In [0]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%
b&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2
www.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly
ttps%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:

```
# ## https://stackoverflow.com/questions/32834731/how-to-delete-a-file-by-extension-in-python
# ## to delete all .npy files
# import os
# dir_name = '/content/drive/My Drive/Colab Notebooks/Medical data case study'
# test = os.listdir(dir_name)

# for item in test:
#     if item.endswith(".npy"):
#         os.remove(os.path.join(dir_name, item))
```

In [0]:

```
import tensorflow as tf

# You'll generate plots of attention in order to see which parts of an image
# our model focuses on during captioning
import matplotlib.pyplot as plt
from tqdm import tqdm
# Scikit-learn includes many helpful utilities
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
import os
# import tensorflow as tf
import xml.etree.ElementTree
import numpy as np
import pandas as pd
import re
import re
import numpy as np
import os
import time
import json
from glob import glob
from PIL import Image
import pickle
```

In [0]:

```
print(tf.__version__)
```

```
2.2.0-rc2
```

In [0]:

```
tf.test.gpu_device_name()
```

Out[0]:

```
'/device:GPU:0'
```

In [0]:

In [0]:

```
!pip install -q tqdm
```

In [0]:

```
image_folder = '/content/drive/My Drive/Colab Notebooks/Medical data case study'
annotation_folder = '/content/drive/My Drive/Colab Notebooks/Medical data case study/ecgen-radiology'
```

In [0]:

```
## https://github.com/wisdal/diagnose-and-explain/blob/master/prepare_dataset.py

def extract_data():
    all_findings = []
    all_impressions = []
    all_img_names = []
    rids = []

    total_count = 0 # Count of reports available in the dataset
    no_image_count = 0 # Count of reports having no associated chest image
    no_impression_count = 0 # Count of reports having an empty "Impression" section
    no_findings_count = 0 # Count of reports having an empty "Findings" section

    # Storing impressions, findings and the image names in vectors
    for file in tqdm(os.listdir(annotation_folder)):
    # for file in tqdm(annotation_folder):
        total_count += 1
        file = os.path.abspath(annotation_folder) + '/' + file
        e = xml.etree.ElementTree.parse(file).getroot()

        rid = e.find('pmcId').get('id') # Report Id
        # We choose to ignore reports having no associated image
        image_id = e.find('parentImage')
        if image_id is None:
            no_image_count += 1
            continue

        image_id = image_id.get('id')
#         image_name = os.path.abspath('.') + '/' + image_id + '.png'
        image_name = image_folder + '/' + image_id + '.png'
        findings = ''
        impression = ''

        # Parsing "Impression" and "Findings"
        for element in e.findall('MedlineCitation/Article/Abstract/AbstractText'):
            if element.get('Label') == 'FINDINGS':
                findings = element.text
            if element.get('Label') == 'IMPRESSION':
                impression = element.text

        # Sanity check: Skip this report if it has an empty "Impression" section
        if findings is None:
            no_findings_count += 1
            #findings = 'No finding'
            continue
        if impression is None:
            no_impression_count += 1
            continue

        # Transforming findings and impressions into lists of sentences
        # https://stackoverflow.com/questions/21840389/python-regular-expression-remove-period-from-number-at-end-of-sentence
        findings = findings.replace("XXXX", "") #"XXXX" represents information anonymized
        findings=re.sub('((\d+)[\.])(?!([\d]+))','\g<2>',findings)
        findings = re.sub(" \d+", " ", findings)
#         sentences = findings.split('.')
        sentences = findings
#         del sentences[-1]
#         sentences = ['<start> ' + sentence + ' <end>' for sentence in sentences]
        # sentences = ['<start> ' + sentences + ' <end>']
        findings = sentences

        impression = impression.replace("XXXX", "") #"XXXX" represents information anonymized
        impression=re.sub('((\d+)[\.])(?!([\d]+))','\g<2>',impression)
```

```
        impression = re.sub(" \d+", " ", impression)
#https://www.tutorialspoint.com/How-to-remove-specific-characters-from-a-string-in-Python
        impression=impression.replace("1", "")
#         sentences = impression.split('.')
#         del sentences[-1]
        sentences = impression
#         sentences = ['<start> ' + sentence + ' <end>' for sentence in sentences]
        sentences = ['<start> ' + sentences + ' <end>' ]
        impression = sentences

        #appending to vectors
        all_img_names.append(image_name)
        all_findings.append(findings)
        all_impressions.append(impression)
        rids.append(rid)

    print("Number of reports available:", total_count)
    print("Number of reports selected:", len(all_img_names))
    print("Number of reports not having images (skipped):", no_image_count)
    print("Number of reports with Impression section empty (skipped):", no_impression_count)
    print("Number of reports with Findings section empty:", no_findings_count)
    print("Total skipped:", no_image_count + no_impression_count + no_findings_count)

    return all_findings, all_impressions, all_img_names, rids
```

