A Major Project Report on

BOOT STRAPPING LANGUAGE IMAGE PRE-TRAINED

Submitted to

JAWAHARLAL NEHRU TECNOLOGICAL UNIVERSITY, HYDERABAD

In partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING(AI&ML)

Submitted by

EDLA YESHWANTH	**20UK1A6649**
SAMALA HARSHITH KUMAR	**20UK1A6624**
GANGULA RISHITHA	**20UK1A6651**
JAKKULA SAI KUMAR	**20UK1A6608**

Under the guidance of

Mr. T. Sanath Kumar

Assistant Professor

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING(AI&ML)

VAAGDEVI ENGINEERING COLLEGE

Affiliated to JNTUH, HYDERABAD

BOLLIKUNTA, WARANGAL (T.S) – 506005

2020-2024

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING(AI&ML)

VAAGDEVI ENGINEERING COLLEGE(WARANGAL)

CERTIFICATE OF COMPLETION
MAJOR PROJECT

This is to certify that the Major Project entitled "**BOOT STRAPPING LANGUAGE IMAGE PRE-TRAINED**" is being submitted by **EDLA YESHWANTH(20UK1A6649),SAMALA HARSHITH KUMAR (20UK1A6624), GANGULA RISHITHA (20UK1A6651),JAKKULA SAI KUMAR (20UK1A6608)** in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering to Jawaharlal Nehru Technological University Hyderabad during the academic year 2023- 2024.

Project Guide
Mr. T. Sanath kumar
(Assistant Professor)

HOD
Dr. K. Sharmila Reddy
(Professor)

External

ACKNOWLEDGEMENT

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved **Dr. P. PRASAD RAO,** Principal, Vaagdevi Engineering College for making us available all the required assistance and for his support and inspiration to carry out this UG Project Phase-1 in the institute.

We extend our heartfelt thanks to **Dr. K. SHARMILA REDDY**, Head of the Department of CSE(AI&ML), Vaagdevi Engineering College for providing us necessary infrastructure and thereby giving us freedom to carry out the UG Project Phase-1.

We express heartfelt thanks to Smart Bridge Educational Services Private Limited, for their constant supervision as well as for providing necessary information regarding the UG Project Phase-1 and for their support in completing the UG Project Phase-1.

We express heartfelt thanks to the guide, **T.SANATH KUMAR ,** Assistant professor, Department of CSE(AI&ML) for his constant support and giving necessary guidance for completion of this UG Project Phase-1.

Finally, we express our sincere thanks and gratitude to my family members, friends for their encouragement and outpouring their knowledge and experience throughout the thesis.

EDLA YESHWANTH (20UK1A6649)
SAMALA HARSHITH KUMAR (20UK1A6624)
GANGULA RISHITHA (20UK1A6651)
JAKKULA SAI KUMAR (20UK1A6608)

TABLE OF CONTENTS: -

1. ABSTRACT

- The convergence of artificial intelligence, computer vision, and healthcare has propelled the field of medical image analysis to new heights, with a particular focus on the revolutionary capabilities of medical Visual Question Answering (VQA) models.
- This abstract encapsulates the essence of the topic by highlighting the recent advancements exemplified by BLIP (Bootstrapping Language-Image Pre-training), a paradigmatic Vision-Language Pre-training (VLP) framework proposed by Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi.
- BLIP stands out as an exemplary model, adept at seamlessly navigating a spectrum of multi-modal tasks, including Visual Question Answering, Image-Text retrieval, and Image Captioning.
- Its unique contribution lies in overcoming the historical trade-off between understanding-based and generation-based tasks. This is achieved through a sophisticated approach to harnessing noisy web data, characterized by the bootstrapping of captions.
- Synthetic captions, generated by a dedicated captioner, undergo filtration to discern and discard noise, resulting in a refined and effective source of supervision.
- The implications of successful deployment of medical VQA models, as illustrated by BLIP, are transformative for healthcare professionals, researchers, and patients.
- By enhancing clinical decision support, mitigating interpretation errors, and expediting diagnostic processes, these models redefine the efficiency and accuracy of medical image analysis. Moreover, their accessibility empowers a broader audience, including patients and non-specialist healthcare providers, contributing to improved health literacy.

- As we explore the intersection of technology and healthcare, the abstract underscores the broader applications of models like BLIP. Beyond clinical settings, they serve as educational tools, supporting the learning journey of medical professionals, and as invaluable assets for researchers delving into the complexities of large medical image datasets.
- However, the deployment of such models necessitates a nuanced consideration of ethical dimensions, emphasizing the critical need for privacy, security, and fairness in handling sensitive patient data.
- Collaboration between machine learning experts, healthcare professionals, and ethicists is paramount to ensure responsible and effective utilization of this transformative technology.
- This abstract sets the stage for an in-depth exploration of key conclusions and takeaways from the development and deployment of successful medical VQA models, drawing inspiration from the achievements and insights offered by the BLIP model.

INTRODUCTION

OVERVIEW:

- The convergence of artificial intelligence, computer vision, and healthcare has spurred remarkable advancements in medical image analysis. A notable milestone in this arena is the emergence of medical Visual Question Answering (VQA) models, exemplified by innovations such as BLIP (Bootstrapping Language-Image Pre-training).

PURPOSE:

- BLIP, conceived by Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi, embodies a fusion of computer vision and natural language processing tailored specifically for the complexities of medical data.
- It serves as a paradigmatic example of a Vision-Language Pre-training (VLP) framework, showcasing versatility across various multi-modal tasks like Visual Question Answering, Image-Text retrieval, and Image Captioning.
- The core premise of BLIP lies in bridging the gap between understanding-based tasks and generation-based tasks, which sets it apart from previous models that excelled in one aspect over the other.
- Its innovative approach leverages noisy web data through a process of bootstrapping captions, generating synthetic captions and employing a filtering mechanism to discard irrelevant or noisy information.
- Successfully deploying medical VQA models, as demonstrated by BLIP, holds transformative implications for healthcare stakeholders. These models enhance clinical decision support, reduce interpretation errors, and expedite diagnostics and treatment planning.

- Moreover, they enhance accessibility, empowering patients and non-specialist healthcare providers with understandable information, thereby contributing to improved health literacy.
- BLIP's impact extends beyond healthcare, with applications in education for medical professionals' learning journeys and supporting research initiatives through effective analysis of large medical image datasets.
- However, the deployment of such models also raises ethical considerations, emphasizing the importance of privacy, security, and fairness in handling sensitive patient data. Collaboration between machine learning experts, healthcare professionals, and ethicists is crucial for responsible and effective utilization of this technology.

2. LITERATURE SURVEY

EXISTING SOLUTION:

- In the realm of healthcare and medical imaging, the application of the BLIP VQA model has ushered in notable advancements. This innovative model has been strategically employed to address critical challenges within the medical domain, showcasing its versatility and efficacy.
- One prominent application involves harnessing the power of BLIP VQA for medical visual question answering (VQA), a task that plays a pivotal role in enhancing clinical decision support and expediting diagnostic processes.
- Healthcare professionals can leverage the model's capabilities to pose questions about medical images, ranging from X-rays to MRIs and CT scans, and receive accurate and rapid responses. This capability significantly contributes to more informed and efficient clinical decision-making.
- Additionally, the model has demonstrated its prowess in the realm of image captioning within medical contexts. By generating descriptive and contextually relevant captions for medical images, BLIP VQA enhances the interpretability of complex imaging findings, offering valuable insights for both medical professionals and researchers.
- The successful deployment of BLIP VQA in the medical domain underscores its potential as a transformative solution, fostering advancements in healthcare and contributing to the intersection of artificial intelligence and medical imaging sciences.

DISADVANTAGES OF THE EXISTING SOLUTION:

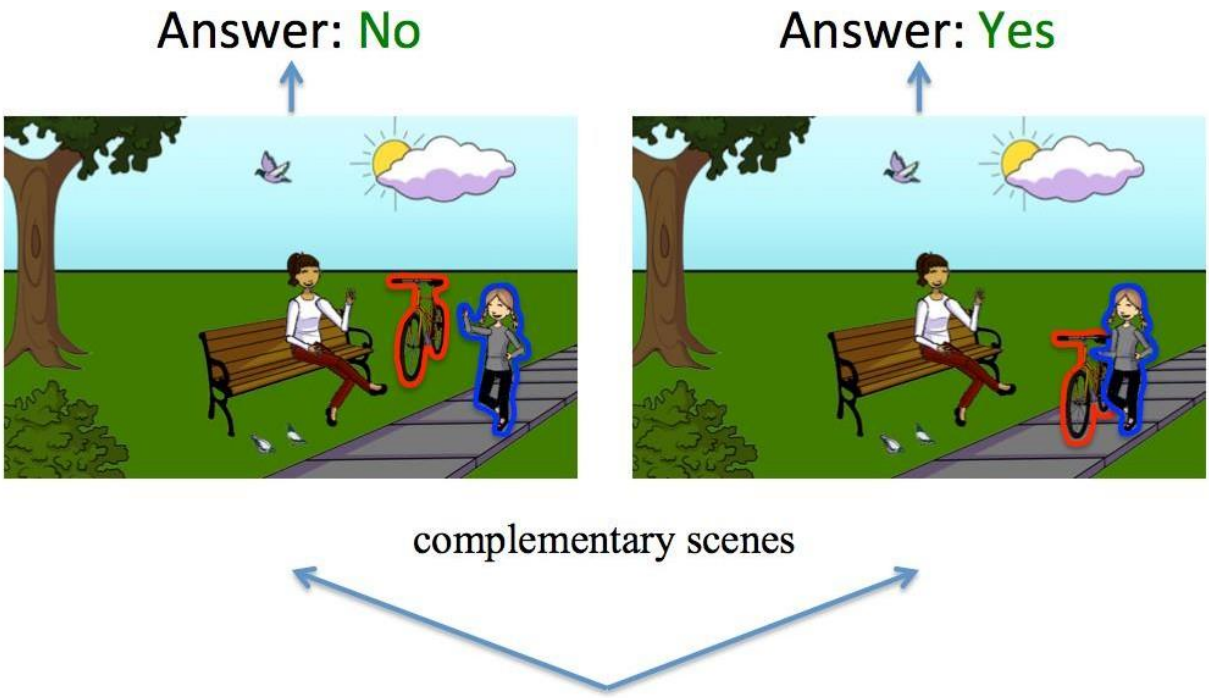
- **Limited Training Data Diversity:** Despite its effectiveness, the BLIP VQA model may suffer from limited diversity in training data, particularly concerning rare medical

conditions or imaging modalities. This limitation could result in reduced accuracy and reliability when addressing less common medical scenarios.

