پروژه سوم درس آنالیز داده

آنوشا شریعتی 9923041

املین غازاریان 9923056

استاد درس:

دکتر شریفیان

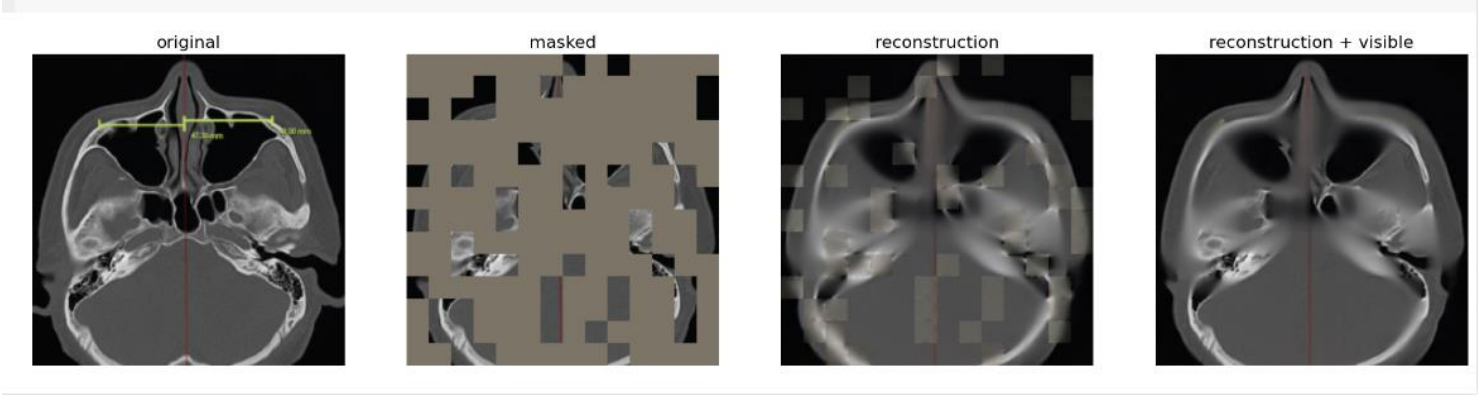
بهار 1403

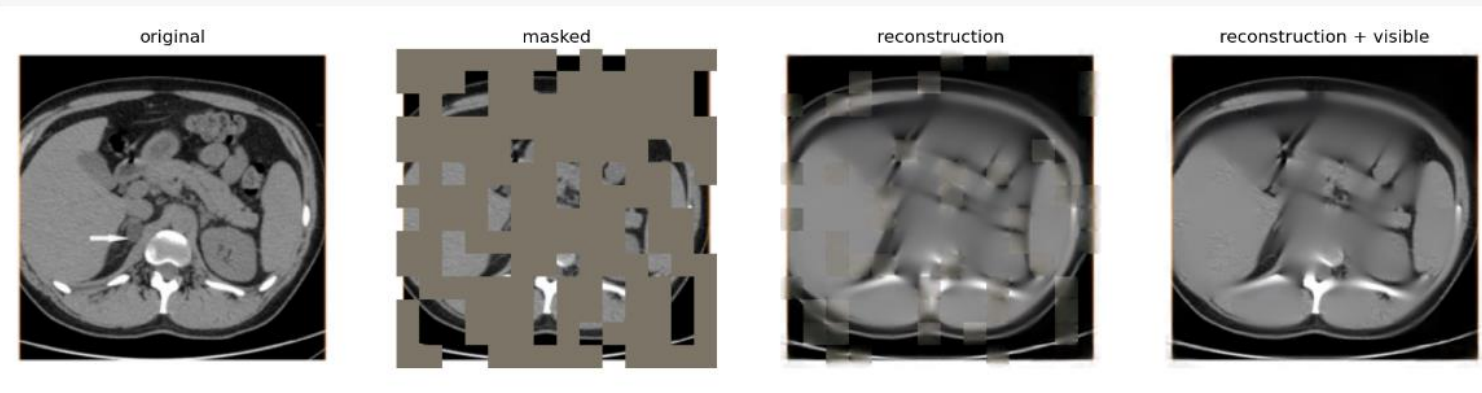
بخش اول:

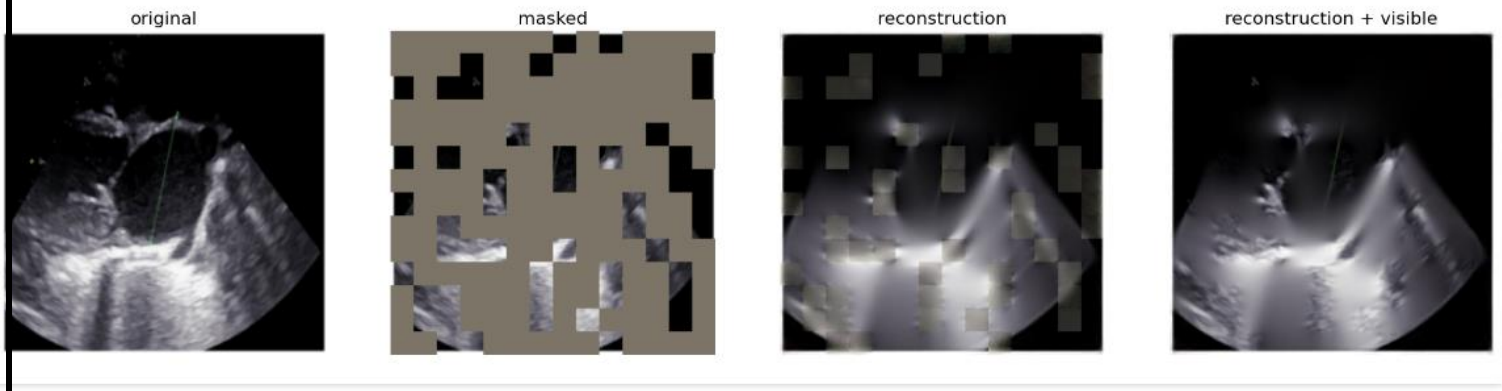
https://colab.research.google.com/drive/1PvMvS6YyNTbhdslhqQLaNGDV6JsTY6FS?usp=sharing

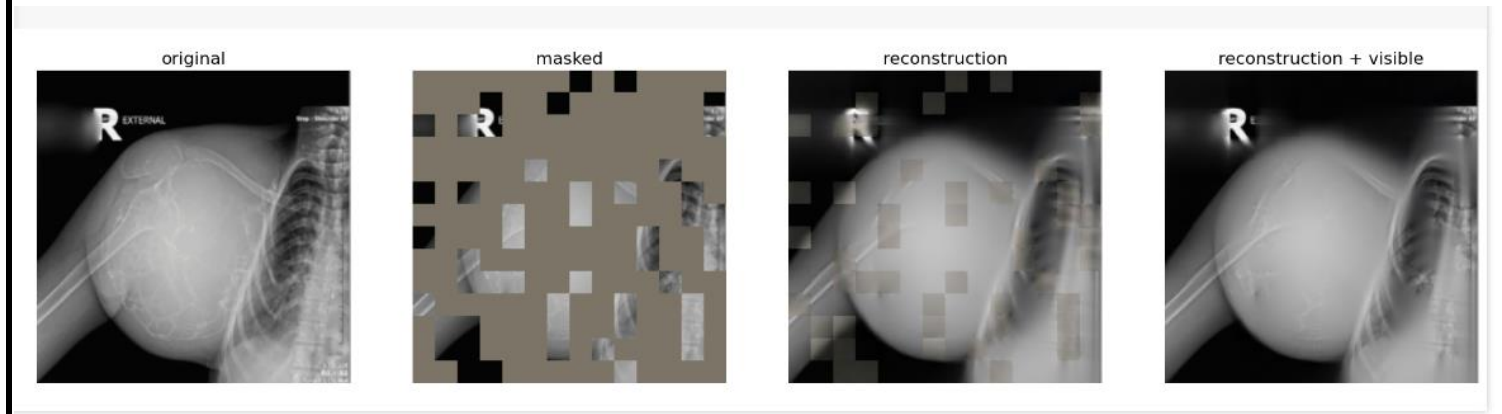
سوال یک: تصاویر مربوط به تست و آموزش دیتاست را در گوگل درایو بارگزاری کردیم تا هنگام نیاز به دیتاست فقط درایو را لود کرده و به تصاویر دسترسی پیدا کنیم و دیگر نیازی به دانلود دیتاست نداشته باشیم.

سوال دو : مدل VITMAE که از پیش آموزش دیده است را لود کرده و روی 5 تصویر تست از دیتاست داده شده اعمال کردیم. نتایج بازسازی با اعمال75 درصد ماسک به صورت زیر شد. همانطور که مشاهده میشود کلیات تصاویر به خوبی بازسازی شده است ولی مقداری از جزئیات تصویر حذف شده است.









سوال سه :

https://colab.research.google.com/drive/18DcCBdQkIo31IdVCQDku9bbo9oOmISJO?usp=sharing

در این قسمت در ابتدا کتابخانه های مورد نیاز را ایمپورت کرده. برای قابل استفاده کردن دیتاست برای مدل vitmae کلاسCustomImageDataset به صورت زیر تعریف شد تا با استفاده از آن مقادیر pixel_valuesبه صورت ورودی به مدل داده شود.

```
class CustomImageDataset(Dataset):
    def __init__(self, image_dir , feature_extractor):
        self.image_dir = image_dir
        self.feature_extractor = feature_extractor
        # Get a list of all image files in the directory
        self.image_files = [f for f in os.listdir(image_dir) if
os.path.isfile(os.path.join(image_dir, f))]

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.image_dir,
self.image_files[idx])
        image = Image.open(img_path).convert("RGB")
        inputs = self.feature_extractor(images=image,
return_tensors="pt")
        image_file_name = os.path.basename(img_path)

        return {
            'pixel_values': inputs['pixel_values'].squeeze(),
            'noise': None,
            'head_mask': None,
            'output_attentions': None,
            'output_hidden_states': None,
            'return_dict': None,
            'interpolate_pos_encoding': None,
            'label_ids': None,
            'labels': None
        }
```

