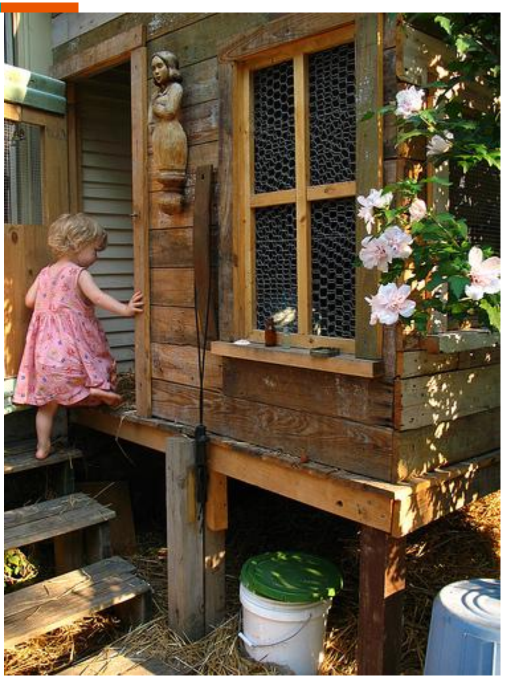
Midterm Deep Learning Images Caption

Nguyễn Trung Nguyên-520V0015, Hồ Trọng Nghĩa-520K0163

What is images caption?

Image Captioning is the task of describing the content of an image in words. 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.



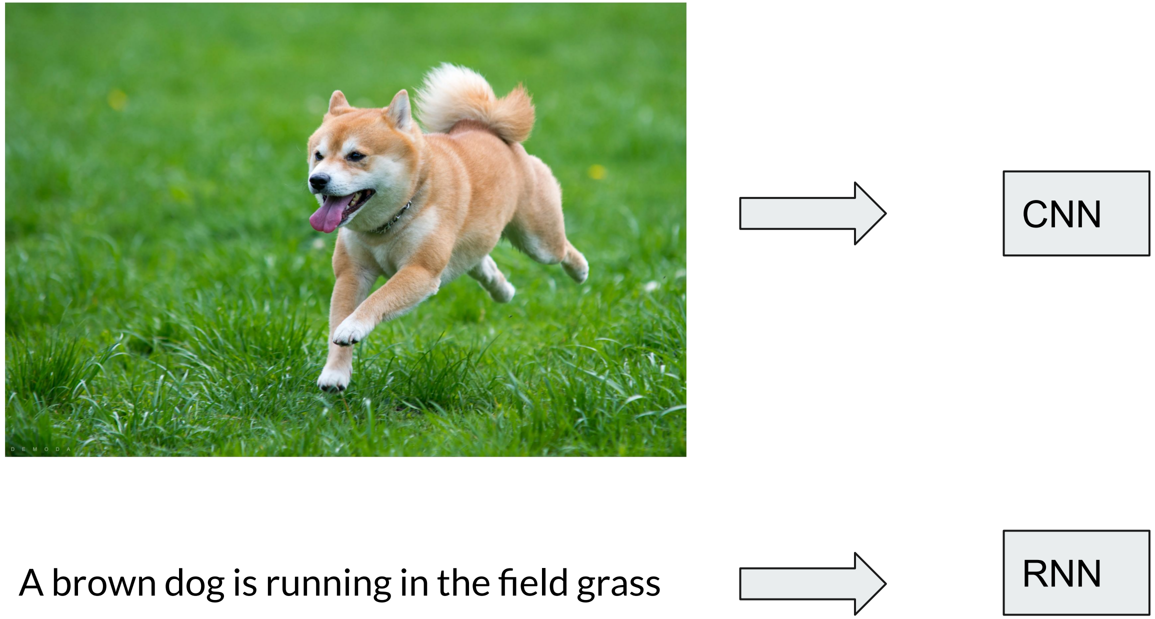
Input

A child in a pink dress is climbing up a set of stairs in an entry way .

Output

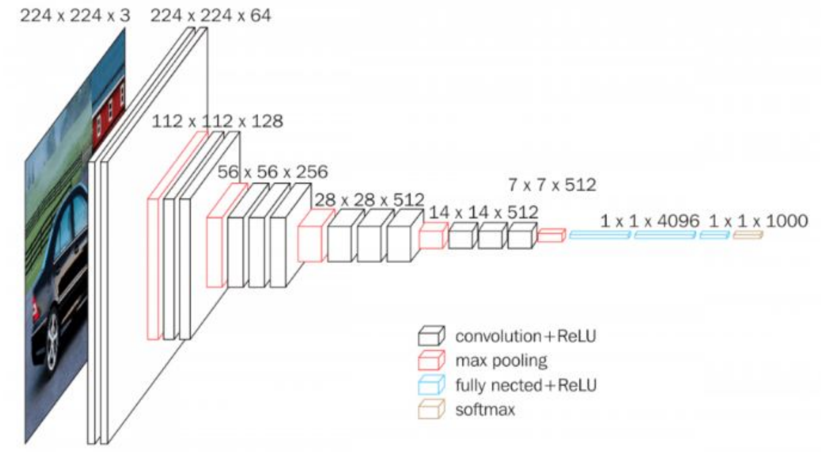
Idea

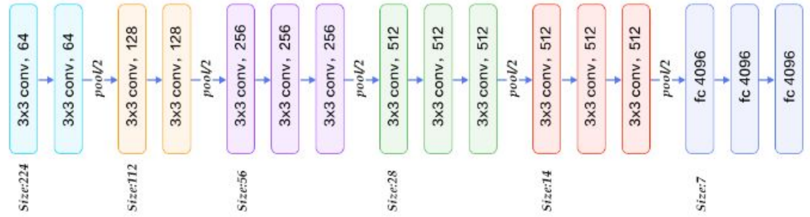
The first problem of this problem is, this problem has 2 input data sources at the time of training. The first is the photo, the second is the caption of the photo.

