Νευρωνικά Δίκτυα και Ευφυή Υπολογιστικά Συστήματα

3η Εργασία

Team 31

Κυριακίδης Δημήτριος (03117077)

Καραντώνης Αντώνιος (03117439)

Κανελλόπουλος Σωτήριος (03117101)

Σε αυτήν την εργασία κατασκευάζουμε ένα σύστημα - μοντέλο παραγωγής περιγραφών (captions) για φωτογραφίες. Βασιζόμεθα στην εκπαιδευτική σελίδα του tensorflow: https://www.tensorflow.org/tutorials/text/image_captioning, η οποία χρησιμοποιεί τον μηχανισμό attention ("Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"), και επιχειρούμε βελτιώσεις επί τούτου. Πρώτη εξ αυτών η χρήση διαφορετικού συνόλου δεδομένων εικόνων: "flickr30k" dataset αντί του "ms-coco train 2014".

Σε κάθε βήμα εκπαιδεύουμε έως ότου φθάσουμε σε ικανοποιητικό επίπεδο loss, και ύστερα αξιολογούμε την νέα επίδοση κυρίως βάσει του corpus bleu score υπολογισμένου επί του validation set, αλλά και εξετάζοντας μεμονωμένα παραδείγματα.

Το τελικό μοντέλο στο οποίο καταλήξαμε εκπαιδεύθηκε για 30 εποχές σημείωσε BLEU score 0.0983 επί του validation set, και 0.1476 χρησιμοποιώντας beam search. Οι υποθέσεις που παρήγαγε για το test set βαθμολογήθηκαν με 0.203244 στον διαγωνισμό του μαθήματος στο codalab.

ΠΕΡΙΕΧΟΜΕΝΑ

Στην ενότητα Improvements έχουν συμπεριληφθεί λεπτομερώς και αναλυτικά όλες οι δοκιμές βελτιώσεων που πραγματοποιήσαμε, με τα output logs των εκπαιδεύσεων, τα αντίστοιχα αποτελέσματα, μετρήσεις BLEU scores, αξιολογήσεις μεμονωμένων φωτογραφιών, και τα συνεπακόλουθα πορίσματα. Επειδή η ενότητα αυτή είναι αρκετά εκτενής, όλες οι χρήσιμες παρατηρήσεις και μετρήσεις που έχουν γίνει αναφέρονται περιληπτικά και στην τελική ενότητα συμπερασμάτων.

In [ ]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import style
from tensorflow import keras
import collections
import random
import numpy as np
import os
import time
import json
```

```
from PIL import Image
from tqdm import tqdm
import pickle
```

Dataset

Download & limit size

In [ ]:
```
## ANTONIS
PATH = os.path.abspath('.') + './image_dir'
test_file = "./test_images.csv"
annotation_file = "./train_captions.csv"
```

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
## Download image files
## image_zip is a string of the directory's name
image_folder = '/drive/My Drive/Colab Notebooks/NN/EX3/images/'
if not os.path.exists(os.path.abspath('.') + image_folder):
    image_zip = tf.keras.utils.get_file('flickr30k-images-ecemod.zip',
                                        cache_subdir = os.path.abspath('.') + image_folder,
                                        origin = 'https://www.dropbox.com/s/efqe8131slq1zah/flickr30k-images-ecemod.zip?
                                        extract = True)
    PATH = os.path.dirname(image_zip) + image_folder
    os.remove(image_zip)
else:
    PATH = os.path.abspath('.') + image_folder
```

```
Downloading data from https://www.dropbox.com/s/efqe8131slq1zah/flickr30k-images-ecemod.zip?dl=1
4376387584/4376381805 [==============================] - 75s 0us/step
4376395776/4376381805 [==============================] - 75s 0us/step
```

In [ ]:
```
PATH = os.path.abspath('.') + '/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir'
```

In [ ]:
```
## https://www.tensorflow.org/tutorials/load_data/images
import pathlib

image_dir = pathlib.Path(PATH)
```

In [ ]:
```
print(image_dir)
```

```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir
```

In [ ]:
```
imageList = list(image_dir.glob('*.jpg'))
print(len(imageList))
print(imageList[0])
Image.open(imageList[0])
```

```
31783
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_278519448.jpg
```

Out[ ]:

In [ ]:
```
# Download caption annotation files
annotation_folder = '/drive/My Drive/Colab Notebooks/NN/EX3/annotations/'
#if not os.path.exists(os.path.abspath('.') + annotation_folder):
annotation_zip = tf.keras.utils.get_file('train_captions.csv',
                                         cache_subdir = os.path.abspath('.') + annotation_folder,
                                         origin = 'https://www.dropbox.com/s/qqrsitwecf7fjd3/train_captions.csv?dl=1',
                                         extract = False)
annotation_file = os.path.dirname(annotation_zip) + "/train_captions.csv"
os.remove(annotation_zip)
```

In [ ]:
```
# Download caption annotation files
annotation_folder = '/drive/My Drive/Colab Notebooks/NN/EX3/annotations/'
#if not os.path.exists(os.path.abspath('.') + annotation_folder):
test_zip = tf.keras.utils.get_file('test_images.csv',
                                   cache_subdir = os.path.abspath('.') + annotation_folder,
                                   origin = 'https://www.dropbox.com/s/ahtgk6isemsqldf/test_images.csv?dl=1',
                                   extract = False)
test_file = os.path.dirname(test_zip) + "/test_images.csv"
os.remove(test_zip)
```

```
Downloading data from https://www.dropbox.com/s/ahtgk6isemsqldf/test_images.csv?dl=1
32768/29242 [================================] - 0s 0us/step
40960/29242 [========================================] - 0s 0us/step
```

In [ ]:
```
annotation_file = "/content/drive/My Drive/Colab Notebooks/NN/EX3/annotations/train_captions.csv"
```

In [ ]:
```
## Structure the data for the rest of the cells to run as are
## Every line in the file has the following form:
## <IMAGE NAME> | <CAPTION NUMBER> | <CAPTION TEXT>
annotations = []
with open(annotation_file, 'r', encoding='utf8') as f:
    for line in f:
        imageID, _, captionText = line.split('|')
        imageID = imageID.strip()
        captionText = captionText.strip()
        annotations.append({"imageID" : imageID, "caption" : captionText})
```

In [ ]:
```
len(annotations)
```

Out[ ]:
```
148915
```

In [ ]:
```
test = random.choice(annotations)
allCaptions = [i["caption"] for i in annotations if i["imageID"] == test["imageID"]]
print(allCaptions)
```

Out[ ]:
```
['A man in a blue collared shirt reaches to his back , his face obscured , in front of a body of water containing a sail boat , illuminated by a pink-tinted sky .',
 'A man walking down the pier scratching his back with a boat sailing in the background as the sun sets over the water .',
 'A man is scratching his back while walking along the shore .',
 'A boat selling along side an ocean front sidewalk .',
 'a man takes a walk on the pier .']
```

In [ ]:
```
# Group all captions together having the same image ID.
```

```
image_path_to_caption = collections.defaultdict(list)
for entry in annotations:
    caption = f"<start> {entry['caption']} <end>"
    image_path = PATH + '/' + entry['imageID']
    image_path_to_caption[image_path].append(caption)
```

Επειδή οι χρόνοι εκπαίδευσης των νευρωνικών δικτύων σε αυτήν την εργασία είναι αρκετά μεγάλοι, και χρησιμοποιούνται checkpoints και σταδιακή εκπαίδευση, προκειμένου αυτή να βασίζεται στο ίδιο σύνολο δεδομένων σε κάθε βήμα (όπως είναι λογικό), οπουδήποτε εισάγεται τυχαιότητα θα χρησιμοποιείται το ίδιο random seed.

In [ ]:
```
from random import Random

rnd = Random(31)
image_paths = list(image_path_to_caption.keys())
rnd.shuffle(image_paths)

# Select the first 6000 image_paths from the shuffled set.
# Approximately each image id has 5 captions associated with it, so that will
# lead to 30,000 examples.
train_image_paths = image_paths[:6000]
print(len(train_image_paths))
```

```
6000
```

In [ ]:
```
train_captions = []
img_name_vector = []

for image_path in train_image_paths:
    caption_list = image_path_to_caption[image_path]
    train_captions.extend(caption_list)
    img_name_vector.extend([image_path] * len(caption_list))
```

In [ ]:
```
print(train_captions[31])
Image.open(img_name_vector[31])
```

```
<start> The adult is covering their face with a piece of clothing outside <end>
```

Out[ ]:

Έτοιμο Δίκτυο

Προκειμένου να έχουμε σημείο αναφοράς για τις επιδόσεις, πρώτα θα εκπαιδεύσουμε το δίκτυο ακριβώς όπως και στο tutorial, με μοναδική αλλαγή το dataset.

Preprocess the images using InceptionV3

In [ ]:
```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(299, 299)(img)
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
```

Initialize InceptionV3 and load the pretrained Imagenet weights

In [ ]:
```
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.keras.Model(new_input, hidden_layer)
```

In [ ]:
```
len(os.listdir(PATH))
```

Out[ ]:
```
37784
```

Caching the features extracted from InceptionV3

In [ ]:
```
%%time
from tqdm import tqdm

# Get unique images
encode_train = sorted(set(img_name_vector))

# Feel free to change batch_size according to your system configuration
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(
    load_image, num_parallel_calls=tf.data.experimental.AUTOTUNE).batch(16) # edw evala .experimental

for img, path in tqdm(image_dataset):
    batch_features = image_features_extract_model(img)
    batch_features = tf.reshape(batch_features,
                                (batch_features.shape[0], -1, batch_features.shape[3]))

    for bf, p in zip(batch_features, path):
        path_of_feature = p.numpy().decode("utf-8")
        np.save(path_of_feature, bf.numpy())
```

```
100%|██████████| 375/375 [01:22<00:00,  4.54it/s]
Wall time: 1min 22s
```

Preprocess and tokenize the captions

In [ ]:
```
%%time
caption_dataset = tf.data.Dataset.from_tensor_slices(train_captions)

# We will override the default standardization of TextVectorization to preserve
# "<>" characters, so we preserve the tokens for the <start> and <end>.
def standardize(inputs):
    inputs = tf.strings.lower(inputs)
    return tf.strings.regex_replace(inputs,
                                    r"!\"#$%&\(\)\*\+.,-/:;=?@\[\\\]^_`{|}~", "")

# Max word count for a caption.
max_length = 50
# Use the top 5000 words for a vocabulary.
vocabulary_size = 5000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    output_sequence_length=max_length)
# Learn the vocabulary from the caption data.
tokenizer.adapt(caption_dataset)
```

```
Wall time: 49 s
```

In [ ]:
```
# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))
```

In [ ]:
```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary(),
    invert = True)
```

Split the data into training and testing

In [ ]:
```
%%time
img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(img_name_vector, cap_vector):
    img_to_cap_vector[img].append(cap)

# Create training and validation sets using an 80-20 split randomly.
img_keys = list(img_to_cap_vector.keys())
rnd.shuffle(img_keys)

slice_index = int(len(img_keys)*0.8)
img_name_train_keys, img_name_val_keys = img_keys[:slice_index], img_keys[slice_index:]

img_name_train = []
cap_train = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    img_name_train.extend([imgt] * capt_len)
    cap_train.extend(img_to_cap_vector[imgt])

img_name_val = []
cap_val = []
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    img_name_val.extend([imgv] * capv_len)
    cap_val.extend(img_to_cap_vector[imgv])
```

```
Wall time: 6.31 s
```

In [ ]:
```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

Out[ ]:
```
(24000, 24000, 6000, 6000)
```

Create a tf.data dataset for training

In [ ]:
```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from InceptionV3 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 64
```

In [ ]:
```
# Load the numpy files
def map_func(img_name, cap):
    img_tensor = np.load(img_name.decode('utf-8')+'.npy')
    return img_tensor, cap
```

In [ ]:
```
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int64]),
          num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model

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

    def call(self, features, hidden):
```

```
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [ ]:
```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

In [ ]:
```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)

        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
```

```
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

In [ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Checkpoint

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_pathb
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

In [ ]:
```
print(start_epoch)
```

```
4
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
@tf.function
def train_step(img_tensor, target):
    loss = 0

    # initializing the hidden state for each batch
    # because the captions are not related from image to image
    hidden = decoder.reset_state(batch_size=target.shape[0])

    dec_input = tf.expand_dims([word_to_index('<start>')] * target.shape[0], 1)

    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)

            loss += loss_function(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)

    total_loss = (loss / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, trainable_variables)

    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss
```

In [ ]:
```
%%time
EPOCHS = 20
```

```python
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        #if batch % 100 == 0:
        #    average_batch_loss = batch_loss.numpy()/int(target.shape[1])
        #    print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f}')
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ckpt_manager.save()

    print(f'Epoch {epoch+1} Loss {total_loss/num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

```
100%|██████████| 375/375 [06:27<00:00,  1.03s/it]
Epoch 1 Loss 1.269665
Time taken for 1 epoch 387.58 sec

100%|██████████| 375/375 [05:13<00:00,  1.20it/s]
Epoch 2 Loss 1.008131
Time taken for 1 epoch 314.00 sec

100%|██████████| 375/375 [05:32<00:00,  1.13it/s]
Epoch 3 Loss 0.922189
Time taken for 1 epoch 332.54 sec

100%|██████████| 375/375 [05:27<00:00,  1.14it/s]
Epoch 4 Loss 0.863198
Time taken for 1 epoch 328.11 sec

100%|██████████| 375/375 [05:29<00:00,  1.14it/s]
Epoch 5 Loss 0.813685
Time taken for 1 epoch 329.78 sec

100%|██████████| 375/375 [05:38<00:00,  1.11it/s]
Epoch 6 Loss 0.770569
Time taken for 1 epoch 338.74 sec

100%|██████████| 375/375 [05:32<00:00,  1.13it/s]
Epoch 7 Loss 0.730728
Time taken for 1 epoch 332.70 sec

100%|██████████| 375/375 [05:41<00:00,  1.10it/s]
Epoch 8 Loss 0.692568
Time taken for 1 epoch 341.27 sec

100%|██████████| 375/375 [05:35<00:00,  1.12it/s]
Epoch 9 Loss 0.656523
Time taken for 1 epoch 335.50 sec

100%|██████████| 375/375 [05:35<00:00,  1.12it/s]
Epoch 10 Loss 0.623185
Time taken for 1 epoch 336.00 sec

100%|██████████| 375/375 [05:35<00:00,  1.12it/s]
Epoch 11 Loss 0.591871
Time taken for 1 epoch 336.04 sec

100%|██████████| 375/375 [05:34<00:00,  1.12it/s]
Epoch 12 Loss 0.561048
Time taken for 1 epoch 334.60 sec

100%|██████████| 375/375 [05:38<00:00,  1.11it/s]
Epoch 13 Loss 0.533337
Time taken for 1 epoch 338.78 sec

100%|██████████| 375/375 [05:29<00:00,  1.14it/s]
Epoch 14 Loss 0.507034
Time taken for 1 epoch 329.31 sec

100%|██████████| 375/375 [05:38<00:00,  1.11it/s]
```

```
Epoch 15 Loss 0.482123
Time taken for 1 epoch 338.74 sec

100%|██████████| 375/375 [05:31<00:00,  1.13it/s]
Epoch 16 Loss 0.460488
Time taken for 1 epoch 332.15 sec

100%|██████████| 375/375 [05:26<00:00,  1.15it/s]
Epoch 17 Loss 0.439154
Time taken for 1 epoch 326.48 sec

100%|██████████| 375/375 [05:35<00:00,  1.12it/s]
Epoch 18 Loss 0.420027
Time taken for 1 epoch 336.21 sec

100%|██████████| 375/375 [05:30<00:00,  1.14it/s]
Epoch 19 Loss 0.402836
Time taken for 1 epoch 330.35 sec

100%|██████████| 375/375 [05:34<00:00,  1.12it/s]
Epoch 20 Loss 0.383438
Time taken for 1 epoch 334.92 sec

Wall time: 1h 51min 53s
```

In [ ]:
```
start_epoch = 20
```

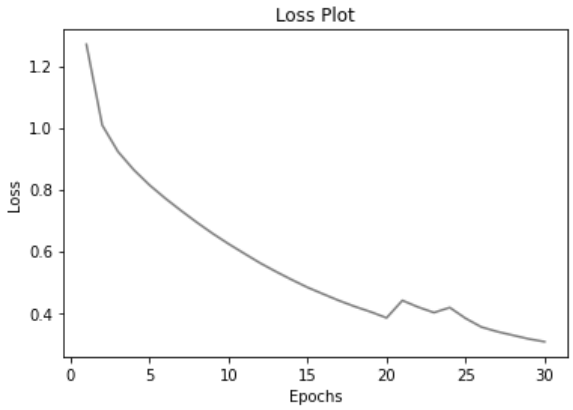
In [ ]:
```
from tqdm import tqdm
```

In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        #if batch % 100 == 0:
        #    average_batch_loss = batch_loss.numpy()/int(target.shape[1])
        #    print(f'Epoch {epoch+1} Batch {batch} Loss {average_batch_loss:.4f}')
    # storing the epoch end loss value to plot later
    # loss_plot.append(total_loss / num_steps)

    ckpt_manager.save()

    print(f'Epoch {epoch+1} Loss {total_loss/num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time()-start:.2f} sec\n')
```

```
100%|██████████| 375/375 [02:49<00:00,  2.21it/s]
Epoch 21 Loss 0.439877
Time taken for 1 epoch 169.62 sec

100%|██████████| 375/375 [01:43<00:00,  3.62it/s]
Epoch 22 Loss 0.418337
Time taken for 1 epoch 103.85 sec

100%|██████████| 375/375 [01:42<00:00,  3.64it/s]
Epoch 23 Loss 0.400571
Time taken for 1 epoch 103.16 sec

100%|██████████| 375/375 [01:41<00:00,  3.70it/s]
Epoch 24 Loss 0.416929
Time taken for 1 epoch 101.72 sec

100%|██████████| 375/375 [01:42<00:00,  3.66it/s]
Epoch 25 Loss 0.381929
Time taken for 1 epoch 102.62 sec

100%|██████████| 375/375 [01:42<00:00,  3.66it/s]
Epoch 26 Loss 0.353806
Time taken for 1 epoch 102.73 sec

100%|██████████| 375/375 [01:43<00:00,  3.64it/s]
```

```
Epoch 27 Loss 0.338955
Time taken for 1 epoch 103.43 sec

100%|██████████| 375/375 [01:42<00:00,  3.64it/s]
Epoch 28 Loss 0.327033
Time taken for 1 epoch 103.29 sec

100%|██████████| 375/375 [01:42<00:00,  3.66it/s]
Epoch 29 Loss 0.315220
Time taken for 1 epoch 102.68 sec

100%|██████████| 375/375 [01:42<00:00,  3.66it/s]
Epoch 30 Loss 0.306210
Time taken for 1 epoch 102.70 sec
```

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



Evaluation of Captions

In [ ]:
```
def evaluate(image):
    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0],
                                                 -1,
                                                 img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([word_to_index('<start>')], 0)
    result = []

    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input,
                                                         features,
                                                         hidden)

        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

        predicted_id = tf.random.categorical(predictions, 1)[0][0].numpy()
        predicted_word = tf.compat.as_text(index_to_word(predicted_id).numpy())


        if predicted_word == '<end>':
            return result, attention_plot

        ## CHECK FOR <end> BEFORE APPENDING
        result.append(predicted_word)
        dec_input = tf.expand_dims([predicted_id], 0)
```

```
    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot
```

In [ ]:
```
def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for i in range(len_result):
        temp_att = np.resize(attention_plot[i], (8, 8))
        grid_size = max(int(np.ceil(len_result/2)), 2)
        ax = fig.add_subplot(grid_size, grid_size, i+1)
        ax.set_title(result[i])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

    plt.tight_layout()
    plt.show()
```

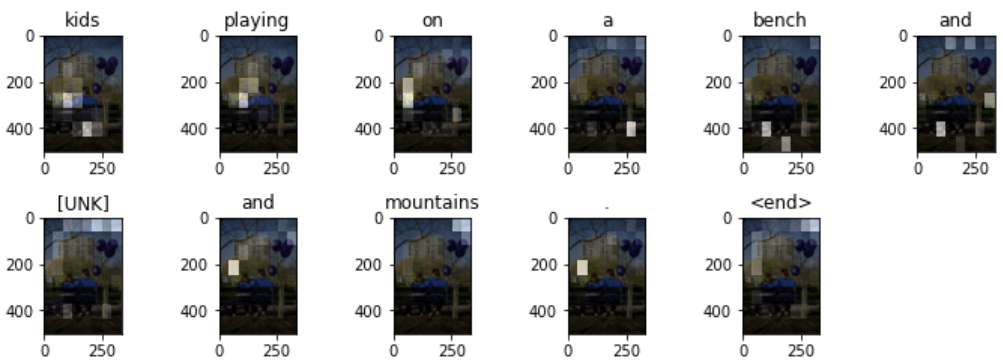
BLEU Scores of the Train Validation Set

In [ ]:
```
utilDict = {}
## by the end of the loop below, utilDict will have all the relevant image ids as keys, and a list of all the correspond
with open(annotation_file, 'r', encoding = 'utf8') as f:
    for line in f:
        imageID, _, captionText = line.split('|')
        imageID = imageID.strip()
        captionText = captionText.strip()
        if(imageID not in utilDict):
            utilDict.update({imageID : []})
        utilDict.get(imageID).append(captionText)
```

In [ ]:
```
# captions on the validation set
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu, SmoothingFunction

weights = (0.4, 0.3, 0.2, 0.1)

## From https://www.nltk.org/_modules/nltk/translate/bleu_score.html
## sentence_bleu
## :param references: reference sentences
## :type references: list(list(str))
## :param hypothesis: a hypothesis sentence
## :type hypothesis: list(str)
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

def randomCheck(rid):
    #random.choice(list(utilDict.keys()))
    image = img_name_val[rid]
    real_caption = ' '.join([tf.compat.as_text(index_to_word(i).numpy())
                             for i in cap_val[rid] if i not in [0]])
    result, attention_plot = evaluate(image)

    imgkey = image.split("/")[-1]
    references = [c.split() for c in utilDict[imgkey]]

    print('REAL CAPTIONS:', '\n'.join(utilDict[imgkey]))
    print('PREDICTION CAPTION:', ' '.join(result))
    print('Sentence BLEU score: ', sentence_bleu(references = references, hypothesis = result, weights = weights, smooth
    plot_attention(image, result, attention_plot)
```

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a couple is seated on a park bench near some blue balloons . <end>
Prediction Caption: kids playing on a bench and [UNK] and mountains . <end>
Sentence BLEU score:  1.0741468747791084e-93
```



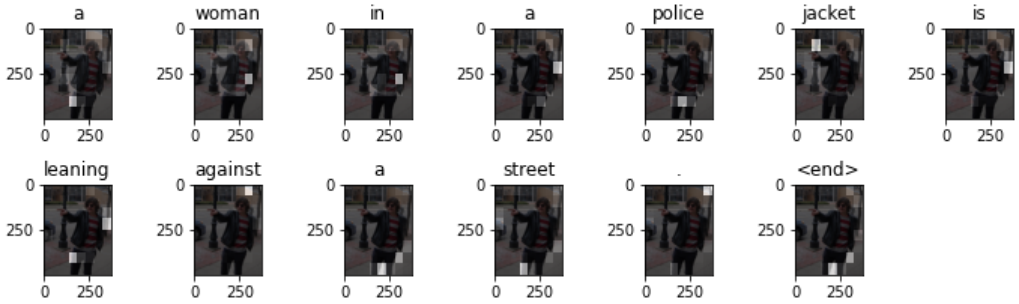
C:/Users/anton/Documents/hmmy_mathimata/neurwnika/ergasia_3./image_dir/_460748661.jpg

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a man with dark glasses , a black jacket , and a white and red [UNK] shirt pointing at something w
hile standing on a sidewalk . <end>
Prediction Caption: a woman in a police jacket is leaning against a street . <end>
Sentence BLEU score:  6.428695236386838e-186
```



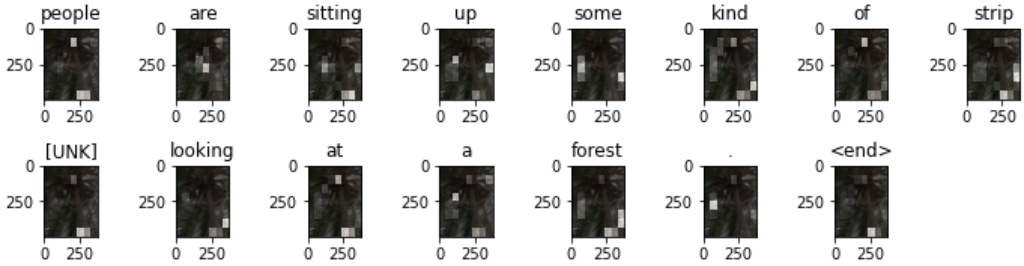
C:/Users/anton/Documents/hmmy_mathimata/neurwnika/ergasia_3./image_dir/_94578381.jpg

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a man is climbing up a coconut tree while holding coconut . <end>
Prediction Caption: people are sitting up some kind of strip [UNK] looking at a forest . <end>
Sentence BLEU score:  7.173174488978214e-186
```




```
C:/Users/anton/Documents/hmmy_mathimata/neurwnika/ergasia_3./image_dir/_340139192.jpg
```

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
## entire val set

## :param list_of_references: a corpus of lists of reference sentences, w.r.t. hypotheses
## :type list_of_references: list(list(list(str)))
## :param hypotheses: a list of hypothesis sentences
## :type hypotheses: list(list(str))
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 1200/1200 [06:52<00:00,  2.91it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.08326342410126537
```

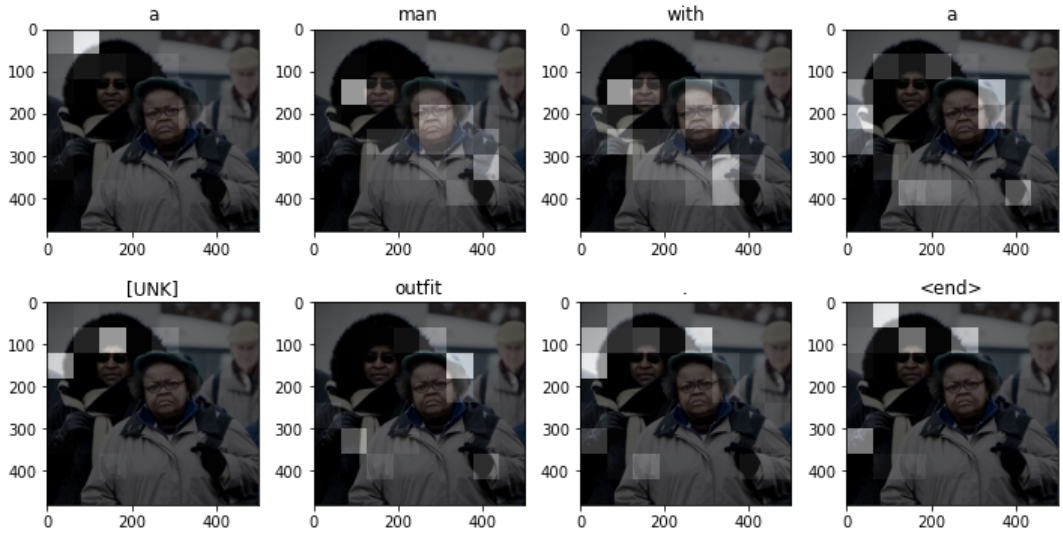
Samples from the Test Set

In [ ]:
```
testList = []
## in this one each line is an image id
with open(test_file, 'r') as f:
    for line in f: testList.append(line.strip())
```

