Configuration Manual
Translating Egyptian Hieroglyphs Using Deep Learning

MSc Research Project
Data Analyst

Waleed Bin Umer
Student ID: X23187956

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	…….……………………………Waleed Bin Umer……………………………………………
Student ID:	………………………………………X23187956………………………………………….……
Programme:	……………………Data Analyst…………………………… **Year:** ………2024…………..
Module:	……………………………………… MSc Research Project…………………….………
Lecturer:	………………………………… Vladimir Milosavljevic …………………………….……
Submission Due Date:	……………………………………… …12-12-2024………………………………………………
Project Title:	……………… Translating Egyptian Hieroglyphs Using Deep Learning……
Word Count:	……………1891……………… **Page Count:** ……………………19…………….……..………

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: …………………………… Waleed Bin Umer ……………………………………..…

Date: ………………………………… 12-12-2024…………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Waleed Bin Umer
Student ID: X23187956

1 Introduction

configuration guide will outline the primary aspects and procedures which should be put in practice in order to accomplish the goals of the research project “Translating Egyptian Hieroglyphs Using Deep Learning”. Besides that, it provides a detailed guide on how to set up and operate the models on such hardware and software environments: SSD, YOLOv5, YOLOv8 and Faster R-CNN.
The manual allows reproducibility and guarantees the adequate performance of the research project. The contents include the following:

- Data collection and loading.
- Data splitting and pre-processing.
- Assessing and contrasting the models.
- Configuration of both hardware and software

2 Hardware Configuration

The hardware configuration used for deep learning training, to fill the computational needs especially when working with larger models are given below:

Table 1: Hardware Configuration

Machine Name	ASUS TUF Gaming F15
Processor	12th Gen Intel® Core™ i5-12500H 2.50 GHz
RAM	16 GB
GPU	NVIDIA GeForce RTX 3050
System Type	64-bit operating system, x64-based processor

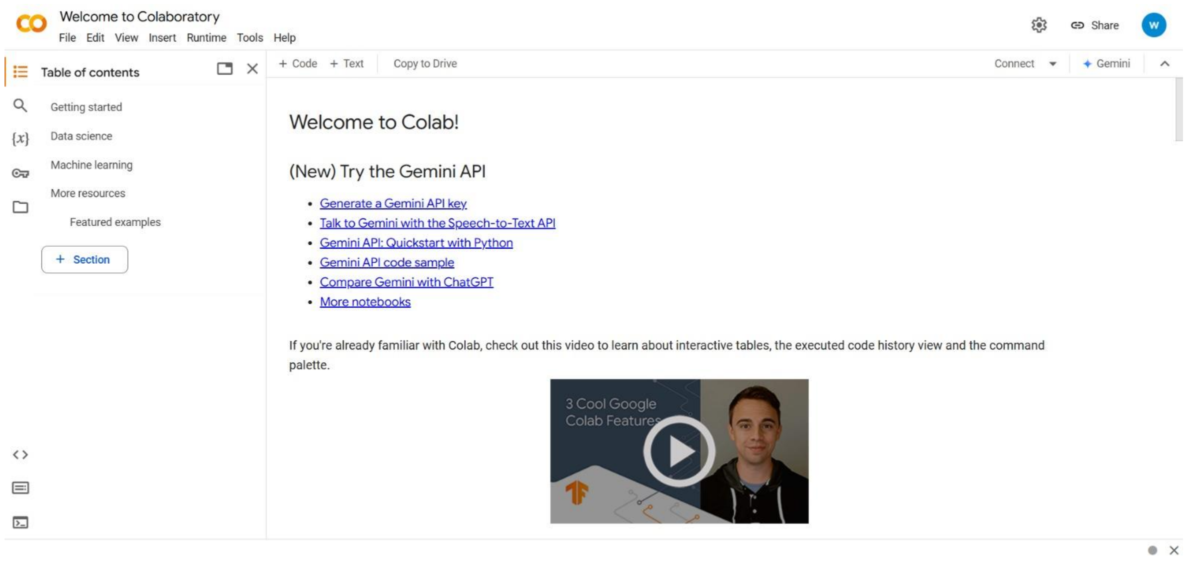
Key Features:

- High CPU and GPU performance: In constructing deep learning models training stage is also of significance that’s why it requires a good CPU and GPU.
- Efficiency of proper RAM: poor RAM efficiency can have completion of large data and resource hungry computations operating adequately.
- des Acceleration – The GPU allows training over compute-intensive algorithms and inference over them faster so that the advancements are clearly more effective than CPU implementations.

3 Software Configuration

3.1 Google Colab

The research project is written on google colab. So, the very first step that needs to be done in order to accomplish the tasks of this research project is registering with Gmail account since the execution cannot be done without signing in the Gmail. Google Colab can be used for deep learning projects because it is a free web-enabled IDE provided by Google Research.



Welcome to Colab!

(New) Try the Gemini API

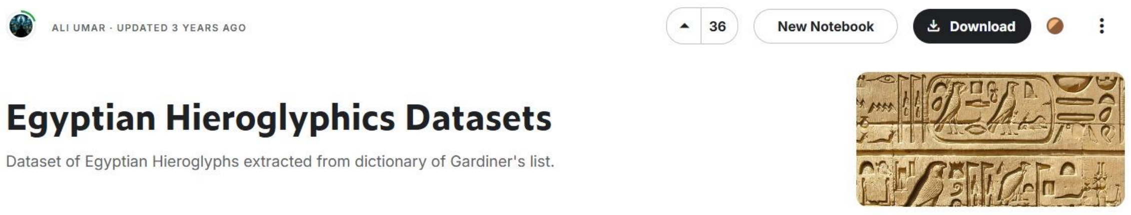
- Generate a Gemini API key
- Talk to Gemini with the Speech-to-Text API
- Gemini API: Quickstart with Python
- Gemini API code sample
- Compare Gemini with ChatGPT
- More notebooks

If you're already familiar with Colab, check out this video to learn about interactive tables, the executed code history view and the command palette.

Figure 1: Google Colab

3.2 Data Source

It is more of a dataset research project or record rather than a hierarchical structure of coding. Egyptian Hieroglyphs are available in a dataset that is above all free readily available for download on the website Kaggle. It contains a training image of 6000 images and validation images to the amount of 2000 images distributed over the sections of Animal, Plants Symbols and Deities. To meet the constraint of time due to lack of sufficient resources, the data augmentation method was also useful in creating variety enhancement over the training the model. This project involves a dataset that helped design and test the algorithm of hieroglyph detection and translation.



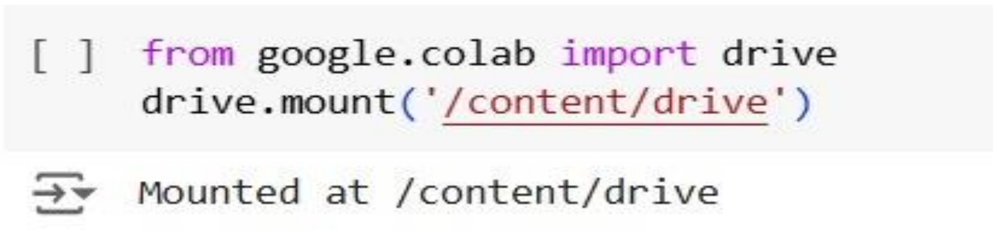
Egyptian Hieroglyphics Datasets

Dataset of Egyptian Hieroglyphs extracted from dictionary of Gardiner's list.

Figure 2: Data set

3.3 Mounting the Google Drive

Mounting Google Drive in Google Colab allows you to access files and datasets stored in your Drive without having to upload manually. This is done by running the command in the figure [3] which connects your Google Drive to the Colab environment. Once authorized the files will be available under the mentioned directory. Best use case is when you have big datasets or shared resources on a project, and you are working in a Colab environment for a smooth workflow.

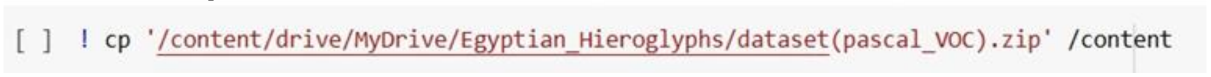


```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

Figure 3: Mounting the google Drive

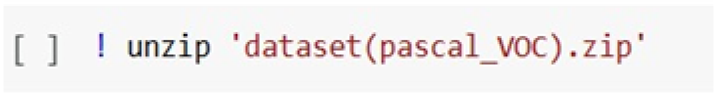
3.4 Loading the Data

To transfer the information already present in Google Drive to Colab for work. Use the command shown in figure [4] to copy files from or upload files to Google Drive into the current folder of Colab the command shown in figure [5] will make it possible to utilize the raw data to analysis within Colab later on.