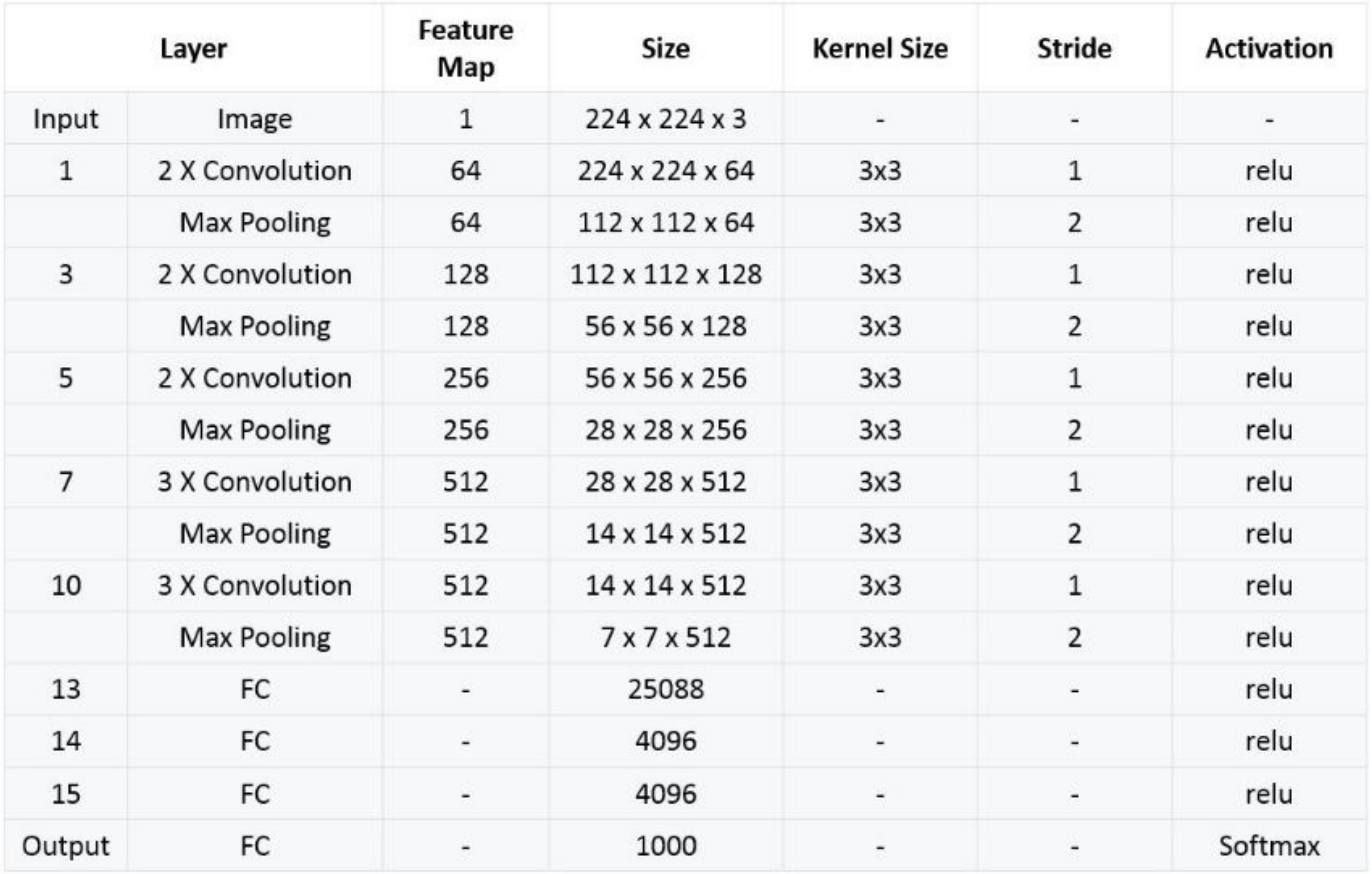


VGG16

The full name of VGG is the Visual Geometry Group, which belongs to the Department of Science and Engineering of Oxford University. It has released a series of convolutional network models beginning with VGG, which can be applied to face recognition and image classification, from VGG16 to VGG19. The original purpose of VGG's research on the depth of convolutional networks is to understand how the depth of convolutional networks affects the accuracy and accuracy of large-scale image classification and recognition. -Deep-16 CNN), in order to deepen the number of network layers and to avoid too many parameters, a small 3x3 convolution kernel is used in all layers.



