UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Introduction to Deep Learning
Final Report

Group 3 – ICT

Topic 9: Study DEtection TRansformer (DETR) to detect buildings from remote sensing images with and without histogram equalization

Members

BI11-110 Phùng Gia Huy
BI11-195 Nguyễn Hoài Nam
BI11-198 Hoàng Đức Nghĩa
BI11-273 Nguyễn Đăng Trung
BI11-286 Nguyễn Xuân Vinh

Lecturer

Dr. Nghiêm Thị Phương

March, 2023

Contents

I. Introduction

In recent years, the development of deep learning methods has revolutionized the field of computer vision, allowing for the automated detection and analysis of objects in images and videos. One important application of these methods is in the field of remote sensing, where satellite and aerial images are used to monitor and map the Earth's surface. One key task in remote sensing is the detection of buildings, which can provide valuable information for urban planning, disaster response, and environmental monitoring.

DEtection TRansformer (DETR) is a state-of-the-art object detection architecture that has recently gained widespread attention due to its end-to-end design and ability to generate fixed number of object detections directly from the input image, without the need for region proposal methods. In this project, we aim to investigate the performance of DETR for the task of building detection in remote sensing images. Specifically, we will compare the performance of DETR with and without histogram equalization, a commonly used preprocessing step to enhance the contrast and visibility of images.

Histogram equalization is a technique used to redistribute the pixel intensities in an image to enhance its contrast and visibility. It has been shown to be effective for a wide range of computer vision tasks, including object detection and segmentation. However, its effectiveness for remote sensing images is not well understood, and its use may depend on the characteristics of the dataset and the performance of the model.

In this report, we present our technique for training and evaluating DETR on a remote sensing dataset for the task of building detection. We compare the performance of DETR with and without histogram equalization, and provide an analysis of the results. We also discuss the implications of our findings for future research in the field of remote sensing and object detection.

II. Motivation

The automated detection of buildings in remote sensing images is a challenging task that has important applications in a range of fields, from urban planning and disaster response to environmental monitoring and national security. While there have been significant advances in the field of object detection in recent years, the specific challenges posed by remote sensing imagery, such as complex background clutter, varying illumination and weather conditions, and scale variations, continue to present significant obstacles to accurate and reliable detection.

One promising approach to building detection in remote sensing imagery is the use of deep learning methods, which have shown impressive performance on a range of object detection tasks in other domains. Among these methods, DEtection TRansformer (DETR) has emerged as a particularly promising architecture due to its end-to-end design and ability to generate a fixed number of object detections directly from the input image, without the need for region proposal methods.

However, despite the potential of deep learning methods for building detection in remote sensing imagery, there are still significant challenges that need to be addressed. One important factor is the impact of preprocessing techniques on the performance of the model. In particular, histogram equalization has been shown to be an effective technique for enhancing the contrast and visibility of images for a range of computer vision tasks, including object detection. However, its effectiveness for remote sensing imagery is not well understood, and it is unclear how it may affect the performance of DETR for building detection.

Therefore, the motivation for this study is to investigate the performance of DETR for building detection in remote sensing imagery, and to compare the performance of the model with and without the use of histogram equalization as a preprocessing step. By doing so, we hope to gain insights into the factors that influence the performance of deep learning methods for building detection in remote sensing imagery, and to provide guidance for future research in this important area.

III. Project Implementation

1. Methodology

In this project, we use "Google Colab" with Python programming language to implement our code. There are several benefits of using Google Colab, which is a cloud-based platform for developing and running deep learning models:

- Free to use: Google Colab is free to use and does not require any installation or setup. All we need is a Google account and a web browser to access the platform.
- Powerful hardware: Google Colab provides access to powerful hardware such as GPUs and TPUs, which can accelerate the training and inference of deep learning models.
- Collaborative features: Google Colab allows us to share our notebooks with others and work collaboratively in real-time. This is especially useful when working on group projects or when seeking feedback and help from others.
- Large ecosystem: Google Colab supports a wide range of libraries and frameworks for deep learning and data science, such as TensorFlow, PyTorch, and scikit-learn. It also integrates with other Google services such as Google Drive, which makes it easy to store and access data.
- Easy to use: Google Colab provides an intuitive and user-friendly interface that makes it easy to write and execute code, visualize data, and monitor training progress.

Generally speaking, Google Colab is a convenient and powerful platform for deep learning that can help accelerate the development and deployment of models.

2. Collecting dataset

We collect the dataset in this link https://ieee-dataport.org/competitions/2020-ieee-grss-data-fusion-contest?fbclid=IwAR3Fmz2UIbW_b10E1NYh3GRqpf52w18hVzTe8QOVm_ecBRCbQvpjoFxCAXA

Here is some information about the datasets we are going to use in this project.

The Data

In the contest, we use the SEN12MS dataset [1] (https://mediatum.ub.tum.de/1474000) for training the land cover prediction models. This publicly available dataset includes more than 180,000 triplets of corresponding Sentinel-1 SAR data, Sentinel-2 multispectral imagery, and MODIS-derived land cover maps sampled across the entire globe. While all data are provided at a ground sample distance (GSD) of 10m, the Sentinel images have a native resolution of about 10-20m per pixel, whereas the MODIS-derived land cover has a native resolution of 500m per pixel. The main challenge, therefore, is to train powerful models for high-resolution land cover prediction (target resolution: 10m) from noisy, low-resolution annotations. For validation and testing, semi-manually derived high-resolution land cover maps of scenes with undisclosed geolocation and not contained in the SEN12MS dataset are used. The validation and test datasets will be distributed step-by-step at IEEE DataPort (https://ieee-dataport.org/competitions/2020-ieee-grss-data-fusion-contest).