- **Interpretability Challenges:** While BLIP VQA excels in generating responses to visual questions, the model's decision-making process may lack transparency, hindering its interpretability. Healthcare professionals may find it challenging to trust and understand the reasoning behind the model's answers, potentially leading to skepticism and reluctance in adopting its recommendations.
- **Dependency on High-Quality Imaging Data:** The performance of BLIP VQA heavily relies on the quality and consistency of the input medical imaging data. Variations in image quality, artifacts, or inconsistencies in image acquisition protocols could negatively impact the model's ability to provide accurate responses, leading to potential errors or misinterpretations.
- **Ethical and Privacy Concerns:** As with any AI-driven solution in healthcare, the deployment of BLIP VQA raises ethical and privacy concerns regarding patient data protection and confidentiality. The model's access to sensitive medical images and associated information necessitates robust measures to ensure compliance with privacy regulations and prevent unauthorized access or misuse of patient data.
- **Generalization to Clinical Practice:** While BLIP VQA performs well in controlled experimental settings, its generalization to real-world clinical practice environments may pose challenges. Factors such as patient variability, environmental noise, and unexpected clinical scenarios could affect the model's performance, requiring ongoing validation and adaptation to ensure its relevance and reliability in diverse healthcare settings.

PROPOSED SOLUTION:

- The successful training and implementation of a medical Visual Question Answering (VQA) model represent a groundbreaking development with the potential to revolutionize healthcare practices.
- By seamlessly combining computer vision and natural language processing, this model has the capacity to significantly elevate diagnostic accuracy in medical settings. Its ability to swiftly process and analyze medical images in response to user queries can lead to faster and more precise diagnoses, thereby improving overall efficiency in healthcare delivery.
- Furthermore, the model's deployment offers the promise of expanding access to valuable medical information, empowering not only healthcare professionals but also patients and non-specialist providers.
- Despite these promising prospects, the integration of a medical VQA model necessitates a meticulous approach to ethical considerations. Striking the right balance between innovation and safeguarding patient privacy, addressing potential biases, and adhering to regulatory standards is imperative.
- A robust framework ensuring responsible use must be established to uphold the integrity and trustworthiness of the technology in clinical practice. This holistic approach acknowledges that the transformative power of medical VQA models can only be fully realized when accompanied by a steadfast commitment to ethical, privacy, and regulatory principles.



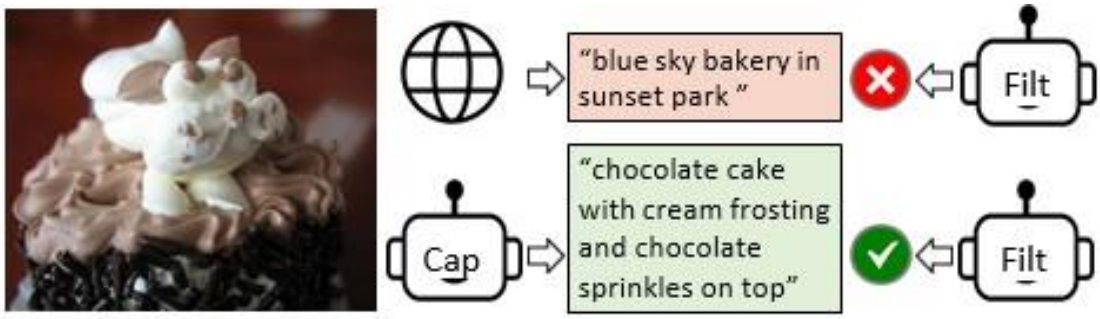


Figure 1. We use a Captioner (Cap) to generate synthetic captions for web images, and a Filter (Filt) to remove noisy captions.

3. THEORITICAL ANALYSIS

3.1 MODEL ARCHITECTURE

1. **Model Loading:**
    - The from_pretrained method is used to load a pre-trained BLIP model for Visual Question Answering. The model is loaded from the "Salesforce/blip-vqa-base" checkpoint. This indicates that the model has been pre-trained on a large dataset and fine-tuned specifically for VQA tasks.
2. **Model Selection:**
    - The choice of the BLIP model indicates the utilization of the Bootstrapping Language-Image Pre-training (BLIP) framework. BLIP is designed to flexibly transfer knowledge to both vision-language understanding and generation tasks, making it suitable for a variety of multi-modal applications, including VQA.
3. **Device Placement:**
    - The to(device) method is used to move the model to a specific device (e.g., GPU or CPU) for computation. This is a common step to leverage hardware acceleration, especially for large models like BLIP that involve significant computational resources.

While the provided code initializes the BLIP model for VQA, it doesn't reveal the intricate details of the BLIP algorithm itself. To gain a deeper understanding of the algorithm, its architecture, and the specific pre-training and fine-tuning processes involved, you would need to refer to the original BLIP paper or the documentation provided by Salesforce for their implementation.

Consider exploring the official Salesforce BLIP repository or documentation for detailed information on the model's architecture, training procedure, and any additional components specific to BLIP. This will provide you with a comprehensive understanding of how the model processes visual and textual information to perform VQA tasks.

3.2 TRAINING A ALGORITHM:

1. **Model Training Setup:**
    - The model.train() method indicates that the model is set to training mode. This ensures that parameters are updated during the training process. In contrast, during evaluation, it's common to use model.eval() to set the model to evaluation mode, where parameters are fixed.
2. **Epoch Iteration:**
    - The outer loop iterates over epochs, with a total of 90 epochs specified. An epoch represents a complete pass through the entire training dataset.
3. **Batch Iteration:**
    - The inner loop iterates over batches of data from the training dataloader (train_dataloader). Each batch typically consists of input data (e.g., images and questions) and their corresponding labels (e.g., answers).
4. **Data Transfer to Device:**
    - The line batch = {k:v.to(device) for k,v in batch.items()} ensures that the batch of data is transferred to the specified device (e.g., GPU or CPU).
5. **Zeroing Gradients:**
    - optimizer.zero_grad() initializes the gradients of the model parameters to zero before the backward pass.
6. **Forward Pass:**

- outputs = model(**batch) performs a forward pass through the model using the input batch, obtaining the model's predictions.

7. Loss Computation:

- loss = outputs.loss retrieves the loss from the model's output. This loss is typically a measure of the difference between the model's predictions and the ground truth.

8. Backward Pass and Optimization:

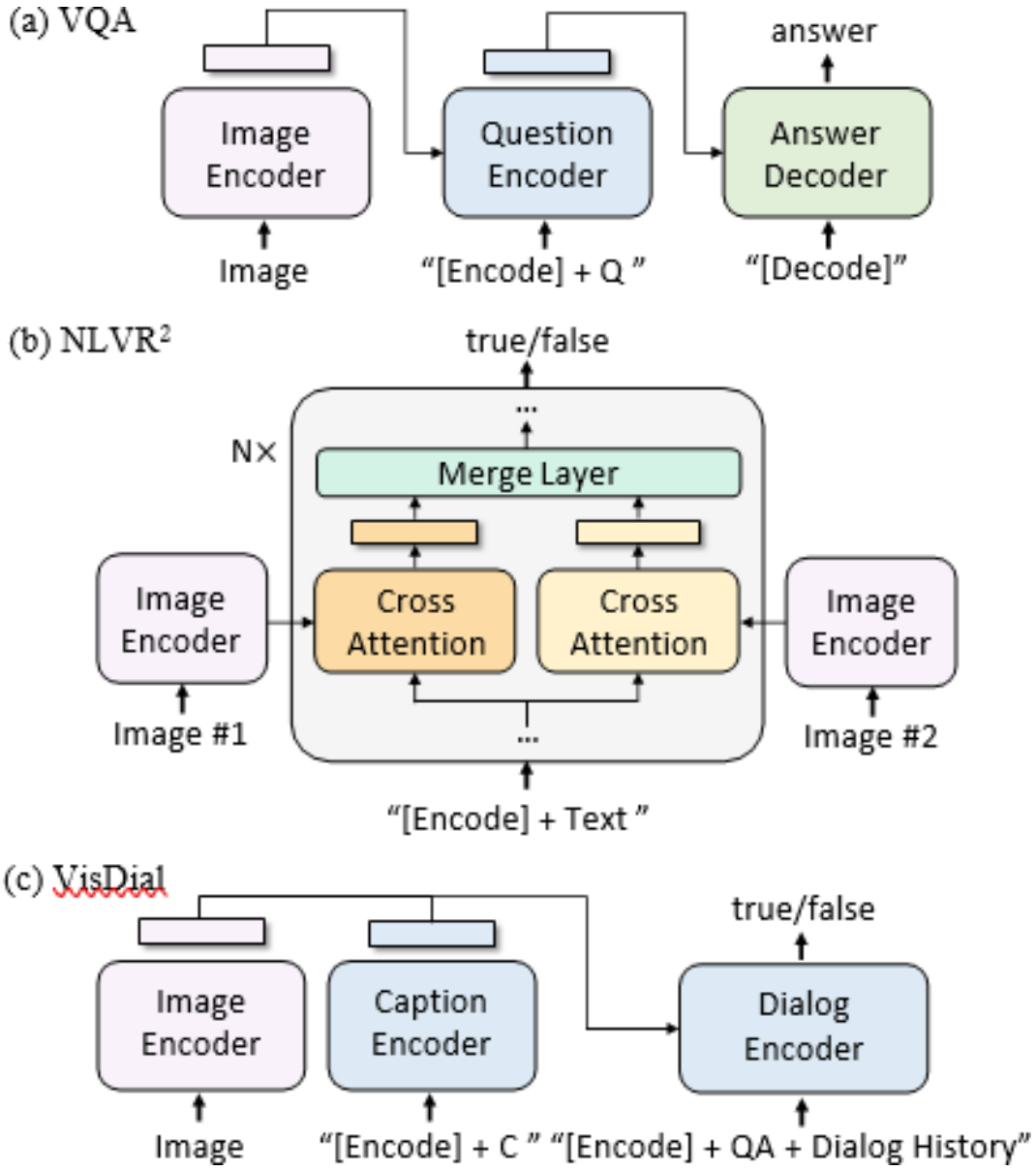
- loss.backward() computes the gradients of the model parameters with respect to the loss.
- optimizer.step() updates the model parameters based on the computed gradients using the specified optimization algorithm (e.g., SGD, Adam).

