A PROJECT REPORT ON

Multimodal (image and text) representation learning in healthcare to combine text and image data in the medical domain

A project report submitted in fulfillment for the Diploma Degree in AI & ML

Under

Applied Roots with University of Hyderabad

Project submitted by
Santhosh Kurnapally

Under the Guidance of
Mentor: Aniket Vishnu

Approved by:
Mentor: Aniket Vishnu

University of Hyderabad

Declaration of Authorship

We hereby declare that this thesis titled “Multimodal (image and text) representation learning in healthcare to combine text and image data in the medical domain” and the work presented by the undersigned candidate, as part of Diploma Degree in AI & ML.

All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name: Santhosh Kurnapally

Thesis Title: Multimodal (image and text) representation learning in healthcare to combine text and image data in the medical domain

CERTIFICATE OF RECOMMENDATION

We hereby recommend that the thesis entitled “Multimodal (image and text) representation learning in healthcare to combine text and image data in the medical domain” prepared under my supervision and guidance by Santhosh Kurnapally be accepted in fulfilment of the requirement for awarding the degree of Diploma in AI & ML Under applied roots with University of Hyderabad. The project, in our opinion, is worthy for its acceptance.

Mentor: Aniket Vishnu

Under Applied roots with

University of Hyderabad

ACKNOWLEDGEMENT

Every project big or small is successful largely due to the effort of a number of wonderfulpeople who have always given their valuable advice or lent a helping hand. I sincerelyappreciate the inspiration; support and guidance of all those people who have been instrumental in making this project a success. I, Santhosh Kurnapally student of applied roots, is extremely grateful to mentors for the confidence bestowed in me and entrusting my project entitled “Multimodal (image and text) representation learning in healthcare to combine text and image data in the medical domain” with special reference.

At this juncture, I express my sincere thanks to Mentor: Aniket Vishnu of applied roots for making the resources available at right time and providing valuable insights leading to the successful completion of our project who even assisted me in completing the project.

Name: Santhosh Kurnapally

Contents

Abstract:

X-rays, CT scans and MRIs are all diagnostic tools that allow doctors to see the internal structures of the body. They create images using various forms of electromagnetic energy such as radio waves and X-rays. Once these exams are performed, analysis of these X-rays, CT scans or MRIs and preparing radiology reports is an arduous affair for the radiologists. As the number of radiologists are less and these tests became common for patients now a days, report making is delayed which effecting the medical diagnosis of a patient. We are trying to create an image captioning model using Memory Based RNNs like LSTM and GRU used in Encoder – Decoder and Attention strategies which can take the X-Rays of the patient and give us the findings from the X-Rays which reduce the Analysis and Radiology report making efforts to the maximum percentage and speeds up medical diagnosis of a patient. Our main aim was to generate findings from the X-Rays as best as possible and achieve the best possible results.

1. Introduction:

Radiology, also known as diagnostic imaging, is a series of tests that take pictures or images of parts of the body to diagnose and treat diseases. While there are several different imaging exams, some of the most common include X-Ray, MRI, ultrasound, CT scan and PET scan.

A radiologist will look at the outcome of a certain imaging test to find a relevant image that evaluates and supports a diagnosis. After a patient undergoes imaging tests, radiologists will give reports of their interpretations to the referring clinical doctors. A typical radiology reports includes these sections Name or Type of Exam, Date of Exam, MeSH (Medical Subject Headings thesaurus), Interpreting Radiologist Details, Clinical History, Technique, Comparison, Findings, Impression etc.,

X-Rays are a form of Electromagnetic Radiation that is used for medical imaging. Analysis of X-ray reports is a very important task of radiologists and pathologists to recommend the correct diagnosis to the patients. In this Thesis, we are tackling the image captioning problem for a dataset containing Chest X-ray images. With the help of the state of the art deep learning architecture and optimizing parameters of the architecture. The problem statement here is to find the “findings” from the given chest X-Ray images. These images are in two types: Frontal and Lateral view of the chest. With these X-Ray images as input we need to find the “findings” for given X-Ray and report. To resolve this problem statement, we will be building a predictive model which involves both image and text processing to build a deep learning model. Image captioning is an interesting problem, where we can learn both Natural Language Processing(NLP) and Computer Vision(CV) techniques.

Exploratory analysis uses the Python data manipulation and visualization modules NumPy, pandas and matplotlib. Image Captioning Modelling uses the Python machine learning module TensorFlow and Keras. For this problem we are using LSTMs and GRUs in Encoder-Decoder models with Attention Mechanism.

Problem Statement:

In this project we need to generate the “findings” from the X-Ray reports for a given X-Ray. Our model gets trained on X-Ray reports and corresponding Radiology reports of the patients so that it can predict the “findings” when an X-Ray of the new patient is provided to the model.

This an important problem to solve, as per American journal of roentgenology and BMJ: British Medical Journal there are very few radiologists when compared to the population in a particular area, especially in the rural and smaller community settings, hence there are huge delays in the medical image interpretations and cataloguing which delays the medical diagnosis and leaves patient care at risk. Our developed Model Can speed up the medical image interpretation and cataloguing without any intervention of the radiologist and cataloguers address the issues effectively.

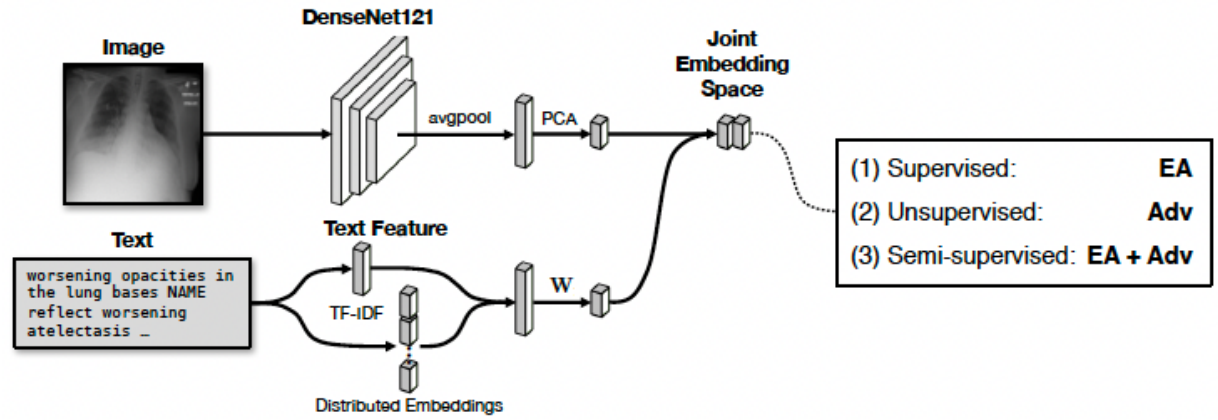
Business/Real-world impact of solving this problem:

Diagnostic radiology is central to disease management in today’s world, with a wide choice of tools and techniques available for the detection, staging and treatment. Diagnostic imaging provides detailed information about structural or disease related changes. Early diagnosis saves lives. Without diagnosis there can be no treatment, there can be no cure. As there are very few radiologists it is very tough and time taking process to analyze the X-Rays and prepare radiology reports manually so generating reports fast will make the medical business standout in the competition and saves lives.

2. Literature Review

Unsupervised Multimodal Representation Learning across Medical Images and Reports [1]

In this paper the authors used Joint embeddings between medical imaging modalities and associated radiology reports and believed that these two modalities have potential to offer significant benefits to the clinical community, ranging from cross-domain retrieval to conditional generation of reports to the broader goals of multimodal representation learning. In this work, authors established baseline joint embedding results measured via both local and global retrieval methods on the MIMIC-CXR dataset consisting of both chest X-ray images and the associated radiology reports by transfer learning from DenseNet121 for X-Rays and Glove model for Text. Authors examined both supervised and unsupervised methods on this task and showed that for document retrieval tasks with the learned representations, only a limited amount of supervision is needed to yield results comparable to those of fully-supervised methods.



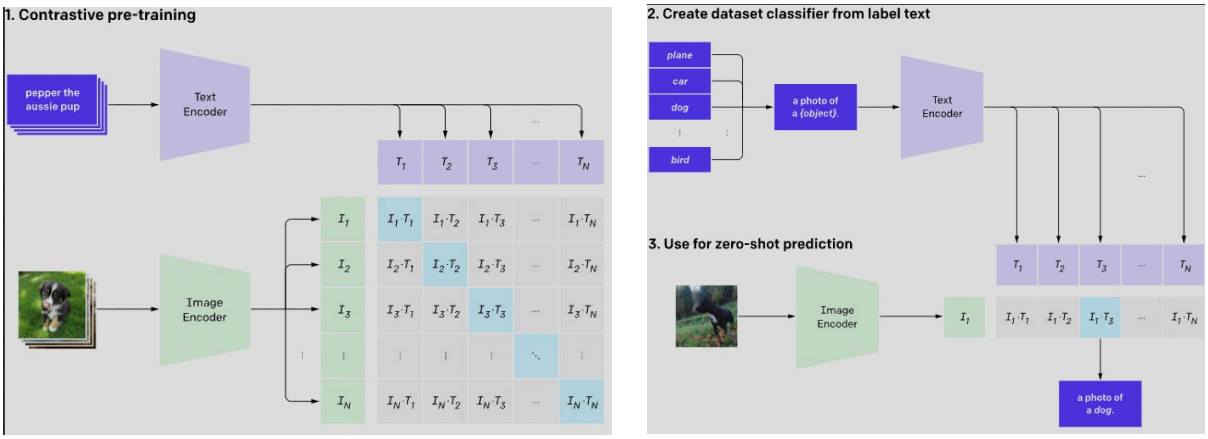
Contrastive Learning of Medical Visual Representations from Paired Images And Text [2]

In this paper authors presented Contrastive VIsual Representation Learning from Text (ConVIRT), a framework for learning visual representations by exploiting the naturally occurring pairing of images and textual data. ConVIRT improves visual representations by maximizing the agreement between true image-text pairs versus random pairs via a bidirectional contrastive objective between the image and text modalities. ConVIRT has the advantages of utilizing the paired text data in

a way agnostic to the medical specialty and requiring no additional expert input. Authors used Random Init, ImageNet Init, Caption-LSTM, Caption-Transformer, Contrastive-Binary to compare with conVIRT and showed that conVIRT outperformed all these models in both image-image retrieval and text-image retrieval.

Learning Transferable Visual Models from Natural Language supervision, CLIP (Contrastive Language-Image Pre-training) [3]

CLIP (*Contrastive Language–Image Pre-training*) builds on a large body of work on zero-shot transfer, natural language supervision, and multimodal learning. Authors in this paper say that scaling a simple pre-training task is sufficient to achieve competitive zero-shot performance on a great variety of image classification datasets. CLIP pre-trains an image encoder and a text encoder to predict which images were paired with which texts in the dataset. Then use this behavior to turn CLIP into a zero-shot classifier. CLIP convert all of a dataset's classes into captions such as "a photo of a *dog*" and predict the class of the caption, CLIP estimates best pairs with a given image.



3. Data Description:

Data Acquisition:

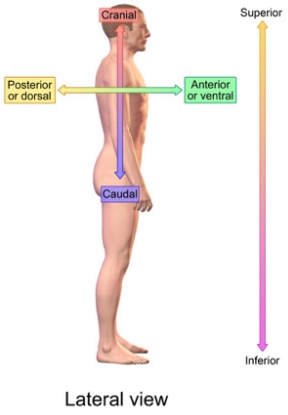
Open-Source data - The dataset for this project is retrieved from Indiana University hospital network.
Source: Indiana University Hospital Network

Image Data:

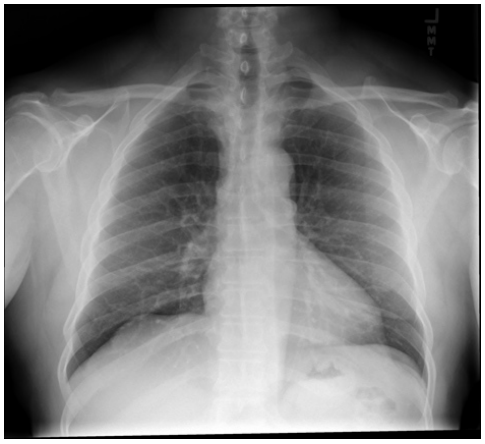
Indiana University - Chest X-Rays (PNG Images):
https://academictorrents.com/details/5a3a439df24931f410fac269b87b050203d9467d

- Data Size: 1.36 GB
- Number of Images: 7470
- All Images are in .png format.
- No challenges in processing these images, used OpenCV to work with images.
- All the images have same width of 512 px.
- But height is varying from 362 p to 873 px.
- Image Types available: Frontal and Lateral

FRONTAL X-RAY **LATERAL X-RAY**



Lateral view



XML Report Data:

Indiana University - Chest X-Rays (XML Reports):
https://academictorrents.com/details/66450ba52ba3f83fbf82ef9c91f2bde0e845aba9

- Data Size: 20.7 MB
- Total Number of reports: 3955
- No challenges in processing the data, used xml.etree.ElementTree to parse XML reports
- Xml Contains following important data and it need to be extracted from XML.

a. **Indication**:

This data describes a simple, concise statement of the reason for the study and/or applicable clinical information or diagnosis. A clear understanding of the indication may also clarify appropriate clinical questions that should be addressed by the study.

Sample Data: Positive TB test, Chest pain etc.,

b. **Comparison**:

This data describes whether this new imaging exam is compared with any available previous exams. Comparisons usually involve exams of the same body area and exam type.

Sample Data: Portable chest dated XXXX, PA and lateral chest redressed XXXX etc.,

c. **Findings**:

This data lists what the radiologist saw in each area of the body in the exam. This have notes whether the area thought to be normal, abnormal, or potentially abnormal.

Sample Data: The heart is normal in size. The mediastinum is unremarkable. The lungs are clear etc.,

d. **Impression**:

This data contains the summary of the findings and reports the most important findings that they see and possible causes for those findings. This section offers the most important information for decision-making.

Sample Data: No acute disease, Clear lungs Etc.,

Sample Report:

Indiana University Chest X-ray Collection

Kohli MD, Rosenman M - (2013)

Affiliation: Indiana University

ABSTRACT

Comparison: None.

Indication: Positive TB test

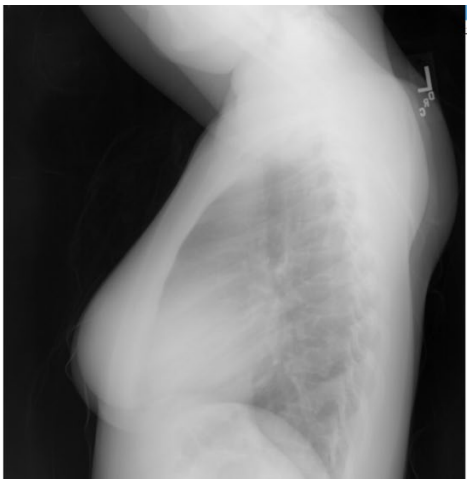
Findings: The cardiac silhouette and mediastinum size are within normal limits. There is no pulmonary edema. There is no focal consolidation. There are no XXXX of a pleural effusion. There is no evidence of pneumothorax.

Impression: Normal chest x-XXXX.

NOTE: The data are drawn from multiple hospital systems.

Show MeSH

Related in: MedlinePlus Request Collection



4. Approach, Methodology and Results

Step 1: Selecting the KPI metric

BLEU: Bilingual Evaluation Understudy

BLEU is an algorithm for evaluating the quality of text which has been machine-translated. The closer a machine translation is to a professional human translation, the better it is and this is the central idea behind BLEU.BLEU was one of the first metrics to claim a high correlation with human judgements of quality, and remains one of the most popular automated and inexpensive metrics.

BLEU's output is always a number between 0 and 1. This value indicates how similar the

candidate text is to the reference texts, with values closer to 1 representing more similar texts. We used BLEU is based on modified n-grams precision as it uses n-grams to compare and rate the quality of the generated text and give the score, it is fast and simple to calculate and widely used.

The way BLEU works is simple. Given some candidate translation of a sentence and a group of reference sentences, we use a bag-of-word approach to see how many occurrences of BOWs co-occur in both the translation and reference sentences. BOW is a simple yet highly effective way of ensuring that the machine translation contains key phrases or words that reference translations also contain. In other words, BLEU compares candidate translations with human-produced, annotated reference translations and compares how many hits there are in the candidate sentence. The more BOW hits there are, the better the translation.

