Basic_Model

July 2, 2020

[3]:

```
import pandas as pd
import numpy as np
import random as rn
import string
import matplotlib.pyplot as plt
from tqdm import tqdm
%matplotlib inline
import re
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import nltk
import warnings
warnings.filterwarnings('ignore')
```

[4]:

```
import tensorflow as tf
```

1 Read preprocessed data

[5]:

```
data = pd.read_csv("data.csv")
data.head()
```

[5]:

```
                                          image_name  … image_count
0    CXR1_1_IM-0001-3001.png,CXR1_1_IM-0001-4001.png  …           2
1      CXR10_IM-0002-1001.png,CXR10_IM-0002-2001.png  …           2
2    CXR100_IM-0002-1001.png,CXR100_IM-0002-2001.png  …           2
3  CXR1000_IM-0003-2001.png,CXR1000_IM-0003-1001…  …           3
4  CXR1001_IM-0004-1002.png,CXR1001_IM-0004-1001.png  …           2

[5 rows x 9 columns]
```

[6]:

```
data_projecttions = pd.read_csv("data_projections.csv")
data_projecttions.head()
```

[6]:

```
   uid                  filename projection
0    1  1_IM-0001-4001.dcm.png    Frontal
1    1  1_IM-0001-3001.dcm.png    Lateral
```

```
2     2  2_IM-0652-1001.dcm.png    Frontal
3     2  2_IM-0652-2001.dcm.png    Lateral
4     3  3_IM-1384-1001.dcm.png    Frontal
```

```
[7]: image_path = "img/"
```

2 Structure the data

limiting the data point to 2 images per data point, if we have 5 images, its 4+1 (all image + last image) so make it as 4 data points as below

if i have 5 images then
1 image + 5th image
2nd image + 5th image
3rd image + 5th image
4th image + 5th image
4 data point

like wise for other data point,

if i have 3 images then
1st + 3rd
2nd + 3rd
2 data point

if i have 4 images then
1st + 4th
2nd + 4th
3rd + 4th
3 data point

```
[ ]: data.shape
```

```
[ ]: (3851, 9)
```

```
[ ]: data['image_count'].value_counts()
```

```
[ ]: 2    3208
     1     446
     3     181
     4      15
     5       1
     Name: image_count, dtype: int64
```

Validate output:
2 images = 3208
3 images = 181*2*
4 images = 153

5 images = 1*4
Total = 3619

Total data point_

we should create duplicate dataframe separately to keep it in all dataset train test validate sets
1 images = 446
3619+446 = 4065

[8]:
```
def find_Fr_la(li):
    """Function to find the lateral anf frontal images
    Returns All frontal as list and Lateral as last image"""
    img_list = []
    last_img = ""
    for i in li:
        projection = data_projecttions[data_projecttions['filename'].str.
↪contains(re.search(r"\d.*\_IM-\d.*\.", i).group())]['projection'].values
        if "Lateral" == projection:
            last_img = i
        else:
            img_list.append(i)
    return img_list, last_img
```

[9]:
```
columns = ["image_1", "image_2", "impression"]
df = pd.DataFrame(columns = columns)
columns = ["image_1", "image_2", "impression"]
df_dup = pd.DataFrame(columns = columns)
no_lateral = 0
for item in tqdm(data.iterrows()):
    l = item[1]['image_name'].split(',')
    if len(l) > 2:
        li, last_img = find_Fr_la(l)
        if last_img == "":
            no_lateral +=1
            li, last_img = li[:-1], li[-1]
        for i in li:
            image_1 = i
            image_2 = last_img
            df = df.append(pd.Series([image_1, image_2, item[1]['impression']],␣
↪index = columns), ignore_index = True)
    elif len(l) == 2:
        image_1 = l[0]
        image_2 = l[1]
        df = df.append(pd.Series([image_1, image_2, item[1]['impression']],␣
↪index = columns), ignore_index = True)
    elif len(l) == 1:
        #creating duplicate dataframe separately to keep it in all dataset␣
↪train test validate
```

```
        df_dup = df_dup.append(pd.Series([l[0], l[0], item[1]['impression']],␣
 ↪index = columns), ignore_index = True)
print("Total Report without Lateral images {}".format(no_lateral))
```

```
3851it [00:13, 283.67it/s]

Total Report without Lateral images 1
```

```
[ ]: df.shape
```

```
[ ]: (3532, 3)
```

```
[ ]: df_dup.shape
```

```
[ ]: (446, 3)
```

3 Create start and end token

```
[10]: def add_start_end_token(data):
          # Combining all the above stundents
          preprocessed_reviews_eng = []

          # tqdm is for printing the status bar
          for sentance in tqdm(data.values):
              sentance = '<start> ' + sentance + ' <end>'
              preprocessed_reviews_eng.append(sentance.strip())
          return preprocessed_reviews_eng
```

```
[11]: df['impression'] = add_start_end_token(df['impression'])
      df_dup['impression'] = add_start_end_token(df_dup['impression'])
```

```
100%|          | 3532/3532 [00:00<00:00, 695081.96it/s]
100%|          | 446/446 [00:00<00:00, 443599.62it/s]
```

```
[ ]: df[['image_1','image_2', 'impression']].head()
```

```
[ ]:                     image_1  …
 impression
 0   CXR1_1_IM-0001-3001.png  …                       <start> normal chest x
 <end>
 1    CXR10_IM-0002-1001.png  …     <start> no acute cardiopulmonary process
 <end>
 2   CXR100_IM-0002-1001.png  …                    <start> no active disease
 <end>
 3  CXR1000_IM-0003-1001.png  …  <start> increased opacity in the right upper
```

```
l…
4  CXR1000_IM-0003-3001.png  …  <start> increased opacity in the right upper
l…

 [5 rows x 3 columns]
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3532 entries, 0 to 3531
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   image_1     3532 non-null   object
 1   image_2     3532 non-null   object
 2   impression  3532 non-null   object
dtypes: object(3)
memory usage: 82.9+ KB
```

