

```
!pip install -q kaggle
```

```
from google.colab import files  
files.upload()
```

kaggle.json

- **kaggle.json**(application/json) - 68 bytes, last modified: 1/19/2023 - 100% done
Saving kaggle.json to kaggle.json
{'kaggle.json':
b'{"username": "carhinlaykan" "key": "A2h667h756a3Ae7h125d5eac8f48h44" }' }

```
! mkdir ~/.kaggle
```

```
! cp kaggle.json ~/.kaggle/
```

```
! chmod 600 ~/.kaggle/kaggle.json
```

```
!kaggle datasets download -d adityajn105/flickr8k
```

```
Downloading flickr8k.zip to /content  
99% 1.02G/1.04G [00:05<00:00, 274MB/s]  
100% 1.04G/1.04G [00:05<00:00, 204MB/s]
```

```
!unzip flickr8k.zip
```

```
initiating: images/97751718_e070a11tu3.jpg
inflating: Images/977856234_0d9caee7b2.jpg
inflating: Images/978580450_e862715aba.jpg
inflating: Images/979201222_75b6456d34.jpg
inflating: Images/979383193_0a542a059d.jpg
inflating: Images/98377566_e4674d1ehd.jpg

import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import Sequence
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Activation, Dropout, Flatten, Dense, Input, Layer
from tensorflow.keras.layers import Embedding, LSTM, add, Concatenate, Reshape, concatenate, Bidirectional
from tensorflow.keras.applications import VGG16, ResNet50, DenseNet201
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from textwrap import wrap

plt.rcParams['font.size'] = 12
sns.set_style("dark")
warnings.filterwarnings('ignore')
```

imgPath = '../content/Images'

data = pd.read_csv("../content/captions.txt")
data.head()

	image	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

```
def readIMG(path,image_size=224):
    img = load_img(path,color_mode='rgb',target_size=(image_size,image_size))
    img = img_to_array(img)
    img = img/255.

    return img

def displayImgs(temp_df):
    temp_df = temp_df.reset_index(drop=True)
    plt.figure(figsize = (20 , 20))
    n = 0
    for i in range(15):
        n+=1
        plt.subplot(5 , 5, n)
        plt.subplots_adjust(hspace = 0.7, wspace = 0.3)
        image = readIMG(f"../content/Images/{temp_df.image[i]}")
        plt.imshow(image)
        plt.title("\n".join(wrap(temp_df.caption[i], 20)))
        plt.axis("off")
```

Visualization

displayImgs(data.sample(15))

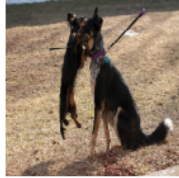
A black and brown dog walking in deep snow .



A child in a red shirt is sitting against a wall with a map while eating .



A large dog holding a smaller dog .



a black dog digging through the snow .



Two black and one white dog interacting in the grass .



Football players are on the field .



Sooners football player number 50 plays in a game .



blonde girl wearing green dress standing



A little boy walks around with a white t-shirt and black shorts outside .



A man is sitting on a bench , cooking some food .



a brown and white dog jumping over blue and white poles



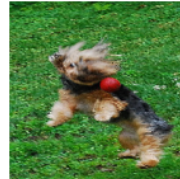
Many people hang out in a body of water .



a young woman wearing a purple hat with a pink feather in it



The dog jumps up to catch the red ball .



A woman in a black and white dress is holding up a protest sign .



Caption Text Preprocessing Steps

- Convert sentences into lowercase
- Remove special characters and numbers present in the text
- Remove extra spaces
- Remove single characters
- Add a starting and an ending tag to the sentences to indicate the beginning and the ending of a sentence

```
def txtPreProcessing(data):
    data['caption'] = data['caption'].apply(lambda x: x.lower())
    data['caption'] = data['caption'].apply(lambda x: x.replace("[^A-Za-z]", ""))
    data['caption'] = data['caption'].apply(lambda x: x.replace("\s+", " "))
    data['caption'] = data['caption'].apply(lambda x: " ".join([word for word in x.split() if len(word)>1]))
    data['caption'] = "startseq "+data['caption']+" endseq"
    return data
```

Preprocessed Text

```
data = txtPreProcessing(data)
imgCaptions = data['caption'].tolist()
imgCaptions[:10]
```

