IMAGE CAPTION GENERATION USING ViT AND TRANSFORMER

DIVIJA L	PES1UG20CS134
GAURAV MAHAJAN	PES1UG20CS150

SUMMARY:

The project deals with the task of generation of images using an Encoder – Decoder Model which takes an image as an input and outputs a relevant caption. The encoder used is a ViT which is the Vision Transformer, a state-of-the-art pre-trained model developed by Google for image recognition and classification tasks, object detection and semantic image segmentation. The decoder used is GPT2. The model is trained on the Flickr8K dataset. Seq2Seq Trainer is used for fine-tuning the model. The huggingface library for transformers is been used for parameter and model fine-tuning and Pytorch for data processing.

TASK:

Image Captioning is the task of describing the content of an image in words. This task requires both Computer Vision and Natural Language Processing technologies. Most image captioning systems use an encoder – decoder framework. The input image is encoded into an intermediate representation of the information in the image and then further decoding into a descriptive text sequence takes place.

DATASET:

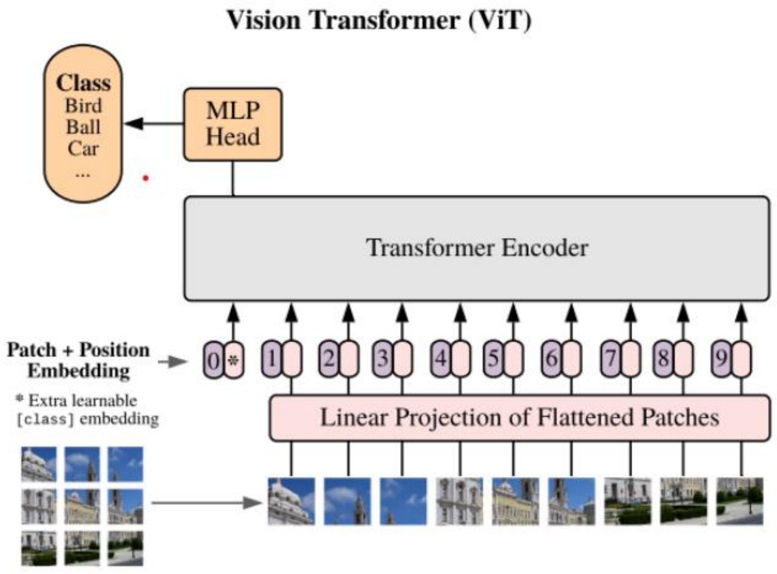
Flickr 8k Dataset

The dataset is the a new benchmark collection for sentences-based image description and search, consisting of 8,000 images that are each paired with five different captions which provide clear descriptions of the salient entities and events. The images were chosen from six different Flickr groups, and tend not to contain any well-known people or locations, but were manually selected to depict a variety of scenes and situations.

The size is 1GB.

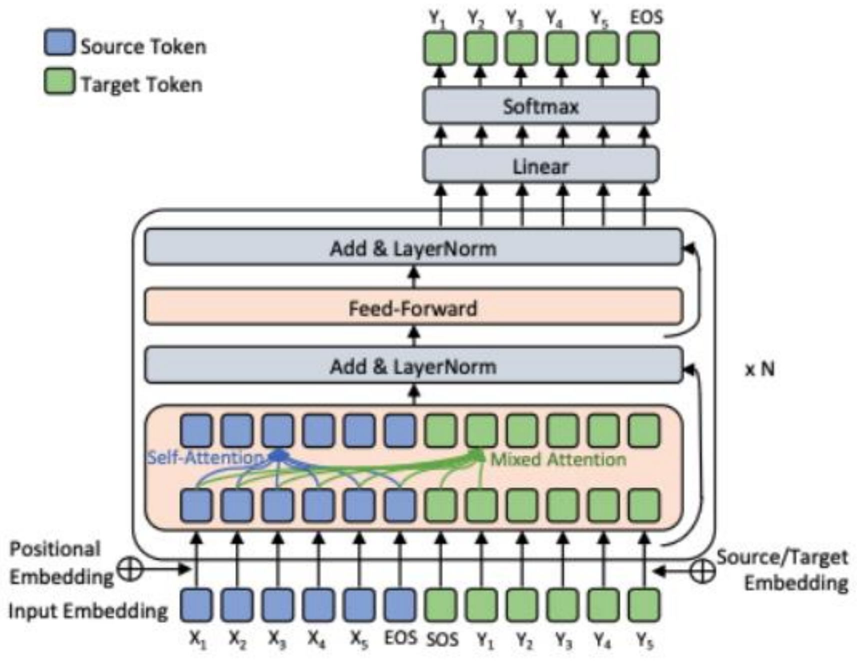
MODEL:

ENCODER


Vision Transformer (ViT)
Class
Bird
Ball
Car
...
MLP Head
Transformer Encoder
Patch + Position Embedding
* Extra learnable [class] embedding
0
*
1
2
3
4
5
6
7
8
9
Linear Projection of Flattened Patches

The Vision Transformer, or ViT, is a model for image classification that employs a Transformer-like architecture over patches of the image. An image is split into fixed-size patches, each of them are then linearly embedded, position embedding are added, and the resulting sequence of vectors is fed to a standard Transformer encoder. In order to perform classification, the standard approach of adding an extra learnable "classification token" to the sequence is used.

DECODER

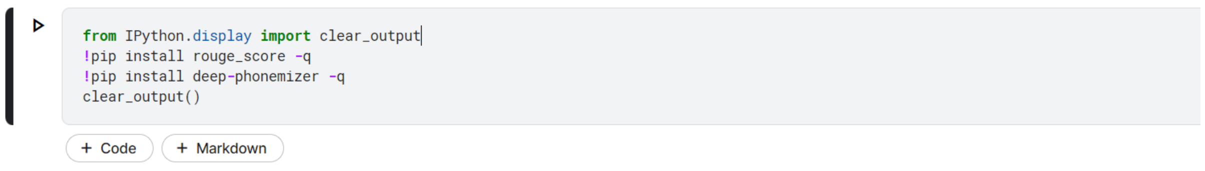


GPT-2 is a transformers model pre-trained on a very large corpus of English data in a self-supervised fashion. Inputs are sequences of continuous text of a certain length and the targets are the same sequence, shifted one token (word or piece of word) to the right. The model uses internally a mask-mechanism to make sure the predictions for the token i only uses the inputs from 1 to i but not the future tokens.