[ ]:
```
df_dup[['image_1','image_2', 'impression']].head()
```

[ ]:
```
                    image_1  …
impression
0  CXR1003_IM-0005-2002.png  …  <start> retrocardiac soft tissue density the
a…
1  CXR1012_IM-0013-1001.png  …  <start> bibasilar airspace disease and
bilater…
2  CXR1024_IM-0019-1001.png  …               <start> no acute abnormality
<end>
3  CXR1026_IM-0021-2002.png  …     <start> no acute cardiopulmonary disease
<end>
4  CXR1029_IM-0022-1001.png  …  <start> no pneumonia heart size normal
scolios…

 [5 rows x 3 columns]
```

[ ]:
```
df_dup.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 446 entries, 0 to 445
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   image_1     446 non-null    object
 1   image_2     446 non-null    object
 2   impression  446 non-null    object
dtypes: object(3)
memory usage: 10.6+ KB
```

[12]:

```
image_name = []
for img in tqdm(data['image_name'].str.split(',')):
    for i in range(len(img)):
        image_name.append(img[i])
```

```
100%|     | 3851/3851 [00:00<00:00, 503766.48it/s]
```

[13]:

```
from tensorflow.keras.applications.inception_v3  import InceptionV3,␣
 ↪preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
```

[14]:

```
image_model = InceptionV3(include_top=False, weights='imagenet', pooling='avg')
input_layer = image_model.input
print(image_model.input)
output_layer = image_model.layers[-1].output
print(image_model.layers[-1].output)

image_features_model = Model(input_layer, output_layer)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applicatio
ns/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87916544/87910968 [==============================] - 1s 0us/step
Tensor("input_1:0", shape=(None, None, None, 3), dtype=float32)
Tensor("global_average_pooling2d/Identity:0", shape=(None, 2048), dtype=float32)
```

[15]:

```
img_tensor = []
for img in tqdm(image_name):
    img = tf.io.read_file(image_path + str(img))
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = preprocess_input(img)
    img_features = image_features_model(tf.constant(img)[None, :])
    img_features = tf.reshape(img_features,
                              (-1, img_features.shape[1]))
    img_tensor.append(img_features)
```

```
100%|     | 7470/7470 [15:56<00:00,  7.81it/s]
```

4 Train Test and Validation split

[16]:

```
##fixing numpy RS
np.random.seed(42)
##fixing tensorflow RS
tf.random.set_seed(32)
##python RS
rn.seed(12)
```

[17]:

```
i_train, input_test, o_train, output_test =␣
 ↪train_test_split(df[['image_1','image_2']].values, df['impression'].values,␣
 ↪test_size=0.1, random_state=15)
input_train, input_val, output_train, output_val = train_test_split(i_train,␣
 ↪o_train, test_size=0.2, random_state=15)
input_train.shape, output_train.shape, input_val.shape, output_val.shape,␣
 ↪input_test.shape, output_test.shape
```

[17]:

```
((2542, 2), (2542,), (636, 2), (636,), (354, 2), (354,))
```

- Train test and validation split for duplicate dataframe

[18]:

```
i_train_dup, input_test_dup, o_train_dup, output_test_dup =␣
 ↪train_test_split(df_dup[['image_1','image_2']].values, df_dup['impression'].
 ↪values, test_size=0.1, random_state=15)
input_train_dup, input_val_dup, output_train_dup, output_val_dup =␣
 ↪train_test_split(i_train_dup, o_train_dup, test_size=0.2, random_state=15)
input_train_dup.shape, output_train_dup.shape, input_val_dup.shape,␣
 ↪output_val_dup.shape, input_test_dup.shape, output_test_dup.shape
```

[18]:

```
((320, 2), (320,), (81, 2), (81,), (45, 2), (45,))
```

- Append duplicate data equally with train test, and validation dataset

[19]:

```
in_train = np.append(input_train, input_train_dup, axis=0)
out_train = np.append(output_train, output_train_dup, axis=0)
in_val = np.append(input_val, input_val_dup, axis=0)
out_val = np.append(output_val, output_val_dup, axis=0)
in_test = np.append(input_test, input_test_dup, axis=0)
out_test = np.append(output_test, output_test_dup, axis=0)
print("===== Final data point shape =====")
in_train.shape, out_train.shape, in_val.shape, out_val.shape, in_test.shape,␣
 ↪out_test.shape
```

```
===== Final data point shape =====
```

[19]:

```
((2862, 2), (2862,), (717, 2), (717,), (399, 2), (399,))
```

[ ]:

```
in_train[0]
```

[ ]:

```
array(['CXR914_IM-2417-1001.png', 'CXR914_IM-2417-3001.png'], dtype=object)
```

Shuffle the data point

[20]:

```
# Shuffle captions and image_names together
# Set a random state
for i in range(3):
    in_train, out_train = shuffle(in_train, out_train, random_state=15)
    in_val, out_val = shuffle(in_val, out_val, random_state=15)
    in_test, out_test = shuffle(in_test, out_test, random_state=15)
```

5 Text Tokenization

[21]:

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len_output = 60

tokenizer = Tokenizer(oov_token="<unk>", filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~␣
 ↪')
tokenizer.fit_on_texts(out_train)
text_train = tokenizer.texts_to_sequences(out_train)
text_test = tokenizer.texts_to_sequences(out_test)
text_val = tokenizer.texts_to_sequences(out_val)
dictionary = tokenizer.word_index

word2idx = {}
idx2word = {}
for k, v in dictionary.items():
    word2idx[k] = v
    idx2word[v] = k
```

[22]:

```
vocab_size = len(word2idx)+1
vocab_size
```

[22]:

```
1339
```

[ ]:

```
print("===== Top 6 Word and its Index =====")
list(dictionary.items())[:6]
```

```
===== Top 6 Word and its Index =====
```

[ ]:

```
[('<unk>', 1),
 ('<start>', 2),
 ('<end>', 3),
 ('no', 4),
 ('acute', 5),
 ('cardiopulmonary', 6)]
```

[23]:

```
text_output_train = pad_sequences(text_train, maxlen=max_len_output,␣
 ↪dtype='int32', padding='post', truncating='post')
text_output_val = pad_sequences(text_val, maxlen=max_len_output, dtype='int32',␣
 ↪padding='post', truncating='post')
```

```
text_output_test = pad_sequences(text_test, maxlen=max_len_output,␣
 ↪dtype='int32', padding='post', truncating='post')
```

[ ]:

```
text_output_train.shape
```

[ ]:

```
(2862, 60)
```

[24]:

```
def multi_image(img, imp):
    return tf.convert_to_tensor([img_tensor[image_name.index(img[0].
 ↪decode('utf-8'))], img_tensor[image_name.index(img[1].decode('utf-8'))]]),␣
 ↪imp
```

[25]:

```
dataset_train = tf.data.Dataset.from_tensor_slices((in_train,␣
 ↪text_output_train))

# Use map to load the numpy files in parallel
dataset_train = dataset_train.map(lambda item1, item2: tf.numpy_function(
          multi_image, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)

dataset_val = tf.data.Dataset.from_tensor_slices((in_val, text_output_val))

# Use map to load the numpy files in parallel
dataset_val = dataset_val.map(lambda item1, item2: tf.numpy_function(
          multi_image, [item1, item2], [tf.float32, tf.int32]),
          num_parallel_calls=tf.data.experimental.AUTOTUNE)
```

[ ]:

```
for i, j in dataset_train:
  print(i,j)
  break
```

```
tf.Tensor(
[[[0.42520657 0.1407503  0.12797835 … 0.27148038 0.12671113 0.44165108]]

 [[0.47862026 0.37916934 0.49698234 … 0.21267073 0.05756407 0.33507892]]],
shape=(2, 1, 2048), dtype=float32) tf.Tensor(
[ 2 67 28 74 88 22  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0], shape=(60,), dtype=int32)
```

[ ]:

```
for i, j in dataset_val:
  print(i,j)
  break
```

```
tf.Tensor(
[[[0.5016685  0.46150106 0.410387   … 0.20937526 0.31092942 0.96214116]]
```

```
 [[0.50561136 0.44661537 0.4367092  … 0.7984372  0.01216017 1.5053163 ]]],
shape=(2, 1, 2048), dtype=float32) tf.Tensor(
[ 2  4  5  6 38  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0], shape=(60,), dtype=int32)
```

[26]:

```python
BATCH_SIZE = 32
BUFFER_SIZE = 500
embedding_dim = 256
units = 512
```

[27]:

```python
# Shuffle and batch
dataset_train = dataset_train.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_train = dataset_train.prefetch(buffer_size=tf.data.experimental.
 ↪AUTOTUNE)
# Shuffle and batch
dataset_val = dataset_val.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset_val = dataset_val.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

6 Encoder Decoder model

[28]:

```python
class Encoder(tf.keras.Model):
    def __init__(self, embedding_dim):
        super(Encoder, self).__init__()
        self.fc = tf.keras.layers.Dense(embedding_dim, kernel_initializer=tf.
 ↪keras.initializers.glorot_uniform(seed=45),
                  name="encoder_output_layer")

    def call(self, x):
        x = tf.reshape(x, [x.shape[0], x.shape[1], x.shape[3]])
        encoder_concat = tf.keras.layers.concatenate([x[:,0], x[:,1]])
        x = self.fc(encoder_concat)
        x = tf.nn.relu(x)
        return x
```

[29]:

```python
class Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(Decoder, self).__init__()
        self.units = units
        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=tf.keras.
 ↪initializers.glorot_uniform(seed=45))
```

```
        self.dense = tf.keras.layers.Dense(vocab_size, kernel_initializer=tf.
↪keras.initializers.glorot_uniform(seed=45))

    def call(self, x, features):
        #input x = input word teach forcing
        #input features = encoder image features
        x = self.embedding(x)
        x = tf.concat([x, tf.expand_dims(features,1)], axis=-1)
        output, state, _ = self.lstm(x)
        x = self.dense(output)
        return x
```

[30]:

```
optimizer = tf.keras.optimizers.Adam()
loss_obj = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

acc_obj = tf.keras.metrics.SparseCategoricalAccuracy()

def loss_func(real, pred):
    loss_f = loss_obj(real, pred)
    return tf.reduce_mean(loss_f)


def acc_func(real, pred):
    acc_f = acc_obj(real, pred)
    return tf.reduce_mean(acc_f)
```

[31]:

```
import datetime
current_time = datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
train_log_dir = 'logs/gradient_tape/' + current_time + '/train'
val_log_dir = 'logs/gradient_tape/' + current_time + '/test'
train_summary_writer = tf.summary.create_file_writer(train_log_dir)
val_summary_writer = tf.summary.create_file_writer(val_log_dir)
```

[32]:

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

[ ]:

```
!rm -r logs/
```

[33]:

```
@tf.function
def train_step(tensor, target):
    loss = 0
    accuracy = 0
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.
↪shape[0], 1)
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions = decoder(dec_input, features)
            loss += loss_func(target[:, i], predictions)
            accuracy += acc_func(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i],1)
            #print("decoder input teacher", dec_input.shape)
    total_loss = (loss / int(target.shape[1]))
    total_acc = (accuracy / 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, total_acc

#validation function
@tf.function
def val_step(tensor, target):
    loss_val = 0
    accuracy_val = 0
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']] * target.
 ↪shape[0], 1)
    with tf.GradientTape() as tape:
        features = encoder(tensor)
        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions_val = decoder(dec_input, features)
            loss_val += loss_func(target[:, i], predictions_val)
            accuracy_val += acc_func(target[:, i], predictions_val)
            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i],1)
            #print("decoder input teacher", dec_input)
    total_loss_val = (loss_val / int(target.shape[1]))
    total_acc_val = (accuracy_val / int(target.shape[1]))
    return loss_val, total_loss_val, total_acc_val
```