In [ ]:
```
def randomTestCheck(randomImage):
    image = os.path.abspath('.') + "/image_dir/" + randomImage
    result, attention_plot = evaluate(image)

    print('Prediction Caption:', ' '.join(result))
    plot_attention(image, result, attention_plot)
```

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man with a [UNK] outfit . <end>
```



In [ ]:
```
Image.open(os.path.abspath('.') + "/image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a woman wearing a black black and black suit [UNK] and a black leather jacket and tan jacket looking at an electronic device next to a portfolio . <end>
```

a	woman	wearing	a	black	black	and	black	suit	[UNK]	and	a	black	leather	jacket
250	250	250	250	250	250	250	250	250	250	250	250	250	250	250

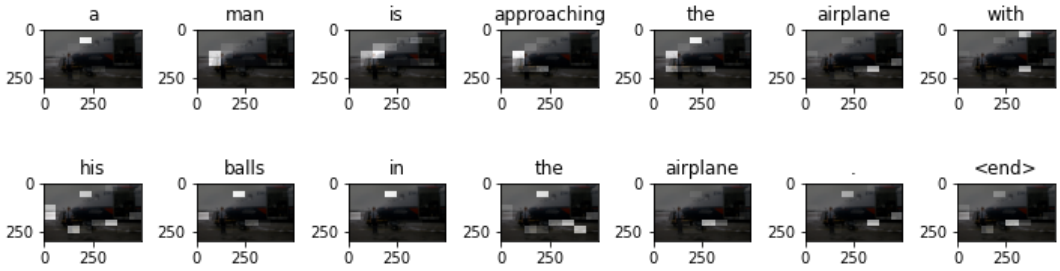
and	tan	jacket	looking	at	an	electronicdevice	next	to	a	portfolio	.	<end>
250	250	250	250	250	250	250	250	250	250	250	250	250

In [ ]:
```
Image.open(os.path.abspath('.') + "/image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man is approaching the airplane with his balls in the airplane . <end>
```



In [ ]:
```
Image.open(os.path.abspath('.') + "/image_dir/" + randomImage)
```

Out[ ]:

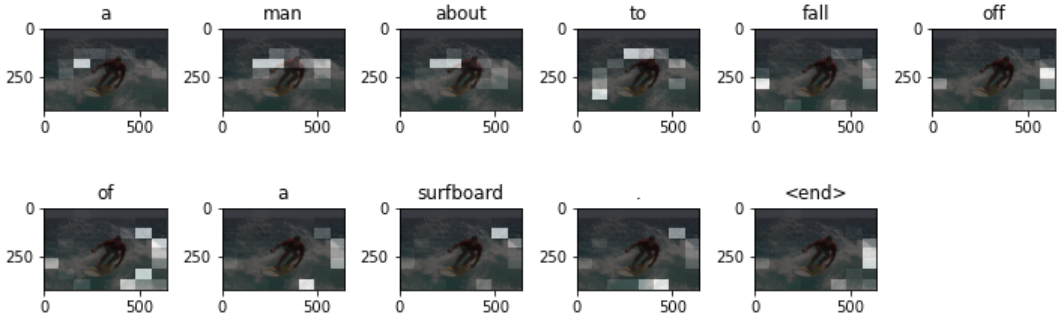
In [ ]:
```
## predictions for entire test set

testPreds = [evaluate(os.path.abspath('.') + "/image_dir/" + i)[0] for i in testList]
```

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)
```

```
result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: a man about to fall off of a surfboard . <end>
```



Out[ ]:

Improvements

Α' Βελτίωση με το προεκπαιδευμένο δίκτυο ResNet152V2

https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet_v2
https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet_v2/ResNet152V2

Η πρώτη δοκιμή βελτίωσης συνίσταται στην αύξηση των δεδομένων εκπαίδευσης κατά 50% (9000 αντί για 6000 εικόνες) και στην αντικατάσταση του συνελικτικού δικτύου - encoder που χρησιμοποιείται από το tutorial (InceptionV3), με το ResNet152V2, όπως υπαγορεύει η εκφώνηση για την ομάδα μας. $31 \equiv 3 \mod 4$

In [ ]:
```
# Select the first 9000 image_paths from the shuffled set.
# Approximately each image id has 5 captions associated with it, so that will
# lead to 45,000 examples.
train_image_paths = image_paths[:9000]
print(len(train_image_paths))
```

```
9000
```

In [ ]:
```
train_captions = []
img_name_vector = []

for image_path in train_image_paths:
    caption_list = image_path_to_caption[image_path]
```

```
    train_captions.extend(caption_list)
    img_name_vector.extend([image_path] * len(caption_list))
```

Preprocess the images using ResNet152V2

In [ ]:
```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(224, 224)(img)
    ## preprocessing with proper method
    img = tf.keras.applications.resnet_v2.preprocess_input(img)
    return img, image_path
```

Initialize ResNet152V2 and load the pretrained Imagenet weights

In [ ]:
```
image_model = tf.keras.applications.resnet_v2.ResNet152V2(include_top = False, weights = 'imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet152v2_weights_tf_dim_ordering_tf_kernels_notop.h5
234553344/234545216 [==============================] - 2s 0us/step
234561536/234545216 [==============================] - 2s 0us/step
```

Caching the features extracted from ResNet152V2

In [ ]:
```
%%time
from tqdm import tqdm

# Get unique images
encode_train = sorted(set(img_name_vector))

# Feel free to change batch_size according to your system configuration
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(load_image, num_parallel_calls = tf.data.AUTOTUNE).batch(16)

for img, path in tqdm(image_dataset):
    batch_features = image_features_extract_model(img)
    batch_features = tf.reshape(batch_features,
                                (batch_features.shape[0], -1, batch_features.shape[3]))

    for bf, p in zip(batch_features, path):
        path_of_feature = p.numpy().decode("utf-8")
        np.save(path_of_feature, bf.numpy())
```

```
100%|██████████| 563/563 [2:20:22<00:00, 14.96s/it]
CPU times: user 4h 4min 57s, sys: 7min 2s, total: 4h 11min 59s
Wall time: 2h 20min 22s
```

Preprocess and tokenize the captions

In [ ]:
```
%%time
caption_dataset = tf.data.Dataset.from_tensor_slices(train_captions)

# We will override the default standardization of TextVectorization to preserve
# "<>" characters, so we preserve the tokens for the <start> and <end>.
def standardize(inputs):
    inputs = tf.strings.lower(inputs)
    return tf.strings.regex_replace(inputs, r"!\"#$%&\(\)\*\+.,-/:;=?@\[\\\]^_`{|}~", "")

# Max word count for a caption.
max_length = 50
# Use the top 5000 words for a vocabulary.
vocabulary_size = 5000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    output_sequence_length=max_length)
# Learn the vocabulary from the caption data.
tokenizer.adapt(caption_dataset)
```

```
CPU times: user 38.6 s, sys: 2.52 s, total: 41.1 s
Wall time: 38.4 s
```

In [ ]:
```
# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))
```

In [ ]:
```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary(),
    invert = True)
```

Split the data into training and testing

In [ ]:
```
%%time
img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(img_name_vector, cap_vector):
    img_to_cap_vector[img].append(cap)

# Create training and validation sets using an 80-20 split randomly.
img_keys = list(img_to_cap_vector.keys())
rnd.shuffle(img_keys)

slice_index = int(len(img_keys) * 0.8)
img_name_train_keys, img_name_val_keys = img_keys[:slice_index], img_keys[slice_index:]

img_name_train = []
cap_train = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    img_name_train.extend([imgt] * capt_len)
    cap_train.extend(img_to_cap_vector[imgt])

img_name_val = []
cap_val = []
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    img_name_val.extend([imgv] * capv_len)
    cap_val.extend(img_to_cap_vector[imgv])
```

```
CPU times: user 25.9 s, sys: 2.32 s, total: 28.2 s
Wall time: 21.6 s
```

In [ ]:
```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

Out[ ]:
```
(36005, 36005, 9000, 9000)
```

Αυτό σημαίνει ότι έχουμε 9000 ζεύγη image - caption στο validation set (δηλαδή περίπου 1800 διαφορετικές εικόνες), και αναλόγως για το train set.

Create a tf.data dataset for training

In [ ]:
```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from ResNet152V2 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 49
```

In [ ]:
```
# Load the numpy files
def map_func(img_name, cap):
    ## this is correct, on my drive resnets npys are with the images
    img_tensor = np.load(img_name.decode('utf-8')+'.npy')
    return img_tensor, cap
```

In [ ]:
```
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int64]),
```

```
          num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE, seed = 31).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model

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

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [ ]:
```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

In [ ]:
```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
```

```
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)

        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

In [ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Checkpoint

Για την αναγκαία εκπαίδευση χρησιμοποιήθηκαν checkpoints αποθήκευσης της κατάστασης του μοντέλου, καθώς χρειάστηκαν πολλαπλά runtimes του notebook για την ολοκλήρωση αυτής. Ως εκ τούτου, μέρος του output log της εκπαίδευσης έχει διατηρηθεί σε μορφή κειμένου.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
@tf.function
def train_step(img_tensor, target):
    loss = 0

    # initializing the hidden state for each batch
    # because the captions are not related from image to image
    hidden = decoder.reset_state(batch_size = target.shape[0])

    dec_input = tf.expand_dims([word_to_index('<start>')] * target.shape[0], 1)

    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)
```

```
            loss += loss_function(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)

    total_loss = (loss / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, trainable_variables)

    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss
```

In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    ## loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 563/563 [1:13:19<00:00,  7.81s/it]
Epoch 26 Loss 0.272228
Time taken for 1 epoch 4400.28 sec

100%|██████████| 563/563 [1:12:21<00:00,  7.71s/it]
Epoch 27 Loss 0.263935
Time taken for 1 epoch 4342.67 sec

100%|██████████| 563/563 [1:16:22<00:00,  8.14s/it]
Epoch 28 Loss 0.262798
Time taken for 1 epoch 4583.05 sec

100%|██████████| 563/563 [1:17:22<00:00,  8.25s/it]
Epoch 29 Loss 0.259070
Time taken for 1 epoch 4642.87 sec

100%|██████████| 563/563 [1:17:42<00:00,  8.28s/it]
Epoch 30 Loss 0.253237
Time taken for 1 epoch 4663.41 sec
```

100%|██████████| 563/563 [1:11:13<00:00, 7.59s/it]
Epoch 1 Loss 0.821947
Time taken for 1 epoch 4274.95 sec

100%|██████████| 563/563 [1:16:21<00:00, 8.14s/it]
Epoch 2 Loss 0.772072
Time taken for 1 epoch 4582.77 sec

100%|██████████| 563/563 [1:12:06<00:00, 7.69s/it]
Epoch 3 Loss 0.725181
Time taken for 1 epoch 4327.51 sec

100%|██████████| 563/563 [1:12:21<00:00, 7.71s/it]
Epoch 4 Loss 0.683155
Time taken for 1 epoch 4342.64 sec

100%|██████████| 563/563 [1:11:43<00:00, 7.64s/it]

Epoch 5 Loss 0.644570
Time taken for 1 epoch 4304.36 sec

100%|██████████| 563/563 [1:38:37<00:00, 10.51s/it]
Epoch 6 Loss 0.608604
Time taken for 1 epoch 5917.92 sec

100%|██████████| 563/563 [1:46:21<00:00, 11.34s/it]
Epoch 7 Loss 0.575646
Time taken for 1 epoch 6382.83 sec

100%|██████████| 563/563 [1:11:35<00:00, 7.63s/it]
Epoch 8 Loss 0.545126
Time taken for 1 epoch 4295.81 sec

100%|██████████| 563/563 [1:12:50<00:00, 7.76s/it]
Epoch 9 Loss 0.516869
Time taken for 1 epoch 4371.37 sec

100%|██████████| 563/563 [1:12:21<00:00, 7.71s/it]
Epoch 10 Loss 0.490563
Time taken for 1 epoch 4342.88 sec

100%|██████████| 563/563 [1:12:22<00:00, 7.71s/it]
Epoch 11 Loss 0.466888
Time taken for 1 epoch 4343.31 sec

100%|██████████| 563/563 [1:12:21<00:00, 7.71s/it]
Epoch 12 Loss 0.445049
Time taken for 1 epoch 4343.11 sec

100%|██████████| 563/563 [1:17:56<00:00, 8.31s/it]
Epoch 13 Loss 0.425790
Time taken for 1 epoch 4677.59 sec

100%|██████████| 563/563 [1:12:38<00:00, 7.74s/it]
Epoch 14 Loss 0.406785
Time taken for 1 epoch 4359.64 sec

100%|██████████| 563/563 [1:12:48<00:00, 7.76s/it]
Epoch 15 Loss 0.390223
Time taken for 1 epoch 4369.39 sec

100%|██████████| 563/563 [1:24:21<00:00, 8.99s/it]
Epoch 16 Loss 0.373930
Time taken for 1 epoch 5063.62 sec

100%|██████████| 563/563 [1:17:40<00:00, 8.28s/it]
Epoch 17 Loss 0.360243
Time taken for 1 epoch 4661.34 sec

100%|██████████| 563/563 [1:26:21<00:00, 9.20s/it]
Epoch 18 Loss 0.347949
Time taken for 1 epoch 5182.92 sec

100%|██████████| 563/563 [1:22:21<00:00, 8.78s/it]
Epoch 19 Loss 0.332293
Time taken for 1 epoch 4942.67 sec

100%|██████████| 563/563 [1:22:21<00:00, 8.78s/it]
Epoch 20 Loss 0.320750
Time taken for 1 epoch 4942.62 sec

100%|██████████| 563/563 [1:21:21<00:00, 8.67s/it]
Epoch 21 Loss 0.309717
Time taken for 1 epoch 4882.70 sec

100%|██████████| 563/563 [1:25:21<00:00, 9.10s/it]
Epoch 22 Loss 0.302756
Time taken for 1 epoch 5122.83 sec

100%|██████████| 563/563 [1:23:22<00:00, 8.88s/it]
Epoch 23 Loss 0.302889
Time taken for 1 epoch 5003.08 sec

100%|██████████| 563/563 [1:24:25<00:00, 9.00s/it]
Epoch 24 Loss 0.293444
Time taken for 1 epoch 5066.41 sec

100%|██████████| 563/563 [1:23:51<00:00, 8.94s/it]
Epoch 25 Loss 0.283950
Time taken for 1 epoch 5032.61 sec

100%|██████████| 563/563 [1:13:19<00:00, 7.81s/it]
Epoch 26 Loss 0.272228
Time taken for 1 epoch 4400.28 sec

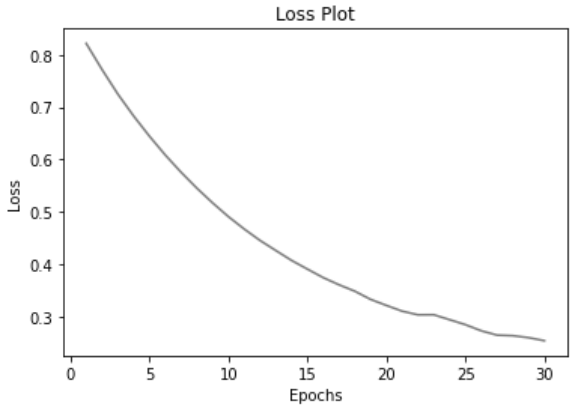
100%|██████████| 563/563 [1:12:21<00:00, 7.71s/it]
Epoch 27 Loss 0.263935
Time taken for 1 epoch 4342.67 sec

100%|██████████| 563/563 [1:16:22<00:00, 8.14s/it]
Epoch 28 Loss 0.262798
Time taken for 1 epoch 4583.05 sec

100%|██████████| 563/563 [1:17:22<00:00, 8.25s/it]
Epoch 29 Loss 0.259070
Time taken for 1 epoch 4642.87 sec

100%|██████████| 563/563 [1:17:42<00:00, 8.28s/it]
Epoch 30 Loss 0.253237
Time taken for 1 epoch 4663.41 sec

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



Ύστερα από 30 εποχές παρατηρούμε ικανοποιητική τιμή loss για το πρώτο αυτό πείραμα.

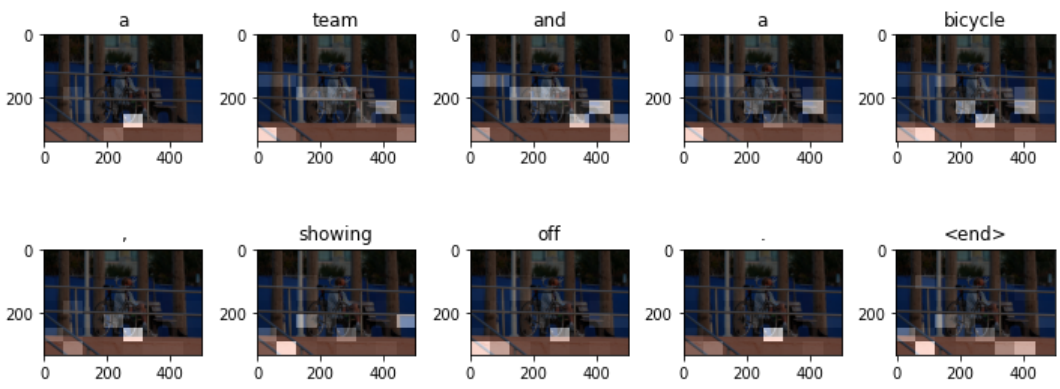
Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
# captions on the validation set
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu, SmoothingFunction

weights = (0.4, 0.3, 0.2, 0.1)

## From https://www.nltk.org/_modules/nltk/translate/bleu_score.html
## sentence_bleu
## :param references: reference sentences
## :type references: list(list(str))
## :param hypothesis: a hypothesis sentence
## :type hypothesis: list(str)
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

def randomCheck(rid):
    #random.choice(list(utilDict.keys()))
    image = img_name_val[rid]
    real_caption = ' '.join([tf.compat.as_text(index_to_word(i).numpy())
                             for i in cap_val[rid] if i not in [0]])
    result, attention_plot = evaluate(image)

    imgkey = image.split("/")[-1]
    references = [c.split() for c in utilDict[imgkey]]

    print('REAL CAPTIONS:', '\n'.join(utilDict[imgkey]))
    print('PREDICTION CAPTION:', ' '.join(result))
    print('Sentence BLEU score: ', sentence_bleu(references = references, hypothesis = result, weights = weights, smooth
    plot_attention(image, result, attention_plot)
```

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a man sitting on a bench with his bike parked at the rail . <end>
Prediction Caption: a team and a bicycle , showing off . <end>
Sentence BLEU score:  0.46462888317312373
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```



/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_257300996.jpg

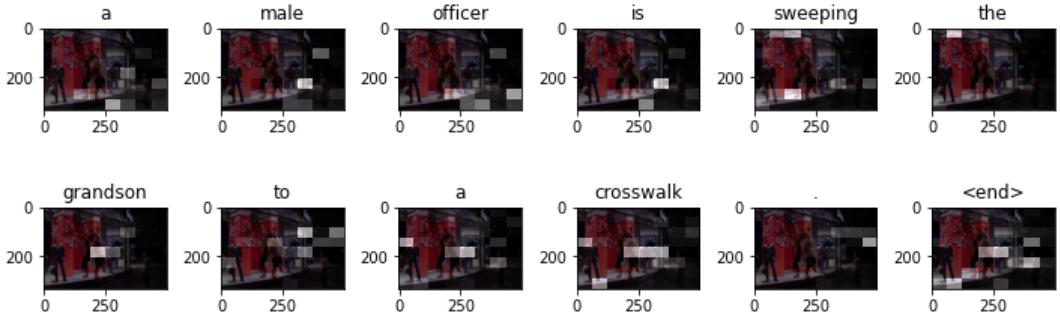
In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

Στο πείραμα αυτό, η περιγραφή που παρήχθη έλαβε το σχετικά υψηλό score 0.46, το οποίο σύμφωνα με την κλίμακα BLEU την χαρακτηρίζει ως ποιοτικά καλή. Στην πραγματικότητα, ωστόσο, δεν είναι καθόλου καλή, αφού έχει αναγνωρίσει μόνο την παρουσία ενός ποδηλάτου. Επομένως, πιθανώς για το πρόβλημα αυτό να μην αρκεί η μετρική BLEU.

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a woman looks at a store window display with mannequins and an image of a giant dog who looks like he is licking the window display . <end>
Prediction Caption: a male officer is sweeping the grandson to a crosswalk . <end>
Sentence BLEU score:  0.08023638017664907
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```



/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_25094965.jpg

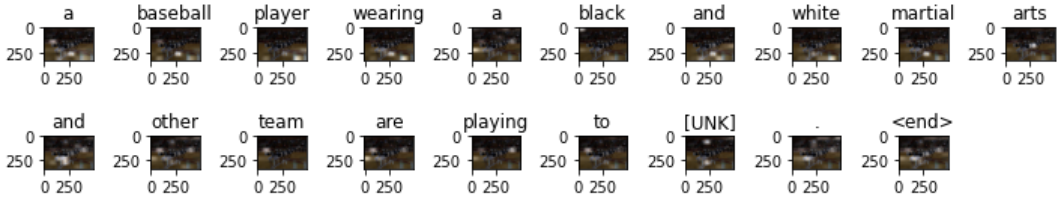
In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
Real Caption: <start> a cheerleading squad performing for a crowd at a sporting event . <end>
Prediction Caption: a baseball player wearing a black and white martial arts and other team are playing to [UNK] . <end>
Sentence BLEU score:  0.12140437992217352
```

```
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 3-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
```



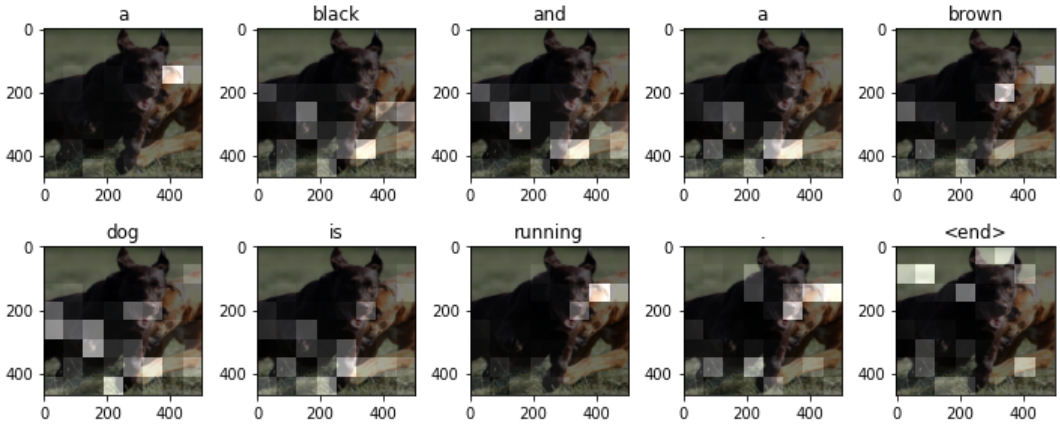

```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_464727873.jpg
```

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
## SAMPLE FROM TRAIN SET
result, attention_plot = evaluate(image)
randomCheck(31)
```

```
Real Caption: <start> two dogs are chasing each other on green grass . <end>
Prediction Caption: a black and a brown dog is running . <end>
Sentence BLEU score:  0.31850931710991254
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_217948755.jpg
```

Εξετάζοντας μεμονωμένα δείγματα παρατηρούμε σημαντική διακύμανση στην ποιότητα των captions. Ορισμένα πλησιάζουν υψηλά σκορ, ενώ άλλα πολύ χαμηλό. Δεν έχουν όλες οι εικόνες με καλό σκορ καλές περιγραφές. Φαίνεται πως το μοντέλο attention εντοπίζει με επιτυχία συγκεκριμένα υποκείμενα και αντικείμενα στις φωτογραφίες, αλλά δεν συνθέτει πάντα σωστή περιγραφή.

In [ ]:
```
## entire val set
## :param list_of_references: a corpus of lists of reference sentences, w.r.t. hypotheses
## :type list_of_references: list(list(list(str)))
## :param hypotheses: a list of hypothesis sentences
## :type hypotheses: list(list(str))
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

#allReferences = [[c.split() for c in utilDict[i]] for i in utilDict.keys()]
#allHypotheses = [evaluate(img_name_val[i])[0] for i in utilDict.keys()]
#allHypotheses = [evaluate(utilDict[i])[0] for i in utilDict.keys()] #utilDict[image.split("/")[-1]]
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.09576449897236472
```

Pickling because the cell above took 1 hour to run.

In [ ]:
```
with open(os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/pickles/all_references.pkl", 'wb') as f:
    pickle.dump(allReferences, f)

with open(os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/pickles/all_hypotheses.pkl", 'wb') as f:
    pickle.dump(allHypotheses, f)
```

In [ ]:
```
with open(os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/pickles/all_references.pkl", 'rb') as f:
    allReferences = pickle.load(f)
with open(os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/pickles/all_hypotheses.pkl", 'rb') as f:
    allHypotheses = pickle.load(f)
```

Samples from the Test Set

In [ ]:
```
test_file = "/content/drive/My Drive/Colab Notebooks/NN/EX3/annotations/test_images.csv"
```

In [ ]:
```
testList = []
## in this one each line is an image id
with open(test_file, 'r') as f:
    for line in f: testList.append(line.strip())
```

In [ ]:
```
def randomTestCheck(randomImage):
    image = "/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/" + randomImage
    result, attention_plot = evaluate(image)

    print('Prediction Caption:', ' '.join(result))
    plot_attention(image, result, attention_plot)
```

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man uses a large pot while on a pottery how to his [UNK] on a pink pitcher is smiling while on a b
lue with [UNK] something to a large fish well as a woman is using a large fish well , and a woman prepares to her hands
.
```