Before understanding BLEU, we need to understand precision, Modified Precision and Brevity Penalty.

Precision:

$$\text{Precision} = \frac{tp}{tp + fp}$$

- where tp and fp stand for true and false positives, respectively.
- we can consider positives as roughly corresponding to the notion of hits or matches.
- In other words, the positives are the bag of word n-grams we can construct from a given candidate translation.
- True positives are n-grams that appear in both the candidate and some reference translation.
- false positives are those that only appear in the candidate translation.

Modified Precision:

Simple precision-based metric has huge problems like if we have a sample candidate as "It it it it it it it it it it it it" the simple precision gives 1 as output but given candidate is very bad. This is because precision simply involves checking whether a hit occurs or not: it does not check for repeated bag of words. Hence modified precision is implemented and in modified precision we will clip the n-gram count if those are repeated multiple times. And formula is as follows

$$\text{Count} = \min(m_w, m_{\max})$$

- Count refers to the number of hits we assign to a certain n-gram.
- m_w refers to the number of occurrences of a n-gram in the candidate sentence.
- m_{max}, which is the maximum number of occurrences of that n-gram in any one of the reference sentences.

Brevity Penalty:

Brevity penalty penalises short candidate translations thus ensuring that only sufficiently long machine translations are ascribed a high score. The goal is to find the length of the reference sentence whose length is closest to that of the candidate translation in question. If the length of that reference sentence is larger than the candidate sentence, we apply some penalty; if the candidate sentence is longer, than we do not apply any penalization. The specific formula for penalization looks as follows:

$$BP = \begin{cases} 1 & l_{ca} > l_{ref} \\ \exp\left(1 - \frac{l_{ref}}{l_{ca}}\right) & l_{ca} \leq l_{ref} \end{cases}$$

BLEU:

The formula for BLEU is as follows

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

- n specifies the size of the bag of word, or the n-gram
- w_n denotes the weight we will ascribe to the modified precision p_n
- p_n modified precision produced under that n-gram configuration.

Code Implementation of BLEU:

```
# cumulative BLEU scores
from nltk.translate.bleu_score import sentence_bleu
reference = [['this', 'is', 'small', 'test']]
candidate = ['this', 'is', 'a', 'test']
print('Cumulative 1-gram: %f' % sentence_bleu(reference, candidate, weights=(1, 0, 0, 0)))
print('Cumulative 2-gram: %f' % sentence_bleu(reference, candidate, weights=(0.5, 0.5, 0, 0)))
print('Cumulative 3-gram: %f' % sentence_bleu(reference, candidate, weights=(0.33, 0.33, 0.33, 0)))
print('Cumulative 4-gram: %f' % sentence_bleu(reference, candidate, weights=(0.25, 0.25, 0.25, 0.25)))
```

```
Cumulative 1-gram: 0.750000
Cumulative 2-gram: 0.500000
Cumulative 3-gram: 0.000000
Cumulative 4-gram: 0.000000
```

PROs of BLEU:

- Uses n-grams and gives the qualitative comparison of two texts.
- Fast and simple to calculate
- Widely used

CONs of BLEU:

- Doesn't consider meaning
- Struggles with non-English languages
- Hard to compare scores with different tokenizers

We can use common metrics like accuracy and F1 score but they do not measure the quality the text for comparison BLEU does that using modified n-grams precision approach. Code for BLEU is implemented from scratch and is attached with Phase1 Document with file name "BLEU Implementation.ipynb".

Step 2: EDA (Exploratory Data Analysis)

I. Requirement Analysis

Functional Requirements:

Purpose: To Generate "Findings" from the X-Ray of the patients for report making.

Inputs: X-Rays and the Radiology Reports of the patients.

Output: Model Generates the Findings from the X-Ray Reports of the patients.

Hardware Requirements

- Computer system
- Google colab pro subscription
- High End GPU with good VRAM
- High End CPU with High RAM
- Stream lit server for deployment
- Internet connectivity

Software Requirements

- Notepad++/BBEdit or any Code Editor
- Anaconda Navigator, Jupyter Notebook
- Streamlit
- MS-Excel, MS-Word
- Python 3
- Python libraries like pandas, NumPy, Nltk, scikit-learn, pickle, os, tqdm, matplotlib, seaborn, xml,PIL,collections,regex,wordcloud, joblib, tensorflow, keras, cv2, streamlit etc.,

- Client environment may be Windows, Mac or Linux
- Web Browser

II. Information Extraction and Analysis

a. Extracting Information from XML:

Using XML library, we are extracting "Findings", Image path and patient id Information from each patient XML reports and forming a dataset with them.

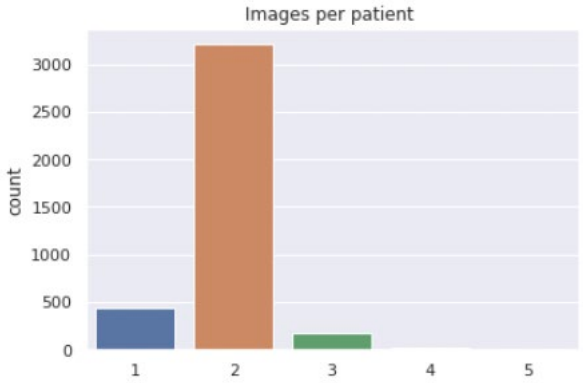
Code Sample:

```
images = []
patient_ids = []
img_findings = []
for filename in tqdm(os.listdir(os.getcwd()+'/reports/ecgen-radiology')):
    if filename.endswith(".xml"):
        f = os.path.join(os.getcwd()+'/reports/ecgen-radiology',filename)
        tree = ET.parse(f)
        root = tree.getroot()
        for child in root:
            if child.tag == 'uId':
              patient = child.attrib['id']
            if child.tag == 'MedlineCitation':
                for attr in child:
                    if attr.tag == 'Article':
                        for i in attr:
                            if i.tag == 'Abstract':
                                for name in i:
                                    if name.get('Label') == 'FINDINGS':
                                        findings=name.text
        for p_image in root.findall('parentImage'):
            patient_ids.append(patient)
            images.append(p_image.get('id'))
            img_findings.append(findings)
```

```
100%|██████████| 3955/3955 [00:01<00:00, 2359.22it/s]
```

b. Understanding Number of Images per patient:

- In total there are 3851 unique patients
    - Patients with 1 image: 446
    - Patients with 2 images: 3208
    - Patients with 3 images: 181
    - Patients with 4 images: 15
    - Patients with 5 images: 1



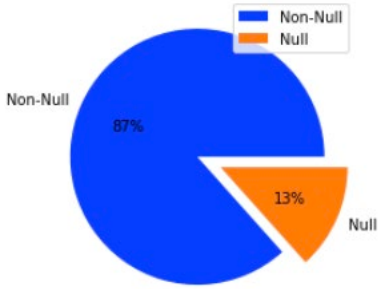
- So, to capture most of the information we give input of two images to the model. as below
    - If the patient has one X-ray associated with report, we replicate same image twice

as image1 and image2.

- If the patient has two X-rays associated with the report, we replicate first image as image1 and second as image2.
- If the patient has more than two X-rays associated with the report, we randomly chose 2 X-rays as image1 and image2.

III. Data Preprocessing for Findings:

a. Null value analysis:



- There are around 13 % null values(None) in findings column.
- Will drop the rows with null value in findings column as we cannot fill the null values with some random findings.

b. Converting to lowercase and expanding contractions:

- Convert findings into lower case and expand few words like won't to will not, can't to cannot etc.

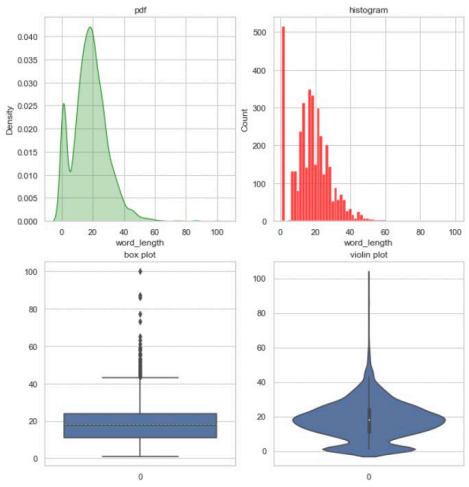
c. Removing Numbers and junk words like 'xxxx' and 'xxx'

- We have few numbers in the findings and junk words like XXXX etc are removed from the findings.

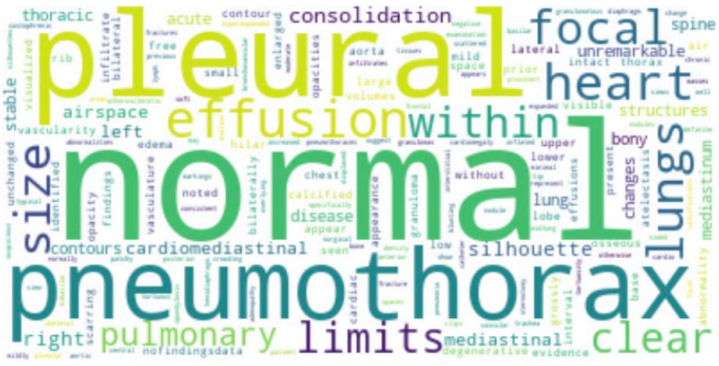
d. Understanding the words statistics of findings

Before preprocessing:

Pdf, Box Violin Plot



Word Cloud



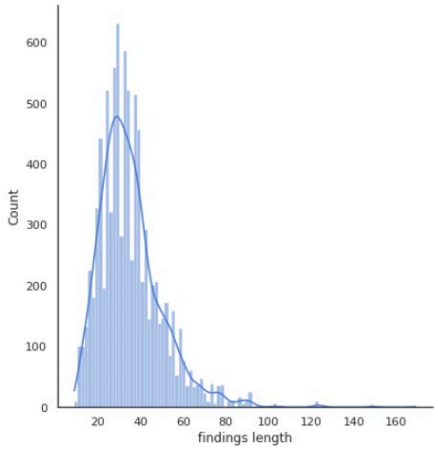
- Here we can see that there is a spike at 0 indicating the null values and word cloud also shows many junk values and we can see there many outliers in box plot aswell. Lets see how this statistics looks like after preprocessing.

After Preprocessing:

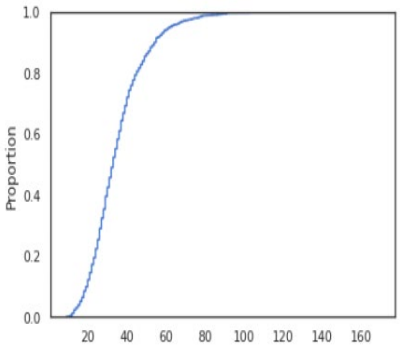
Metrics

	findings
count	3337.000000
mean	30.830087
std	14.078474
min	7.000000
25%	21.000000
50%	29.000000
75%	37.000000
95%	57.000000
99%	76.000000
max	167.000000

PDF

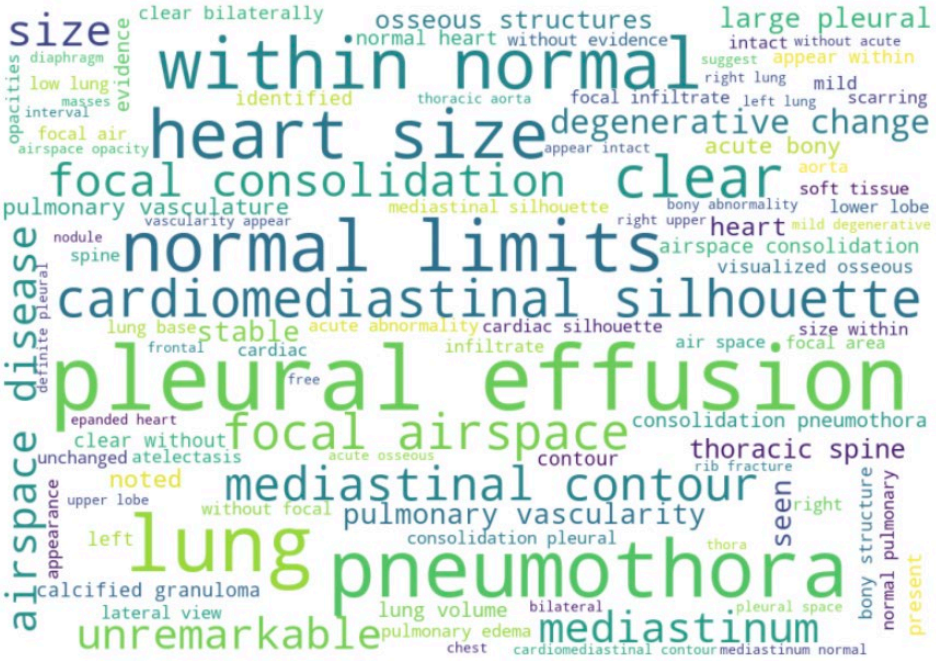


CDF



- The maximum length of Findings is 167 words
- The minimum length of Findings is 7 words
- mean Findings are 30 words
- 75th percentile is 37 words
- 99th percentile is 76

Word Cloud



- We can see that word like pleural effusion, pneumothora, cardiomediastnal silhouette, normal limits are mostly occurring and we can observe that these words are not normal words but these are special to medical domain. And looks very clean after preprocessing.

IV. Train Test Split and Tokenization

a. Train Test Split:

Data biased towards non-disease observations

```
image_findings_dataset['findings'].value_counts()

the heart is normal in size. the mediastinum is unremarkable. the lungs are clear.
51
the heart and lungs have in the interval. both lungs are clear and epanded. heart and mediastinum normal.
51
heart size normal. lungs are clear. are normal. no pneumonia, effusions, edema, pneumothora, adenopathy, nodules or masses.
46
the lungs are clear bilaterally. specifically, no evidence of focal consolidation, pneumothora, or pleural effusion. cardio medias
of the thora are without acute abnormality.
cardiac and mediastinal contours are within normal limits. the lungs are clear. bony structures are intact.
35

..
the cardiac silhouette mediastinal contours are within normal limits. the lungs are clear bilaterally. no focal opacities. there i
deformities involving multiple vertebral bodies of the thoracic spine which appear stable compared to the previous eam.
the lungs are clear bilaterally. specifically, no evidence of focal consolidation, pneumothora, or pleural effusion. cardio medias
of the thora demonstrate stable, mild multilevel thoracolumbar degenerative disc disease without acute abnormality. upper abdomina
there is a large airspace opacity in the right lower and middle lobes. there is no pneumothora. heart size is normal. soft tissue
1
there is a prominent calcified head to the right anterior first rib. the aorta is tortuous. there are t-spine osteophytes. the car
normal limits. there is no pneumothora or pleural effusion. there are no focal areas of consolidation.
normal cardiac contour. stable calcified granuloma left upper lobe. no pleural effusion or pneumothora. clear lungs bilaterally.
1
```

- If we carefully observe the findings column, we can see that data in findings column is biased towards the non-disease data and as we have very data which is around 3300 records this is not at all enough for using in deep learning approach but we will try to resample data and make data balanced by below approach (we have tried multiple approaches and fixed on below values)
- as we can see that data is repeated a lot in the findings columns lets employ a strategy to train a better model.

Step1: let's divide the dataset into two parts

- one with findings greater than 25 occurrences.
- second with findings less than or equal to 5 occurrences.

Step2: let's divide the train test sets with test_size = 0.1 for findings greater than 5.