7 Model Training

[35]:

```
tf.keras.backend.clear_session()
EPOCHS = 10
loss_plot_train = []
loss_plot_val = []
for epoch in range(0, EPOCHS):
    print("====== Start Epoch " +str(epoch + 1)+ " ========")

    total_loss_train = 0
    total_acc_train = 0
    total_loss_val = 0
    total_acc_val = 0
    print('Batchwise Train loss')
    for (batch, (jpg_tensor, target)) in enumerate(dataset_train):
        batch_loss, t_loss, t_acc = train_step(jpg_tensor, target)
        total_loss_train += t_loss
        total_acc_train += t_acc

        if batch % 40 == 0:
            print ('Epoch {} Batch {} Loss {:.4f} acc {:.4f}'.format(
              epoch + 1, batch, batch_loss / int(target.shape[1]), t_acc))

    loss_plot_train.append(total_loss_train / int(len(in_train) // BATCH_SIZE))
    with train_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_train/ int(len(in_train) //␣
 ↪BATCH_SIZE), step=epoch)
        tf.summary.scalar('accuracy', total_acc_train/ int(len(in_train) //␣
 ↪BATCH_SIZE), step=epoch)

    print('Batchwise validation loss')
    for (batch, (jpg_tensor, target)) in enumerate(dataset_val):
        batch_loss_val, t_loss_val, t_acc_val = val_step(jpg_tensor, target)
        total_loss_val += t_loss_val
        total_acc_val += t_acc_val
        if batch % 40 == 0:
            print ('Epoch {} Batch {} Loss {:.4f} acc {:.4f}'.format(
              epoch + 1, batch, batch_loss_val / int(target.shape[1]),␣
 ↪t_acc_val))

    with val_summary_writer.as_default():
        tf.summary.scalar('loss', total_loss_val/int(len(in_val) //␣
 ↪BATCH_SIZE), step=epoch)
        tf.summary.scalar('accuracy', total_acc_val/int(len(in_val) //␣
 ↪BATCH_SIZE), step=epoch)
```

```
    template = 'Epoch {}, Loss: {}, Accuracy: {}, Test Loss: {}, Test Accuracy:␣
↪{}'
    print (template.format(epoch+1,
                                total_loss_train/ int(len(in_train) // BATCH_SIZE),
                                (total_acc_train/ int(len(in_train) //␣
↪BATCH_SIZE))*100,
                                total_loss_val/int(len(in_val) // BATCH_SIZE),
                                (total_acc_val/int(len(in_val) // BATCH_SIZE))*100))

    #print ('Epoch {} Train Loss {:.4f} Validation Loss {:.4f}'.format(epoch +␣
↪1,
                                            #(total_loss_train/int(len(in_train) //
↪ BATCH_SIZE)), (total_loss_val/int(len(in_val) // BATCH_SIZE))))
```

```
====== Start Epoch 1 ========
Batchwise Train loss
Epoch 1 Batch 0 Loss 1.1135 acc 0.7976
Epoch 1 Batch 40 Loss 1.2703 acc 0.7975
Epoch 1 Batch 80 Loss 1.3873 acc 0.8014
Batchwise validation loss
Epoch 1 Batch 0 Loss 1.0186 acc 0.8018
Epoch 1, Loss: 1.239553451538086, Accuracy: 80.79592895507812, Test Loss:
1.2054529190063477, Test Accuracy: 83.8067855834961
====== Start Epoch 2 ========
Batchwise Train loss
Epoch 2 Batch 0 Loss 1.4046 acc 0.8015
Epoch 2 Batch 40 Loss 1.1139 acc 0.8012
Epoch 2 Batch 80 Loss 0.8677 acc 0.8029
Batchwise validation loss
Epoch 2 Batch 0 Loss 1.2532 acc 0.8030
Epoch 2, Loss: 1.0313900709152222, Accuracy: 81.08914184570312, Test Loss:
1.0791555643081665, Test Accuracy: 83.95227813720703
====== Start Epoch 3 ========
Batchwise Train loss
Epoch 3 Batch 0 Loss 0.6735 acc 0.8029
Epoch 3 Batch 40 Loss 0.8768 acc 0.8026
Epoch 3 Batch 80 Loss 0.9350 acc 0.8035
Batchwise validation loss
Epoch 3 Batch 0 Loss 0.8808 acc 0.8037
Epoch 3, Loss: 0.9629520177841187, Accuracy: 81.19535064697266, Test Loss:
1.0505082607269287, Test Accuracy: 84.0234375
====== Start Epoch 4 ========
Batchwise Train loss
Epoch 4 Batch 0 Loss 0.9241 acc 0.8035
Epoch 4 Batch 40 Loss 1.2356 acc 0.8019
Epoch 4 Batch 80 Loss 0.6748 acc 0.8001
Batchwise validation loss
```

```
Epoch 4 Batch 0 Loss 1.0439 acc 0.7989
Epoch 4, Loss: 0.9457923769950867, Accuracy: 81.09061431884766, Test Loss:
1.0551683902740479, Test Accuracy: 83.4227523803711
====== Start Epoch 5 ========
Batchwise Train loss
Epoch 5 Batch 0 Loss 1.1116 acc 0.7968
Epoch 5 Batch 40 Loss 1.0570 acc 0.7933
Epoch 5 Batch 80 Loss 0.6418 acc 0.7908
Batchwise validation loss
Epoch 5 Batch 0 Loss 1.0131 acc 0.7902
Epoch 5, Loss: 0.9304273128509521, Accuracy: 80.19268798828125, Test Loss:
1.0312944650650024, Test Accuracy: 82.5200424194336
====== Start Epoch 6 ========
Batchwise Train loss
Epoch 6 Batch 0 Loss 0.9439 acc 0.7886
Epoch 6 Batch 40 Loss 0.9329 acc 0.7857
Epoch 6 Batch 80 Loss 0.5870 acc 0.7846
Batchwise validation loss
Epoch 6 Batch 0 Loss 0.9525 acc 0.7845
Epoch 6, Loss: 0.9189270734786987, Accuracy: 79.48471069335938, Test Loss:
1.0021772384643555, Test Accuracy: 81.97039031982422
====== Start Epoch 7 ========
Batchwise Train loss
Epoch 7 Batch 0 Loss 1.0685 acc 0.7833
Epoch 7 Batch 40 Loss 0.8657 acc 0.7818
Epoch 7 Batch 80 Loss 1.0985 acc 0.7815
Batchwise validation loss
Epoch 7 Batch 0 Loss 0.9967 acc 0.7812
Epoch 7, Loss: 0.9027255773544312, Accuracy: 79.07437896728516, Test Loss:
0.9851518869400024, Test Accuracy: 81.65069580078125
====== Start Epoch 8 ========
Batchwise Train loss
Epoch 8 Batch 0 Loss 0.7119 acc 0.7808
Epoch 8 Batch 40 Loss 0.7705 acc 0.7800
Epoch 8 Batch 80 Loss 1.0963 acc 0.7803
Batchwise validation loss
Epoch 8 Batch 0 Loss 0.8099 acc 0.7803
Epoch 8, Loss: 0.8775861859321594, Accuracy: 78.89749145507812, Test Loss:
0.9408712387084961, Test Accuracy: 81.56993865966797
====== Start Epoch 9 ========
Batchwise Train loss
Epoch 9 Batch 0 Loss 0.9416 acc 0.7802
Epoch 9 Batch 40 Loss 0.6182 acc 0.7797
Epoch 9 Batch 80 Loss 0.8512 acc 0.7799
Batchwise validation loss
Epoch 9 Batch 0 Loss 0.4514 acc 0.7801
Epoch 9, Loss: 0.815396249294281, Accuracy: 78.86933135986328, Test Loss:
0.8767755627632141, Test Accuracy: 81.54573822021484
```