در ادامه تابع تعریف شده روی دیرکتوری مربوط به دیتای تست و ترین اعمال شده ودیتاست تست و ترین به دست می آید. سپس تنظیمات مروبط به آموزش را انجام میدهیم برای مثال تعداد ایپاک ها را روی 5 ست میکنیم. و در نهایت شروع به ترین کردن میکنیم. بعد از اینکه ترین تمام شد مدل را ذخیره میکنیم که بتوان در قسمت بعد از آن استفاده کرد.

```
feature_extractor = ViTFeatureExtractor.from_pretrained("facebook/vit-mae-base")
```

Show hidden output

```
image_dir_train = '/content/train_images/train'
image_dir_test = '/content/test_images/test'
```

```
train_dataset = CustomImageDataset(image_dir=image_dir_train,  feature_extractor=feature_extractor)
test_dataset = CustomImageDataset(image_dir=image_dir_test,  feature_extractor=feature_extractor)
```

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=5,
    per_device_eval_batch_size=5,
    evaluation_strategy="epoch",
    logging_dir='./logs',
)
```

```
model = ViTMAEForPreTraining.from_pretrained("facebook/vit-mae-base")

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
```

```
# Step 5: Train the model
trainer.train()
```

[66/33065 00:12 < 1:48:49, 5.05 it/s, Epoch 0.01/5]

Epoch	Training Loss	Validation Loss

سوال چهار: در این قسمت مدل finetune شده در قسمت قبل را لود کرده و مانند قسمت دوم عملکرد آن را در بازسازی 5 تصویر از مجموعه تست را بررسی میکنیم.

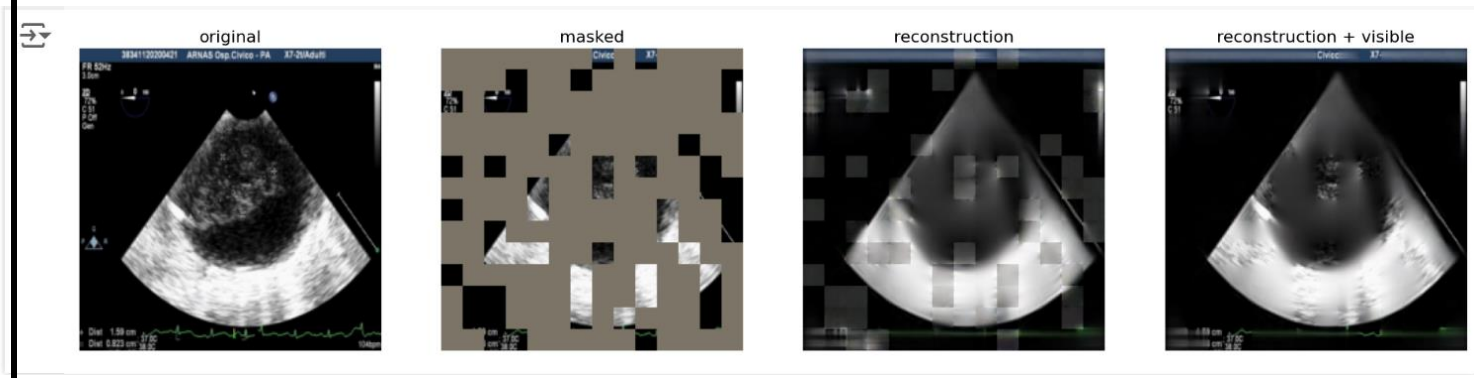
```
feature_extractor = ViTFeatureExtractor.from_pretrained("facebook/vit-mae-base")
pixel_values = feature_extractor(img, return_tensors="pt").pixel_values
```

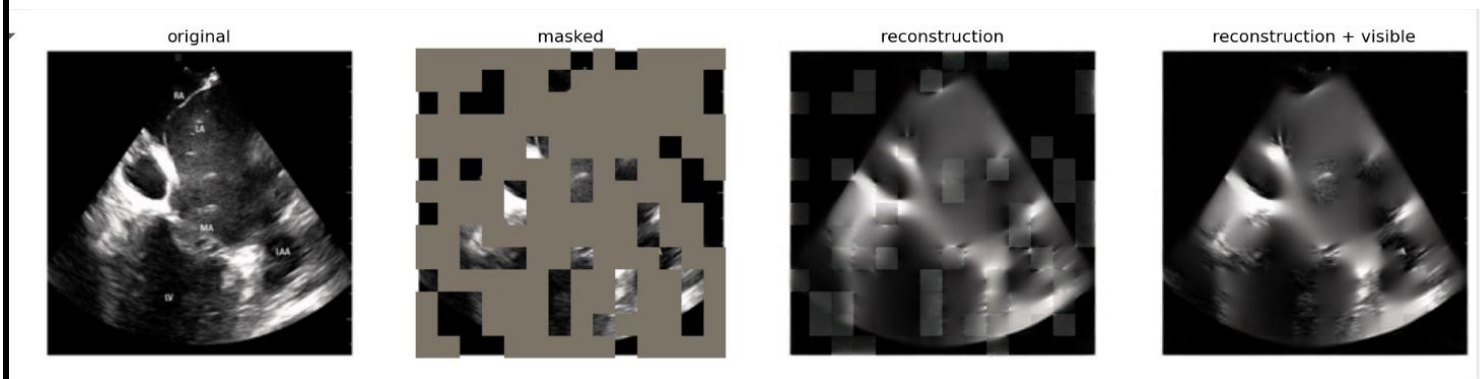
```
import torch
import numpy as np
import matplotlib.pyplot as plt