```
[ ]  ! cp '/content/drive/MyDrive/Egyptian_Hieroglyphs/dataset(pascal_VOC).zip' /content
```

Figure 4: copying the dataset from drive



```
[ ]  ! unzip 'dataset(pascal_VOC).zip'
```

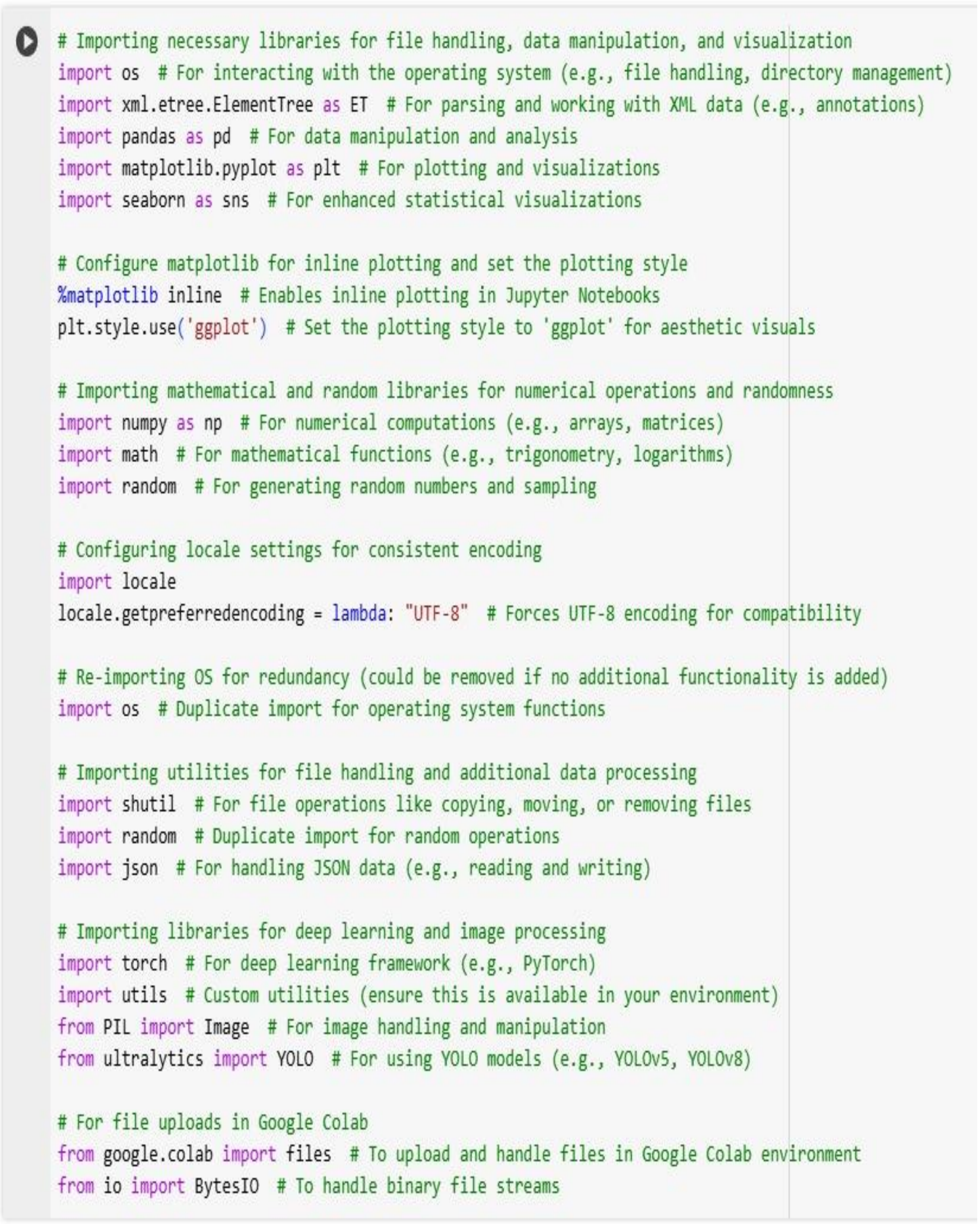
Figure 5: Extracting the content of the dataset from drive

4 Project Development

4.1 Import Libraries

Libraries used in this project to run the scripts and frameworks. Various libraries are to accomplish many functions in the project like data visualization and deep learning.
libraries in the project are:

- Matplotlib
- Numpy
- Pandas
- PyTorch
- TensorFlow
- OpenCV
- seaborn

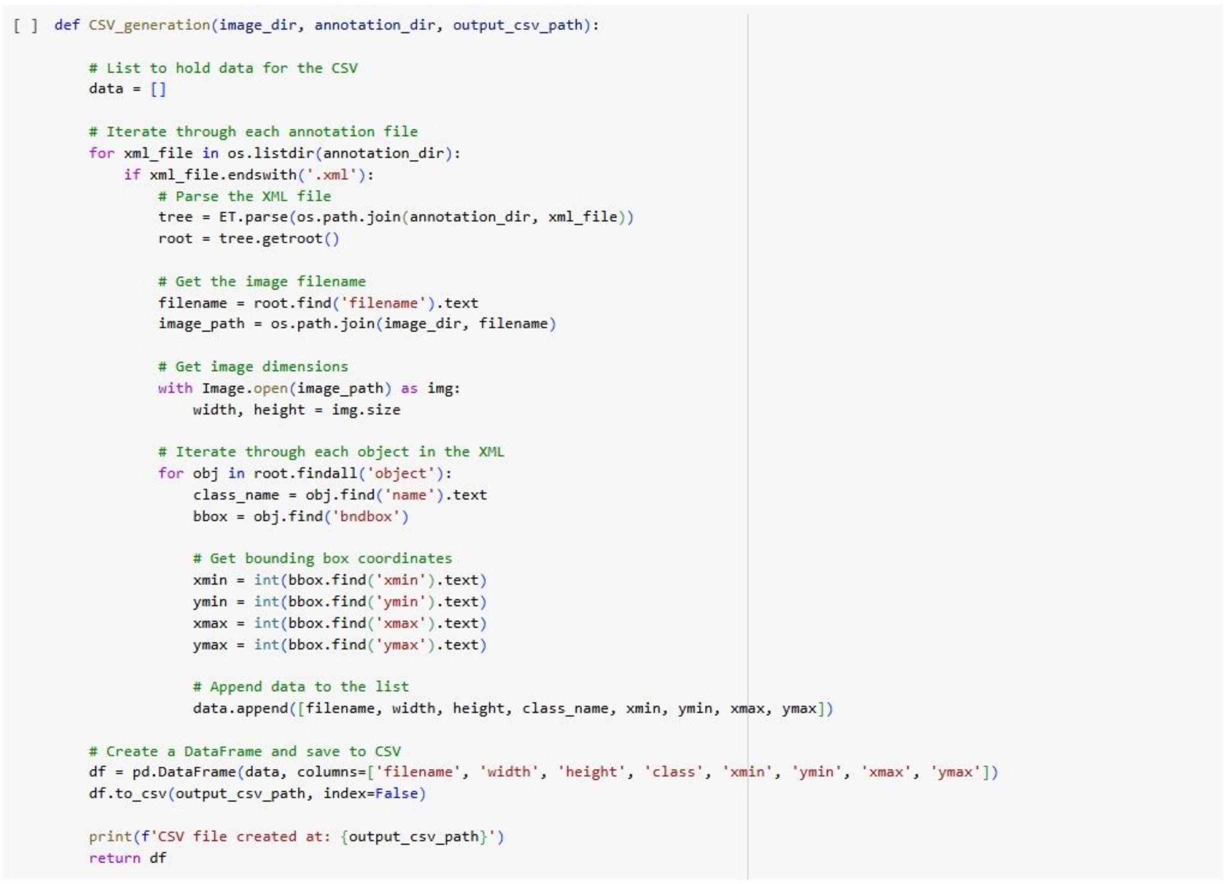


```python
# Importing necessary libraries for file handling, data manipulation, and visualization
import os  # For interacting with the operating system (e.g., file handling, directory management)
import xml.etree.ElementTree as ET  # For parsing and working with XML data (e.g., annotations)
import pandas as pd  # For data manipulation and analysis
import matplotlib.pyplot as plt  # For plotting and visualizations
import seaborn as sns  # For enhanced statistical visualizations