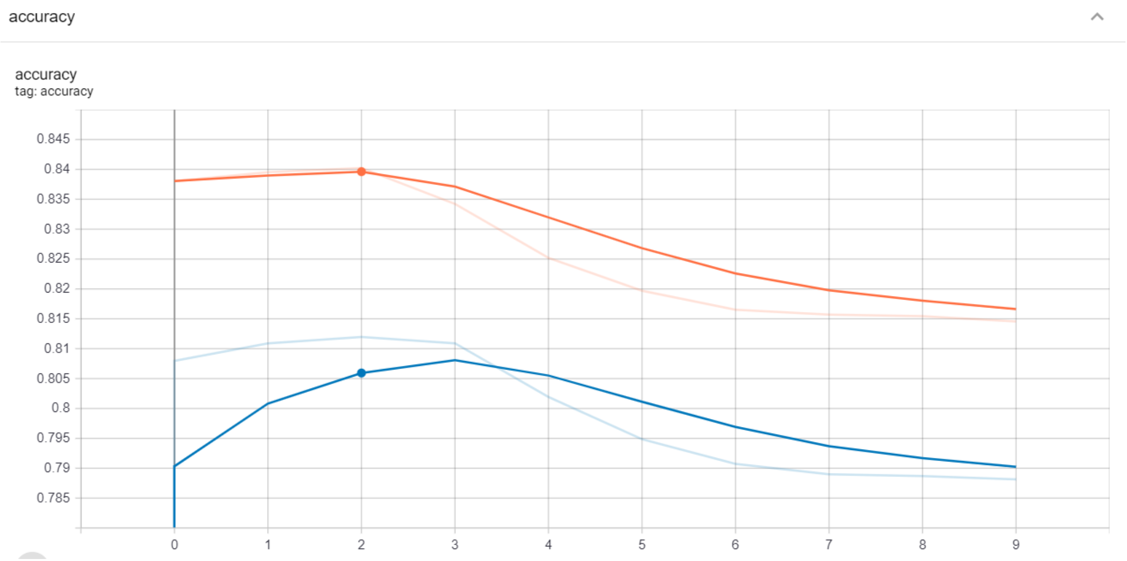
```
====== Start Epoch 10 ========
Batchwise Train loss
Epoch 10 Batch 0 Loss 0.8586 acc 0.7798
Epoch 10 Batch 40 Loss 0.7384 acc 0.7793
Epoch 10 Batch 80 Loss 1.0863 acc 0.7793
Batchwise validation loss
Epoch 10 Batch 0 Loss 0.6645 acc 0.7793
Epoch 10, Loss: 0.7516942620277405, Accuracy: 78.81513214111328, Test Loss:
0.8225345611572266, Test Accuracy: 81.45719909667969
```

[1]:

```python
from IPython.display import Image
print("Accuracy")
Image(filename='basic_acc.png')
```

```
Accuracy
```