imagenet_mean = np.array(feature_extractor.image_mean)
imagenet_std = np.array(feature_extractor.image_std)

def show_image(img, title=''):
    # image is [H, W, 3]
    assert img.shape[2] == 3
    plt.imshow(torch.clip((img * imagenet_std + imagenet_mean) * 255, 0, 255).int())
    plt.title(title, fontsize=16)
    plt.axis('off')
    return

def visualize(pixel_values, model):
    # forward pass
    outputs = model(pixel_values)
    y = model.unpatchify(outputs.logits)
```

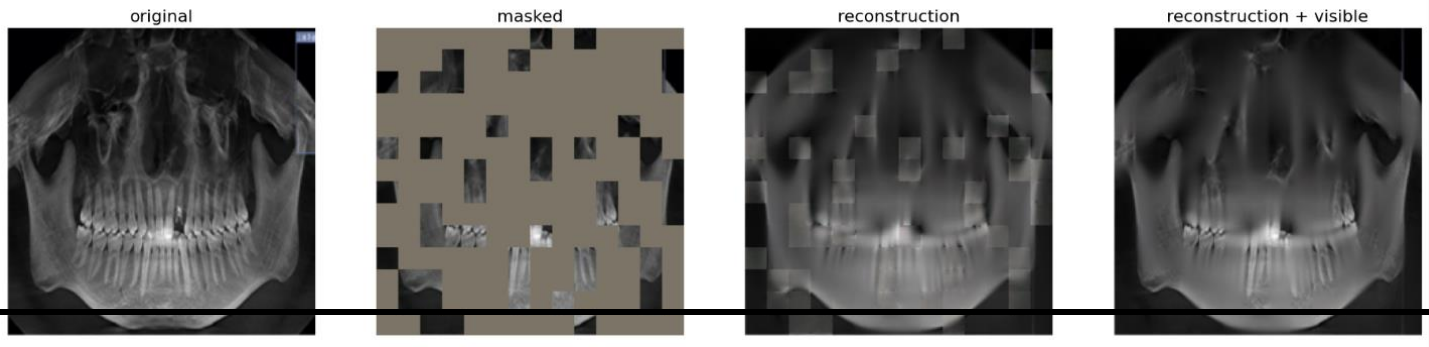
```
# visualize the mask
mask = outputs.mask.detach()
mask = mask.unsqueeze(-1).repeat(1, 1, model.config.patch_size**2 *3)  # (N, H*W, p*p*3)
mask = model.unpatchify(mask)  # 1 is removing, 0 is keeping
mask = torch.einsum('nchw->nhwc', mask).detach().cpu()
x = torch.einsum('nchw->nhwc', pixel_values)
# masked image
im_masked = x * (1 - mask)
# MAE reconstruction pasted with visible patches
im_paste = x * (1 - mask) + y * mask

plt.rcParams['figure.figsize'] = [24, 24]
plt.subplot(1, 4, 1)
show_image(x[0], "original")
plt.subplot(1, 4, 2)
show_image(im_masked[0], "masked")
plt.subplot(1, 4, 3)
show_image(y[0], "reconstruction")
plt.subplot(1, 4, 4)
show_image(im_paste[0], "reconstruction + visible")
plt.show()
```

```
from transformers import ViTMAEForPreTraining

torch.manual_seed(2)

model = ViTMAEForPreTraining.from_pretrained("./finetuned-vit-mae")

visualize(pixel_values, model)
```

همان گونه که مشاهده میشود مدل شده که ذخیره کردیم برای بازسازی تصاویر تست دیتاست استفاده شده و نتایج زیر به دست آمد.







سوال پنج :

https://colab.research.google.com/drive/1T5ZNquhWrBlccEYmD2e0ppbasBpoQUc4?usp=sharing

برای این سوال نیاز بود یک دیتاست جدید شامل تصاویر پزشکی انتخاب کرده و آن را لود کنیم. در ابتدا دیتاست 2را انتخاب کرده و در درایو لود کردیم ولی این دیتاست بیشتر برای کلاسیفیکیشن تومورهای مغزی استفاده میشود و هر کلاس آن تعداد مناسبی عکس ندارد که بتوان با استفاده از آن مدل را ترین کرد به همین علت از دیتاست 3 که تعداد عکسهای بیشتری را شامل میشد استفاده کردیم.

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

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