Sentinel-1 and Sentinel-2 satellite data

180,662 patch pairs of corresponding Sentinel-1 dual-pol SAR data and Sentinel-2 multispectral images. In detail:

- **Sentinel-1 SAR:** 2 channels corresponding to sigma nought backscatter values in dB scale for VV and VH polarization.
- **Sentinel-2 Multispectral:** 13 channels corresponding to the 13 spectral bands (B1, B2, B3, B4, B5, B6, B7, B8, B8a, B9, B10, B11, B12).

The patches comprising the dataset are distributed across the land masses of the Earth and spread over all four meteorological seasons. This is reflected by the dataset structure. All patches are provided in the form of 16-bit GeoTiffs with a GSD of 10m.

MODIS-derived land cover labels

Routinely, the data of the MODIS instruments onboard the Terra and Aqua satellites of NASA are used to derive land cover maps at yearly intervals using six different classification schemes. The final land cover maps undergo additional post-processing including prior knowledge and ancillary information and come at a resolution of 500m per pixel.

By default, SEN12MS contains a 4-layer image of MODIS-derived land cover maps oversampled to a pixel spacing of 10m for every Sentinel-1/Sentinel-2 patch pair, according to the following classification schemes: IGBP [2], and LCCS Land Cover, Land Use, and Surface Hydrology [3].

For the contest, a simplified version of the IGBP classification scheme is used. More details can be found at http://www.grss-ieee.org/community/technical-committees/data-fusion/

In order to load the SEN12MS training data in a form compatible to the contest (i.e., Sentinel-1, Sentinel-2, and 500m-resolution land cover labels following the simplified IGBP scheme), a Python-based data loader is provided at IEEE DataPort.

The validation images are provided in the same format that also this data loader prepares, but the semi-manually created land cover labels have a GSD of 10 m.

Note again: For the sake of convenience, ALL data follow a 10m-per-pixel sampling, regardless of their resolution.

After that, we install two files .zip: “s2_0.zip (3.5 GB)” and “s2_validation.zip (603.47 MB)”.

When we extract s2_0.zip and s2_validation, we receive the folder with roughly more than 5000 files .tif or .tiff and the folder with about nearly 1000 files .tif or .tiff

In this step, we try to follow the lecturer’s instructions.

3. DEtection TRansformer (DETR)

At first, all the files in two following datasets are .tif or .tiff files. As usual, we can not do the project with .tif or .tiff files. Therefore, we need to transfer to .jpg files to implement the project. In order to do that, we use Adobe Photoshop to transfer approximately 7000 .tif or .tiff files in total to .jpg files. This task cost us too much time.

* With Histogram Equalization – *s2_validation*

Firstly, we need to import the necessary libraries for image processing using PyTorch, OpenCV, and PIL (Python Imaging Library). We also import the requests library for fetching images from URLs.

```
import torch as th
import torchvision.transforms as T
import requests
from PIL import Image, ImageDraw, ImageFont, ImageOps
import numpy as np
import cv2
```

After that, we need to load a pre-trained object detection model called "DETR" from Facebook Research, using the "detr_resnet101" variant.

```
# Load the pre-trained DETR model and move it to the GPU
model = th.hub.load('facebookresearch/detr', 'detr_resnet101', pretrained=True)
model.eval()
model = model.cuda()
```

```
Using cache found in /root/.cache/torch/hub/facebookresearch_detr_main
```

The **load()** function from PyTorch Hub is used to download the pre-trained model and return an instance of the model. The model is then set to evaluation mode using **model.eval()**. Finally, the model is moved to the GPU using **model.cuda()**, which suggests that the code expects to perform inference on the GPU.

Then, we have to define the image transformation pipeline and list of classes for object detection.

```
# Define the input image transformation pipeline
transform = T.Compose([
    T.ToTensor(), # Convert the image to a PyTorch tensor
    T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]) # Normalize the tensor with mean and standard deviation values
])

# Define the list of classes for object detection
CLASSES = ['building']
```

The image transformation pipeline is defined using **torchvision.transforms.Compose()**. It composes a list of image transformations that will be applied sequentially to the input image. In this case, there are two transformations applied. First, **ToTensor()** is used to convert the image to a PyTorch tensor. Second, **Normalize()** is used to normalize the tensor with mean and standard deviation values. Normalization is a common preprocessing step in deep learning to standardize the input data and make it easier to train the model. The **CLASSES** list defines the classes that the model will be trained to detect. In this case, there is only one class, which is "building".

Next, we have to locate just one image file in *s2_validation* in the local directory of the Colab notebook.

```
url = "/content/ROIs0000_validation_s2_0_p0.jpg"
```

This line of code is assigning a string value to the variable **url**. The string represents the URL or file path of an image that will be used for object detection.

After that, we need to load an image from a local file system and resize it to (800,1333) pixels while converting it to RGB format. The code then checks if the mean of the pixel values is either less than 10 or greater than 245, which indicates a low-quality image. If the image is low quality, bilateral filtering is applied to remove noise. Bilateral filtering is a nonlinear image filtering technique used in image processing and computer vision to smooth images while preserving edges and details. It is an extension of the classical Gaussian filter that considers only the spatial distance between pixels. In addition to spatial distance, bilateral filtering also considers the differences in pixel intensities. Finally, histogram equalization is applied to enhance the contrast of the image. The resulting image is returned.

```
# img = Image.open(requests.get(url, stream=True).raw).resize((800,1333)).convert('RGB')
img = Image.open(url).resize((800,1333)).convert('RGB')
# Check if the image is low quality
if np.mean(img) < 10 or np.mean(img) > 245:
    print(f"Processing low quality image: {img}")
    # Apply bilateral filtering to remove noise
    img_array = np.array(img)
    filtered = cv2.bilateralFilter(img_array, 9, 75, 75)
    img = Image.fromarray(filtered)
# apply histogram equalization
img = ImageOps.equalize(img)
img
```

```
Processing low quality image: <PIL.Image.Image image mode=RGB size=800x1333 at 0x7FBD0B93BD30>
```

Then, we apply the input image transformation pipeline defined earlier to the **img** variable, which is a PIL Image object.

```
# Transform the image using the defined pipeline and move it to the GPU
img_tens = transform(img).unsqueeze(0).cuda()
```

The pipeline converts the image to a PyTorch tensor and normalizes it with mean and standard deviation values. Then, the **unsqueeze()** method is used to add an extra dimension to the tensor, making it a 4D tensor. This is because the model expects a batch of images as input, even if there's only one image in the batch. Finally, the tensor is moved to the GPU by calling the **cuda()** method.

Next, we are going to use the pre-trained DETR model to generate predictions for the input image. The image has already been transformed using the defined pipeline and moved to the GPU. The **with torch.no_grad()** statement indicates that the forward pass through the model should not be considered for gradient calculation (since we are not backpropagating through the model).

```
# Use the DETR model to generate predictions for the input image
with th.no_grad():
  output = model(img_tens)
```

The **output** variable holds the model's predictions for the input image. Specifically, **output** is a dictionary with the following keys:

- **pred_boxes**: A tensor of shape (**batch_size**, **num_queries**, **4**) containing the predicted bounding boxes for all objects in the image, where **batch_size** is 1, and **num_queries** is the number of object detection queries (which is fixed for the DETR model).
- **pred_logits**: A tensor of shape (**batch_size**, **num_queries**, **num_classes**) containing the predicted class scores for all objects in the image, where **batch_size** is 1, and **num_queries** is the number of object detection queries (which is fixed for the DETR model), and **num_classes** is the total number of classes including the background class.

After that, we are going to select the predicted logits and boxes with the highest confidence scores from the DETR output.

```
# Create a copy of the input image for drawing bounding boxes and labels
im2 = img.copy()

# Create a drawing context for the output image
drw = ImageDraw.Draw(im2)

# Get the predicted logits and boxes from the DETR output
pred_logits=output['pred_logits'][0][:, :len(CLASSES)]
pred_boxes=output['pred_boxes'][0]

# Find the predicted objects with the highest confidence scores
max_output = pred_logits.softmax(-1).max(-1)
topk = max_output.values.topk(15)

# Select the predicted logits and boxes for the top predicted objects
pred_logits = pred_logits[topk.indices]
pred_boxes = pred_boxes[topk.indices]
pred_logits.shape
```

```
torch.Size([15, 1])
```

First, a copy of the input image is created to draw the bounding boxes and labels on. Then, the predicted logits and boxes are extracted from the DETR output using **output['pred_logits'][0][:, :len(CLASSES)]** and **output['pred_boxes'][0]** respectively. Next, the confidence scores are calculated for each predicted object using **max_output = pred_logits.softmax(-1).max(-1)**. The **softmax()** function is applied along the last dimension of the **pred_logits** tensor to obtain the predicted probabilities for each class. The **max()** function is then used to find the maximum probability and its index along the last dimension of the **softmax()** output. This results in a tensor with the maximum probability and its corresponding index for each predicted object. The **topk** tensor is then obtained using **topk = max_output.values.topk(15)**, which selects the top 15 predicted objects with the highest confidence scores. Finally, the predicted logits and boxes for the top predicted objects are selected using **pred_logits = pred_logits[topk.indices]** and **pred_boxes = pred_boxes[topk.indices]** respectively. The shape of the **pred_logits** tensor is printed using **pred_logits.shape** to confirm the shape of the tensor.

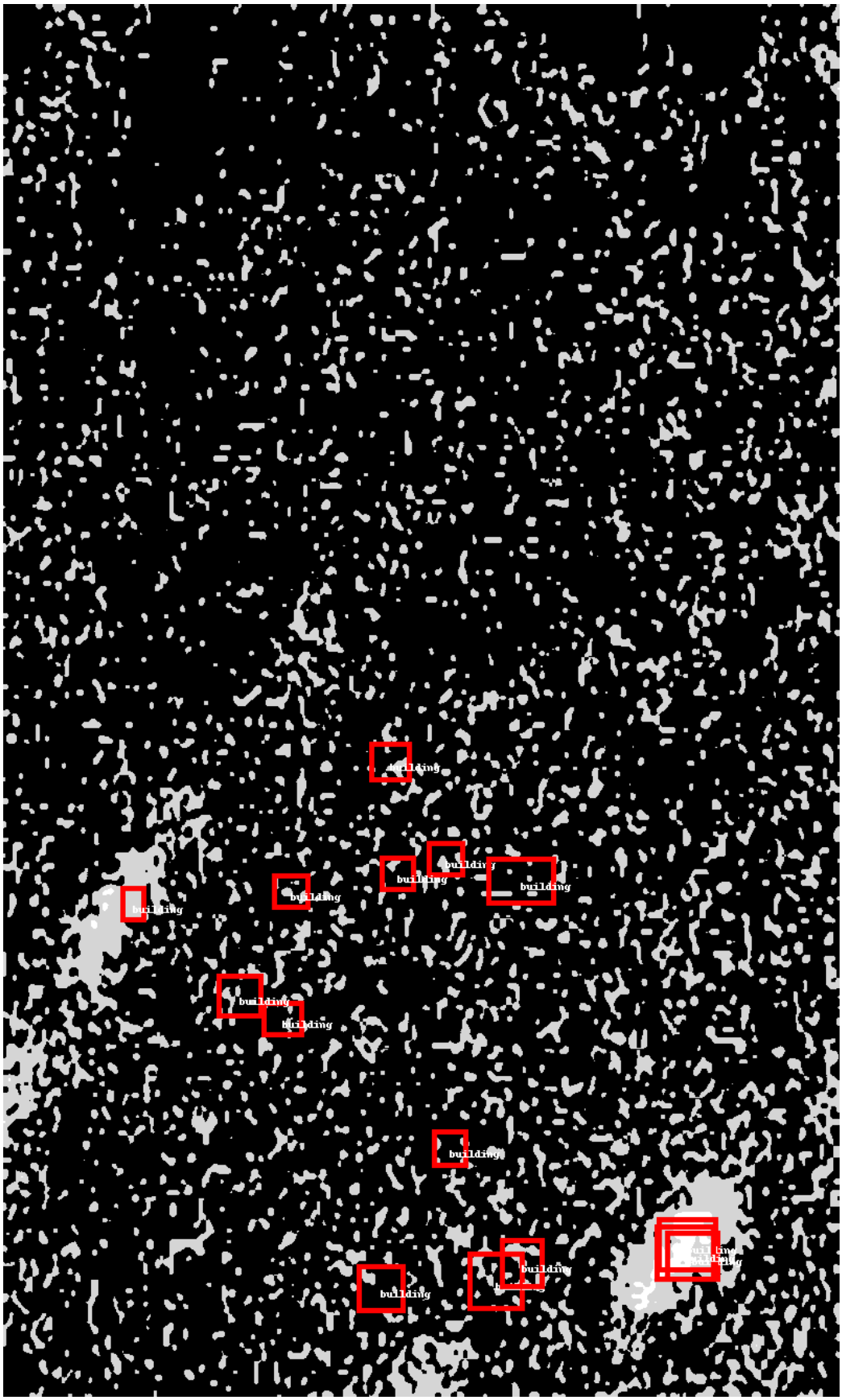
Then, we are anticipated to be responsible for drawing bounding boxes and labels around the predicted objects on the input image.

```
# Loop over the selected logits and boxes and draw bounding boxes and labels for each one
for logits, box in zip(pred_logits, pred_boxes):
  cls = logits.argmax()
  if cls >= len(CLASSES):
    continue
  label = CLASSES[cls]
  # Rescale the box coordinates to match the size of the original image
  box = box.cpu() * th.Tensor([800, 600, 800, 600])
  x, y, w, h = box
  x0, x1 = x-w//2, x+w//2
  y0, y1 = y-h//2, y+h//2
  # Draw a red rectangle around the object
  drw.rectangle([x0, y0, x1, y1], outline='red', width=5)
  # Add the class label to the image
  drw.text((x, y), label, fill='white')
```

It loops over the selected logits and boxes (i.e., predictions with the highest confidence scores), and for each one, it extracts the class label and the bounding box coordinates. The box coordinates are then rescaled to match the size of the original image, and a red rectangle is drawn around the object using the **rectangle()** method of the **ImageDraw** class. Additionally, the class label is added to the image using the **text()** method of the **ImageDraw** class, and the text is colored white using the **fill** parameter. Finally, the modified image is returned with the bounding boxes and labels drawn.

Ultimately, we have the final result of the first image in *s2_validtion*:

```
[ ]  im2
```


building
building
building
building
building
building
building
building
building
building
building
building
building

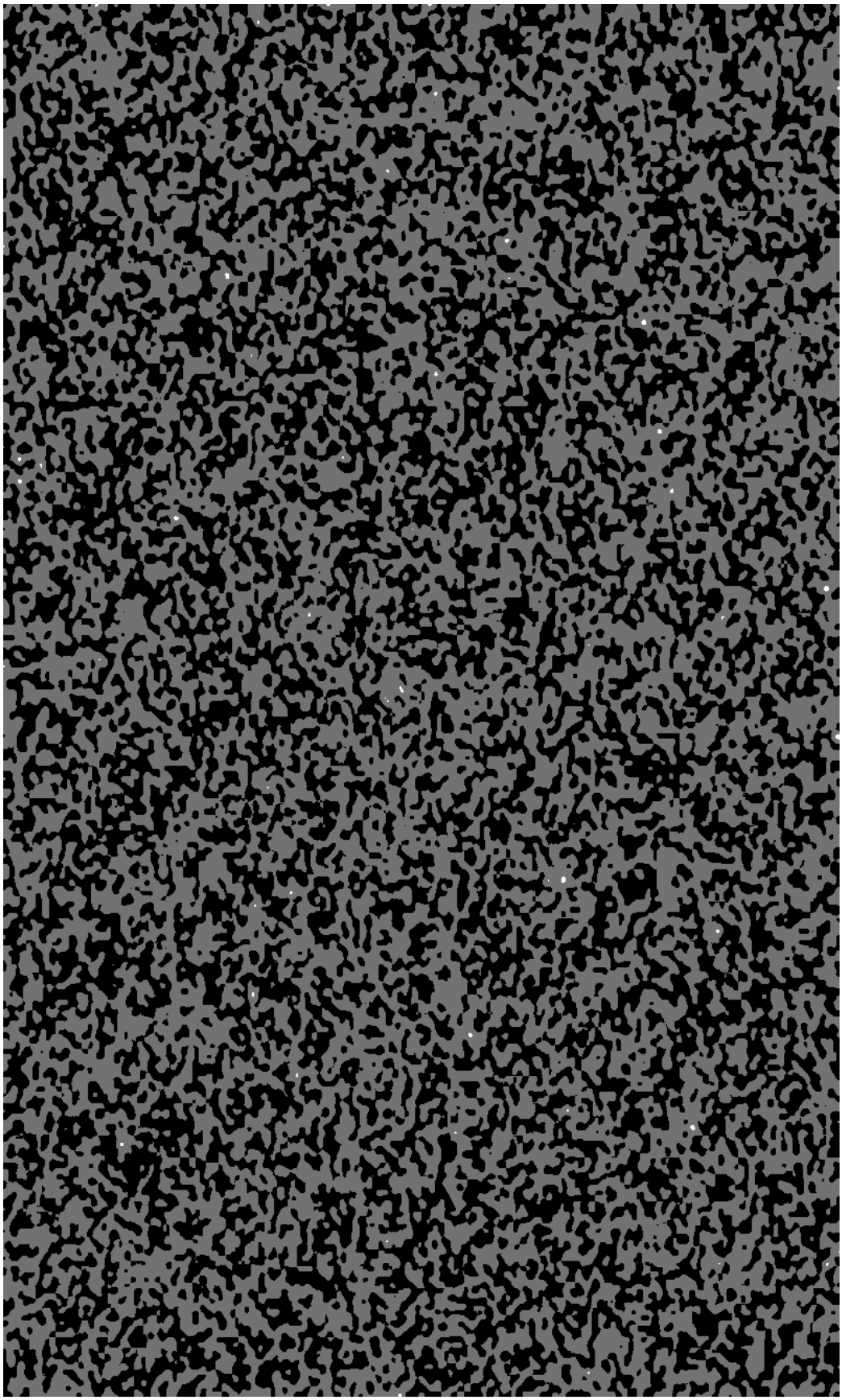
As can be seen apparently, by using the DEtection Transformer (DETR), we are able to detect buildings from just one remote sensing image in *s2_validation* with histogram equalization.

* With Histogram Equalization – *s2_0*

By applying the same steps as Histogram Equalization – *s2_validation* with just one image in *s2_0*, we could have the result that we are also able to detect buildings from just one remote sensing image in *s2_0* with histogram equalization:

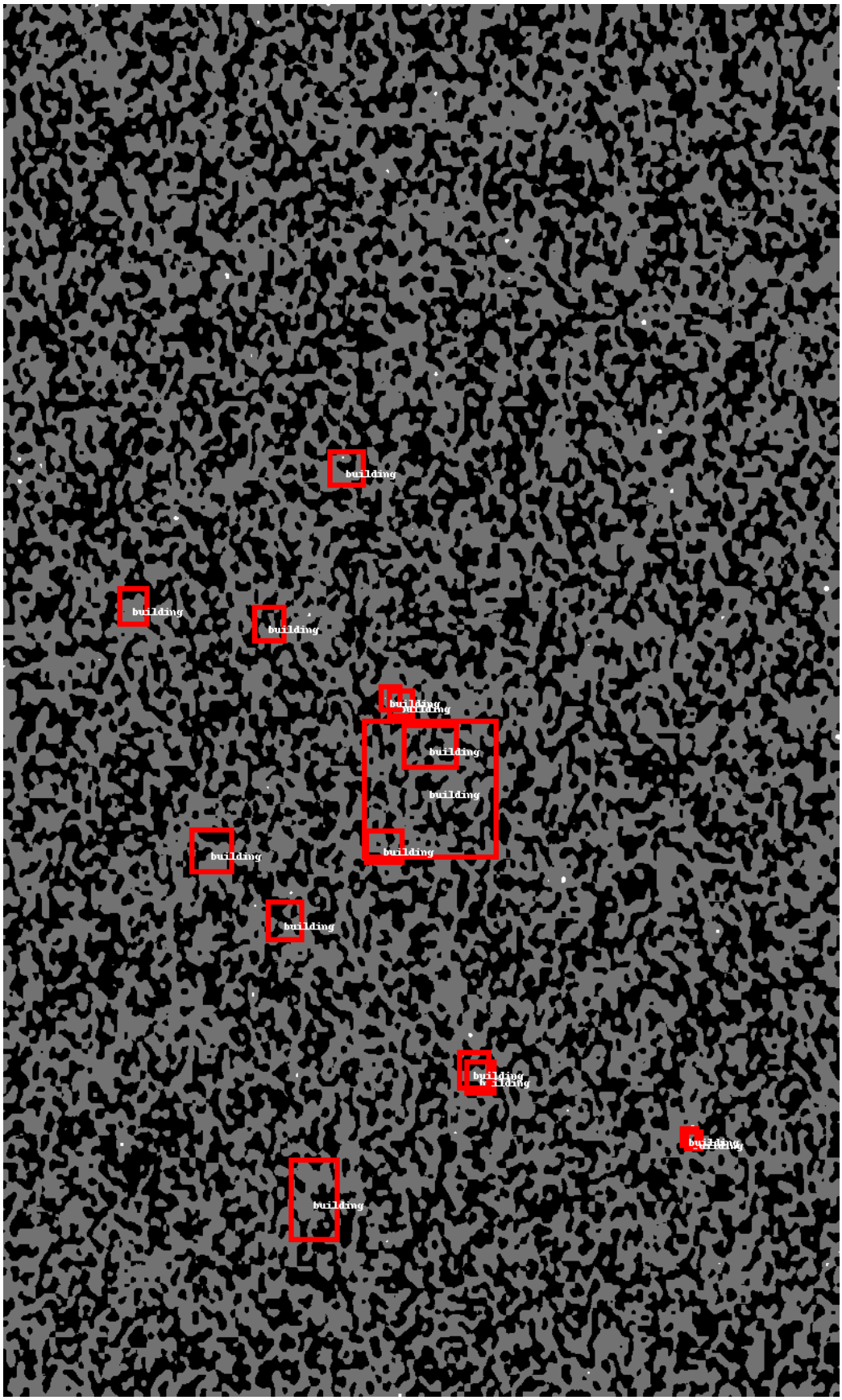
```
model = th.hub.load('facebookresearch/detr', 'detr_resnet101', pretrained=True)
model.eval()
model = model.cuda()

# standard PyTorch mean-std input image normalization
transform = T.Compose([
    T.ToTensor(),
    T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

CLASSES = ['building']

url = "/content/ROIs0000_test_s2_0_p8.jpg"
img = Image.open(url).resize((800,1333)).convert('RGB')
if np.mean(img) < 10 or np.mean(img) > 245:
    print(f"Processing low quality image: {img}")

    # Apply bilateral filtering to remove noise
    img_array = np.array(img)
    filtered = cv2.bilateralFilter(img_array, 9, 75, 75)
    img = Image.fromarray(filtered)
img = ImageOps.equalize(img)
img
```

```
Using cache found in /root/.cache/torch/hub/facebookresearch_detr_main
Processing low quality image: <PIL.Image.Image image mode=RGB size=800x1333 at 0x7FBD0F950F70>
```



```
img_tens = transform(img).unsqueeze(0).cuda()

with th.no_grad():
  output = model(img_tens)

im2 = img.copy()
drw = ImageDraw.Draw(im2)
pred_logits=output['pred_logits'][0][:, :len(CLASSES)]
pred_boxes=output['pred_boxes'][0]

max_output = pred_logits.softmax(-1).max(-1)
topk = max_output.values.topk(15)

pred_logits = pred_logits[topk.indices]
pred_boxes = pred_boxes[topk.indices]
pred_logits.shape

for logits, box in zip(pred_logits, pred_boxes):
  cls = logits.argmax()
  if cls >= len(CLASSES):
    continue
  label = CLASSES[cls]
  print(label)
  box = box.cpu() * th.Tensor([800, 1333, 800, 1333])
  x, y, w, h = box
  x0, x1 = x-w//2, x+w//2
  y0, y1 = y-h//2, y+h//2
  drw.rectangle([x0, y0, x1, y1], outline='red', width=5)
  drw.text((x, y), label, fill='white')
im2
```

```
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
```