Step3: let's divide the train test sets with 20 % sample size for findings less than or equal to 5.and add that sample test and remaining to train

Step4: Up sample the minority points and down sample the majority points

- by doing this we are reducing the imbalance in the dataset with respect to findings

Code implementation

```
findings_gt_5 = image_findings_dataset[image_findings_dataset['findings_count']>5]
findings_lte_5 = image_findings_dataset[image_findings_dataset['findings_count']<=5]
train,test = train_test_split(findings_gt_5,stratify = findings_gt_5['findings'].values,test_size = 0.1,random_state = 420)
test_findings_lte_5_sample = findings_lte_5.sample(int(0.2*findings_lte_5.shape[0]),random_state = 420)
findings_lte_5 = findings_lte_5.drop(test_findings_lte_5_sample.index,axis=0)
test = test.append(test_findings_lte_5_sample)
test = test.reset_index(drop=True)
train = train.append(findings_lte_5)
train = train.reset_index(drop=True)
train.shape[0],test.shape[0]
```

- and after up sampling and down sampling of minority and majority points respectively we ended up having 8795 records in train and 604 records in test data set, this process was mandatory as we do not have data and whatever data we had is biased to non-disease case.

b. Creating Word Tokenizer:

```
#tokenizer
tokenizer = Tokenizer(filters = '',oov_token = '<unk>') #setting filters to none
tokenizer.fit_on_texts(train.findings_total.values)
train_captions = tokenizer.texts_to_sequences(train.findings_total)
test_captions = tokenizer.texts_to_sequences(test.findings_total)
vocab_size = len(tokenizer.word_index)
caption_len = np.array([len(i) for i in train_captions])
start_index = tokenizer.word_index['<start>'] #tokened value of <start>
end_index = tokenizer.word_index['<end>'] #tokened value of <end>
```

- We create a tokenizer using tensorflow text tokenizer.
- We will save train and test dataset, tokenizer using pickle for further use.

Step3: Building simple Encoder-Decoder model

I. Understanding Prerequisites

a. CheXNet:

ChexNet is a deep learning algorithm that can detect and localize 14 kinds of diseases from chest X-ray images. A 121-layer densely connected convolutional neural network is trained on ChestX-ray14 dataset, which contains 112,120 frontal view X-ray images from 30,805 unique patients. The result is so good that it surpasses the performance of practicing radiologists.

We used ChexNet pre-trained weights to obtain the embeddings for the X-Rays using transfer learning. As ChexNet weights are well converged in the tasks like disease classification on ChestX-ray14 dataset.

Resource: https://arxiv.org/pdf/1711.05225v3.pdf

Weights file: https://www.kaggle.com/datasets/theewok/chexnet-keras-weights

ChexNet Using Similar kind of Architecture as its backbone is DenseNet121, below is the DenseNet architecture.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6	[1 × 1 conv, 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12	[1 × 1 conv, 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv, 3 × 3 conv] × 24	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48	[1 × 1 conv, 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv, 3 × 3 conv] × 16	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 32	[1 × 1 conv, 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

b. GloVe:

GloVe stands for Global Vectors. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

GloVe is essentially a log-bilinear model with a weighted least-squares objective. The main intuition underlying the model is the simple observation that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning.

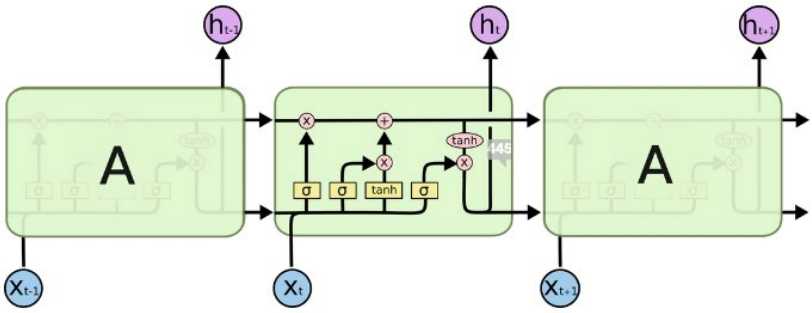
We used pre-trained word vectors for converting words to embeddings and GloVe provide Multidimensional re-trained words vectors and out of them we used 300-dimensional word vectors for word embedding conversion.

Source: https://nlp.stanford.edu/projects/glove/

Glove300d.zip: https://nlp.stanford.edu/data/glove.6B.zip

c. LSTM:

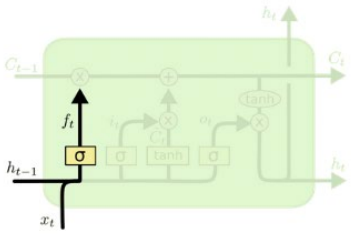
LSTM stands for Long Short-Term Memory RNN. The simple RNN cannot handle or work well for long term dependencies. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior.



LSTMs have three inputs and two outputs. The LSTM does have the ability to remove or add information to the cell state and is also possible to pass information unchanged. LSTMs have three gates and are as below

1. Forget Gate:

The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer."

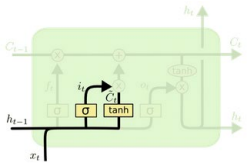


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

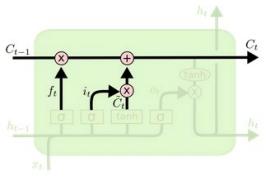
2. Input Gate:

This next layer is called the “input gate layer” as it will decide what new information, we’re going to store in the cell state. This has two parts. First, a sigmoid layer decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values. We will do element-wise multiplication to output from these two parts.

While calculating present cell state value we consider values from forget gate and input gate. Forget gates can forget or retain information from the previous cell state to the next cell state and the input gate will decide what new information we’re going to store in the cell state based on present state inputs.



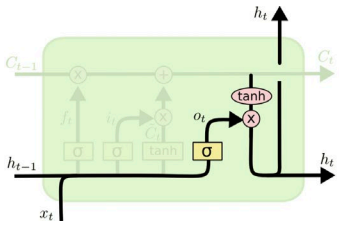
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

3. Output Gate:

The final layer in LSTM is called the “output gate layer”. This output will be based on our Current cell state, Previous output and input (Xt). We will send the current cell state through Tanh to push the values to be between −1 and 1 and multiply it by the output of the sigmoid gate.



$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

We used LSTM to encode and decode as image and text are kind of sequence information’s to start with and RNN and types of RNNs are the best fit for this type of sequence data.

II. Encoder-Decoder Model Building

a. Creating Input data Pipeline:

We are creating a data pipeline which will perform tasks on Image and Text and make them ready to be consumed by the model.

- Resizes image to 255x255 pixels.

```
def __getitem__(self,i):
  #gets the datapoint at i th index, we will extract the feature vectors of images after resizing the image  and apply augmentation
  image1 = cv2.imread(self.image1[i],cv2.IMREAD_UNCHANGED)/255
  image2 = cv2.imread(self.image2[i],cv2.IMREAD_UNCHANGED)/255 #here there are 3 channels
  image1 = cv2.resize(image1,self.input_size,interpolation = cv2.INTER_NEAREST)
  image2 = cv2.resize(image2,self.input_size,interpolation = cv2.INTER_NEAREST)
  if image1.any()==None:
    print("%i , %s image sent null value"%(i,self.image1[i]))
  if image2.any()==None:
    print("%i , %s image sent null value"%(i,self.image2[i]))
```

- Tokenizes the findings and pads all findings to the same length for all the findings and padding is done in the end.

```
#tokenizing and padding
caption = self.tokenizer.texts_to_sequences(self.caption[i:i+1]) #the input should be an array for tokenizer ie [self.caption[i]]
caption = pad_sequences(caption,maxlen = self.max_pad,padding = 'post') #opshape:(input_length,)
caption = tf.squeeze(caption,axis=0) #opshape = (input_length,) removing unwanted axis if present
caption1 = self.tokenizer.texts_to_sequences(self.caption1[i:i+1]) #the input should be an array for tokenizer ie [self.caption[i]]
caption1 = pad_sequences(caption1,maxlen = self.max_pad,padding = 'post') #opshape: (input_length,)
caption1 = tf.squeeze(caption1,axis=0) #opshape = (input_length,) removing unwanted axis if present
```

- Image augmentation is applied and techniques we are applying are flipping images horizontally and vertically with uniform probability. If probability less than 33% then flip horizontally if between 33 and 66% flip vertically else no image augmentation.

```
#image augmentation
#https://imgaug.readthedocs.io/en/latest/source/overview/flip.html?highlight=Fliplr
self.aug1 = iaa.Fliplr(1) #flip images horizaontally
self.aug2 = iaa.Flipud(1) #flip images vertically
```

```
if self.augmentation: #we will not apply augmentation that crops the image
      a = np.random.uniform()
      if a<0.333:
          image1 = self.aug1.augment_image(image1)
          image2 = self.aug1.augment_image(image2)
      elif a<0.667:
          image1 = self.aug2.augment_image(image1)
          image2 = self.aug2.augment_image(image2)
      else: #applying no augmentation
        pass;
return image1,image2,caption,caption1
```

- Implemented Data loader for batching and in-place shuffling.

```
class Dataloader(tf.keras.utils.Sequence):     #for batching
    def __init__(self, dataset, batch_size=1, shuffle=True):
        self.dataset = dataset
        self.batch_size = batch_size
        self.shuffle = shuffle
        self.indexes = np.arange(len(self.dataset))

    def __getitem__(self, i):
        # collect batch data
        start = i * self.batch_size
        stop = (i + 1) * self.batch_size
        indexes = [self.indexes[j] for j in range(start,stop)] #getting the shuffled index values
        data = [self.dataset[j] for j in indexes] #taken from Data class (calls __getitem__ of Data) here the shape is batch_size*3, (image_1,image_2,caption)

        batch = [np.stack(samples, axis=0) for samples in zip(*data)] #here the shape will become batch_size*input_size(of image)*3,batch_size*input_size(of image)*3
                                                                      #,batch_size*1*max_pad
        return tuple([[batch[0],batch[1],batch[2]],batch[3]]) #here [image1,image2, caption(without <END>)],caption(without <start>) (op)

    def __len__(self): #returns total number of batches in an epoch
        return len(self.indexes) // self.batch_size

    def on_epoch_end(self): #it runs at the end of epoch
        if self.shuffle:
            np.random.shuffle(self.indexes) #in-place shuffling takes place
```

b. ChexNet for Transfer Learning:

```
#chexnet weights ; https://drive.google.com/file/d/19BllaOvs2x5PLV_vlWMy4i8LapLb2j6b/view
def create_chexnet(chexnet_weights = chexnet_weights):
  """
  chexnet_weights: weights value in .h5 format of chexnet
  creates a chexnet model with preloaded weights present in chexnet_weights file
  """
  model = tf.keras.applications.DenseNet121(include_top=False) #importing densenet the last layer will be a relu activation layer

  #we need to load the weights so setting the architecture of the model as same as the one of tha chexnet
  x = model.output #output from chexnet
  x = GlobalAveragePooling2D()(x)
  x = Dense(14, activation="sigmoid", name="chexnet_output")(x) #here activation is sigmoid as seen in research paper

  chexnet = tf.keras.Model(inputs = model.input,outputs = x)
  chexnet.load_weights(chexnet_weights)
  chexnet = tf.keras.Model(inputs = model.input,outputs = chexnet.layers[-2].output)  #we will be taking the penultimate layer (second last layer here it is global avgpooling)
  return chexnet
```

Implemented ChexNet by loading weights which we downloaded and not including the top layers after loading the model we set the ChexNet model trainable parameters to false as we want to use the same weights every time and don't want to update those weights in back propagation.

C. Downloading and Implementing GloVe:

```
#!wget https://nlp.stanford.edu/data/glove.6B.zip
#!unzip glove.6B.zip
#!cp -r 'glove.6B.50d.txt' '/content/drive/MyDrive/Project_on_Drive/glove'
#!cp -r 'glove.6B.100d.txt' '/content/drive/MyDrive/Project_on_Drive/glove'
#!cp -r 'glove.6B.200d.txt' '/content/drive/MyDrive/Project_on_Drive/glove'
#!cp -r 'glove.6B.300d.txt' '/content/drive/MyDrive/Project_on_Drive/glove'
#! rm /content/glove.txtmililjjwtmp /content/glove.txt /content/glove.6B.zip /content/glove.zipey89q8_xtmp glove.6B.50d.txt glove.6B.100d.txt glove.6B.200d.txt
```

```
glove = {}
with open('/content/drive/MyDrive/Project_on_Drive/glove/glove.6B.300d.txt',encoding='utf-8') as f: #taking 300 dimesions
  for line in tqdm(f):
    word = line.split() #it is stored as string like this "'the': '.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.4"
    glove[word[0]] = np.asarray(word[1:], dtype='float32')

embedding_dim = 300
# create a weight matrix for words in training docs for embedding purpose
embedding_matrix = np.zeros((vocab_size+1, embedding_dim)) #https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding

for word, i in tqdm(tokenizer.word_index.items()):
  embedding_vector = glove.get(word)
  if embedding_vector is not None: #if the word is found in glove vectors
      embedding_matrix[i] = embedding_vector[:embedding_dim]
```

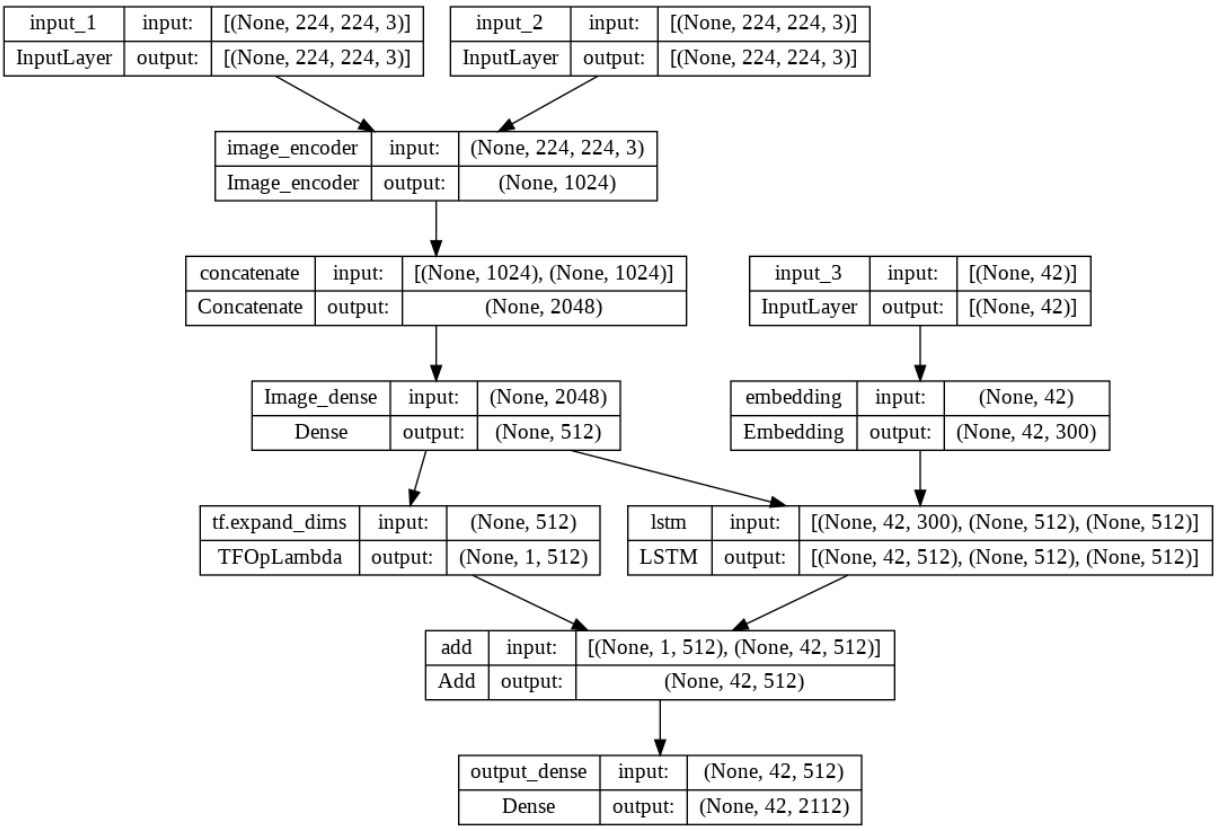
We downloaded and used 300 dimensions pre trained GloVe vectors and we are creating weight matrix for words in training docs for the purpose of word embeddings, from the pre-trained GloVe vectors.

d. **Image Encoder Layer**

```
class Image_encoder(tf.keras.layers.Layer):
  """
  This layer will output image backbone features after passing it through chexnet here chexnet will be not be trainable
  """
  def __init__(self,name = "image_encoder_block"):
    super().__init__()
    self.chexnet = create_chexnet()
    self.chexnet.trainable = False
  def call(self,data):
    op = self.chexnet(data)
    return op
```

This layer will output image backbone features after passing it through ChexNet here ChexNet will be not be trainable.

e. **Encode-Decoder Model**



Here in this model, we are using three inputs, two for image1 and image2 and another caption. We are using ChexNet for Image embeddings and we are using keras. Embedding layer with predefined weights as the Embedding Matrix which we created from GloVe Vectors. As we have very less data, we are only using single LSTM layer in order not to overfit (tried with multiple layers but observed overfitting issues).