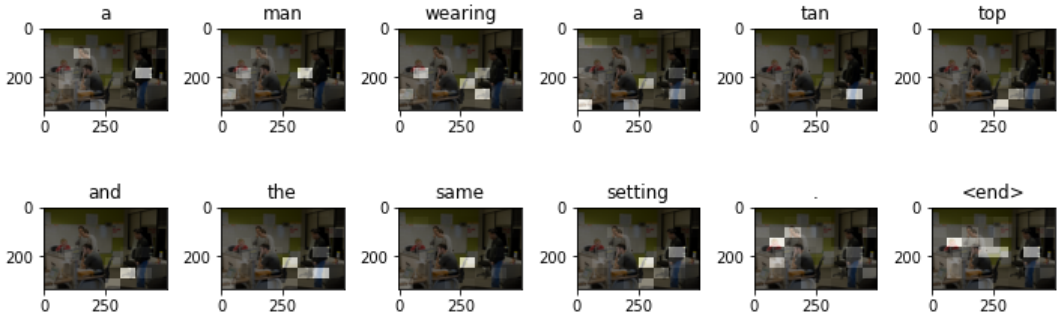
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:15: UserWarning: Tight layout not applied. tight_layout can
not make axes width small enough to accommodate all axes decorations
  from ipykernel import kernelapp as app
```

In [ ]:
```
Image.open("/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man wearing a tan top and the same setting . <end>
```

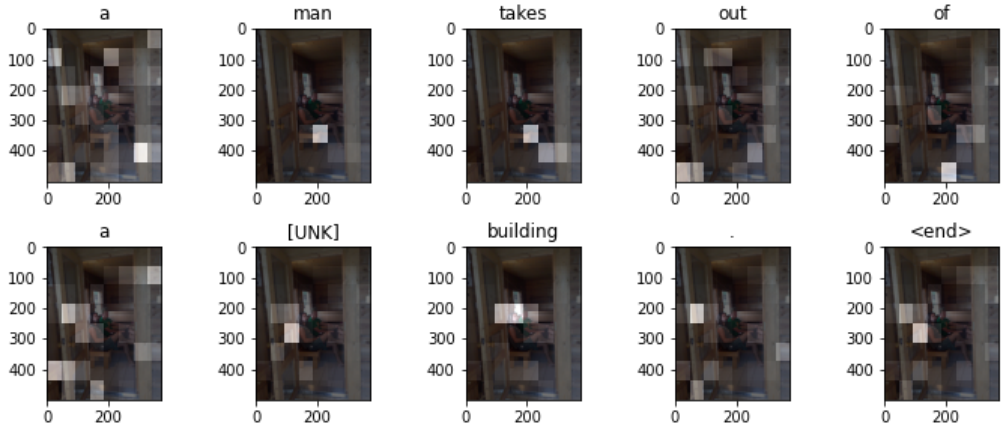


In [ ]:
```
Image.open("/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man takes out of a [UNK] building . <end>
```



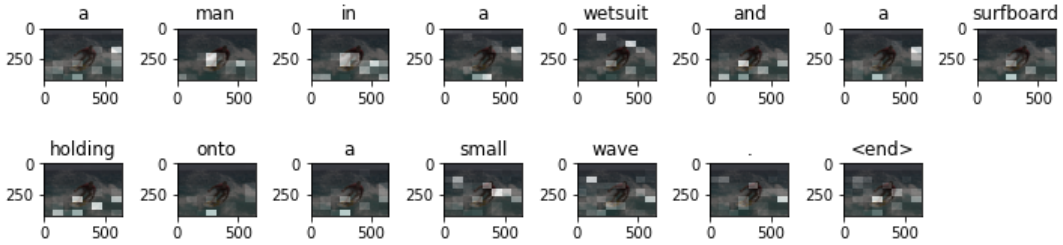
In [ ]:
```
Image.open("/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/" + randomImage)
```

Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Downloading data from https://tensorflow.org/images/surf.jpg
65536/64400 [==============================] - 0s 1us/step
73728/64400 [==============================] - 0s 0us/step
Prediction Caption: a man in a wetsuit and a surfboard holding onto a small wave . <end>
```



Out[ ]:

Β' Βελτίωση με προεπεξεργασία κειμένου

Δεύτερον, θα επιχειρήσουμε να βελτιώσουμε την ποιότητα των captions κάνοντας καλύτερη προεπεξεργασία των περιγραφών αναφοράς. Συγκεκριμένα:

- *θα ορίσουμε διαφορετικά φράγματα ελαχίστου και μεγίστου μήκους αποδεκτών captions,*
- *θα κανονικοποιήσουμε τα κείμενα, και*
- *θα αυξήσουμε το μέγιστο μέγεθος vocabulary διαφορετικών λέξεων σε 7,000. Αρχικά δοκιμάσαμε διπλασιασμό του vocabulary σε 10,000 και φάνηκε να βοηθά, όμως ο απαιτούμενος χρόνος ικανοποιητικής ελάττωσης του loss ήταν πολύ μεγάλος, και συνεπώς επιλέξαμε να χρησιμοποιήσουμε 7,000.*

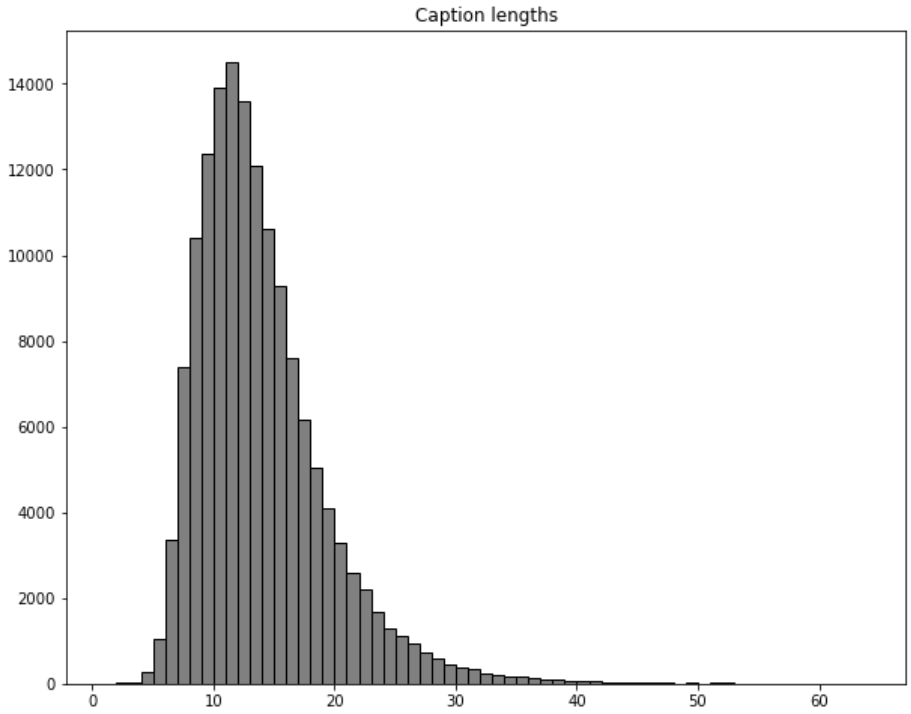
Επιπλέον, στο εξής θα χρησιμοποιούμε ακόμη περισσότερα δεδομένα για την εκπαίδευση (15,000 δείγματα συνολικά), καθώς είδαμε ότι η αύξηση δεδομένων βελτιώνει την επίδοση.

In [ ]:
```
train_image_paths = image_paths[:15000]
print(len(train_image_paths))
```

```
15000
```

In [ ]:
```
all_captions = [i["caption"] for i in annotations]
trainCaptionLengths = [len(i.split()) for i in all_captions]
plt.figure(figsize = (10, 8))
```

```
plt.hist(trainCaptionLengths, bins = np.linspace(start = 1, stop = 64, num = 64), color = "gray", edgecolor = "black")
plt.title("Caption lengths")
plt.show()
```



Όπως φαίνεται στο ανωτέρω ιστόγραμμα, η πλειοψηφία των περιγραφών έχουν μήκος από 6 έως 32 λέξεις, οπότε θα επιλεγεί αυτό το διάστημα πλάτους 27.

In [ ]:
```
train_captions = []
img_name_vector = []

for image_path in train_image_paths:
    caption_list = image_path_to_caption[image_path]
    caption_list = [i for i in caption_list if len(i.split()) >= 6 and len(i.split()) <= 32]
    train_captions.extend(caption_list)
    img_name_vector.extend([image_path] * len(caption_list))
```

Preprocess the images using ResNet152V2

Δεν χρειάζεται να επαναληφθεί η εξαγωγή των features.

Preprocess and tokenize the captions

Ως προς την κανονικοποίηση, επειδή το generative model που θα φτιάξουμε πρέπει να φτιάχνει φυσικές προτάσεις, χρειάζεται να παραμείνουν στο vocabulary όλα τα μέρη του λόγου, και συνεπώς δεν θα εφαρμόσουμε stemming ή αποκοπή λέξεων μικρού μήκους. Θα αφαιρέσουμε όμως, τα σημεία στίξης, εκτός των <, > που σηματοδοτούν αφετηρία και λήξη και του κόμματος και της τελείας που συνεισφέρουν νοηματικά στην νοηματική ουσία της πρότασης, και βρίσκονται σε αφθονία και στα reference captions. Η μετατροπή όλων των γραμμάτων σε πεζά γίνεται ήδη από το tutorial.

In [ ]:
```
# We will override the default standardization of TextVectorization to preserve
# "<>" characters, so we preserve the tokens for the <start> and <end>.
def standardize(inputs):
    res = tf.strings.lower(inputs)
    ## the following removes all characters except alphanumerics, whitespace, and <, >
    ## return tf.strings.regex_replace(res,r"!\"#$%&\(\)\*\+.,-/:;=?@\[\\\]^_`{|}~", "")
    res = tf.strings.regex_replace(res, r"[^a-zA-Z<> ,.]+", "")
    return res

# Max word count for a caption.
max_length = 50
## https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization
## το layer αυτό δεν έχει παράμετρο κάτω φράγματος, οπότε θα υλοποιηθεί χειροκίνητα
## limited_captions = [c for c in train_captions if len(c.split()) >= 7 and len(c.split()) <= 31]
caption_dataset = tf.data.Dataset.from_tensor_slices(train_captions)
```

```
# Use the top 5000 words for a vocabulary.
vocabulary_size = 5000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    output_sequence_length=max_length)
# Learn the vocabulary from the caption data.
tokenizer.adapt(caption_dataset)
```

In [ ]:
```
# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))
```

In [ ]:
```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary(),
    invert = True)
```

Split the data into training and testing

In [ ]:
```
%%time
img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(img_name_vector, cap_vector):
    img_to_cap_vector[img].append(cap)

# Create training and validation sets using an 80-20 split randomly.
img_keys = list(img_to_cap_vector.keys())
rnd.shuffle(img_keys)

slice_index = int(len(img_keys) * 0.8)
img_name_train_keys, img_name_val_keys = img_keys[:slice_index], img_keys[slice_index:]

img_name_train = []
cap_train = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    img_name_train.extend([imgt] * capt_len)
    cap_train.extend(img_to_cap_vector[imgt])

img_name_val = []
cap_val = []
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    img_name_val.extend([imgv] * capv_len)
    cap_val.extend(img_to_cap_vector[imgv])
```

```
Wall time: 15.1 s
```

In [ ]:
```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

Out[ ]:
```
(59225, 59225, 14788, 14788)
```

Επιτυγχάνουμε μείωση της διασποράς των μηκών, χάνοντας ελάχιστα δεδομένα.

Create a tf.data dataset for training

In [ ]:
```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from ResNet152V2 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 49
```

In [ ]:
```
# Load the numpy files
def map_func(img_name, cap):
    img_tensor = np.load("C:\\Users\\anton\\Documents\\hmmy_mathimata\\neurwnika\\ergasia_3\\resnet_features\\" + img_na
    return img_tensor, cap
```

In [ ]:
```
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int64]),
          num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE, seed = 31).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model

Δεν έχει γίνει αλλαγή του μοντέλου σε αυτό το πείραμα.

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

Checkpoint

Μεταφέρουμε το τελευταίο checkpoint σε νέο φάκελο και συνεχίζουμε την εκπαίδευση εκεί.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints3"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

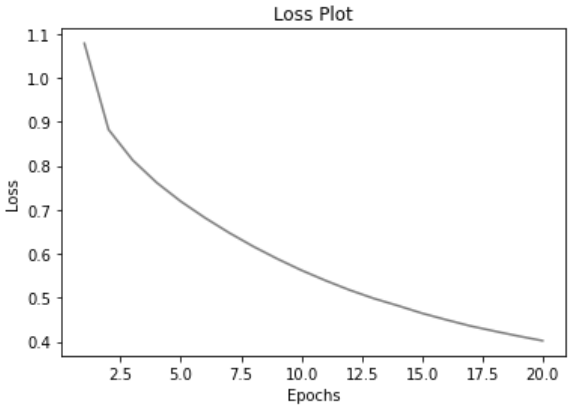
```
100%|██████████| 926/926 [05:24<00:00,  2.85it/s]
Epoch 1: Loss 1.077754
        Time taken 324.70 sec

100%|██████████| 926/926 [03:46<00:00,  4.10it/s]
Epoch 2: Loss 0.882146
        Time taken 226.44 sec

100%|██████████| 926/926 [03:45<00:00,  4.11it/s]
Epoch 3: Loss 0.812730
        Time taken 225.66 sec

100%|██████████| 926/926 [09:16<00:00,  1.66it/s]
```

```
Epoch 4: Loss 0.761414
        Time taken 556.96 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:28<00:00,  1.15it/s]
Epoch 5: Loss 0.718762
        Time taken 808.70 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:16<00:00,  1.16it/s]
Epoch 6: Loss 0.681591
        Time taken 796.50 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 7: Loss 0.647647
        Time taken 801.34 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 8: Loss 0.616584
        Time taken 800.93 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 9: Loss 0.588400
        Time taken 801.24 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:21<00:00,  1.16it/s]
Epoch 10: Loss 0.562378
        Time taken 801.87 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 11: Loss 0.539029
        Time taken 800.77 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 12: Loss 0.517428
        Time taken 800.42 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 13: Loss 0.498340
        Time taken 800.79 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 14: Loss 0.481960
        Time taken 800.59 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 15: Loss 0.464729
        Time taken 800.55 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 16: Loss 0.449944
        Time taken 801.00 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 17: Loss 0.435874
        Time taken 800.80 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 18: Loss 0.424085
        Time taken 800.97 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 19: Loss 0.412794
        Time taken 800.95 sec

100%|████████████████████████████████████████████████████████████| 926/926 [13:20<00:00,  1.16it/s]
Epoch 20: Loss 0.402323
        Time taken 800.88 sec

  3%|██                                                            | 30/926 [00:28<14:21,  1.04it/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8592/919303692.py in <module>
      7
      8     for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
----> 9         batch_loss, t_loss = train_step(img_tensor, target)
     10         total_loss += t_loss
     11

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\util\traceback_utils.py in error_handler(*args, **kwarg
s)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\def_function.py in __call__(self, *args, **kwds)
    908
    909       with OptionalXlaContext(self._jit_compile):
--> 910         result = self._call(*args, **kwds)
    911
    912       new_tracing_count = self.experimental_get_tracing_count()

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\def_function.py in _call(self, *args, **kwds)
    940       # In this case we have created variables on the first call, so we run the
    941       # defunned version which is guaranteed to never create variables.
--> 942       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    943     elif self._stateful_fn is not None:
    944       # Release the lock early so that multiple threads can perform the call

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in __call__(self, *args, **kwargs)
   3128       (graph_function,
   3129        filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 3130     return graph_function._call_flat(
   3131         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=protected-access
   3132

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in _call_flat(self, args, captured_inp
uts, cancellation_manager)
   1957         and executing_eagerly):
   1958       # No tape is watching; skip to running the function.
-> 1959       return self._build_call_outputs(self._inference_function.call(
   1960           ctx, args, cancellation_manager=cancellation_manager))
   1961     forward_backward = self._select_forward_and_backward_functions(

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in call(self, ctx, args, cancellation_
manager)
    596       with _InterpolateFunctionError(self):
    597         if cancellation_manager is None:
--> 598           outputs = execute.execute(
    599               str(self.signature.name),
    600               num_outputs=self._num_outputs,

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\execute.py in quick_execute(op_name, num_outputs,
 inputs, attrs, ctx, name)
     56   try:
     57     ctx.ensure_initialized()
---> 58     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     59                                         inputs, attrs, num_outputs)
     60   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

In [ ]:

```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



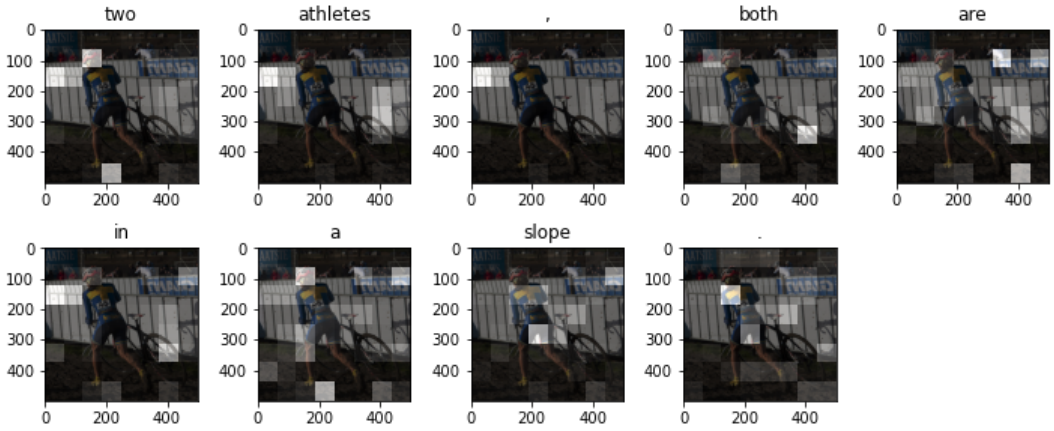
Σε αυτό το πείραμα διακόψαμε την εκπαίδευση νωρίτερα λόγω υπερβολικής αύξησης του χρόνου εκτέλεσης στις τελευταίες εποχές. Όμως, αξιολογώντας την ποιότητα των captions, διαπιστώσαμε ότι η προεπεξεργασία κειμένου πράγματι βελτιώνει την ποιότητα και ως εκ τούτου στο εξής θα την εφαρμόζουμε.

Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A bicyclist is walking her bike through deep mud in a race .
A woman walking bike in a mud trap during a bicycle race .
A woman is walking along side her bicycle during a race .
Cyclist number forty-eight treks through the mud .
A bicyclist is in the mud walking their bicycle .
PREDICTION CAPTION: two athletes , both are in a slope .
Sentence BLEU score:  0.09803694489756798
```

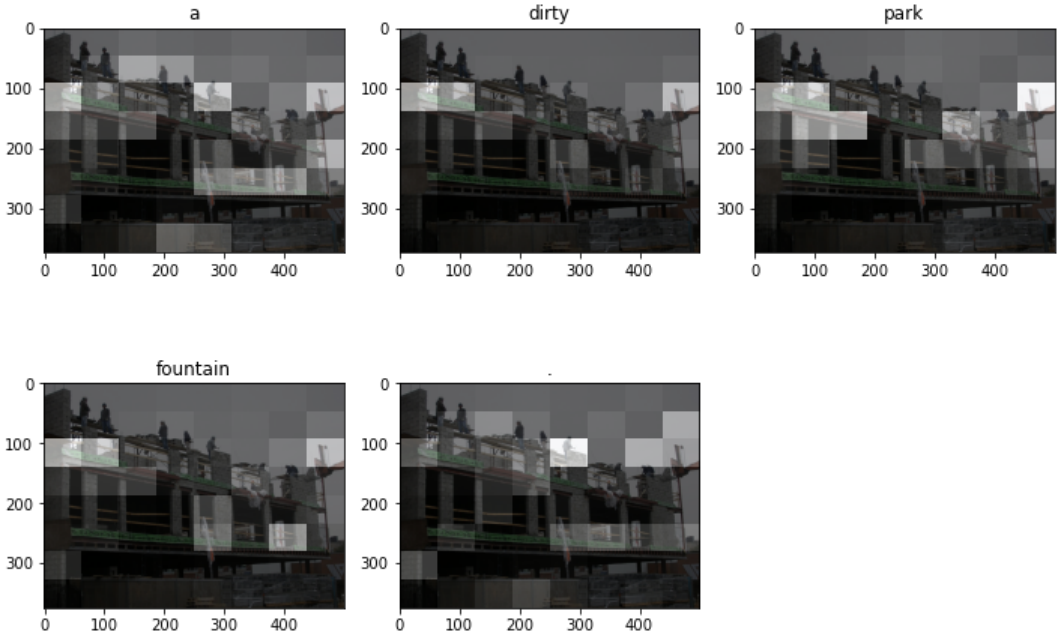


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: There are several construction workers at the top of a building which is in the early stages of construct
ion and has only gray bricks for walls .
Construction masons at work laying concrete blocks on a multistory building .
Construction workers add cinder blocks to the top of a structure .
The men are working on the cement building .
six workers building concrete building .
PREDICTION CAPTION: a dirty park fountain .
Sentence BLEU score:  0.0704400818994
```

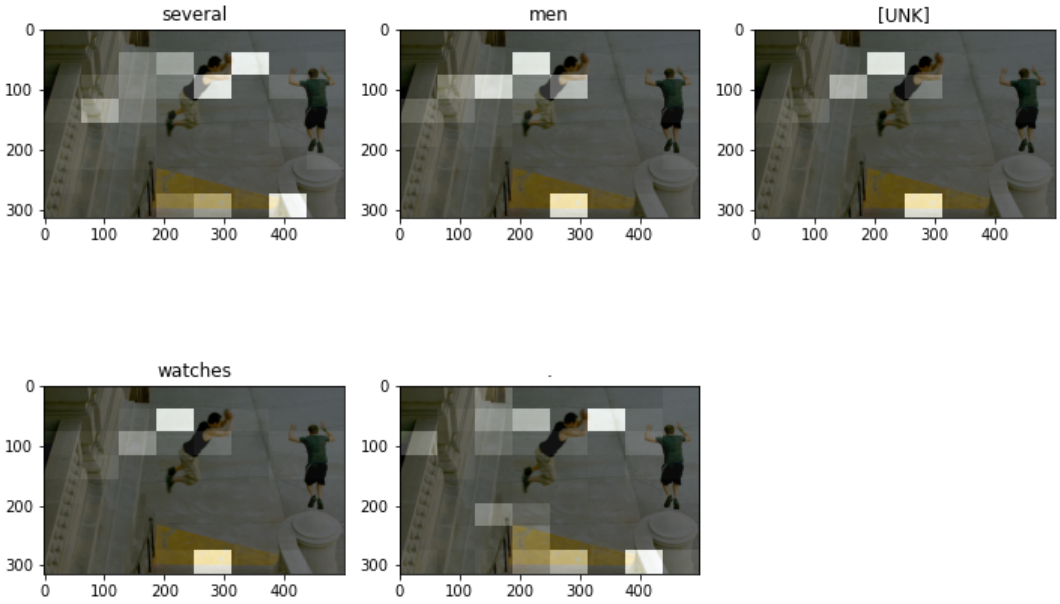


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Two men in shorts and short sleeves leap in the air .
Two athletic men doing parkour off of buildings .
two men jumping in the air for a picture .
Two men are jumping outside on concrete .
Two guys in shorts jumping .
PREDICTION CAPTION: several men [UNK] watches .
Sentence BLEU score:  0.0704400818994
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['a', 'brown', 'dog', 'is', 'climbing', 'the', 'top', 'of', 'a', 'coconut', 'tree', 'with', '[UNK]', '.']
```

In [ ]:
```
## :param list_of_references: a corpus of lists of reference sentences, w.r.t. hypotheses
## :type list_of_references: list(list(list(str)))
## :param hypotheses: a list of hypothesis sentences
## :type hypotheses: list(list(str))
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [21:31<00:00,  2.32it/s]
```

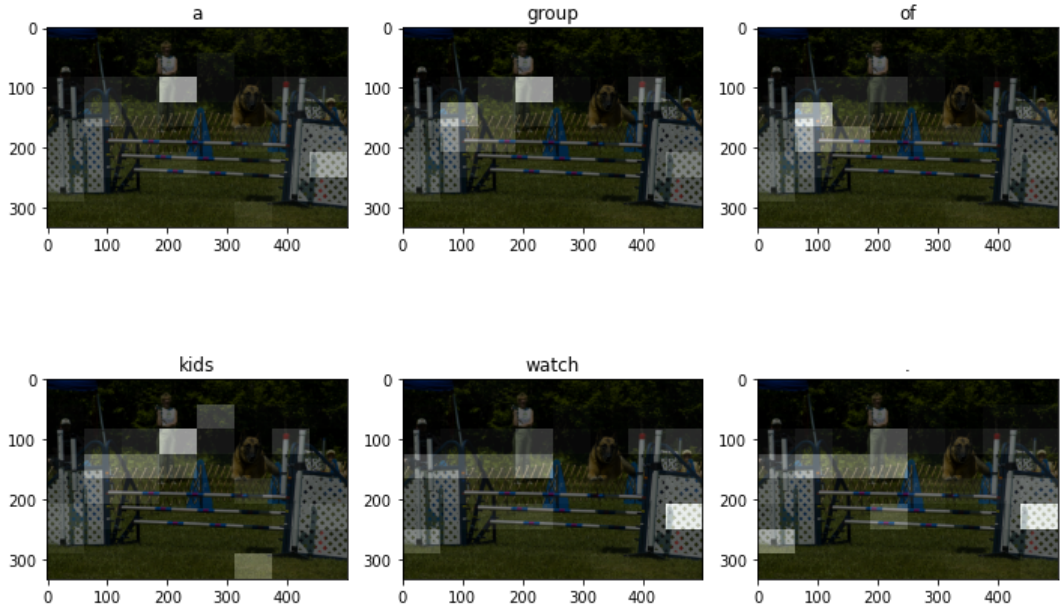
In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.08775300382288816
```

Samples from the Test Set

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a group of kids watch .
```

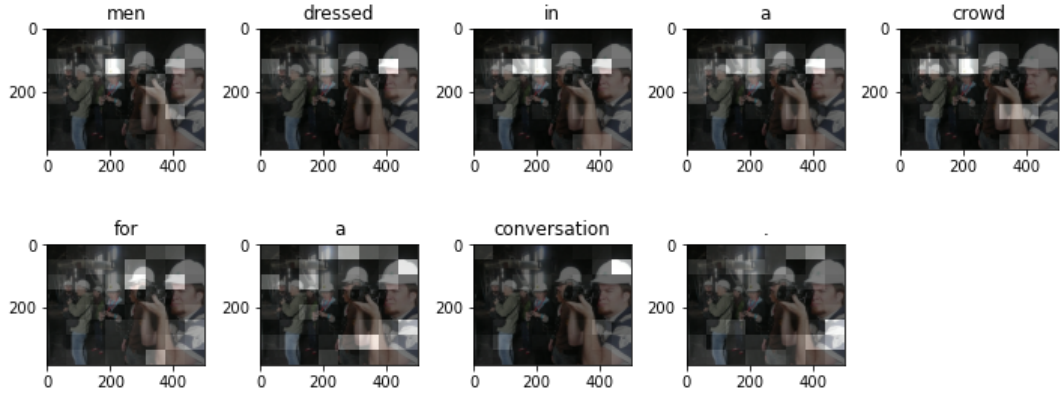


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: men dressed in a crowd for a conversation .
```



In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: several men are sitting in a blue and white and blue and blue and blue and blue , a black and blue a
nd white and black and blue and black and red light blue carpet .
```

```
C:\Users\anton\AppData\Local\Temp/ipykernel_8592/3853974765.py:15: UserWarning: Tight layout not applied. tight_layout c
annot make axes height small enough to accommodate all axes decorations.
  plt.tight_layout()
```

In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

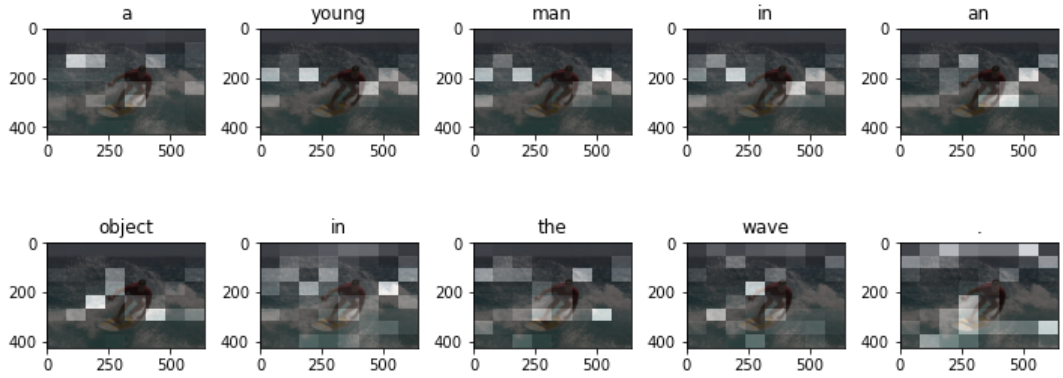
Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)
```

```
result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: a young man in an object in the wave .
```



Out[ ]:

Γ' Βελτίωση με embeddings

Τρίτον, θα χρησιμοποιήσουμε προεκπαιδευμένα word embeddings από το Gensim, αντί να δημιουργούμε δικά μας διανύσματα κατά την εκπαίδευση. Αυτό αναμένουμε να ελαττώσει τον χρόνο εκπαίδευσης καθώς μειώνεται το πλήθος των εκπαιδευόμενων βαρών. Συγκεκριμένα θα δοκιμάσουμε διαστάσεις 50 και 300.