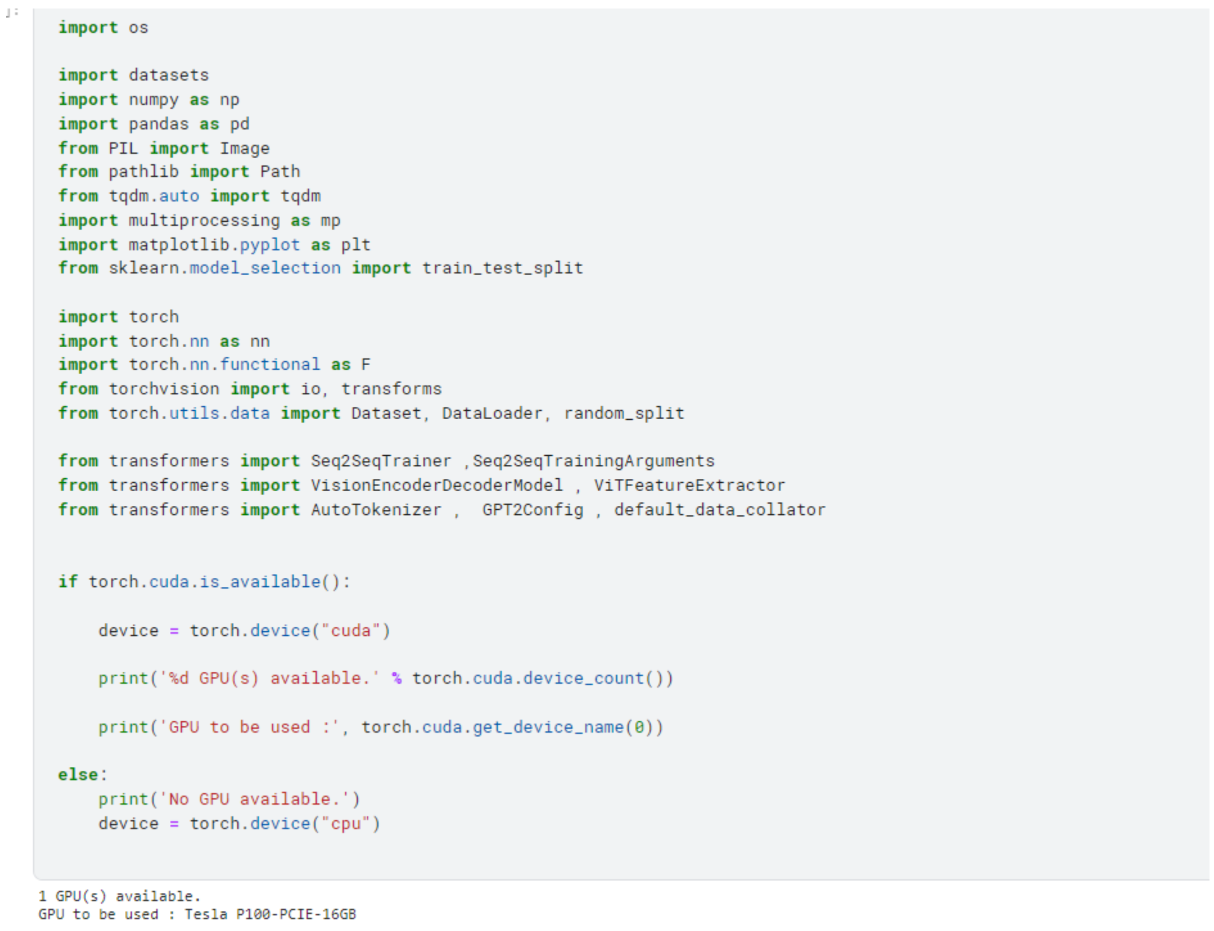
This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks. The model is best at what it was pre-trained for however, which is generating texts from a prompt.

CODE:

Imports and initialising the gpu



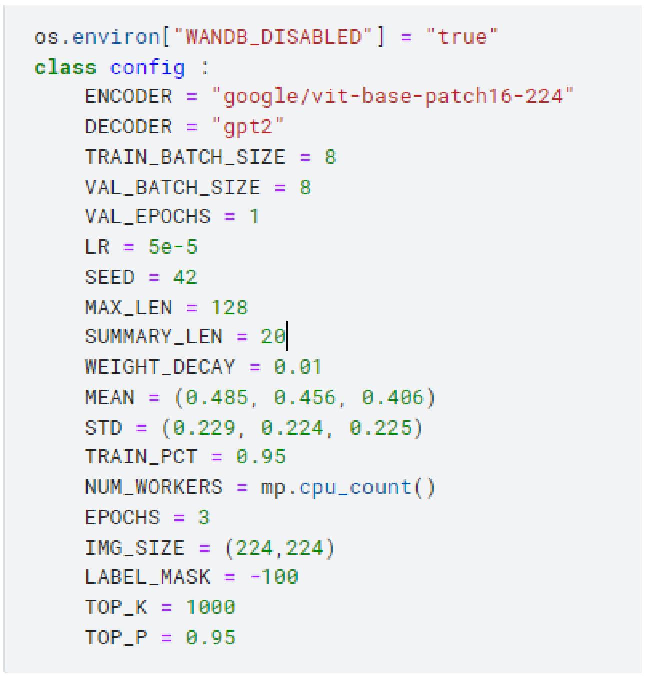
```
from IPython.display import clear_output
!pip install rouge_score -q
!pip install deep-phonemizer -q
clear_output()
```



```
import os

import datasets
import numpy as np
import pandas as pd
from PIL import Image
from pathlib import Path
from tqdm.auto import tqdm
import multiprocessing as mp
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision import io, transforms
from torch.utils.data import Dataset, DataLoader, random_split

from transformers import Seq2SeqTrainer ,Seq2SeqTrainingArguments
from transformers import VisionEncoderDecoderModel , ViTFeatureExtractor
from transformers import AutoTokenizer ,  GPT2Config , default_data_collator


if torch.cuda.is_available():

    device = torch.device("cuda")

    print('%d GPU(s) available.' % torch.cuda.device_count())

    print('GPU to be used :', torch.cuda.get_device_name(0))

else:
    print('No GPU available.')
    device = torch.device("cpu")
```

