image-captioning-using-cnns-lstms

June 16, 2023

[1]:
```python
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')
```

1 Image Captioning

What is Image Captioning ? - Image Captioning is the process of generating textual description of an image. It uses both Natural Language Processing and Computer Vision to generate the captions. - This task lies at the intersection of computer vision and natural language processing. 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.

CNNs + RNNs (LSTMs) - To perform Image Captioning we will require two deep learning models combined into one for the training purpose - CNNs extract the features from the image of some vector size aka the vector embeddings. The size of these embeddings depend on the type of pretrained network being used for the feature extraction - LSTMs are used for the text generation process. The image embeddings are concatenated with the word embeddings and passed to the LSTM to generate the next word - For a more illustrative explanation of this architecture check the Modelling section for a picture representation

[2]:
```
image_path = '../input/flickr8k/Images'
```

[3]:
```
data = pd.read_csv("../input/flickr8k/captions.txt")
data.head()
```

[3]:
```
                       image  \
0  1000268201_693b08cb0e.jpg
1  1000268201_693b08cb0e.jpg
2  1000268201_693b08cb0e.jpg
3  1000268201_693b08cb0e.jpg
4  1000268201_693b08cb0e.jpg

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

[4]:
```
def readImage(path,img_size=224):
    img = load_img(path,color_mode='rgb',target_size=(img_size,img_size))
    img = img_to_array(img)
    img = img/255.

    return img

def display_images(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 = readImage(f"../input/flickr8k/Images/{temp_df.image[i]}")
        plt.imshow(image)
        plt.title("\n".join(wrap(temp_df.caption[i], 20)))
        plt.axis("off")
```

2 Visualization

- Images and their corresponding captions

[5]:
```
display_images(data.sample(15))
```

3 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

[6]:
```
def text_preprocessing(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
```

3.1 Preprocessed Text

[7]:
```
data = text_preprocessing(data)
captions = data['caption'].tolist()
captions[:10]
```

[7]:
```
['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']
```

3.2 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

[8]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(captions)
vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(caption.split()) for caption in captions)

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

split_index = round(0.85*nimages)
train_images = images[:split_index]
val_images = images[split_index:]

train = data[data['image'].isin(train_images)]
test = data[data['image'].isin(val_images)]

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

tokenizer.texts_to_sequences([captions[1]])[0]
```

[8]:
```
[1, 18, 315, 63, 195, 116, 2]
```

4 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

[9]:
```
model = DenseNet201()
fe = Model(inputs=model.input, outputs=model.layers[-2].output)

img_size = 224
features = {}
for image in tqdm(data['image'].unique().tolist()):
    img = load_img(os.path.
 ↪join(image_path,image),target_size=(img_size,img_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.h5
82526208/82524592 [==============================] - 1s 0us/step
82534400/82524592 [==============================] - 1s 0us/step

100%|      | 8091/8091 [10:17<00:00, 13.11it/s]
```

5 Data Generation

- Since Image Caption model training like any other neural network training is a highly resource utillizing 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 corresonding caption text embeddings for the training process
- The text embeddings are passed word by word for the caption generation during inference time

[10]:
```
class CustomDataGenerator(Sequence):

    def __init__(self, df, X_col, y_col, batch_size, directory, tokenizer,
                 vocab_size, max_length, 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.vocab_size = vocab_size
        self.max_length = max_length
        self.features = features
        self.shuffle = shuffle
        self.n = len(self.df)

    def on_epoch_end(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.__get_data(batch)
        return (X1, X2), y

    def __get_data(self,batch):

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

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

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

            captions = batch.loc[batch[self.X_col]==image, self.y_col].tolist()
            for caption in captions:
                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.max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=self.
↪vocab_size)[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
```