Preprocess the images using ResNet152V2

Καμία αλλαγή.

Preprocess and tokenize the captions

In [ ]:

```
# We will override the default standardization of TextVectorization to preserve
# "<>" characters, so we preserve the tokens for the <start> and <end>.
def standardize(inputs):
    res = tf.strings.lower(inputs)
    ## the following removes all characters except alphanumerics, whitespace, and <, >
    res = tf.strings.regex_replace(res, r"[^a-zA-Z<> ,.]+", "")
    return res

# Max word count for a caption.
max_length = 50
```

```
## https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization
## το layer αυτό δεν έχει παράμετρο κάτω φράγματος, οπότε θα υλοποιηθεί χειροκίνητα
## limited_captions = [c for c in train_captions if len(c.split()) >= 7]
caption_dataset = tf.data.Dataset.from_tensor_slices(train_captions)
# Use the top 7000 words for a vocabulary.
vocabulary_size = 7000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    output_sequence_length=max_length)
# Learn the vocabulary from the caption data.
tokenizer.adapt(caption_dataset)
```

In [ ]:
```
# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))
```

In [ ]:
```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary(),
    invert = True)
```

In [ ]:
```
import gensim.downloader

glove_vectors_50 = gensim.downloader.load('glove-twitter-50')
glove_vectors_300 = gensim.downloader.load('glove-wiki-gigaword-300')
```

In [ ]:
```
%%time
gensim_weights = []
counter = 0

for i in range(vocabulary_size):
    try:
        gensim_weights.append(glove_vectors_50[index_to_word(i).numpy().decode("utf-8")])
    except KeyError:
        counter += 1
        gensim_weights.append(np.zeros(50))
```

```
Wall time: 39.3 s
```

In [ ]:
```
counter
```

Out[ ]:
```
148
```

148 εκ των 7000 του λεξιλογίου μας δεν κωδικοποιούνται από το μοντέλο glove-twitter-50, και χρησιμοποιούμε μηδενικά διανύσματα.

In [ ]:
```
len(gensim_weights)
```

Out[ ]:
```
7000
```

Split the data into training and testing

In [ ]:
```
%%time
img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(img_name_vector, cap_vector):
    img_to_cap_vector[img].append(cap)

# Create training and validation sets using an 80-20 split randomly.
img_keys = list(img_to_cap_vector.keys())
rnd.shuffle(img_keys)

slice_index = int(len(img_keys) * 0.8)
img_name_train_keys, img_name_val_keys = img_keys[:slice_index], img_keys[slice_index:]

img_name_train = []
cap_train = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    img_name_train.extend([imgt] * capt_len)
    cap_train.extend(img_to_cap_vector[imgt])

img_name_val = []
cap_val = []
```

```
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    img_name_val.extend([imgv] * capv_len)
    cap_val.extend(img_to_cap_vector[imgv])
```

```
Wall time: 14.9 s
```

In [ ]:
```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

Out[ ]:
```
(59225, 59225, 14788, 14788)
```

Επιτυγχάνουμε μείωση της διασποράς των μηκών, χάνοντας ελάχιστα δεδομένα.

Create a tf.data dataset for training

Αλλάζουμε καταλλήλως τις διαστάσεις των embeddings παρακάτω

In [ ]:
```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 50
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from ResNet152V2 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 49
```

In [ ]:
```
# Load the numpy files
def map_func(img_name, cap):
    img_tensor = np.load("C:\\Users\\anton\\Documents\\hmmy_mathimata\\neurwnika\\ergasia_3\\resnet_features\\" + img_na
    return img_tensor, cap
```

In [ ]:
```
dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
          map_func, [item1, item2], [tf.float32, tf.int64]),
          num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE, seed = 31).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model

Τροποποιούμε καταλλήλως το Embedding layer, ώστε να μην εκπαιδεύεται και να έχει εξ αρχής τα υπολογισμένα βάρη.

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

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
```

```
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [ ]:
```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

In [ ]:
```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, trainable = False, weights = [np.asarray(g
        #self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)

        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

In [ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Checkpoint

Μεταφέρουμε το τελευταίο checkpoint σε νέο φάκελο και συνεχίζουμε την εκπαίδευση εκεί.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints4"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
@tf.function
def train_step(img_tensor, target):
    loss = 0

    # initializing the hidden state for each batch
    # because the captions are not related from image to image
    hidden = decoder.reset_state(batch_size = target.shape[0])

    dec_input = tf.expand_dims([word_to_index('<start>')] * target.shape[0], 1)

    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)

            loss += loss_function(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)

    total_loss = (loss / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables + decoder.trainable_variables

    gradients = tape.gradient(loss, trainable_variables)

    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss
```

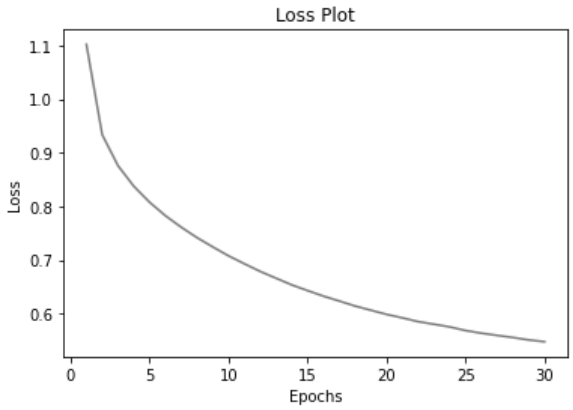
In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 926/926 [04:28<00:00,  3.44it/s]
```

```
Epoch 1: Loss 1.101941
        Time taken 269.13 sec

100%|██████████| 926/926 [03:04<00:00,  5.02it/s]
Epoch 2: Loss 0.933513
        Time taken 184.52 sec

100%|██████████| 926/926 [03:04<00:00,  5.02it/s]
Epoch 3: Loss 0.875719
        Time taken 184.51 sec

100%|██████████| 926/926 [03:04<00:00,  5.03it/s]
Epoch 4: Loss 0.837373
        Time taken 184.39 sec

100%|██████████| 926/926 [03:04<00:00,  5.03it/s]
Epoch 5: Loss 0.807477
        Time taken 184.38 sec

100%|██████████| 926/926 [03:04<00:00,  5.03it/s]
Epoch 6: Loss 0.782465
        Time taken 184.45 sec

100%|██████████| 926/926 [03:04<00:00,  5.03it/s]
Epoch 7: Loss 0.760963
        Time taken 184.43 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 8: Loss 0.741550
        Time taken 254.20 sec

100%|██████████| 926/926 [10:59<00:00,  1.40it/s]
Epoch 9: Loss 0.724158
        Time taken 659.98 sec

100%|██████████| 926/926 [11:01<00:00,  1.40it/s]
Epoch 10: Loss 0.707475
        Time taken 662.52 sec

100%|██████████| 926/926 [11:01<00:00,  1.40it/s]
Epoch 11: Loss 0.692812
        Time taken 662.04 sec

100%|██████████| 926/926 [11:01<00:00,  1.40it/s]
Epoch 12: Loss 0.678482
        Time taken 661.77 sec

100%|██████████| 926/926 [11:00<00:00,  1.40it/s]
Epoch 13: Loss 0.665842
        Time taken 661.22 sec

100%|██████████| 926/926 [11:01<00:00,  1.40it/s]
Epoch 14: Loss 0.653342
        Time taken 661.96 sec

100%|██████████| 926/926 [10:54<00:00,  1.41it/s]
Epoch 15: Loss 0.642744
        Time taken 654.74 sec

100%|██████████| 926/926 [10:59<00:00,  1.41it/s]
Epoch 16: Loss 0.632404
        Time taken 659.30 sec

100%|██████████| 926/926 [10:59<00:00,  1.40it/s]
Epoch 17: Loss 0.623322
        Time taken 659.36 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 18: Loss 0.613955
        Time taken 659.21 sec

100%|██████████| 926/926 [10:59<00:00,  1.41it/s]
```

```
Epoch 19: Loss 0.606156
        Time taken 659.32 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 20: Loss 0.598367
        Time taken 658.66 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 21: Loss 0.591850
        Time taken 659.02 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 22: Loss 0.584809
        Time taken 659.12 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 23: Loss 0.579970
        Time taken 658.59 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 24: Loss 0.574944
        Time taken 659.13 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 25: Loss 0.568167
        Time taken 659.11 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 26: Loss 0.563281
        Time taken 659.18 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 27: Loss 0.559122
        Time taken 658.98 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 28: Loss 0.555192
        Time taken 659.27 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 29: Loss 0.550647
        Time taken 659.26 sec

100%|██████████| 926/926 [10:58<00:00,  1.41it/s]
Epoch 30: Loss 0.547370
        Time taken 659.01 sec
```

In [ ]:
```python
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



Ύστερα από 30 εποχές, η τιμή loss παραμένει σε υπερβολικά υψηλό επίπεδο. Αυτό πιθανώς οφείλεται στην μείωση των διαστάσεων των embeddings. Παρ' όλο που τώρα χρησιμοποιούμε προεκπαιδευμένα -και θεωρητικά καλύτερα από αυτά που προκύπτουν από

την δική μας εκπαίδευση- τα διανύσματα για κάθε λέξη είναι αρκετά μικρότερα (50 αντί 256) και συνεπώς μπορεί να περιέχουν λιγότερη πληροφορία.

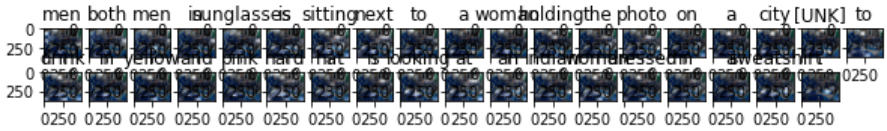
Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A parade with men and women in blue blowing instruments , and beating drums standing in front of a buildi
ng .
Members of a marching band , clad in blue with painted faces , perform on the street .
School band dressed in blue with their faces painted playing music .
A high school band all decked out for spirit week .
A band dressed in blue are playing instruments .
PREDICTION CAPTION: men both men in sunglasses is sitting next to a woman holding the photo on a city [UNK] to drink in
yellow and pink hard hat is looking at an indian woman dressed in a sweatshirt .
Sentence BLEU score:  0.03354007762068305
```

```
C:\Users\anton\AppData\Local\Temp/ipykernel_13688/3853974765.py:15: UserWarning: Tight layout not applied. tight_layout
cannot make axes height small enough to accommodate all axes decorations.
  plt.tight_layout()
```

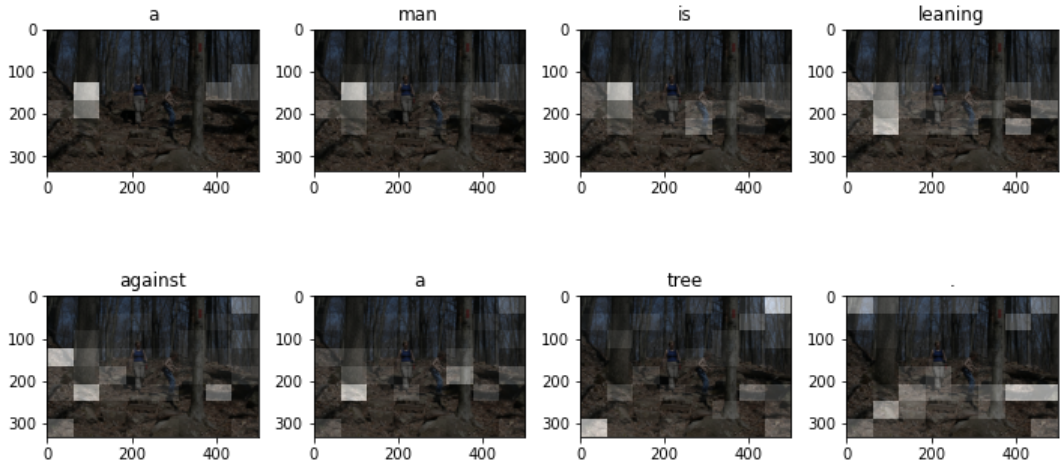


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A woman wearing a blue shirt and a shirtless little boy are walking through a forest .
A woman and a younger child are walking in the woods .
A mother and her child hiking in the woods .
Two people are walking through the woods .
A woman and child hiking during the day .
PREDICTION CAPTION: a man is leaning against a tree .
Sentence BLEU score:  0.05630615150681703
```

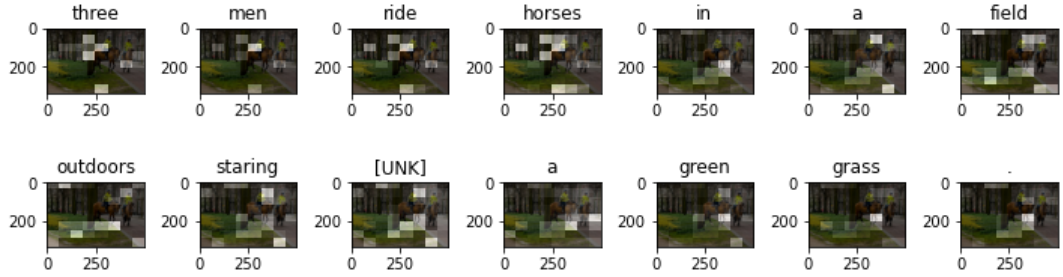


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Two men in neon yellow jackets ride two brown horses down a street next to a bed of bright yellow flowers
.
Two officers ride their horses down a street bordering a green and yellow park .
Two officers are riding horses in the city near some trees and flowers .
Policemen riding horses while patrolling the streets .
Two men are riding on horses through the street .
PREDICTION CAPTION: three men ride horses in a field outdoors staring [UNK] a green grass .
Sentence BLEU score:  0.10937932924924645
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['a', 'black', 'man', 'rock', 'with', 'a', 'coconut', 'in', 'a', 'brown', ',', 'cane', 'near', 'large', 'tree', '.']
```

In [ ]:
```
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [20:16<00:00,  2.47it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.08526776570242207
```

Φαίνεται πως τα μικρού μεγέθους embeddings έχουν ως αποτέλεσμα ελάττωση του corpus bleu.

Samples from the Test Set

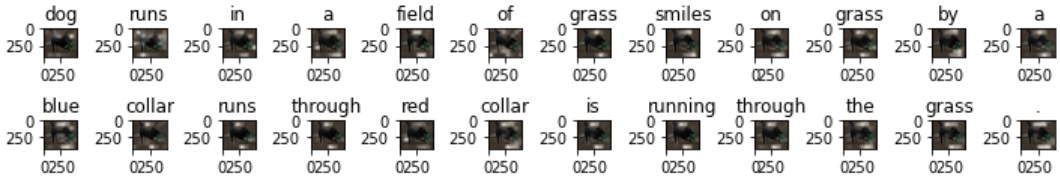
In [ ]:
```
test_file = "/content/drive/My Drive/Colab Notebooks/NN/EX3/annotations/test_images.csv"
```

In [ ]:
```
testList = []
## in this one each line is an image id
with open(test_file, 'r') as f:
    for line in f: testList.append(line.strip())
```

In [ ]:
```
def randomTestCheck(randomImage):
    image = "./image_dir/" + randomImage
    result, attention_plot = evaluate(image)

    print('Prediction Caption:', ' '.join(result))
    plot_attention(image, result, attention_plot)
```

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: dog runs in a field of grass smiles on grass by a blue collar runs through red collar is running thr
ough the grass .
```

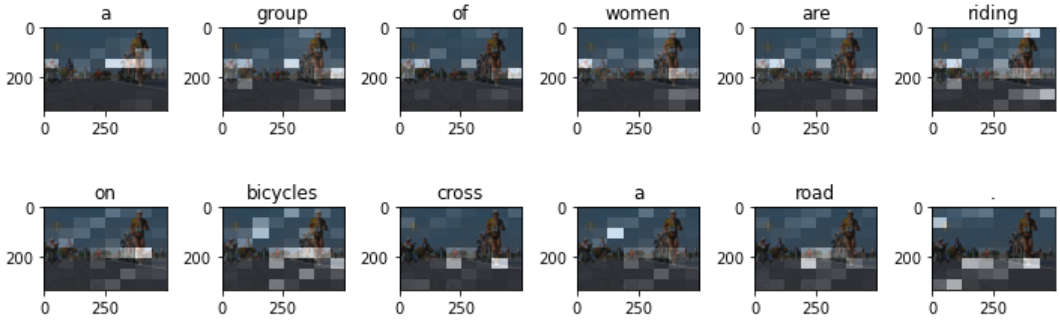


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a group of women are riding on bicycles cross a road .
```

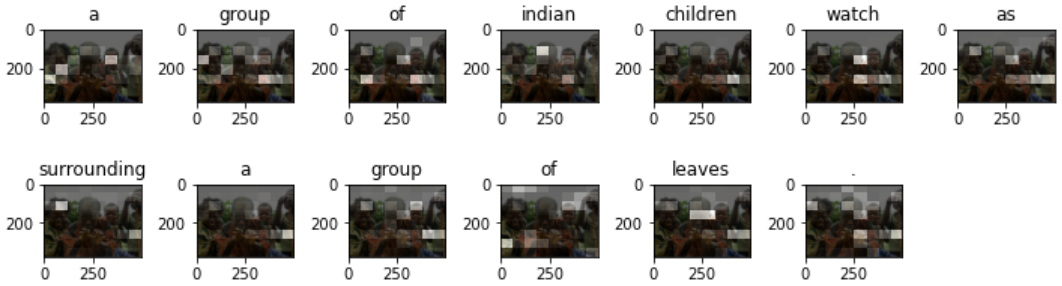


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a group of indian children watch as surrounding a group of leaves .
```



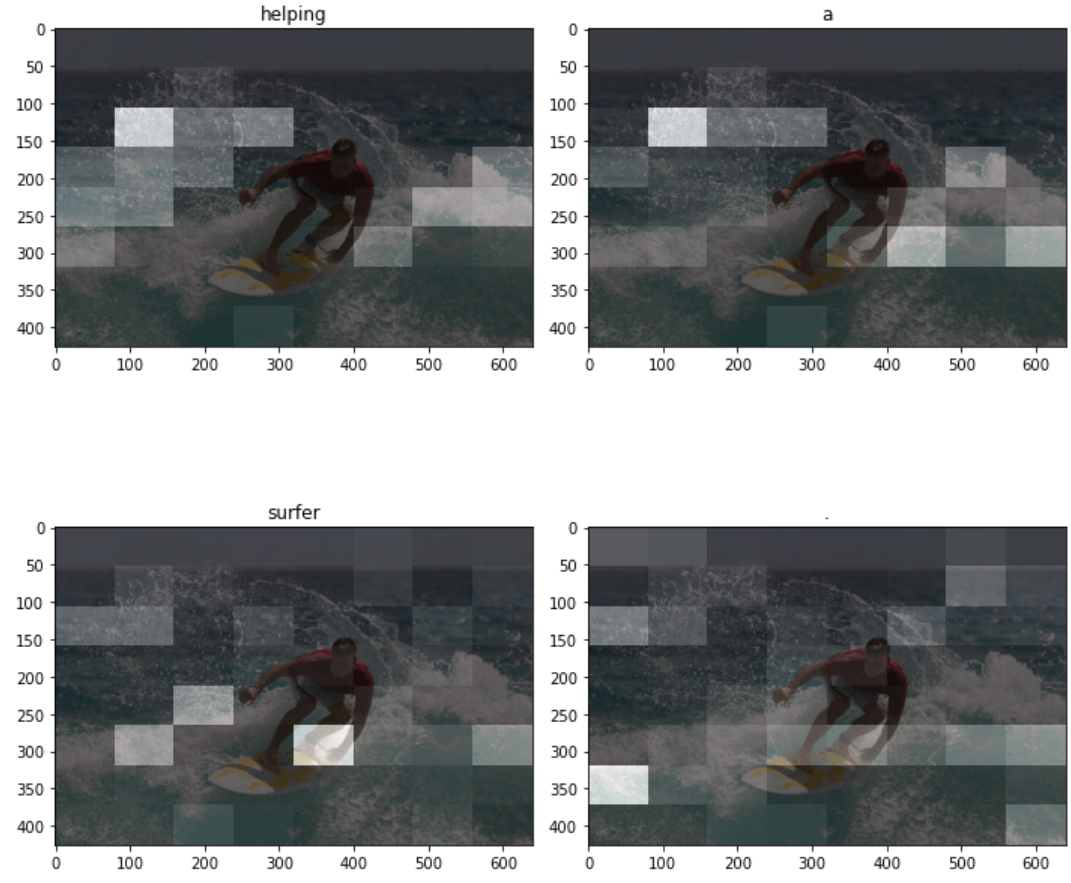
In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: helping a surfer .
```



Out[ ]:

Ακολουθούμε την ίδια ακριβώς διαδικασία για τα emdeddings διάστασης 300, αλλάζοντας καταλλήλως τις τιμές των παραμέτρων όπου χρειάζεται.

Preprocess and tokenize the captions

In [ ]:
```
# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))
```

In [ ]:
```
# Create mappings for words to indices and indices to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
```

```
    mask_token = "",
    vocabulary = tokenizer.get_vocabulary(),
    invert = True)
```

In [ ]:
```
import gensim.downloader

glove_vectors_300 = gensim.downloader.load('glove-wiki-gigaword-300')
```

In [ ]:
```
%%time
gensim_weights = []
counter = 0

for i in range(vocabulary_size):
    try:
        gensim_weights.append(glove_vectors_300[index_to_word(i).numpy().decode("utf-8")])
    except KeyError:
        counter += 1
        gensim_weights.append(np.zeros(300))
```

```
Wall time: 36.8 s
```

In [ ]:
```
counter
```

Out[ ]:
```
133
```

Τώρα οι άγνωστες λέξεις είναι λιγότερες, 133 αντί 148, πολύ μικρό ποσοστό του συνόλου.