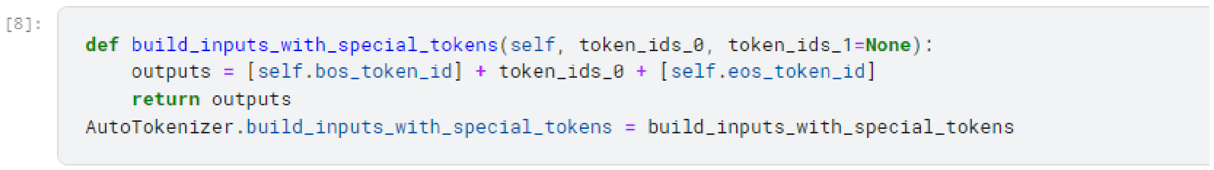
```
1 GPU(s) available.
GPU to be used : Tesla P100-PCIE-16GB
```

Hyperparameter Definitions:



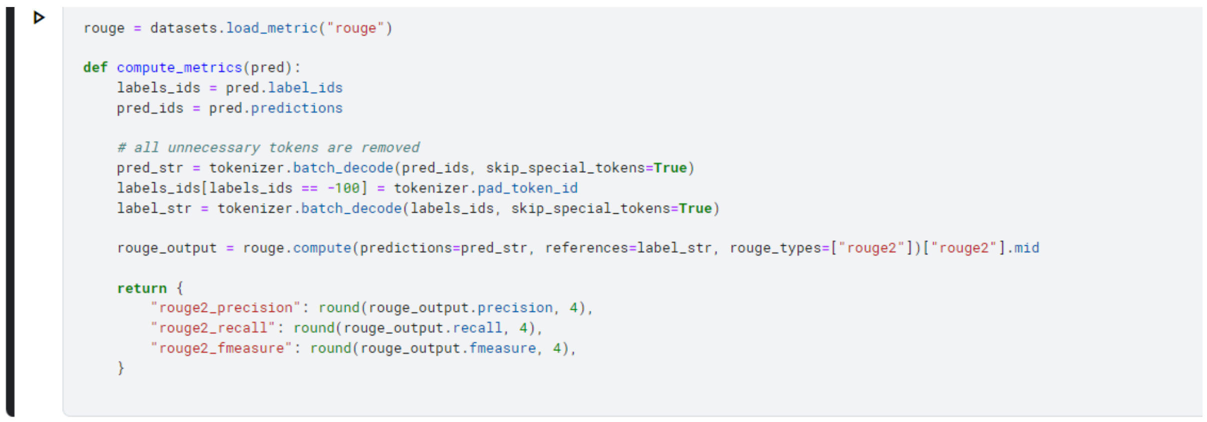
```
os.environ["WANDB_DISABLED"] = "true"
class config :
    ENCODER = "google/vit-base-patch16-224"
    DECODER = "gpt2"
    TRAIN_BATCH_SIZE = 8
    VAL_BATCH_SIZE = 8
    VAL_EPOCHS = 1
    LR = 5e-5
    SEED = 42
    MAX_LEN = 128
    SUMMARY_LEN = 20
    WEIGHT_DECAY = 0.01
    MEAN = (0.485, 0.456, 0.406)
    STD = (0.229, 0.224, 0.225)
    TRAIN_PCT = 0.95
    NUM_WORKERS = mp.cpu_count()
    EPOCHS = 3
    IMG_SIZE = (224,224)
    LABEL_MASK = -100
    TOP_K = 1000
    TOP_P = 0.95
```