9. Monitoring Loss:

- The total loss for each batch is recorded and printed after each epoch (print("Loss:", sum(total_loss))).

This training loop is a standard procedure for fine-tuning a neural network on a specific task. It involves iterating over multiple epochs, processing data in batches, and updating the model parameters to minimize the specified loss function. The choice of 90 epochs, the optimizer used, and other hyperparameters would depend on the specifics of the task and dataset.

3.3 BLOCK DIAGRAM :



5. EXPERIMENTAL INVESTIGATION

We propose BLIP, a unified VLP framework to learn from noisy image-text pairs. This section first introduces our new model architecture MED and its pre-training objectives, and then delineates CapFilt for dataset bootstrapping.

MODEL ARCHITECTURE:

We employ a visual transformer (Dosovitskiy et al., 2021) as our image encoder, which divides an input image into patches and encodes them as a sequence of embeddings, with an additional [CLS] token to represent the global im- age feature. Compared to using pre-trained object detectors for visual feature extraction (Chen et al., 2020), using a ViT is more computation-friendly and has been adopted by the more recent methods (Li et al., 2021a; Kim et al., 2021).

In order to pre-train a unified model with both understanding and generation capabilities, we propose multimodal mixture of encoder-decoder (MED), a multi-task model which can operate in one of the three functionalities:

(1) **Unimodal encoder**, which separately encodes image and text. The text encoder is the same as BERT (Devlin et al., 2019), where a [CLS] token is appended to the beginning of the text input to summarize the sentence.

(2) **Image-grounded text encoder**, which injects visual information by inserting one additional cross-attention (CA) layer between the self-attention (SA) layer and the feed forward network (FFN) for each transformer block of the text encoder. A task-specific [Encode] token is appended to the text, and the output embedding of [Encode] is used as the multimodal representation of the image-text pair.

(3) **Image-grounded text decoder**, which replaces the bi- directional self-attention layers in the image-grounded text encoder with causal self-attention layers. A [Decode]

token is used to signal the beginning of a sequence, and an end-of-sequence token is used to signal its end.

PRE-TRAINING OBJECTIVES:

We jointly optimize three objectives during pre-training, with two understanding-based objectives and one generation-based objective. Each image-text pair only requires one for- ward pass through the computational-heavier visual trans- former, and three forward passes through the text trans- former, where different functionalities are activated to compute the three losses as delineated below.

Image-Text Contrastive Loss (ITC) activates the unimodal encoder. It aims to align the feature space of the visual trans- former and the text transformer by encouraging positive image-text pairs to have similar representations in contrast to the negative pairs. It has been shown to be an effective objective for improving vision and language understanding (Radford et al., 2021; Li et al., 2021a). We follow the ITC loss by Li et al. (2021a), where a momentum encoder is introduced to produce features, and soft labels are created from the momentum encoder as training targets to account for the potential positives in the negative pairs.

Image-Text Matching Loss (ITM) activates the image- grounded text encoder. It aims to learn image-text multi- modal representation that captures the fine-grained alignment between vision and language. ITM is a binary classification task, where the model uses an ITM head (a linear layer) to predict whether an image-text pair is positive (matched) or negative (unmatched) given their multimodal feature. In order to find more informative negatives, we adopt the hard negative mining strategy by Li et al. (2021a), where negatives pairs with higher contrastive similarity in a batch are more likely to be selected to compute the loss.

Language Modeling Loss (LM) activates the image- grounded text decoder, which aims to generate textual descriptions given an image. It optimizes a cross entropy loss which trains the model to maximize the likelihood of the text in an autoregressive manner. We apply a label smoothing of 0.1 when computing the loss. Compared to the MLM loss that has been widely-used for VLP, LM enables the model with the generalization capability to convert visual information into coherent captions.

In order to perform efficient pre-training while leveraging multi-task learning, the text encoder and text decoder share all parameters except for the SA layers. The reason is that the differences between the encoding and decoding tasks are best captured by the SA layers. In particular, the encoder employs bi-directional self-attention to build representations for the current input tokens, while the decoder employs causal self-attention to predict next tokens. On the other hand, the embedding layers, CA layers and FFN function similarly between encoding and decoding tasks, therefore sharing these layers can improve training efficiency while benefiting from multi-task learning,

CAPFILT:

Due to the prohibitive annotation cost, there exist a limited number of high-quality human-annotated image-text pairs (Ih, Th) (e.g., COCO (Lin et al., 2014)). Recent work (Li et al., 2021a; Wang et al., 2021) utilizes a much larger number of image and alt-text pairs (Iw, Tw) that are automatically collected from the web. However, the alt-texts oftendo not accurately describe the visual content of the images, making them a noisy signal that is suboptimal for learning vision-language alignment.

We propose Captioning and Filtering (CapFilt), a new method to improve the quality of the text corpus. Figure 3 gives an illustration of CapFilt. It introduces two modules: a captioner to generate captions given web images, and a filter to remove noisy image-text pairs. Both the captioner and the filter are initialized from the same pre-trained MED

model, and finetuned individually on the COCO dataset. The finetuning is a lightweight procedure.

Specifically, the captioner is an image-grounded text de- coder. It is finetuned with the LM objective to decode texts given images. Given the web images Iw, the captioner generates synthetic captions Ts with one caption per image. The filter is an image-grounded text encoder. It is finetuned with the ITC and ITM objectives to learn whether a text matches an image. The filter removes noisy texts in both the original web texts Tw and the synthetic texts Ts, where a text is considered to be noisy if the ITM head predicts it as unmatched to the image. Finally, we combine the filtered image-text pairs with the human-annotated pairs to form a new dataset, which we use to pre-train a new model.

Diversity is Key for Synthetic Captions:

- Nucleus sampling employed to generate synthetic captions.
- Compared with beam search, nucleus sampling resulted in better performance.
- Nucleus sampling generates more diverse and surprising captions, leading to better model performance.

Parameter Sharing and Decoupling:

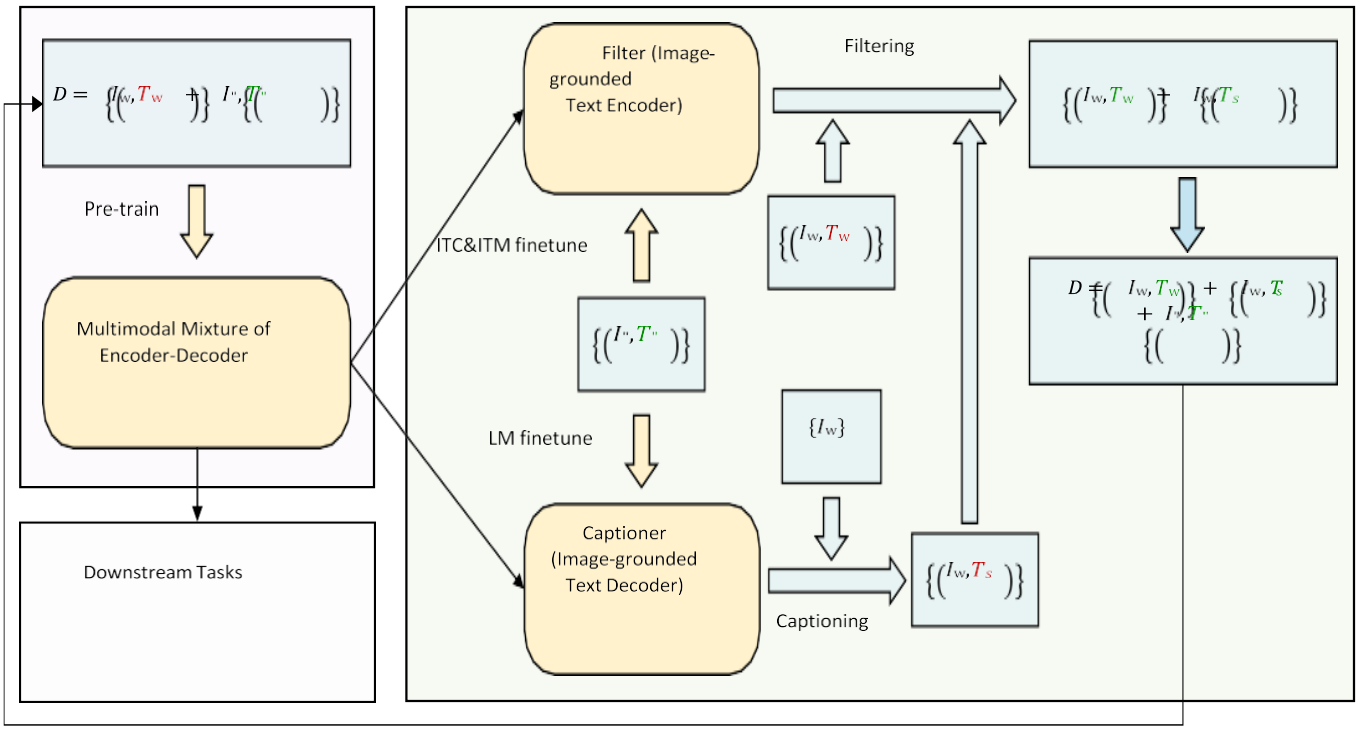
- Text encoder and decoder share all parameters except for the self-attention layers.
- Sharing most layers except self-attention improved performance and reduced model size.
- Captioner and filter end-to-end finetuned individually on COCO.
- Captioner and filter sharing parameters led to decreased performance due to confirmation bias.