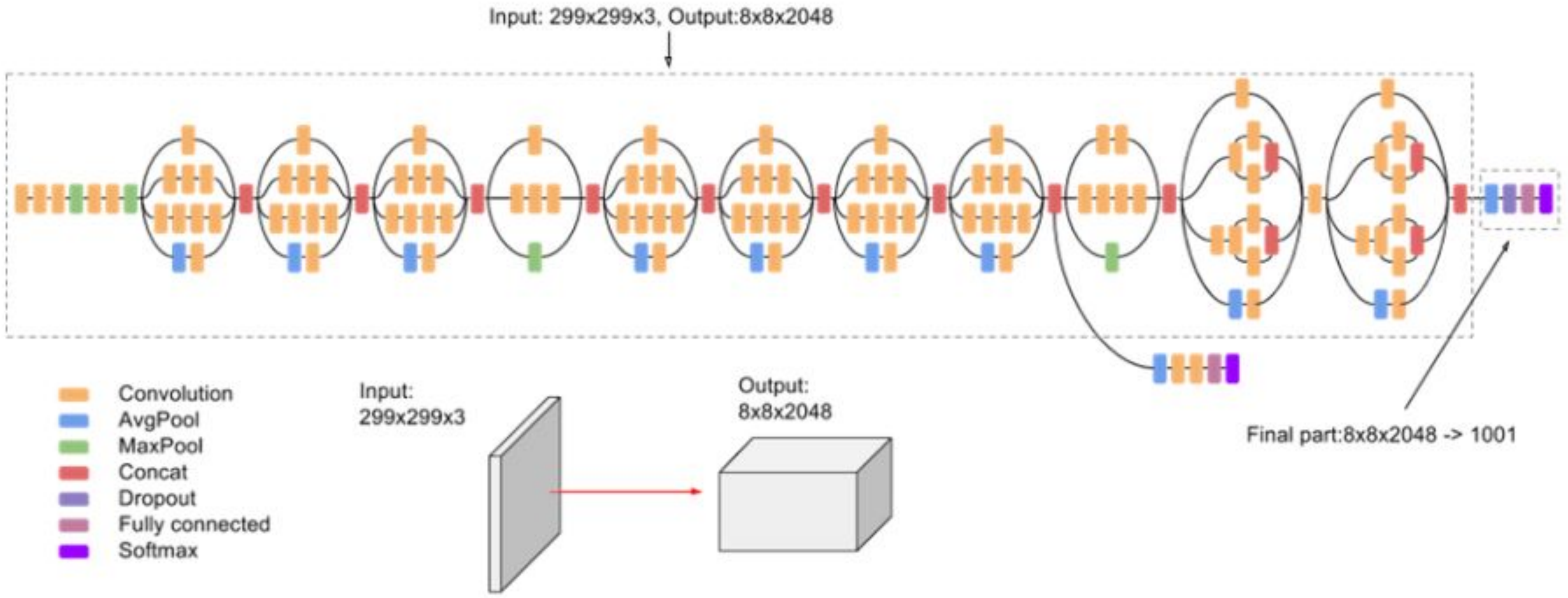




Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

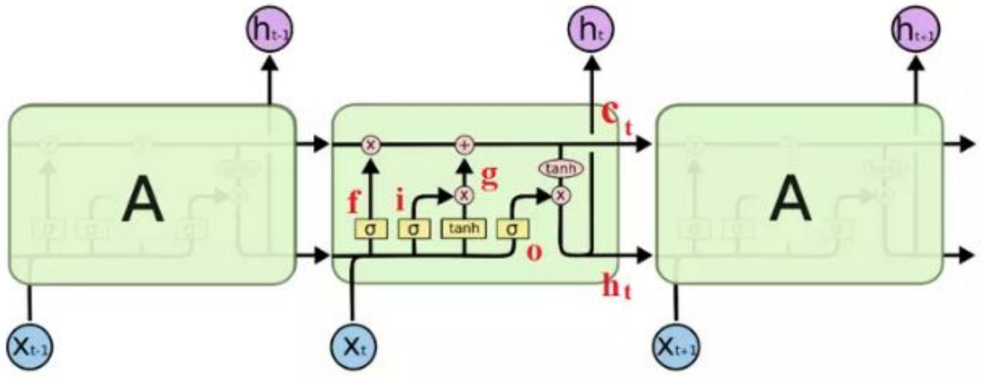
InceptionV3

Inception-v3 is a **pre-trained convolutional neural network** that is 48 layers deep, which is a version of the network already trained on more than a million images from the ImageNet database. This pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 299-by-299. The model extracts general features from input images in the first part and classifies them based on those features in the second part.


Input: 299x299x3, Output:8x8x2048
Final part:8x8x2048 -> 1001
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Input: 299x299x3
Output: 8x8x2048

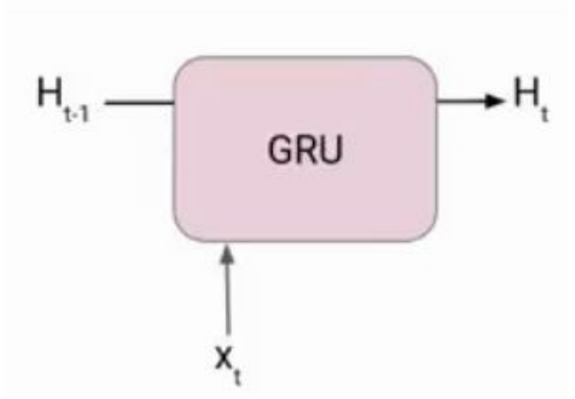
LSTM

Long Short Term Memory networks, commonly known as LSTMs - are a special type of RNN that is capable of learning distant dependencies. LSTM was introduced by Hochreiter & Schmidhuber (1997), and has since been refined and popularized by many people in the industry. They work extremely effectively on many different problems, so they have gradually become as popular as they are today.



GRU

GRUs are very similar to Long Short Term Memory(LSTM). Just like LSTM, GRU uses gates to control the flow of information. They are relatively new as compared to LSTM. This is the reason they offer some improvement over LSTM and have simpler architecture.

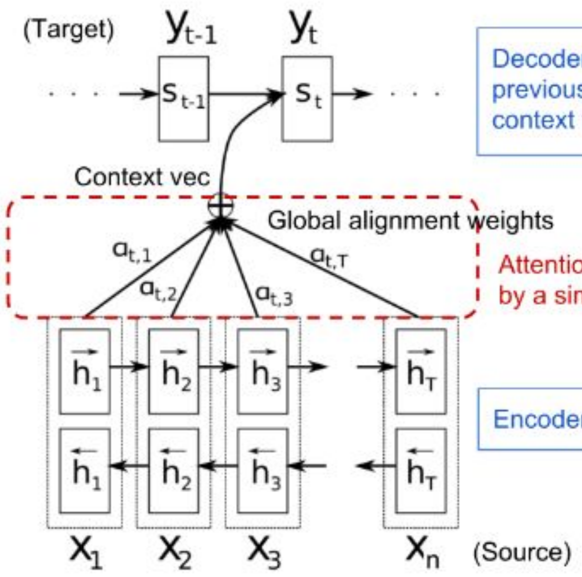


Reset Gate (Short term memory): $r_t = \sigma(x_t * U_r + H_{t-1} * W_r)$

Update Gate (Long Term memory): $u_t = \sigma(x_t * U_u + H_{t-1} * W_u)$

Attention mechanism

ATTENTION MECHANISM is a mechanism that makes it possible for the model to focus on important parts of the data, by creating an alignment model a that calculates the alignment score aij to REWEIGHT the hidden state hj of the encoder. In the NMT problem using seq2seq, that helps the model focus more on the important words in the input x sentence, thereby predicting the next yi word at the decoder, which is represented by brightly colored boxes as shown in the confusion matrix below. above, is also the degree of correlation from xj (source) and from yi (target)



Decoder: RNN with input from previous state + dynamic context vector.

Attention layer: parameterized by a simple feed-forward network

Additive Attention

Encoder: bidirectional RNN

BLEU score

BLEU stands for Bilingual Evaluation Understudy, which is a method for evaluating a translation based on reference translations, introduced in the paper BLEU: a Method for Automatic Evaluation of Machine Translation). BLEU is designed for use in machine translation, but in fact, this measurement is also used in tasks such as text summarization, speech recognition, image label generation, etc. Besides, this measurement can completely be used to assess the quality of the staff's translation.

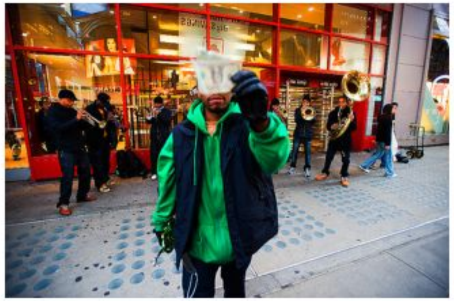
$$Bleu\ (N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

[illegible]

Implement

We use Flickr8k data for our model

It contains 8000 pictures and each picture has 5 captions



"A man holding money

A man holds a dollar bill in front of his face while posing in front of a street band

A man holds money in the air

A man in a green jacket is standing outside a store holding some money in front of his face

"An African-American man wearing a green sweatshirt and blue vest is holding up 2 dollar bills in front of his face