6 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

[11]:
```
input1 = Input(shape=(1920,))
input2 = Input(shape=(max_length,))

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

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

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

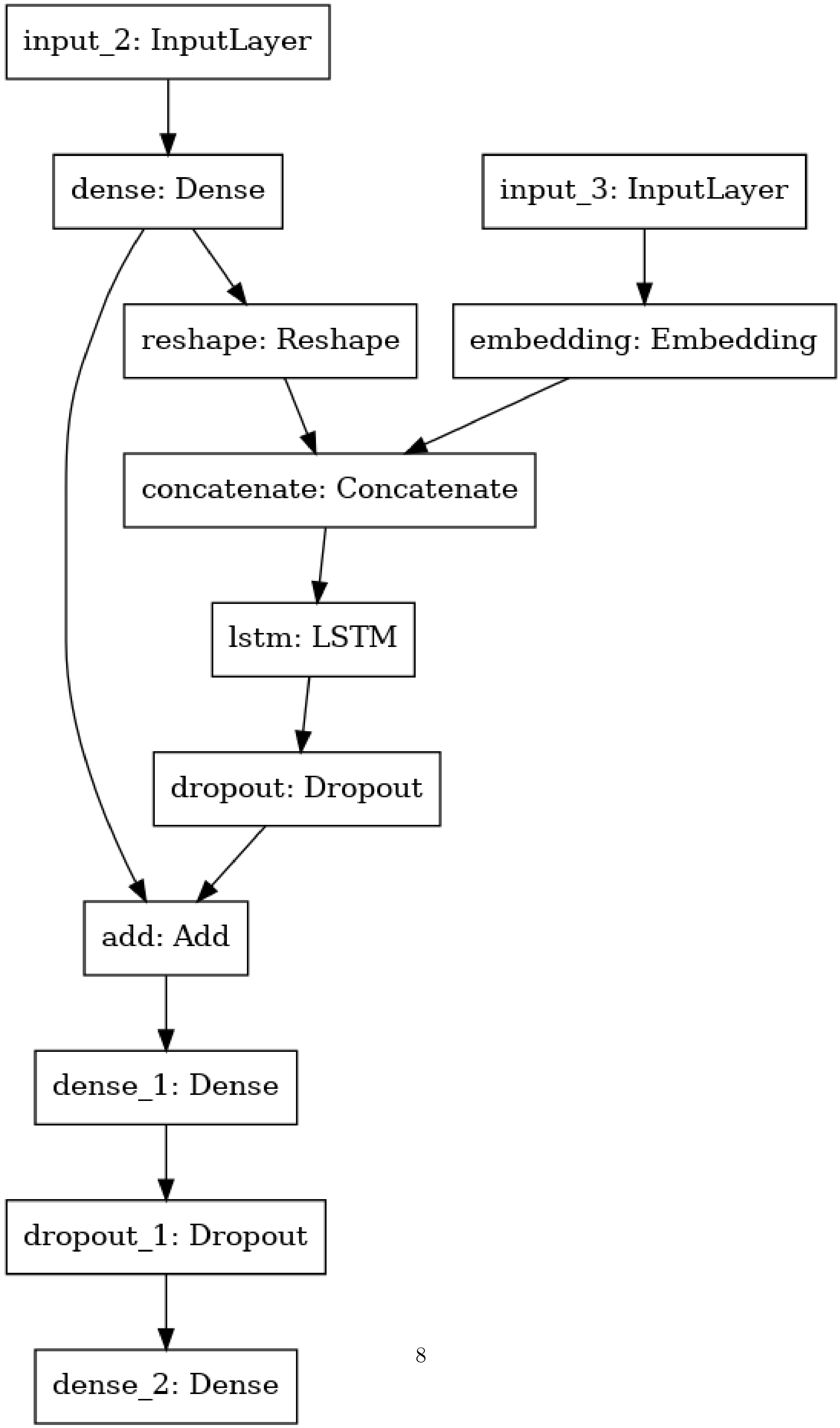
[12]:
```
from tensorflow.keras.utils import plot_model
```

6.1 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 orignally proposed back in 2014: **Show and Tell: A Neural Image Caption Generator** (https://arxiv.org/pdf/1411.4555.pdf)

[13]:
```
plot_model(caption_model)
```

[13]:


input_2: InputLayer
dense: Dense
input_3: InputLayer
reshape: Reshape
embedding: Embedding
concatenate: Concatenate
lstm: LSTM
dropout: Dropout
add: Add
dense_1: Dense
dropout_1: Dropout
dense_2: Dense

[14]:
```python
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
----------------------------------------------------------------------------------
------------------
```

[15]:
```python
train_generator = 
  CustomDataGenerator(df=train,X_col='image',y_col='caption',batch_size=64,directory=image_pa

  tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)

validation_generator = 
  CustomDataGenerator(df=test,X_col='image',y_col='caption',batch_size=64,directory=image_pat

  tokenizer=tokenizer,vocab_size=vocab_size,max_length=max_length,features=features)
```

[16]:
```python
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)
```

6.2 Let's train the Model !

[17]:
```python
history = caption_model.fit(
        train_generator,
        epochs=50,
        validation_data=validation_generator,
        callbacks=[checkpoint,earlystopping,learning_rate_reduction])
```