Comparison with State-of-the-Arts:

- BLIP achieved substantial performance improvement compared to existing methods in various tasks like image-text retrieval, image captioning, VQA, NLVR2, VisDial, text-to-video retrieval, and video question answering.

- **Image-Text Retrieval**: Matching images to relevant textual descriptions or vice versa.
- **Image Captioning**: Generating descriptive textual captions for given images.
- **VQA (Visual Question Answering)**: Answering questions about images using both textual and visual information.
- **NLVR2 (Natural Language for Visual Reasoning 2)**: Determining the truth value of textual statements given corresponding images.
- **VisDial (Visual Dialogue)**: Engaging in a multi-turn conversation about an image.
- **Text-to-Video Retrieval**: Finding relevant videos based on textual queries.
- **Video Question Answering**: Answering questions about videos using both textual and visual information.

6. FLOW CHART



To model

To data

I_w: web images
I_h: human-annotated images

T_w: web texts
T_w: filtered web texts
T_s: synthetic texts
T_s: filtered synthetic texts
T_h: human-annotated texts

7. CODE SNIPPETS

BLIP Medical Visual-Major proje... Draft saved

File Edit View Run Add-ons Help

Share Save Version 0

Run All Markdown Draft Session off (run a cell to start)

Introduction

A successfully trained and inferred medical Visual Question Answering (VQA) model represents a significant advancement in the field of healthcare and medical image analysis. Such a model combines computer vision and natural language processing to provide valuable insights and answers to medical professionals, researchers, and patients.

BLIP Medical Visual-Major proje... Draft saved

File Edit View Run Add-ons Help

Share Save Version 0

Run All Markdown Draft Session off (run a cell to start)

BLIP

Overview

The BLIP model was proposed in BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation by Junnan Li, Dongxu Li, Caiming Xiong, Steven Hoi.

BLIP is a model that is able to perform various multi-modal tasks including

- Visual Question Answering
- Image-Text retrieval (Image-text matching)
- Image Captioning

The abstract from the paper is the following:

```
Vision-Language Pre-training (VLP) has advanced the performance for many vision-language tasks. However, most existing pre-trained models only excel in either understanding-based tasks or generation-based tasks. Furthermore, performance improvement has been largely achieved by scaling up the dataset with noisy image-text pairs collected from the web, which is a suboptimal source of supervision. In this paper, we propose BLIP, a new VLP framework which transfers flexibly to both vision-language understanding and generation tasks. BLIP effectively
```

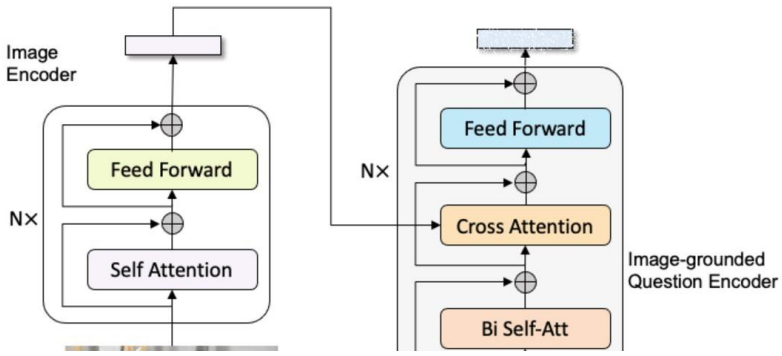
BLIP Medical Visual-Major proje... Draft saved

File Edit View Run Add-ons Help

Share Save Version 0

Run All Markdown Draft Session off (run a cell to start)

utilizes the noisy web data by bootstrapping the captions, where a captioner generates synthetic captions and a filter removes the noisy ones. We achieve state-of-the-art results on a wide range of vision-language tasks, such as image-text retrieval (+2.7% in average recall@1), image captioning (+2.8% in CIDEr), and VQA (+1.6% in VQA score). BLIP also demonstrates strong generalization ability when directly transferred to videolanguage tasks in a zero-shot manner. Code, models, and datasets are released.

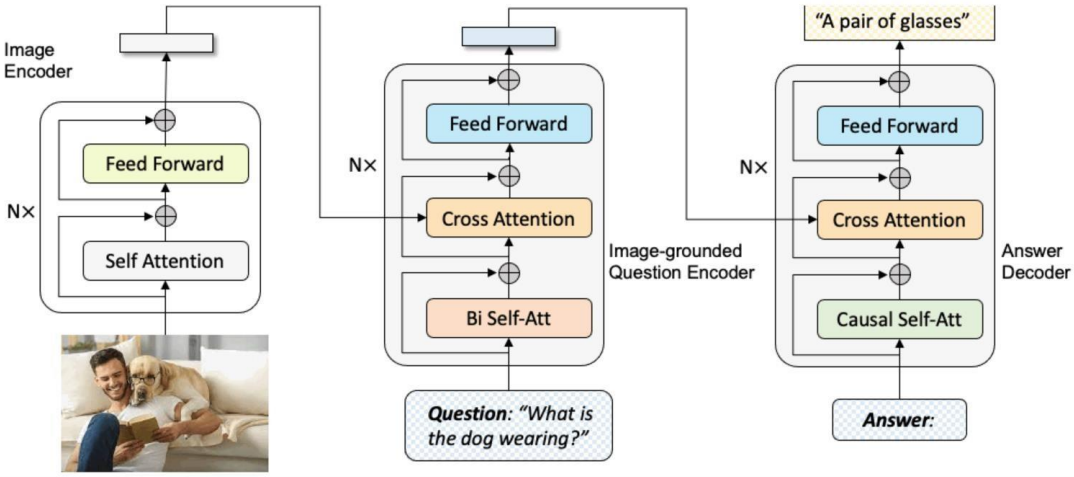


BLIP Medical Visual-Major proje... Draft saved

File Edit View Run Add-ons Help

Share Save Version 0

Run All Markdown Draft Session off (run a cell to start)



Requirement Installation

In [1]:
```
!pip install transformers[torch] datasets -q
```

Import Libraries

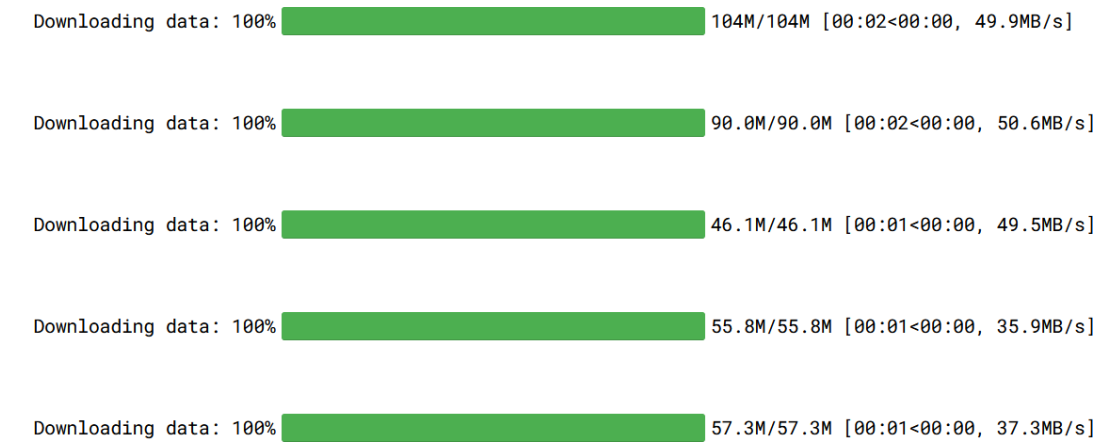
In [2]:
```
import requests
from PIL import Image
import torch
from transformers import BlipProcessor, BlipForQuestionAnswering,BlipImageProcessor, AutoProcessor
from transformers import BlipConfig
from datasets import load_dataset
from torch.utils.data import DataLoader
from tqdm.notebook import tqdm

import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
```

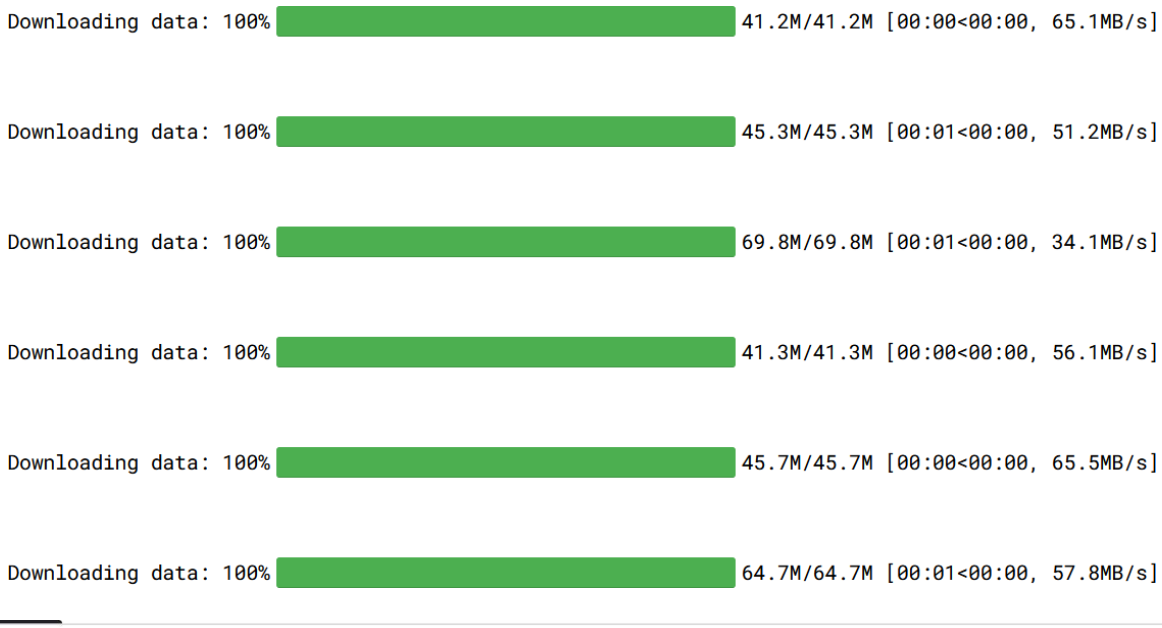
```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0
is required for this version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [3]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [4]:
```
dataset = load_dataset("flaviagiammarino/path-vqa")
```