building
building
building
building
building
building
building
building
building
building
building
building
building

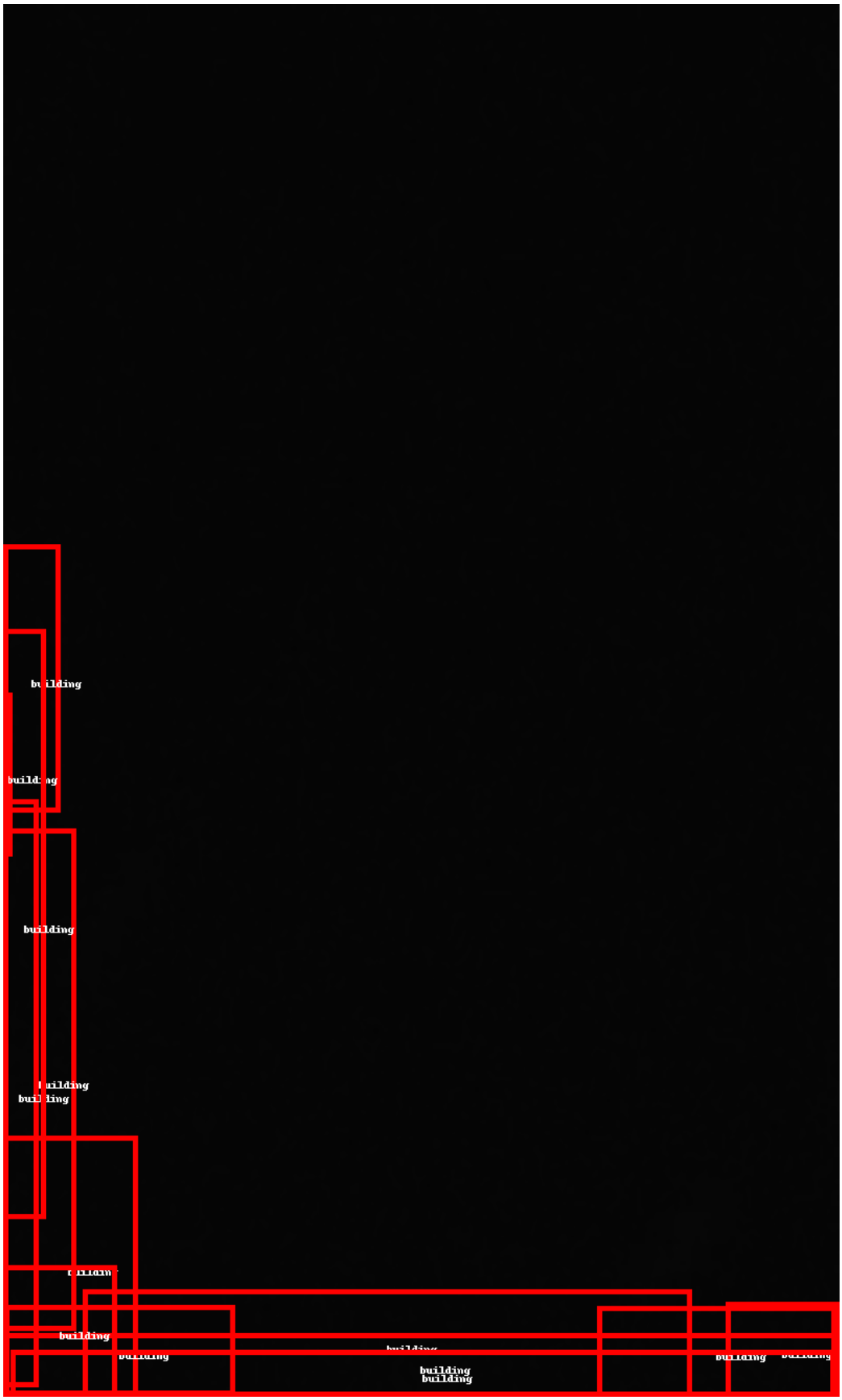
* Without Histogram Equalization – *s2_validation:*

By applying the same steps as Histogram Equalization – *s2_validation* with just one image in *s2_validation* (but we do not apply histogram equalization), we could have the result that we are also able to detect buildings from just one remote sensing image in *s2_validation* without histogram equalization:

```
model = th.hub.load('facebookresearch/detr', 'detr_resnet101', pretrained=True)
model.eval()
model = model.cuda()

# standard PyTorch mean-std input image normalization
transform = T.Compose([
    T.ToTensor(),
    T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

CLASSES = ['building']

url = "/content/ROIs0000_validation_s2_0_p8.jpg"
img = Image.open(url).resize((800,1333)).convert('RGB')
if np.mean(img) < 10 or np.mean(img) > 245:
    print(f"Processing low quality image: {img}")

    # Apply bilateral filtering to remove noise
    img_array = np.array(img)
    filtered = cv2.bilateralFilter(img_array, 9, 75, 75)
    img = Image.fromarray(filtered)

img_tens = transform(img).unsqueeze(0).cuda()

with th.no_grad():
  output = model(img_tens)

im2 = img.copy()
drw = ImageDraw.Draw(im2)
pred_logits=output['pred_logits'][0][:, :len(CLASSES)]
pred_boxes=output['pred_boxes'][0]

max_output = pred_logits.softmax(-1).max(-1)
topk = max_output.values.topk(15)
```

```
pred_logits = pred_logits[topk.indices]
pred_boxes = pred_boxes[topk.indices]
pred_logits.shape

for logits, box in zip(pred_logits, pred_boxes):
  cls = logits.argmax()
  if cls >= len(CLASSES):
    continue
  label = CLASSES[cls]
  print(label)
  box = box.cpu() * th.Tensor([800, 1333, 800, 1333])
  x, y, w, h = box
  x0, x1 = x-w//2, x+w//2
  y0, y1 = y-h//2, y+h//2
  drw.rectangle([x0, y0, x1, y1], outline='red', width=5)
  drw.text((x, y), label, fill='white')
im2
```

```
Using cache found in /root/.cache/torch/hub/facebookresearch_detr_main
Processing low quality image: <PIL.Image.Image image mode=RGB size=800x1333 at 0x7FBD0F9C4FA0>
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
```


building
building
building
building
building
building
building
building
building
building
building
building

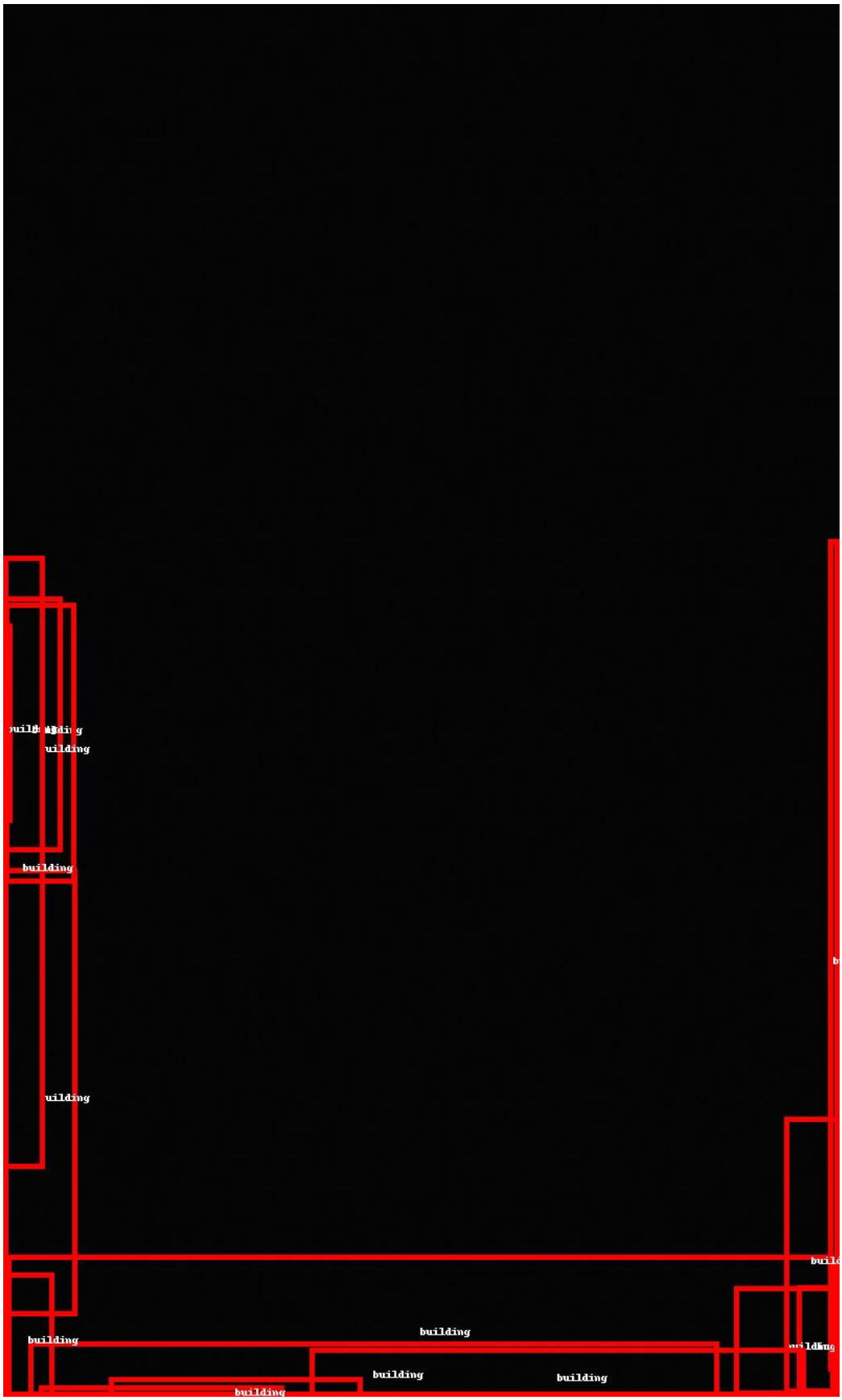
* Without Histogram Equalization – *s2_0:*