```
Epoch 1/50
537/537 [==============================] - 238s 438ms/step - loss: 5.1614 -
val_loss: 4.2591

Epoch 00001: val_loss improved from inf to 4.25914, saving model to model.h5
```

```
Epoch 2/50
537/537 [==============================] - 49s 91ms/step - loss: 4.2048 -
val_loss: 3.9205

Epoch 00002: val_loss improved from 4.25914 to 3.92045, saving model to model.h5
Epoch 3/50
537/537 [==============================] - 49s 92ms/step - loss: 3.9412 -
val_loss: 3.8001

Epoch 00003: val_loss improved from 3.92045 to 3.80013, saving model to model.h5
Epoch 4/50
537/537 [==============================] - 48s 90ms/step - loss: 3.7803 -
val_loss: 3.7085

Epoch 00004: val_loss improved from 3.80013 to 3.70848, saving model to model.h5
Epoch 5/50
537/537 [==============================] - 48s 88ms/step - loss: 3.6648 -
val_loss: 3.6745

Epoch 00005: val_loss improved from 3.70848 to 3.67446, saving model to model.h5
Epoch 6/50
537/537 [==============================] - 49s 91ms/step - loss: 3.5691 -
val_loss: 3.6394

Epoch 00006: val_loss improved from 3.67446 to 3.63944, saving model to model.h5
Epoch 7/50
537/537 [==============================] - 49s 91ms/step - loss: 3.4928 -
val_loss: 3.6347

Epoch 00007: val_loss improved from 3.63944 to 3.63475, saving model to model.h5
Epoch 8/50
537/537 [==============================] - 48s 90ms/step - loss: 3.4293 -
val_loss: 3.6243

Epoch 00008: val_loss improved from 3.63475 to 3.62433, saving model to model.h5
Epoch 9/50
537/537 [==============================] - 48s 89ms/step - loss: 3.3706 -
val_loss: 3.6378

Epoch 00009: val_loss did not improve from 3.62433
Epoch 10/50
537/537 [==============================] - 49s 91ms/step - loss: 3.3207 -
val_loss: 3.6527

Epoch 00010: val_loss did not improve from 3.62433
Epoch 11/50
537/537 [==============================] - 49s 90ms/step - loss: 3.2741 -
val_loss: 3.6727
```

```
Epoch 00011: val_loss did not improve from 3.62433

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.
Epoch 12/50
537/537 [==============================] - 48s 90ms/step - loss: 3.1623 -
val_loss: 3.6590

Epoch 00012: val_loss did not improve from 3.62433
Epoch 13/50
537/537 [==============================] - 48s 90ms/step - loss: 3.1374 -
val_loss: 3.6609

Epoch 00013: val_loss did not improve from 3.62433
Restoring model weights from the end of the best epoch.
Epoch 00013: early stopping
```

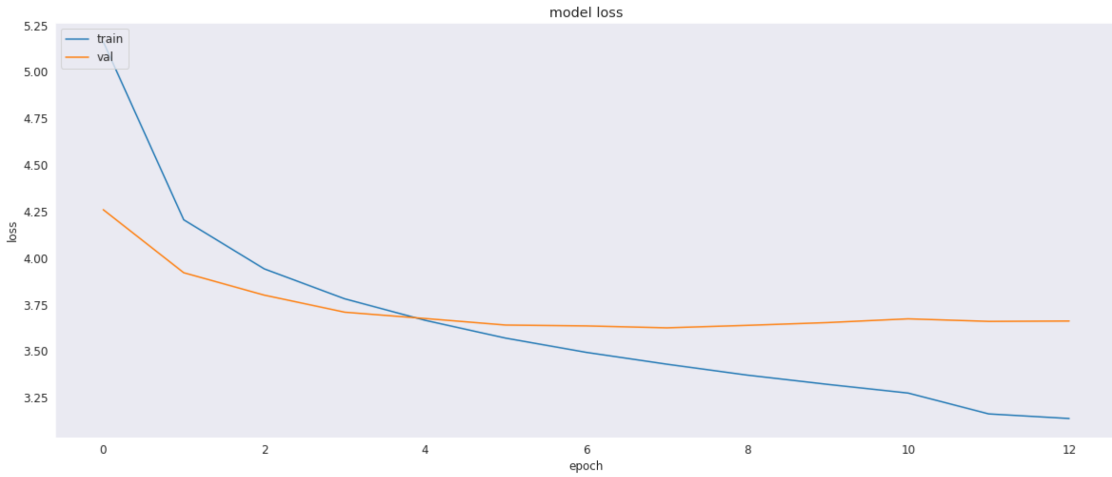
7 Inference

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

7.1 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

[18]:
```
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()
```



7.2 Caption Generation Utility Functions

- Utility functions to generate the captions 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

[19]:
```
def idx_to_word(integer,tokenizer):