```
Downloading and preparing dataset parquet/flaviagiammarino--path-vqa to /root/.cache/huggingface/datasets/parque
t/flaviagiammarino--path-vqa-77e24095dbbb9822/0.0.0/0b6d5799bb726b24ad7fc7be720c170d8e497f575d02d47537de9a5bac07
4901...

Downloading data files: 100% 3/3 [00:26<00:00, 7.63s/it]

Downloading data: 100% 42.8M/42.8M [00:01<00:00, 36.8MB/s]

Downloading data: 100% 81.0M/81.0M [00:02<00:00, 48.7MB/s]
```





Dataset parquet downloaded and prepared to /root/.cache/huggingface/datasets/parquet/flaviagiammarino--path-vqa-77e24095dbbb9822/0.0.0/0b6d5799bb726b24ad7fc7be720c170d8e497f575d02d47537de9a5bac074901. Subsequent calls will reuse this data.

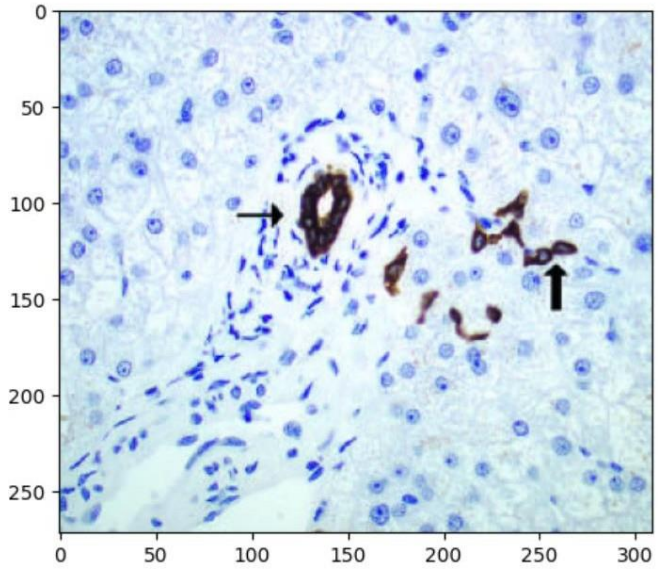
In [5]:
```
dataset
```

Out[5]:
```
DatasetDict({
    train: Dataset({
        features: ['image', 'question', 'answer'],
        num_rows: 19654
    })
    test: Dataset({
        features: ['image', 'question', 'answer'],
        num_rows: 6719
    })
    validation: Dataset({
        features: ['image', 'question', 'answer'],
        num_rows: 6259
    })
})
```

Sample Visualization ¶

In [6]:
```
sample = dataset['train'][1]
PIL_image = Image.fromarray(np.array(sample['image'])).convert('RGB')
plt.imshow(sample['image'].convert('RGB'))
print("Question: {}".format(sample['question']))
print("Answer: {}".format(sample['answer']))
```

```
Question: what are stained here with an immunohistochemical stain for cytokeratin 7?
Answer: bile duct cells and canals of hering
```



In [7]:
```
config = BlipConfig.from_pretrained("Salesforce/blip-vqa-base")
```

```
Downloading (…)lve/main/config.json: 100% 4.56k/4.56k [00:00<00:00, 117kB/s]
```

In [8]:
```
train_data = dataset['train'].select(range(10000))
val_data = dataset['validation'].select(range(1000))
```

Build Data-loader

In [9]:
```
class VQADataset(torch.utils.data.Dataset):
    def __init__(self, data, segment, text_processor, image_processor):
        self.data = data
        self.questions = data['question']
        self.answers = data['answer']
        self.text_processor = text_processor
        self.image_processor = image_processor
        self.max_length = 32
        self.image_height = 128
        self.image_width = 128

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        # get image + text
        answers = self.answers[idx]
        questions = self.questions[idx]
        image = self.data[idx]['image'].convert('RGB')
```

```
        text = self.questions[idx]

        image_encoding = self.image_processor(image,
                                do_resize=True,
                                size=(self.image_height,self.image_width),
                                return_tensors="pt")

        encoding = self.text_processor(
                                None,
                                text,
                                padding="max_length",
                                truncation=True,
                                max_length = self.max_length,
                                return_tensors="pt"
                                )
        # # remove batch dimension
        for k,v in encoding.items():
            encoding[k] = v.squeeze()
        encoding["pixel_values"] = image_encoding["pixel_values"][0]
        # # add labels
        labels = self.text_processor.tokenizer.encode(
            answers,
```

```
           max_length= self.max_length,
           padding="max_length",
           truncation=True,
           return_tensors='pt'
        )[0]
        encoding["labels"] = labels

        return encoding
```

```
text_processor = BlipProcessor.from_pretrained("Salesforce/blip-vqa-base")
image_processor = BlipImageProcessor.from_pretrained("Salesforce/blip-vqa-base")
```

```
Downloading (…)rocessor_config.json: 100% 445/445 [00:00<00:00, 19.7kB/s]

Downloading (…)okenizer_config.json: 100% 592/592 [00:00<00:00, 26.9kB/s]

Downloading (…)solve/main/vocab.txt: 100% 232k/232k [00:00<00:00, 3.29MB/s]
```

```
Downloading (…)/main/tokenizer.json: 100% 711k/711k [00:00<00:00, 9.91MB/s]

Downloading (…)cial_tokens_map.json: 100% 125/125 [00:00<00:00, 11.8kB/s]
```

In [11]:

```
train_vqa_dataset = VQADataset(data=train_data,
                    segment='train',
                    text_processor = text_processor,
                    image_processor = image_processor
                        )

val_vqa_dataset = VQADataset(data=train_data,
                    segment='validation',
                    text_processor = text_processor,
                    image_processor = image_processor
                        )
```

In [12]:

```
train_vqa_dataset[0]
```

Out[12]:

```
{'input_ids': tensor([  101,  2073,  2024, 11290,  7872,  4442,  1006,  9242,  4442,  1007,
         2284,  1029,   102,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0]), 'attention_mask': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0]), 'pixel_values': tensor([[[0.6895, 0.8647, 0.9814,  ..., 0.9668, 1.2150, 1.419
4],
         [1.0252, 0.9814, 1.0106,  ..., 1.0398, 1.3902, 1.2880],
         [0.9960, 1.0836, 0.8501,  ..., 1.0398, 1.2442, 1.1420],
         ...,
         [0.9668, 0.5581, 0.0617,  ..., 1.0106, 0.9376, 0.8792],
         [0.7917, 0.9960, 0.8647,  ..., 1.1420, 1.0252, 0.9230],
         [0.5581, 0.9522, 0.9376,  ..., 1.1420, 0.9376, 0.9668]],

        [[1.1594, 1.3995, 1.5196,  ..., 1.3695, 1.6247, 1.8948],
         [1.5946, 1.4746, 1.5046,  ..., 1.4446, 1.8047, 1.6847],
         [1.5196, 1.5946, 1.5196,  ..., 1.4746, 1.5646, 1.6247],
         ...,
```

```
          [1.5196, 1.2645, 0.7542,  ..., 1.5046, 1.4145, 1.3545],
          [1.2645, 1.5346, 1.4446,  ..., 1.6997, 1.5646, 1.5496],
          [1.0393, 1.5046, 1.5346,  ..., 1.7147, 1.3995, 1.5646]],

         [[1.9610, 1.9326, 1.8188,  ..., 1.8188, 1.9468, 2.0321],
          [1.8899, 1.9468, 1.8757,  ..., 1.8046, 1.9895, 1.9610],
          [1.9184, 1.9326, 1.7762,  ..., 1.8757, 1.9326, 1.7620],
          ...,
          [1.6055, 1.5487, 1.5487,  ..., 1.6766, 1.6340, 1.5913],
          [1.5771, 1.6624, 1.6198,  ..., 1.7193, 1.6340, 1.6340],
          [1.5344, 1.6198, 1.6624,  ..., 1.7051, 1.6482, 1.6198]]]), 'labels': tensor([  101,  1999,  1996, 17263
,  1997,  2014,  2075,   102,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0])}
```

In [13]:

```
def collate_fn(batch):
    input_ids = [item['input_ids'] for item in batch]
    pixel_values = [item['pixel_values'] for item in batch]
    attention_mask = [item['attention_mask'] for item in batch]
    labels = [item['labels'] for item in batch]
    # create new batch
    batch = {}
    batch['input_ids'] = torch.stack(input_ids)
    batch['attention_mask'] = torch.stack(attention_mask)
    batch['pixel_values'] = torch.stack(pixel_values)
    batch['labels'] = torch.stack(labels)

    return batch

train_dataloader = DataLoader(train_vqa_dataset,
                              collate_fn=collate_fn,
                              batch_size=64,
                              shuffle=False)
val_dataloader = DataLoader(val_vqa_dataset,
                            collate_fn=collate_fn,
                            batch_size=64,
                            shuffle=False)
```

In [14]:

```
batch = next(iter(train_dataloader))
for k,v in batch.items():
    print(k, v.shape)
```

```
input_ids torch.Size([64, 32])
attention_mask torch.Size([64, 32])
pixel_values torch.Size([64, 3, 128, 128])
labels torch.Size([64, 32])
```

Build Model

In [15]:
```
model = BlipForQuestionAnswering.from_pretrained("Salesforce/blip-vqa-base" )
model.to(device)
```

```
Downloading pytorch_model.bin: 100% 1.54G/1.54G [00:07<00:00, 194MB/s]
```

Out[15]:
```
BlipForQuestionAnswering(
  (vision_model): BlipVisionModel(
    (embeddings): BlipVisionEmbeddings(
      (patch_embedding): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
    )
    (encoder): BlipEncoder(
      (layers): ModuleList(
        (0-11): 12 x BlipEncoderLayer(
          (self_attn): BlipAttention(
            (dropout): Dropout(p=0.0, inplace=False)
            (qkv): Linear(in_features=768, out_features=2304, bias=True)
            (projection): Linear(in_features=768, out_features=768, bias=True)
          )
          (layer_norm1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (mlp): BlipMLP(
            (activation_fn): GELUActivation()
            (fc1): Linear(in_features=768, out_features=3072, bias=True)
            (fc2): Linear(in_features=3072, out_features=768, bias=True)
          )
          (layer_norm2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
    (post_layernorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (text_encoder): BlipTextModel(
    (embeddings): BlipTextEmbeddings(
      (word_embeddings): Embedding(30524, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): BlipTextEncoder(
      (layer): ModuleList(
        (0-11): 12 x BlipTextLayer(
          (attention): BlipTextAttention(
```

```
  (self): BlipTextSelfAttention(
    (query): Linear(in_features=768, out_features=768, bias=True)
    (key): Linear(in_features=768, out_features=768, bias=True)
    (value): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.0, inplace=False)
  )
  (output): BlipTextSelfOutput(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.0, inplace=False)
  )
)
(crossattention): BlipTextAttention(
  (self): BlipTextSelfAttention(
    (query): Linear(in_features=768, out_features=768, bias=True)
    (key): Linear(in_features=768, out_features=768, bias=True)
    (value): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.0, inplace=False)
  )
  (output): BlipTextSelfOutput(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)


  (crossattention): BlipTextAttention(
    (self): BlipTextSelfAttention(
      (query): Linear(in_features=768, out_features=768, bias=True)
      (key): Linear(in_features=768, out_features=768, bias=True)
      (value): Linear(in_features=768, out_features=768, bias=True)
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (output): BlipTextSelfOutput(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.0, inplace=False)
    )
  )
  (intermediate): BlipTextIntermediate(
    (dense): Linear(in_features=768, out_features=3072, bias=True)
    (intermediate_act_fn): GELUActivation()
  )
  (output): BlipTextOutput(
    (dense): Linear(in_features=3072, out_features=768, bias=True)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.0, inplace=False)
  )
)
```

```
    )
  )
)
(text_decoder): BlipTextLMHeadModel(
  (bert): BlipTextModel(
    (embeddings): BlipTextEmbeddings(
      (word_embeddings): Embedding(30524, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): BlipTextEncoder(
      (layer): ModuleList(
        (0-11): 12 x BlipTextLayer(
          (attention): BlipTextAttention(
            (self): BlipTextSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (output): BlipTextSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
        )
        (crossattention): BlipTextAttention(
          (self): BlipTextSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (output): BlipTextSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
        )
        (intermediate): BlipTextIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
          (intermediate_act_fn): GELUActivation()
        )
        (output): BlipTextOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
```

```
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
        )
      )
    )
    (cls): BlipTextOnlyMLMHead(
      (predictions): BlipTextLMPredictionHead(
        (transform): BlipTextPredictionHeadTransform(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (transform_act_fn): GELUActivation()
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
        (decoder): Linear(in_features=768, out_features=30524, bias=True)
      )
    )
  )
)
```

In [16]:

```
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)
image_mean = image_processor.image_mean
image_std = image_processor.image_std
```

In [17]:

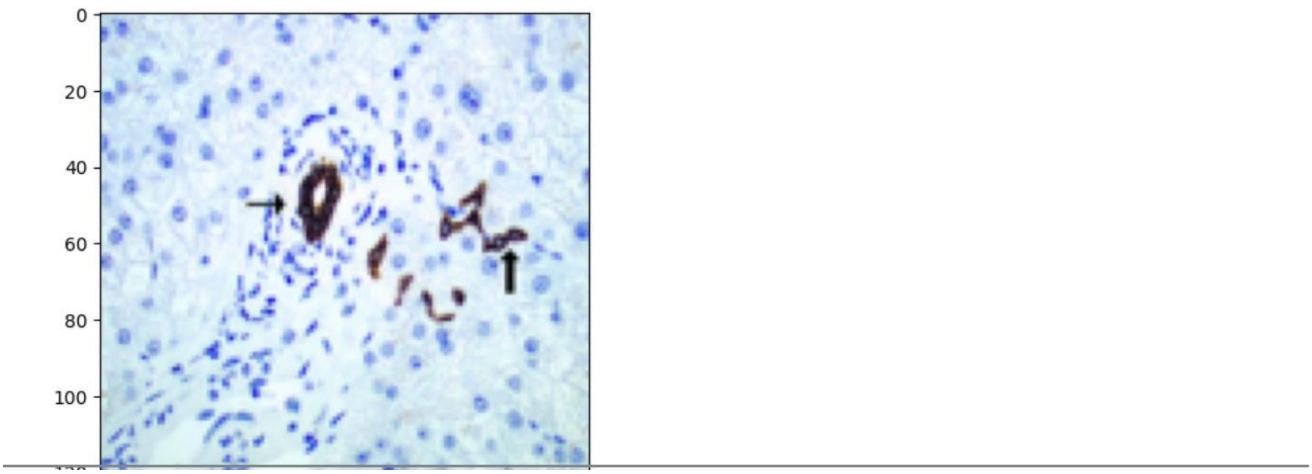
```
batch_idx = 1

unnormalized_image = (batch["pixel_values"][batch_idx].cpu().numpy() * np.array(image_std)[:, None, None]) + np.arra
y(image_mean)[:, None, None]
unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
unnormalized_image = (unnormalized_image * 255).astype(np.uint8)

print("Question: ",text_processor.decode(batch["input_ids"][batch_idx]))
print("Answer: ",text_processor.decode(batch["labels"][batch_idx]))
plt.imshow(Image.fromarray(unnormalized_image))
```

```
Question:  [CLS] what are stained here with an immunohistochemical stain for cytokeratin 7? [SEP] [PAD] [PAD] [P
AD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
Answer:  [CLS] bile duct cells and canals of hering [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

```
Question:  [CLS] what are stained here with an immunohistochemical stain for cytokeratin 7? [SEP] [PAD] [PAD] [P
AD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
Answer:  [CLS] bile duct cells and canals of hering [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

17]:

```
<matplotlib.image.AxesImage at 0x7b6b586fa980>
```



Model Training

[18]:

```python
model.train()
for epoch in range(90):
    print(f"Epoch: {epoch}")
    total_loss = []
    for batch in tqdm(train_dataloader):
        # get the inputs;
        batch = {k:v.to(device) for k,v in batch.items()}

        # zero the parameter gradients
        optimizer.zero_grad()
        # forward + backward + optimize
        outputs = model(**batch)
        loss = outputs.loss
        total_loss.append(loss.item())
        loss.backward()
        optimizer.step()
    print("Loss:", sum(total_loss))
```

```
Epoch: 0
```


```
We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://huggingfac
e.co/docs/transformers/troubleshooting#incorrect-output-when-padding-tokens-arent-masked.

Loss: 574.7836776971817
Epoch: 1
```


```
Loss: 257.98637652397156
Epoch: 2
```

```
Epoch: 2
```


```
Loss: 249.79364371299744
Epoch: 3
```


```
Loss: 244.079598903656
Epoch: 4
```


```
Loss: 239.29888260364532
```

```
  Epoch: 5
100% |████████████████████████████████| 157/157 [06:12<00:00, 1.99s/it]
  Loss: 235.4876273870468
  Epoch: 6
100% |████████████████████████████████| 157/157 [06:11<00:00, 1.99s/it]
  Loss: 232.84861755371094
  Epoch: 7
100% |████████████████████████████████| 157/157 [06:11<00:00, 1.97s/it]
  Loss: 230.61730003356934
Epoch: 8
100% |████████████████████████████████| 157/157 [06:11<00:00, 1.97s/it]
Loss: 229.09221410751343
Epoch: 9
100% |████████████████████████████████| 157/157 [06:12<00:00, 1.97s/it]
Loss: 227.45366883277893
Epoch: 10
100% |████████████████████████████████| 157/157 [06:11<00:00, 2.03s/it]
Loss: 226.34723448753357
```

```
Epoch: 11
100% ████████████████████████████████ 157/157 [06:10<00:00, 1.98s/it]
Loss: 225.8037050962448
Epoch: 12
100% ████████████████████████████████ 157/157 [06:10<00:00, 1.97s/it]
Loss: 225.02576446533203
Epoch: 13
100% ████████████████████████████████ 157/157 [06:11<00:00, 2.01s/it]
Loss: 224.46152532100677
Epoch: 14
100% ████████████████████████████████ 157/157 [06:09<00:00, 1.99s/it]
Loss: 223.5590137243271
Epoch: 15
100% ████████████████████████████████ 157/157 [06:10<00:00, 1.99s/it]
Loss: 223.07532668113708
Epoch: 16
100% ████████████████████████████████ 157/157 [06:06<00:00, 1.96s/it]
Loss: 222.6379153728485
```