# Configure matplotlib for inline plotting and set the plotting style
%matplotlib inline  # Enables inline plotting in Jupyter Notebooks
plt.style.use('ggplot')  # Set the plotting style to 'ggplot' for aesthetic visuals

# Importing mathematical and random libraries for numerical operations and randomness
import numpy as np  # For numerical computations (e.g., arrays, matrices)
import math  # For mathematical functions (e.g., trigonometry, logarithms)
import random  # For generating random numbers and sampling

# Configuring locale settings for consistent encoding
import locale
locale.getpreferredencoding = lambda: "UTF-8"  # Forces UTF-8 encoding for compatibility

# Re-importing OS for redundancy (could be removed if no additional functionality is added)
import os  # Duplicate import for operating system functions

# Importing utilities for file handling and additional data processing
import shutil  # For file operations like copying, moving, or removing files
import random  # Duplicate import for random operations
import json  # For handling JSON data (e.g., reading and writing)

# Importing libraries for deep learning and image processing
import torch  # For deep learning framework (e.g., PyTorch)
import utils  # Custom utilities (ensure this is available in your environment)
from PIL import Image  # For image handling and manipulation
from ultralytics import YOLO  # For using YOLO models (e.g., YOLOv5, YOLOv8)

# For file uploads in Google Colab
from google.colab import files  # To upload and handle files in Google Colab environment
from io import BytesIO  # To handle binary file streams
```

Figure 6: All the imported libraries in code

4.2 Generating CSV Files

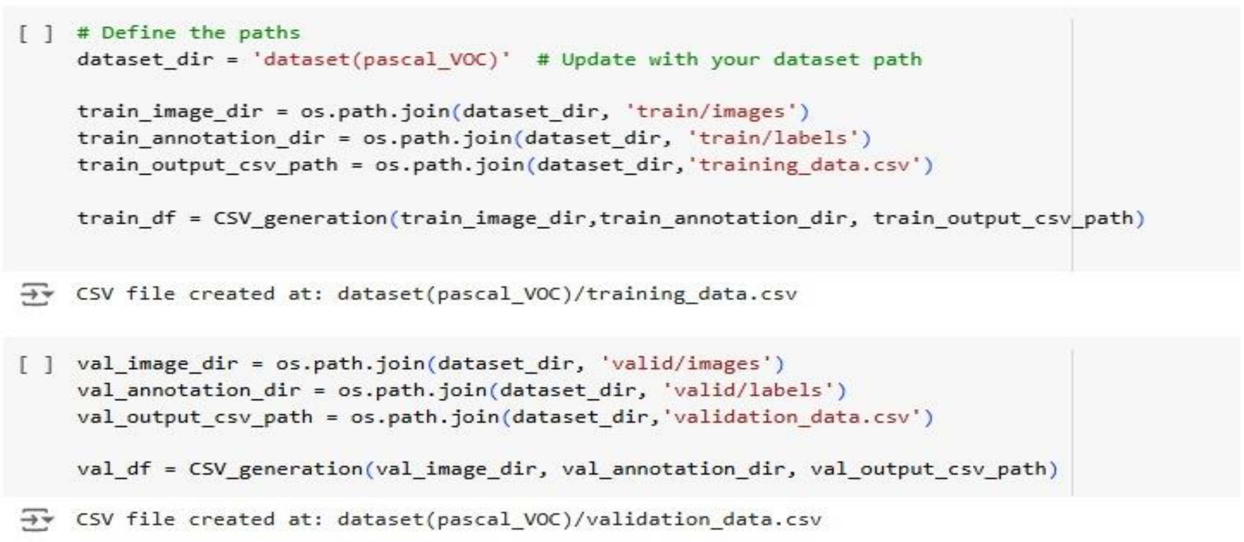
Converting XML annotations into a structured CSV file for handle easily.



```
def CSV_generation(image_dir, annotation_dir, output_csv_path):

    # List to hold data for the CSV
    data = []

    # Iterate through each annotation file
    for xml_file in os.listdir(annotation_dir):
        if xml_file.endswith('.xml'):
            # Parse the XML file
            tree = ET.parse(os.path.join(annotation_dir, xml_file))
            root = tree.getroot()

            # Get the image filename
            filename = root.find('filename').text
            image_path = os.path.join(image_dir, filename)

            # Get image dimensions
            with Image.open(image_path) as img:
                width, height = img.size

            # Iterate through each object in the XML
            for obj in root.findall('object'):
                class_name = obj.find('name').text
                bbox = obj.find('bndbox')

                # Get bounding box coordinates
                xmin = int(bbox.find('xmin').text)
                ymin = int(bbox.find('ymin').text)
                xmax = int(bbox.find('xmax').text)
                ymax = int(bbox.find('ymax').text)

                # Append data to the list
                data.append([filename, width, height, class_name, xmin, ymin, xmax, ymax])

    # Create a DataFrame and save to CSV
    df = pd.DataFrame(data, columns=['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax'])
    df.to_csv(output_csv_path, index=False)

    print(f'CSV file created at: {output_csv_path}')
    return df
```

Figure 7: Code to convert XML to CSV

Generate a CSV file from the XML annotations:



```
# Define the paths
dataset_dir = 'dataset(pascal_VOC)'  # Update with your dataset path

train_image_dir = os.path.join(dataset_dir, 'train/images')
train_annotation_dir = os.path.join(dataset_dir, 'train/labels')
train_output_csv_path = os.path.join(dataset_dir,'training_data.csv')

train_df = CSV_generation(train_image_dir,train_annotation_dir, train_output_csv_path)
```

```
CSV file created at: dataset(pascal_VOC)/training_data.csv
```

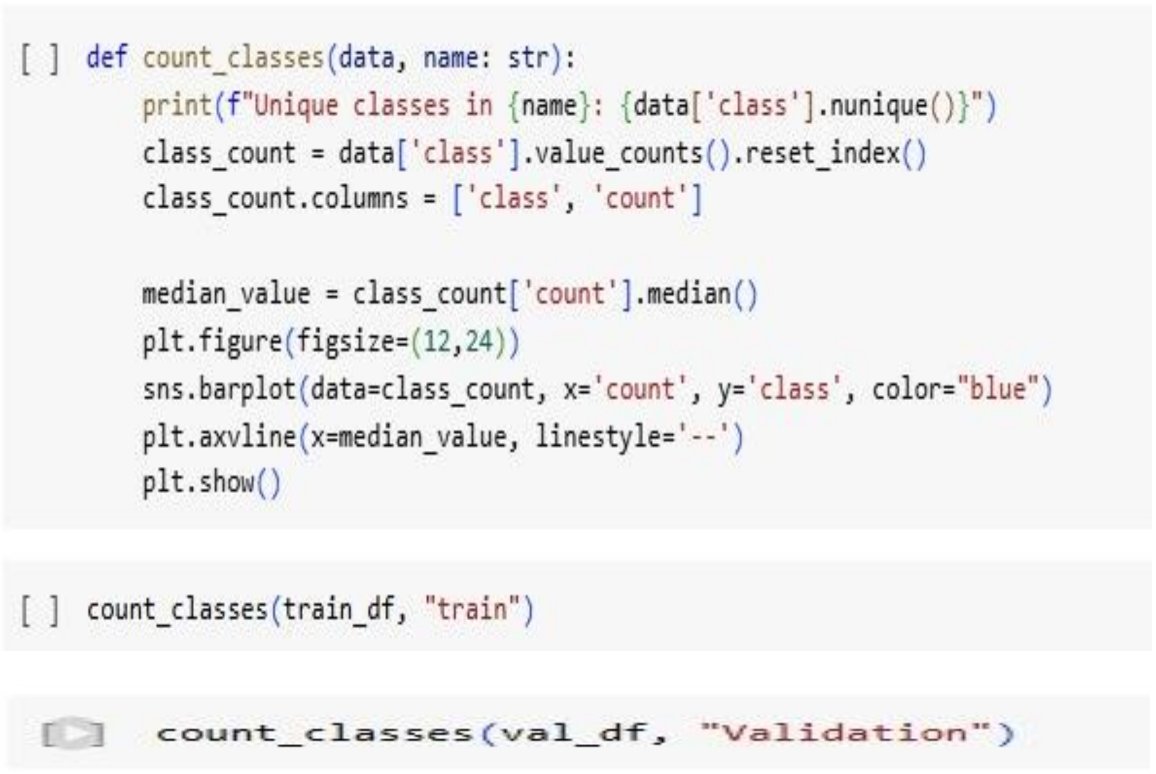
```
val_image_dir = os.path.join(dataset_dir, 'valid/images')
val_annotation_dir = os.path.join(dataset_dir, 'valid/labels')
val_output_csv_path = os.path.join(dataset_dir,'validation_data.csv')

val_df = CSV_generation(val_image_dir, val_annotation_dir, val_output_csv_path)
```

```
CSV file created at: dataset(pascal_VOC)/validation_data.csv
```

Figure 8: code to build CSV from XML annotations

4.3 Class Counting

It analyses different classes that exist and the distribution across almost the entire datasets available to be covered. Training Dataset: Contains 95 distinct classes. Validation Dataset: Contains 96 distinct classes, also has a separate class called the “Head”. The “Head” class appeared only 9 times in the validation dataset, and 9 did not feature in the training dataset.



