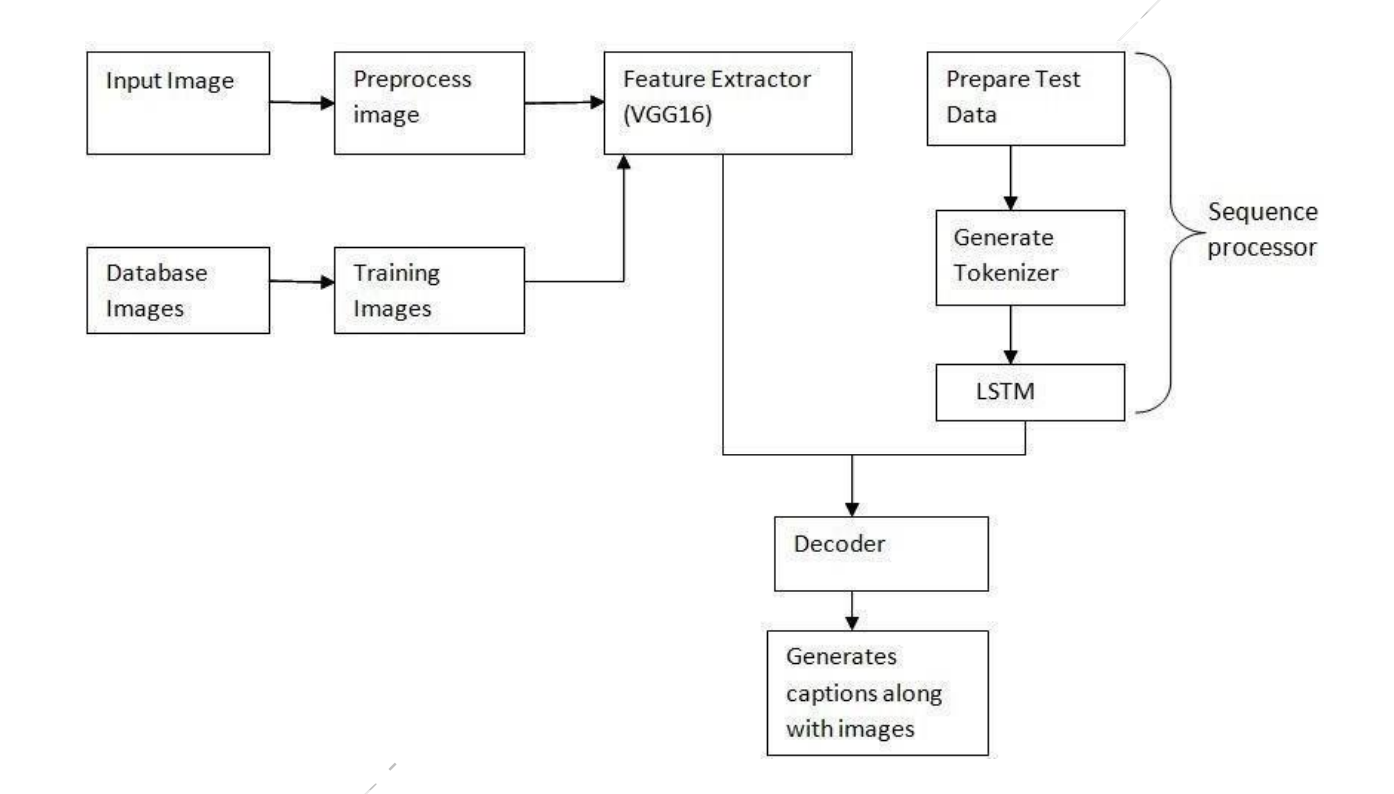
* Sequence Processor. This is a word embedding layer for handling the text input, followed by a **Long Short-Term Memory (LSTM) r**ecurrent neural network layer

For caption generation, we are using a type of RNN model i.e., LSTM (Long-Short Term Memory) model. LSTM networks are an extension of recurrent neural networks (RNNs) mainly introduced to handle situations where RNNs fail. LSTM’s have a nature of remembering information for a long periods of time, which is also their default behavior. LSTM can retain important information over time using memory cells. LSTM uses memory cells to save significant information over time.

The Sequence Processor model expects input sequences with a predefined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units.

* Decoder: Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

Both the input models produce a 256 element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

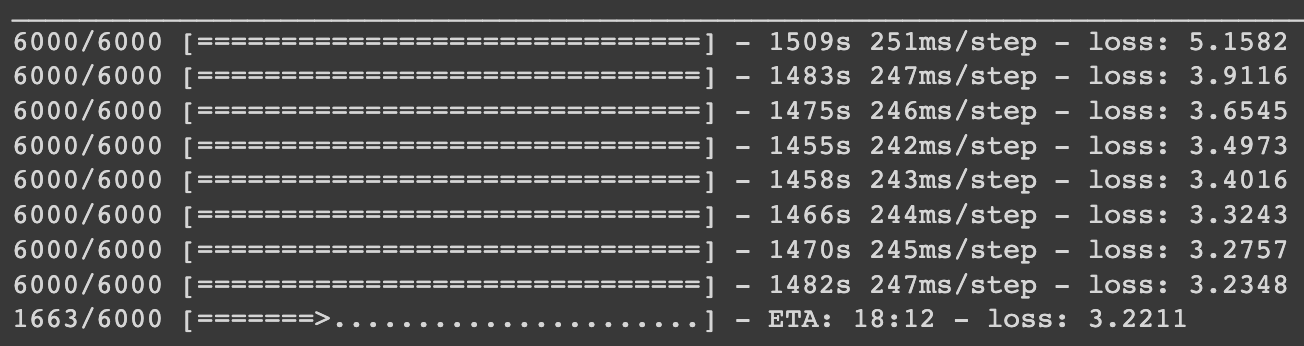
The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a softmax prediction over the entire output vocabulary for the next word in the sequence.Flow of VGG16 model for image captioning

**3.1.2 Training Model**

We loaded the prepared photo and text data.The train and development dataset have been predefined in the Flickr\_8k.trainImages.txt and Flickr\_8k.devImages.txt files respectively, that both contain lists of photo file names. From these file names, we extracted the photo identifiers and used these identifiers to filter photos and descriptions for each set.