In [19]:

```
all_findings, all_impressions, all_img_names, report_id=extract_data()
```

```
100%|██████████| 3965/3965 [14:40<00:00,  4.50it/s]

Number of reports available: 3965
Number of reports selected: 3341
Number of reports not having images (skipped): 104
Number of reports with Impression section empty (skipped): 6
Number of reports with Findings section empty: 514
Total skipped: 624
```

In [0]:

```
# df=pd.read_csv("/content/drive/My Drive/Colab Notebooks/x_ray_image_with_report (1).csv")
```

In [28]:

```
all_impressions[0]
```

Out[28]:

```
['<start> No acute cardiopulmonary abnormalities. <end>']
```

In [0]:

```
def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (512, 624))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    return img, image_path
```

In [0]:

```
## Converting Lists to Dataframe
## https://stackoverflow.com/questions/30522724/take-multiple-lists-into-dataframe
# df = pd.DataFrame(
#     { 'report_id':report_id,
#         'findings': all_findings,
#      'impressions': all_impressions,
#      'image_names': all_img_names
#     })
# df.to_csv('x_ray_image_with_report.csv')
```

```
# df=pd.read_csv('x_ray_image_with_report.csv')
```

In [49]:

```
impressions=[]
for i in all_impressions:
  for j in i:
    impressions.append(j)

print(impressions[:1])
all_impressions=impressions
```

```
['<start> No acute cardiopulmonary abnormalities. <end>']
```

In [23]:

```
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)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
```

In [0]:

```
# Get unique images
# all_findings, all_impressions, all_img_names, report_id

encode_train = sorted(set(all_img_names))

# 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).bat
ch(16)
```

In [25]:

```
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())
```

```
209it [07:46,  2.23s/it]
```

In [0]:

```
# Find the maximum length of any caption in our dataset
def calc_max_length(tensor):
    return max(len(t) for t in tensor)

# Choose the top 5000 words from the vocabulary
top_k = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                  oov_token="<unk>",split=' ', char_level=False,
                                                  filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')
tokenizer.fit_on_texts(all_impressions)
train_seqs = tokenizer.texts_to_sequences(all_impressions)
# tokenizer.fit_on_texts(df['impressions'])
# train_seqs = tokenizer.texts_to_sequences(df['impressions'])
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'
```

In [55]:

```
len(tokenizer.word_index)
```

Out[55]:

```
1225
```

In [62]:

```
print(all_impressions[3])
print(tokenizer.index_word[10])
print(tokenizer.word_index['<start>'])
print(tokenizer.index_word[1224])
print(tokenizer.word_index['<start>'])
```

```
<start> No acute disease. <end>
the
2
contusion
2
```

In [0]:

```
# Create the tokenized vectors
# train_seqs = tokenizer.texts_to_sequences(all_impressions)
# Pad each vector to the max_length of the captions
# If you do not provide a max_length value, pad_sequences calculates it automatically
cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding='post')

# Calculates the max_length, which is used to store the attention weights
max_length = calc_max_length(train_seqs)
```

In [64]:

```
max_length
```

Out[64]:

```
114
```

In [67]:

```
train_seqs[10]
```

Out[67]:

```
[2, 4, 5, 34, 6, 16, 3]
```

In [0]:

```
# Create training and validation sets using an 80-20 split
img_name_train, img_name_val, cap_train, cap_val = train_test_split(all_img_names,
                                                                    cap_vector,
                                                                    test_size=0.2,
                                                                    random_state=0)
```

In [69]:

```
len(img_name_train), len(cap_train), len(img_name_val), len(cap_val)
```

Out[69]:

```
(2672, 2672, 669, 669)
```

In [0]:

```
# Feel free to change these parameters according to your system's configuration
```

```
BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
vocab_size = top_k + 1
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 [0]:

```
# 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 [0]:

```
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.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)

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

In [0]:

```
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)

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

    # attention_weights shape == (batch_size, 64, 1)
    # you get 1 at the last axis because you are applying score to self.V
    attention_weights = tf.nn.softmax(self.V(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


class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it using pickle
    # 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
```

```
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 [0]:

```
encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, vocab_size)
```

In [0]:

```
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_)
```

In [0]:

```
checkpoint_path = "./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 [0]:

```
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 [0]:

```
# 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 [79]:

```
len(tokenizer.word_index)
```

Out[79]:

1225

In [80]:

```
tokenizer.word_index['<pad>']
```

Out[80]:

0

In [81]:

```
tokenizer.index_word[1222]
```

Out[81]:

'thickness'

In [82]:

```
tokenizer.word_index['<start>']
```

Out[82]:

2

In [0]:

```
# tokenizer.index_word[1223]='<start>'
# tokenizer.word_index['<start>'] = 1223
# tokenizer.index_word[1224]='<end>'
# tokenizer.word_index['<end>'] = 1224
```

In [0]:

```
# tokenizer.word_index['<start> ']
```

In [0]:

```
@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([tokenizer.word_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 [0]:

```
# print(img_tensor.shape)
# print(target.shape)
```

In [89]:

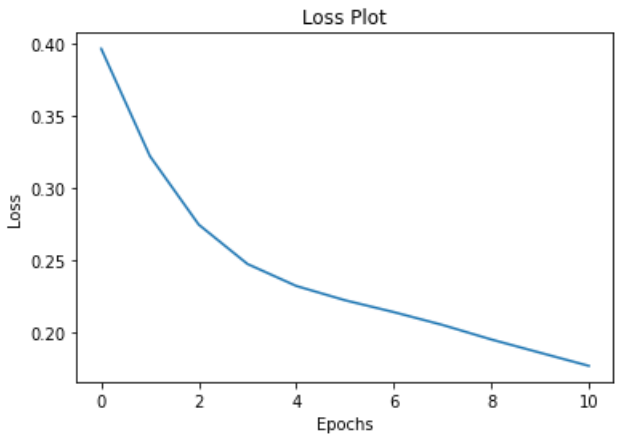
```
EPOCHS = 11
# with strategy.scope():
for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

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

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

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

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

```
Epoch 1 Batch 0 Loss 0.5977
Epoch 1 Loss 0.396087
Time taken for 1 epoch 368.83429646492004 sec

Epoch 2 Batch 0 Loss 0.3455
Epoch 2 Loss 0.321712
Time taken for 1 epoch 157.97896194458008 sec

Epoch 3 Batch 0 Loss 0.3333
Epoch 3 Loss 0.274314
Time taken for 1 epoch 162.1726553440094 sec

Epoch 4 Batch 0 Loss 0.1958
Epoch 4 Loss 0.247043
Time taken for 1 epoch 163.10213565826416 sec

Epoch 5 Batch 0 Loss 0.2843
Epoch 5 Loss 0.231827
Time taken for 1 epoch 162.9009621143341 sec

Epoch 6 Batch 0 Loss 0.1763
Epoch 6 Loss 0.221931
Time taken for 1 epoch 162.6929795742035 sec
```

```
Epoch 7 Batch 0 Loss 0.1965
Epoch 7 Loss 0.213717
Time taken for 1 epoch 162.3735854625702 sec

Epoch 8 Batch 0 Loss 0.1609
Epoch 8 Loss 0.204830
Time taken for 1 epoch 162.4220209121704 sec

Epoch 9 Batch 0 Loss 0.1808
Epoch 9 Loss 0.194778
Time taken for 1 epoch 162.6420075893402 sec

Epoch 10 Batch 0 Loss 0.2129
Epoch 10 Loss 0.185614
Time taken for 1 epoch 162.40754747390747 sec

Epoch 11 Batch 0 Loss 0.1838
Epoch 11 Loss 0.176504
Time taken for 1 epoch 162.49075555801392 sec
```

In [90]:

```
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.show()
```




In [0]:

```
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([tokenizer.word_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()
        result.append(tokenizer.index_word[predicted_id])

        if tokenizer.index_word[predicted_id] == ' <end>':
            return result, attention_plot
```

```
        dec_input = tf.expand_dims([predicted_id], 0)

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

In [0]:

```
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 l in range(len_result):
        temp_att = np.resize(attention_plot[l], (8, 8))
        ax = fig.add_subplot(len_result//2, len_result//2, l+1)
        ax.set_title(result[l])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

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

In [104]:

```
# captions on the validation set
rid = np.random.randint(0, len(img_name_val))   #from live session code on URL Shortner
image = img_name_val[rid]
real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid] if i not in [0]])
result, attention_plot = evaluate(image)

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

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-104-8d5cf176abf5> in <module>()
      2 image = img_name_val[rid]
      3 real_caption = ' '.join([tokenizer.index_word[i] for i in cap_val[rid] if i not in [0]])
----> 4 result, attention_plot = evaluate(image)
      5
      6 print ('Real Caption:', real_caption)

<ipython-input-102-e6a8331b4de0> in evaluate(image)
     16         predictions, hidden, attention_weights = decoder(dec_input, features, hidden)
     17
---> 18         attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()
     19
     20

ValueError: could not broadcast input array from shape (252) into shape (64)
```

In [106]:

```
from IPython.display import Image
Image(filename=image)
```

Out[106]: