def preprocess_image(image_path):

image = cv2.imread(image_path)

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)

return blurred_image

- The function is named preprocess_image and it takes one argument, the image_path
- The openCV library is used to convert the image to an array consisting of pixel values, where each pixel value is a tuple of 3 values for B,G,R intensities
- The cvtColour method is used to convert the coloured image to gray scale.
- So now, gray_image consistes of pixel values of grayscale intensities
- A blurred_image is obtained using Gaussian blur which reduces the noise in the image, and makes the transition of colours smoother.
- Gaussian blur is a technique where a matrix/kernel of size 5x5 (can be anything) consists of a set of values, and this kernel is applied to each pixel, where the values are multiplied and summed. This ensures that neighbouring pixels are given higher importance that farther away pixels

def extract_text(image):

return pytesseract.image_to_string(image)

- The function is named extract_text and it takes the value returned by the previous function (blurred_image) as the argument.
- With the help of Tesseract OCR (Optical Character Recognition), this function identifies the text from the image.
- Tesseract is an open source OCR engine mainted by Google which can identify text.
- OCR works using the following steps: Preprocessing (noise reduction, binarization), Character Segmentation (isolating individual characters from the text) and Character Recognition (identifies segmented character using pattern matching, ML).

def image_caption(image_paths):

model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

images = []

for image_path in image_paths:

i_image = Image.open(image_path)

if i_image.mode != "RGB":

i_image = i_image.convert(mode="RGB")

images.append(i_image)

pixel_values = feature_extractor(images=images, return_tensors="pt").pixel_values

pixel_values = pixel_values.to(device)

output_ids = model.generate(pixel_values)

preds = tokenizer.batch_decode(output_ids, skip_special_tokens=True)

preds = [pred.strip() for pred in preds]

return preds

- The function image_caption is used to generate captions for the images that we give as an input.
- We are making use of two open source models available on the Hugging Face library 'transformers'– ViT (Vision Transformer) and GPT-2.
- The first three lines of code load the models, EncoderDecoder, FeatureExtractor and Tokenizer:

model = VisionEncoderDecoderModel.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

feature_extractor = ViTFeatureExtractor.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

tokenizer = AutoTokenizer.from_pretrained("nlpconnect/vit-gpt2-image-captioning")

- EncoderDecoder is used for generating captions, FeatureExtractor is used to transform image into model compatible format and Tokenizer is used to convert model output to human readable format.
- These lines are used to test if the devise is running on CPU or GPU, so that it can appropriately optimize the functioning:

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

- The input image is then checked if it is in RGB format, if not it is converted.
- The features are extracted from the image and then they are passed to the model to predict the caption. The output is obtained after passing through a tokenizer which can convert it to human readable format.

def llm(text, desc):

tokenizer = BartTokenizer.from_pretrained("facebook/bart-large-cnn")

model = BartForConditionalGeneration.from_pretrained("facebook/bart-large-cnn")

text_to_summarize = desc + "\n" + text

input_ids = tokenizer.encode("What does the image depict?" + text_to_summarize, return_tensors="pt", max_length=1024, truncation=True)

summary_ids = model.generate(input_ids, max_length=150, num_beams=4, length_penalty=2.0, min_length=30, early_stopping=True)

return tokenizer.decode(summary_ids[0], skip_special_tokens=True)

- BART (Bidirectional and Auto-Regressive Transformers) is an LLM model trained by Facebook.
- A tokenizer (converting text to model readable format and vice versa) and the model are loaded
- A part of the prompt is the text extracted from the image and the image caption which are concatenated. This is appended to the main prompt (“What does the image describe?”) and is passed to the tokenizer so that it can convert it into model readable format.
- These tokens (named as input_ids) are passed to the model to obtain the caption, which is then decoded using the decoder.

def translate_text(text, target_language):

translator = Translator(to_lang=target_language)

return translator.translate(text)

- A Translator is imported from the translate library. Although this library is open source, it makes use of APIs
- The translator is initialized with the language and then the text is passed to the translator.
- The output is printed.