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 926/926 [05:35<00:00,  2.76it/s]
Epoch 18: Loss 0.420017
        Time taken 335.47 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 19: Loss 0.409982
        Time taken 253.94 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 20: Loss 0.401399
        Time taken 253.75 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 21: Loss 0.392163
        Time taken 253.79 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 22: Loss 0.387366
        Time taken 253.79 sec

100%|██████████| 926/926 [04:13<00:00,  3.65it/s]
```

```
Epoch 23: Loss 0.379748
        Time taken 253.73 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 24: Loss 0.372963
        Time taken 253.97 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:13<00:00,  3.65it/s]
Epoch 25: Loss 0.366688
        Time taken 253.97 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:12<00:00,  3.66it/s]
Epoch 26: Loss 0.359766
        Time taken 253.25 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:13<00:00,  3.66it/s]
Epoch 27: Loss 0.353465
        Time taken 253.39 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:13<00:00,  3.66it/s]
Epoch 28: Loss 0.349289
        Time taken 253.56 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:13<00:00,  3.66it/s]
Epoch 29: Loss 0.342819
        Time taken 253.44 sec

100%|████████████████████████████████████████████████████████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 30: Loss 0.339338
        Time taken 252.12 sec

100%|████████████████████████████████████████████████████████████| 926/926 [16:06<00:00,  1.04s/it] Epoch 1: Loss 1.076667 Time taken 966.64 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:46<00:00,  1.04it/s] Epoch 2: Loss 0.889005 Time taken 887.24 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:46<00:00,  1.04it/s] Epoch 3: Loss 0.815938 Time taken 887.04 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 4: Loss 0.760837 Time taken 885.53 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 5: Loss 0.714140 Time taken 885.96 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 6: Loss 0.673444 Time taken 886.25 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 7: Loss 0.636835 Time taken 885.88 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:46<00:00,  1.04it/s] Epoch 8: Loss 0.604879 Time taken 887.23 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 9: Loss 0.577150 Time taken 885.98 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 10: Loss 0.550659 Time taken 886.11 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:45<00:00,  1.05it/s] Epoch 11: Loss 0.527969 Time taken 885.97 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:44<00:00,  1.05it/s] Epoch 12: Loss 0.507900 Time taken 884.48 sec

100%|████████████████████████████████████████████████████████████| 926/926 [14:31<00:00,  1.06it/s] Epoch 13: Loss 0.489934 Time taken 871.44 sec
```

100%|██████████| 926/926 [05:36<00:00, 2.75it/s] Epoch 14: Loss 0.473340 Time taken 337.04 sec

100%|██████████| 926/926 [04:10<00:00, 3.69it/s] Epoch 15: Loss 0.459624 Time taken 251.10 sec

100%|██████████| 926/926 [04:10<00:00, 3.69it/s] Epoch 16: Loss 0.444941 Time taken 251.15 sec

100%|██████████| 926/926 [04:10<00:00, 3.70it/s] Epoch 17: Loss 0.432703 Time taken 250.83 sec

100%|██████████| 926/926 [05:35<00:00, 2.76it/s] Epoch 18: Loss 0.420017 Time taken 335.47 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 19: Loss 0.409982 Time taken 253.94 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 20: Loss 0.401399 Time taken 253.75 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 21: Loss 0.392163 Time taken 253.79 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 22: Loss 0.387366 Time taken 253.79 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 23: Loss 0.379748 Time taken 253.73 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 24: Loss 0.372963 Time taken 253.97 sec

100%|██████████| 926/926 [04:13<00:00, 3.65it/s] Epoch 25: Loss 0.366688 Time taken 253.97 sec

100%|██████████| 926/926 [04:12<00:00, 3.66it/s] Epoch 26: Loss 0.359766 Time taken 253.25 sec

100%|██████████| 926/926 [04:13<00:00, 3.66it/s] Epoch 27: Loss 0.353465 Time taken 253.39 sec

100%|██████████| 926/926 [04:13<00:00, 3.66it/s] Epoch 28: Loss 0.349289 Time taken 253.56 sec

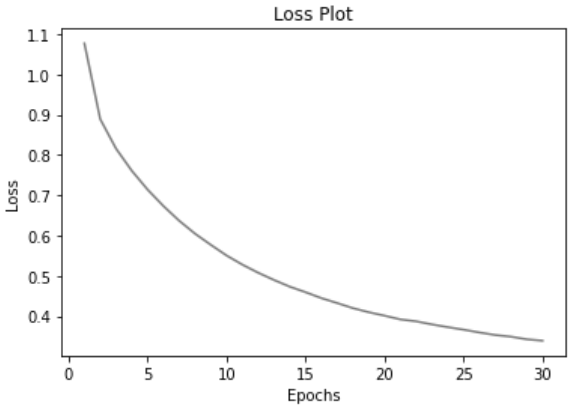
100%|██████████| 926/926 [04:13<00:00, 3.66it/s] Epoch 29: Loss 0.342819 Time taken 253.44 sec

100%|██████████| 926/926 [04:11<00:00, 3.68it/s] Epoch 30: Loss 0.339338 Time taken 252.12 sec

In [ ]:
```
loss_plot = [1.076667,0.889005,0.815938,0.760837,0.714140,0.673444,0.636835,0.604879,0.577150,0.550659,0.527969,0.507900
print(loss_plot)
```

```
[1.076667, 0.889005, 0.815938, 0.760837, 0.71414, 0.673444, 0.636835, 0.604879, 0.57715, 0.550659, 0.527969, 0.5079, 0.4
89934, 0.47334, 0.459624, 0.444941, 0.432703, 0.42001712, 0.40998152, 0.40139934, 0.39216343, 0.38736615, 0.37974834, 0.
37296313, 0.36668777, 0.35976583, 0.35346514, 0.3492887, 0.34281886, 0.33933768]
```

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



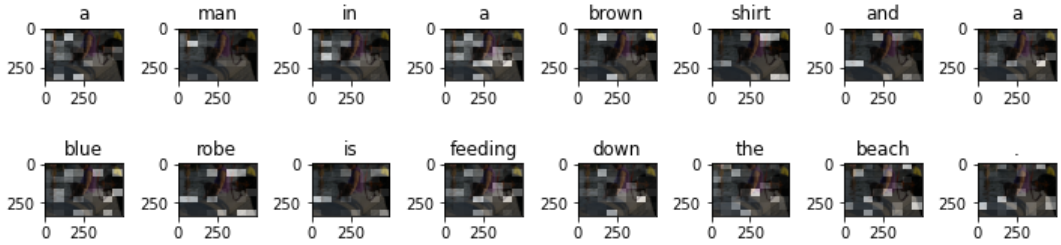
Επειδή η εκπαίδευση πραγματοποιήθηκε σταδιακά, μέρος του output παρατίθεται σε μορφή κειμένου. Σε αυτό το πείραμα η τιμή της loss είναι σημαντικά χαμηλότερη σε σχέση με το προηγούμενο, ύστερα από ισάριθμες εποχές.

Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: While the humans put their feet into the fountain , the puppy on the side with his master checks out the
muzzle leash on the large rottweiler .
A woman sits with her feet in a shallow pool and looks at a puppy .
A woman sits by the pool with a dog while another dog is nearby .
A woman in a purple shirt and a brown puppy sit near a fountain
Woman and dog cooling off at fountain
PREDICTION CAPTION: a man in a brown shirt and a blue robe is feeding down the beach .
Sentence BLEU score:  0.20207969803579098
```

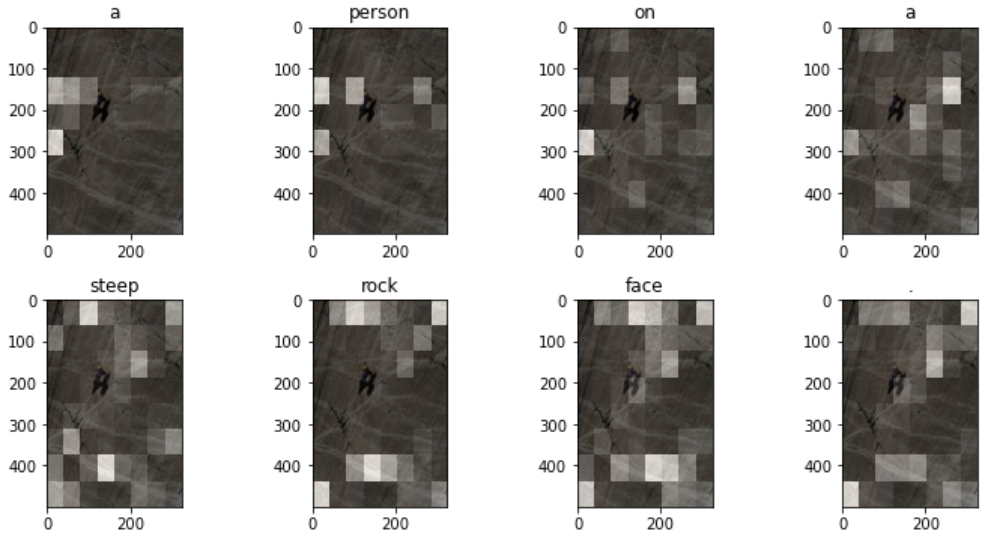


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A person climbing a large mountainside .
A person climbs up a rock face .
a man climbing a sheer rock wall
A climber ascending a mountain
A rock climber
PREDICTION CAPTION: a person on a steep rock face .
Sentence BLEU score:  0.2892523137172855
```

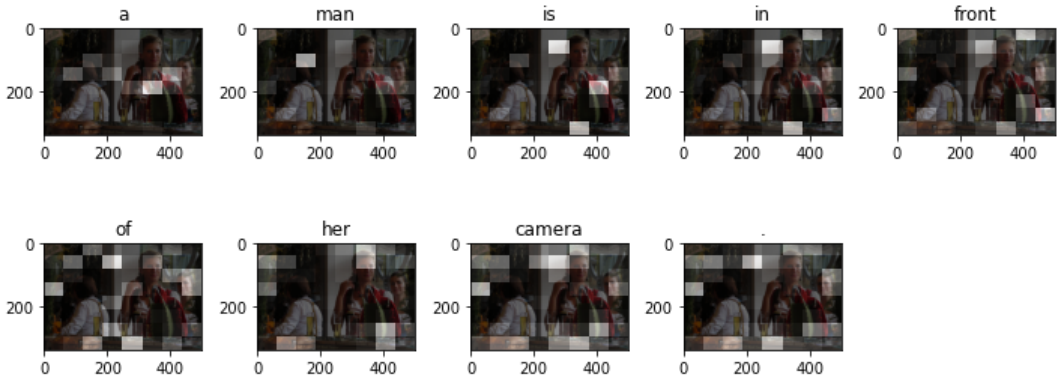


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A beautiful blond woman and brunette woman are at a bar where beer is being enjoyed dressed in traditiona
l German or European laced up dresses .
Three woman are at the same place , all wearing white shirts and are enjoying themselves .
A young lady with a green purse sits with a drink , while people in the back smile .
A woman with a green purse sits at a bar with an empty glass in front of her .
Two women are standing at a bar .
PREDICTION CAPTION: a man is in front of her camera .
Sentence BLEU score:  0.4384528108764514
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['a', 'young', 'boy', 'holds', 'a', 'coconut', 'to', 'the', 'person', '.']
```

In [ ]:
```
## :param list_of_references: a corpus of lists of reference sentences, w.r.t. hypotheses
## :type list_of_references: list(list(list(str)))
## :param hypotheses: a list of hypothesis sentences
## :type hypotheses: list(list(str))
## :param weights: weights for unigrams, bigrams, trigrams and so on (one or a list of weights)
## :type weights: tuple(float) / list(tuple(float))

allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [20:06<00:00,  2.49it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

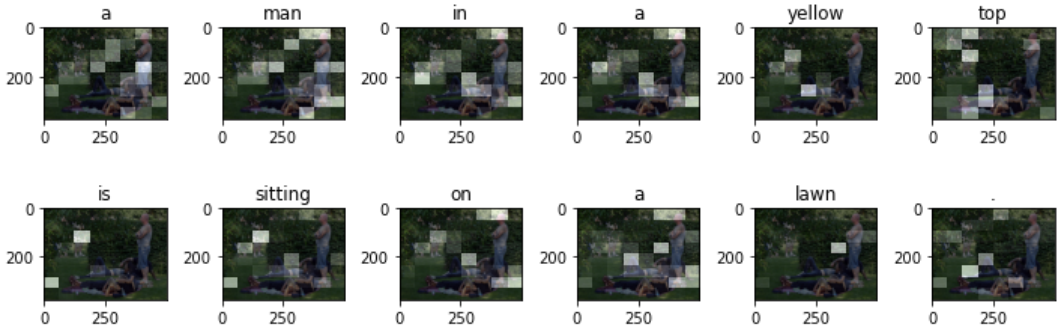
```
Corpus BLEU score:  0.09019639780744894
```

Χρησιμοποιώντας μεγαλύτερα embeddings, το σκορ βελτιώθηκε.

Samples from the Test Set

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man in a yellow top is sitting on a lawn .
```

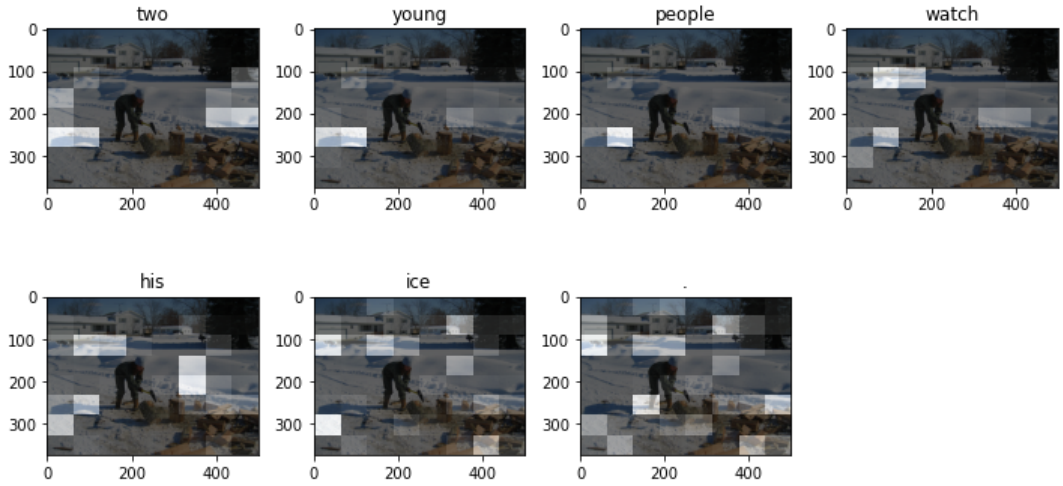


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: two young people watch his ice .
```



In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a girl with the help washing a wooden wall and cameraman is trying to open work in a blue shirt is r
eading sheet .
```

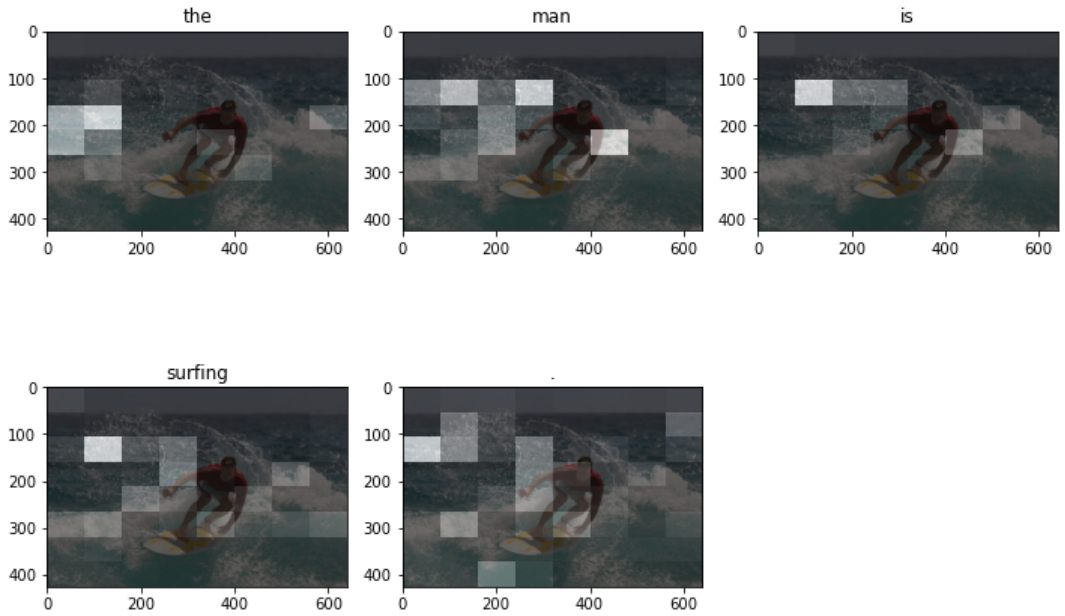
In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: the man is surfing .
```



Out[ ]:

Δ' Βελτίωση με Dropout

Τέταρτον, θα προσθέσουμε επίπεδα Dropout στο δίκτυο ώστε να αποφευχθεί το overfitting. Η μοναδική τροποποίηση του βήματος αυτού σε σχέση με το προηγούμενο βρίσκεται στο μοντέλο του decoder, όπου προσθέτουμε Dropout layers ύστερα από κάθε fully connected dense layer. Στον encoder δεν θα εφαρμόσουμε dropout, αφού σε αυτόν δεν εκτελείται εκπαίδευση. Για το πείραμα αυτό χρησιμοποιήθηκε dropout rate 0.25, το οποίο είναι σύνηθες για RNN.

Model

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

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64, embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

In [ ]:
```
class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 64, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)
```

```
    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x
```

In [ ]:
```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, trainable = False, weights = [np.asarray(g
        #self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.dropout1 = tf.keras.layers.Dropout(rate = 0.25)
        self.fc2 = tf.keras.layers.Dense(vocab_size)
        self.dropout2 = tf.keras.layers.Dropout(rate = 0.25)

        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)
        x = self.dropout1(x)


        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)
        x = self.dropout2(x)
        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

In [ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Checkpoint

Μεταφέρουμε το τελευταίο checkpoint σε νέο φάκελο και συνεχίζουμε την εκπαίδευση εκεί.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints6"
ckpt = tf.train.Checkpoint(encoder=encoder,
```

```
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

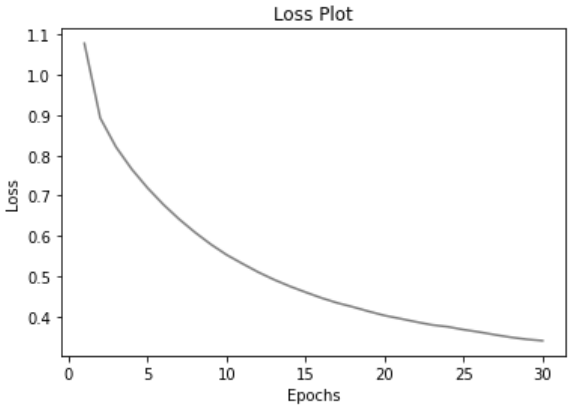
In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    loss_plot.append(total_loss / num_steps)

    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 926/926 [05:35<00:00,  2.76it/s]
Epoch 1: Loss 1.078500
        Time taken 335.87 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 2: Loss 0.893266
        Time taken 251.47 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 3: Loss 0.821032
        Time taken 251.46 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 4: Loss 0.765789
        Time taken 251.31 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 5: Loss 0.719138
        Time taken 251.61 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 6: Loss 0.678013
        Time taken 251.47 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 7: Loss 0.641489
        Time taken 251.27 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 8: Loss 0.608830
        Time taken 251.36 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 9: Loss 0.579161
        Time taken 251.42 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 10: Loss 0.553042
        Time taken 251.21 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
```

```
Epoch 11: Loss 0.531098
        Time taken 251.34 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 12: Loss 0.509982
        Time taken 251.39 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 13: Loss 0.491317
        Time taken 251.30 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 14: Loss 0.474817
        Time taken 251.43 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 15: Loss 0.460008
        Time taken 251.32 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 16: Loss 0.445846
        Time taken 251.40 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 17: Loss 0.433457
        Time taken 251.21 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 18: Loss 0.423366
        Time taken 251.23 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 19: Loss 0.412344
        Time taken 251.37 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 20: Loss 0.402112
        Time taken 251.55 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 21: Loss 0.394286
        Time taken 251.40 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 22: Loss 0.386210
        Time taken 251.40 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 23: Loss 0.378723
        Time taken 251.45 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 24: Loss 0.374119
        Time taken 251.46 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 25: Loss 0.366925
        Time taken 251.30 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 26: Loss 0.361005
        Time taken 251.45 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 27: Loss 0.354095
        Time taken 251.45 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 28: Loss 0.347923
        Time taken 251.73 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
```

```
Epoch 29: Loss 0.342893
        Time taken 251.39 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 30: Loss 0.339482
        Time taken 251.36 sec
```

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```

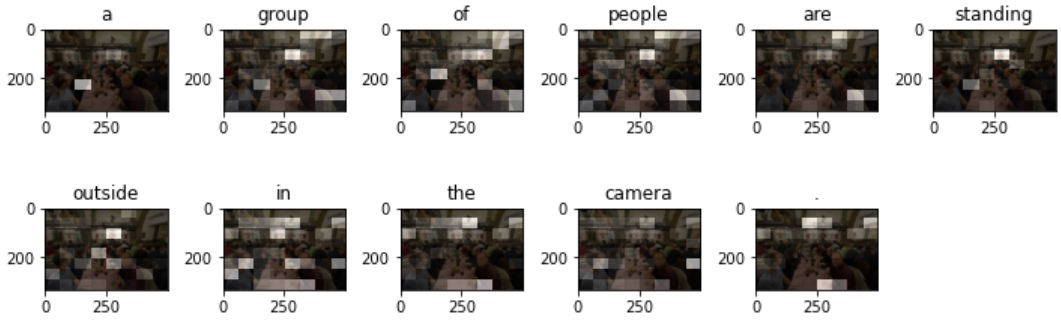


Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Groups of people sitting at maroon colored tablecloth covered tables in a restaurant .
Lots of people in a restaurant eating and talking .
Men in a busy restaurant observe decorations .
Several men sitting at a long table .
A group of people gather for dinner .
PREDICTION CAPTION: a group of people are standing outside in the camera .
Sentence BLEU score:  0.20130516788893235
```

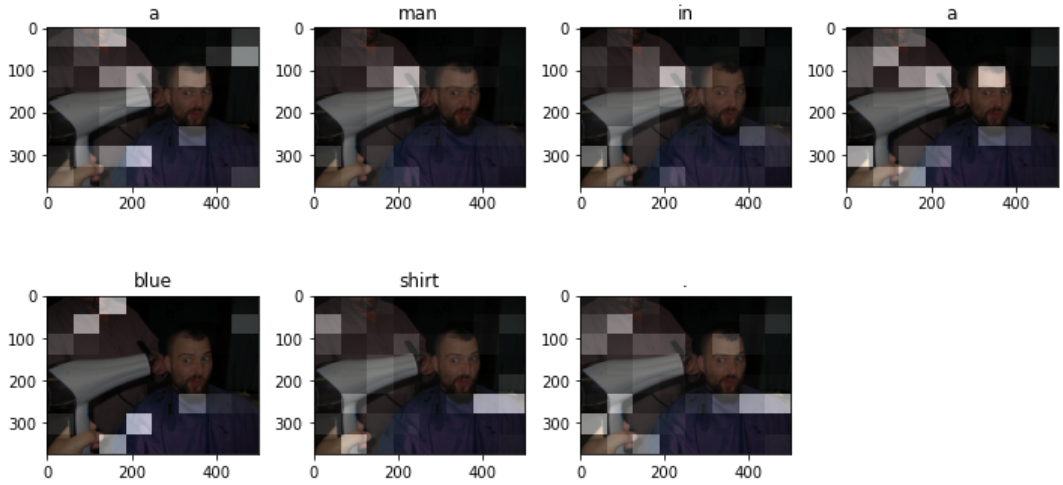


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A bearded man in a purple cape getting his haircut makes a twisted face at the hair dryer .
A man in a purple getting his haircut with a giant blow dryer faced towards him .
Man getting his haircut while looking with a strange expression at a blow drier .
A man reacting to his hair being dried .
A man is making a face at a hair dryer .
PREDICTION CAPTION: a man in a blue shirt .
Sentence BLEU score:  0.23677087091880406
```



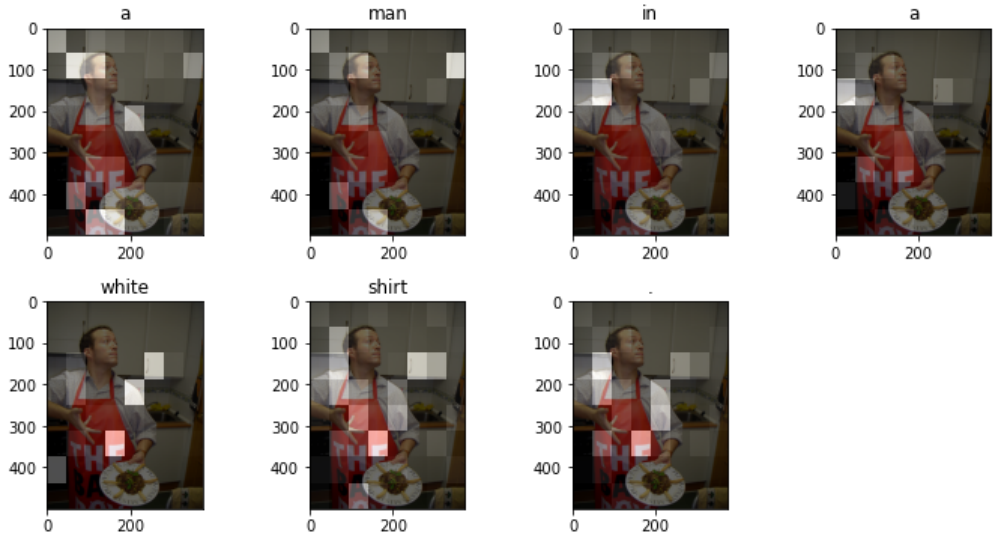
In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
```

```
randomCheck(rid)
```

```
REAL CAPTIONS: A man in an orange apron standing in a kitchen , ostentatiously presenting a plate of food .
A man is wearing an apron , standing in the kitchen holding a plate .
A man in a red apron displays a gourmet plated entree .
A guy in the kitchen points to his new pasta creation
Male chef playfully presenting his latest creation .
PREDICTION CAPTION: a man in a white shirt .
Sentence BLEU score:  0.27313058850276045
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['this', 'is', 'climbing', 'up', 'a', 'tree', '.']
```

In [ ]:
```
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [20:21<00:00,  2.46it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

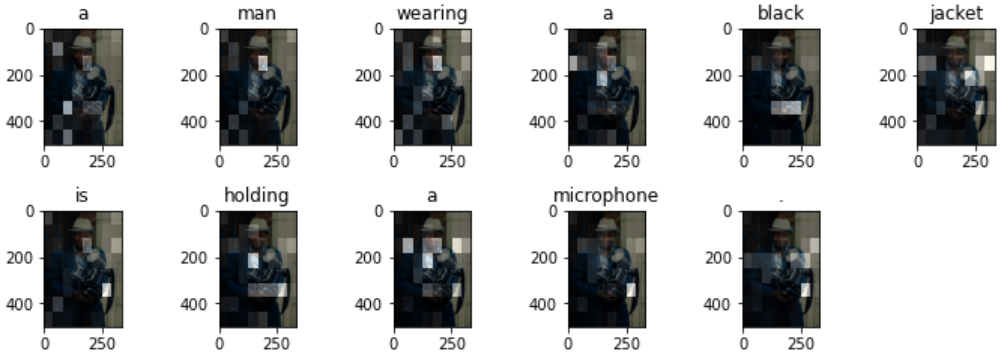
```
Corpus BLEU score:  0.09128380446789612
```

Παρατηρούμε βελτίωση οπότε θα υιοθετήσουμε και αυτήν την αλλαγή.

Samples from the Test Set

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man wearing a black jacket is holding a microphone .
```

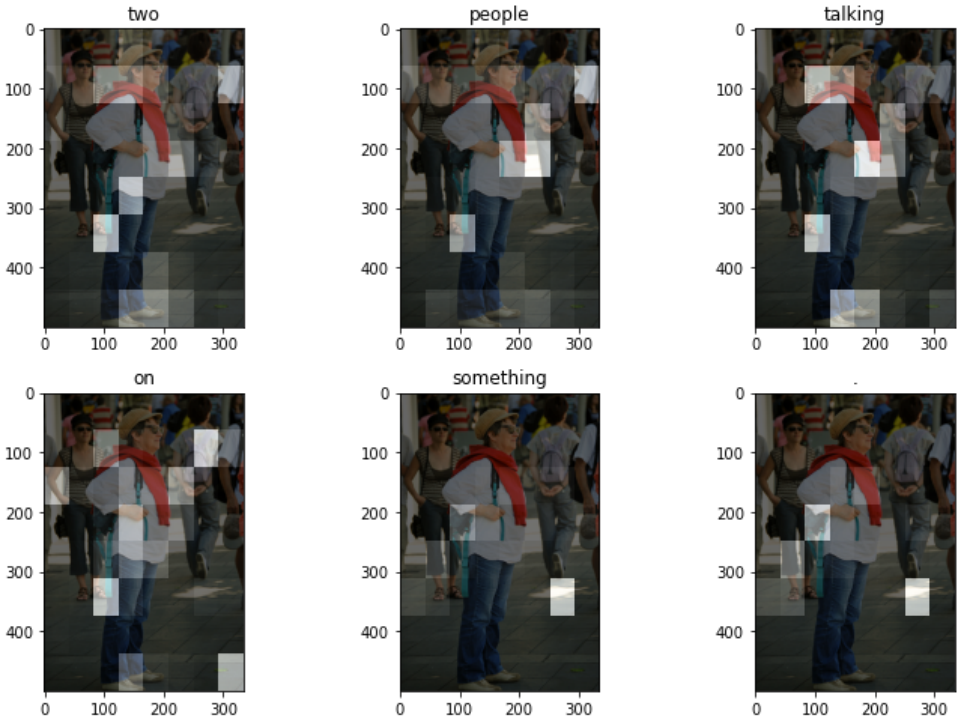


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: two people talking on something .
```

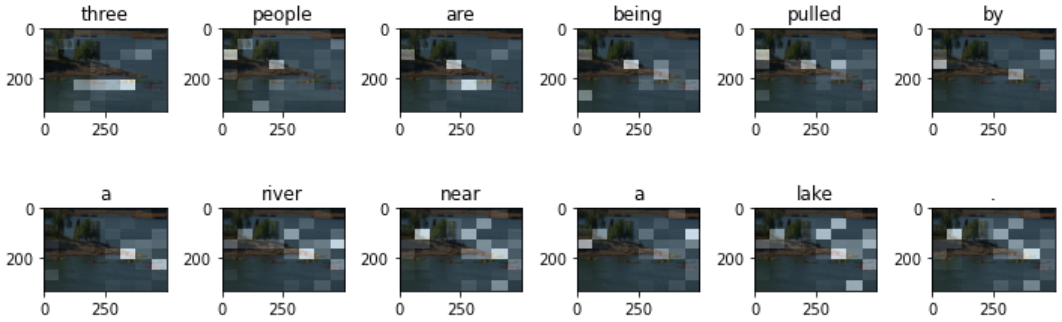


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: three people are being pulled by a river near a lake .
```