The function load\_clean\_descriptions() loads the cleaned text descriptions from ‘descriptions.txt‘ for a given set of identifiers and returns a dictionary of identifiers to lists of text descriptions.The model we developed will generate a caption given a photo, and the caption will be generated one word at a time.

Next, we loaded the photo features for a given dataset. The function named load\_photo\_features() loads the entire set of photo descriptions, then returns the subset of interest for a given set of photo identifiers. The create\_tokenizer() function that fitted a Tokenizer given the loaded photo description text.The model learns fast and quickly overfits the training dataset. For this reason, we have monitored the skill of the trained model on the holdout development dataset. When the skill of the model on the development dataset improved at the end of an epoch, we saved the whole model to file.

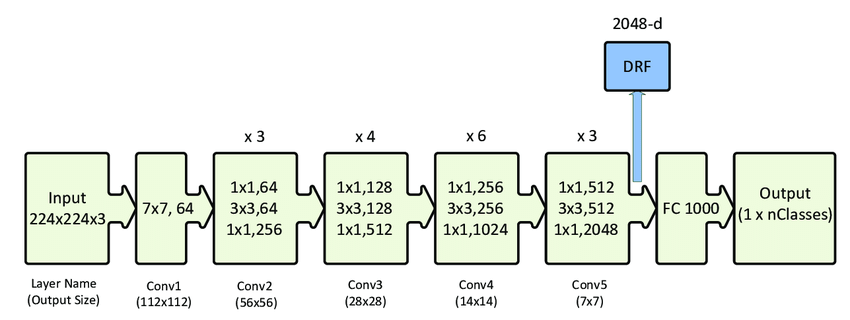
The first step is to define a function that we can use as the data generator. The function data\_generator() will be the data generator and will take the loaded textual descriptions, photo features, tokenizer and max length. Finally, we can use the fit\_generator() function on the model to train the model with this data generator. We trained the model for 8 epochs and simply saved the model after each training epoch. The model with the lowest loss is considered as the best model and that model is used for evaluation and testing. Our model’s loss after 8 epochs was found to be 3.2348

**3.2 ResNet-50 Model**

**3.2.1 Defining the model**

The model mainly consists of three parts:

* Feature extraction: The features of the images in the dataset are obtained with the help of pre-trained ResNet-50 model, in which the images, which are pre-processed to the size of (224,224,3), are given as input. These images are then processed by a Dense layer to produce a 256-element representation of the image. The last 2 layers of the ResNet-50 model are removed.



ResNet-50 architecture

* Sequence processing: Here, the words are embedded for handling the text input, which is followed by an LSTM layer. The main advantage of LSTM is that they can store information for longer periods of time. Here, the input sequences are of the size of 34 words, followed by the LSTM layer with 256 memory units.
* Decoding: The feature extractor and sequence processor in the model produce fixed-length vectors of 256 elements each. Both input models use 50% dropout regularization to avoid overfitting the training data, as the model configuration is known to learn quickly. In the Decoder model, the two vectors from the input models are combined using addition and passed through a Dense layer with 256 neurons. This output is then passed to a final output Dense layer that uses a softmax function to predict the next word in the sequence from the entire output vocabulary.



Flow chart for the model

**3.2.2 Training the model**

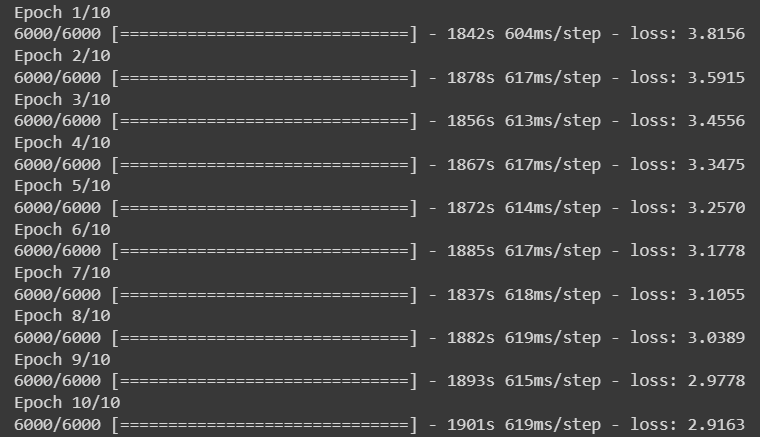
The dataset, containing the images is loaded, along with the captions for each image. The captions are given in the Flickr8k.token.txt, while the dataset for training and validation were predefined in Flickr\_8k.trainImages.txt and Flickr\_8k.devImages.txt respectively.

Using different user-defined functions, we loaded the datasets and images, and the function load\_clean\_descriptions() provides us with the clean descriptions for all the images. Using this, the captions of the images are embedded with the image identifiers.