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

[20]:
```
def predict_caption(model, image, tokenizer, max_length, features):

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

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

        word = idx_to_word(y_pred, tokenizer)

        if word is None:
            break

        in_text+= " " + word
```

```
        if word == 'endseq':
            break

    return in_text
```

7.3 Taking 15 Random Samples for Caption Prediction

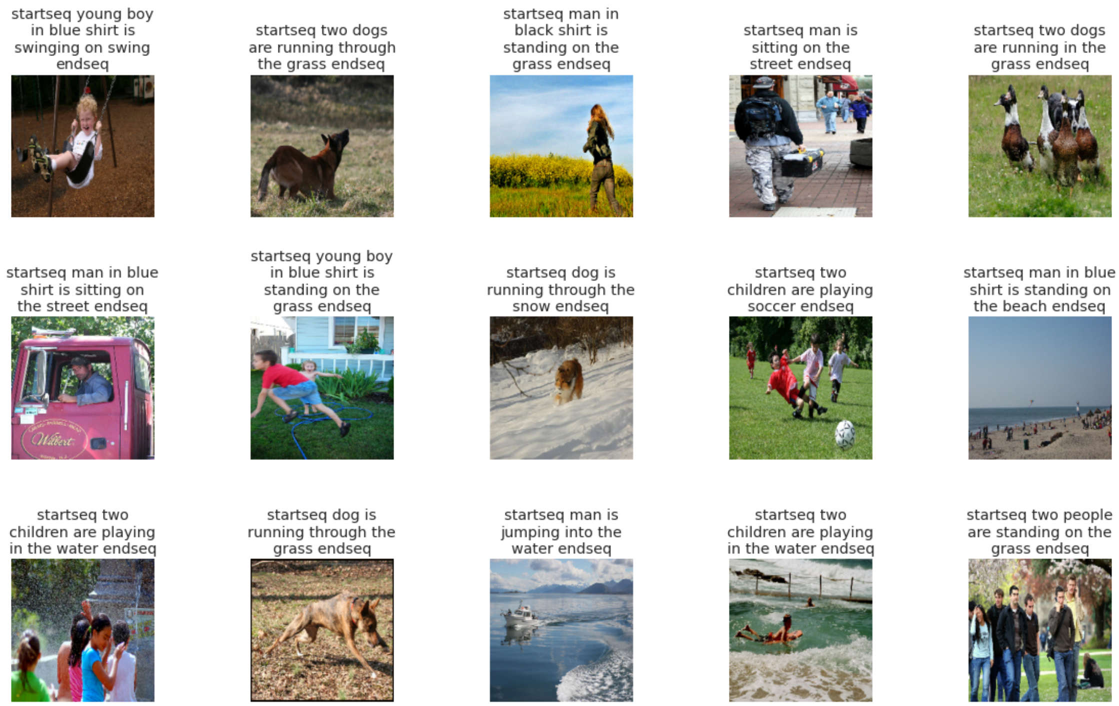
[21]:
```
samples = test.sample(15)
samples.reset_index(drop=True,inplace=True)
```

[22]:
```
for index,record in samples.iterrows():

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

    caption = predict_caption(caption_model, record['image'], tokenizer,␣
↪max_length, features)
    samples.loc[index,'caption'] = caption
```

8 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 interprettability of the attention mechanism to understand which areas of the image leads to the generation of which word

[23]:
```
display_images(samples)
```



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.