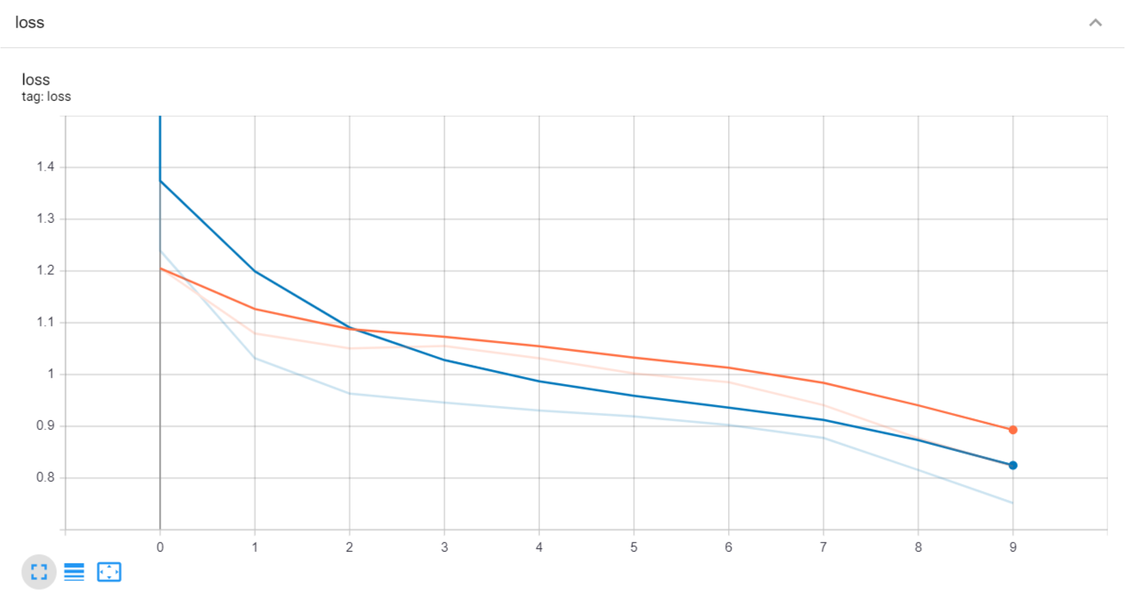
[1]:



[2]:

```python
Image(filename='basic_loss.png')
```

[2]:



[53]:
```
%load_ext tensorboard
```

[54]:
```
tensorboard --logdir=logs/
```

```
<IPython.core.display.Javascript object>
```

8 Model Evaluation

[37]:
```
def get_img_tensor(image_path, img_name, model_image):
    img = tf.io.read_file(image_path + str(img_name))
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (299, 299))
    img = tf.keras.applications.inception_v3.preprocess_input(img)
    img_features = model_image(tf.constant(img)[None, :])
    return img_features
```

[42]:
```
def evaluate(img_name):
    img_tensor = tf.convert_to_tensor([get_img_tensor("img/",img_name[0],␣
 ↪image_features_model),
                                      get_img_tensor("img/",img_name[1],␣
 ↪image_features_model)])
    img_features = tf.constant(img_tensor)[None, :]
    features_val = encoder(img_features)
    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 1)
    result = []
    text = ""
    for i in range(max_len_output):

        predictions = decoder(dec_input, features_val)
        predictions = tf.reshape(predictions, [predictions.shape[0],predictions.
 ↪shape[2]])
        predicted_id = tf.argmax(predictions, axis=1)[0].numpy()
        result.append(tokenizer.index_word[predicted_id])
        text += " " + tokenizer.index_word[predicted_id]
        if tokenizer.index_word[predicted_id] == '<end>':
            return result, text

        dec_input = tf.expand_dims([predicted_id], 1)
    return result, text
```

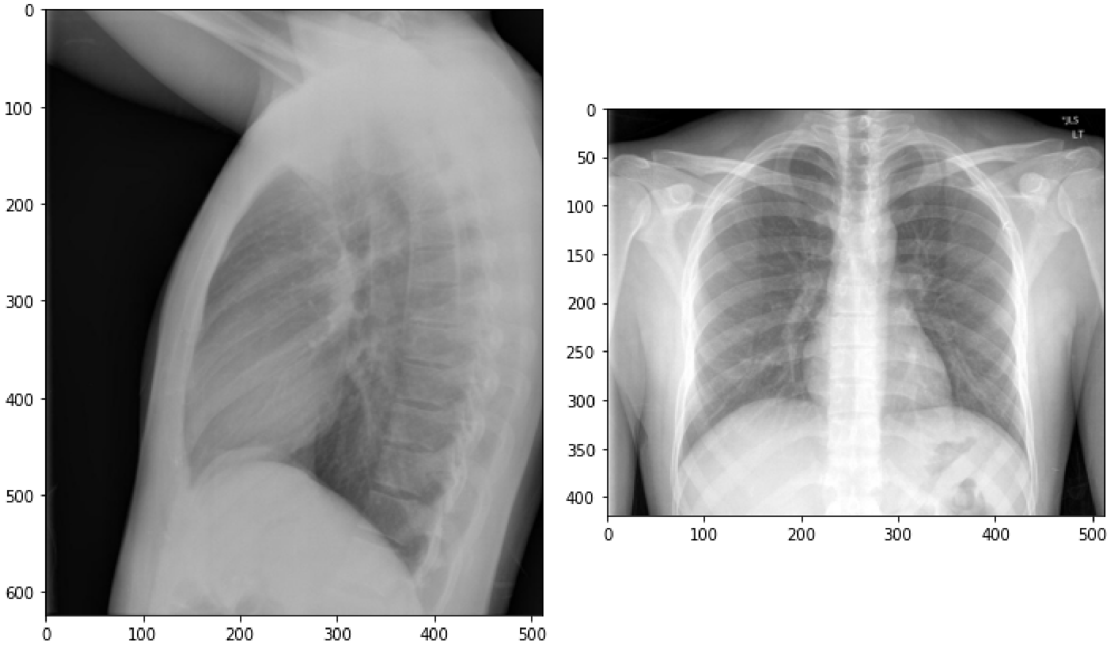
[43]:

```
import matplotlib.image as mpimg
def test_img_cap(img_data):
    result, text = evaluate(img_data)
    """Displays images for given input array of image names"""
    fig, axs = plt.subplots(1, len(img_data), figsize = (10,10),␣
 ↪tight_layout=True)
    count = 0
    for img, subplot in zip(img_data, axs.flatten()):
        img_=mpimg.imread(image_path+img)
        imgplot = axs[count].imshow(img_, cmap = 'bone')
        count +=1
    plt.show()
    print("Predicted:",text)
```

[44]:

```
print("Actual", out_test[164])
test_img_cap(in_test[164])
```

```
Actual <start> no acute cardiopulmonary abnormalities <end>
```