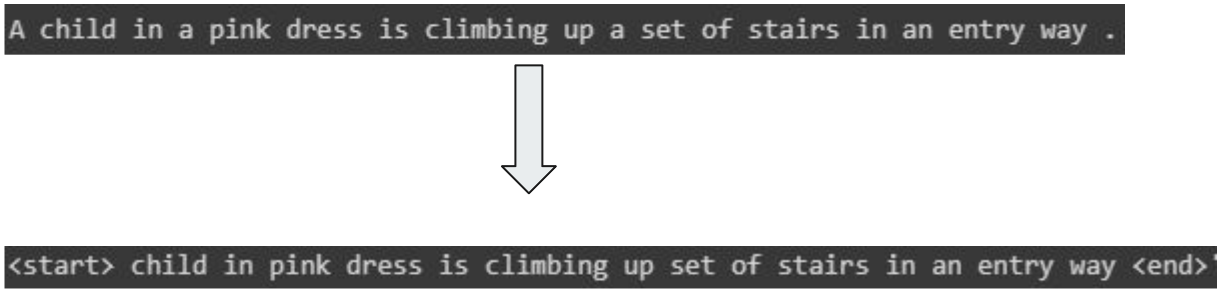
Clean data

Converting to lowercase

Remove punctuation

Remove numeric values

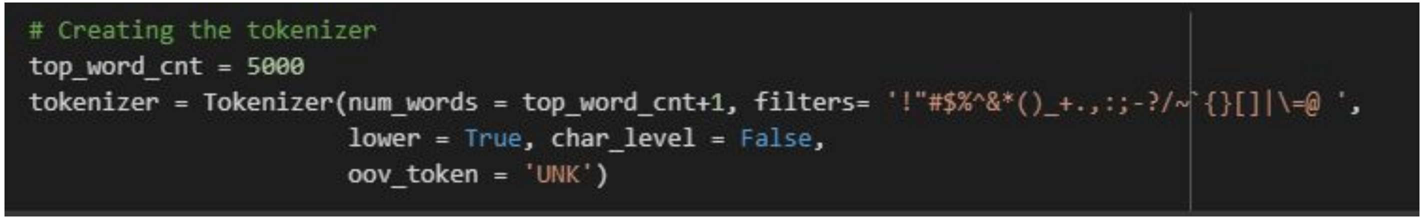
Add <start> and end tag



```
A child in a pink dress is climbing up a set of stairs in an entry way .
```

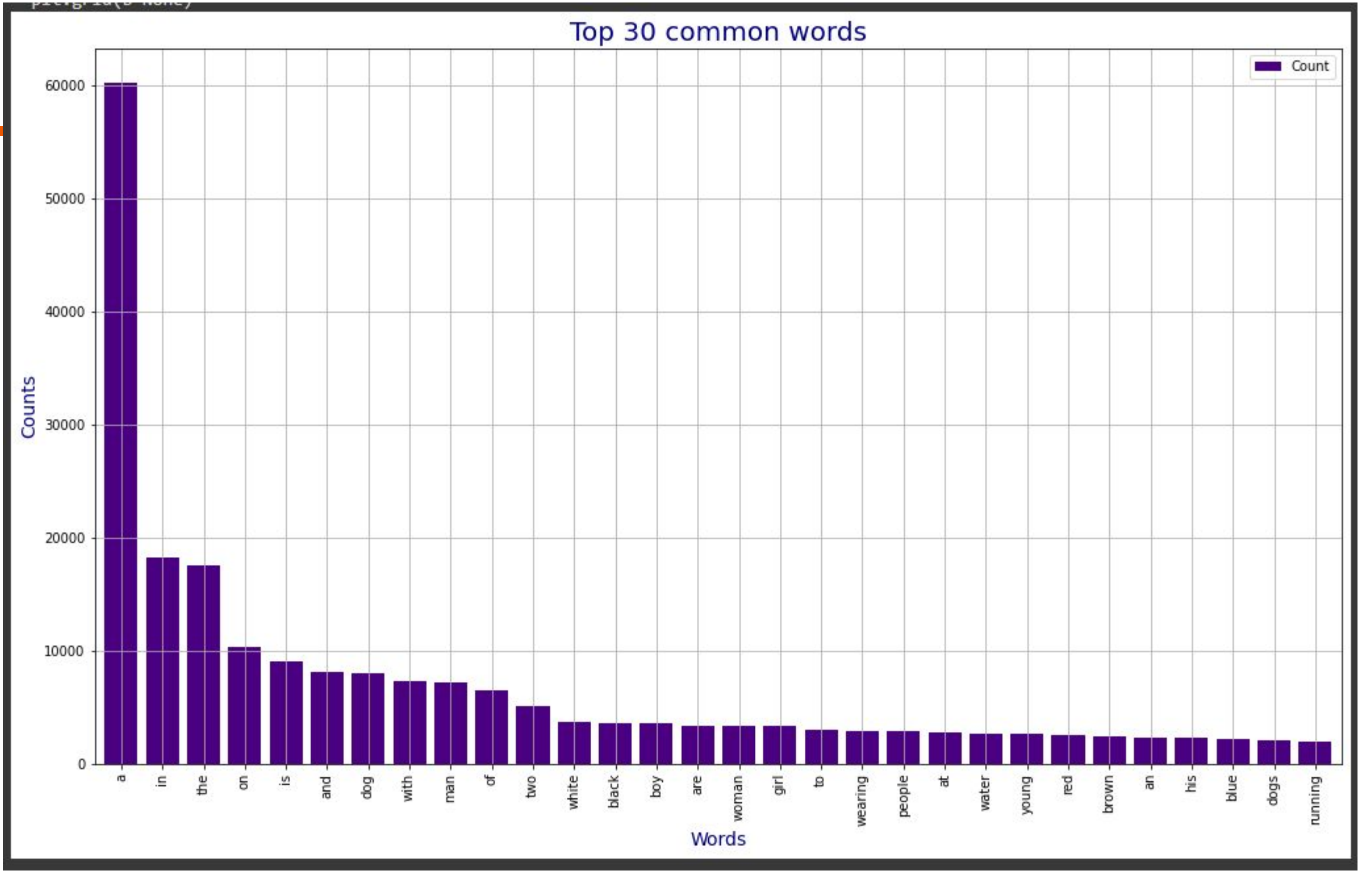
```
<start> child in pink dress is climbing up set of stairs in an entry way <end>
```