```
%cd /content/drive/MyDrive/
```

```
/content/drive/MyDrive
```

```
from google.colab import files

# Upload the .zip file
uploaded = files.upload()

# Extract the contents of the .zip file
import zipfile
import io

with zipfile.ZipFile("/content/drive/MyDrive/brain_tumor_dataset.zip", 'r') as zip_ref:
    zip_ref.extractall('/content/dataset3')
    zip_ref.close()
```

سوال شش: با همان روند پیاده شده در سوال سه مدل VITMAE را برای دیتاست جدید finetune میکنیم.

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=5,
    per_device_train_batch_size=5,
    per_device_eval_batch_size=5,
    evaluation_strategy="epoch",
    logging_dir='./logs',
)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1494: FutureWarning: `evaluation_strategy` is
  warnings.warn(
```

```
model = ViTMAEForPreTraining.from_pretrained("facebook/vit-mae-base")

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
)
```

نتیجه آموزش به صورت زیر شد:

```
TrainOutput(global_step=155, training_loss=0.17275587512600807,
metrics={'train_runtime': 75.9864, 'train_samples_per_second':
10.199, 'train_steps_per_second': 2.04, 'total_flos':
7.8330474528768e+16, 'train_loss': 0.17275587512600807, 'epoch':
5.0})
```

همانطور که مشاهده میشود چون تعداد عکس های داخل دیتاست کم بوده ومناسب برای مدل VITMAE نبود نتایج مناسبی به دست نیامده.

بخش دوم:

https://colab.research.google.com/drive/16Sv0KwHTDp25ncNnG9G0U_0vrEirV-aE?usp=sharing

ابتدا در اینجا باید دیتاستی درست کنیم که شامل تصاویری باشد که دارای کپشن هستند. پس تابعی به این صورت می نویسیم:

```
class CustomImageCaptioningDataset(Dataset):
    def __init__(self, zip_file_path, captions_file_path):
        self.zip_file_path = zip_file_path
        self.captions_df = pd.read_csv(captions_file_path)

        # Open the zip file and list image files
        self.archive = zipfile.ZipFile(zip_file_path, 'r')
        self.image_list = [file for file in self.archive.namelist()
if file.lower().endswith(('png', 'jpg', 'jpeg'))]

        # Ensure the filenames in the CSV match those in the zip
file
        self.dataset = []
        for index, row in self.captions_df.iterrows():
            img_name = row['ID']
            caption = row['Caption']
            img_file = f"train/{img_name}.jpg"  # Adjust this if
your paths are different in the zip
            if img_file in self.image_list:
                self.dataset.append({
                    'image_file': img_file,
                    'caption': caption
                })

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

    def __getitem__(self, idx):
        item = self.dataset[idx]
        img_file = item['image_file']
        caption = item['caption']
```

```
        with self.archive.open(img_file) as file:
            image = Image.open(file).convert('RGB')

        return {"image": image, "text": caption}
```

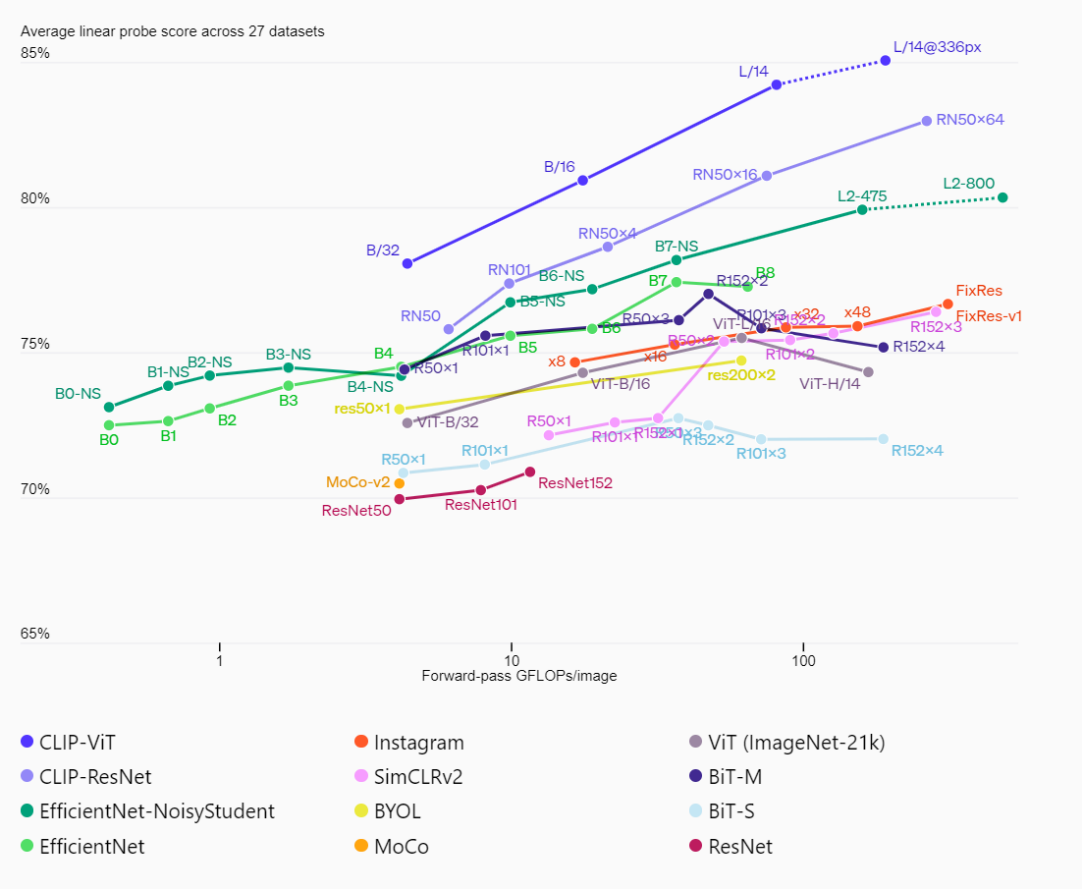
که این تابع با خواندن نام فایل‌های تصویر از یک فایل فشرده و تطبیق آن‌ها با زیرنویس‌های یک فایل CSV، انجام می شود.