Function definitions



```
[8]:
def build_inputs_with_special_tokens(self, token_ids_0, token_ids_1=None):
    outputs = [self.bos_token_id] + token_ids_0 + [self.eos_token_id]
    return outputs
AutoTokenizer.build_inputs_with_special_tokens = build_inputs_with_special_tokens
```

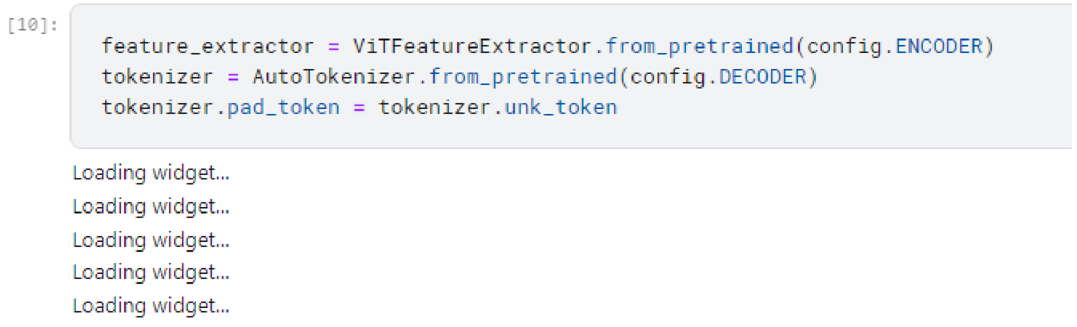
This function helps building special tokens while tokenising the captions.



```
rouge = datasets.load_metric("rouge")

def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    # all unnecessary tokens are removed
    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)

    rouge_output = rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge2"])["rouge2"].mid

    return {
        "rouge2_precision": round(rouge_output.precision, 4),
        "rouge2_recall": round(rouge_output.recall, 4),
        "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
    }
```

This function is used to compute the ROUGE-2 metrics.

Feature Extraction and Tokenisation:



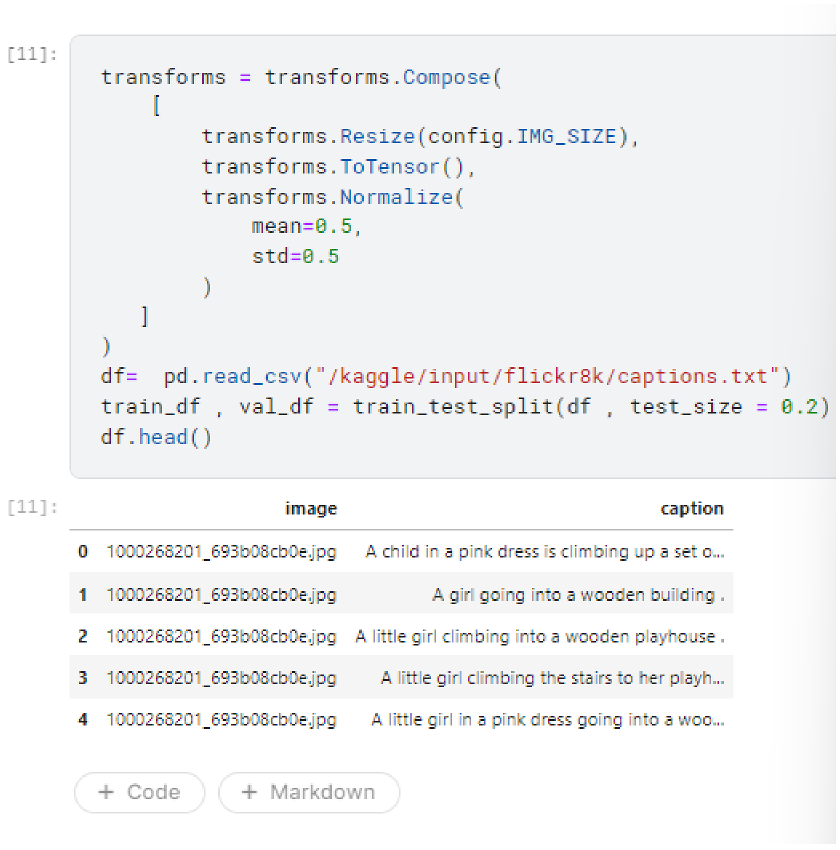
```
[10]:
feature_extractor = ViTFeatureExtractor.from_pretrained(config.ENCODER)
tokenizer = AutoTokenizer.from_pretrained(config.DECODER)
tokenizer.pad_token = tokenizer.unk_token
```

```
Loading widget...
Loading widget...
Loading widget...
Loading widget...
Loading widget...
```