Data processing

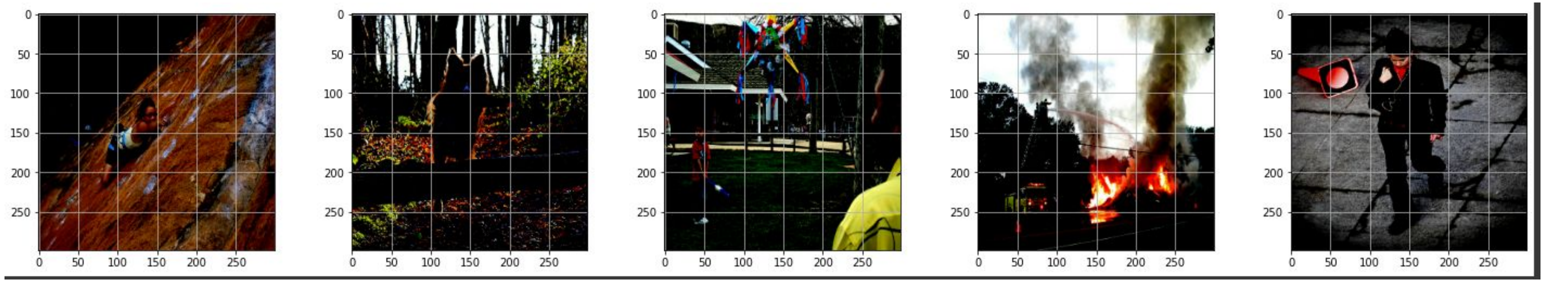


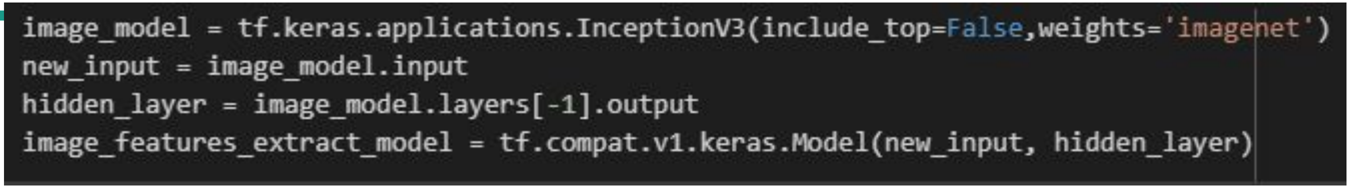
```python
# Creating the tokenizer
top_word_cnt = 5000
tokenizer = Tokenizer(num_words = top_word_cnt+1, filters= '!"#$%^&*()_+.,:;-?/~`{}[]|\=@ ',
                      lower = True, char_level = False,
                      oov_token = 'UNK')
```

Remove symbols and add UNK for unknown words

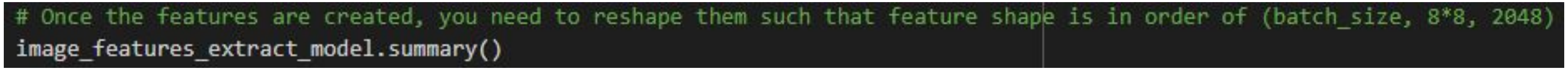

Top 30 common words
Count
Counts
60000
50000
40000
30000
20000
10000
0
a
in
the
on
is
and
dog
with
man
of
two
white
black
boy
are
woman
girl
to
wearing
people
at
water
young
red
brown
an
his
blue
dogs
running
Words

Store images and resize(299, 299, 3)

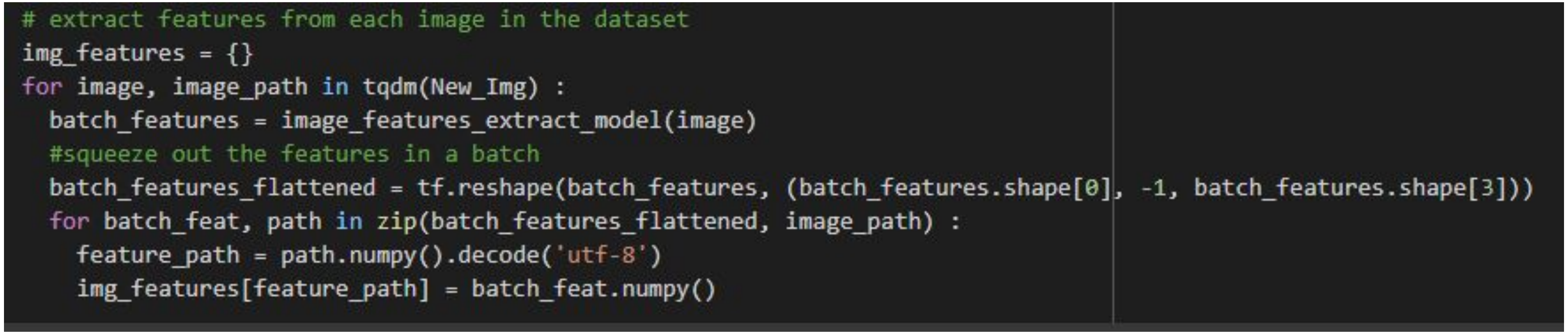




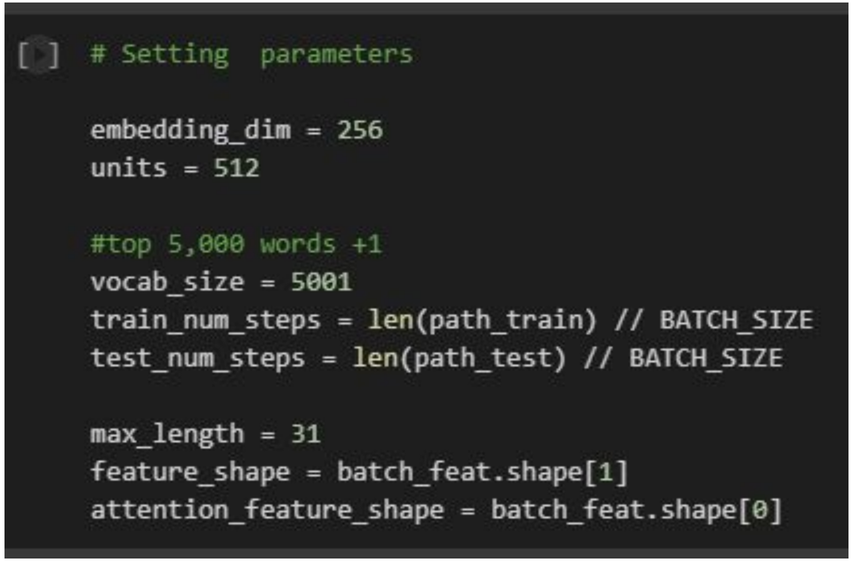
```python
image_model = tf.keras.applications.InceptionV3(include_top=False,weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output
image_features_extract_model = tf.compat.v1.keras.Model(new_input, hidden_layer)
```



```python
# Once the features are created, you need to reshape them such that feature shape is in order of (batch_size, 8*8, 2048)
image_features_extract_model.summary()
```



```
# extract features from each image in the dataset
img_features = {}
for image, image_path in tqdm(New_Img) :
  batch_features = image_features_extract_model(image)
  #squeeze out the features in a batch
  batch_features_flattened = tf.reshape(batch_features, (batch_features.shape[0], -1, batch_features.shape[3]))
  for batch_feat, path in zip(batch_features_flattened, image_path) :
    feature_path = path.numpy().decode('utf-8')
    img_features[feature_path] = batch_feat.numpy()
```