دلیل اینکه این تابع را نوشتیم این بود که تابع ImageCaptioningDataset که در گیت هابی که در فایل پروژه قرار دارد دو ورودی میگیرد(dataset و processor) که باید dataset را طوری میساختیم تا مطابق این ساختار شود.

انتخاب مدل و پروسسور:

```
from PIL import Image
import requests
from transformers import CLIPProcessor, CLIPModel
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch16")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-
patch16")
```

به دلیل لینکه جز گروه های اخر مدل خود را انتخاب کردیم خیلی انتخاب های خوبی باقی نمانده بود. ابتدا Tinnylava انتخاب شد ولی به دلیل نبودن فایلی در گیت هاب این مدل ارور برطرف نمی شد سپس درنهایت به مدل های clip رسیدیم با وزن `openai/clip-vit-base-patch16` که در تصویر زیر می توان مشاهده کرد که عملکرد نسبتا خوبی داشته:



بهتر بود مدل L14 انتخاب میشد ولی از قبل این مدل برای گروه دیگه بود.

با توجه به اینکه هرچقدر مدل بزرگ تر باشد یعنی تعداد پارامتر های بیشتری داشته باشد که در ادامه با استفاده از کتابخانه peft و استفاده از lora می توان مدل را کوچک تر کرد سپس fine-tune کرد ولی اگر از همان ابتدا مدل کوچک باشد نیازی به lora نیست(هرچند ما این مراحل را نیز انجام دادیم)

با استفاده از گیت هاب قرار داده شده مراحل تعریف lora را انجام می دهیم.

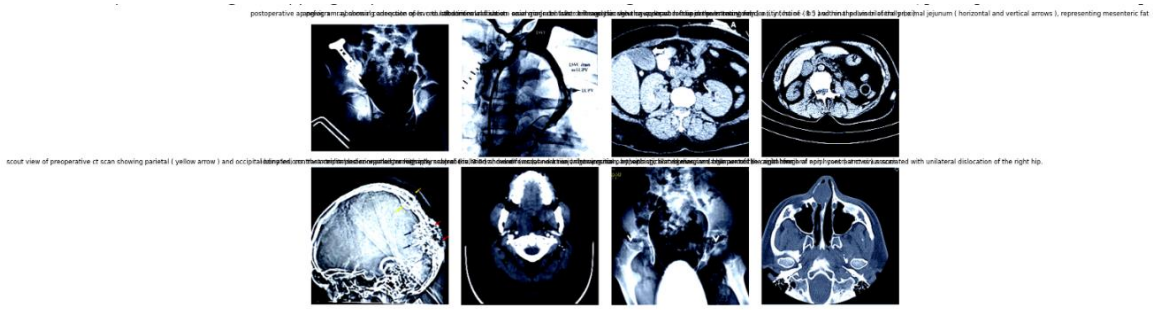
```
from peft import LoraConfig, get_peft_model

# Let's define the LoraConfig
config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    target_modules=["q_proj", "k_proj"]
)

model = get_peft_model(model, config)
model.print_trainable_parameters()
```

```
trainable params: 983040 || all params: 150603777 || trainable%: 0.6527326336576539
```

سپس با استفاده از این تابع، تصویر و کپشن متناظر آن را پرینت می کنیم:

```
# Function to show images and print their captions
def show_images_with_captions(batch, rows):
    num_images = len(batch["input_ids"])
    cols = (num_images + rows - 1) // rows  # Calculate the number
of columns

    plt.figure(figsize=(cols * 4, rows * 4))

    for i in range(num_images):
        img_tensor = batch["pixel_values"][i]
        img = img_tensor.permute(1, 2, 0).numpy()  # Convert tensor
to numpy array
        caption = processor.tokenizer.decode(batch["input_ids"][i],
skip_special_tokens=True)
        plt.subplot(rows, cols, i+1)
        plt.imshow(img)
        plt.title(caption)  # Show the caption as the title
        plt.axis('off')  # Hide axes for better visualization

    plt.tight_layout()
    plt.show()

# Show images with captions
show_images_with_captions(batch, 2)
```

خروجی:



در ادامه موقع آموزش اروری به وجود میاید که به برابر نبودن سایز تصاویر با کپشن ها اشاره می کند.
پس تابعی می نویسیم تا این مشکل برطرف شود:

```
from transformers import CLIPProcessor

processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-
patch16")

def collate_fn(batch):
    images = [item["image"] for item in batch]
    texts = [item["text"] for item in batch]

    # Process images and texts using the processor
    inputs = processor(text=texts, images=images,
return_tensors="pt", padding=True)

    return {
        "input_ids": inputs["input_ids"],
        "attention_mask": inputs["attention_mask"],
        "pixel_values": inputs["pixel_values"]
    }
```