The Feature extractor is loaded using **ViTFeatureExtractor**.

The tokenizer for GPT2 is loaded using the **AutoTokenizer**

Transforms and Dataframe formation



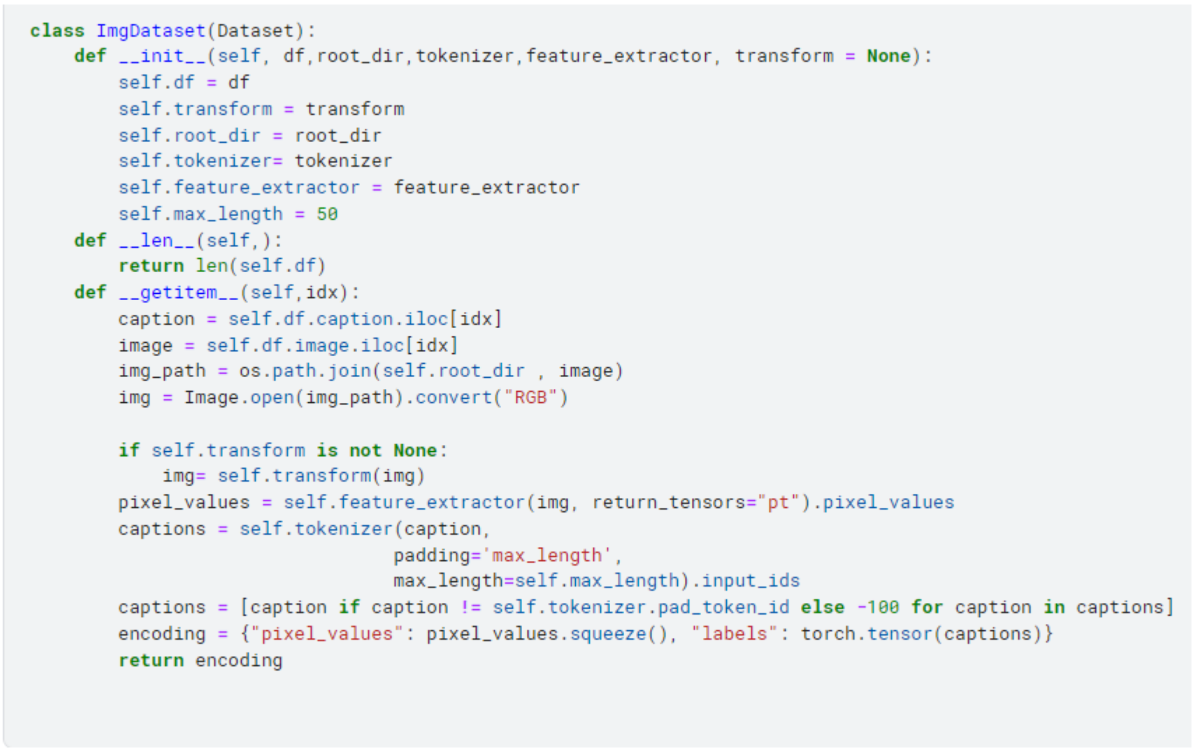
```
[11]:
transforms = transforms.Compose(
    [
        transforms.Resize(config.IMG_SIZE),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=0.5,
            std=0.5
        )
   ]
)
df=  pd.read_csv("/kaggle/input/flickr8k/captions.txt")
train_df , val_df = train_test_split(df , test_size = 0.2)
df.head()
```