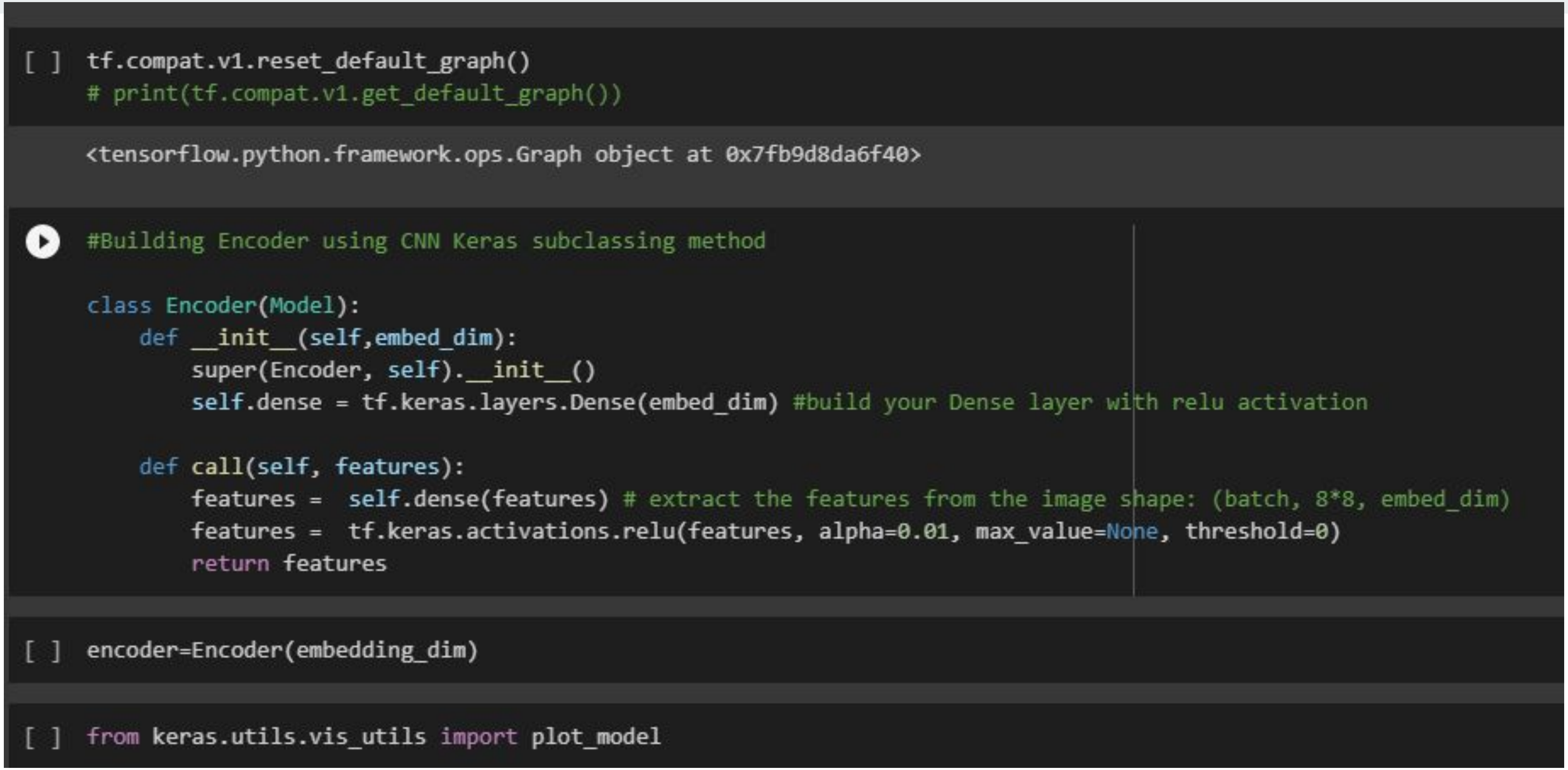


```
# Setting  parameters

embedding_dim = 256
units = 512

#top 5,000 words +1
vocab_size = 5001
train_num_steps = len(path_train) // BATCH_SIZE
test_num_steps = len(path_test) // BATCH_SIZE

max_length = 31
feature_shape = batch_feat.shape[1]
attention_feature_shape = batch_feat.shape[0]
```



```
[ ] tf.compat.v1.reset_default_graph()
    # print(tf.compat.v1.get_default_graph())
```

```
<tensorflow.python.framework.ops.Graph object at 0x7fb9d8da6f40>
```

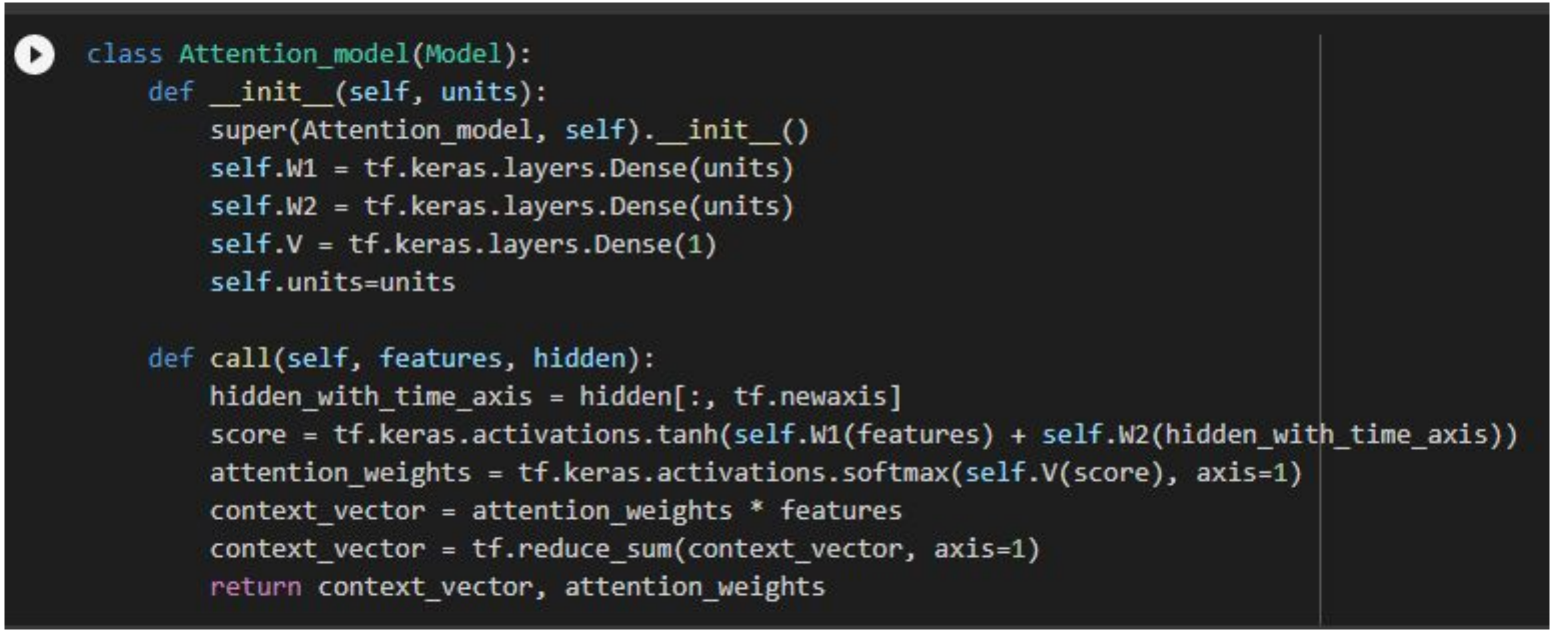
```
#Building Encoder using CNN Keras subclassing method

class Encoder(Model):
    def __init__(self,embed_dim):
        super(Encoder, self).__init__()
        self.dense = tf.keras.layers.Dense(embed_dim) #build your Dense layer with relu activation

    def call(self, features):
        features =  self.dense(features) # extract the features from the image shape: (batch, 8*8, embed_dim)
        features =  tf.keras.activations.relu(features, alpha=0.01, max_value=None, threshold=0)
        return features
```