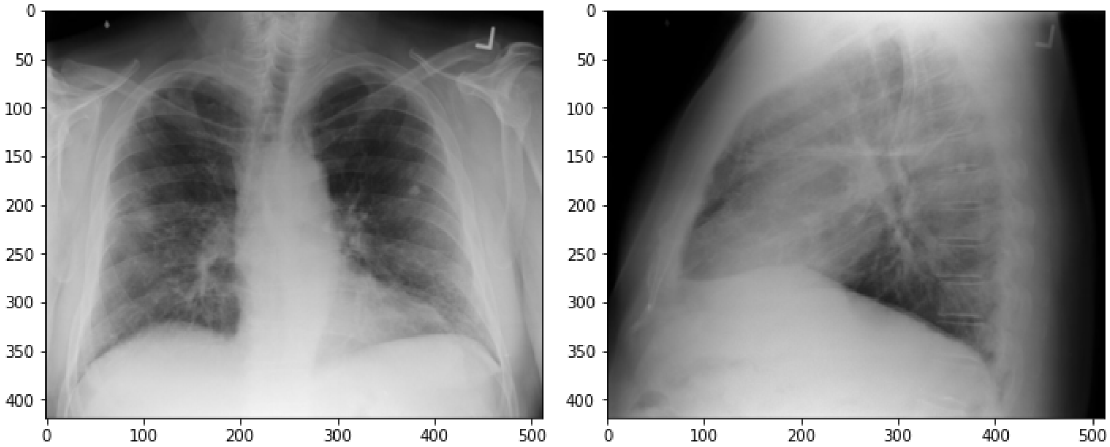


```
Predicted:  no evidence of be focus base opacity <end>
```

[46]:

```
print("Actual", out_test[66])
test_img_cap(in_test[66])
```

```
Actual <start> round density within the anterior segment of the right upper lobe
this may represent pulmonary nodule the primordial was employed to notify the
referring physicians of this critical finding <end>
```

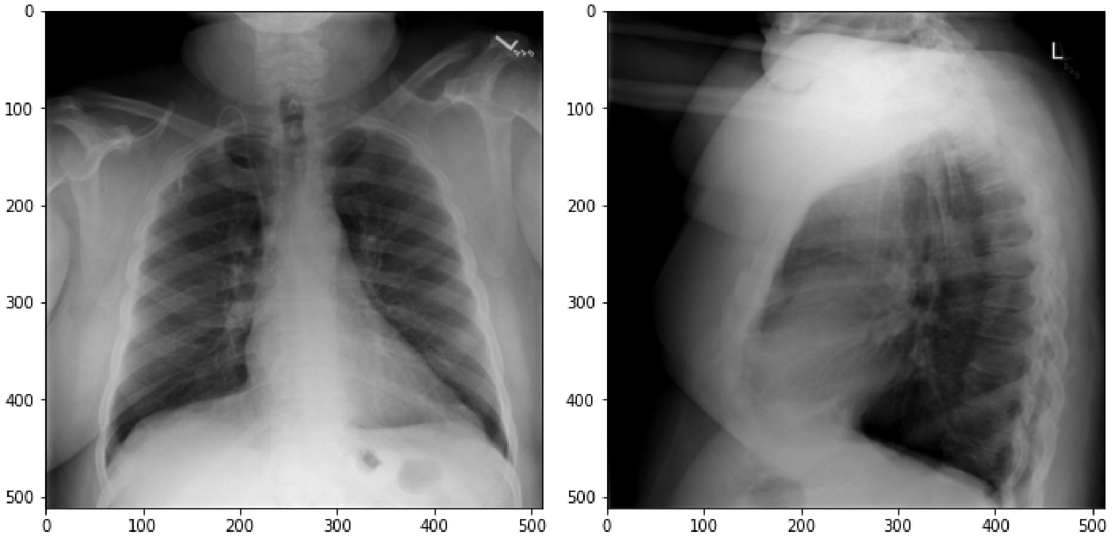


```
Predicted:  negative loculation heart size persistent infiltrate <end>
```

[47]:

```
print("Actual: ", out_test[29])
test_img_cap(in_test[29])
```

```
Actual:  <start> rightsided chest in without demonstration of an acute
cardiopulmonary abnormality <end>
```

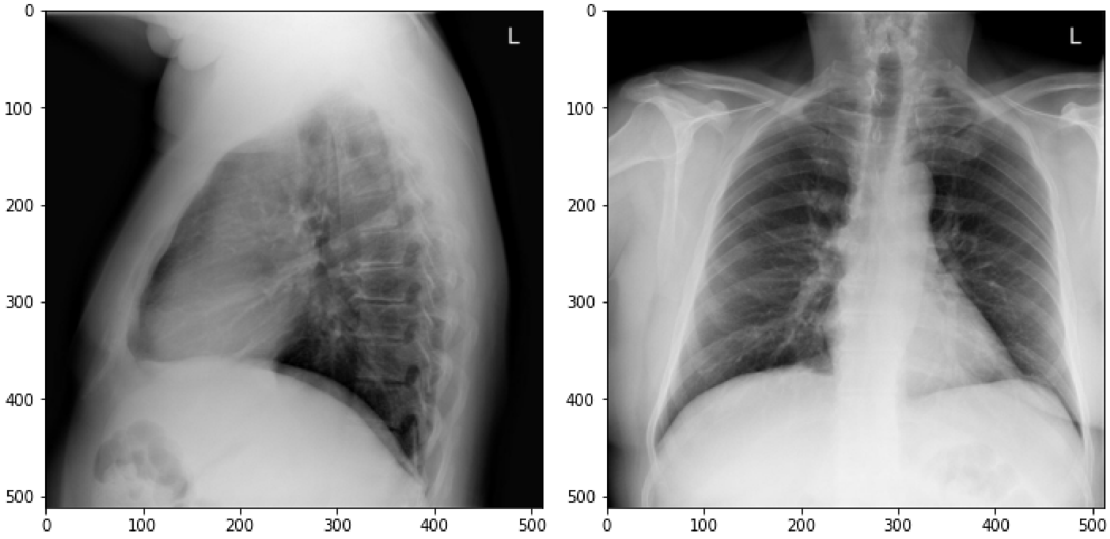


```
Predicted:  no acute findings <end>
```