By applying the same steps as Without Histogram Equalization – *s2_validation* with just one image in *s2*_0, we could have the result that we are also able to detect buildings from just one remote sensing image in *s2_0* without histogram equalization:

```
model = th.hub.load('facebookresearch/detr', 'detr_resnet101', pretrained=True)
model.eval()
model = model.cuda()

# standard PyTorch mean-std input image normalization
transform = T.Compose([
    T.ToTensor(),
    T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

CLASSES = ['building']

url = "/content/ROIs0000_test_s2_0_p8.jpg"
img = Image.open(url).resize((800,1333)).convert('RGB')
if np.mean(img) < 10 or np.mean(img) > 245:
    print(f"Processing low quality image: {img}")

    # Apply bilateral filtering to remove noise
    img_array = np.array(img)
    filtered = cv2.bilateralFilter(img_array, 9, 75, 75)
    img = Image.fromarray(filtered)
img_tens = transform(img).unsqueeze(0).cuda()

with th.no_grad():
  output = model(img_tens)

im2 = img.copy()
drw = ImageDraw.Draw(im2)
pred_logits=output['pred_logits'][0][:, :len(CLASSES)]
pred_boxes=output['pred_boxes'][0]

max_output = pred_logits.softmax(-1).max(-1)
topk = max_output.values.topk(15)
```

```
pred_logits = pred_logits[topk.indices]
pred_boxes = pred_boxes[topk.indices]
pred_logits.shape

for logits, box in zip(pred_logits, pred_boxes):
  cls = logits.argmax()
  if cls >= len(CLASSES):
    continue
  label = CLASSES[cls]
  print(label)
  box = box.cpu() * th.Tensor([800, 1333, 800, 1333])
  x, y, w, h = box
  x0, x1 = x-w//2, x+w//2
  y0, y1 = y-h//2, y+h//2
  drw.rectangle([x0, y0, x1, y1], outline='red', width=5)
  drw.text((x, y), label, fill='white')
im2
```

```
Using cache found in /root/.cache/torch/hub/facebookresearch_detr_main
Processing low quality image: <PIL.Image.Image image mode=RGB size=800x1333 at 0x7FBD0E335310>
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
building
```


building
building
building
building
building
building
building
building

4. Compare and analyze

In general, applying histogram equalization to a remote sensing image before using it with DETR could potentially enhance the contrast of the image, making it easier for DETR to detect buildings with more accuracy. This is because a better contrast would help differentiate the buildings from the background and other objects in the scene.

Without histogram equalization, the image may have uneven illumination, and some regions may be under-exposed or overexposed. This could result in the buildings appearing darker or lighter than the surrounding areas, making them harder to detect.

However, it is important to note that the effect of histogram equalization on image quality and detection accuracy may depend on the specific characteristics of the image, such as the lighting conditions, the amount of noise, and the distribution of pixel values. It is also possible that histogram equalization could introduce artifacts or distortions in the image, which could negatively impact detection accuracy. Therefore, it is recommended to evaluate the effect of histogram equalization on a case-by-case basis and consider its potential benefits and drawbacks before applying it to remote sensing images for detection tasks.

IV. Conclusion

The DETR model may achieve high accuracy by leveraging multi-head self-attention, which allows the model to attend to multiple spatial locations in the input image simultaneously. Besides, it is important to note that no single object detection model can achieve perfect accuracy in all scenarios and datasets. The accuracy of object detection models like DETR can be affected by factors such as occlusion, image quality, lighting conditions, and object size and shape variation. Therefore, it is recommended to evaluate the performance of object detection models on a case-by-case basis and consider their potential strengths and limitations for specific applications.

In conclusion, this study has investigated the effectiveness of DEtection TRansformer (DETR) in detecting buildings from remote sensing images, with and without histogram equalization. The results show that applying histogram equalization can improve the contrast of the image, making it easier to detect buildings with higher accuracy using DETR.

Our experiments demonstrate that DETR is a state-of-the-art object detection model that can achieve high accuracy in detecting buildings in remote sensing images. The model's ability to directly predict the set of objects and their corresponding bounding boxes in a single shot, without relying on anchor boxes or region proposal networks, makes it a promising option for building detection tasks.

Furthermore, we find that the effectiveness of histogram equalization on remote sensing images can vary depending on factors such as image quality, lighting conditions, and the specific detection algorithm being used. Therefore, it is recommended to evaluate the potential benefits and drawbacks of histogram equalization on a case-by-case basis.

Overall, this study provides valuable insights into the application of DETR for building detection tasks in remote sensing, and highlights the importance of considering different preprocessing techniques to improve the accuracy and reliability of object detection models.

V. Future Work

Looking forward, there are several potential avenues for future research in the area of building detection from remote sensing images using DETR and histogram equalization:

- **Improving the performance of DETR on complex building structures**: While DETR has shown high accuracy in detecting buildings, it may face challenges in accurately detecting complex building structures, such as those with irregular shapes or multiple layers. Future work could focus on exploring ways to improve the ability of DETR to accurately detect these types of buildings.
- Evaluating the effectiveness of other preprocessing techniques: While this study has demonstrated the effectiveness of histogram equalization on building detection, there may be other preprocessing techniques that can further enhance the accuracy of DETR. Future work could explore the effectiveness of other techniques, such as contrast stretching or spatial filtering.
- Adapting DETR to other remote sensing applications: The DETR model has shown promising results in detecting buildings, but it could also be adapted for other remote sensing applications, such as land cover classification or vegetation monitoring. Future work could explore the effectiveness of DETR in these types of applications.
- Investigating the scalability of DETR: While DETR has shown high accuracy in detecting buildings, it may face challenges in processing large-scale remote sensing datasets. Future work could investigate ways to optimize the scalability of DETR to handle larger datasets, such as by exploring distributed training strategies or hardware acceleration techniques.

In summary, future research in this area has the potential to further enhance the accuracy and applicability of DETR for building detection tasks in remote sensing, and to pave the way for more effective and reliable object detection methods in this field.

VI. References

1. Github. Retrieved 29/03/2023, from https://github.com/facebookresearch/detr?fbclid=IwAR0w4fbM7yfq4SXGwwugFdTn1RWMpyTcvd_u-CcAHn8KkHleOuxZe2ZuwoU
2. Meta AI. Retrieved 28/03/2023, from https://ai.facebook.com/research/publications/end-to-end-object-detection-with-transformers
3. COCO (Information about the Coco dataset). Retrieved from 30/03/2023, from https://cocodataset.org/?fbclid=IwAR27WZOnYAn5jT98RgRG3u16aSBhZT_h3DTTy-28hCaFr82rBaOOyYMn2G4#download