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tri-colored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the street endseq',
 'startseq two dogs of different breeds looking at each other on the road endseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

▼ Tokenization and Encoded Representation

- The words in a sentence are separated/tokenized and encoded in a one hot representation
- These encodings are then passed to the embeddings layer to generate word embeddings

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(imgCaptions)
vocabulary_length = len(tokenizer.word_index) + 1
maximum_len = max(len(caption.split()) for caption in imgCaptions)

images = data['image'].unique().tolist()
nimgs = len(images)

split_index = round(0.85*nimgs)
trainImgs = images[:split_index]
valImgs = images[split_index:]

train = data[data['image'].isin(trainImgs)]
test = data[data['image'].isin(valImgs)]

train.reset_index(inplace=True,drop=True)
test.reset_index(inplace=True,drop=True)

tokenizer.texts_to_sequences([imgCaptions[1]])[0]

[1, 18, 315, 63, 195, 116, 2]
```

▼ Image Feature Extraction

- DenseNet 201 Architecture is used to extract the features from the images
- Any other pretrained architecture can also be used for extracting features from these images
- Since the Global Average Pooling layer is selected as the final layer of the DenseNet201 model for our feature extraction, our image embeddings will be a vector of size 1920

```
model = DenseNet201()
fe = Model(inputs=model.input, outputs=model.layers[-2].output)

image_size = 224
features = {}
for image in tqdm(data['image'].unique().tolist()):
    img = load_img(os.path.join(imgPath,image),target_size=(image_size,image_size))
    img = img_to_array(img)
    img = img/255.
    img = np.expand_dims(img,axis=0)
    feature = fe.predict(img, verbose=0)
    features[image] = feature
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet201_weights_tf_dim_ordering_tf_kernels_82524592/82524592 [=====] - 0s 0us/step
100%|██████████| 8091/8091 [13:28<00:00, 10.00it/s]

▼ Data Generation

- Since Image Caption model training like any other neural network training is a highly resource utilizing process we cannot load the data into the main memory all at once, and hence we need to generate the data in the required format batch wise
- The inputs will be the image embeddings and their corresponding caption text embeddings for the training process
- The text embeddings are passed word by word for the caption generation during inference time

```
class CustomDataGeneration(Sequence):

    def __init__(self, df, X_col, y_col, batch_size, directory, tokenizer,
                 vocabulary_length, maximum_len, features,shuffle=True):

        self.df = df.copy()
        self.X_col = X_col
        self.y_col = y_col
```

```

self.directory = directory
self.batch_size = batch_size
self.tokenizer = tokenizer
self.vocabulary_length = vocabulary_length
self.maximum_len = maximum_len
self.features = features
self.shuffle = shuffle
self.n = len(self.df)

def epoch_ending(self):
    if self.shuffle:
        self.df = self.df.sample(frac=1).reset_index(drop=True)

def __len__(self):
    return self.n // self.batch_size

def __getitem__(self, index):

    batch = self.df.iloc[index * self.batch_size:(index + 1) * self.batch_size,:]
    X1, X2, y = self.__getData(batch)
    return (X1, X2), y

def __getData(self, batch):

    X1, X2, y = list(), list(), list()

    images = batch[self.X_col].tolist()

    for image in images:
        feature = self.features[image][0]

        imgCaptions = batch.loc[batch[self.X_col]==image, self.y_col].tolist()
        for caption in imgCaptions:
            seq = self.tokenizer.texts_to_sequences([caption])[0]

            for i in range(1, len(seq)):
                in_seq, out_seq = seq[:i], seq[i]
                in_seq = pad_sequences([in_seq], maxlen=self.maximum_len)[0]
                out_seq = to_categorical([out_seq], num_classes=self.vocabulary_length)[0]
                X1.append(feature)
                X2.append(in_seq)
                y.append(out_seq)

    X1, X2, y = np.array(X1), np.array(X2), np.array(y)

    return X1, X2, y