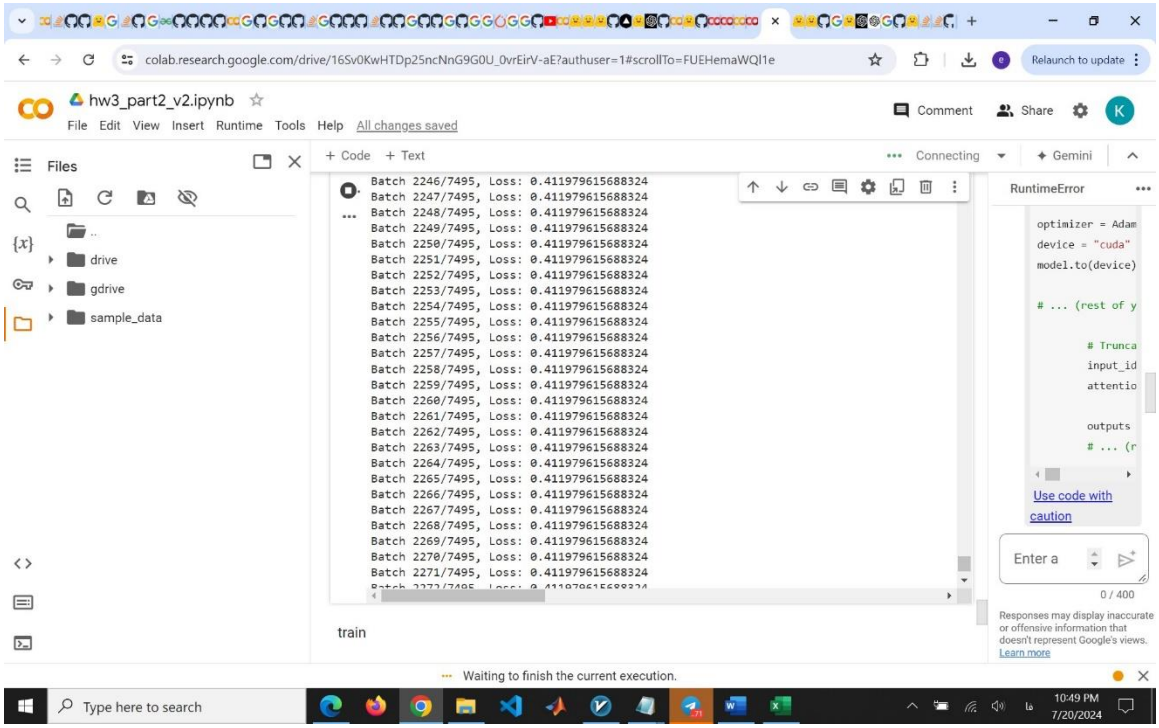
سپس train را انجام می دهیم:

```
import torch
from transformers import CLIPModel, AdamW,
get_linear_schedule_with_warmup
import os
```

```
# Initialize dataset and dataloader
zip_file_path = "train_images.zip"
captions_file_path = '/content/drive/MyDrive/train_captions.csv'
custom_dataset = CustomImageCaptioningDataset(zip_file_path,
captions_file_path)
train_loader = DataLoader(custom_dataset, batch_size=8,
shuffle=True, collate_fn=collate_fn)

# Initialize model and optimizer
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch16")

old_position_embedding =
model.text_model.embeddings.position_embedding
num_position_embeddings = old_position_embedding.num_embeddings
#embedding_dim = old_position_embedding.embedding_dim
#new_position_embedding = torch.nn.Embedding(num_position_embeddings
* 2, embedding_dim)  # Double the possible positions
#new_position_embedding.to(device)  # Move to the same device as the
model
#model.text_model.embeddings.position_embedding =
new_position_embedding

optimizer = AdamW(model.parameters(), lr=1e-4)
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)

num_epochs = 3
total_steps = len(train_loader) * num_epochs
scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=0, num_training_steps=total_steps)

# Contrastive loss function
def contrastive_loss(logits_per_image, logits_per_text):
    batch_size = logits_per_image.size(0)
    labels = torch.arange(batch_size,
device=logits_per_image.device)
    loss_img = torch.nn.functional.cross_entropy(logits_per_image,
labels)
    loss_text = torch.nn.functional.cross_entropy(logits_per_text,
labels)
    return (loss_img + loss_text) / 2

# Training loop with debug prints
model.train()
save_directory = "/content/gdrive/MyDrive/final"
```

```
for epoch in range(num_epochs):
    print(f"Epoch: {epoch+1}/{num_epochs}")
    for idx, batch in enumerate(train_loader):
        #input_ids = batch["input_ids"].to(device)
        #attention_mask = batch["attention_mask"].to(device)
        input_ids = input_ids[:, :num_position_embeddings]
        attention_mask = attention_mask[:, :num_position_embeddings]
        #pixel_values = batch["pixel_values"].to(device)

        #print(f"Batch {idx+1}: input_ids shape: {input_ids.shape},
pixel_values shape: {pixel_values.shape}")

        #if input_ids.size(0) != pixel_values.size(0):
            #raise ValueError(f"Batch size mismatch: input_ids size
{input_ids.size(0)} != pixel_values size {pixel_values.size(0)}")

        outputs = model(input_ids=input_ids,
pixel_values=pixel_values, attention_mask=attention_mask)
        logits_per_image = outputs.logits_per_image
        logits_per_text = outputs.logits_per_text

        loss = contrastive_loss(logits_per_image, logits_per_text)

        print(f"Batch {idx+1}/{len(train_loader)}, Loss:
{loss.item()}")

        loss.backward()

        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

    # Save the model at the end of each epoch
    epoch_save_path = os.path.join(save_directory,
f"model_epoch_{epoch+1}")
    model.save_pretrained(epoch_save_path)
    processor.save_pretrained(epoch_save_path)

# Save the model at the end of training
final_save_path = os.path.join(save_directory, "final_model")
model.save_pretrained(final_save_path)
processor.save_pretrained(final_save_path)
```