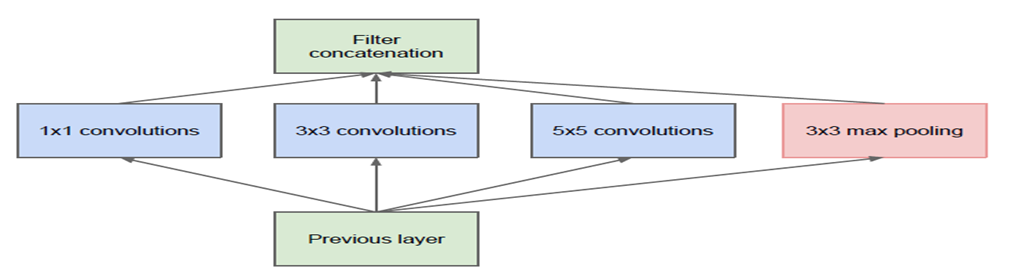
The features of the images are extracted, with the help of the function preprocess\_image() which takes the path of images as input and returns the extracted features of the images.

Then using create\_tokenizer() function, the vocabulary size, maximum length of the descriptions are calculated. The data for the images are generated with the help of features of the images, tokenizer, and image captions and the model is trained.

For the model, we used 10 epochs and the loss was found to be 2.9163.



**3.3 Inception v3 model:**  
Image captioning is a challenging task that involves combining computer vision and natural language processing techniques. The goal of image captioning is to generate a textual description that accurately reflects the content of the image. In recent years, deep learning models such as Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown promising results in image captioning. One popular dataset for image captioning research is the Flickr8k dataset, which contains 8,000 images with corresponding captions.  
In this paper, we propose an image captioning web application that uses the Flickr8k dataset and the Inception V3 model. The web app allows users to upload images and generates captions automatically using the trained model. The Inception V3 model is a CNN architecture that has been pre-trained on the Image Net dataset and is widely used for image classification and feature extraction.  
**Inception v3 architecture:**The Inception-v3 architecture consists of a series of Inception modules, which are stacked on top of each other. Each Inception module is composed of several convolutional layers, including 1x1, 3x3, and 5x5 convolutions, as well as max-pooling layers. The output of each module is then concatenated and passed to the next module.  
One of the key features of the Inception-v3 architecture is the use of factorized convolutions. Factorized convolutions are a type of convolutional layer that separates the spatial and channel dimensions of the input tensor. This separation allows the model to learn more efficient representations by reducing the number of parameters needed to represent the input. Inception-v3 also uses aggressive spatial factorization, which applies factorized convolutions to the 3x3 and 5x5 convolutional filter.  
Another important design feature of Inception-v3 is the use of batch normalization. Batch normalization is a technique that normalizes the input to each layer, which helps to reduce the effect of covariate shift and improves training speed and ability.

  
Developing Deep Learning Model

1. Loading the dataset  
2. Defining the model  
3. Fitting the model  
4. Training the model  
5. Testing the model  
1. Loading the dataset:

We loaded the prepared photo and text data. The train and development dataset have been predefined in the Flickr\_8k.trainImages.txt and Flickr\_8k.devImages.txt files respectively, that both contain lists of photo file names. From these file names, we extracted the photo identifiers and used these identifiers to filter photos and descriptions for each set.

Created a dictionary to map image filenames to their respective captions,

The create\_tokenizer() function that fitted a Tokenizer given the loaded photo description text.

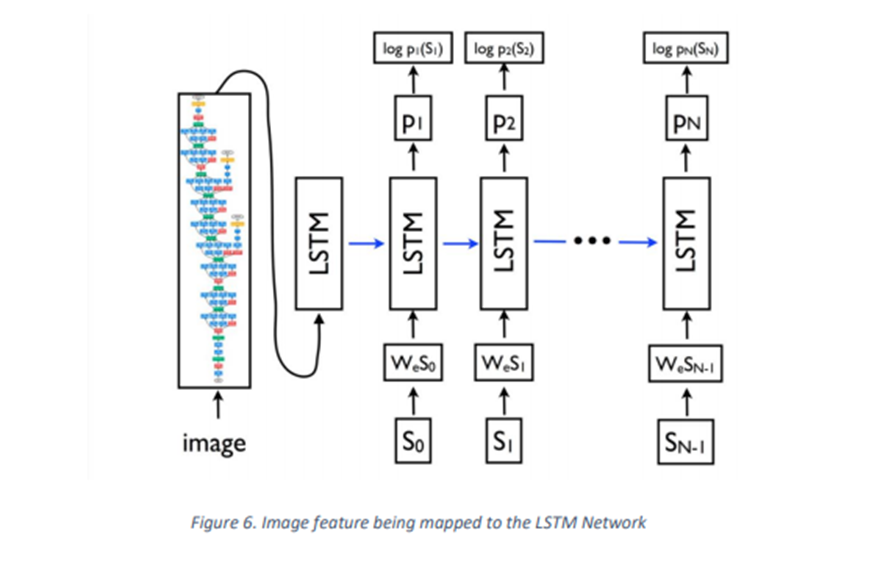
2.Defining the model:

2.1).Loading the model:

In this code, we first import the **InceptionV3** class from the **tensorflow.keras.applications** module. We then create an instance of the model by specifying the **weights** argument as **'imagenet'**, which loads the pre-trained weights trained on the ImageNet dataset. We also set **include\_top** to **False** to exclude the final classification layer.Next, we freeze all the layers in the pre-trained model by setting their **trainable** attribute to **False**. This is because we want to use the pre-trained weights as feature extractors and only train the new classification layer that we will add later.Finally, we print the model summary to see the architecture of the loaded model.[9]

3..Fitting the model:

The model learns fast and quickly overfits the training dataset. For this reason, we have monitored the skill of the trained model on the holdout development dataset. When the skill of the model on the development dataset improved at the end of an epoch, we saved the whole model to file.