[11]:

	image	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

The Transformations used are

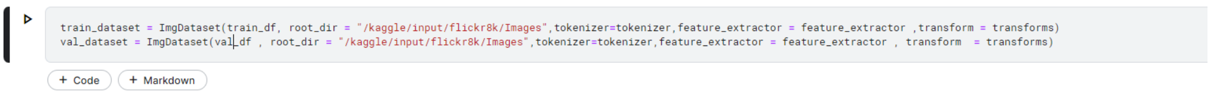
1. **Resizing** the image to (224,224)
2. **Normalizing** the image
3. Converting the image to **Tensor**



```
class ImgDataset(Dataset):
    def __init__(self, df,root_dir,tokenizer,feature_extractor, transform = None):
        self.df = df
        self.transform = transform
        self.root_dir = root_dir
        self.tokenizer= tokenizer
        self.feature_extractor = feature_extractor
        self.max_length = 50
    def __len__(self,):
        return len(self.df)
    def __getitem__(self,idx):
        caption = self.df.caption.iloc[idx]
        image = self.df.image.iloc[idx]
        img_path = os.path.join(self.root_dir , image)
        img = Image.open(img_path).convert("RGB")

        if self.transform is not None:
            img= self.transform(img)
        pixel_values = self.feature_extractor(img, return_tensors="pt").pixel_values
        captions = self.tokenizer(caption,
                                  padding='max_length',
                                  max_length=self.max_length).input_ids
        captions = [caption if caption != self.tokenizer.pad_token_id else -100 for caption in captions]
        encoding = {"pixel_values": pixel_values.squeeze(), "labels": torch.tensor(captions)}
        return encoding
```

We read the image using the Image function of PIL library
The image is transformed using the transformed defined above
The transformed image is passed through the feature extractor to extract the pixel values from the image
The captions are loaded from the dataframe
The captions are tokenized
The tokenized captions are padded to max length
The images and tokenized captions are returned

Formation of training set and validation set



```
train_dataset = ImgDataset(train_df, root_dir = "/kaggle/input/flickr8k/Images",tokenizer=tokenizer,feature_extractor = feature_extractor ,transform = transforms)
val_dataset = ImgDataset(val_df , root_dir = "/kaggle/input/flickr8k/Images",tokenizer=tokenizer,feature_extractor = feature_extractor , transform  = transforms)
```

Model Initialisation

```
model = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(config.ENCODER, config.DECODER)
```

```
model.config.decoder_start_token_id = tokenizer.cls_token_id
model.config.pad_token_id = tokenizer.pad_token_id
# make sure vocab size is set correctly
model.config.vocab_size = model.config.decoder.vocab_size
# set beam search parameters
model.config.eos_token_id = tokenizer.sep_token_id
model.config.decoder_start_token_id = tokenizer.bos_token_id
model.config.max_length = 128
model.config.early_stopping = True
model.config.no_repeat_ngram_size = 3
model.config.length_penalty = 2.0
model.config.num_beams = 4
```