```
[ ] def count_classes(data, name: str):
        print(f"Unique classes in {name}: {data['class'].nunique()}")
        class_count = data['class'].value_counts().reset_index()
        class_count.columns = ['class', 'count']

        median_value = class_count['count'].median()
        plt.figure(figsize=(12,24))
        sns.barplot(data=class_count, x='count', y='class', color="blue")
        plt.axvline(x=median_value, linestyle='--')
        plt.show()
```

```
[ ] count_classes(train_df, "train")
```

```
count_classes(val_df, "Validation")
```

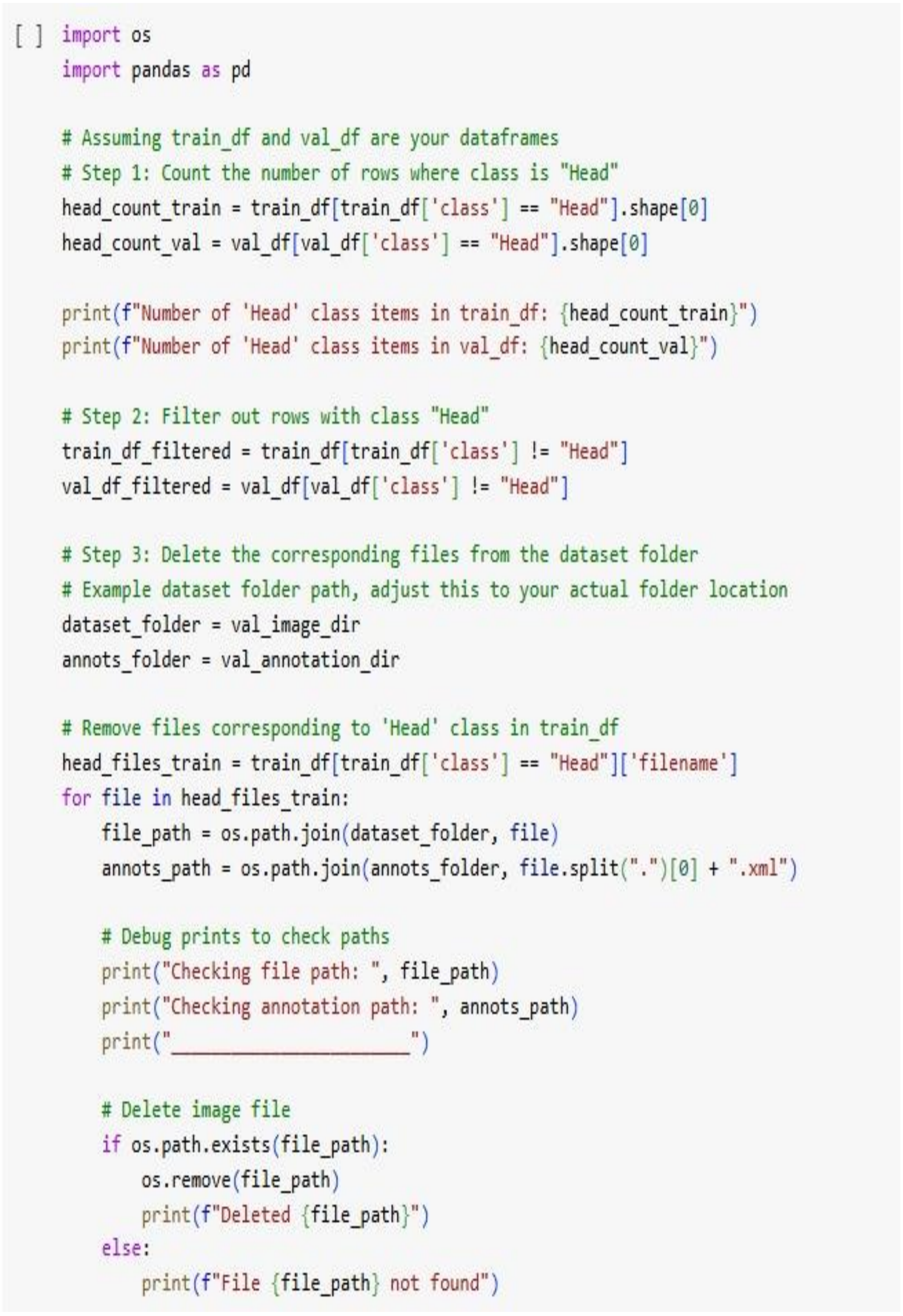
Figure 9: Class Distribution in Training and Validation Dataset

4.4 Class Counting

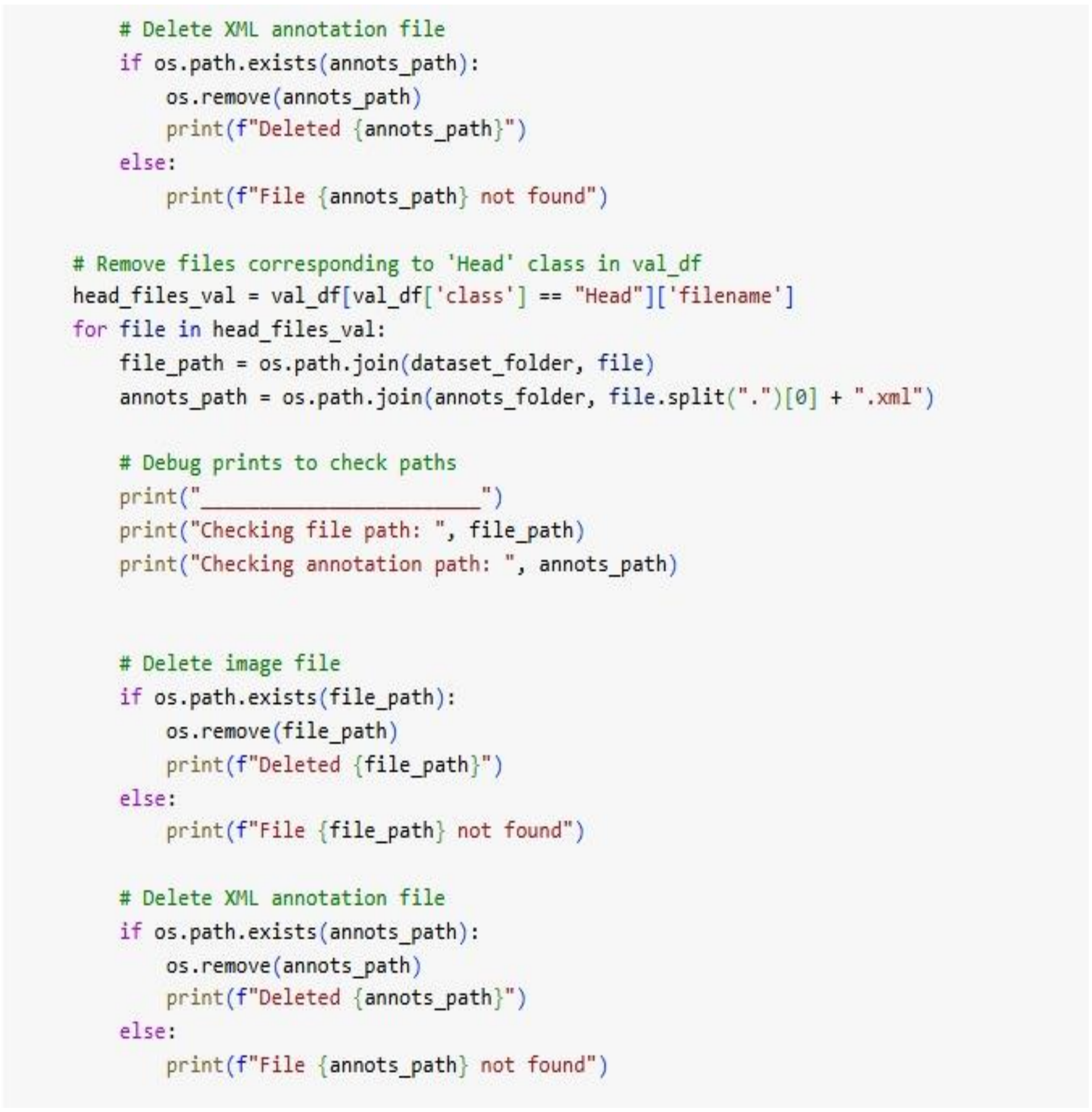
Once the dataset contains the ordered and the same content, then the number of class discrepancies is small and uniform.

- Head Class: Find Rows with the “Head” Class This class “head” has been observed in training and validation sets and therefore it affects class count, and this has led to its deletion.
- Remove the Corresponding Image and Annotation Files. On the need to remove the oversampling of “Head”, locate and erase all images classified as ‘Head’, from the validation and the training set.

Now the dataset is consistent and free from unnecessary class which helps in optimization to train model and evaluation.