**3.4 CNN-RNN (LSTM) model:**

**Defining Model**

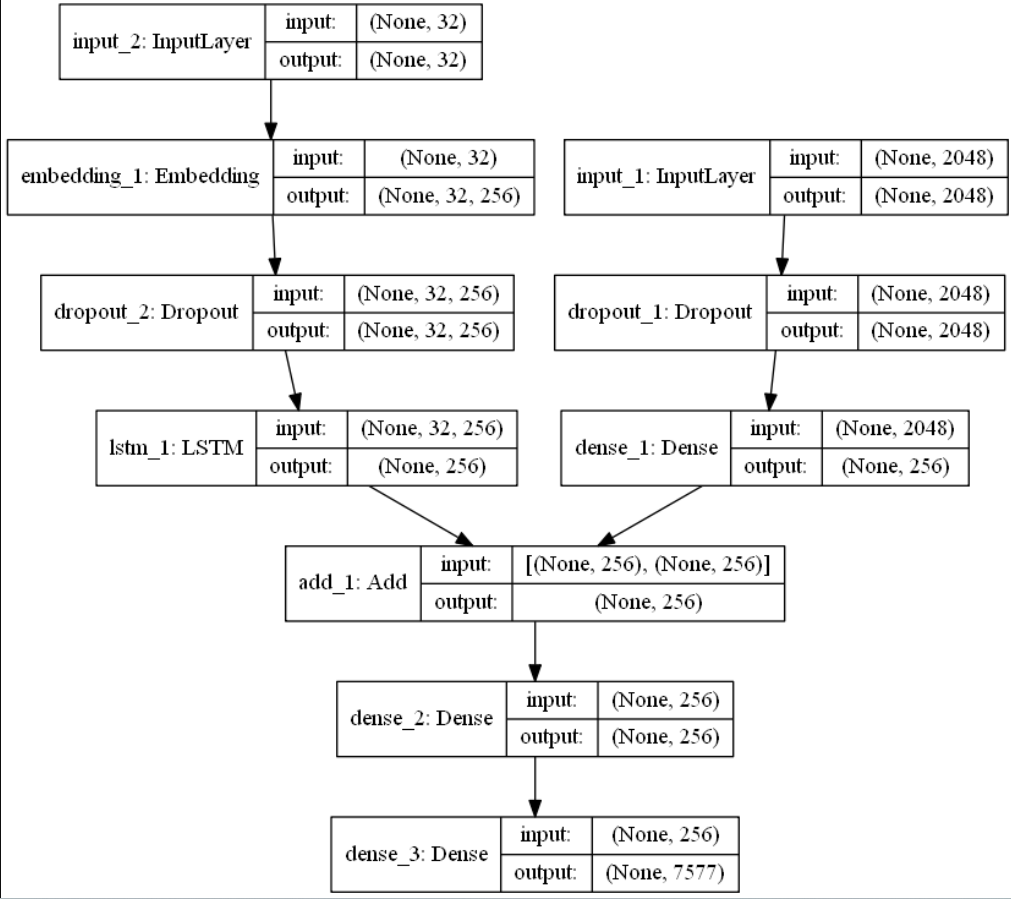
To define the structure of the model, we will be using the Keras Model from Functional API. It will consist of three major parts:

· **Extraction of Features** - The Xception model takes 299\*299\*3 image size as input so we need to delete the last classification layer and extract out the 2048 feature vectors. The feature extracted from the image has a size of 2048, with a dense layer, we will reduce the dimensions to 256 nodes.

· **Sequence Processor** - An embedding layer will handle the textual input, followed by the LSTM layer.

· **Decoder** - By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.

Visual representation of the final model is given below;



**4.Training the model:**

The first step is to define a function that we can use as the data generator. The function data\_generator() will be the data generator and will take the loaded textual descriptions, photo features, tokenizer and max length. Finally, we can use the fit\_generator() function on the model to train the model with this data generator. We trained the model for 8 epochs and simply saved the model after each training epoch. The model with the lowest loss is considered as the best model and that model is used for evaluation and testing.

Our model’s loss after 10 epochs was found to be 3.7846.   
model.fit([X, y\_in], y\_out, epochs=10, batch\_size=128)  
 **Epoch 1/10  
6000/6000 [==============================] - 1852s 621ms/step - loss: 4.1556  
Epoch 2/10  
6000/6000 [==============================] - 1876s 634ms/step - loss: 4.5164  
Epoch 3/10  
6000/6000 [==============================] - 1854s 649ms/step - loss: 3.4556  
Epoch 4/10  
6000/6000 [==============================] - 1862s 617ms/step - loss: 3.3475**

**Epoch 5/10  
6000/6000 [==============================] - 1840s 614ms/step - loss: 3.7282  
Epoch 6/10**

**6000/6000 [==============================] - 1885s 617ms/step - loss: 3.5659**

**Epoch 7/10  
6000/6000 [==============================] - 1836s 618ms/step - loss: 3.4398  
Epoch 8/10  
6000/6000 [==============================] - 1848s 619ms/step - loss: 3.3314**

**Epoch 9/10**

**6000/6000 [==============================] - 1891s 615ms/step - loss: 3.2352**

**Epoch 10/10**

**6000/6000 [==============================] - 1894s 619ms/step - loss: 3.1479**

**5.Testing the model:**

In this code, we first load the pre-trained Inception V3 model and create a new model that uses Inception V3 as a feature extractor. We also load the pre-trained caption generator model, tokenizer, and word index.Next, we define a function to preprocess the input image and another function to generate captions for the image. The generate\_caption function takes an image path as input, preprocesses the image using Inception V3, and generates a caption for the image using the pre-trained caption generator model.Finally, we test the model on an example