```
Epoch: 17
100% 157/157 [06:04<00:00, 1.94s/it]
Loss: 222.42142593860626
Epoch: 18
100% 157/157 [06:03<00:00, 1.99s/it]
Loss: 222.47393131256104
Epoch: 19
100% 157/157 [06:02<00:00, 1.96s/it]
Loss: 222.94616162776947
Epoch: 71
100% 157/157 [06:10<00:00, 1.97s/it]
Loss: 220.73871207237244
Epoch: 72
100% 157/157 [06:13<00:00, 2.01s/it]
Loss: 220.57244634628296
Epoch: 73
100% 157/157 [06:13<00:00, 2.00s/it]
Loss: 220.4587084054947
```

```
Epoch: 74
100% 157/157 [06:13<00:00, 2.01s/it]
Loss: 220.42365729808807
Epoch: 75
100% 157/157 [06:12<00:00, 1.97s/it]
Loss: 220.6499571800232
Epoch: 76
100% 157/157 [06:14<00:00, 1.98s/it]
Loss: 220.5023900270462
Epoch: 77
100% 157/157 [06:12<00:00, 2.00s/it]
Loss: 220.1287806034088
Epoch: 78
100% 157/157 [06:12<00:00, 2.07s/it]
Loss: 220.07679069042206
Epoch: 79
100% 157/157 [06:07<00:00, 1.94s/it]
Loss: 220.12000811100006
```

```
Epoch: 80
100% ████████████████████████████████ 157/157 [06:04<00:00, 2.00s/it]
Loss: 220.8427780866623
Epoch: 81
100% ████████████████████████████████ 157/157 [06:02<00:00, 1.94s/it]
Loss: 220.70872151851654
Epoch: 82
100% ████████████████████████████████ 157/157 [06:03<00:00, 2.04s/it]
Loss: 220.66473019123077
Epoch: 83
100% ████████████████████████████████ 157/157 [06:04<00:00, 1.99s/it]
Loss: 220.74199163913727
Epoch: 84
100% ████████████████████████████████ 157/157 [06:07<00:00, 2.01s/it]
Loss: 221.01178693771362
Epoch: 85
100% ████████████████████████████████ 157/157 [06:03<00:00, 1.94s/it]
Loss: 221.20216274261475
```

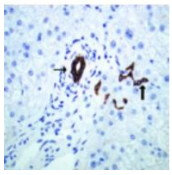
```
Epoch: 86
100% ████████████████████████████████ 157/157 [06:03<00:00, 2.02s/it]
Loss: 220.70832359790802
Epoch: 87
100% ████████████████████████████████ 157/157 [06:02<00:00, 1.95s/it]
Loss: 220.52759456634521
Epoch: 88
100% ████████████████████████████████ 157/157 [06:04<00:00, 1.98s/it]
Loss: 220.39693200588226
Epoch: 89
100% ████████████████████████████████ 157/157 [06:04<00:00, 1.94s/it]
Loss: 220.27243375778198
```

Inference

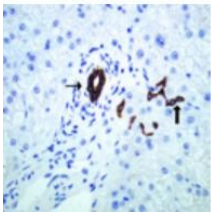
In [19]:

```
# add batch dimension + move to GPU|
for x in range(100):
    sample = val_vqa_dataset[x]
    print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
    sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

    # forward pass
    outputs = model.generate(pixel_values=sample['pixel_values'],
                             input_ids=sample['input_ids'])
    print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
    print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
    ########################################################################
    unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(i
mage_mean)[:, None, None]
    unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
    unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
    display(Image.fromarray(unnormalized_image))
    ########################################################################
    print("########################################################################")
```

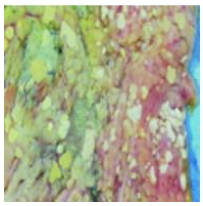
```
Question:  where are liver stem cells ( oval cells ) located?

/opt/conda/lib/python3.10/site-packages/transformers/generation/utils.py:1260: UserWarning: Using the model-agno
stic default `max_length` (=20) to control thegeneration length. We recommend setting `max_new_tokens` to contro
l the maximum length of the generation.
  warnings.warn(

Predicted Answer:  in the canals of hering
Actual Answer:  in the canals of hering
```



##
Question: what are stained here with an immunohistochemical stain for cytokeratin 7?
Predicted Answer: bile duct cells and canals of hering
Actual Answer: bile duct cells and canals of hering



##
Question: what do the areas of white chalky deposits represent?
Predicted Answer: foci of fat necrosis
Actual Answer: foci of fat necrosis



Question: is embolus derived from a lower - extremity deep venous thrombus lodged in a pulmonary artery branch?
Predicted Answer: yes
Actual Answer: yes

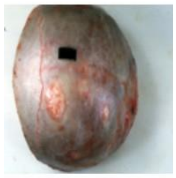
In [20]:

```
for x in range(500,600):
    sample = val_vqa_dataset[x]
    print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
    sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

    # forward pass
    outputs = model.generate(pixel_values=sample['pixel_values'],
                             input_ids=sample['input_ids'])
    print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
    print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
    ######################################################################
    unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(i
mage_mean)[:, None, None]
    unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
    unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
    display(Image.fromarray(unnormalized_image))
    ######################################################################
    print("######################################################################")
```

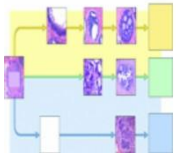
```
Question:  is cytomegalovirus present?
Predicted Answer:  yes
Actual Answer:  no
```

```
######################################################################
Question:  what is present?
Predicted Answer:  a cluster of basophilic
Actual Answer:  metastatic pancreas carcinoma
```

Question: is metastatic pancreas carcinoma present?
Predicted Answer: yes
Actual Answer: yes



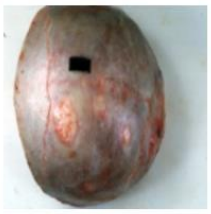
##
Question: do the principal cellular alterations that characterize reversible cell injury and necrosis include f
lat epithelial atypia, adh, and dc
Predicted Answer: no
Actual Answer: no



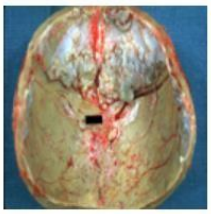
##
Question: is eye present?
Predicted Answer: no
Actual Answer: no

##
Question: what is present?
Predicted Answer: a cluster of basophilic
Actual Answer: bone, calvarium

```
Question:  what is present?
Predicted Answer:  a cluster of basophilic
Actual Answer:  metastatic pancreas carcinoma
```



```
######################################################################
Question:  what does this image show?
Predicted Answer:  rocky mountain
Actual Answer:  several rather large lesions
```



In [21]:

```
idx = 751
sample = val_vqa_dataset[idx]
print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

# forward pass
outputs = model.generate(pixel_values=sample['pixel_values'],
                         input_ids=sample['input_ids'])
print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
######################################################################
unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(image
_mean)[:, None, None]
unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
plt.imshow(Image.fromarray(unnormalized_image))
```

```
Question:  what is present?
Predicted Answer:  this lesion
Actual Answer:  potters facies
```

ıt[21]:

```
<matplotlib.image.AxesImage at 0x7b6ab2e45b40>
```

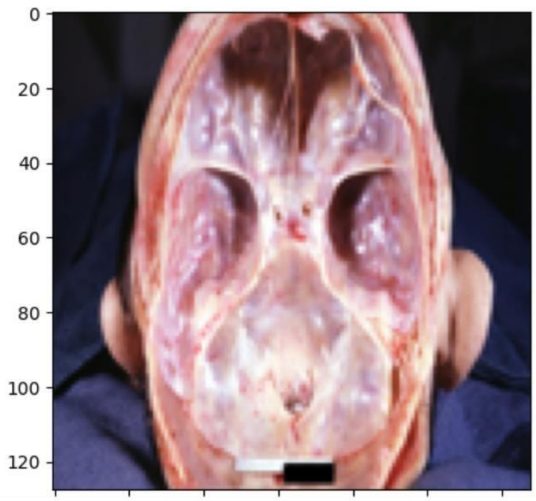
In [22]:

```
idx = 700
sample = val_vqa_dataset[idx]
print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

# forward pass
outputs = model.generate(pixel_values=sample['pixel_values'],
                         input_ids=sample['input_ids'])
print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
#####################################################################
unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(image
_mean)[:, None, None]
unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
plt.imshow(Image.fromarray(unnormalized_image))
```

```
Question:  does this image show stenosis of foramen magnum due to subluxation of atlas vertebra case 31?
Predicted Answer:  yes
Actual Answer:  yes
```

Out[22]:

```
<matplotlib.image.AxesImage at 0x7b6ab2e03190>
```



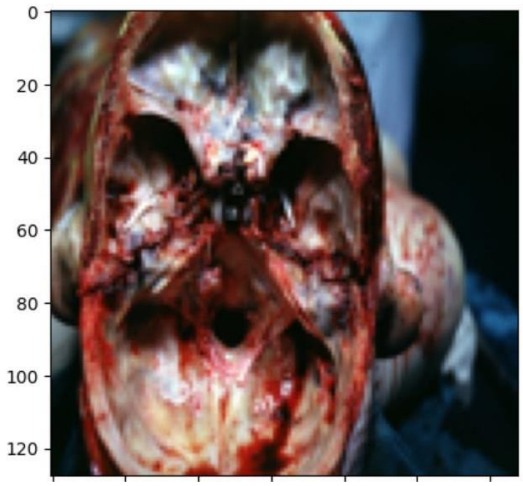
In [23]:

```python
idx = 720
sample = val_vqa_dataset[idx]
print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

# forward pass
outputs = model.generate(pixel_values=sample['pixel_values'],
                         input_ids=sample['input_ids'])
print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
#########################################################################
unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(image
_mean)[:, None, None]
unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
plt.imshow(Image.fromarray(unnormalized_image))
```