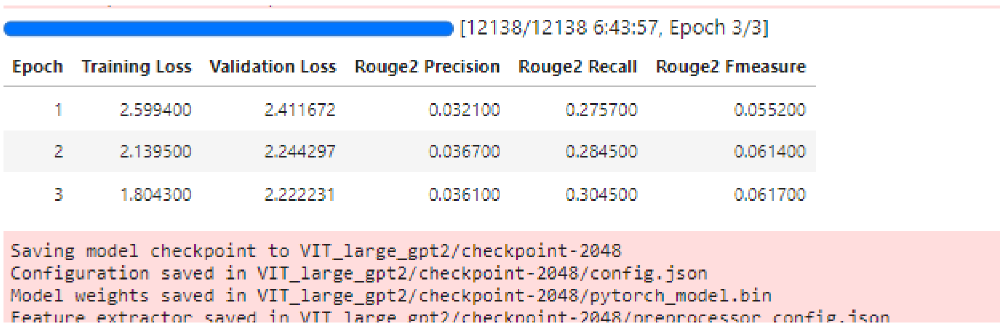
Definition of training arguments

```
training_args = Seq2SeqTrainingArguments(
    output_dir='VIT_large_gpt2',
    per_device_train_batch_size=config.TRAIN_BATCH_SIZE,
    per_device_eval_batch_size=config.VAL_BATCH_SIZE,
    predict_with_generate=True,
    evaluation_strategy="epoch",
    do_train=True,
    do_eval=True,
    logging_steps=1024,
    save_steps=2048,
    warmup_steps=1024,
    learning_rate = 5e-5,
    #max_steps=1500, # delete for full training
    num_train_epochs = config.EPOCHS, #TRAIN_EPOCHS
    overwrite_output_dir=True,
    save_total_limit=1,
)
```

Training using Seq2Seq Trainer:

```
# instantiate trainer
trainer = Seq2SeqTrainer(
    tokenizer=feature_extractor,
    model=model,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    data_collator=default_data_collator,
)
trainer.train()
```

Results:



[12138/12138 6:43:57, Epoch 3/3]

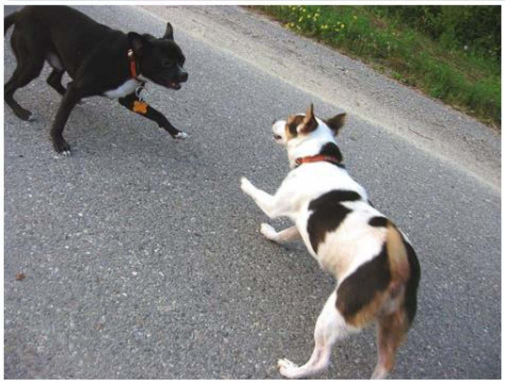
Epoch	Training Loss	Validation Loss	Rouge2 Precision	Rouge2 Recall	Rouge2 Fmeasure
1	2.599400	2.411672	0.032100	0.275700	0.055200
2	2.139500	2.244297	0.036700	0.284500	0.061400
3	1.804300	2.222231	0.036100	0.304500	0.061700

```
Saving model checkpoint to VIT_large_gpt2/checkpoint-2048
Configuration saved in VIT_large_gpt2/checkpoint-2048/config.json
Model weights saved in VIT_large_gpt2/checkpoint-2048/pytorch_model.bin
Feature extractor saved in VIT_large_gpt2/checkpoint-2048/preprocessor_config.json
```

Output:

```
img =  Image.open("/kaggle/input/flickr8k/Images/1001773457_577c3a7d70.jpg").convert("RGB")
img
```

[22]:



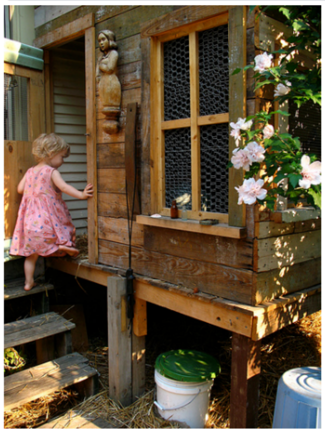
[23]:

```
generated_caption = tokenizer.decode(model.generate(feature_extractor(img, return_tensors="pt").pixel_values.to("cuda"))[0])
print('\033[96m' +generated_caption[:85]+ '\033[0m')
```

```
<|endoftext|>A black dog and a white dog are running down the street... the black dog
```

```
img =  Image.open("/kaggle/input/flickr8k/Images/1000268201_693b08cb0e.jpg").convert("RGB")
img
```

[24]:



[27]:

```
generated_caption = tokenizer.decode(model.generate(feature_extractor(img, return_tensors="pt").pixel_values.to("cuda"))[0])
print('\033[96m' +generated_caption[:100]+ '\033[0m')
```

```
<|endoftext|>A little girl in a pink dress is sitting on a wooden bench in front of a wooden house.
```