At last, we a have a dense layer with SoftMax activation and number of units as vocabulary size + 1. We are converting this image captioning problem as a multi class classification and training the model to understand the sequence of the words.

f. Code Sample of Encoder - Decoder Model and Model Summary:

```
tf.keras.backend.clear_session()
image1 = Input(shape = (input_size + (3,))) #shape = 224,224,3
image2 = Input(shape = (input_size + (3,))) #https://www.w3resource.com/python-exercises/tuple/python-tuple-exercise-5.php
caption = Input(shape = (max_pad,))
img_encoder = Image_encoder() #contains chexnet model which is set trainable  =  False
bk_feat1 = img_encoder(image1)
#bk_dense1 = Dense(dense_dim,activation = 'relu',name = 'bk_dense1') # avoiding deep network due to over fitting on train data
#bk_feat1 = bk_dense1(bk_feat1) #dense for the first image op: (?,dense_dim)  # avoiding deep network due to over fitting on train data
bk_feat2 = img_encoder(image2)
#bk_dense2 = Dense(dense_dim,activation = 'relu',name = 'bk_dense2') # avoiding deep network due to over fitting on train data
#bk_feat2 = bk_dense2(bk_feat2) #dense for the 2nd image op: (?,dense_dim) # avoiding deep network due to over fitting on train data
bk_features_concat = Concatenate(axis=-1)([bk_feat1,bk_feat2]) #concatenating the backbone images op_shape: (?,1024)
#bk_features_concat = BatchNormalization()(bk_features_concat) #applying batch norm
#bk_features_concat = Dropout(dropout_rate)(bk_features_concat)  # avoiding deep network due to over fitting on train data
image_dense = Dense(dense_dim,activation = 'relu',name = 'Image_dense',use_bias='False')
image_bkbone = image_dense(bk_features_concat) #final op from dense op_shape: (?,dense_dim) this will be added as initial states to the lstm
image_dense_op = tf.keras.backend.expand_dims(image_bkbone,axis=1) #op_shape: (?,1,dense_dim)
embedding = Embedding(input_dim = vocab_size+1,output_dim = embedding_dim,input_length = max_pad,mask_zero = True,weights = [embedding_matrix],name = 'embedding')
embed_op = embedding(caption) #op_shape: (?,input_length,embedding_dim)
lstm_layer = LSTM(units = lstm_units,return_sequences= True,return_state = True)
lstm_op,lstm_h,lstm_c = lstm_layer(embed_op,initial_state = [image_bkbone,image_bkbone]) #op_shape = batch_size*input_length*lstm_units
add = Add()([image_dense_op,lstm_op]) #op_shape: (?,input_lenght,lstm_units/dense_dim) here lstm_dims=dense_dim
#add = BatchNormalization()(add)
#add = Dropout(dropout_rate)(add)
op_dense = Dense(vocab_size+1,activation = 'softmax',name = 'output_dense') #op: (?,input_length,vocab_size+1)
output = op_dense(add)
model = tf.keras.Model(inputs = [image1,image2,caption], outputs = output)
```

```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 image_encoder (Image_encoder)  (None, 1024)         7037504     ['input_1[0][0]',
                                                                  'input_2[0][0]']

 concatenate (Concatenate)      (None, 2048)         0           ['image_encoder[0][0]',
                                                                  'image_encoder[1][0]']

 input_3 (InputLayer)           [(None, 42)]         0           []

 Image_dense (Dense)            (None, 512)          1049088     ['concatenate[0][0]']

 embedding (Embedding)          (None, 42, 300)      633600      ['input_3[0][0]']

 tf.expand_dims (TFOpLambda)    (None, 1, 512)       0           ['Image_dense[0][0]']

 lstm (LSTM)                    [(None, 42, 512),    1665024     ['embedding[0][0]',
                                 (None, 512),                     'Image_dense[0][0]',
                                 (None, 512)]                     'Image_dense[0][0]']

 add (Add)                      (None, 42, 512)      0           ['tf.expand_dims[0][0]',
                                                                  'lstm[0][0]']

 output_dense (Dense)           (None, 42, 2112)     1083456     ['add[0][0]']

==================================================================================================
Total params: 11,468,672
Trainable params: 4,431,168
Non-trainable params: 7,037,504
__________________________________________________________________________________________________
```

Code Sample and model summary are self-explanatory and we can observe that there around 7 million non-trainable parameters which are of the ChexNet models which we set those parameters as non-trainable.

g. Custom Loss Function:

```
loss_func = tf.keras.losses.SparseCategoricalCrossentropy()

def custom_loss(y_true, y_pred):
    #getting mask value to not consider those words which are not present in the true caption
    mask = tf.math.logical_not(tf.math.equal(y_true, 0))

    #y_pred = y_pred+10**-7 #to prevent loss becoming null

    #calculating the loss
    loss_ = loss_func(y_true, y_pred)

    #converting mask dtype to loss_ dtype
    mask = tf.cast(mask, dtype=loss_.dtype)

    #applying the mask to loss
    loss_ = loss_*mask

    #returning mean over all the values
    return tf.reduce_mean(loss_)
```

Created custom loss function in order to apply mask on the existing loss function SparseCategoricalCrossentropy (), and we are using custom loss for getting mask value to not consider those words which are not present in the true caption.

h. Adding Callbacks:

```
#tf.keras.backend.clear_session()
tb_filename = 'Simple_Encoder_Decoder/'
tb_file = os.path.join('/content/drive/MyDrive/Project_on_Drive/',tb_filename)
model_filename = 'Simple_Encoder_Decoder.h5'
model_save = os.path.join('/content/drive/MyDrive/Project_on_Drive/',model_filename)
my_callbacks = [
    tf.keras.callbacks.EarlyStopping(patience = 5,verbose = 2),
    tf.keras.callbacks.ModelCheckpoint(filepath=model_save,save_best_only = True,save_weights_only = True,verbose = 2),
    tf.keras.callbacks.TensorBoard(histogram_freq=1,log_dir=tb_file),
    tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1,patience=2, min_lr=10**-7, verbose = 2)
] #from keras documentation
```

We are adding four call backs and they are:

1) Early Stopping:

   For early stopping of the training if the loss is not improving than expected we are using patient level as 5 so our model looks for 5 iterations and still loss is not improving then training process is stopped.

2) Model checkpoint:

   We are checkpointing our training and saving best weights only during the process of training. For that we need to specify the path for saving the weights.

3) Tensor board:

   We are using Tensor board for the bias variance trade off and keep a tab

on the weights whether are there any exploding or diminishing gradients.

4) ReduceLROnPlateau:

This callback is used to monitor the validation loss and reduce the learning rate if there is no improvement in the validation loss for the patience level of 2 that means for continuous 2 iterations, we used initial learning as 10^{-3} and using this call back learning rate can reduce to minimum of 10^{-7} as per our setting.

I. Model Training and hyper parameter tuning:

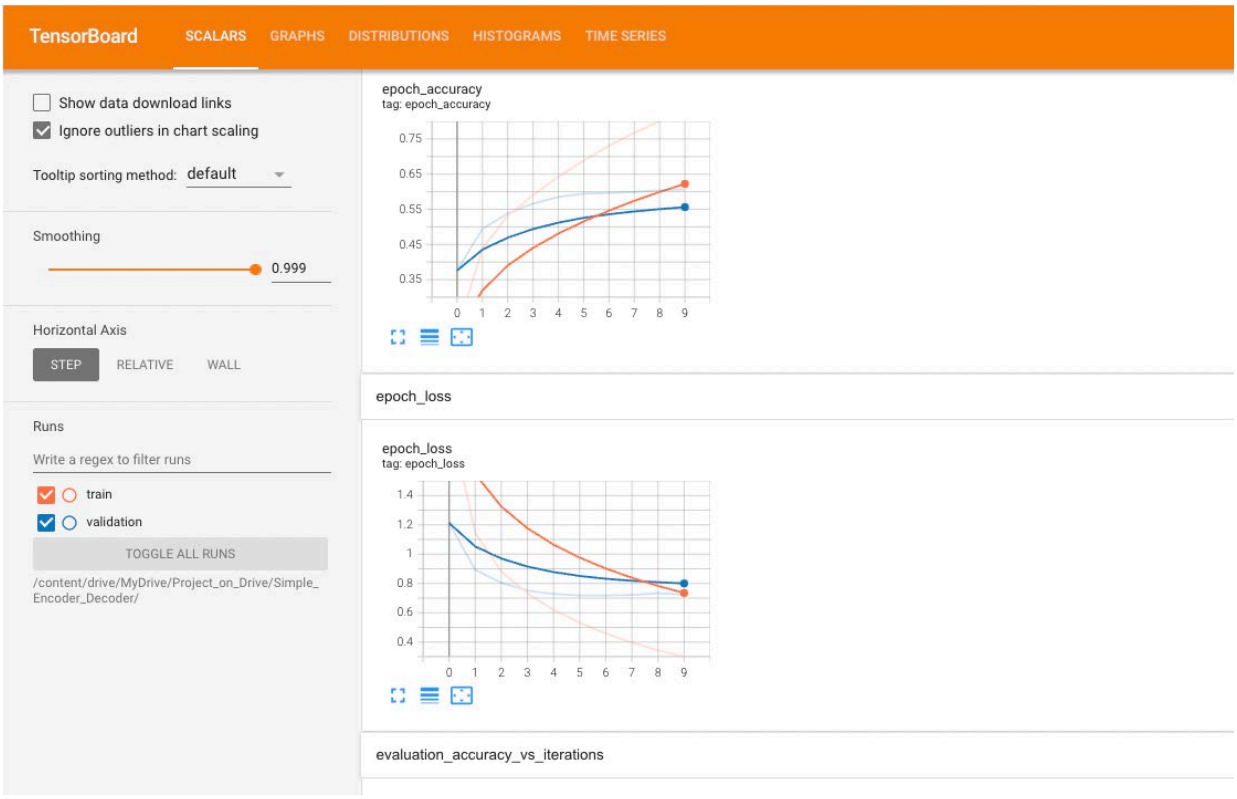
```
model.fit(train_dataloader,validation_data = test_dataloader,epochs = 10,callbacks = my_callbacks)
```

```
Epoch 1/10
87/87 [==============================] - ETA: 0s - loss: 1.9479 - accuracy: 0.2002
Epoch 1: val_loss improved from inf to 1.21250, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 177s 2s/step - loss: 1.9479 - accuracy: 0.2002 - val_loss: 1.2125 - val_accuracy: 0.3760 - lr: 0.0010
Epoch 2/10
87/87 [==============================] - ETA: 0s - loss: 1.1461 - accuracy: 0.4384
Epoch 2: val_loss improved from 1.21250 to 0.89277, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 152s 2s/step - loss: 1.1461 - accuracy: 0.4384 - val_loss: 0.8928 - val_accuracy: 0.4942 - lr: 0.0010
Epoch 3/10
87/87 [==============================] - ETA: 0s - loss: 0.8805 - accuracy: 0.5318
Epoch 3: val_loss improved from 0.89277 to 0.80477, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 153s 2s/step - loss: 0.8805 - accuracy: 0.5318 - val_loss: 0.8048 - val_accuracy: 0.5383 - lr: 0.0010
Epoch 4/10
87/87 [==============================] - ETA: 0s - loss: 0.7331 - accuracy: 0.5898
Epoch 4: val_loss improved from 0.80477 to 0.75141, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 152s 2s/step - loss: 0.7331 - accuracy: 0.5898 - val_loss: 0.7514 - val_accuracy: 0.5663 - lr: 0.0010
Epoch 5/10
87/87 [==============================] - ETA: 0s - loss: 0.6195 - accuracy: 0.6427
Epoch 5: val_loss improved from 0.75141 to 0.72767, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 152s 2s/step - loss: 0.6195 - accuracy: 0.6427 - val_loss: 0.7277 - val_accuracy: 0.5848 - lr: 0.0010
Epoch 6/10
87/87 [==============================] - ETA: 0s - loss: 0.5344 - accuracy: 0.6882
Epoch 6: val_loss improved from 0.72767 to 0.71866, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 151s 2s/step - loss: 0.5344 - accuracy: 0.6882 - val_loss: 0.7187 - val_accuracy: 0.5946 - lr: 0.0010
Epoch 7/10
87/87 [==============================] - ETA: 0s - loss: 0.4586 - accuracy: 0.7305
Epoch 7: val_loss improved from 0.71866 to 0.71796, saving model to /content/drive/MyDrive/Project_on_Drive/Simple_Encoder_Decoder.h5
87/87 [==============================] - 152s 2s/step - loss: 0.4586 - accuracy: 0.7305 - val_loss: 0.7180 - val_accuracy: 0.5961 - lr: 0.0010
Epoch 8/10
87/87 [==============================] - ETA: 0s - loss: 0.3961 - accuracy: 0.7674
Epoch 8: val_loss did not improve from 0.71796
87/87 [==============================] - 151s 2s/step - loss: 0.3961 - accuracy: 0.7674 - val_loss: 0.7212 - val_accuracy: 0.5994 - lr: 0.0010
Epoch 9/10
87/87 [==============================] - ETA: 0s - loss: 0.3438 - accuracy: 0.8003
Epoch 9: val_loss did not improve from 0.71796

Epoch 9: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
87/87 [==============================] - 151s 2s/step - loss: 0.3438 - accuracy: 0.8003 - val_loss: 0.7327 - val_accuracy: 0.6037 - lr: 0.0010
Epoch 10/10
87/87 [==============================] - ETA: 0s - loss: 0.3007 - accuracy: 0.8295
Epoch 10: val_loss did not improve from 0.71796
87/87 [==============================] - 151s 2s/step - loss: 0.3007 - accuracy: 0.8295 - val_loss: 0.7246 - val_accuracy: 0.6065 - lr: 1.0000e-04
<keras.callbacks.History at 0x7fdlca6c6590>
```

We trained our model for 10 epochs and able to achieve the train loss as 0.3007 and train accuracy as 0.8295 and validation loss as 0.7246 and validation accuracy as 0.6065.

We can clearly see that learning reduced to $1.000e^{-4}$ due to ReduceLROnPlateau callback. And we can clearly see that validation loss is not improving from Epoch 8 as we have very less data, we are not able to reach the global minima in optimization and it looks like simple Encoder – Decoder model is not enough for achieving good accuracy for this image captioning problem.

And below Tensor board we can clearly see that as the number of epochs increasing train accuracy and increasing and validation accuracy is decreasing which indicate over fitting of the model and hence stopped as 10 iterations for bias - variance tradeoff.



Epoch loss looks converging but we have train loss as 0.3007 validation loss as 0.7246 so we have around 0.42 difference in the loss which is not a huge difference but we can still work on it and make our model better by adding attention mechanism in the encoder – decoder stack.

After multiple Runs and find tuning we are able to best results with the following Hyper Parameters.

- batch_size = 100
- embedding_dim = 300
- dense_dim = 512
- lstm_units = dense_dim
- dropout_rate = 0.5
- lr (Learning Rate) = 10**-3
- number of epochs = 10
- min_lr (Minimum Learning rate) =10**-7

J. Test caption prediction and BLEU score using Greedy search:

Greedy Search:

A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall

optimal result. The algorithm never reverses the earlier decision even if the choice is wrong.

We decided on greedy search as we are predicting the text and we want to predict next best word probability after each word and it is not computationally expensive, we tried with some of the heuristic search algorithms like beam search they turned out to be computationally expensive and not at all feasible even though we are using good GPU (google colab pro Tesla P100 GPU) as the computer power for this model building.