```
Question:  is beckwith - wiedemann syndrome present?
Predicted Answer:  yes
Actual Answer:  no
```

Out[23]:

```
<matplotlib.image.AxesImage at 0x7b6ab213ab60>
```



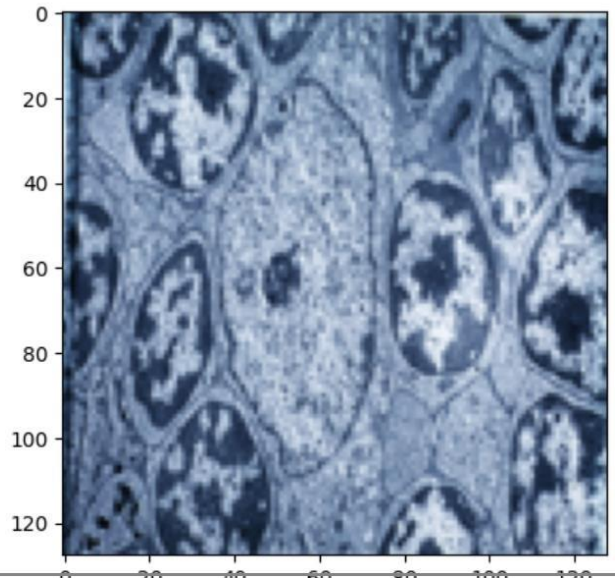
In [24]:

```
idx = 790
sample = val_vqa_dataset[idx]
print("Question: ",text_processor.decode(sample['input_ids'], skip_special_tokens=True))
sample = {k: v.unsqueeze(0).to(device) for k,v in sample.items()}

# forward pass
outputs = model.generate(pixel_values=sample['pixel_values'],
                         input_ids=sample['input_ids'])
print("Predicted Answer: ",text_processor.decode(outputs[0],skip_special_tokens=True))
print("Actual Answer: ",text_processor.decode(sample['labels'][0], skip_special_tokens=True))
######################################################################
unnormalized_image = (sample["pixel_values"][0].cpu().numpy() * np.array(image_std)[:, None, None]) + np.array(image
_mean)[:, None, None]
unnormalized_image = (unnormalized_image * 255).astype(np.uint8)
unnormalized_image = np.moveaxis(unnormalized_image, 0, -1)
plt.imshow(Image.fromarray(unnormalized_image))
```

```
Question:  is thymus present?
Predicted Answer:  yes
Actual Answer:  yes
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7b6ab21b4e50>
```



Conclusion

A successfully trained and inferred medical Visual Question Answering model has the potential to revolutionize healthcare by enhancing diagnostic accuracy, improving efficiency, and expanding access to medical information. However, careful consideration of ethical, privacy, and regulatory issues is crucial to ensure its safe and responsible use in clinical practice.

8. RESULT&APPLICATIONS

```
Question:  what is present?
Predicted Answer:  this lesion
Actual Answer:  potters facies
```

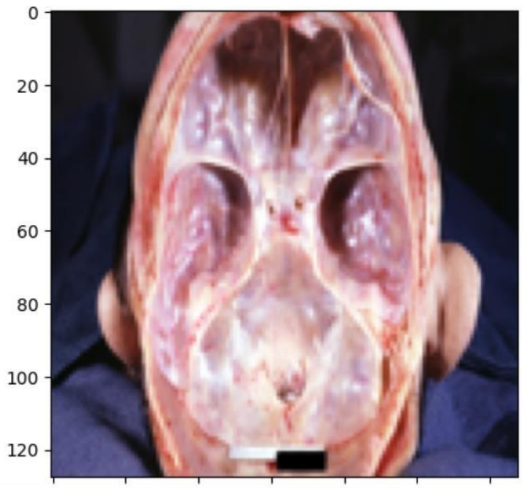
Out[21]:

```
<matplotlib.image.AxesImage at 0x7b6ab2e45b40>
```


```
Question:  does this image show stenosis of foramen magnum due to subluxation of atlas vertebra case 31?
Predicted Answer:  yes
Actual Answer:  yes
```

Out[22]:

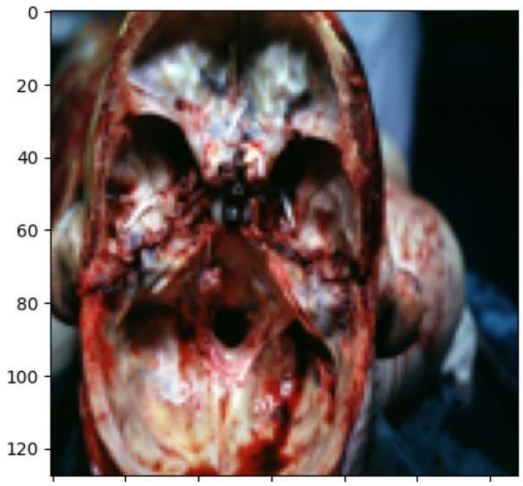
```
<matplotlib.image.AxesImage at 0x7b6ab2e03190>
```



```
Question:  is beckwith - wiedemann syndrome present?
Predicted Answer:  yes
Actual Answer:  no
```

Out[23]:

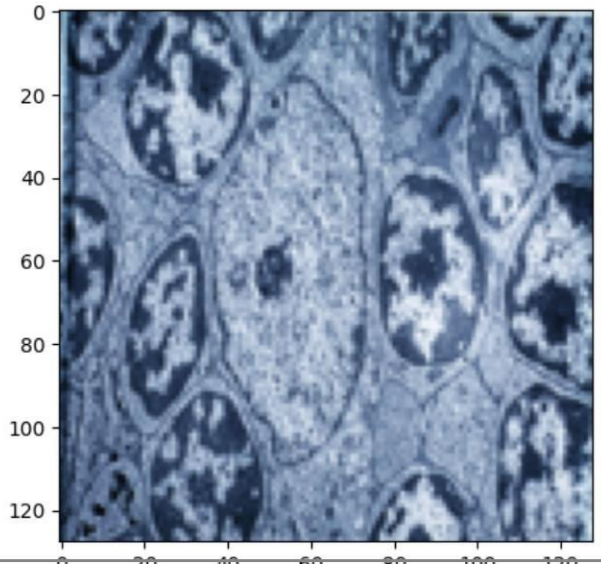
```
<matplotlib.image.AxesImage at 0x7b6ab213ab60>
```




```
Question:  is thymus present?
Predicted Answer:  yes
Actual Answer:  yes
```

Out[24]:

```
<matplotlib.image.AxesImage at 0x7b6ab21b4e50>
```



APPLICATIONS:

- **Enhanced Diagnostic Accuracy**: The VQA model can assist healthcare professionals in accurately diagnosing medical conditions by providing insights and answers to medical questions based on visual data.
- **Improved Efficiency**: By automating parts of the diagnostic process, the VQA model can streamline healthcare workflows, reduce the time taken for diagnosis, and optimize resource utilization in medical facilities.
- **Expanded Access to Medical Information**: Patients, particularly in underserved areas or remote locations, can benefit from access to medical information provided by the VQA model, enabling them to make informed decisions about their health.
- **Ethical Considerations**: Careful consideration of ethical implications is necessary to ensure that the use of the VQA model respects patient autonomy, confidentiality, and consent, and avoids bias or discrimination in diagnosis and treatment.
- **Privacy Concerns**: Safeguarding patient privacy is critical to prevent unauthorized access or misuse of sensitive medical data used by the VQA model, necessitating robust security measures and data protection protocols.
- **Regulatory Compliance**: Compliance with healthcare regulations and standards, such as HIPAA (Health Insurance Portability and Accountability Act) in the United States, is essential to ensure that the deployment of the VQA model in clinical practice adheres to legal requirements and patient rights.
- **Responsible Deployment**: Responsible deployment of the VQA model involves ongoing monitoring, evaluation, and validation to assess its performance, address any biases or errors, and ensure its safe and effective use in clinical settings.

9. CONCLUSION

- We propose BLIP, a new VLP framework with state of-the-art performance on a wide range of downstream vision-language tasks, including understanding-based and generation-based tasks.
- BLIP pre-trains a multimodal mixture of encoder-decoder model using a dataset bootstrapped from large-scale noisy image-text pairs by injecting diverse synthetic captions and removing noisy captions.
- Our boot strapped dataset will be released to facilitate future vision language research. There are a few potential directions that can further enhance the performance of BLIP, which we do not explore in this paper due to the increased computation cost from these approaches: (1) Multiple rounds of dataset bootstrapping
- (2) Generate multiplesynthetic captions per image to further enlarge the pre-training corpus
- (3) Model ensemble by training multiple different captioners and filters and combining their forces in CapFilt.

10. FUTURE SCOPE

- **Multimodal Learning Advancements**: Further exploration of multimodal learning techniques to enhance the model's understanding of the relationship between different modalities such as images, text, and videos.
- **Cross-Domain Generalization**: Extending the model's capabilities to generalize across different domains, allowing it to perform well on tasks with diverse datasets and contexts.
- **Fine-Grained Analysis**: Conducting in-depth analyses to understand the strengths and weaknesses of the model on specific types of data or tasks, leading to targeted improvements.
- **Real-World Applications**: Applying the BLIP model to real-world problems such as image and video search, content recommendation systems, and assistive technologies for people with disabilities.
- **Ethical Considerations**: Investigating the ethical implications of deploying such advanced AI systems, including issues related to bias, fairness, and privacy, and developing strategies to mitigate potential harms.
- **User Interface Integration**: Integrating BLIP into user-friendly interfaces to make its capabilities accessible to a wider audience, potentially leading to innovative applications in creative industries, education, and healthcare.
- **Incremental Learning**: Exploring techniques for incremental learning that allow the model to adapt and improve over time as it encounters new data and tasks, enabling lifelong learning capabilities.
- **Collaborative Research**: Collaborating with researchers from diverse fields such as computer vision, natural language processing, cognitive science, and neuroscience to gain deeper insights and drive interdisciplinary advancements in AI