```

▼ Modelling

- The image embedding representations are concatenated with the first word of sentence ie. starseq and passed to the LSTM network
- The LSTM network starts generating words after each input thus forming a sentence at the end

```

input1 = Input(shape=(1920,))
input2 = Input(shape=(maximum_len,))

img_features = Dense(256, activation='relu')(input1)
img_features_reshap = Reshape((1, 256), input_shape=(256,))(img_features)

sentenceFeatures = Embedding(vocabulary_length, 256, mask_zero=False)(input2)
merged = concatenate([img_features_reshap, sentenceFeatures], axis=1)
sentenceFeatures = LSTM(256)(merged)
x = Dropout(0.5)(sentenceFeatures)
x = add([x, img_features])
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(vocabulary_length, activation='softmax')(x)

caption_model = Model(inputs=[input1, input2], outputs=output)
caption_model.compile(loss='categorical_crossentropy', optimizer='adam')

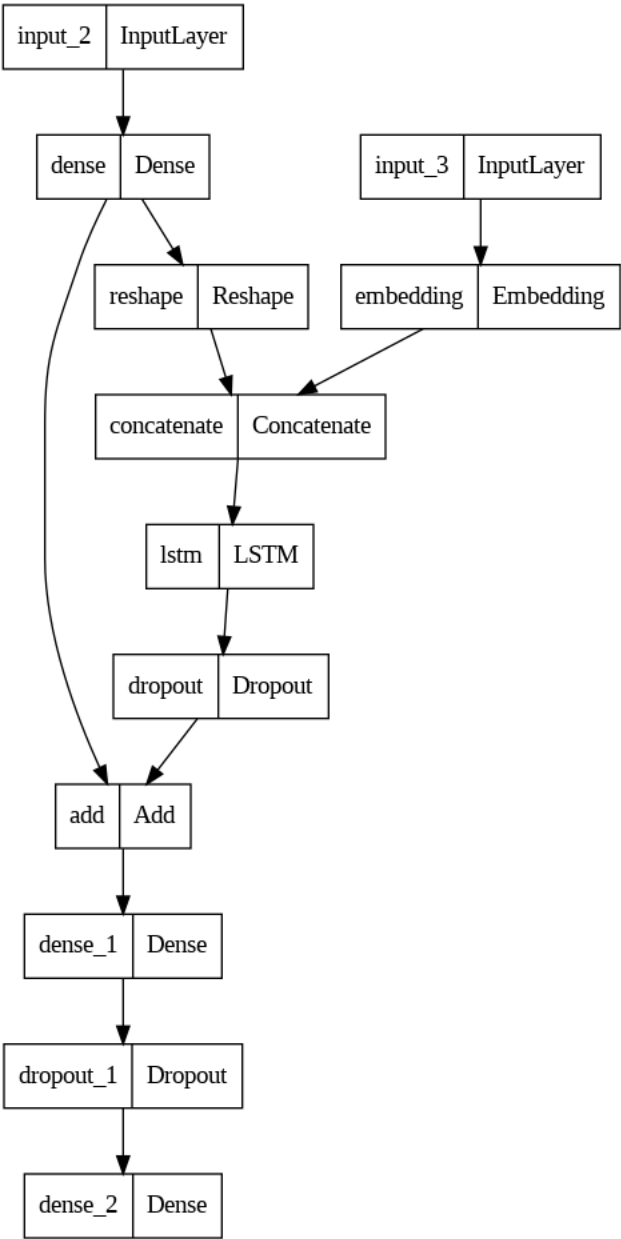
from tensorflow.keras.utils import plot_model

```

▼ Model Modification

- A slight change has been made in the original model architecture to push the performance. The image feature embeddings are added to the output of the LSTMs and then passed on to the fully connected layers
- This slightly improves the performance of the model originally proposed back in 2014: **Show and Tell: A Neural Image Caption Generator** (<https://arxiv.org/pdf/1411.4555.pdf>)

```
plot_model(caption_model)
```



```
caption_model.summary()
```

Model: "model_1"