1. Code to compute bleu scores 1-gram,2-gram, 3-gram and 4-gram for each sentence.

```
def bleu_score(reference,prediction):
  """
  Given a reference and prediction string, outputs the 1-gram,2-gram,3-gram and 4-gram bleu scores
  """
  reference = [reference.split()] #should be in an array (cos of multiple references can be there here only 1)
  prediction = prediction.split()
  bleu1 = sentence_bleu(reference,prediction,weights = (1,0,0,0),)
  bleu2 = sentence_bleu(reference,prediction,weights = (0.5,0.5,0,0))
  bleu3 = sentence_bleu(reference,prediction,weights = (0.33,0.33,0.33,0))
  bleu4 = sentence_bleu(reference,prediction,weights = (0.25,0.25,0.25,0.25))

  return bleu1,bleu2,bleu3,bleu4
```

2. Code for computing mean BLEU scores

```
#calculate bleu scores for every datapoint
def mean_bleu_score(test,predict,model=model1,**kwargs):
  """
  given a df and predict function which predicts the Findings of the caption
  outpus the mean bleu1,bleu2,bleu3, bleu4 for entire datapoints in df
  """
  if kwargs!=None:
    top_k = kwargs.get('top_k')
  else:
    top_k = None
  bleu1,bleu2,bleu3,bleu4 = [],[],[],[]
  for index,data in tqdm(test.iterrows()):
    if top_k==None:
      predict_val = predict(data['image1'],data['image2'],model = model) #predicted sentence
    else:
      predict_val = predict(data['image1'],data['image2'],model = model,top_k = top_k)
    true = data.findings
    scores = bleu_score(true,predict_val)
    bleu1.append(scores[0])
    bleu2.append(scores[1])
    bleu3.append(scores[2])
    bleu4.append(scores[3])
  return np.array(bleu1).mean(),np.array(bleu2).mean(),np.array(bleu3).mean(),np.array(bleu4).mean()
```

3. Code for test caption prediction using greedy search

```
def greedy_search_predict(image1,image2,model = model1):
  """
  Given paths to two x-ray images predicts the Findings part of the x-ray in a greedy search algorithm
  https://www.youtube.com/watch?v=HzeK7g8cD0Y
  """
  image1 = cv2.imread(image1,cv2.IMREAD_UNCHANGED)/255
  image2 = cv2.imread(image2,cv2.IMREAD_UNCHANGED)/255
  image1 = tf.expand_dims(cv2.resize(image1,input_size,interpolation = cv2.INTER_NEAREST),axis=0) #introduce batch and resize
  image2 = tf.expand_dims(cv2.resize(image2,input_size,interpolation = cv2.INTER_NEAREST),axis=0)


  image1 = model.get_layer('image_encoder')(image1) #output from chexnet
  image2 = model.get_layer('image_encoder')(image2)

  # image1 = model.get_layer('bk_dense')(image1) #op from dense layer
  # image2 = model.get_layer('bk_dense')(image2)
```

```
concat = model.get_layer('concatenate')([image1,image2])
image_dense = model.get_layer('Image_dense')(concat)
# concat = model.get_layer('batch_normalization')(concat)
# image_dense = model.get_layer('Image_dense')(concat)
bk_feat = tf.keras.backend.expand_dims(image_dense,axis=1)
states = [image_dense, image_dense]
a = []
pred = []
for i in range(max_pad):
  if i==0: #if first word
    caption = np.array(tokenizer.texts_to_sequences(['<start>'])) #shape: (1,1)
  caption= model.get_layer('embedding')(caption) #embedding shape = 1*1*300
  caption,state_h,state_c = model.get_layer('lstm')(caption,initial_state = states) #lstm 1*1*512
  states = [state_h,state_c]
  add = model.get_layer('add')([bk_feat,caption]) #add
  output = model.get_layer('output_dense')(add) #1*1*vocab_size (here batch_size=1)
  #prediction
  max_prob = tf.argmax(output,axis=-1) #tf.Tensor of shape = (1,1)
  caption = np.array(max_prob) #will be sent to embedding for next iteration
  if max_prob==np.squeeze(tokenizer.texts_to_sequences(['<end>'])):
    break;
  else:
    a.append(tf.squeeze(max_prob).numpy())
return tokenizer.sequences_to_texts([a])[0] #here output would be 1,1 so subscripting to open the array
```

4. BLEU scores on test data using Encoder – decoder model

```
result
```

	bleu1	bleu2	bleu3	bleu4
greedy search	0.261584	0.142195	0.086271	0.040068

5. Inference of the test captions

```
test['prediction_gs'].value_counts()*100/test.shape[0] #greedy search
```

```
the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal.    79.470199
the heart is normal in size. the mediastinum is unremarkable. the lungs are clear.                                                            15.728477
the cardiomediastinal silhouette is normal in size and contour. no focal consolidation, pneumothora or large pleural effusion. negative for acute displaced rib fracture.    3.807947
the heart is normal in size. the mediastinum is unremarkable. the lungs are clear. there is no pleural effusion or pneumothora.               0.827815
the cardiomediastinal silhouette is within normal limits for appearance. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. the thoracic spine appears intact.    0.165563
Name: prediction_gs, dtype: float64
```

As we can clearly see that our encoder – decoder model is predicting only very few captions and most of the captions are non-disease captions like “the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal.” And “the heart is normal in size. the mediastinum is unremarkable. the lungs are clear.”

It is a good sign that our model is predicting sensible captions and which are meaningful but biased towards the non-disease captions as the data we have is very less for applying a deep learning model. And let’s add attention in the encoder – decoder stack and make use of GRU instead LSTM and make the model better and un-biased towards the non-disease data.

Step4: Building Encoder-Decoder model with Global Attention

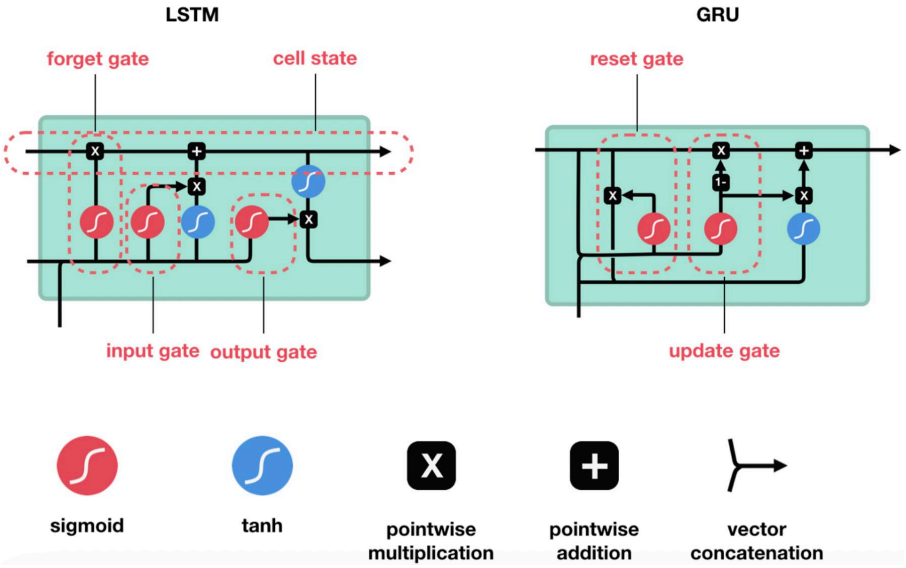
I. Understanding Prerequisites

a. GRU:

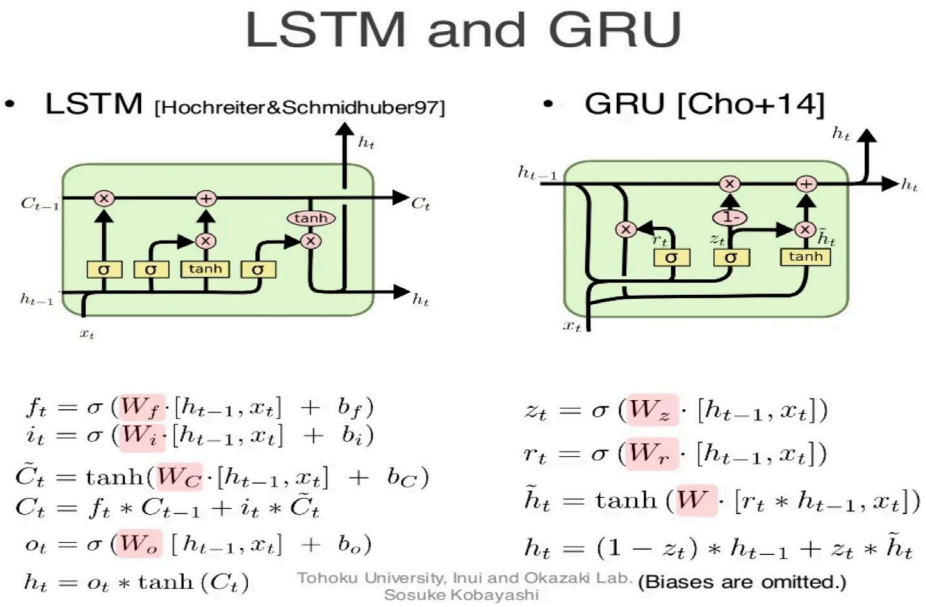
GRU stands for Gated Recurrent Unit, these are modern simplified versions of the LSTM, LSTM have vanishing gradient problem due to long time dependency, this problem is addressed by GRU and GRU is as powerful as LSTM and faster to train.

GRU have only 2 gates instead of 3 gates like LSTMs, these gates are Reset and Update gates.as we have fewer equations in GRU to solve we will be having fewer partial derivatives and thus have faster back propagation compared to LSTM.

Understanding architecture difference between LSTM and GRU:



Understanding Equations of LSTM and GRU:



b. Attention Mechanisms:

Attention models, also called attention mechanisms, are deep learning techniques used to provide an additional focus on a specific component. In deep learning, attention relates to focus on something in particular and note its specific importance. The model typically focuses on one component within the network's architecture that's responsible for managing and quantifying the interdependent relationships within input elements, called self-attention, or between input and output elements, called general attention.

The aim of attention models is to reduce larger, more complicated tasks into smaller, more manageable areas of attention to understand and process sequentially. The models work within neural networks, which are a type of network model with a similar structure and processing methods as the human brain for simplifying and processing information. Using attention models enables the network to focus on a few particular aspects at a time and ignoring the rest. This allows for efficient and sequential data processing, especially when the network needs to categorize entire datasets.

The initial purpose of attention models was to help improve computer vision and the encoder-decoder-based neural machine translation system. This system uses natural language processing (NLP) and relies on huge data libraries that have complex functions. However, using attention models helps create maps to fixed-length vectors to generate translations and understanding. While these may not be wholly accurate, it provides a result that represents the general sentiment and intention of the initial input.

Attention models strive to resolve the potential limits of the encoder-decoder model. It helps accurately align and translate the input items. However, it develops a context vector filtered specifically for each output, rather than encoding the input sequence as a single fixed content vector.

Types of Attention Models:

1. Self-Attention Model
2. Global Attention Model
3. Local Attention Model

Here we are using Global Attention Model for our thesis which is suggested by Bahdanau and Loung. And Global Attention Model is widely used Attention Mechanism for Attention.

Global Attention Mechanism:

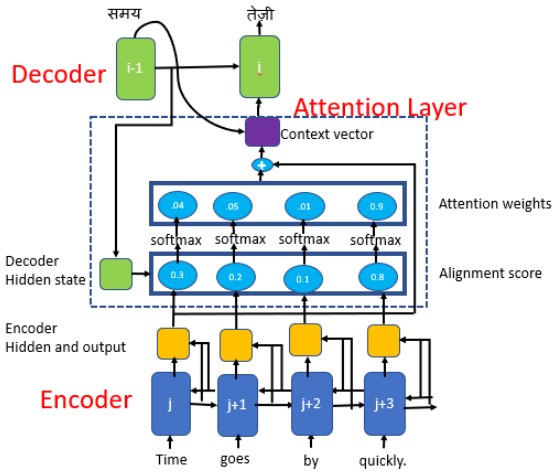
The global attention model, which is also similar to the soft attention model, collects inputs from all encoder and decoder states before evaluating the current state to determine the output. This model uses each encoder step and each decoder preview step to calculate the attention weights or align weights. It also multiplies each encoder step by global align weights to determine the context value to feed to the recurrent neural networks (RNN) cell. This allows the model to find the decoder output.

Bahdanau et al. proposed an attention mechanism that learns to align and translate jointly. It is also known as Additive attention as it performs a linear combination of encoder states and the decoder states.

All hidden states of the encoder(forward and backward) and the decoder are used to generate the context vector, unlike how just the last encoder hidden state is used in seq2seq without attention.

The attention mechanism aligns the input and output sequences, with an alignment score parameterized by a feed-forward network. It helps to pay attention to the most relevant information in the source sequence.

The model predicts a target word based on the context vectors associated with the source position and the previously generated target words. Seq2Seq model with an attention mechanism consists of an encoder, decoder, and attention layer.



Bahdanau et al. attention mechanism

Attention Layer consists of

- Alignment Layer
- Attention Weights
- Context Vector

Alignment Layer:

The alignment score maps how well the inputs around position *"j"* and the output at position "*i"* match. The score is based on the previous decoder's hidden state, $\mathbf{s_{(i-1)}}$ just before predicting the target word and the hidden state, h_j of the input sentence

The decoder decides which part of the source sentence it needs to pay attention to, instead of having encoder encode all the information of the source sentence into a fixed-length vector.

$$e_{ij.} = a(s_{i-1}, h_j) \qquad \text{Alignment Score}$$

Attention Weights:

We apply a SoftMax activation function to the alignment scores to obtain the attention weights. SoftMax activation function will get the probabilities whose sum will be equal to 1, This will help to represent the weight of influence for each of the input sequence. Higher the attention weight of the input sequence, the higher will be its influence on predicting the target word.

$$\alpha_{i_j} = \exp(e_{ij}) \Big/ \sum_{k=1}^{Tx} \exp(e_{ik}) \qquad \text{Attention weight}$$

Context Vector:

The context vector is used to compute the final output of the decoder. The context vector c_i is the weighted sum of attention weights and the encoder hidden states ($h_1, h_2, \ldots, h_{tx}$), which maps to the input sentence.

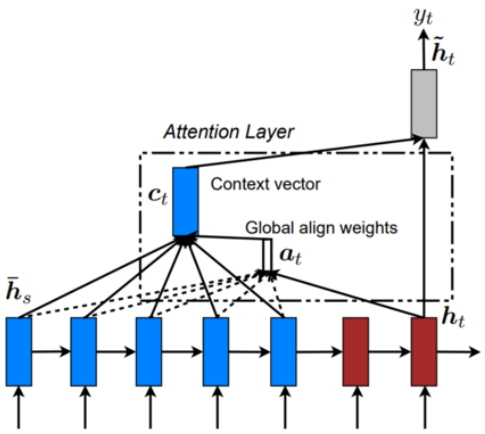
$$C_i = \sum_{j=1}^{Tx} \alpha_{ij} h_j \quad \text{Context vector}$$

Predicting the Target Word:

To predict the target word, the decoder uses

a. Context vector(c_i),
b. Decoder's output from the previous time step (y_{i-1}) and
c. Previous decoder's hidden state($\mathbf{s_{i-1}}$)

$$s_i = f(s_{i-1}, c_i, y_{i-1})$$



Global Attention Mechanism

The global attentional model considers all the hidden states of the encoder when calculating the context vector c_t. A variable-length alignment vectors a_t equal to the size of the number of time steps in the source sequence is derived by comparing the current target hidden state h_t with each of the source hidden state h_s The alignment score is referred to as a content-based function for which we consider three different alternatives and out of those 3 alternatives we are using concat version of score.

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_a \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_a[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

Global context vector c_t is calculated as the weighted average according to alignment vector a_t over all the source hidden states h_s. As Global attention model considers all the words of the source sequence to predict the target words, it becomes computationally expensive and can be challenging to translate longer sentences. And this solved by using Local Attention.