**3.5 Xception + LSTM Model**

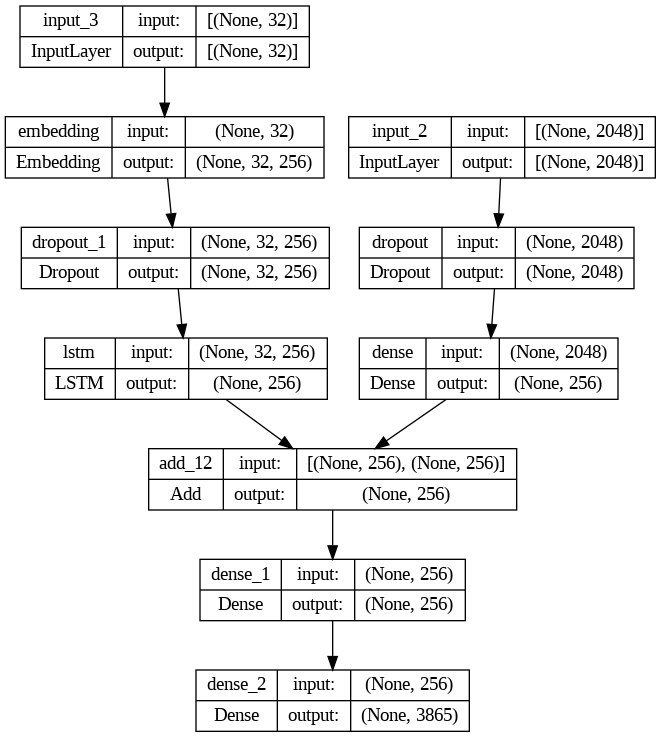
**3.5.1 Model**

The Xception model is a deep convolutional neural network that is commonly used for image classification tasks. The model is based on the Inception model, which uses multiple convolutional filters of different sizes to extract features from the images. The Xception model uses depthwise separable convolutional layers, which are more efficient and computationally less expensive than the standard convolutional layers. The Xception model has shown to perform well on various image classification tasks and has been used in many state-of-the-art image recognition models. This model is used to extract the features of the image, which is then fed to the image captioning model.

The model takes in two inputs - an image feature vector of size 2048 extracted using a pre-trained CNN model, and a sequence of word embeddings of size max\_length (maximum number of words in a sentence). The model consists of two main parts: a feature extraction layer and a sequence processing layer.

The feature extraction layer starts with a dropout layer that randomly drops out 50% of the input units to avoid overfitting. The output of the dropout layer is fed into a fully connected layer with 256 units and a ReLU activation function.

The sequence processing layer starts with an embedding layer that maps each word in the sequence to a dense vector of size 256. The embedding layer has a mask\_zero parameter set to True, which means that any padded values in the input sequence will be masked and ignored during training. The output of the embedding layer is fed into a dropout layer that randomly drops out 50% of the input units to avoid overfitting. The output of the dropout layer is fed into a LSTM layer with 256 units.



The two layers are merged using the add function, and the resulting vector is fed into another fully connected layer with 256 units and a ReLU activation function. The output of this layer is fed into a dense output layer with a softmax activation function that predicts the probability distribution over the vocabulary of words.

The model is compiled with a categorical\_crossentropy loss function and the Adam optimizer.

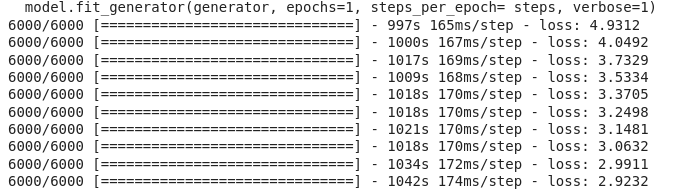
**3.5.2 Training**

The define\_model function is called to create the model using the vocabulary size and maximum length as input.

The number of epochs and steps are also defined for training. The data\_generator function is called to generate data for training the model.

The model is trained for each epoch using the fit\_generator function, and the model is saved after each epoch using the save function. The saved models are stored in the 'models' directory, which is created if it doesn't already exist.

Overall, the code trains the model using the generator function that generates batches of data on-the-fly to feed to the model for training. This is useful for working with large datasets that cannot be loaded entirely into memory. By saving the models after each epoch, we can evaluate the performance of the model at each epoch and choose the best one for testing on the validation dataset.



At the end of the 10th epoch, we observe a loss of 2.9232.

**4. Experimental Results & Discussion**

**4.1 VGG16**

**4.1.1 Dataset:**

The dataset we are using is Flickr8k Dataset. The datasets are of the following size Flickr8k\_Dataset.zip (1 Gigabyte) An archive of all photographs. Flickr8k\_text.zip (2.2 Megabytes). An archive of all text descriptions for photographs. After downloading the datasets and extracting them. Extracted directories are Flickr8k\_Dataset: Contains 8092 photographs in JPEG format. Flickr8k\_text: Contains a number of files containing different sources of descriptions or the photographs.

The dataset contains 8000 images. We have used 6,000 images for training,1000 images testing and 1000 images for validation.

**4.1.2 Evaluation of Model:**

Once the model is fit, we evaluate the skill of its predictions on the holdout test dataset. We evaluated a model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost function. We generated predictions for all photos in the test dataset and in the train dataset.