In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

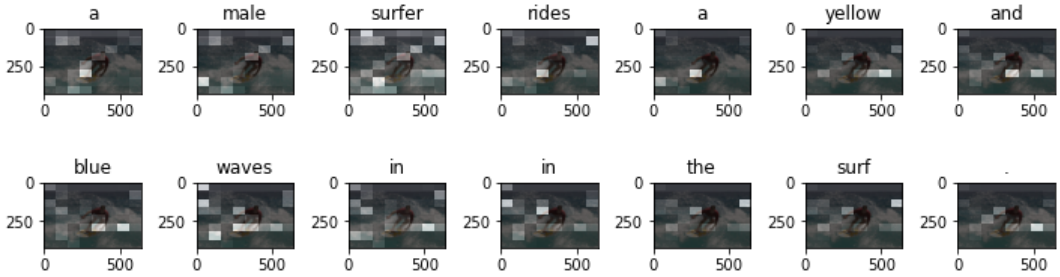
Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)
```

```
result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: a male surfer rides a yellow and blue waves in in the surf .
```



Out[ ]:

Ε' Βελτίωση με επεξεργασία των Recurrent Layers (GRU - LSTM)

Πέμπτον, θα δούμε πώς επιδρά στην επίδοση το πλήθος των μονάδων (units) καθώς και το είδος των επαναληπτικών επιπέδων. Αρχικά, θα δοκιμάσουμε να υποδιπλασιάσουμε το πλήθος των μονάδων του επιπέδου GRU του RNN decoder και στη συνέχεια θα δοκιμάσουμε να το αντικαταστήσουμε με επίπεδο LSTM. Αλλαγές γίνονται μόνο στο μοντέλο του decoder και τον ορισμό της σταθεράς units.

Model

In [ ]:

```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, trainable = False, weights = [np.asarray(g
        #self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.dropout1 = tf.keras.layers.Dropout(rate = 0.25)
        self.fc2 = tf.keras.layers.Dense(vocab_size)
        self.dropout2 = tf.keras.layers.Dropout(rate = 0.25)
        self.attention = BahdanauAttention(self.units)
```

```
    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)
        x = self.dropout1(x)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)
        x = self.dropout2(x)

        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

In [ ]:
```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)
```

Checkpoint

Μεταφέρουμε το τελευταίο checkpoint σε νέο φάκελο και συνεχίζουμε την εκπαίδευση εκεί.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints7"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

In [ ]:
```
start_epoch
```

Out[ ]:
```
0
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

In [ ]:
```
EPOCHS = 30
```

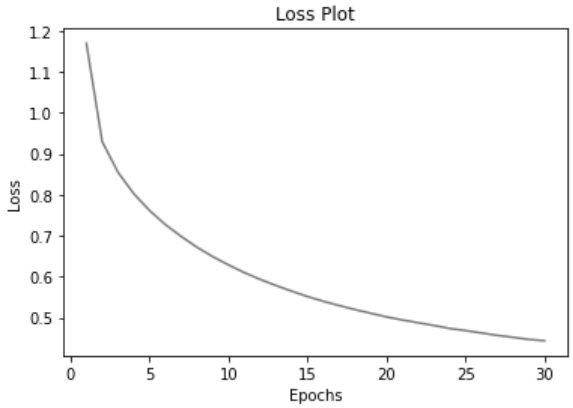
```python
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 926/926 [04:23<00:00,  3.52it/s]
Epoch 1: Loss 1.168832
        Time taken 263.51 sec

100%|██████████| 926/926 [02:48<00:00,  5.50it/s]
Epoch 2: Loss 0.929176
        Time taken 168.46 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 3: Loss 0.854710
        Time taken 169.00 sec

100%|██████████| 926/926 [02:48<00:00,  5.48it/s]
Epoch 4: Loss 0.802364
        Time taken 169.10 sec

100%|██████████| 926/926 [02:48<00:00,  5.48it/s]
Epoch 5: Loss 0.761254
        Time taken 169.01 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 6: Loss 0.727033
        Time taken 168.91 sec

100%|██████████| 926/926 [02:48<00:00,  5.48it/s]
Epoch 7: Loss 0.698197
        Time taken 169.06 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 8: Loss 0.671832
        Time taken 168.90 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 9: Loss 0.648850
        Time taken 168.79 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 10: Loss 0.628720
        Time taken 168.87 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 11: Loss 0.610077
        Time taken 168.87 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 12: Loss 0.593378
        Time taken 168.83 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 13: Loss 0.578479
        Time taken 168.78 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 14: Loss 0.564703
        Time taken 168.80 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
```

```
Epoch 15: Loss 0.551661
        Time taken 168.77 sec

100%|██████████| 926/926 [02:48<00:00,  5.50it/s]
Epoch 16: Loss 0.540072
        Time taken 168.60 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 17: Loss 0.529696
        Time taken 168.69 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 18: Loss 0.519807
        Time taken 168.83 sec

100%|██████████| 926/926 [02:48<00:00,  5.50it/s]
Epoch 19: Loss 0.510872
        Time taken 168.71 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 20: Loss 0.501989
        Time taken 168.78 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 21: Loss 0.494578
        Time taken 168.88 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 22: Loss 0.487779
        Time taken 168.70 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 23: Loss 0.481109
        Time taken 168.79 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 24: Loss 0.473740
        Time taken 168.78 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 25: Loss 0.468600
        Time taken 168.71 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 26: Loss 0.462924
        Time taken 168.95 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 27: Loss 0.456966
        Time taken 168.86 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 28: Loss 0.452260
        Time taken 168.95 sec

100%|██████████| 926/926 [02:49<00:00,  5.48it/s]
Epoch 29: Loss 0.447065
        Time taken 169.26 sec

100%|██████████| 926/926 [02:48<00:00,  5.49it/s]
Epoch 30: Loss 0.443978
        Time taken 168.92 sec
```

In [ ]:

```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



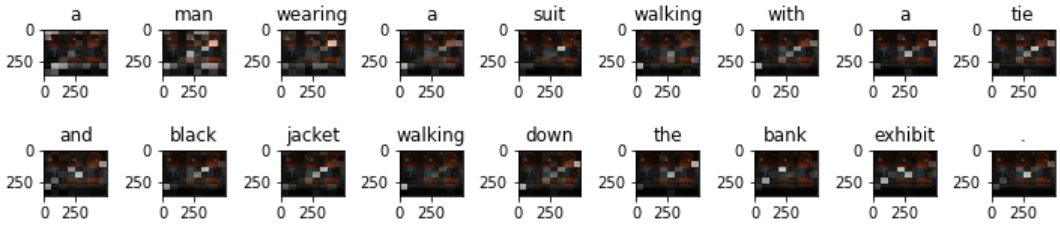
Ύστερα από 30 εποχές παρατηρούμε ικανοποιητική τιμή loss για το πρώτο αυτό πείραμα.

Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A man in a blue collared shirt and tan pants is wandering through a London subway station .
A man in a blue shirt walking in a subway tunnel with advertisements .
A man walking along the platform at a subway station .
A man walks by a building at night .
The marquee of movies now playing .
PREDICTION CAPTION: a man wearing a suit walking with a tie and black jacket walking down the bank exhibit .
Sentence BLEU score:  0.034006285287353034
```

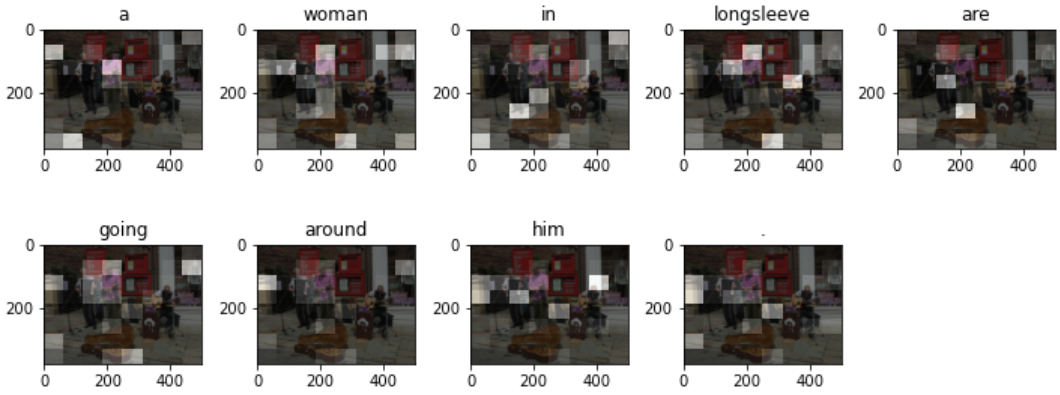


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Colorful musicians playing on a street corner near a church and shopping area .
On a strip mall sidewalk , three men in a band play their music .
Street musicians playing a accordion , violin and a guitar .
This street band is playing music for people passing by .
Musicians playing in front of a shop .
PREDICTION CAPTION: a woman in longsleeve are going around him .
Sentence BLEU score:  0.04913486520648141
```

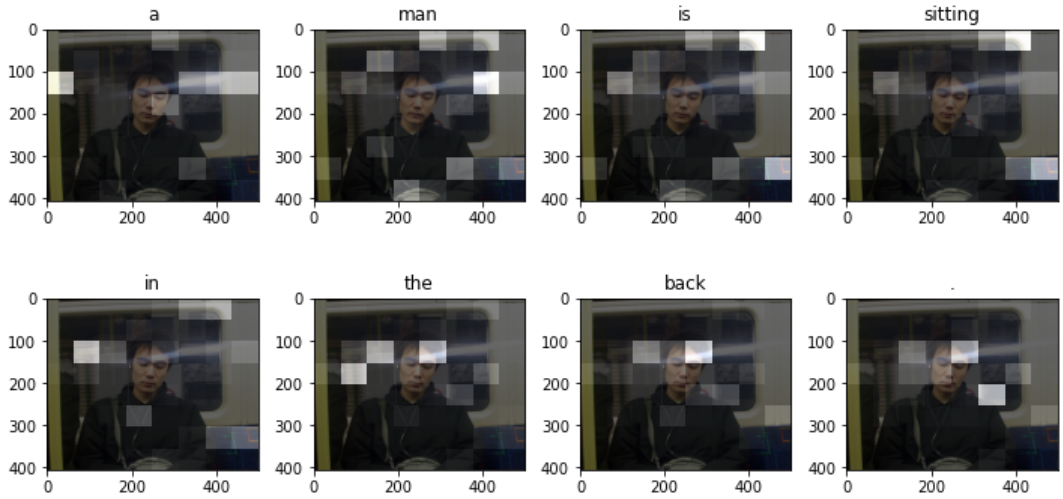


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: The young man is listening to some smoothing music , he looks so peaceful .
Someone falls asleep on the train listening to their music .
A man with headphones in , is sitting on a bus or train .
A man in a black jacket sitting on a bus with earphones .
An Asian man on a train wearing headphones .
PREDICTION CAPTION: a man is sitting in the back .
Sentence BLEU score:  0.17130449035712855
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['a', 'man', 'stands', 'in', 'a', 'climbing', '.']
```

In [ ]:
```
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [20:56<00:00,  2.39it/s]
```

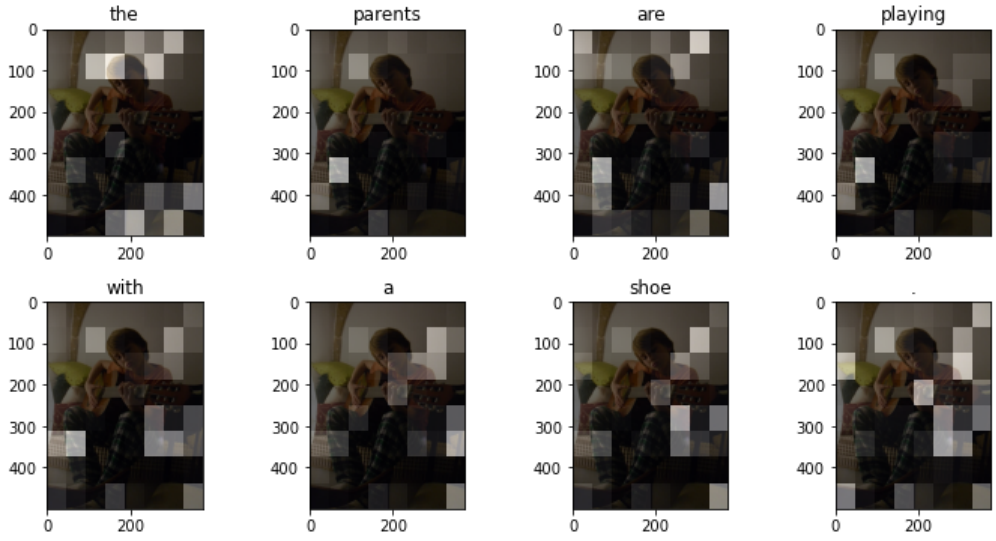
In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.08564548291528767
```

Samples from the Test Set

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: the parents are playing with a shoe .
```

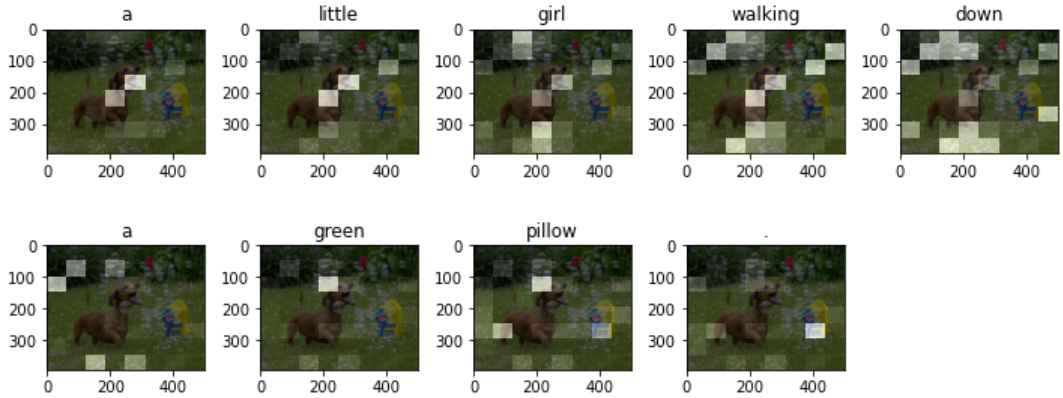


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a little girl walking down a green pillow .
```

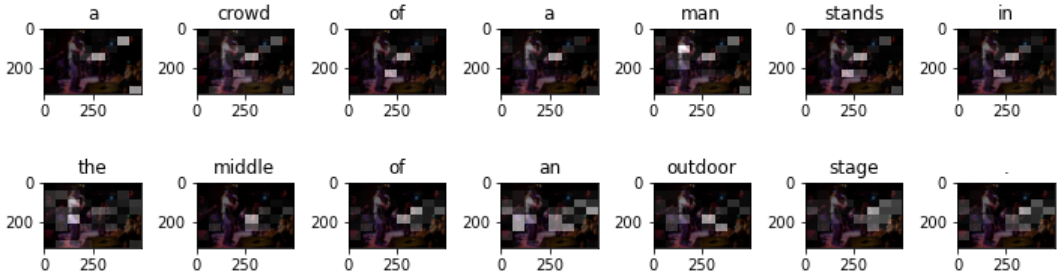


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a crowd of a man stands in the middle of an outdoor stage .
```



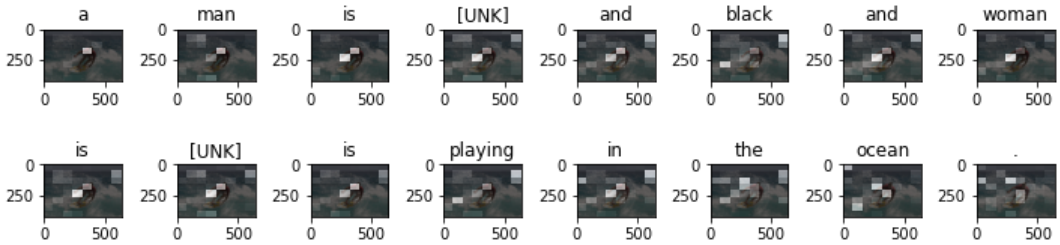
In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: a man is [UNK] and black and woman is [UNK] is playing in the ocean .
```



Out[ ]:

Ακολουθούμε την ίδια ακριβώς διαδικασία για μονάδα RNN LSTM με 512 units, αλλάζοντας καταλλήλως τις τιμές των παραμέτρων όπου χρειάζεται.

Αλλάζουμε καταλλήλως τις διαστάσεις των units παρακάτω

In [ ]:
```
# Feel free to change these parameters according to your system's configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 300
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# Shape of the vector extracted from ResNet152V2 is (64, 2048)
# These two variables represent that vector shape
features_shape = 2048
attention_features_shape = 49
```

Model

In [ ]:
```
class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim, trainable = False, weights = [np.asarray(g
        #self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)

        self.lstm = tf.keras.layers.LSTM(self.units,
                                         return_sequences=True,
                                         return_state=True,
                                         recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.dropout1 = tf.keras.layers.Dropout(rate = 0.25)
        self.fc2 = tf.keras.layers.Dense(vocab_size)
        self.dropout2 = tf.keras.layers.Dropout(rate = 0.25)
        self.attention = BahdanauAttention(self.units)

    def call(self, x, features, hidden):
```

```
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state, _ = self.lstm(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)
        x = self.dropout1(x)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)
        x = self.dropout2(x)

        return x, state, attention_weights

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

In [ ]:
```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())
```

Checkpoint

Μεταφέρουμε το τελευταίο checkpoint σε νέο φάκελο και συνεχίζουμε την εκπαίδευση εκεί.

In [ ]:
```
checkpoint_path = os.path.abspath('.') + "/checkpoints8"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep = 5)
```

In [ ]:
```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)
```

Training

In [ ]:
```
# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []
```

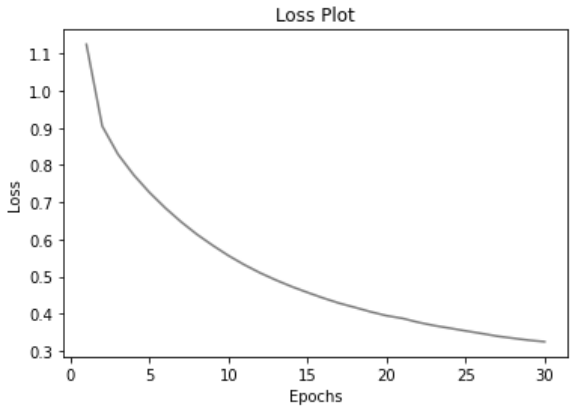
In [ ]:
```
EPOCHS = 30

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    ## if(epoch % 5 == 0):
    ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 926/926 [05:53<00:00,  2.62it/s]
```

```
Epoch 1: Loss 1.124432
        Time taken 353.38 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 2: Loss 0.904798
        Time taken 267.32 sec

100%|██████████| 926/926 [04:26<00:00,  3.47it/s]
Epoch 3: Loss 0.829003
        Time taken 268.44 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 4: Loss 0.773160
        Time taken 267.79 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 5: Loss 0.725985
        Time taken 268.71 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 6: Loss 0.684443
        Time taken 267.55 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 7: Loss 0.647080
        Time taken 267.68 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 8: Loss 0.613473
        Time taken 267.42 sec

100%|██████████| 926/926 [04:26<00:00,  3.47it/s]
Epoch 9: Loss 0.583825
        Time taken 267.36 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 10: Loss 0.556237
        Time taken 267.65 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 11: Loss 0.531795
        Time taken 267.73 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 12: Loss 0.509917
        Time taken 267.67 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 13: Loss 0.490647
        Time taken 267.31 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 14: Loss 0.473037
        Time taken 267.63 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 15: Loss 0.457281
        Time taken 267.47 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 16: Loss 0.442271
        Time taken 267.77 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 17: Loss 0.428556
        Time taken 267.66 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 18: Loss 0.417149
        Time taken 267.97 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
```

```
Epoch 19: Loss 0.405274
        Time taken 267.76 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 20: Loss 0.394870
        Time taken 267.45 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 21: Loss 0.387808
        Time taken 267.51 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 22: Loss 0.377122
        Time taken 267.62 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 23: Loss 0.368447
        Time taken 267.47 sec

100%|██████████| 926/926 [04:29<00:00,  3.44it/s]
Epoch 24: Loss 0.361356
        Time taken 269.73 sec

100%|██████████| 926/926 [04:28<00:00,  3.45it/s]
Epoch 25: Loss 0.354065
        Time taken 269.17 sec

100%|██████████| 926/926 [04:29<00:00,  3.44it/s]
Epoch 26: Loss 0.347172
        Time taken 269.45 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 27: Loss 0.339838
        Time taken 267.95 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 28: Loss 0.334356
        Time taken 267.56 sec

100%|██████████| 926/926 [04:27<00:00,  3.46it/s]
Epoch 29: Loss 0.328963
        Time taken 267.87 sec

100%|██████████| 926/926 [04:27<00:00,  3.47it/s]
Epoch 30: Loss 0.324956
        Time taken 267.54 sec
```

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```

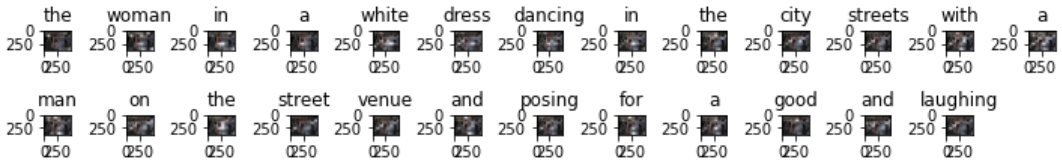


Evaluation of Captions

BLEU Scores of the Train Validation Set

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Several Israeli men standing in the middle of a busy intersection .
The group of linen dressed men are standing in the street .
Men in white and pink gowns stand on a plaza .
Men in traditional dress stand outside .
Men wearing robes waiting to perform .
PREDICTION CAPTION: the woman in a white dress dancing in the city streets with a man on the street venue and posing for a good and laughing
Sentence BLEU score:  0.05911773442966007
```

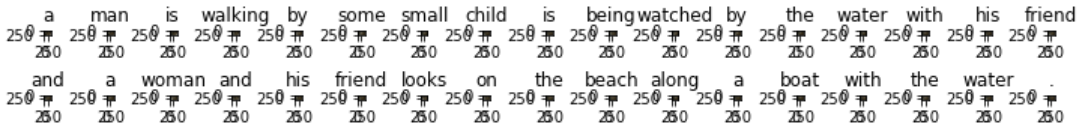


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: three guys are on the beach and one is doing some kind of move where is body is off the ground .
Three young men wearing shorts or swimsuits are playing on a beach during a sunset
Two people stand and a third jumps in the air on a sunny day on an ocean beach .
Three people are seen on a beach during sunset , one is jumping .
Three friends have the time of their life at the beach .
PREDICTION CAPTION: a man is walking by some small child is being watched by the water with his friend and a woman and h is friend looks on the beach along a boat with the water .
Sentence BLEU score:  0.08737386287015152
```



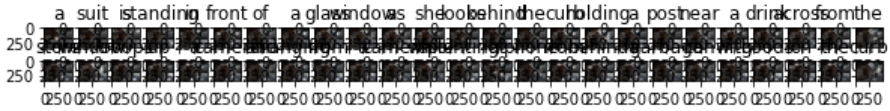
In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Two young girls sitting on the corner of a street with a white car passing in the background .
Two Asian women sitting on a street corner , while a white car passes by .
Two young ladies sit at the corner of a building .
Two woman sitting in a doorway on a city street .
Two young women sitting on a doorstep .
PREDICTION CAPTION: a suit is standing in front of a glass window as she looks behind the curb holding a post near a dri
nk across from the store window troops up a camera and hanging from a camera while pointing at phone from behind a garba
ge can with goods on the curb
Sentence BLEU score:  0.031637644350259175
```

```
C:\Users\anton\AppData\Local\Temp/ipykernel_7208/3853974765.py:15: UserWarning: Tight layout not applied. tight_layout c
annot make axes width small enough to accommodate all axes decorations
  plt.tight_layout()
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['an', 'individual', 'ears', '.']
```

In [ ]:
```
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 3000/3000 [21:08<00:00,  2.37it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

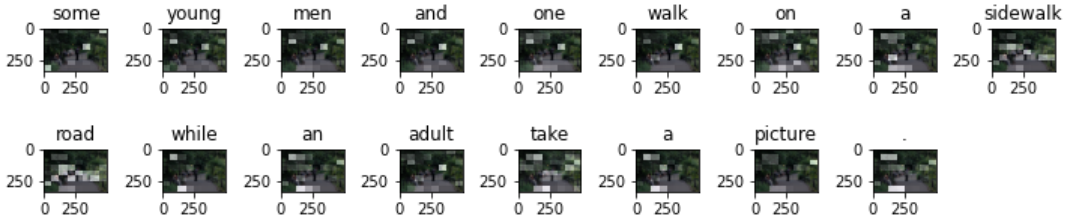
```
Corpus BLEU score:  0.07871496078043 53
```

Τόσο τα BLEU scores, όσο και οι περιγραφές μεμονωμένων δειγμάτων, μας οδηγούν στην απόφαση να μην κρατήσουμε καμμία από τις αλλαγές που δοκιμάσθηκαν σε αυτό το βήμα.

Samples from the Test Set

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: some young men and one walk on a sidewalk road while an adult take a picture .
```