II. Encoder-Decoder with Attention Model Building

a. Global Attention Layer

We implemented Global Attention Layer as discussed above and we used concat version of score to get the alignment score and later used SoftMax activation on it to get the alignment weights and got the context vector by the weighted sum of attention weights and the encoder hidden states.

```
class global_attention(tf.keras.layers.Layer):
  """
  calculate global attention
  """
  def __init__(self,dense_dim = dense_dim):
    super().__init__()
    # Intialize variables needed for Concat score function here
    self.W1 = Dense(units = dense_dim) #weight matrix of shape enc_units*dense_dim
    self.W2 = Dense(units = dense_dim) #weight matrix of shape dec_units*dense_dim
    self.V = Dense(units = 1) #weight matrix of shape dense_dim*1
      #op (None,98,1)


  def call(self,encoder_output,decoder_h): #here the encoded output will be the concatted image bk features shape: (None,98,dense_dim)
    decoder_h = tf.expand_dims(decoder_h,axis=1) #shape: (None,1,dense_dim)
    tanh_input = self.W1(encoder_output) + self.W2(decoder_h) #ouput_shape: batch_size*98*dense_dim
    tanh_output =  tf.nn.tanh(tanh_input)
    attention_weights = tf.nn.softmax(self.V(tanh_output),axis=1) #shape= batch_size*98*1 getting attention alphas
    op = attention_weights*encoder_output#op_shape: batch_size*98*dense_dim  multiply all aplhas with corresponding context vector
    context_vector = tf.reduce_sum(op,axis=1) #summing all context vector over the time period ie input length, output_shape: batch_size*dense_dim


    return context_vector,attention_weights
```

b. Data Pipeline, ChexNet and GloVe word embedding:

Built data pipeline and used the ChexNet pre-trained weights for the image embeddings and GloVe pretrained 300-dimensional word vectors for word embeddings as explained in the encoder-decoder model section.

c. Encoder Layer:

```
def encoder(image1,image2,dense_dim = dense_dim,dropout_rate = dropout_rate):
  """
  Takes image1,image2
  gets the final encoded vector of these
  """
  #image1
  im_encoder = Image_encoder()
  bkfeat1 = im_encoder(image1) #shape: (None,9,1024)
  bk_dense = Dense(dense_dim,name = 'bkdense',activation = 'relu') #shape: (None,9,512)
  bkfeat1 = bk_dense(bkfeat1)

  #image2
  bkfeat2 = im_encoder(image2) #shape: (None,9,1024)
  bkfeat2 = bk_dense(bkfeat2) #shape: (None,9,512)


  #combining image1 and image2
  concat = Concatenate(axis=1)([bkfeat1,bkfeat2]) #concatenating through the second axis shape: (None,18,1024)
  bn = BatchNormalization(name = "encoder_batch_norm")(concat)
  dropout = Dropout(dropout_rate,name = "encoder_dropout")(bn)
  return dropout
```

Implemented Encoder Layer which encodes our input X-Rays with the ChexNet weights and concatenates then on the second axis and later we used Batch Normalization and dropouts and returning the encoded image vector.

d. One Step Decoder:

One step decoder mechanism step by step:

1. Pass the input_to_decoder to the embedding layer and then get the output (batch_size,1, embedding_dim)
2. Using the encoder_output and decoder hidden state, compute the context vector.
3. Concat the context vector with the step A output
4. Pass the Step-C output to LSTM/GRU and get the decoder output and states (hidden and cell state)

5. Pass the decoder output to dense layer (vocab size) and store the result into output.
6. Return the states from step D, output from Step E, attention weights from Step -B

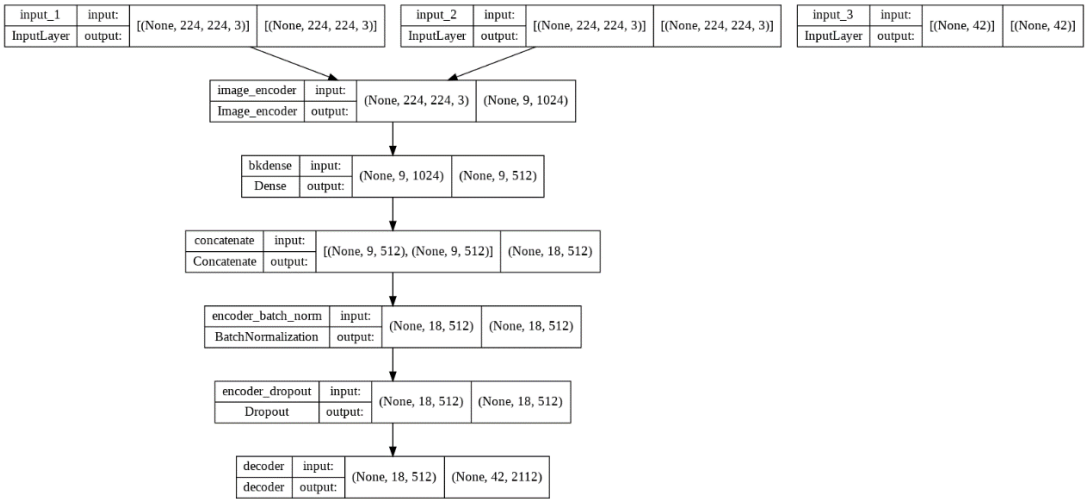
```
class One_Step_Decoder(tf.keras.layers.Layer):
  """
  decodes a single token
  """
  def __init__(self,vocab_size = vocab_size, embedding_dim = embedding_dim, max_pad = max_pad, dense_dim = dense_dim ,name = "onestepdecoder"):
    # Initialize decoder embedding layer, LSTM and any other objects needed
    super().__init__()
    self.dense_dim = dense_dim
    self.embedding = Embedding(input_dim = vocab_size+1,
                               output_dim = embedding_dim,
                               input_length=max_pad,
                               weights = [embedding_matrix],
                               mask_zero=True,
                               name = 'onestepdecoder_embedding'
                              )
    self.LSTM = GRU(units=self.dense_dim,return_sequences=True,return_state=True,name = 'onestepdecoder_LSTM')
    self.LSTM1 = GRU(units=self.dense_dim,return_sequences=False,return_state=True,name = 'onestepdecoder_LSTM1')
    self.attention = global_attention(dense_dim = dense_dim)
    self.concat = Concatenate(axis=-1)
    self.dense = Dense(dense_dim,name = 'onestepdecoder_embedding_dense',activation = 'relu')
    self.final = Dense(vocab_size+1,activation='softmax')
    self.concat = Concatenate(axis=-1)
    self.add =Add()
  @tf.function
  def call(self,input_to_decoder, encoder_output, decoder_h):#,decoder_c):
    '''
        One step decoder mechanisim step by step:
      A. Pass the input_to_decoder to the embedding layer and then get the output(batch_size,1,embedding_dim)
      B. Using the encoder_output and decoder hidden state, compute the context vector.
      C. Concat the context vector with the step A output
      D. Pass the Step-C output to LSTM/GRU and get the decoder output and states(hidden and cell state)
      E. Pass the decoder output to dense layer(vocab size) and store the result into output.
      F. Return the states from step D, output from Step E, attention weights from Step -B

      here state_h,state_c are decoder states
    '''
    embedding_op = self.embedding(input_to_decoder) #output shape = batch_size*1*embedding_shape (only 1 token)


    context_vector,attention_weights = self.attention(encoder_output,decoder_h) #passing hidden state h of decoder and encoder output
    #context_vector shape: batch_size*dense_dim we need to add time dimension
    context_vector_time_axis = tf.expand_dims(context_vector,axis=1)
    #now we will combine attention output context vector with next word input to the lstm here we will be teacher forcing
    concat_input = self.concat([context_vector_time_axis,embedding_op])#output dimension = batch_size*input_length(here it is 1)*(dense_dim+embedding_dim)

    output,decoder_h = self.LSTM(concat_input,initial_state = decoder_h)
    output,decoder_h = self.LSTM1(output,initial_state = decoder_h)
    #output shape = batch*1*dense_dim and decoder_h,decoder_c has shape = batch*dense_dim
    #we need to remove the time axis from this decoder_output


    output = self.final(output)#shape = batch_size*decoder vocab size
    return output,decoder_h,attention_weights
```

e. Decoder Mechanism:

```
class decoder(tf.keras.Model):
  """
  Decodes the encoder output and caption
  """
  def __init__(self,max_pad = max_pad, embedding_dim = embedding_dim,dense_dim = dense_dim,score_fun='general',batch_size = batch_size,vocab_size = vocab_size):
    super().__init__()
    self.onestepdecoder = One_Step_Decoder(vocab_size = vocab_size, embedding_dim = embedding_dim, max_pad = max_pad, dense_dim = dense_dim)
    self.output_array = tf.TensorArray(tf.float32,size=max_pad)
    self.max_pad = max_pad
    self.batch_size = batch_size
    self.dense_dim =dense_dim

  @tf.function
  def call(self,encoder_output,caption):#,decoder_h,decoder_c): #caption : (None,max_pad), encoder_output: (None,dense_dim)
    decoder_h, decoder_c = tf.zeros_like(encoder_output[:,0]), tf.zeros_like(encoder_output[:,0]) #decoder_h, decoder_c
    output_array = tf.TensorArray(tf.float32,size=max_pad)
    for timestep in range(self.max_pad): #iterating through all timesteps ie through max_pad
      output,decoder_h,attention_weights = self.onestepdecoder(caption[:,timestep:timestep+1], encoder_output, decoder_h)
      output_array = output_array.write(timestep,output) #timestep*batch_size*vocab_size

    self.output_array = tf.transpose(output_array.stack(),[1,0,2]) #.stack :Return the values in the TensorArray as a stacked Tensor.)
        #shape output_array: (batch_size,max_pad,vocab_size)
    return self.output_array
```

Decodes the Encoder output and Caption. Decoder iterates through all time steps till the maximum pad value and generates each word one by one taking previous words into picture.

f. Global Attention Encoder-Decoder Model:




```
model.summary()
```

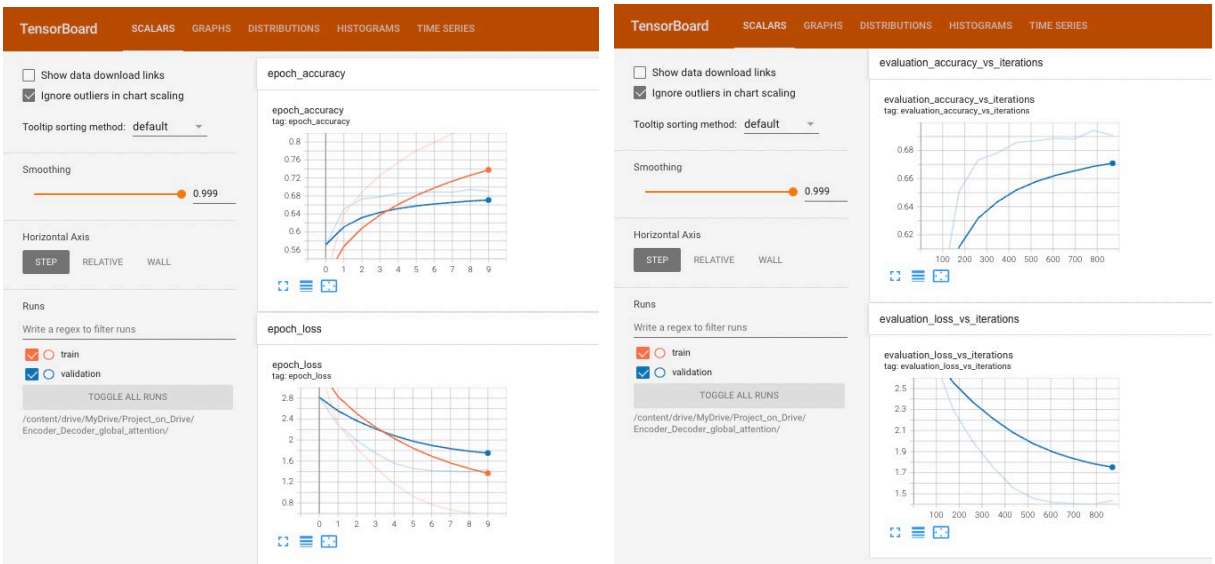
```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_1 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 input_2 (InputLayer)           [(None, 224, 224, 3  0           []
                                )]

 image_encoder (Image_encoder)  (None, 9, 1024)      7037504     ['input_1[0][0]',
                                                                  'input_2[0][0]']

 bkdense (Dense)                (None, 9, 512)       524800      ['image_encoder[0][0]',
                                                                  'image_encoder[1][0]']

 concatenate (Concatenate)      (None, 18, 512)      0           ['bkdense[0][0]',
                                                                  'bkdense[1][0]']

 encoder_batch_norm (BatchNorma  (None, 18, 512)     2048        ['concatenate[0][0]']
 lization)

 encoder_dropout (Dropout)      (None, 18, 512)      0           ['encoder_batch_norm[0][0]']

 input_3 (InputLayer)           [(None, 42)]         0           []

 decoder (decoder)              (None, 42, 2112)     5855553     ['encoder_dropout[0][0]',
                                                                  'input_3[0][0]']

==================================================================================================
Total params: 13,419,905
Trainable params: 6,381,377
Non-trainable params: 7,038,528
__________________________________________________________________________________________________
```

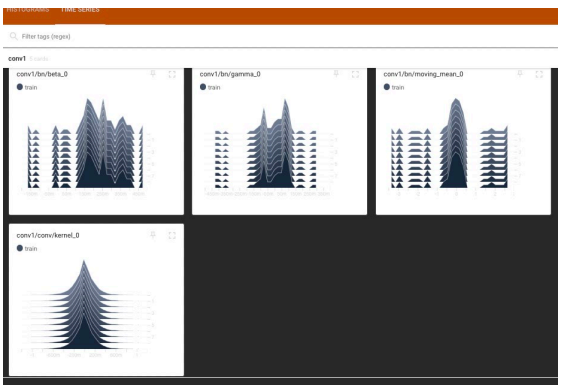
g. Model Training and hyper parameter tuning:

Ran model for 10 epochs and we can see that train loss is 0.5577, train accuracy is 0.8466, validation loss and accuracy are 1.4386 and 0.6907 respectively and if we run the model still further, we can get better loss and accuracy but looks like model is overfitting and we got the best results with 10 epochs, as data we have is less if we run for more and more epochs there is a chance of overfitting.

```
model.fit(train_dataloader,
          validation_data = test_dataloader,
          epochs = 10,
          callbacks = my_callbacks
          )
```

```
WARNING:tensorflow:Model failed to serialize as JSON. Ignoring...
Epoch 1/10
87/87 [==============================] - ETA: 0s - loss: 3.3515 - accuracy: 0.4966
Epoch 1: val_loss improved from inf to 2.81636, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 239s 2s/step - loss: 3.3515 - accuracy: 0.4966 - val_loss: 2.8164 - val_accuracy: 0.5714 - lr: 0.0100
Epoch 2/10
87/87 [==============================] - ETA: 0s - loss: 2.3061 - accuracy: 0.6394
Epoch 2: val_loss improved from 2.81636 to 2.29714, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 173s 2s/step - loss: 2.3061 - accuracy: 0.6394 - val_loss: 2.2971 - val_accuracy: 0.6508 - lr: 0.0100
Epoch 3/10
87/87 [==============================] - ETA: 0s - loss: 1.8525 - accuracy: 0.6887
Epoch 3: val_loss improved from 2.29714 to 1.99100, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 171s 2s/step - loss: 1.8525 - accuracy: 0.6887 - val_loss: 1.9910 - val_accuracy: 0.6733 - lr: 0.0100
Epoch 4/10
87/87 [==============================] - ETA: 0s - loss: 1.4772 - accuracy: 0.7257
Epoch 4: val_loss improved from 1.99100 to 1.75310, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 172s 2s/step - loss: 1.4772 - accuracy: 0.7257 - val_loss: 1.7531 - val_accuracy: 0.6782 - lr: 0.0100
Epoch 5/10
87/87 [==============================] - ETA: 0s - loss: 1.1668 - accuracy: 0.7539
Epoch 5: val_loss improved from 1.75310 to 1.55723, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 173s 2s/step - loss: 1.1668 - accuracy: 0.7539 - val_loss: 1.5572 - val_accuracy: 0.6858 - lr: 0.0100
Epoch 6/10
87/87 [==============================] - ETA: 0s - loss: 0.9258 - accuracy: 0.7805
Epoch 6: val_loss improved from 1.55723 to 1.45838, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 173s 2s/step - loss: 0.9258 - accuracy: 0.7805 - val_loss: 1.4584 - val_accuracy: 0.6869 - lr: 0.0100
Epoch 7/10
87/87 [==============================] - ETA: 0s - loss: 0.7762 - accuracy: 0.7987
Epoch 7: val_loss improved from 1.45838 to 1.41781, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 170s 2s/step - loss: 0.7762 - accuracy: 0.7987 - val_loss: 1.4178 - val_accuracy: 0.6888 - lr: 0.0100
Epoch 8/10
87/87 [==============================] - ETA: 0s - loss: 0.6720 - accuracy: 0.8187
Epoch 8: val_loss improved from 1.41781 to 1.40749, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 173s 2s/step - loss: 0.6720 - accuracy: 0.8187 - val_loss: 1.4075 - val_accuracy: 0.6881 - lr: 0.0100
Epoch 9/10
87/87 [==============================] - ETA: 0s - loss: 0.6113 - accuracy: 0.8305
Epoch 9: val_loss improved from 1.40749 to 1.39743, saving model to /content/drive/MyDrive/Project_on_Drive/Encoder_Decoder_global_attention.h5
87/87 [==============================] - 171s 2s/step - loss: 0.6113 - accuracy: 0.8305 - val_loss: 1.3974 - val_accuracy: 0.6943 - lr: 0.0100
Epoch 10/10
87/87 [==============================] - ETA: 0s - loss: 0.5577 - accuracy: 0.8446
Epoch 10: val_loss did not improve from 1.39743
87/87 [==============================] - 173s 2s/step - loss: 0.5577 - accuracy: 0.8446 - val_loss: 1.4386 - val_accuracy: 0.6907 - lr: 0.0100
```





As per the tensor board evaluation accuracy is increasing as the iterations increases and evaluation loss is decreasing as the iterations increases and it is a good sign to say that weights are converging and all the derivatives are in good range and there is no exploding or dying gradients as we only ran for 10 epochs and used batch normalization and drop outs properly we are not facing above issues.