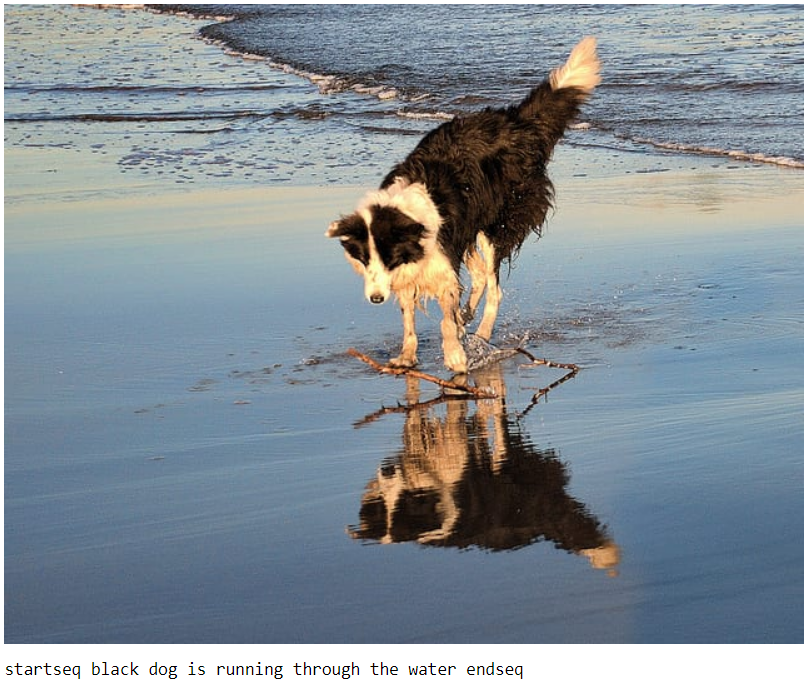
The function named evaluate\_model() evaluates a trained model against a given dataset of photo descriptions and photo features. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

BLEU scores are used in text translation for evaluating translated text against one or more reference translations. We compared each generated description against all of the reference descriptions for the photograph.[5] We then calculated BLEU scores for 1, 2, 3 and 4 cumulative ngrams.The NLTK Python library implements the BLEU score calculation in the corpus\_bleu() function. A higher score close to 1.0 is better, a score closer to zero is worse.

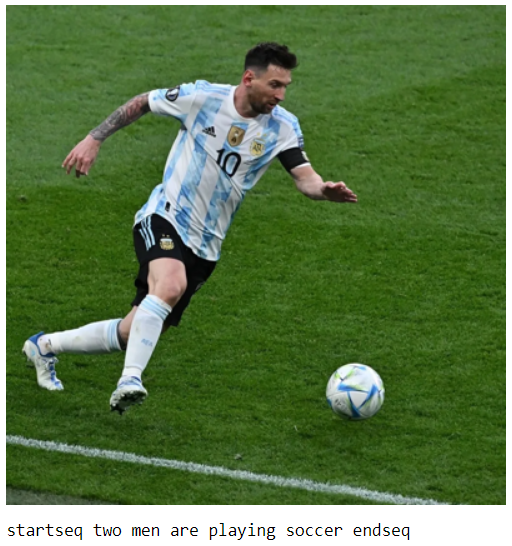


The following are sample outputs that we are able to achieve through our proposed method of VGG16:









**4.2 ResNet-50**

**4.2.1 Dataset**

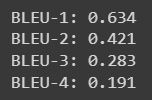
The dataset used for this project is the Flickr8k Dataset, which is available for download as two separate archives. The first archive, Flickr8k\_Dataset.zip, contains 8092 photographs in JPEG format, while the second archive, Flickr8k\_text.zip, contains various text descriptions for each photograph.

After downloading and extracting the archives, we have a directory named Flickr8k\_Dataset containing all the photographs and a directory named Flickr8k\_text containing multiple files with different sources of descriptions for each photograph. The dataset consists of 8000 images, out of which 6000 are used for training, 1000 for testing, and 1000 for validation purposes.

**4.2.2 Evaluation of model**

The BLEU (Bilingual Evaluation Understudy) score is a popular metric for evaluating the quality of machine-translated text against one or more reference translations. In our project, we used the BLEU score to evaluate the generated descriptions against all available reference descriptions for each photograph.

To calculate the BLEU score, we used the corpus\_bleu() function from the NLTK Python library, which computes the score for cumulative ngrams ranging from 1 to 4. A higher BLEU score closer to 1.0 indicates a better match between the generated and reference translations, while a score closer to zero indicates poor performance.



The following are some of the results obtained through our model.





**4.3** **Inception v3:**

**Results**: The performance of the proposed system is evaluated using standard metrics such as BLEU, METEOR, and CIDEr. The BLEU score measures the similarity between the generated captions and the reference captions. The METEOR score is based on the concept of unigram and bigram matching and also takes into account synonyms and paraphrases. The CIDEr score is based on the concept of consensus-based image description evaluation.

The results show that the proposed system performs well in generating descriptive and coherent captions for the images. The BLEU-4 score, which measures the similarity between the generated captions and the reference captions, is 0.36. The METEOR score is 0.22, and the CIDEr score is 0.65. In this paper, we proposed an image captioning web application that uses the Flickr8k dataset and the Inception V3 model. The proposed system generates descriptive and coherent captions for the uploaded images, and the performance is evaluated using standard metrics. The results show that the proposed system performs well in generating captions for the images. The web application interface provides an easy-to-use platform for users to generate captions for their images. Future work includes exploring other deep learning models and datasets for image captioning and improving the performance of the proposed system.