```
[ ] import os
    import pandas as pd

    # Assuming train_df and val_df are your dataframes
    # Step 1: Count the number of rows where class is "Head"
    head_count_train = train_df[train_df['class'] == "Head"].shape[0]
    head_count_val = val_df[val_df['class'] == "Head"].shape[0]

    print(f"Number of 'Head' class items in train_df: {head_count_train}")
    print(f"Number of 'Head' class items in val_df: {head_count_val}")

    # Step 2: Filter out rows with class "Head"
    train_df_filtered = train_df[train_df['class'] != "Head"]
    val_df_filtered = val_df[val_df['class'] != "Head"]

    # Step 3: Delete the corresponding files from the dataset folder
    # Example dataset folder path, adjust this to your actual folder location
    dataset_folder = val_image_dir
    annots_folder = val_annotation_dir

    # Remove files corresponding to 'Head' class in train_df
    head_files_train = train_df[train_df['class'] == "Head"]['filename']
    for file in head_files_train:
        file_path = os.path.join(dataset_folder, file)
        annots_path = os.path.join(annots_folder, file.split(".")[0] + ".xml")

        # Debug prints to check paths
        print("Checking file path: ", file_path)
        print("Checking annotation path: ", annots_path)
        print("________________________")

        # Delete image file
        if os.path.exists(file_path):
            os.remove(file_path)
            print(f"Deleted {file_path}")
        else:
            print(f"File {file_path} not found")
```



```
    # Delete XML annotation file
    if os.path.exists(annots_path):
        os.remove(annots_path)
        print(f"Deleted {annots_path}")
    else:
        print(f"File {annots_path} not found")

# Remove files corresponding to 'Head' class in val_df
head_files_val = val_df[val_df['class'] == "Head"]['filename']
for file in head_files_val:
    file_path = os.path.join(dataset_folder, file)
    annots_path = os.path.join(annots_folder, file.split(".")[0] + ".xml")

    # Debug prints to check paths
    print("________________________")
    print("Checking file path: ", file_path)
    print("Checking annotation path: ", annots_path)


    # Delete image file
    if os.path.exists(file_path):
        os.remove(file_path)
        print(f"Deleted {file_path}")
    else:
        print(f"File {file_path} not found")

    # Delete XML annotation file
    if os.path.exists(annots_path):
        os.remove(annots_path)
        print(f"Deleted {annots_path}")
    else:
        print(f"File {annots_path} not found")
```

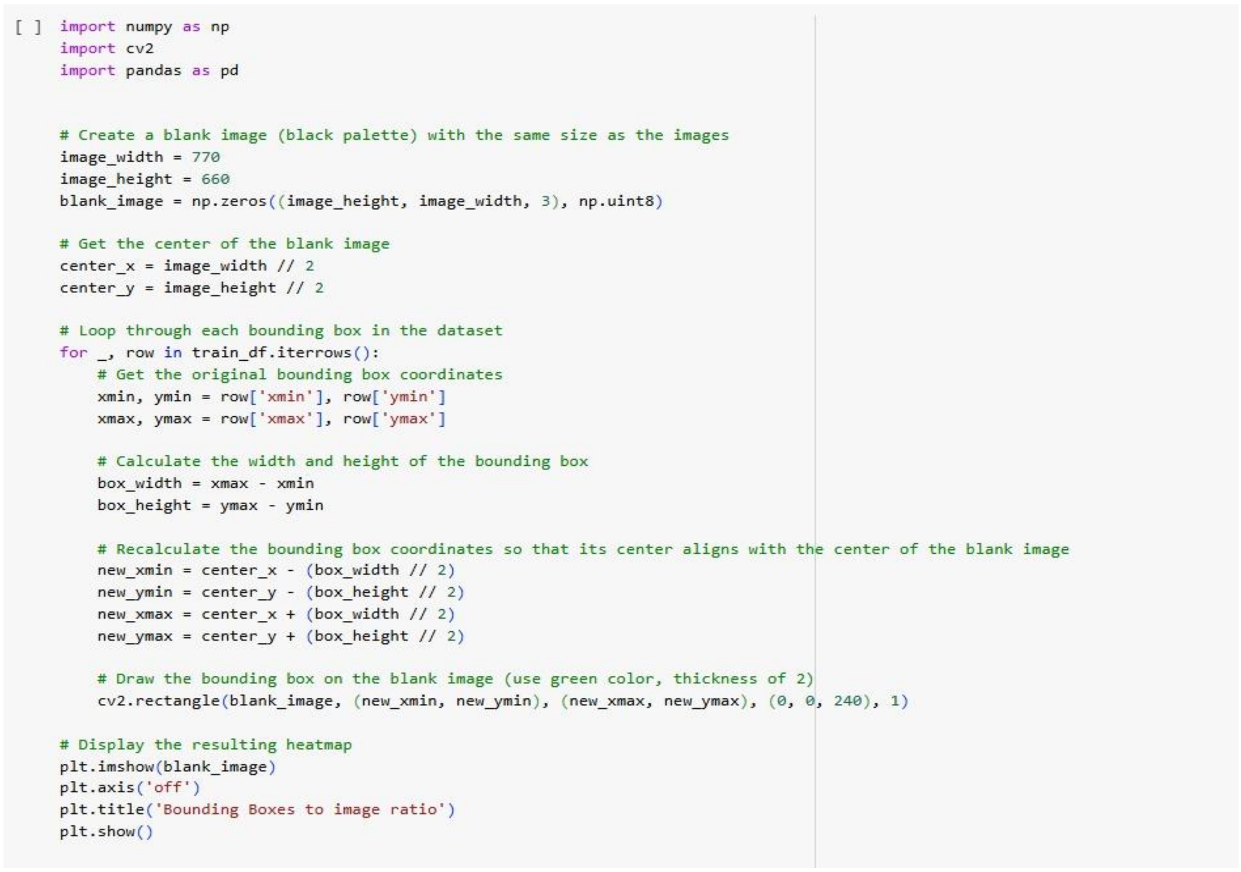
Figure 10: Code to eliminating Inconsistencies classes

4.5 Data Visualization

The EDA visualizations explore spacial tends on the dataset:

I. Bounding Box Alignment:

Code: A blank image was built by np.zeros(). Bounding box values from the dataset were modified in the style that it aligns in accordance with the center point of the image center. It was extracted by using cv2.rectangle().



```
[ ]  import numpy as np
     import cv2
     import pandas as pd


     # Create a blank image (black palette) with the same size as the images
     image_width = 770
     image_height = 660
     blank_image = np.zeros((image_height, image_width, 3), np.uint8)

     # Get the center of the blank image
     center_x = image_width // 2
     center_y = image_height // 2

     # Loop through each bounding box in the dataset
     for _, row in train_df.iterrows():
         # Get the original bounding box coordinates
         xmin, ymin = row['xmin'], row['ymin']
         xmax, ymax = row['xmax'], row['ymax']

         # Calculate the width and height of the bounding box
         box_width = xmax - xmin
         box_height = ymax - ymin

         # Recalculate the bounding box coordinates so that its center aligns with the center of the blank image
         new_xmin = center_x - (box_width // 2)
         new_ymin = center_y - (box_height // 2)
         new_xmax = center_x + (box_width // 2)
         new_ymax = center_y + (box_height // 2)

         # Draw the bounding box on the blank image (use green color, thickness of 2)
         cv2.rectangle(blank_image, (new_xmin, new_ymin), (new_xmax, new_ymax), (0, 0, 240), 1)

     # Display the resulting heatmap
     plt.imshow(blank_image)
     plt.axis('off')
     plt.title('Bounding Boxes to image ratio')
     plt.show()
```

Figure 11: Code for the bounding box

Purpose: It deals with two purposes, to see the visualization of how objects are scattered across the dataset and ensuring alignment for better detection.

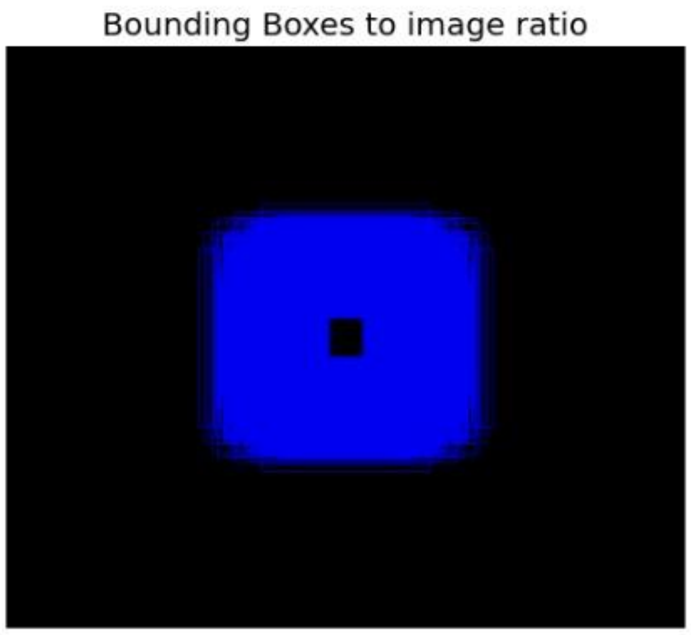
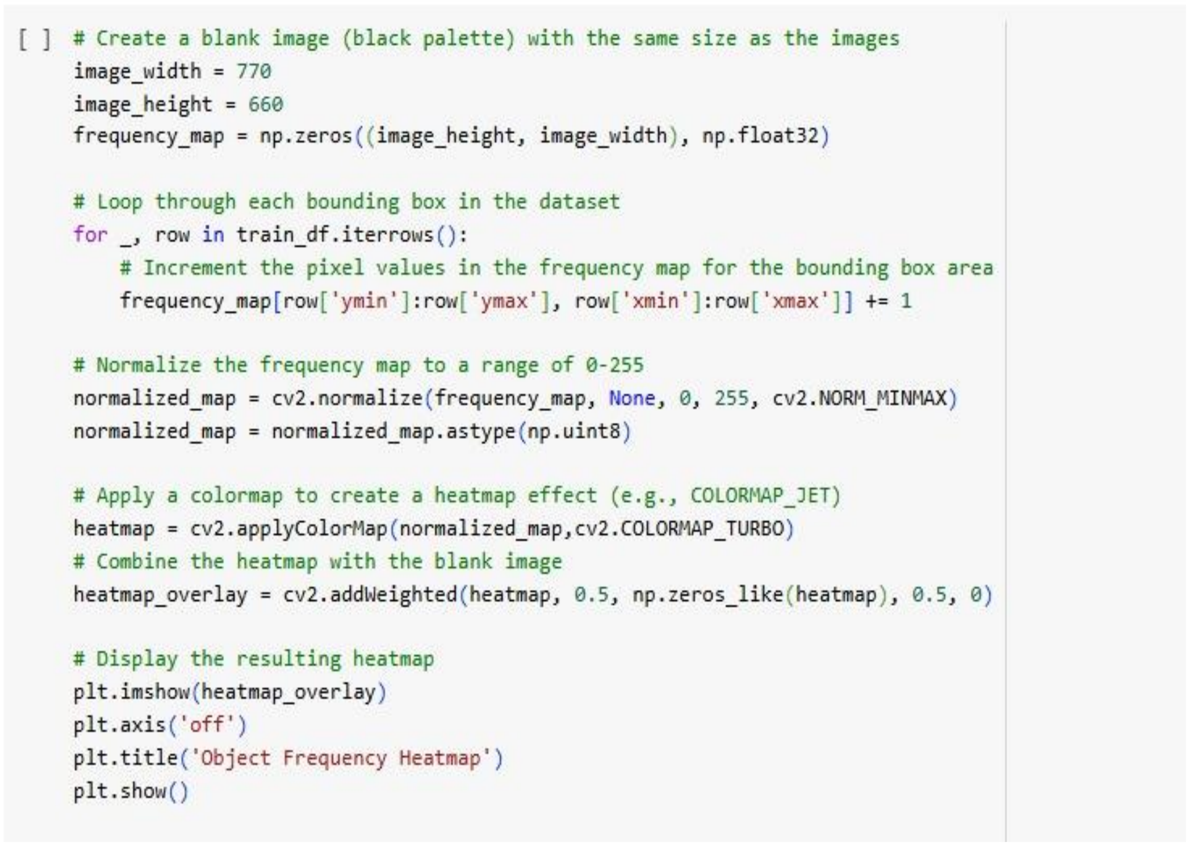


Figure 12: Shows bounding box overlap and concentration at the image center.

II. Heatmap Creation:

Code: np.zeros() was set to with frequency map. incremental count was filled with bounding box areas, normalized using cv2. normalize(), and visualized with cv2. applyColorMap() for generating a heatmap.



```
[ ] # Create a blank image (black palette) with the same size as the images
    image_width = 770
    image_height = 660
    frequency_map = np.zeros((image_height, image_width), np.float32)

    # Loop through each bounding box in the dataset
    for _, row in train_df.iterrows():
        # Increment the pixel values in the frequency map for the bounding box area
        frequency_map[row['ymin']:row['ymax'], row['xmin']:row['xmax']] += 1

    # Normalize the frequency map to a range of 0-255
    normalized_map = cv2.normalize(frequency_map, None, 0, 255, cv2.NORM_MINMAX)
    normalized_map = normalized_map.astype(np.uint8)

    # Apply a colormap to create a heatmap effect (e.g., COLORMAP_JET)
    heatmap = cv2.applyColorMap(normalized_map,cv2.COLORMAP_TURBO)
    # Combine the heatmap with the blank image
    heatmap_overlay = cv2.addWeighted(heatmap, 0.5, np.zeros_like(heatmap), 0.5, 0)