After multiple Runs and find tuning we are able to get best results with the following Hyper Parameters.

- batch_size = 100
- embedding_dim = 300
- dense_dim = 512
- lstm_units = dense_dim
- dropout_rate = 0.2
- lr (Learning Rate) = 10**-2
- number of epochs = 10
- min_lr (Minimum Learning rate) =10**-7

h. Test caption prediction and BLEU score using Greedy search:

	bleu1	bleu2	bleu3	bleu4
greedy search	0.283326	0.166389	0.099237	0.053017

We can clearly see there is a good improvement in the bleu score when compared to simple encoder decoder model and we can see the predictions of the test captions are also improved previously using encoder-decoder model we are only predicting only few captions but with attention model it improved to the greater extent.

This model performed better than simple baseline model since it produced captions which had higher variability and also remained linguistically similar. Even then we can see that most of datapoints were of normal chest or no disease category we need to collect more dataset which have x-rays of patients having diseases so that to improve the model's performance. Even the model predicted tough captions which had similar meaning to the true ones.

Inference of the test captions:

As we can clearly see that our Attention based encoder – decoder model is predicting more variety of captions and both disease and non-disease captions like “there is a calcified granuloma in the left midlung. no focal infiltrate. no suspicious pulmonary nodules are present.” And “the lungs are clear. the heart and pulmonary are normal. the pleural spaces are clear. mediastinal contours are normal.”

```
test['prediction_gs'].value_counts()*100/test.shape[0] #greedy search
```

```
the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal. surgical clips are present in the right upper quadrant.
the cardiomediastinal silhouette is within normal limits for appearance. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. the thoracic spine appears intact.
the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal.
the lungs are clear. there is no pleural effusion or pneumothora. the heart is not significantly enlarged. there are atherosclerotic changes of the aorta. arthritic changes of the skeletal structures are noted.
the cardiomediastinal silhouette is within normal limits for size and contour. no focal airspace consolidation, pleural effusion, or pneumothora. the osseous structures are intact.
the cardiomediastinal silhouette is within normal limits for size and contour. no focal airspace consolidation, pleural effusion, or pneumothora. no acute osseous abnormality.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated with no focal airspace disease, pleural effusion, or pneumothora. no acute bone abnormality.
there is a calcified granuloma in the right midlung. the heart size is normal. no pneumothora.
the cardiomediastinal silhouette is within normal limits for appearance. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. no acute, displaced rib fractures identified.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated without evidence of focal airspace disease, pleural effusion, or pneumothora. no acute bone abnormality.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pleural effusion or pneumothora. heart size is within normal limits. degenerative changes are present within the spine.
the cardiomediastinal silhouette is within normal limits for appearance. the lungs are clear. there is no pneumothora or pleural effusion. the are unremarkable.
the cardiomediastinal silhouette is within normal limits for size and contour. no focal airspace consolidation, pneumothora or large pleural effusion. normal .
the lungs are clear. there is no pleural effusion or pneumothora. the heart is normal. there are atherosclerotic changes of the aorta. arthritic changes of the skeletal structures are noted.
the lungs are clear. the heart and pulmonary are normal. the pleural spaces are clear. mediastinal contours are normal.
the cardiomediastinal silhouette is within normal limits for size and contour. lung volumes are low with bronchovascular crowding. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion.
the cardiomediastinal silhouette is within normal limits for size and contour. lung volumes are low with bronchovascular crowding. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. no acute, displaced rib fractures.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated without evidence of focal airspace disease, pleural effusion, or pneumothora. degenerative changes of the spine.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pneumothora or large pleural effusion. no acute osseous abnormality.
the heart is top normal in size. the mediastinum is unremarkable. there is no pleural effusion, pneumothora, or focal airspace disease. calcific granulomas are present in the right lung.
the cardiomediastinal silhouette is within normal limits for size and contour. calcified granuloma in the right lung base. no focal airspace consolidation, pneumothora or large pleural effusion. negative for acute bone abnormality.
the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal. surgical clips are present in the upper quadrant.
the cardiomediastinal silhouette is within normal limits for size and contour. no focal airspace consolidation. no pleural effusion or pneumothora. no acute bony abnormality.
the lungs are clear. there is no pleural effusion or pneumothora. the heart is normal. there are atherosclerotic changes of the aorta. arthritic changes of the skeletal structures are within normal limits.
there is a large calcified granuloma in the left lung base. there is no pleural effusion or pneumothora. there is no focal airspace opacity to suggest a pneumonia. there are atherosclerotic institutions of the aorta. moderate degenerative changes of the spine.
the lungs are clear. there is no pleural effusion or pneumothora. the heart and mediastinum are normal. the skeletal structures are normal. surgical clips are again noted.
there is a calcified granuloma in the right midlung. the heart size top normal. the lungs are clear.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated with no focal airspace consolidation, pleural effusion, or pneumothora. no acute bone abnormality.
there is a calcified granuloma in the left midlung. no focal infiltrate. no suspicious pulmonary nodules are present.
the lungs are clear. there is no pleural effusion or pneumothora. the heart is normal. there are atherosclerotic changes of the aorta. arthritic changes of the skeletal structures are normal.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated without evidence of focal airspace disease, pleural effusion, or pneumothora. there is no acute bony findings.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated with no focal airspace consolidation, large pleural effusion, or pneumothora. the visualized osseous structures and soft tissues are grossly unremarkable.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pleural effusion or pneumothora. the heart size is within normal limits. the cardiomediastinal silhouette is within normal limits. the lungs are grossly unremarkable.
there is a large calcified lymph node at the left lung base. there is no pleural effusion. the cardiomediastinal silhouette is within normal limits. the lungs are clear bilaterally. there is no focal consolidation, pleural effusion, or pneumothora. the osseous structures are
there is a calcified granuloma in the left midlung. no focal infiltrate. no suspicious pulmonary opacity, large pleural effusion identified. there are mild cardiomegaly and right midlung. the cardiomediastinal silhouette is within normal limits.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pleural effusion or pneumothora. cardiomediastinal silhouette is within normal limits. the lungs are grossly unremarkable.
the cardiomediastinal silhouette is within normal limits for appearance. the lungs are clear. there is no pneumothora or large pleural effusion. mild degenerative changes of the thoracic spine.
the cardiomediastinal silhouette is within normal limits for appearance. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. the thoracic spine appears intact. no acute, displaced rib fractures.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pleural effusion or pneumothora. heart size is at the cardimediastinal silhouette is normal.
the cardiomediastinal silhouette is within normal limits for size and contour. the lungs are normally inflated with no focal airspace consolidation, pleural effusion, or pneumothora. degenerative changes in the spine.
the cardiomediastinal silhouette is within normal limits for appearance. the lung volumes are present. the bilateral lung volumes are present. the bilateral and patchy opacities are demonstrated in the bilateral lower lobes. no pneumothora.
the heart is normal in size. the mediastinum is unremarkable. there is no pleural effusion, pneumothora, or focal airspace disease. calcific granulomas are present in the right lower lobe. calcified lymph are present in the right lung.
the lungs are clear. there is no focal airspace consolidation. no pleural effusion or pneumothora. heart size is at the upper limits of normal. the are unremarkable.
there is a calcified granuloma in the left midlung. the heart size is normal. no pneumothora.
the heart is top normal in size. the mediastinum is unremarkable. there is no pleural effusion, pneumothora, or focal airspace disease. calcific granulomas are present in the right lower lobe. calcified lymph are noted in the right lung.
the lungs are clear. there is no focal airspace consolidation. no pleural effusion or pneumothora. heart size is at the upper limits of normal. degenerative changes are present in the spine.
there is a calcified granuloma in the left midlung. no focal infiltrate. no pleural effusion or pneumothora. heart size is within normal limits. there is no acute bony abnormality identified.
the cardiomediastinal silhouette is within normal limits for appearance. the lung volumes are demonstrated in the bilateral and lung volumes with bronchovascular crowding. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion.
Name: prediction_gs, dtype: float64
```

Our model is predicting sensible captions and which are meaningful and bias towards non-disease captions are reduced to the good extent, as the data we have is very less for applying a deep learning model we are not able to produce exceptional results. We can implement transformer-based image captioning models to get better results but as the data we have is less no model perform better.

Step 5: Discussion of Results

Sl No.	Model	BLEU-1	BLEU-2	BLEU-3	BLEU-4
1.	Attention Model (greedy search)	0.283326	0.166389	0.099237	0.053017
2.	Simple Encoder Decoder (greedy search)	0.261584	0.142195	0.086271	0.040068

These are the best models of each approach in terms of bleu score. Here the list is sorted based on **cumulative BLEU-4** score. The best model we obtained was Attention Model (greedy search). This model performed far better than other model since it was able to output names of diseases and some observation more correctly than others.

Other model only outputted the caption assuming all the datapoints shown were of "no disease category and was only able to output very few varieties of captions. The final model's grammar was good compared to simple encoder-decoder models grammar but not that great suggesting underfitting. More data with much more variability specifically X-rays with diseases can be much more helpful for the models to understand better and can help reduce bias towards "no disease category.

5. Conclusion:

We are successfully able to generate captions (findings) for the X-Ray images and able to achieve BLEU score of around 28.3% with the Global Attention Based Encoder-Decoder models with GRUs. As the data we have is very less and biased towards the non-diseased data we are not able to get exceptionally well BLEU score but if we have good volume of balanced data to work with then same piece of code can do exceptionally well in predicting captions for the images.

6. References:

- Submitted on 10 Feb 2015 (v1), last revised 19 Apr 2016 (this version, v3)] Show, Attend and Tell: Neural Image Caption Generation with Visual Attention Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov , Richard Zemel, Yoshua Bengio
- December 1997 Neural Computation 9(8):1735-80 Long Short-term Memory by Sepp Hochreiter Johannes Kepler University Linz
- [Submitted on 3 Mar 2022] A Deep Neural Framework for Image Caption Generation Using GRU-Based Attention Mechanism Rashid Khan, M Shujah Islam, Khadija Kanwal, Mansoor Iqbal, Md. Imran Hossain, Zhongfu Ye
- [Submitted on 26 Feb 2021] Learning Transferable Visual Models from Natural Language Supervision Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, Ilya Sutskever.
- CLIP: https://openai.com/blog/clip/
- Attention: https://lilianweng.github.io/posts/2018-06-24-attention/ https://github.com/uzaymacar/attention-mechanisms#local-attention
- [Submitted on 14 Nov 2017 (v1), last revised 25 Dec 2017] CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning Pranav Rajpurkar , Jeremy Irvin, Kaylie Zhu , Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding , Aarti Bagul , Curtis Langlotz, Katie Shpanskaya, Matthew P. Lungren, Andrew Y. Ng
- GloVe: Global Vectors for Word Representation Jeffrey Pennington, Richard Socher, Christopher D. Manning Computer Science Department, Stanford University

7. Productionization and Deployment

For Production we will be using the Global Attention Encoder-Decoder Model as it is the best model as per the BLEU score and for Productionization we will convert our code from ".ipynb" colab files to production ready ".py" files.

We divided our code into two files for Productionization one is main file "final.py" and other is "create_model.py". let's understand what these each files contains

Final.py:

This file contains streamlit code which we are using to create the front end for the user using streamlit we are first placing a background, and giving few titles of the project. And we are calling the create_model.py file to load the weights which we saved during the model training and return the tokenizer which we created during the model building. We have two implementations in streamlit using two columns

1. Predict on uploaded files
2. Predict on sample data

```
import streamlit as st
import os
import numpy as np
import time
from PIL import Image
import create_model as cm
import base64

def add_bg_from_local(image_file):
    with open(image_file, "rb") as image_file:
        encoded_string = base64.b64encode(image_file.read())
    st.markdown(
    f"""
    <style>
    .stApp {{
        background-image: url({encoded_string.decode()});
        background-size: cover
    }}
    </style>
    """,
    unsafe_allow_html=True
    )
add_bg_from_local('./Background/Background.png')

original_title = '<p style="font-family:Franklin Gothic Medium;text-align:left;color:Yellow; font-size: 70px;">Chest X-ray Captioning</p>'
st.markdown(original_title, unsafe_allow_html=True)
Author_Name = '<p style="font-family:Franklin Gothic Medium;text-align:left;color:Magenta;font-weight: bold; font-size: 25px;">by Santhosh Kurnapally</p>'
st.markdown(Author_Name,unsafe_allow_html=True)
st.markdown("[<medium>Find Code here (Github)</medium>](https://github.com/skurnapally/Medical_Image_Captioning_on_Chest_X-Rays)",
unsafe_allow_html=True)
text = '<p style="font-family:Arial;text-align:left;color:White; font-size: 20px;">\nThis app will generate Findings from the X-ray report.\nYou can upload 2 X-rays that are either front or lateral view of chest of the same individual.</p>'
st.markdown(text,unsafe_allow_html=True)
note = '<p style="font-family:Arial;text-align:left;color:Snow; font-size: 20px;">The 2nd X-ray is optional.</p>'
st.markdown(note,unsafe_allow_html=True)
col1,col2 = st.beta_columns(2)
note1 = '<p style="font-family:Arial;text-align:left;color:Snow; font-size: 20px;">Upload First X-Ray</p>'
col1.markdown(note1,unsafe_allow_html=True)
image_1 = col1.file_uploader("X-ray 1",type=['png','jpg','jpeg'])
image_2 = None
if image_1:
    note2 = '<p style="font-family:Arial;text-align:left;color:Snow; font-size: 20px;">Upload Second X-Ray (Optional)</p>'
    col2.markdown(note2,unsafe_allow_html=True)
    image_2 = col2.file_uploader("X-ray 2 (optional)",type=['png','jpg','jpeg'])
col1,col2 = st.beta_columns(2)
predict_button = col1.button('Predict on uploaded files')
test_data = col2.button('Predict on sample data')
```

1. Predict on Uploaded samples:

Using this option we can predict findings for any uploaded Chest X-Ray, here we can give two chest X-Rays frontal or lateral of the same patient and second X-Ray is optional, as discussed in the code section if second X-Ray is not provided first X-Ray is used as second X-Ray as well for finding prediction. For this prediction on uploaded X-Rays, we used predict() method.

```
@st.cache
def create_model():
    model_tokenizer = cm.create_model()
    return model_tokenizer


def predict(image_1,image_2,model_tokenizer,predict_button = predict_button):
    start = time.process_time()
    if predict_button:
        if (image_1 is not None):
            start = time.process_time()
            image_1 = Image.open(image_1).convert("RGB") #converting to 3 channels
            image_1 = np.array(image_1)/255
            if image_2 is None:
                image_2 = image_1
            else:
                image_2 = Image.open(image_2).convert("RGB") #converting to 3 channels
                image_2 = np.array(image_2)/255
            st.image([image_1,image_2],width=300)
            caption = cm.function1([image_1],[image_2],model_tokenizer)
            findings_text = '<p style="font-family:Arial;text-align:left;color:red; font-size: 20px;">Findings</p>'
            st.markdown(findings_text,unsafe_allow_html=True)
            findings = '<p style="font-family:Arial;text-align:left;color:white; font-size: 20px;">'+caption[0]+'</p>'
            st.markdown(findings,unsafe_allow_html=True)
            timetook = '<p style="font-family:Arial;text-align:left;color:white; font-size: 20px;">'+"Time Took for prediction: "+str(round(time.process_time()-start,2)) + ' seconds</p>'
            st.markdown(timetook,unsafe_allow_html=True)
            del image_1,image_2
        else:
            st.markdown("## Upload an Image")
```

2. Predict on Sample Data:

Using this option user can get findings of the sample data which we already provided while productionizing the code, as for user it is not always possible to get the X-Rays to check the functionality immediately, we provided few samples which generate randomly to showcase the power of our model. And we used predict_sample() method for this purpose.

```
def predict_sample(model_tokenizer,folder = './test_images'):
    no_files = len(os.listdir(folder))
    file = np.random.randint(1,no_files)
    file_path = os.path.join(folder,str(file))
    if len(os.listdir(file_path))==2:
        image_1 = os.path.join(file_path,os.listdir(file_path)[0])
        image_2 = os.path.join(file_path,os.listdir(file_path)[1])
        print(file_path)
    else:
        image_1 = os.path.join(file_path,os.listdir(file_path)[0])
        image_2 = image_1
    predict(image_1,image_2,model_tokenizer,True)


model_tokenizer = create_model()



if test_data:
    predict_sample(model_tokenizer)
else:
    predict(image_1,image_2,model_tokenizer)
```

create_model.py:

In this python file we have our main Global Attention Encoder – Decoder model , and implementation of all the sections which we discussed in the Attention model section is converted into each method and once final.py file calls create_model.py file , this file loads the model which we trained and make entire code ready for the using the model on user uploaded X-Rays or sample X-Rays.