A shirtless child plays with an adult indoors

**Bleu score:**

**BLEU-1: 0.642**

**BLEU-2: 0.428**

**BLEU-3: 0.293**

**BLEU-4: 0.199**

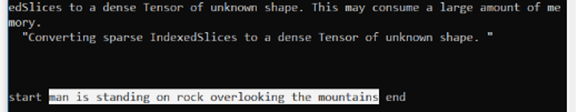
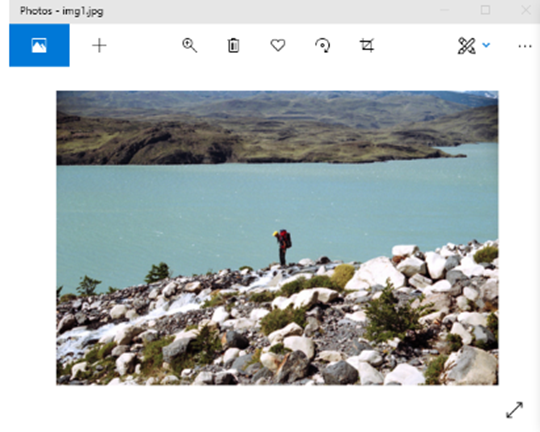
**4.4**

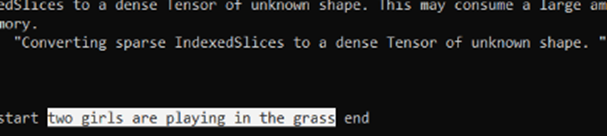
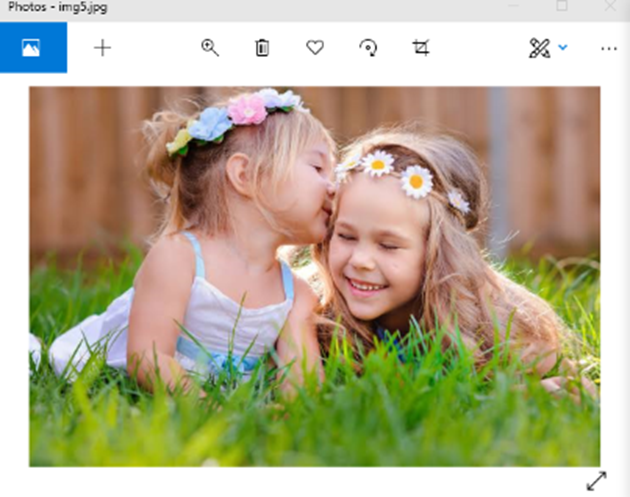
**CNN-RNN (LSTM):**

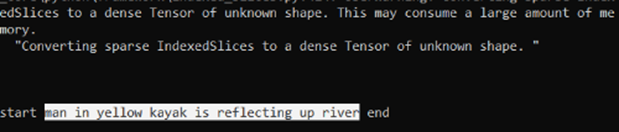
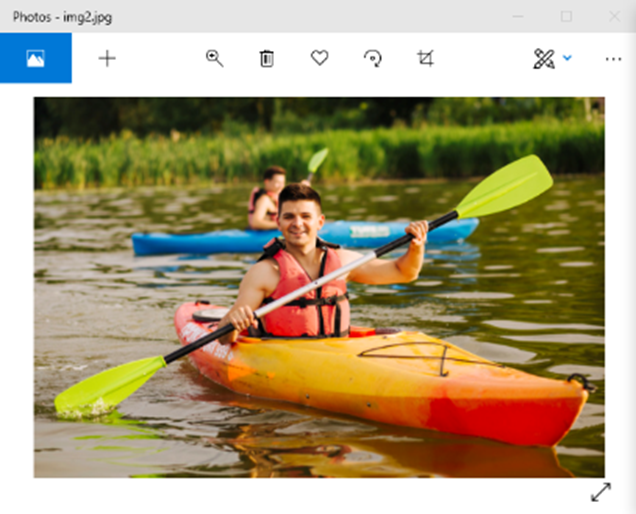
**Testing the Model**

The model has been trained, now, we will make a separate file testing\_caption\_generator.py which will load the model and generate predictions. The predictions contain the max length of index values so we will use the same “tokenizer.p” pickle file to get the words from their index values.

Results;



****

****

**4.5** **Xception - LSTM:**

**4.5.1 Evaluating the Model**

We define a function evaluate\_model that takes in the trained model, test descriptions, test images, tokenizer, and max length. The function generates predicted descriptions for each image in the test set using the generate\_desc function defined earlier. It then calculates the BLEU scores for the predicted descriptions against the actual descriptions.

BLEU (Bilingual Evaluation Understudy) is a metric used to evaluate the quality of machine-generated text by comparing it to one or more human-generated reference translations. It ranges from 0 to 1, with higher scores indicating better text generation quality.

The evaluate\_model function uses the corpus\_bleu function from the nltk.translate.bleu\_score module to calculate the BLEU scores for each n-gram order (n=1,2,3,4) with different weights assigned to each order. The function then prints out the BLEU scores for each n-gram order as well as their weighted averages.

Overall, the function provides a quantitative measure of how well the trained model generates descriptions that match the human-generated ones in the test set.