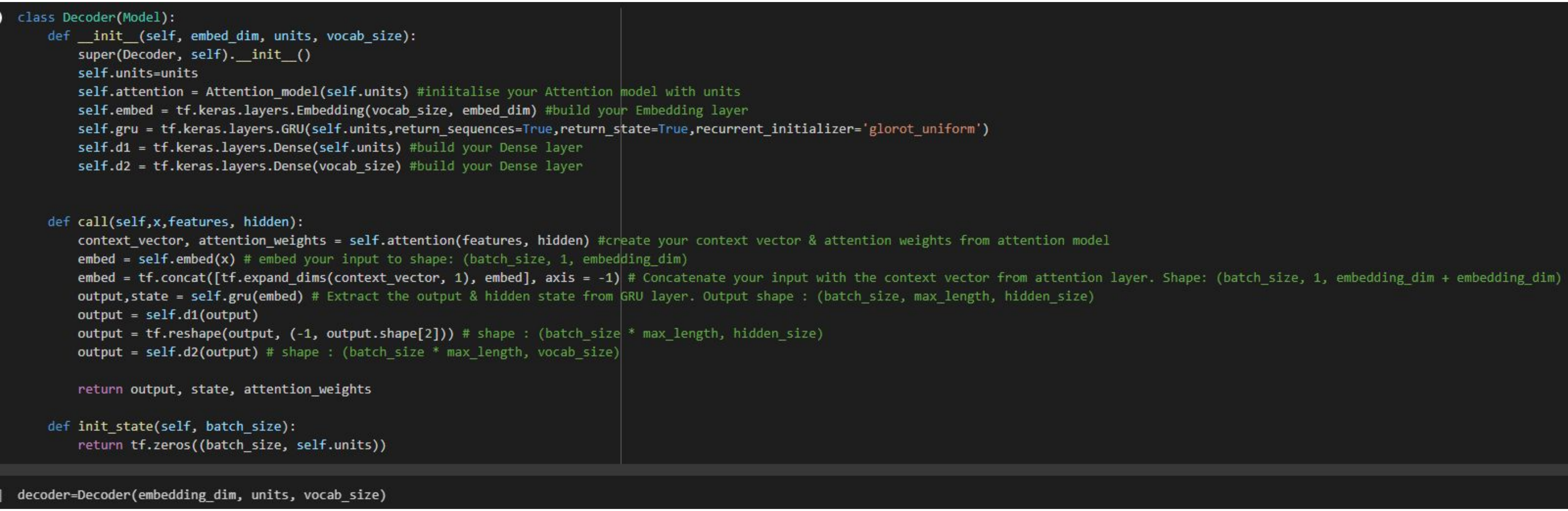
```
[ ] encoder=Encoder(embedding_dim)
```

```
[ ] from keras.utils.vis_utils import plot_model
```



```
class Attention_model(Model):
    def __init__(self, units):
        super(Attention_model, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)
        self.units=units

    def call(self, features, hidden):
        hidden_with_time_axis = hidden[:, tf.newaxis]
        score = tf.keras.activations.tanh(self.W1(features) + self.W2(hidden_with_time_axis))
        attention_weights = tf.keras.activations.softmax(self.V(score), axis=1)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights
```



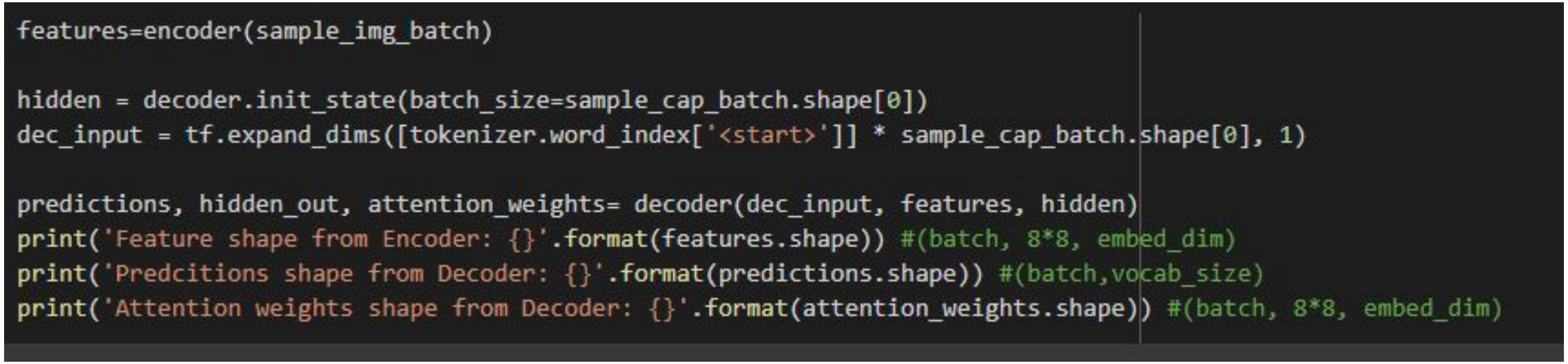
```
class Decoder(Model):
    def __init__(self, embed_dim, units, vocab_size):
        super(Decoder, self).__init__()
        self.units=units
        self.attention = Attention_model(self.units) #iniitalise your Attention model with units
        self.embed = tf.keras.layers.Embedding(vocab_size, embed_dim) #build your Embedding layer
        self.gru = tf.keras.layers.GRU(self.units,return_sequences=True,return_state=True,recurrent_initializer='glorot_uniform')
        self.d1 = tf.keras.layers.Dense(self.units) #build your Dense layer
        self.d2 = tf.keras.layers.Dense(vocab_size) #build your Dense layer


    def call(self,x,features, hidden):
        context_vector, attention_weights = self.attention(features, hidden) #create your context vector & attention weights from attention model
        embed = self.embed(x) # embed your input to shape: (batch_size, 1, embedding_dim)
        embed = tf.concat([tf.expand_dims(context_vector, 1), embed], axis = -1) # Concatenate your input with the context vector from attention layer. Shape: (batch_size, 1, embedding_dim + embedding_dim)
        output,state = self.gru(embed) # Extract the output & hidden state from GRU layer. Output shape : (batch_size, max_length, hidden_size)
        output = self.d1(output)
        output = tf.reshape(output, (-1, output.shape[2])) # shape : (batch_size * max_length, hidden_size)
        output = self.d2(output) # shape : (batch_size * max_length, vocab_size)

        return output, state, attention_weights

    def init_state(self, batch_size):
        return tf.zeros((batch_size, self.units))
```