Additional Files for Productionization of model:

1. brucechou1983_CheXNet_Keras_0.3.0_weights.h5
    - This is the CheXNet pretrained weights file for transfer learning to get image embeddings.

2. Encoder_Decoder_global_attention.h5
    - This is the trained global attention model weights which will be loaded to predict the output of the user given or sample X-Rays.
3. Requirements.txt
    - This file contains requirements for deployment of this model on streamlit server.
4. test_images folder
    - This folder is for the test images which we are providing for the test samples.
5. Background folder
    - This folder contains background image of the streamlit deployment which we used as front end for user.
6. tokenizer.pickle
    - This is tokenizer which we saved using pickle library and is used for the caption prediction by over loaded model.

Deployment Of Model:

We have multiple options to deploy the model and out of them I am listing down the few options which are feasible

1. Deploy on Streamlit Cloud
2. Deploy on Heroku
3. Deploy on Azure/AWS/GCP virtual machines
4. Dockerize and deploy on any cloud container deployment services like (AWS Fargate or Google Cloud Run or Azure Container Instances

For this Thesis Productionization we are going with streamlit cloud as it is free, handles platform dependencies directly, no need of dockerization and pushing to any container registry to deploy and on serverless or cloud VMs.

Deploying on Streamlit Cloud:

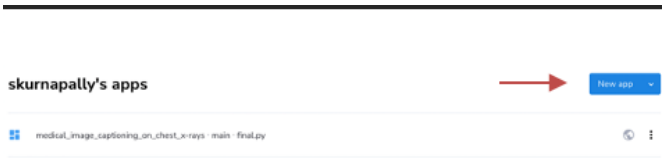
Reference: https://docs.streamlit.io/streamlit-cloud/get-started/deploy-an-app

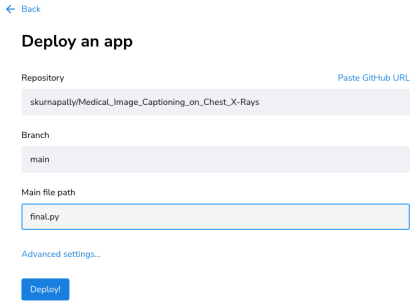
Step1: signup/login streamlit

- Create Account or login if we already have an account to streamlit.
- https://streamlit.io/

Step2: Create New app

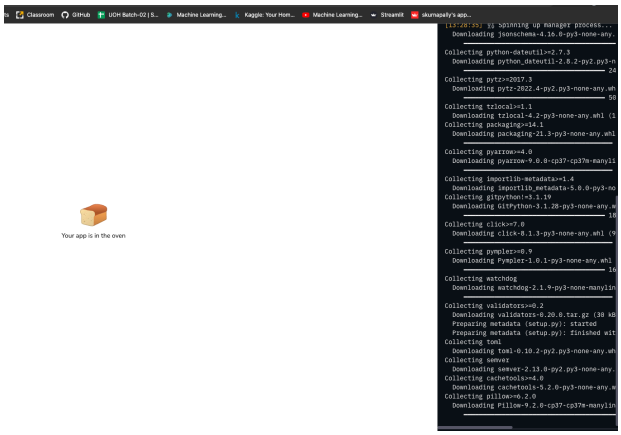
- Click on create new app.





Step3: deploy from GitHub Repository.

- Provide the details of git hub where all the above discussed Productionization files are checked in and Main file name will be the file where our streamlit code is present, in our case it is final.py file and click on deploy. Will deploy the model on the streamlit cloud with all the dependencies from the requirement.txt file which we already added to GitHub.

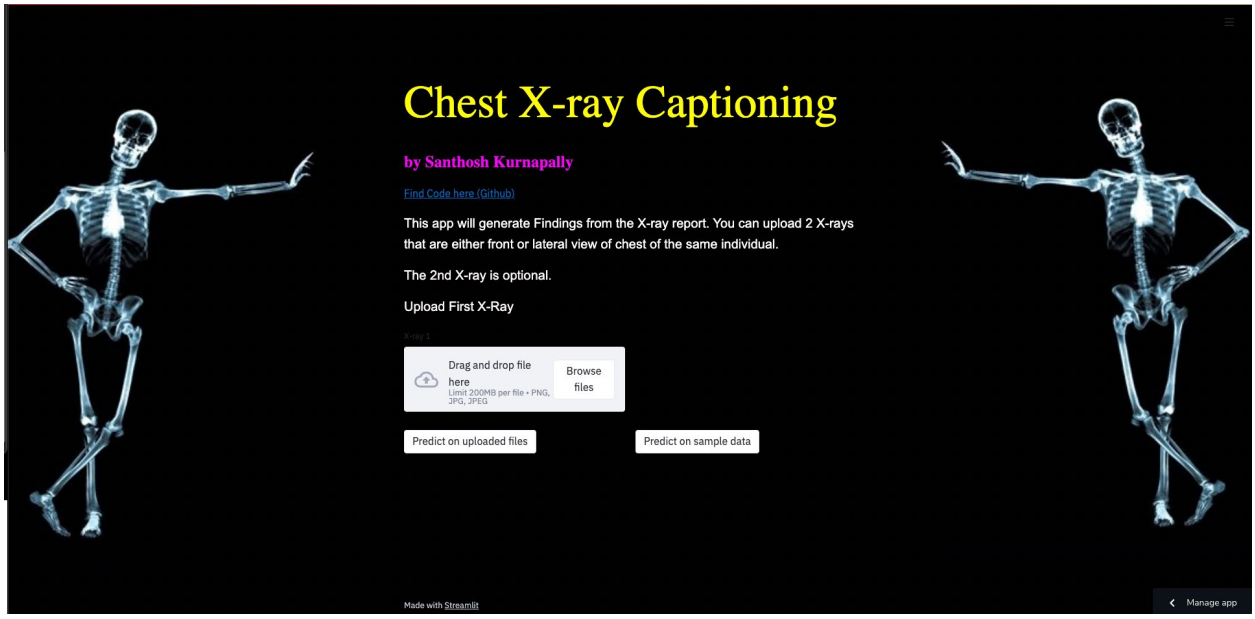


Step 4: Run the Deployed Model

Deployed Model URL: https://skurnapally-medical-image-captioning-on-chest-x-ra-final-zaj8kf.streamlitapp.com/

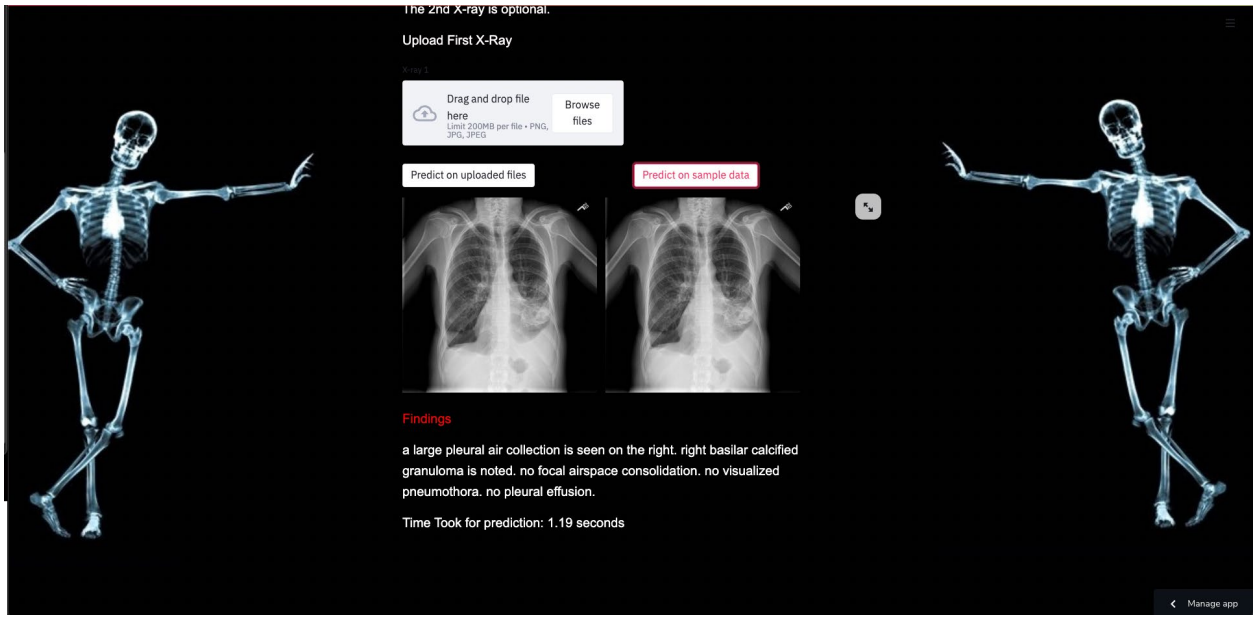
Once the model is deployed, we can click on the deployed model which will automatically takes us to the deployed server and code present in the main file will be automatically executed.

Below is the deployed model of Global Attention Encoder – Decoder model and it is ready for predication as mentioned above we can use sample X-Rays for prediction or we can give our own chest X-Rays for extracting the captions (findings) from the given X-Rays.



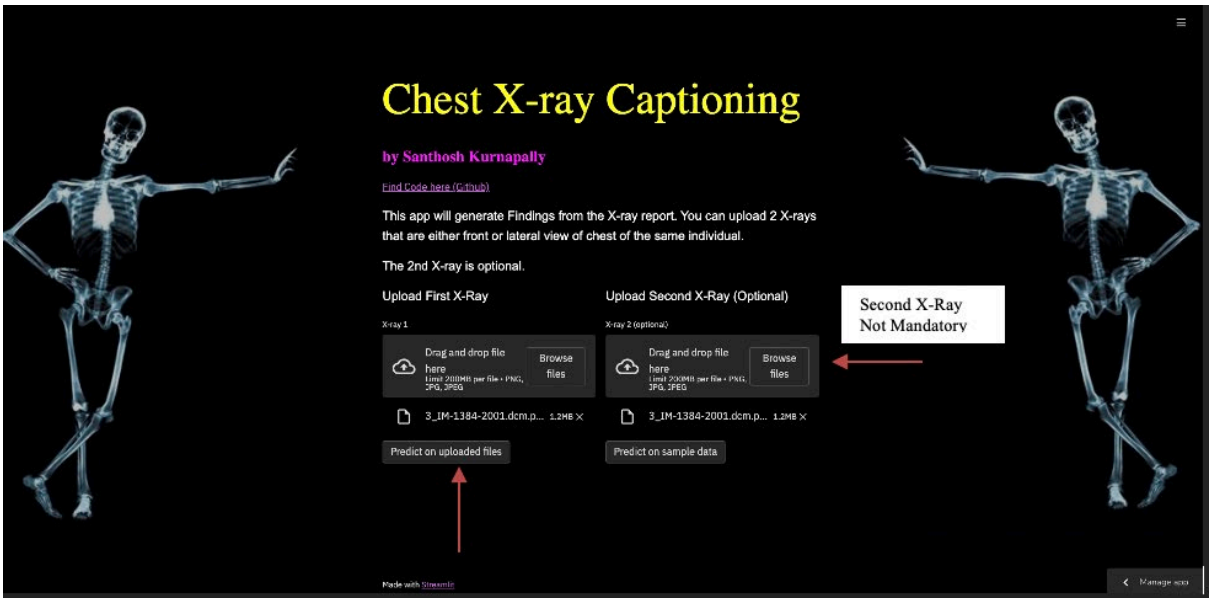
Predict on sample data:

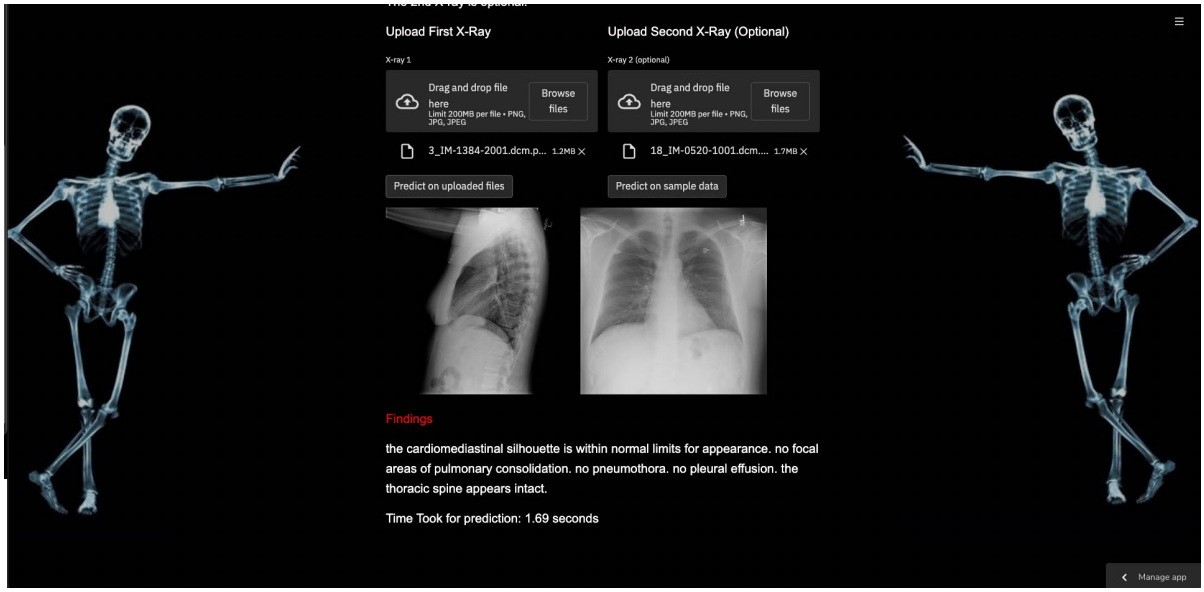
If we click on predict on sample data, random sample images which we already saved in "test images" folder are picked by code and caption is generated on them as below.



Predict on uploaded files:

If we upload files and click on predict on uploaded files, our model predicts the caption from the uploaded files. And once we upload one X-Ray it asks for other X-ray and second x-ray is not mandatory for caption generation. And we are giving time taken for prediction as well to show the latency of the model.


Chest X-ray Captioning
by Santhosh Kurnapally
Find Code here (Github)
This app will generate Findings from the X-ray report. You can upload 2 X-rays that are either front or lateral view of chest of the same individual.
The 2nd X-ray is optional.
Upload First X-Ray
Upload Second X-Ray (Optional)
X-ray 1
X-ray 2 (optional)
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
3_IM-1384-2001.dcm.p... 1.2MB
Predict on uploaded files
Predict on sample data
Second X-Ray Not Mandatory
Made with Streamlit
Manage app


Upload First X-Ray
Upload Second X-Ray (Optional)
X-ray 1
X-ray 2 (optional)
Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG
Browse files
3_IM-1384-2001.dcm.p... 1.2MB
18_IM-0520-1001.dcm.... 1.7MB
Predict on uploaded files
Predict on sample data
Findings
the cardiomediastinal silhouette is within normal limits for appearance. no focal areas of pulmonary consolidation. no pneumothora. no pleural effusion. the thoracic spine appears intact.
Time Took for prediction: 1.69 seconds
Manage app