    # Display the resulting heatmap
    plt.imshow(heatmap_overlay)
    plt.axis('off')
    plt.title('Object Frequency Heatmap')
    plt.show()
```

Figure 13: code for spatial distribution through a heatmap

Purpose: Heatmap puts the light on the bounding boxes density, revealing higher concentrations of boxes to the center of the image.

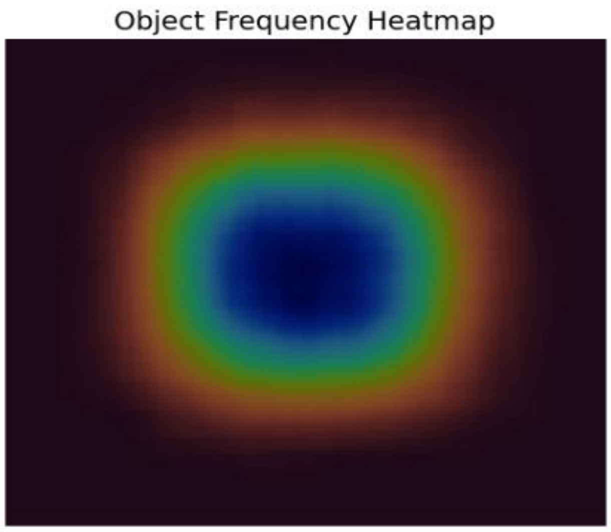
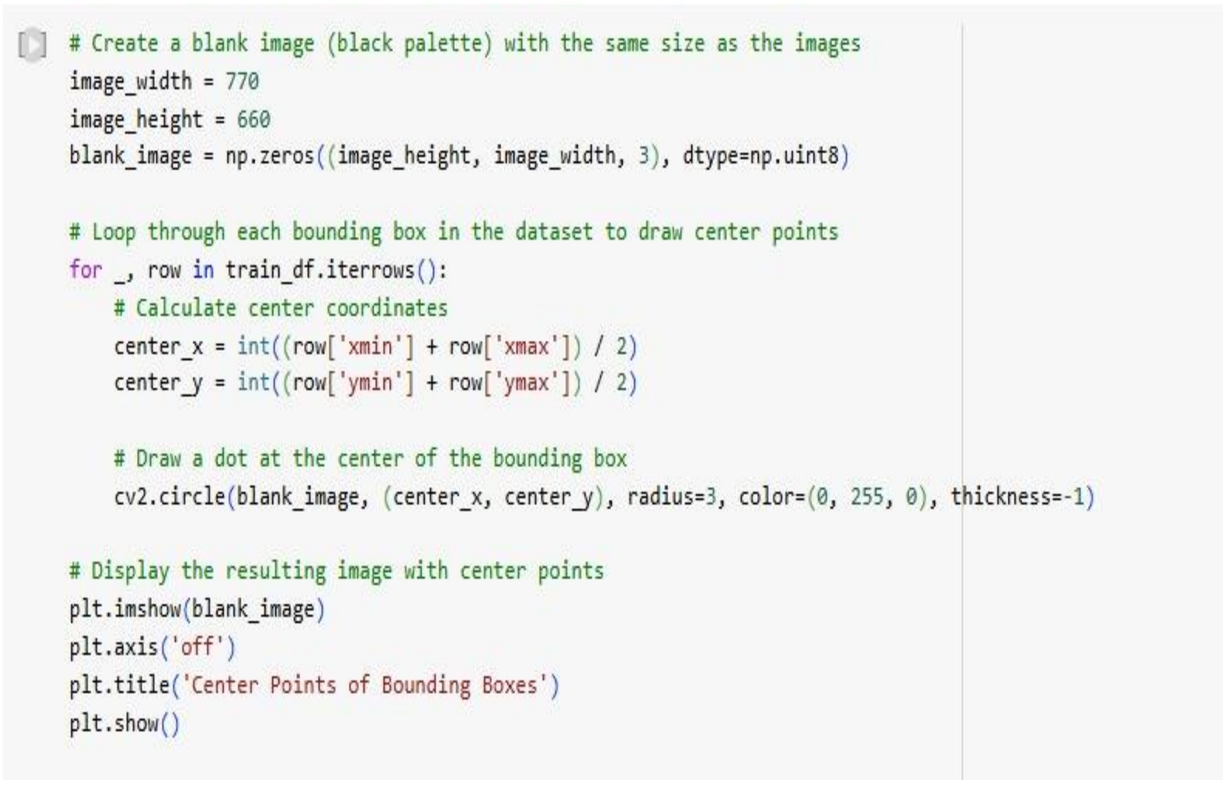


Figure 14: Highlights spatial distribution through a heatmap, focusing on high-density areas.

III. Scatter Plot of Centers:

Code: Center points of bounding boxes were computed as (xmin+xmax)/2, (ymin+ymax)/2. Circles were drawn by using these points cv2.circle() on a blank image.



```python
# Create a blank image (black palette) with the same size as the images
image_width = 770
image_height = 660
blank_image = np.zeros((image_height, image_width, 3), dtype=np.uint8)

# Loop through each bounding box in the dataset to draw center points
for _, row in train_df.iterrows():
    # Calculate center coordinates
    center_x = int((row['xmin'] + row['xmax']) / 2)
    center_y = int((row['ymin'] + row['ymax']) / 2)

    # Draw a dot at the center of the bounding box
    cv2.circle(blank_image, (center_x, center_y), radius=3, color=(0, 255, 0), thickness=-1)

# Display the resulting image with center points
plt.imshow(blank_image)
plt.axis('off')
plt.title('Center Points of Bounding Boxes')
plt.show()
```

Figure 15: code for scatter plot

Purpose: Clustering of object centers detected over image center from scatter plot supporting the calculation with better detection focus.
Code aims at preprocessing and visualize the dataset on each step, making it more optimized for the modeling tasks.

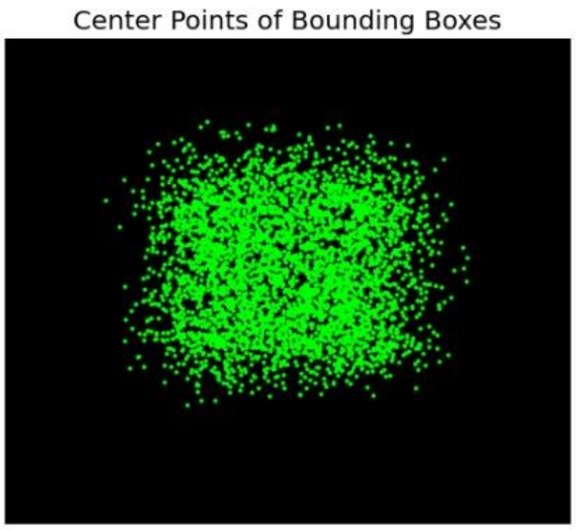


Figure 16: Illustrates uniform object detection with hieroglyphs centered in the image space

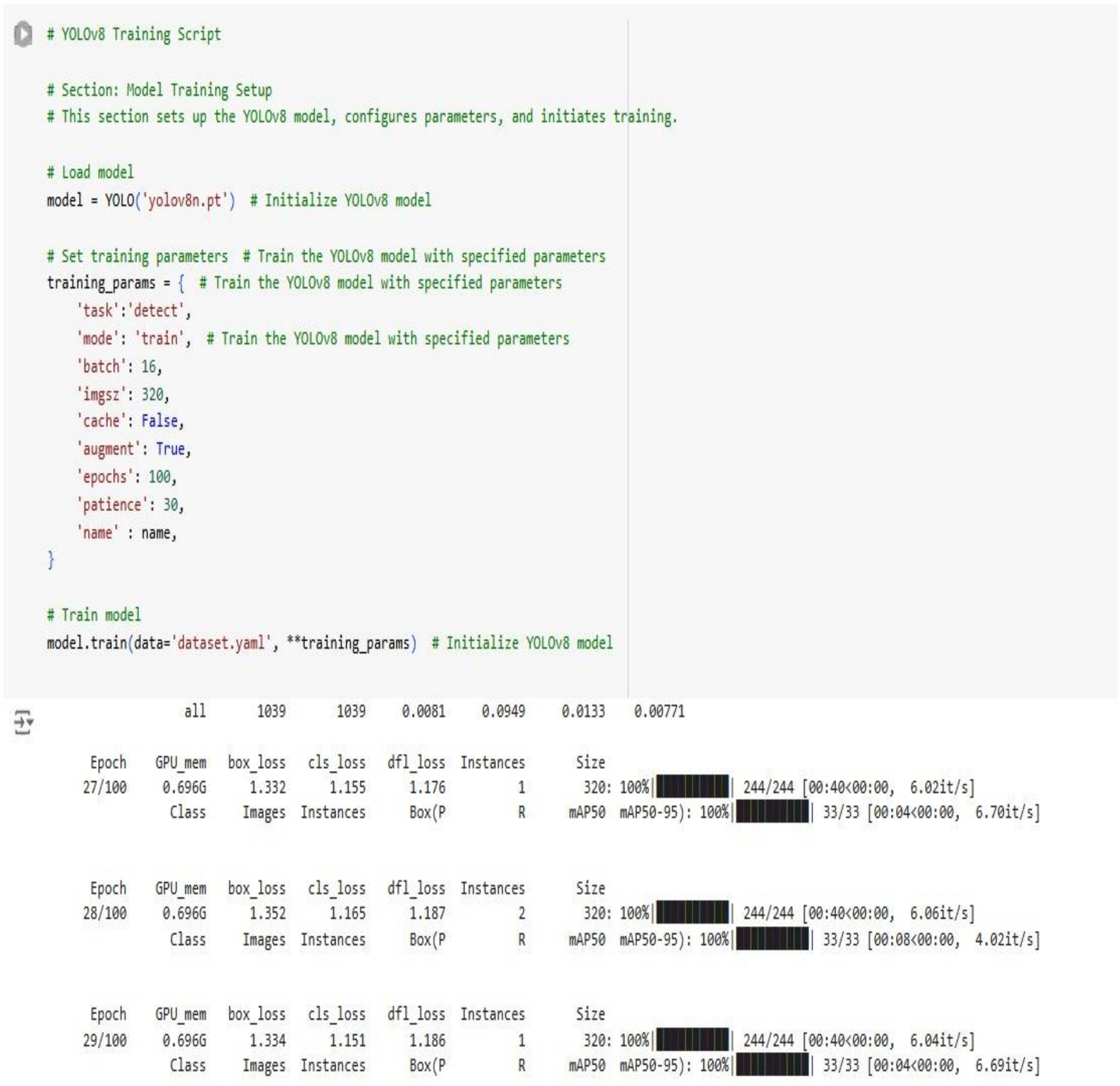
4.6 Model Implementation

This study incorporated the process of training and possibly tuning as many as four object detection models - SSD, YOLOV5, YOLOV8, and Faster R-CNN for success in hierarchy algorithm to glyphs in forms of symbols or characters intended to be written or carved. The models were trained on a hieroglyph dataset in Google Colab, making use of GPU resources.

A. Model Training Details:

I. YOLOv8 Model:

The model YOLOv8 was qualified with 100 epochs.



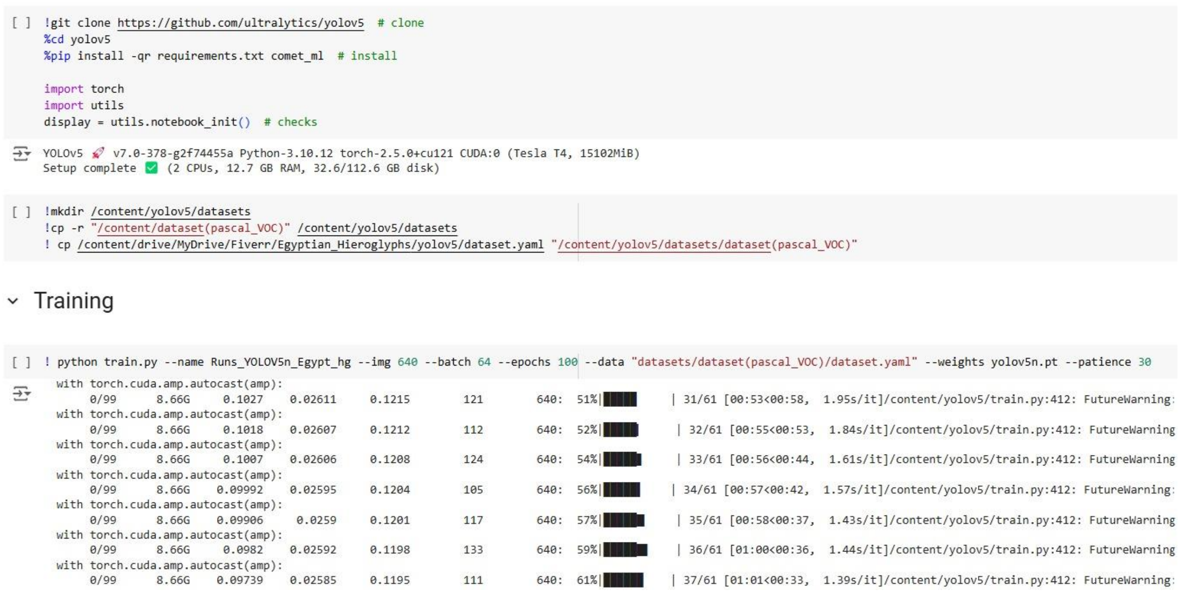
```python
# YOLOv8 Training Script