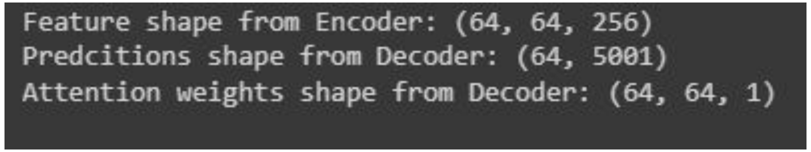
```
decoder=Decoder(embedding_dim, units, vocab_size)
```



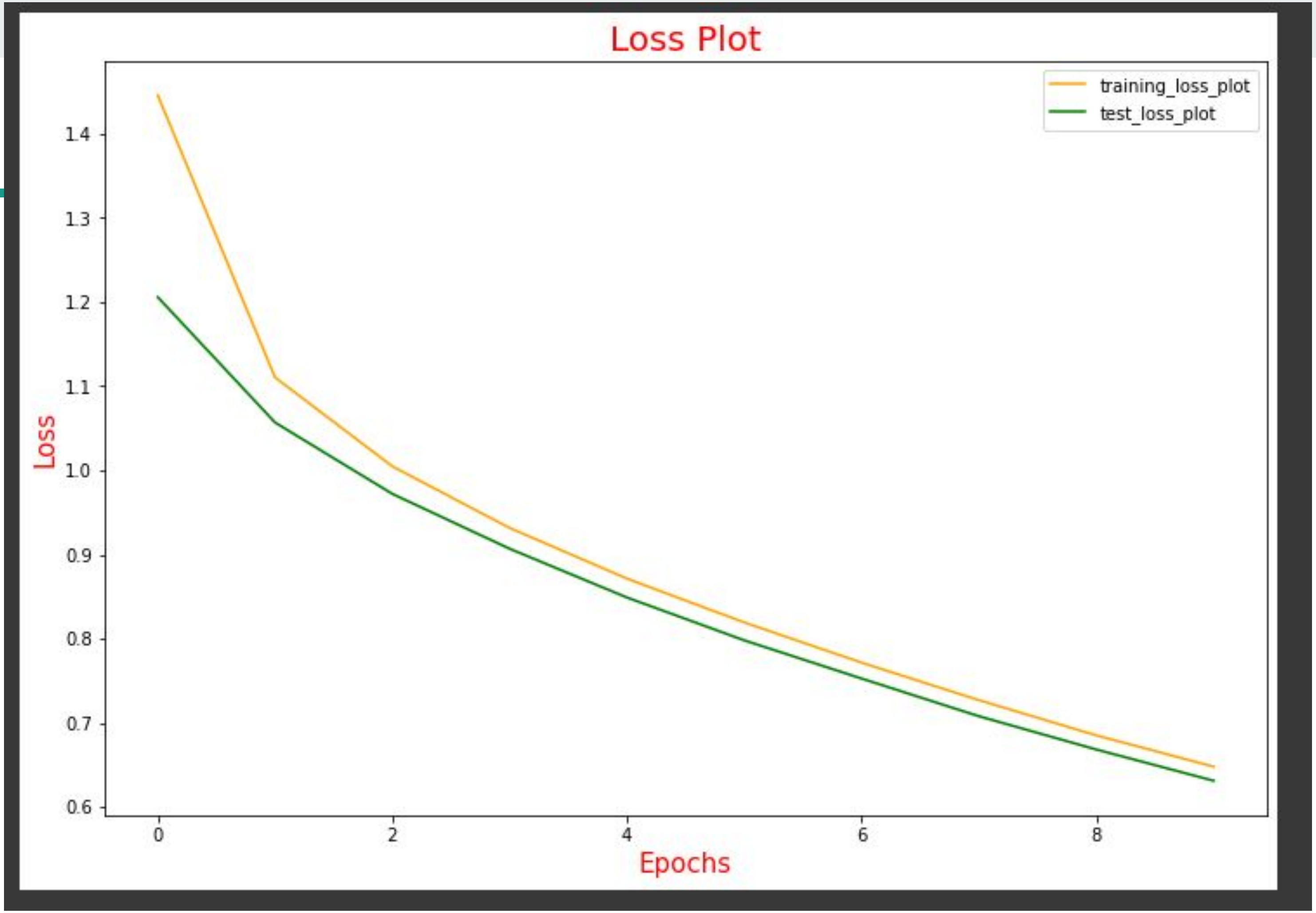
```
features=encoder(sample_img_batch)

hidden = decoder.init_state(batch_size=sample_cap_batch.shape[0])
dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * sample_cap_batch.shape[0], 1)

predictions, hidden_out, attention_weights= decoder(dec_input, features, hidden)
print('Feature shape from Encoder: {}'.format(features.shape)) #(batch, 8*8, embed_dim)
print('Predcitions shape from Decoder: {}'.format(predictions.shape)) #(batch,vocab_size)
print('Attention weights shape from Decoder: {}'.format(attention_weights.shape)) #(batch, 8*8, embed_dim)
```



```
Feature shape from Encoder: (64, 64, 256)
Predcitions shape from Decoder: (64, 5001)
Attention weights shape from Decoder: (64, 64, 1)
```


Loss Plot
training_loss_plot
test_loss_plot
Loss
Epochs
1.4
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
2
4
6
8

```
BLEU score: 48.555993224365864
Real Caption: two girls are wearing life vests and hugging each other
Prediction Caption: two girls are hugging each other
```