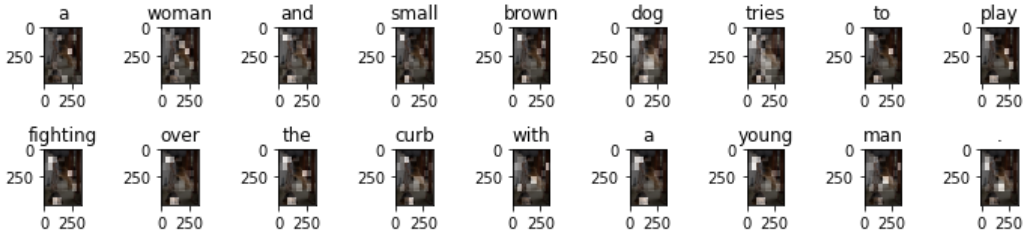


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a woman and small brown dog tries to play fighting over the curb with a young man .
```

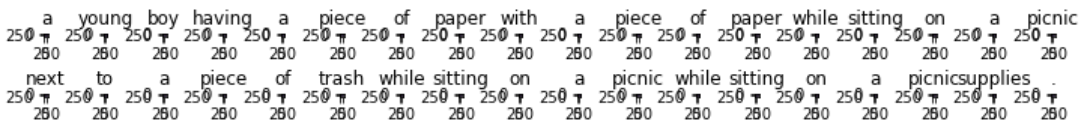


In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a young boy having a piece of paper with a piece of paper while sitting on a picnic next to a piece
of trash while sitting on a picnic while sitting on a picnic supplies .
```



In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

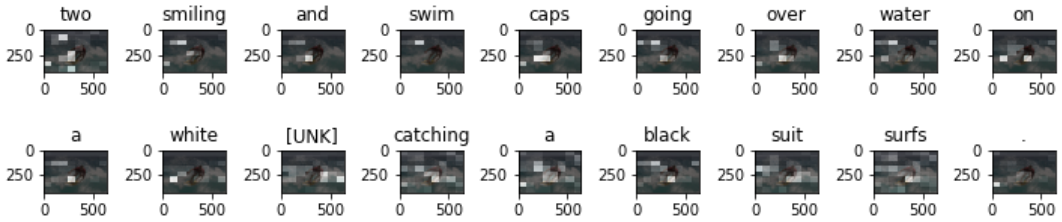
Out[ ]:

In [ ]:
```
## predictions for entire test set

testPreds = [evaluate(os.path.abspath('.') + "/image_dir/" + i)[0] for i in testList]
```

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result, attention_plot = evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
plot_attention(image_path, result, attention_plot)
# opening the image
Image.open(image_path)
```

```
Prediction Caption: two smiling and swim caps going over water on a white [UNK] catching a black suit surfs .
```



Out[ ]:

Τελικό Μοντέλο

Ύστερα από όλες τις δοκιμές, το τελικό μοντέλο διαμορφώθηκε ως εξής:

- ResNet152V2 encoder.
- Προεπεξεργασία κειμένου με αφαίρεση captions μήκους εκτός του διαστήματος [6, 31].
- Χρήση των προεκπαιδευμένων embeddings glove-wiki-gigaword-300 στον RNN decoder.
- Εφαρμογή dropout layers με 0.25 rate ύστερα από τα πυκνά επίπεδα του decoder.
- Διατήρηση του GRU cell του decoder ως είχε από το tutorial.

Για την εκπαίδευση του τελικού μοντέλου έγιναν δύο απόπειρες. Μία σε ολόκληρο το train dataset, και μία στο 50% περίπου αυτού, με 15,000 εικόνες. Τα αποτελέσματα ανά εποχή εκπαίδευσης παρακάτω:

15,000 ΕΙΚΟΝΕΣ

In [ ]:

```
EPOCHS = 80

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if((epoch+1) % 10 == 0):
        ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

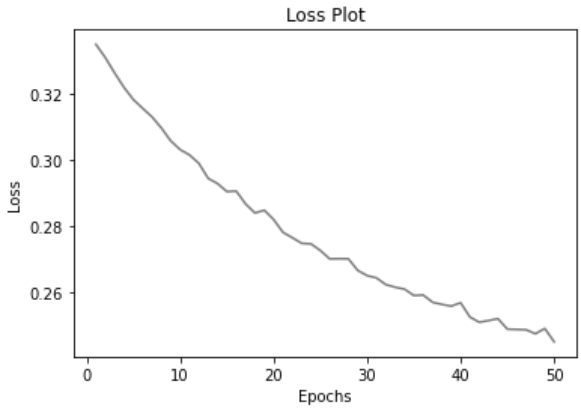
```
100%|██████████| 926/926 [05:32<00:00,  2.78it/s]
Epoch 31: Loss 0.335126
        Time taken 332.59 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 32: Loss 0.331106
        Time taken 251.37 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
```

```
Epoch 33: Loss 0.326442
        Time taken 251.45 sec

100%|██████████| 926/926 [04:10<00:00,  3.69it/s]
Epoch 34: Loss 0.322057
        Time taken 251.03 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 35: Loss 0.318375
        Time taken 251.22 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 36: Loss 0.315705
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 37: Loss 0.313102
        Time taken 251.41 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 38: Loss 0.309736
        Time taken 251.16 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 39: Loss 0.305823
        Time taken 251.06 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 40: Loss 0.303198
        Time taken 251.82 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 41: Loss 0.301570
        Time taken 251.26 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 42: Loss 0.299041
        Time taken 251.14 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 43: Loss 0.294449
        Time taken 251.23 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 44: Loss 0.292880
        Time taken 251.32 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 45: Loss 0.290464
        Time taken 251.36 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 46: Loss 0.290598
        Time taken 251.33 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 47: Loss 0.286715
        Time taken 251.28 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 48: Loss 0.283954
        Time taken 251.29 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 49: Loss 0.284802
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 50: Loss 0.281966
        Time taken 251.58 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
```

```
Epoch 51: Loss 0.278072
        Time taken 251.23 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 52: Loss 0.276449
        Time taken 251.11 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 53: Loss 0.274747
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 54: Loss 0.274562
        Time taken 251.19 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 55: Loss 0.272555
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 56: Loss 0.270014
        Time taken 251.07 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 57: Loss 0.270076
        Time taken 251.10 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 58: Loss 0.270033
        Time taken 251.14 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 59: Loss 0.266502
        Time taken 251.65 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 60: Loss 0.264944
        Time taken 251.44 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 61: Loss 0.264235
        Time taken 251.17 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 62: Loss 0.262207
        Time taken 251.20 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 63: Loss 0.261416
        Time taken 251.12 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 64: Loss 0.260838
        Time taken 251.28 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 65: Loss 0.258897
        Time taken 251.18 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 66: Loss 0.259032
        Time taken 251.30 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 67: Loss 0.256847
        Time taken 251.18 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 68: Loss 0.256179
        Time taken 251.15 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
```

```
Epoch 69: Loss 0.255673
        Time taken 251.11 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 70: Loss 0.256689
        Time taken 251.47 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 71: Loss 0.252342
        Time taken 251.37 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 72: Loss 0.250775
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 73: Loss 0.251303
        Time taken 251.37 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 74: Loss 0.251855
        Time taken 251.16 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 75: Loss 0.248681
        Time taken 251.16 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 76: Loss 0.248594
        Time taken 251.27 sec

100%|██████████| 926/926 [04:11<00:00,  3.68it/s]
Epoch 77: Loss 0.248494
        Time taken 251.40 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 78: Loss 0.247282
        Time taken 251.21 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 79: Loss 0.248875
        Time taken 251.30 sec

100%|██████████| 926/926 [04:11<00:00,  3.69it/s]
Epoch 80: Loss 0.244871
        Time taken 251.68 sec
```

In [ ]:
```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



ΟΛΟΚΛΗΡΟ DATASET

In [ ]:
```
EPOCHS = 80
```

```python
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

    ## storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if((epoch+1) % 10 == 0):
        ckpt_manager.save()

    print(f'Epoch {epoch + 1}: Loss {total_loss / num_steps:.6f}')
    print(f'\tTime taken {time.time() - start:.2f} sec\n')
```

```
100%|██████████| 1838/1838 [09:54<00:00,  3.09it/s]
Epoch 1: Loss 0.998234
        Time taken 594.17 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 2: Loss 0.854786
        Time taken 502.21 sec

100%|██████████| 1838/1838 [08:20<00:00,  3.67it/s]
Epoch 3: Loss 0.800121
        Time taken 501.00 sec

100%|██████████| 1838/1838 [08:20<00:00,  3.67it/s]
Epoch 4: Loss 0.758326
        Time taken 500.81 sec

100%|██████████| 1838/1838 [08:21<00:00,  3.67it/s]
Epoch 5: Loss 0.724277
        Time taken 501.12 sec

100%|██████████| 1838/1838 [08:21<00:00,  3.67it/s]
Epoch 6: Loss 0.696502
        Time taken 501.08 sec

100%|██████████| 1838/1838 [08:20<00:00,  3.67it/s]
Epoch 7: Loss 0.669918
        Time taken 500.94 sec

100%|██████████| 1838/1838 [08:21<00:00,  3.67it/s]
Epoch 8: Loss 0.647289
        Time taken 501.54 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 9: Loss 0.627805
        Time taken 503.57 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 10: Loss 0.610870
        Time taken 503.05 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 11: Loss 0.595200
        Time taken 502.58 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 12: Loss 0.581217
        Time taken 502.69 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 13: Loss 0.569188
        Time taken 502.44 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 14: Loss 0.559425
        Time taken 502.66 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
```

```
Epoch 15: Loss 0.548701
        Time taken 502.09 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 16: Loss 0.539714
        Time taken 502.42 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 17: Loss 0.531754
        Time taken 502.24 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 18: Loss 0.526118
        Time taken 502.68 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 19: Loss 0.517709
        Time taken 502.78 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 20: Loss 0.511115
        Time taken 502.83 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 21: Loss 0.506116
        Time taken 502.94 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:24<00:00,  3.64it/s]
Epoch 22: Loss 0.500287
        Time taken 505.01 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 23: Loss 0.494948
        Time taken 502.86 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 24: Loss 0.490497
        Time taken 502.54 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 25: Loss 0.485671
        Time taken 502.88 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 26: Loss 0.481383
        Time taken 502.71 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 27: Loss 0.479047
        Time taken 503.05 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 28: Loss 0.474631
        Time taken 502.69 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 29: Loss 0.471262
        Time taken 502.92 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 30: Loss 0.467526
        Time taken 503.21 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 31: Loss 0.465148
        Time taken 502.58 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 32: Loss 0.462099
        Time taken 502.80 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
```

```
Epoch 33: Loss 0.459720
        Time taken 502.75 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 34: Loss 0.457316
        Time taken 502.77 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 35: Loss 0.454523
        Time taken 502.75 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 36: Loss 0.452462
        Time taken 502.88 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 37: Loss 0.449411
        Time taken 502.92 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 38: Loss 0.447696
        Time taken 502.90 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 39: Loss 0.445625
        Time taken 503.15 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 40: Loss 0.444003
        Time taken 503.20 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 41: Loss 0.442802
        Time taken 503.11 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 42: Loss 0.440566
        Time taken 502.78 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 43: Loss 0.438949
        Time taken 503.00 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 44: Loss 0.439764
        Time taken 502.77 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 45: Loss 0.437219
        Time taken 503.13 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 46: Loss 0.433502
        Time taken 502.95 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 47: Loss 0.432290
        Time taken 502.97 sec

100%|██████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 48: Loss 0.431183
        Time taken 503.00 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 49: Loss 0.430445
        Time taken 503.13 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 50: Loss 0.428315
        Time taken 503.64 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
```

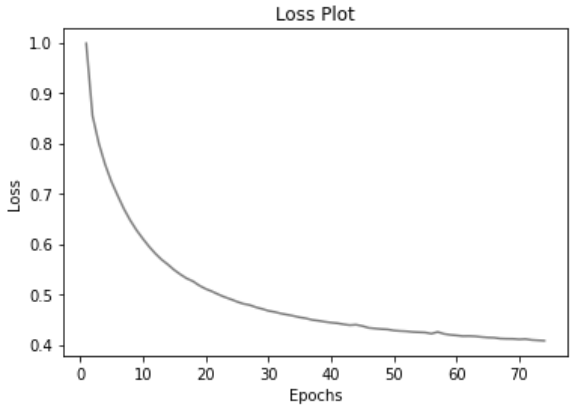
```
Epoch 51: Loss 0.427361
        Time taken 503.13 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 52: Loss 0.426696
        Time taken 503.05 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.66it/s]
Epoch 53: Loss 0.425491
        Time taken 502.86 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:22<00:00,  3.65it/s]
Epoch 54: Loss 0.424880
        Time taken 503.02 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 55: Loss 0.424333
        Time taken 503.27 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 56: Loss 0.421926
        Time taken 503.10 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 57: Loss 0.425475
        Time taken 503.24 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 58: Loss 0.421545
        Time taken 503.10 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 59: Loss 0.419703
        Time taken 503.18 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 60: Loss 0.418562
        Time taken 503.45 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 61: Loss 0.417122
        Time taken 503.30 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 62: Loss 0.417285
        Time taken 503.09 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 63: Loss 0.416714
        Time taken 503.27 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 64: Loss 0.415376
        Time taken 503.13 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 65: Loss 0.414379
        Time taken 503.31 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 66: Loss 0.413961
        Time taken 503.14 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 67: Loss 0.412247
        Time taken 503.55 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 68: Loss 0.411785
        Time taken 503.52 sec

100%|████████████████████████████████████████████████████████████| 1838/1838 [08:23<00:00,  3.65it/s]
```

```
Epoch 69: Loss 0.411646
        Time taken 503.57 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 70: Loss 0.410730
        Time taken 503.68 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 71: Loss 0.411290
        Time taken 503.23 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 72: Loss 0.409498
        Time taken 503.58 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 73: Loss 0.408722
        Time taken 503.20 sec

100%|██████████| 1838/1838 [08:23<00:00,  3.65it/s]
Epoch 74: Loss 0.408044
        Time taken 503.31 sec

 74%|███████▍  | 1351/1838 [06:10<02:13,  3.65it/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_1808/1910246154.py in <module>
      6
      7     for (batch, (img_tensor, target)) in enumerate(tqdm(dataset)):
----> 8         batch_loss, t_loss = train_step(img_tensor, target)
      9         total_loss += t_loss
     10

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\util\traceback_utils.py in error_handler(*args, **kwarg
s)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\def_function.py in __call__(self, *args, **kwds)
    908
    909       with OptionalXlaContext(self._jit_compile):
--> 910         result = self._call(*args, **kwds)
    911
    912       new_tracing_count = self.experimental_get_tracing_count()

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\def_function.py in _call(self, *args, **kwds)
    940       # In this case we have created variables on the first call, so we run the
    941       # defunned version which is guaranteed to never create variables.
--> 942       return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    943     elif self._stateful_fn is not None:
    944       # Release the lock early so that multiple threads can perform the call

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in __call__(self, *args, **kwargs)
   3128       (graph_function,
   3129        filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 3130     return graph_function._call_flat(
   3131         filtered_flat_args, captured_inputs=graph_function.captured_inputs)  # pylint: disable=protected-access
   3132

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in _call_flat(self, args, captured_inp
uts, cancellation_manager)
   1957         and executing_eagerly):
   1958       # No tape is watching; skip to running the function.
-> 1959       return self._build_call_outputs(self._inference_function.call(
   1960           ctx, args, cancellation_manager=cancellation_manager))
   1961     forward_backward = self._select_forward_and_backward_functions(

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\function.py in call(self, ctx, args, cancellation_
manager)
    596       with _InterpolateFunctionError(self):
    597         if cancellation_manager is None:
--> 598           outputs = execute.execute(
    599               str(self.signature.name),
    600               num_outputs=self._num_outputs,

~\anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\eager\execute.py in quick_execute(op_name, num_outputs,
 inputs, attrs, ctx, name)
     56   try:
     57     ctx.ensure_initialized()
---> 58     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     59                                         inputs, attrs, num_outputs)
     60   except core._NotOkStatusException as e:

KeyboardInterrupt:
```

In [ ]:

```
plt.plot(*zip(*enumerate(loss_plot, start = 1)), color = "gray")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```



ΣΥΓΚΡΙΣΗ ΤΩΝ ΔΥΟ ΕΚΠΑΙΔΕΥΣΕΩΝ

Προκειμένου να επιλέξουμε την καλύτερη εποχή από κάθε εκπαίδευση, κρατήσαμε checkpoints σε κάθε δεκάδα και για αυτά υπολογίσαμε corpus bleu scores στο validation set ή τμήματα αυτού.

Μετρήσεις για το μοντέλο με ολόκληρο το dataset:

```
Epoch 10: Loss 0.610870, 2000-Validation-Corpus: 0.08499714308404795

Epoch 20: Loss 0.511115, 2000-Validation-Corpus: 0.08540502454288809

Epoch 30: Loss 0.467526, 2000-Validation-Corpus: 0.08325070468769878

Epoch 40: Loss 0.444003, 2000-Validation-Corpus: 0.0891430748754118

Epoch 50: Loss 0.428315, 2000-Validation-Corpus: 0.08422499426261552

Epoch 60: Loss 0.418562, 2000-Validation-Corpus: 0.08430032850772029

Epoch 70: Loss 0.410730, 2000-Validation-Corpus: 0.08558450388972223
```

Μετρήσεις για το μοντέλο με το dataset 15,000:

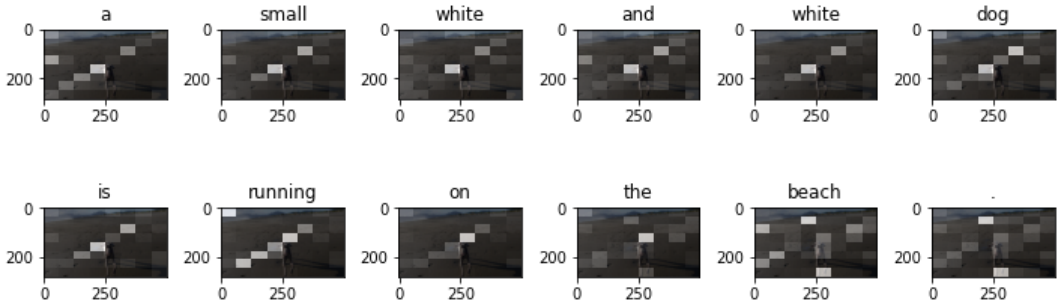
```
Epoch 30: Loss 0.339482, Half-Validation-Corpus: 0.09404551048108365, Full-Validation-Corpus:
0.09246518182176702

Epoch 40: Loss 0.303198, Half-Validation-Corpus: 0.08937733650077502, Full-Validation-Corpus:
0.08539440080150314

Epoch 50: Loss 0.281966, Half-Validation-Corpus: 0.09214500965410857, Full-Validation-Corpus:
0.08058847723819762

Epoch 60: Loss 0.264944, Half-Validation-Corpus: 0.08464197846094393, Full-Validation-Corpus: δεν
κρίθηκε σκόπιμος ο υπολογισμός

Epoch 70: Loss 0.256689, Half-Validation-Corpus: 0.08564050548108684, Full-Validation-Corpus: δεν
κρίθηκε σκόπιμος ο υπολογισμός

Epoch 80: Loss 0.244871, Half-Validation-Corpus: 0.08204551036150406, Full-Validation-Corpus:
0.08087607208819224
```

Από τις παραπάνω προκύπτουν η εποχή 40 και 30 αντιστοίχως. Διατηρώντας checkpoints για τις παραπάνω δύο εκπαιδεύσεις, συγκρίναμε την καλύτερη εποχή της μίας με την καλύτερη εποχή της άλλης, **βάσει κοινών δειγμάτων στα validation sets και των δύο συνόλων δεδομένων**:

ΑΞΙΟΛΟΓΗΣΗ ΕΚΠΑΙΔΕΥΣΗΣ ΣΕ 15,000 ΕΙΚΟΝΕΣ (30 ΕΠΟΧΕΣ)

In [ ]:
```
rid = random.randint(0, len(test))
randomCheck(rid)
print(rid)
```

```
REAL CAPTIONS: A skinny brown dog runs across the wet sand with his mouth open .
The dog is running through the wet sand .
a dog running on the beach
A dog walks on the beach .
A dog runs across sand .
PREDICTION CAPTION: a small white and white dog is running on the beach .
Sentence BLEU score:  0.4298326478766695
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_807814377.jpg
2651
```

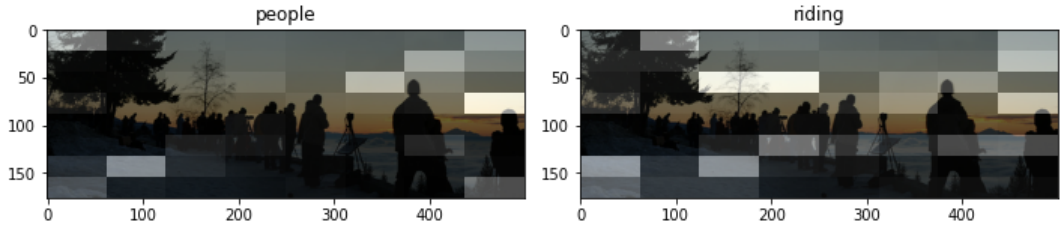
In [ ]:
```
Image.open(test[rid].replace("\\","/"))
```

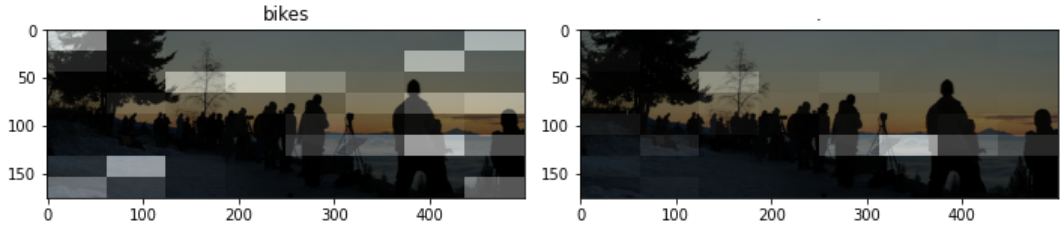
Out[ ]:

Αξίζει να σημειωθεί πόσο εύστοχη είναι η περιγραφή που παρήχθη για αυτήν την εικόνα.

In [ ]:
```
rid = random.randint(0, len(test))
randomCheck(rid)
print(rid)
```

```
REAL CAPTIONS: A line of people are standing on the edge of a snow covered path overlooking the clouds in the mountains
.
A group of photographers stands on a hillside in the snow with the sun just below the horizon .
A line of people are seen in silhouette on a snowy ridge when the sun is low in the sky .
A crowd of people are standing on a snow covered hill watching the sun go down .
A group of people are standing on a ledge overlooking low clouds .
PREDICTION CAPTION: people riding bikes .
Sentence BLEU score:  0.012562701534643611
```





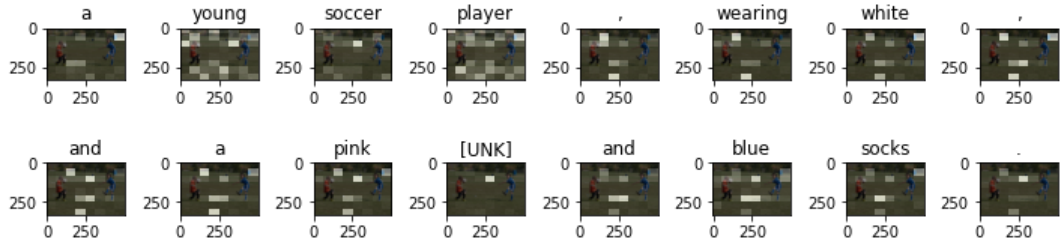

```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_151622523.jpg
1714
```

In [ ]:
```
Image.open(test[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(test))
randomCheck(rid)
print(rid)
```

```
REAL CAPTIONS: Two children playing soccer on the field , one is wearing a blue shirt and the other is wearing a orange
shirt .
One soccer player in an orange uniform getting hit in the face by a ball kicked by a player in blue .
A boy in a blue jersey kicks a soccer ball into the head of a boy in an orange jersey .
Two boys practicing passing a soccer ball .
Two boys play soccer against each other .
PREDICTION CAPTION: a young soccer player , wearing white , and a pink [UNK] and blue socks .
Sentence BLEU score:  0.059004436212568886
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_176959198.jpg
2094
```

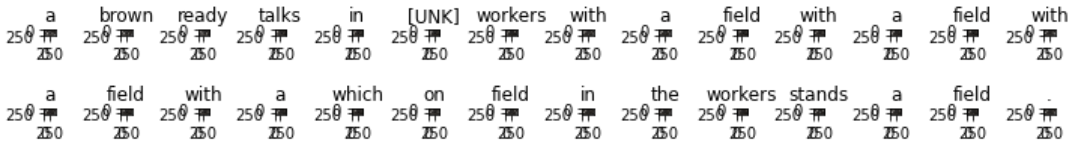
In [ ]:
```
Image.open(test[rid].replace("\\","/"))
```

Out[ ]:

ΑΞΙΟΛΟΓΗΣΗ ΕΚΠΑΙΔΕΥΣΗΣ ΣΕ ΟΛΟΚΛΗΡΟ ΤΟ DATASET (40 ΕΠΟΧΕΣ)

In [ ]:
```
randomCheck(2651)
```

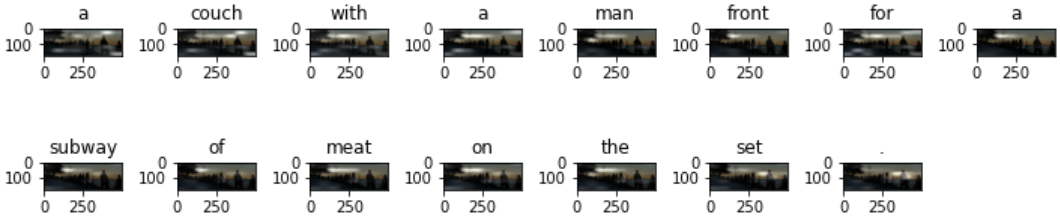
```
REAL CAPTIONS: A skinny brown dog runs across the wet sand with his mouth open .
The dog is running through the wet sand .
a dog running on the beach
A dog walks on the beach .
A dog runs across sand .
PREDICTION CAPTION: a brown ready talks in [UNK] workers with a field with a field with a which on field in
the workers stands a field .
Sentence BLEU score:  0.01906336169302367
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_807814377.jpg
```

In [ ]:
```
randomCheck(1714)
```

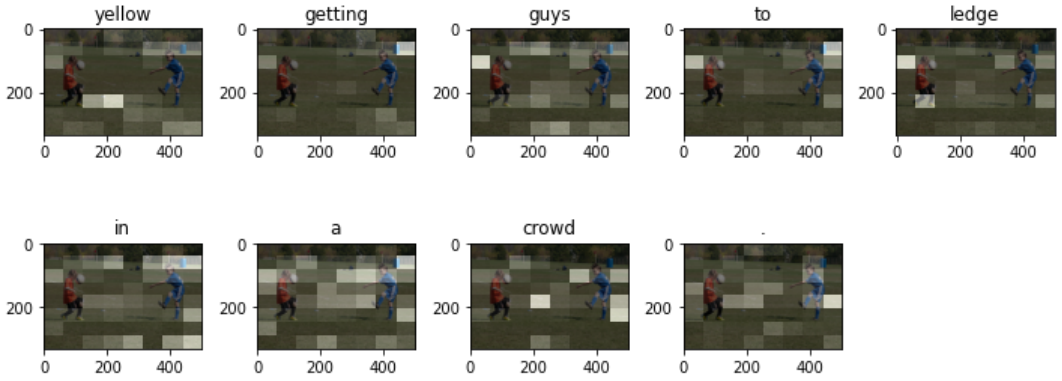
```
REAL CAPTIONS: A line of people are standing on the edge of a snow covered path overlooking the clouds in the mountains
.
A group of photographers stands on a hillside in the snow with the sun just below the horizon .
A line of people are seen in silhouette on a snowy ridge when the sun is low in the sky .
A crowd of people are standing on a snow covered hill watching the sun go down .
A group of people are standing on a ledge overlooking low clouds .
PREDICTION CAPTION: a couch with a man front for a subway of meat on the set .
Sentence BLEU score:  0.07349855329608217
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_151622523.jpg
```

In [ ]:
```
randomCheck(2094)
```

```
REAL CAPTIONS: Two children playing soccer on the field , one is wearing a blue shirt and the other is wearing a orange
shirt .
One soccer player in an orange uniform getting hit in the face by a ball kicked by a player in blue .
A boy in a blue jersey kicks a soccer ball into the head of a boy in an orange jersey .
Two boys practicing passing a soccer ball .
Two boys play soccer against each other .
PREDICTION CAPTION: yellow getting guys to ledge in a crowd .
Sentence BLEU score:  0.1099930547716519
```




```
/content/drive/My Drive/Colab Notebooks/NN/EX3/images/image_dir/_176959198.jpg
```

In [ ]:
```
##allReferences = []
allHypotheses = []

for i in tqdm(list(set(test))[:100]):
    imgname = i.split('/')[-1]
    ##allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 100/100 [01:56<00:00,  1.17s/it]
```

In [ ]:
```
with open(os.path.abspath('.') + "/drive/My Drive/Colab Notebooks/NN/EX3/checkpoints_final/hyp15.pkl", 'wb') as f:
    pickle.dump(allHypotheses, f)
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.06706724161510844
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.09833208532016605
```

Από τα παραπάνω μεμονωμένα δείγματα και τον υπολογισμό corpus BLEU επί ποσοστού του **κοινού** validation set, είναι εμφανές ότι το μοντέλο εκπαιδευμένο με 15,000 εικόνες για 30 εποχές είναι ανώτερο, και συνεπώς **αυτό είναι η τελική μας επιλογή**.