که خروجی به این صورت میباشد:



در اینجا متوجه می شویم که train حتی یک ایپاک خیلی زیاد طول می کشد. راه حل:

1-نصف کردن دیتاست

2-تغییر batch size

لود کردن وزن های مدل به صورت 4 بیتی یا 8 بیتی و انجام محاسبات به صورت float16. با اینکار میتوانیم batch size رو افزایش دهیم.

3-انتخاب مدل کوچکتر

1-کوانتایز و کوچک کردن مدل

```
from transformers import CLIPModel, BitsAndBytesConfig
import torch

# Configuration for loading the model in 4-bit mode
bnb_config = BitsAndBytesConfig(
```

```
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=False,
    bnb_4bit_compute_dtype=torch.float16
)

# Load the model with the quantization configuration
model = CLIPModel.from_pretrained(
    "openai/clip-vit-base-patch16",
    quantization_config=bnb_config,
    device_map="auto"
)
```

سپس dataloader را با توجه به تعاریف بالا اپدیت میکنیم:

```
from torch.utils.data import DataLoader, Subset

# Training settings
batch_size = 16
subset_size = 1000  # Use a smaller subset of the dataset for quick
experiments
zip_file_path = "train_images.zip"
captions_file_path = '/content/drive/MyDrive/train_captions.csv'

custom_dataset = CustomImageCaptioningDataset(zip_file_path,
captions_file_path)
subset_dataset = Subset(custom_dataset, range(subset_size))
train_loader = DataLoader(subset_dataset, batch_size=batch_size,
shuffle=True, collate_fn=collate_fn)
```

تعریف optimizer:

```
from transformers import AdamW, get_linear_schedule_with_warmup

optimizer = AdamW(model.parameters(), lr=1e-4)
device = "cuda" if torch.cuda.is_available() else "cpu"
model.to(device)
num_epochs = 1
```

```
total_steps = len(train_loader) * num_epochs
scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=0, num_training_steps=total_steps)
```

سپس loss function را مانند قبل تعریف می کنیم

```
def contrastive_loss(logits_per_image, logits_per_text):
    batch_size = logits_per_image.size(0)
    labels = torch.arange(batch_size,
device=logits_per_image.device)
    loss_img = torch.nn.functional.cross_entropy(logits_per_image,
labels)
    loss_text = torch.nn.functional.cross_entropy(logits_per_text,
labels)
    return (loss_img + loss_text) / 2
```

در نهایت به مرحله آموزش میرسیم:

```
import os

# Training loop
model.train()
save_directory = "/content/gdrive/MyDrive/final"

for epoch in range(num_epochs):
    print(f"Epoch: {epoch+1}/{num_epochs}")
    for idx, batch in enumerate(train_loader):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        pixel_values = batch["pixel_values"].to(device)

        if input_ids.size(0) != pixel_values.size(0):
            raise ValueError(f"Batch size mismatch: input_ids size
{input_ids.size(0)} != pixel_values size {pixel_values.size(0)}")

        outputs = model(input_ids=input_ids,
pixel_values=pixel_values, attention_mask=attention_mask)
        logits_per_image = outputs.logits_per_image
        logits_per_text = outputs.logits_per_text

        loss = contrastive_loss(logits_per_image, logits_per_text)
```

```
        print(f"Batch {idx+1}/{len(train_loader)}, Loss:
{loss.item()}")

        loss.backward()
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

        # Empty cache every few iterations to free up memory
        if idx % 10 == 0:
            torch.cuda.empty_cache()

    # Save the model at the end of each epoch
    epoch_save_path = os.path.join(save_directory,
f"model_epoch_{epoch+1}")
    model.save_pretrained(epoch_save_path)
    processor.save_pretrained(epoch_save_path)

# Save the model at the end of training
final_save_path = os.path.join(save_directory, "final_model")
model.save_pretrained(final_save_path)
processor.save_pretrained(final_save_path)
```

به دلیل محدودیت های کولب و زمان کمی که تا ددلاین باقی مانده بود فرصت نشد با این روش مدل را train بکنیم.