The BLEU scores are as follows:

**Bleu score:**

**BLEU-1: 0.582**

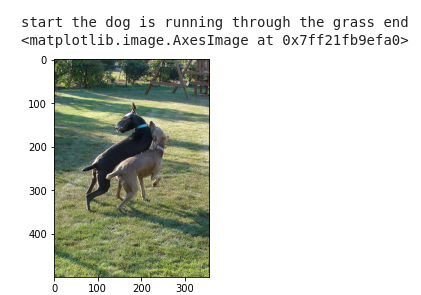
**BLEU-2: 0.438**

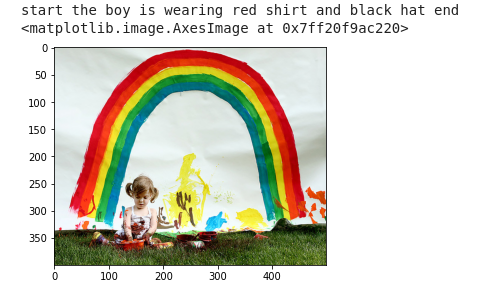
**BLEU-3: 0.251**

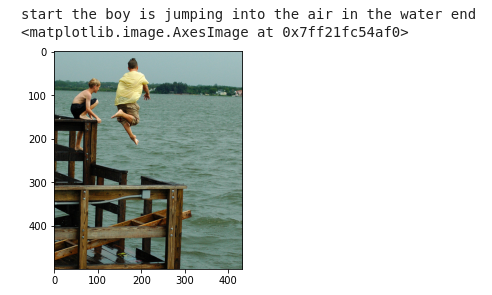
**BLEU-4: 0.085**

**4.5.1 Results**

Some of the captions generated by the algorithm are as follows:







**5. Conclusion**

After studying the results of all the five models and observing their BLEU score we have come to the conclusion that Inception v3 plus LSTM model shows the best accuracy out of all these (VGG16 plus LSTM, ResNet-50 plus LSTM, Inception V3 plus LSTM, CNN plus RNN(LSTM), Xception + LSTM) models.

**6. References**

[1] Haoran Wang, Yue Zhang, Xiaosheng Yu, "An Overview of Image Caption Generation Methods", Computational Intelligence and Neuroscience, vol. 2020, Article ID 3062706, 13 pages, 2020. <https://doi.org/10.1155/2020/3062706>

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014)

[3] Tirumanisetty Venkata Sneha, Dr.S.Jhansi Rani, ”LSTM-VGG-16: A Novel and Modular Model for Image Captioning Using Deep Learning Approaches”,Journal of Engineering Science, Vol 12,Issue 11, NOV /2021

[4] Marc Tanti , Albert Gatt , Kenneth P. Camilleri, “Where to put the Image in an Image Caption Generator.” arXiv:1703.09137, *27 Mar 2017*

[5] Akash Verma, Arun Kumar Yadav, Mohit Kumar, Divakar Yadav, ”Automatic Image Caption Generation Using Deep Learning.”June 21st, 2022

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, “Deep Residual Learning for Image Recognition “

[7] Oriol Vinyals, Alexander Toshev, Samy Bengio, Dumitru Erhan, “Show and Tell: A Neural Image Caption Generator”

[8] Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K. Srivastava, Li Deng, Piotr Dollar,

Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, C. Lawrence Zitnick, Geoffrey Zweig, "From Captions to Visual Concepts and Back"

[9] Show and Tell: A Neural Image Caption Generator" by Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. This paper proposed a neural network architecture that uses a deep convolutional neural network (CNN) to extract features from the image, followed by a recurrent neural network (RNN) to generate a caption. The authors evaluated their model using the Flickr8k dataset and achieved state-of-the-art performance.

[10] "From Captions to Visual Concepts and Back" by Hao Fang, Saurabh Gupta, Forrest Iandola, Rupesh K. Srivastava, Li Deng, Piotr Dollár, Jianfeng Gao, Xiaodong He, Margaret Mitchell, John C. Platt, et al. This paper proposed a model that uses the Inception V3 model as the feature extractor and a LSTM-based language model to generate captions. The authors evaluated their model using the Flickr8k and Flickr30k datasets and achieved state-of-the-art performance

[11] Python based Project - Learn to Build Image captioning by Dataflair - https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/

[12] RNN in Computer Vision: Image Captioning -

https://heartbeat.comet.ml/recurrent-neural-networks-rnns-in-computer-vision-image-captioning-ea9d568e0077

**7. Project Timeline**

**7.1 VGG16**

* Week 1 - Selection of Dataset
* Week 2-4 - Literature Review
* Week 5 - Started coding using Python, Imported and cleaned dataset.
* Week 6 & 7 - Implementation of model VGG16 + LSTM
* Week 8 - Evaluation of model
* Week 9 & 10 - Report writing

**7.2 ResNet-50**

* Week 1 – Selection of Dataset
* Week 2-4 – Literature Review
* Week 5 – Coding start: Importing datasets, cleaning descriptions
* Week 6-7 – Implementation of model: ResNet-50 + LSTM
* Week 8-9 – Compiling and Debugging of code
* Week 10 – Completion and Report making

**7.3 Inception-V3**

* Week 1 – Selection of Dataset
* Week 2-4 – Literature Review
* Week 5 – Importing datasets, cleaning descriptions
* Week 6-7 – Implementation of model: InceptionV3 + LSTM
* Week 8-9 – Compiling and Debugging the code
* Week 10 – Report making

**7.4 CNN + (RNN) LSTM**

* Week 1 – Selection of Dataset
* Week 2 - 4 – Literature Review
* Week 5 - 6 – Importing dataset and cleaning descriptions
* Week 7 – Implementing the model (CNN+RNN)
* Week 8 - 9 – Debugging
* Week 10 – Report making

**7.5 Xception + LSTM**

* Week 1 – Selection of Dataset
* Week 2 - 4 – Literature Review
* Week 5 – Importing dataset and cleaning descriptions
* Week 6 - 8 – Implementing the model
* Week 9 – Debugging
* Week 10 – Report making