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 1920)]	0	[]
dense (Dense)	(None, 256)	491776	['input_2[0][0]']
input_3 (InputLayer)	[(None, 34)]	0	[]
reshape (Reshape)	(None, 1, 256)	0	['dense[0][0]']
embedding (Embedding)	(None, 34, 256)	2172160	['input_3[0][0]']

concatenate (Concatenate)	(None, 35, 256)	0	['reshape[0][0]', 'embedding[0][0]']
lstm (LSTM)	(None, 256)	525312	['concatenate[0][0]']
dropout (Dropout)	(None, 256)	0	['lstm[0][0]']
add (Add)	(None, 256)	0	['dropout[0][0]', 'dense[0][0]']
dense_1 (Dense)	(None, 128)	32896	['add[0][0]']
dropout_1 (Dropout)	(None, 128)	0	['dense_1[0][0]']
dense_2 (Dense)	(None, 8485)	1094565	['dropout_1[0][0]']

```

=====
Total params: 4,316,709
Trainable params: 4,316,709
Non-trainable params: 0

```

```

train_generator = CustomDataGeneration(df=train,X_col='image',y_col='caption',batch_size=64,directory=imgPath,
                                       tokenizer=tokenizer,vocabulary_length=vocabulary_length,maximum_len=maximum_len,features=features)

validation_generator = CustomDataGeneration(df=test,X_col='image',y_col='caption',batch_size=64,directory=imgPath,
                                           tokenizer=tokenizer,vocabulary_length=vocabulary_length,maximum_len=maximum_len,features=features)

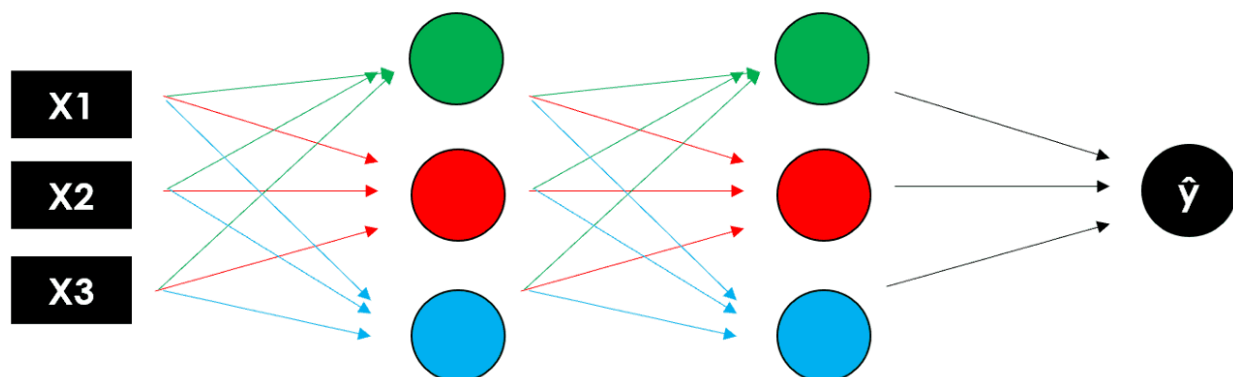
model_name = "model.h5"
checkpoint = ModelCheckpoint(model_name,
                             monitor="val_loss",
                             mode="min",
                             save_best_only = True,
                             verbose=1)

earlystopping = EarlyStopping(monitor='val_loss',min_delta = 0, patience = 5, verbose = 1, restore_best_weights=True)

learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
                                             patience=3,
                                             verbose=1,
                                             factor=0.2,
                                             min_lr=0.00000001)

```

▼ Let's train the Model !



```

history = caption_model.fit(
    train_generator,
    epochs=50,
    validation_data=validation_generator,
    callbacks=[checkpoint,earlystopping,learning_rate_reduction])