[48]:

```
print("Actual: ", out_test[229])
test_img_cap(in_test[229])
```

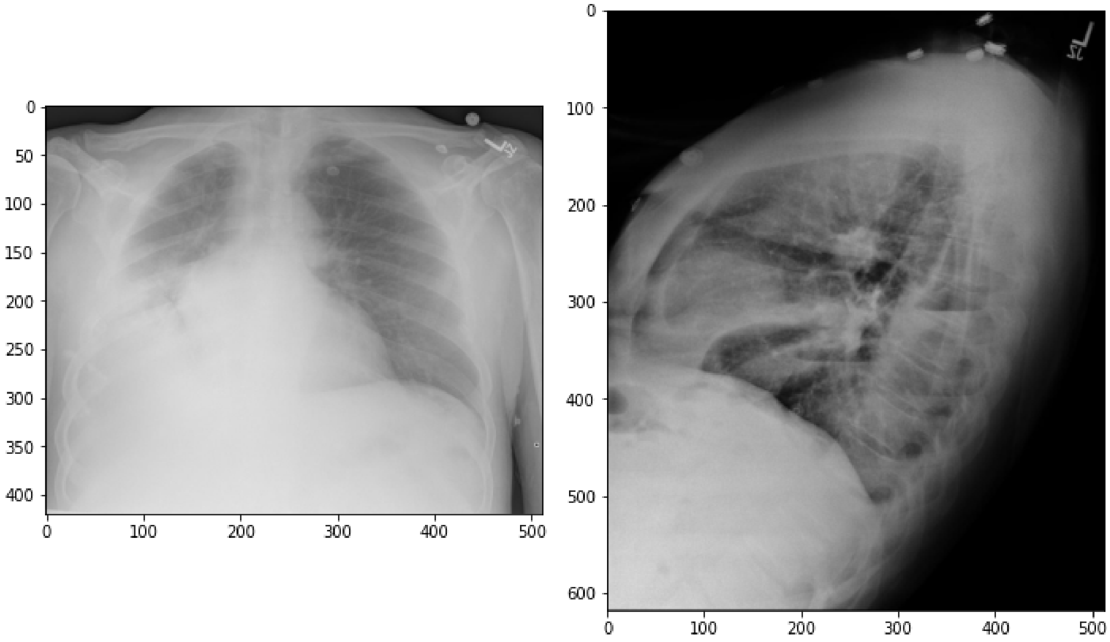
```
Actual:  <start> heart size is normal lungs are clear no nodules or masses no
adenopathy or effusion stable slightly sclerotic posterior inferior of one of
the midthoracic vertebral bodies seen on the lateral radiograph only this most
represents overlying degenerative spurring than metastasis <end>
```



```
Predicted:  no acute cardiopulmonary abnormality was airwaybronchitic are in
arteries bilateral comparison cardiomegaly <end>
```

[50]:
```
print("Actual: ", out_test[366])
test_img_cap(in_test[366])
```

```
Actual:  <start> left lung clear slight cardiomegaly right effusion right lower
lobe infiltrate two airfluid levels in the right hemithorax most representing
hydropneumothorax this radiographic finding could also represent empyema with a
bronchopleural fistula ct scan with iv contrast may be helpful <end>
```

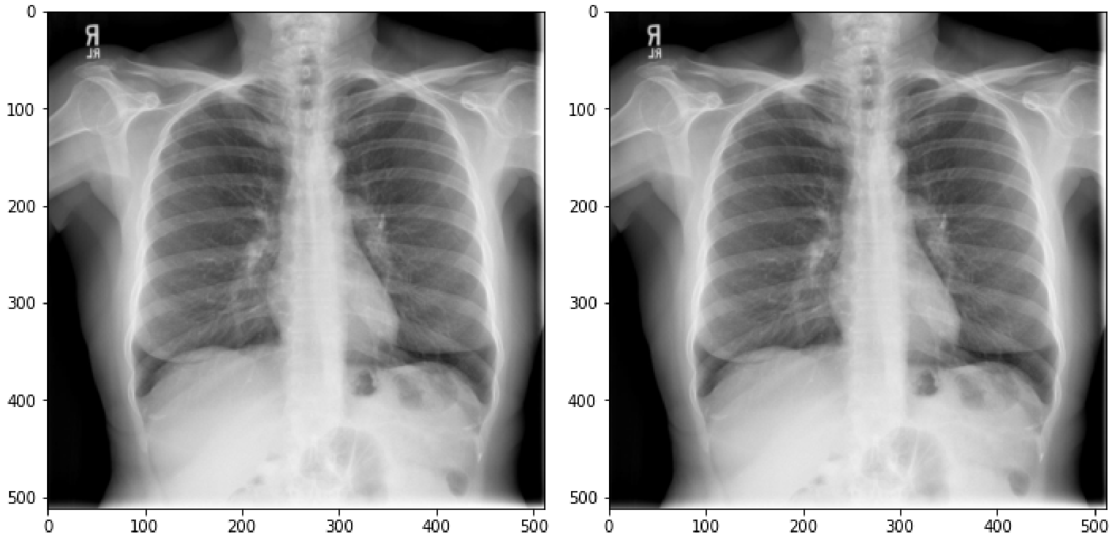


```
Predicted:  no acute cardiopulmonary disease <end>
```

[51]:

```
print("Actual: ", out_test[363])
test_img_cap(in_test[363])
```

```
Actual:  <start> comparison no suspicious appearing lung nodules identified
wellexpanded and clear lungs mediastinal contour within normal limits no acute
cardiopulmonary abnormality identified <end>
```



```
Predicted:  further be indicated wellexpanded normal lungs x pneumonia <end>
```

9 Conclusion

- This model is build on a simple encoder decoder with LSTM.
- getting not perfect or not worst predictions
- validation accuracy is not improving much but loss is converging
- we could even fine tune this model for perform well.