Βελτίωση Τελικού Μοντέλου με Beam Search

Ως τελευταίο βήμα βελτίωσης, θα αντικαταστήσουμε την μέθοδο sentence generation της συνάρτησης evaluate (tf.categorical) με αναζήτηση ακτίνας (beam search). Εφ' όσον αυτή η βελτίωση αφορά ήδη εκπαιδευμένα μοντέλα, θα την εφαρμόσουμε στο καλύτερό μας.

Σε κάθε βήμα βάζουμε στο μέτωπο αναζήτησης τις beam_width πιθανότερες επακόλουθες για κάθε λέξη (βάσει αθροίσματος πιθανοτήτων), διατη΄ρωντας και την αντίστοιχη ακολουθία και hidden state. Στην συνέχεια κρατάμε τις beam_width κορυφαίες βάσει κανονικοποιημένου αθροίσματος με διαίρεση κατά το μήκος της αντίστοιχης πρότασης.

In [ ]:
```
def beam_evaluate(image, beam_width = 10):
    init_hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0],
                                                 -1,
                                                 img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([word_to_index('<start>')], 0)
    allResults = []

    search_front = []
    search_front.append((word_to_index('<start>'), 0, [], init_hidden))
    for _ in range(max_length):
        allcandidates = []
        for word, prob, path, hid in search_front:
            if(index_to_word(word).numpy() == b"<end>"):
                allcandidates.append((word, prob, path, hid))
                continue
```

```
            predictions, hidden, attention_weights = decoder(tf.expand_dims([word_to_index(index_to_word(word))], 0),
                                                             features,
                                                             hid)

            ## changes start here
            predictions = tf.nn.softmax(predictions)
            predictions = tf.math.log(predictions)
            top_preds = tf.math.top_k(input = predictions, k = beam_width)
            pred_probabilities = top_preds[0][0].numpy()
            pred_words = top_preds[1][0].numpy()
            paths = [path + [word]] * beam_width
            hiddens = [hidden] * beam_width
            values = [i + prob for i in pred_probabilities]
            allcandidates.extend(zip(pred_words, values, paths, hiddens))


        allcandidates = sorted(allcandidates, key = lambda f: f[1] / len(f[2]))
        search_front = allcandidates[-beam_width:]

        top_pred = search_front[-1]
        top_pred_word = top_pred[0]
        predicted_word = tf.compat.as_text(index_to_word(top_pred_word).numpy())
        '''
        if predicted_word == '<end>':
            result = []
            top_pred_path = top_pred[2]
            top_pred_val = top_pred[1]
            for i in top_pred_path:
                result.append(tf.compat.as_text(index_to_word(i).numpy()))
            result.remove("<start>")
            allResults.append((result, top_pred_val, top_pred_path))
        '''
        for i in search_front:
            r = []
            w = i[0]
            pw = tf.compat.as_text(index_to_word(w).numpy())
            if pw == '<end>' or pw == b"end":
                pp = i[2]
                pv = i[1]
                for j in pp:
                    r.append(tf.compat.as_text(index_to_word(j).numpy()))
                r.remove("<start>")
                allResults.append((r, pv, pp))

    allResults = sorted(allResults, key = lambda f: f[1] / len(f[2]))
    return allResults[-1][0]
```

BLEU Scores of the Train Validation Set

Παρακάτω μερικά τυχαία captions με την νέα μέθοδο sentence generation:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A man with red shades on is holding a blowtorch while touching some type of sculpture with two onlookers
.
A man crouches in front of a sculpture while holding a blowtorch and a paintbrush .
A man is working on a sculpture with a paintbrush and a torch .
A father and son watch a man welding on a statue .
A man in sunglasses is painting a sculpture .
PREDICTION CAPTION: a child on his side of a bag .
Sentence BLEU score:  0.1202622344844899
C:\Users\anton\Documents\hmmy_mathimata\neurwnika\ergasia_3/image_dir/_614931387.jpg
```

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Elderly man with cane bends down to look at some plants and is steadied from behind .
A man and a woman are standing behind an elderly man who is looking at a bush .
A man holds up an older man as the older man bends down to check out plants .
An older man in a white short-sleeve shirt admiring a bush .
Elderly man with a cane bends over near a man and woman .
PREDICTION CAPTION: two men are standing in a river with their home .
Sentence BLEU score:  0.11597746388529283
C:\Users\anton\Documents\hmmy_mathimata\neurwnika\ergasia_3/image_dir/_413035738.jpg
```

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

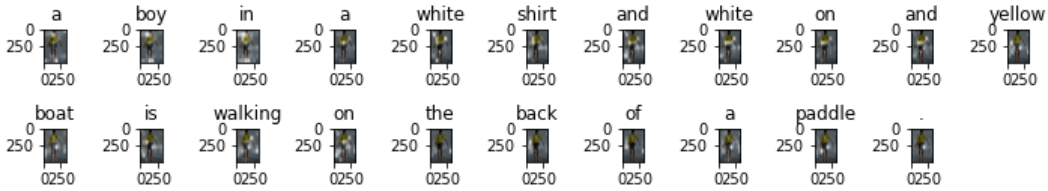
```
REAL CAPTIONS: The brightly dressed middle eastern women are conversing in the market .
Woman with red headwrap at a market with a woman with a brown headwrap .
two woman wearing different colored scarfs over their hair outside .
Two women with head garb looking at each other in a public area .
The two people have their heads covered with scarves .
PREDICTION CAPTION: a man in a hat , stands in the street .
Sentence BLEU score:  0.11808165345282427
C:\Users\anton\Documents\hmmy_mathimata\neurwnika\ergasia_3/image_dir/_114541760.jpg
```

In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A young boy in a yellow rash guard is walking on the shore carrying a surfboard .
a young boy wearing a yellow shirt walking on the beach carrying a surfboard
A boy in a yellow shirt walks with his surfboard out of the ocean .
A boy in a yellow shirt is walking on a beach holding a surfboard .
a little boy at the beach with a surfboard
PREDICTION CAPTION: a boy in a white shirt and white on and yellow boat is walking on the back of a paddle .
Sentence BLEU score:  0.28197974007516224
```

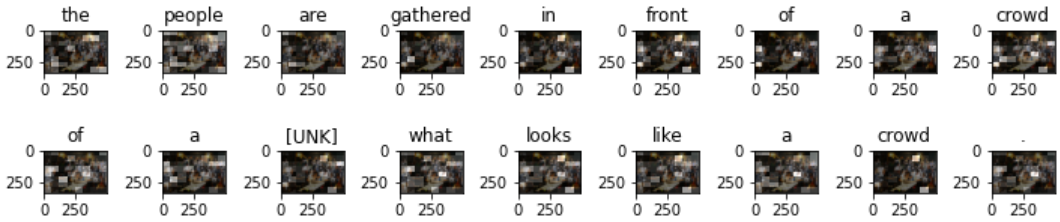


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: A crowd of people stand in a dark room , many looking at a bright light , while sound equipment and a rec
ord player sit in the foreground .
A group of people are standing around dj equipment including a record player and a sound machine .
A group of people are standing around music production equipment .
A group of people dancing in a club in front of a disk jockey .
People watching performance in a nightclub .
PREDICTION CAPTION: the people are gathered in front of a crowd of a [UNK] what looks like a crowd .
Sentence BLEU score:  0.2964098952966254
```

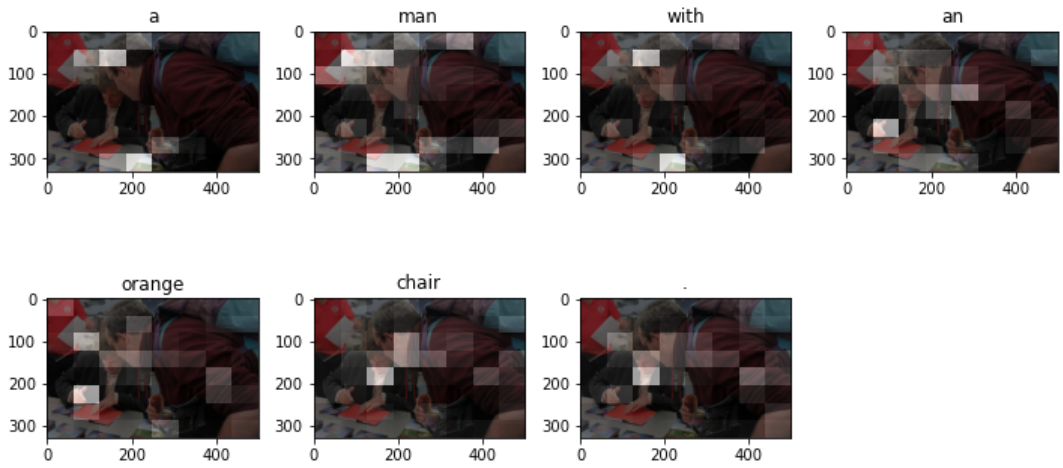


In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
rid = random.randint(0, len(img_name_val))
randomCheck(rid)
```

```
REAL CAPTIONS: Tom Cruise signs some autographs on a book during an autograph session in January during the event of the
raise funds for Children 's event held in Miami .
Jeremy Clarkson ( from top gear ) signing a book while a man with a cannon camera , burgundy hoodie , and blue and purpl
e backpack leaning over to watch .
An author is signing his book as a photographer looks on .
An author signing his book for a fan .
A photographer gets a book signed .
PREDICTION CAPTION: a man with an orange chair .
Sentence BLEU score:  0.3150738859318846
```



In [ ]:
```
Image.open(img_name_val[rid].replace("\\","/"))
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a man is sitting on a horse .
```

In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a crowd watches as a crowd watches .
```

In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
randomImage = random.choice(testList)
randomTestCheck(randomImage)
```

```
Prediction Caption: a brown dog is in water .
```

In [ ]:
```
Image.open("./image_dir/" + randomImage)
```

Out[ ]:

In [ ]:
```
result, attention_plot = evaluate("./image_dir/_340139192.jpg")
Image.open("./image_dir/_340139192.jpg")
```

Out[ ]:

In [ ]:
```
print(result)
```

```
['a', 'man', 'standing', 'at', 'home', '.']
```

Υπολογισμός νέου corpus BLEU και σύγκριση:

Χωρίς Beam Search

In [ ]:
```
allReferences = []
allHypotheses = []

for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    allReferences.append([c.split() for c in utilDict[imgname]])
    allHypotheses.append(evaluate(i)[0])
```

```
100%|██████████| 1500/1500 [09:40<00:00,  2.58it/s]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.09089194047844905
```

Με Beam Search

In [ ]:
```
allReferences = []
allHypotheses = []


for i in tqdm(set(img_name_val)):
    imgname = i.split('/')[-1]
    try:
        allHypotheses.append(beam_evaluate(i, 3))
    except:
        continue
    allReferences.append([c.split() for c in utilDict[imgname]])
```

```
100%|██████████| 1500/1500 [3:52:51<00:00,  9.31s/it]
```

In [ ]:
```
print('Corpus BLEU score: ', corpus_bleu(list_of_references = allReferences, hypotheses = allHypotheses, weights = weigh
```

```
Corpus BLEU score:  0.14760017116660004
```

Παρατηρούμε **αξιοσημείωτη αύξηση του Corpus BLEU score.**

Test on External Images

In [ ]:
```
image_url = 'https://tensorflow.org/images/surf.jpg'
image_extension = image_url[-4:]
image_path = tf.keras.utils.get_file('image'+image_extension, origin=image_url)

result = beam_evaluate(image_path)
print('Prediction Caption:', ' '.join(result))
# opening the image
Image.open(image_path)
```

```
Prediction Caption: a person in a white and blue kayak keeps on the waves .
```

Out[ ]:

Generate Predictions for the Competition Test Set

In [ ]:
```
## predictions for entire test set

hyp_no_bs = []
hyp_bs = []

counter = 0
for t in tqdm(testList):
    img = os.path.abspath('.') + "/image_dir/" + t
    base_pred, _ = evaluate(img)
    hyp_no_bs.append(base_pred)
    try:
        hyp_bs.append(beam_evaluate(img, 3))
    except:
        hyp_bs.append(base_pred)
        counter += 1
```

```
100%|██████████| 2000/2000 [5:25:51<00:00,  9.78s/it]
```

In [ ]:
```
print(counter)
```

```
148
```

In [ ]:
```
with open(os.path.abspath('.') + "/pickle_competition/all_hypotheses_beam.pkl", 'wb') as f:
    pickle.dump(hyp_bs, f)
with open(os.path.abspath('.') + "/pickle_competition/all_hypotheses.pkl", 'wb') as f:
    pickle.dump(hyp_no_bs, f)
```

Συμπεράσματα

Συνοπτική περιγραφή της διαδικασίας που ακολουθήσαμε

Επειδή οι χρόνοι εκπαίδευσης των νευρωνικών δικτύων σε αυτήν την εργασία είναι αρκετά μεγάλοι, και χρησιμοποιούνται checkpoints και σταδιακή εκπαίδευση, προκειμένου αυτή να βασίζεται στο ίδιο σύνολο δεδομένων σε κάθε βήμα (όπως είναι λογικό), οπουδήποτε εισάγεται τυχαιότητα χρησιμοποιήθηκε το ίδιο random seed.

Προκειμένου να έχουμε σημείο αναφοράς για τις επιδόσεις, πρώτα εκπαιδεύσαμε το δίκτυο ακριβώς όπως και στο **tutorial**, με μοναδική αλλαγή το dataset.

Η πρώτη δοκιμή βελτίωσης συνίσταται στην **αύξηση των δεδομένων εκπαίδευσης** κατά 50% (9000 αντί για 6000 εικόνες) και στην αντικατάσταση του συνελικτικού δικτύου - encoder που χρησιμοποιείται από το tutorial (InceptionV3), με το ResNet152V2, όπως υπαγορεύει η εκφώνηση για την ομάδα μας. $31 \equiv 3 \mod 4$

Ύστερα, επιχειρήσαμε να βελτιώσουμε την ποιότητα των captions κάνοντας καλύτερη **προεπεξεργασία** των περιγραφών αναφοράς. Συγκεκριμένα:

- ορίσαμε διαφορετικά φράγματα ελαχίστου και μεγίστου μήκους αποδεκτών captions,
- κανονικοποιήσαμε τα κείμενα, και
- αυξήσαμε το μέγιστο μέγεθος vocabulary διαφορετικών λέξεων σε 7,000. Αρχικά δοκιμάσαμε διπλασιασμό του vocabulary σε 10,000 και φάνηκε να βοηθά, όμως ο απαιτούμενος χρόνος ικανοποιητικής ελάττωσης του loss ήταν πολύ μεγάλος, και συνεπώς επιλέξαμε να χρησιμοποιήσουμε 7,000.

Επιπλέον, στο εξής χρησιμοποιήσαμε ακόμη περισσότερα δεδομένα για την εκπαίδευση (15,000 δείγματα συνολικά), καθώς είδαμε ότι η αύξηση δεδομένων βελτιώνει την επίδοση.

Ως προς την κανονικοποίηση, επειδή το generative model που θα φτιάχναμε πρέπει να παράγει φυσικές προτάσεις, χρειάζεται να παραμείνουν στο vocabulary όλα τα μέρη του λόγου, και συνεπώς δεν εφαρμόσαμε stemming ή αποκοπή λέξεων μικρού μήκους. Αφαιρέσαμε όμως τα σημεία στίξης, εκτός των <, > που σηματοδοτούν αφετηρία και λήξη και του κόμματος και της τελείας που συνεισφέρουν νοηματικά στην πρόταση, και βρίσκονται σε αφθονία και στα reference captions. Η μετατροπή όλων των γραμμάτων σε πεζά γίνεται ήδη από το tutorial.

Στο τρίτο βήμα βελτιστοποίησης, χρησιμοποιήσαμε προεκπαιδευμένα **word embeddings** από το Gensim, αντί να δημιουργούμε δικά μας διανύσματα κατά την εκπαίδευση. Αυτό αναμέναμε να ελαττώσει τον χρόνο εκπαίδευσης καθώς μειώθηκε το πλήθος των εκπαιδευόμενων βαρών. Συγκεκριμένα, δοκιμάσαμε διαστάσεις 50 και 300.

Τέταρτον, προσθέσαμε επίπεδα Dropout στο δίκτυο ώστε να αποφευχθεί το overfitting. Η μοναδική τροποποίηση του βήματος αυτού σε σχέση με το προηγούμενο βρίσκεται στο μοντέλο του decoder, όπου προσθέσαμε Dropout layers ύστερα από κάθε fully connected dense layer. Στον encoder δεν εφαρμόσαμε dropout, αφού σε αυτόν δεν εκτελείται εκπαίδευση. Για το πείραμα αυτό χρησιμοποιήθηκε dropout rate 0.25, το οποίο είναι σύνηθες για RNN. Παρατηρήσαμε ταχύτερη μείωση του loss σεα αυτό το στάδιο, και συνεπώς διατηρήσαμε την αλλαγή.

Πέμπτον, δοκιμάσαμε πώς επιδρά στην επίδοση το πλήθος των μονάδων (units) καθώς και το είδος των επαναληπτικών επιπέδων. Αρχικά, δοκιμάσαμε να υποδιπλασιάσουμε το πλήθος των μονάδων του επιπέδου GRU του RNN decoder και στη συνέχεια δοκιμάσαμε να το αντικαταστήσουμε με επίπεδο LSTM. Αλλαγές έγιναν μόνο στο μοντέλο του decoder και τον ορισμό της σταθεράς units. Τόσο τα BLEU scores, όσο και οι περιγραφές μεμονωμένων δειγμάτων, μας οδήγησαν στην απόφαση να μην κρατήσουμε καμμία από τις αλλαγές που δοκιμάσθηκαν σε αυτό το βήμα.

Ύστερα από όλες τις δοκιμές, το τελικό μοντέλο διαμορφώθηκε ως εξής:

- ResNet152V2 encoder.
- Προεπεξεργασία κειμένου με αφαίρεση captions μήκους εκτός του διαστήματος [6, 31].
- Χρήση των προεκπαιδευμένων embeddings glove-wiki-gigaword-300 στον RNN decoder.
- Εφαρμογή dropout layers με 0.25 rate ύστερα από τα πυκνά επίπεδια του decoder.
- Διατήρηση του GRU cell του decoder ως είχε από το tutorial.

Για την εκπαίδευση του τελικού μοντέλου έγιναν δύο απόπειρες. Μία σε ολόκληρο το train dataset, και μία στο 50% περίπου αυτού, με 15,000 εικόνες.

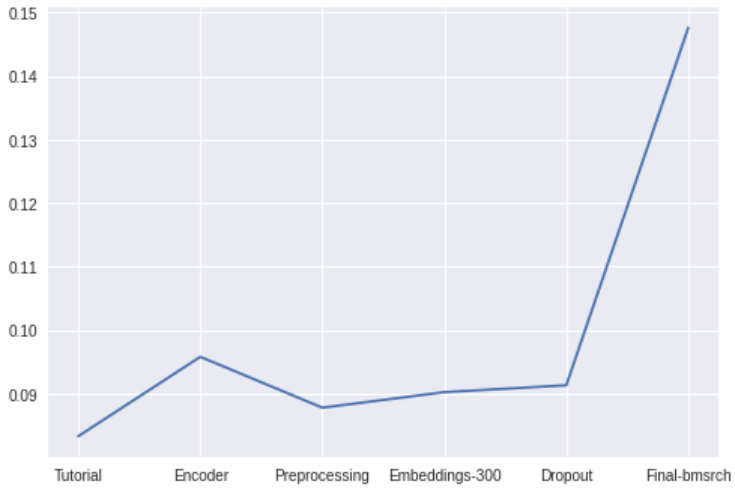
Πρέπει να τονισθεί ότι, παρ' όλο που σε κάθε βήμα εξετάζουμε εκτός των άλλων και δείγματα από το test set και εξωτερικές εικόνες, δεν λαμβάνουμε αποφάσεις για την κατασκευή του μοντέλου βάσει αυτών, παρά μόνο από τα αποτελέσματα, γενικά και μεμονωμένα, επί του validation set.

Αποτελέσματα

Στάδιο Βελτίωσης	Loss (epochs)	Corpus BLEU
Έτοιμο Μοντέλο	0.306 (30)	0.0833
Encoder	0.253 (30)	0.0958
Preprocessing	0.402 (20)	0.0878
Embeddings 50	0.547 (30)	0.0853
Embeddings 300	0.339 (30)	0.0902
Dropout	0.339 (30)	0.0913
GRU 256	0.444 (30)	0.0856
LSTM 512	0.325 (30)	0.0787
Final	0.339 (30)	0.0983
Final bmsrch	-	0.1476

In [ ]:
```
cbscores = {"Tutorial": 0.08326342410126537, "Encoder": 0.09576449897236472, "Preprocessing": 0.08775300382288816, "Embe
```

In [ ]:
```
plt.style.use("seaborn")
plt.plot(*zip(*cbscores.items()))

plt.show()
```



Παρατηρήσεις

- Φαινομενικά στο δεύτερο βήμα βελτίωσης (αλλαγή encoder) το αποτέλεσμα είναι καλύτερο από τα αμέσως επόμενα (βάσει του bleu score μόνο), ωστόσο εξετάζοντας μεμονωμένα παραδείγματα διαπιστώσαμε ότι η πραγματική ποιότητα των περιγραφών βελτιώθηκε στα επόμενα βήματα.

- Στο τρίτο βήμα βελτιστοποίησης παρατηρήσαμε πως τα μικρού μεγέθους embeddings έχουν ως αποτέλεσμα ελάττωση του corpus bleu. Στην πρώτη εκπαίδευση με embeddings που πραγματοποιήσαμε, ύστερα από 30 εποχές η τιμή loss παρέμεινε σε υπερβολικά υψηλό επίπεδο (0.547). Αυτό πιθανώς οφείλεται στην μείωση των διαστάσεων των embeddings. Παρ' όλο που χρησιμοποιήσαμε προεκπαιδευμένα -και θεωρητικά καλύτερα από αυτά που προκύπτουν από την δική μας εκπαίδευση- τα διανύσματα για κάθε λέξη είναι αρκετά μικρότερα (50 αντί 256) και συνεπώς μπορεί να περιέχουν λιγότερη πληροφορία. Αυξάνοντας τις διαστάσεις, βελτιώθηκε σημαντικά η τιμή loss και το corpus bleu score.

- Στο πέμπτο βήμα βελτιστοποίησης (επεξεργασία των recurrent layers), τόσο τα BLEU scores, όσο και οι περιγραφές μεμονωμένων δειγμάτων, μας οδήγησαν στην απόφαση να μην κρατήσουμε καμία από τις αλλαγές που δοκιμάσθηκαν.

Περαιτέρω Βελτιώσεις

Πώς θα μπορούσαμε πιθανώς να βελτιώσουμε ακόμη περισσότερο την επίδοση:

- Αφαιρώντας τα σημεία στίξης που κρατήσαμε, αφού δεν υπάρχουν στο λεξικό των προεκπαιδευμένων embeddings του τελικού μοντέλου. Αυτό βεβαίως δεν θα είχε άμεση επίδραση στο BLEU score.
- Χρησιμοποιώντας καλύτερο image classification cnn - encoder. Στην τεκμηρίωση του Keras (https://keras.io/api/applications/) αναφέρονται ορισμένα ελαφρώς καλύτερα μοντέλα, όπως το Xception και το EfficientNetB6.
- Στην εφαρμογή των προεκπαιδευμένων embeddings, αντιμετωπίζοντας καλύτερα τις «άγνωστες» λέξεις. Στα πειράματά μας για αυτές θέταμε το μηδενικό διάνυσμα, αντ' αυτού θα μπορούσαμε να δοκιμάσουμε διαφορετικό μοντέλο του gensim, όπως το FastText, το οποίο κωδικοποιεί και "out-of-corpus" λέξεις (https://radimrehurek.com/gensim/models/fasttext.html, https://arxiv.org/abs/1607.04606).
- Χρησιμοποιώντας περισσότερες μονάδες (units) στο RNN cell: εφ' όσον είδαμε μείωση της επίδοσης μειώνοντας τα units του GRU cell, ίσως αύξηση αυτών σε περισσότερα από 512 να ωφελούσε. Η δοκιμή αυτή δεν έγινε διότι περισσότερα units οδηγούν σε αύξηση του χρόνου εκπαίδευσης.
- Εξασφαλίζοντας την πιστότητα των reference captions. Μία εικόνα που έχει χρησιμοποιηθεί ανωτέρω σε ένα από τα πειράματά μας, έχει δύο λανθασμένα captions. Αναγνωρίζουν το εικονιζόμενο πρόσωπο η μία ως τον Tom Cruise και η άλλη ως τον Jeremy Clarkson, ενώ είναι ο James May του Top Gear. Αυτά τα λάθη ίσως δεν έχουν μεγάλη σημασία στο περιεχόμενο της εικόνας, αλλά αναδεικνύουν ότι δεν είναι όλες οι περιγραφές αναφοράς του dataset αξιόπιστες.
- Εφαρμόζοντας beam search με μεγαλύτερο πλάτος ακτίνας.