```

```

Epoch 1/50
537/537 [=====] - ETA: 0s - loss: 5.1291
Epoch 1: val_loss improved from inf to 4.23636, saving model to model.h5
537/537 [=====] - 223s 410ms/step - loss: 5.1291 - val_loss: 4.2364 - lr: 0.0010
Epoch 2/50
537/537 [=====] - ETA: 0s - loss: 4.1916
Epoch 2: val_loss improved from 4.23636 to 3.91243, saving model to model.h5
537/537 [=====] - 222s 413ms/step - loss: 4.1916 - val_loss: 3.9124 - lr: 0.0010
Epoch 3/50
537/537 [=====] - ETA: 0s - loss: 3.8843
Epoch 3: val_loss improved from 3.91243 to 3.74151, saving model to model.h5
537/537 [=====] - 219s 408ms/step - loss: 3.8843 - val_loss: 3.7415 - lr: 0.0010
Epoch 4/50
537/537 [=====] - ETA: 0s - loss: 3.7009
Epoch 4: val_loss improved from 3.74151 to 3.66385, saving model to model.h5
537/537 [=====] - 218s 405ms/step - loss: 3.7009 - val_loss: 3.6638 - lr: 0.0010
Epoch 5/50
537/537 [=====] - ETA: 0s - loss: 3.5696
Epoch 5: val_loss improved from 3.66385 to 3.63808, saving model to model.h5
537/537 [=====] - 218s 407ms/step - loss: 3.5696 - val_loss: 3.6381 - lr: 0.0010
Epoch 6/50
537/537 [=====] - ETA: 0s - loss: 3.4677
Epoch 6: val_loss improved from 3.63808 to 3.60417, saving model to model.h5
537/537 [=====] - 219s 407ms/step - loss: 3.4677 - val_loss: 3.6042 - lr: 0.0010
Epoch 7/50
537/537 [=====] - ETA: 0s - loss: 3.3833
Epoch 7: val_loss improved from 3.60417 to 3.60118, saving model to model.h5
537/537 [=====] - 218s 405ms/step - loss: 3.3833 - val_loss: 3.6012 - lr: 0.0010
Epoch 8/50
537/537 [=====] - ETA: 0s - loss: 3.3099
Epoch 8: val_loss did not improve from 3.60118
537/537 [=====] - 219s 408ms/step - loss: 3.3099 - val_loss: 3.6057 - lr: 0.0010
Epoch 9/50
537/537 [=====] - ETA: 0s - loss: 3.2462
Epoch 9: val_loss did not improve from 3.60118
537/537 [=====] - 217s 404ms/step - loss: 3.2462 - val_loss: 3.6114 - lr: 0.0010
Epoch 10/50
537/537 [=====] - ETA: 0s - loss: 3.1921
Epoch 10: val_loss did not improve from 3.60118

Epoch 10: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
537/537 [=====] - 219s 407ms/step - loss: 3.1921 - val_loss: 3.6394 - lr: 0.0010
Epoch 11/50
537/537 [=====] - ETA: 0s - loss: 3.0375
Epoch 11: val_loss did not improve from 3.60118
537/537 [=====] - 218s 407ms/step - loss: 3.0375 - val_loss: 3.6417 - lr: 2.0000e-04
Epoch 12/50
537/537 [=====] - ETA: 0s - loss: 3.0073
Epoch 12: val_loss did not improve from 3.60118
Restoring model weights from the end of the best epoch: 7.
537/537 [=====] - 217s 403ms/step - loss: 3.0073 - val_loss: 3.6418 - lr: 2.0000e-04
Epoch 12: early stopping

```

▼ Inference

- Learning Curve (Loss Curve)
- Assessment of Generated imgCaptions (by checking the relevance of the caption with respect to the image, BLEU Score will not be used in this kernel)

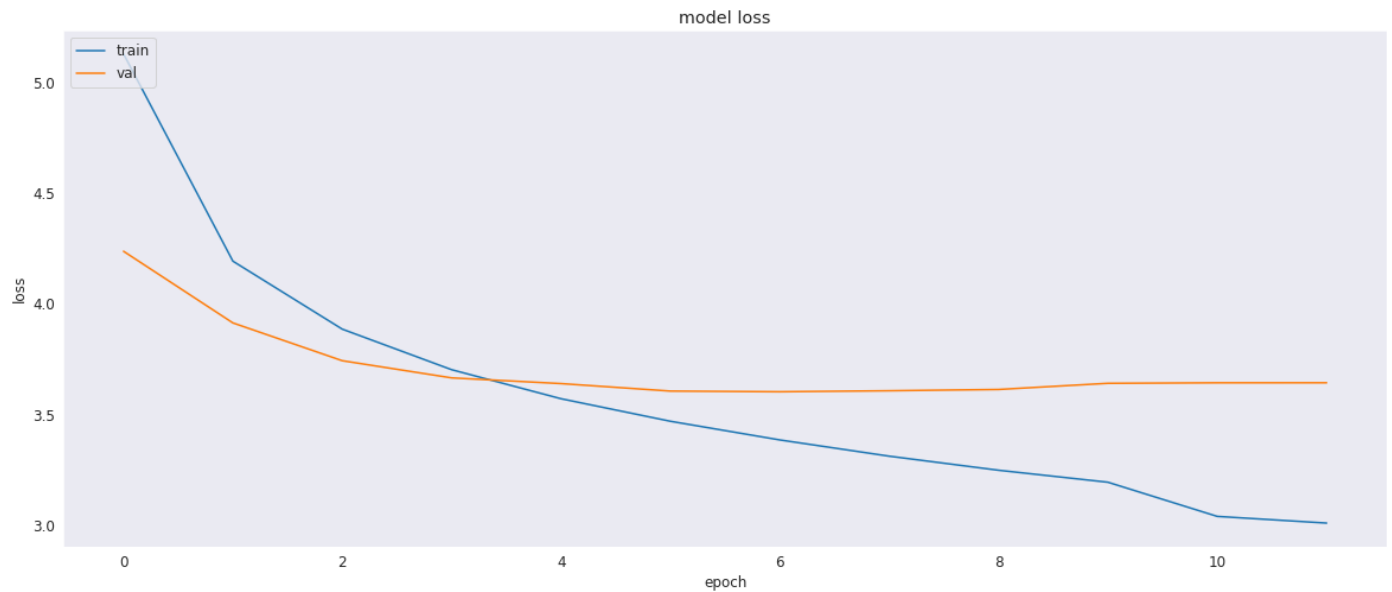
▼ Learning Curve

- The model has clearly overfit, possibly due to less amount of data
- We can tackle this problem in two ways
 1. Train the model on a larger dataset Flickr40k
 2. Attention Models

```

plt.figure(figsize=(20,8))
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

```

▼ Caption Generation Utility Functions

- Utility functions to generate the imgCaptions of input images at the inference time.
- Here the image embeddings are passed along with the first word, followed by which the text embedding of each new word is passed to generate the next word

```
def index_to_word(integer,tokenizer):

    for word, index in tokenizer.word_index.items():
        if index==integer:
            return word
    return None

def caption_predication(model, image, tokenizer, maximum_len, features):

    feature = features[image]
    in_text = "startseq"
    for i in range(maximum_len):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maximum_len)

        y_pred = model.predict([feature,sequence])
        y_pred = np.argmax(y_pred)

        word = index_to_word(y_pred, tokenizer)

        if word is None:
            break

        in_text+= " " + word

        if word == 'endseq':
            break

    return in_text
```

▼ Taking 15 Random Samples for Caption Prediction

```
samples = test.sample(15)
samples.reset_index(drop=True,inplace=True)
```

```

for index,record in samples.iterrows():

    img = load_img(os.path.join(imgPath,record['image']),target_size=(224,224))
    img = img_to_array(img)
    img = img/255.

    caption = caption_predication(caption_model, record['image'], tokenizer, maximum_len, features)
    samples.loc[index,'caption'] = caption


1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 27ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 21ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 17ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 17ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 17ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 17ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 22ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 19ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 18ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 20ms/step
1/1 [=====] - 0s 19ms/step

```


Results

- As we can clearly see there is some redundant caption generation e.g. Dog running through the water, overusage of blue shirt for any other coloured cloth
- The model performance can be further improved by training on more data and using attention mechanism so that our model can focus on relevant areas during the text generation
- We can also leverage the interpretability of the attention mechanism to understand which areas of the image leads to the generation of which word


```
displayImgs(samples)
```




startseq two girls are standing on the street endseq




startseq group of people are standing in front of building endseq




startseq dog is running through the grass endseq




startseq young girl in blue swimsuit is playing in the water endseq




startseq two men in red uniforms are playing soccer endseq




startseq group of people are playing in the water endseq




startseq man is walking down the mountain endseq




startseq two girls are playing with black dog endseq




startseq man is jumping on the ground endseq




startseq man is climbing rock endseq




startseq man in blue shirt is running through the water endseq




startseq two children playing in the water endseq




startseq boy in blue shirt is jumping into the water endseq



startseq man is riding skateboard down the hill endseq



startseq girl is sitting on bench endseq



Conclusion: This may not be the best performing model, but the objective of this kernel is to give a gist of how Image Captioning problems can be approached. In the future work of this kernel **Attention model** training and **BLEU Score** assessment will be performed.