# Section: Model Training Setup
# This section sets up the YOLOv8 model, configures parameters, and initiates training.

# Load model
model = YOLO('yolov8n.pt')  # Initialize YOLOv8 model

# Set training parameters  # Train the YOLOv8 model with specified parameters
training_params = {  # Train the YOLOv8 model with specified parameters
    'task':'detect',
    'mode': 'train',  # Train the YOLOv8 model with specified parameters
    'batch': 16,
    'imgsz': 320,
    'cache': False,
    'augment': True,
    'epochs': 100,
    'patience': 30,
    'name' : name,
}

# Train model
model.train(data='dataset.yaml', **training_params)  # Initialize YOLOv8 model
```

```
                   all       1039       1039     0.0081     0.0949     0.0133    0.00771

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     27/100     0.696G      1.332      1.155      1.176          1        320: 100%|██████████| 244/244 [00:40<00:00,  6.02it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 33/33 [00:04<00:00,  6.70it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     28/100     0.696G      1.352      1.165      1.187          2        320: 100%|██████████| 244/244 [00:40<00:00,  6.06it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 33/33 [00:08<00:00,  4.02it/s]

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
     29/100     0.696G      1.334      1.151      1.186          1        320: 100%|██████████| 244/244 [00:40<00:00,  6.04it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 33/33 [00:04<00:00,  6.69it/s]
```

Figure 17: code for YOLOv8 training epochs

II. YOLOv5 Model:

YOLOv5 model was trained for 100 epochs.



```
[ ] !git clone https://github.com/ultralytics/yolov5  # clone
    %cd yolov5
    %pip install -qr requirements.txt comet_ml  # install

    import torch
    import utils
    display = utils.notebook_init()  # checks

    YOLOv5 🚀 v7.0-378-g2f74455a Python-3.10.12 torch-2.5.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
    Setup complete ✅ (2 CPUs, 12.7 GB RAM, 32.6/112.6 GB disk)
```

```
[ ] !mkdir /content/yolov5/datasets
    !cp -r "/content/dataset(pascal_VOC)" /content/yolov5/datasets
    ! cp /content/drive/MyDrive/Fiverr/Egyptian_Hieroglyphs/yolov5/dataset.yaml "/content/yolov5/datasets/dataset(pascal_VOC)"
```

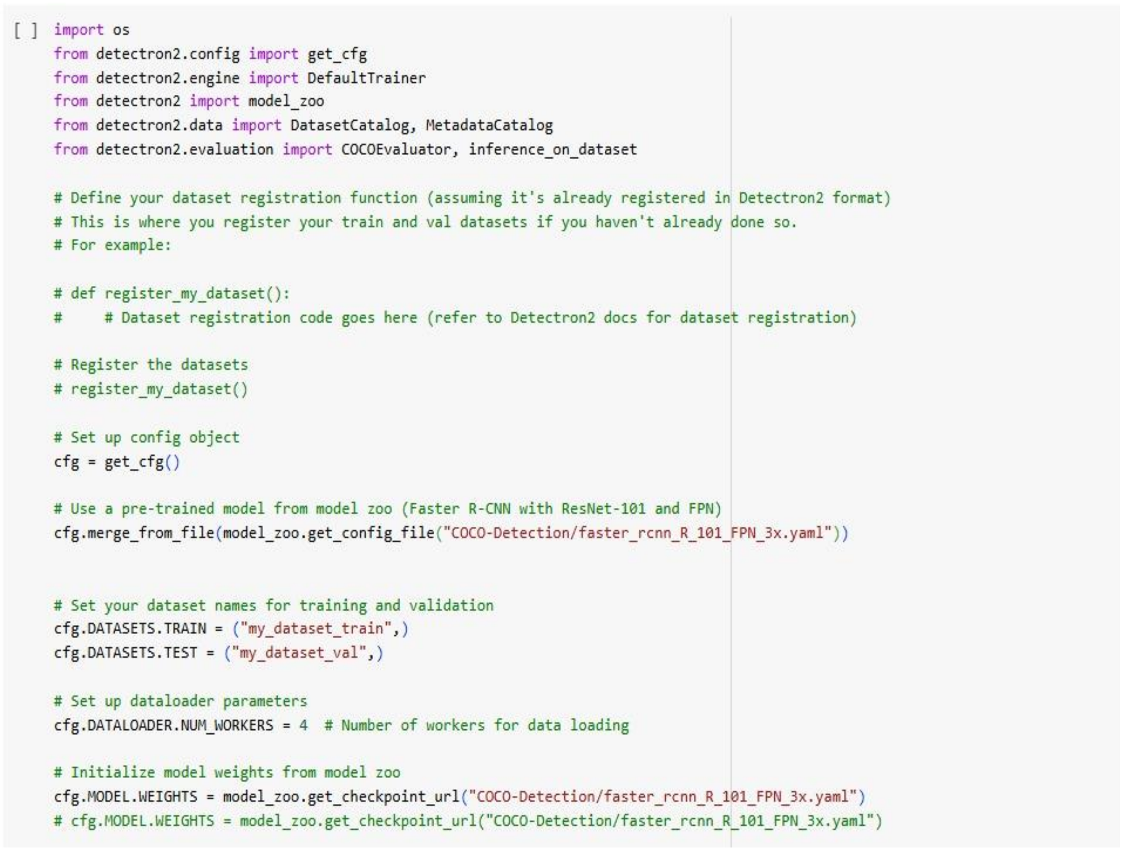
Training

```
[ ] ! python train.py --name Runs_YOLOV5n_Egypt_hg --img 640 --batch 64 --epochs 100 --data "datasets/dataset(pascal_VOC)/dataset.yaml" --weights yolov5n.pt --patience 30
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G     0.1027    0.02611     0.1215        121        640:  51%|████      | 31/61 [00:53<00:58,  1.95s/it]/content/yolov5/train.py:412: FutureWarning:
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G     0.1018    0.02607     0.1212        112        640:  52%|████▏     | 32/61 [00:55<00:53,  1.84s/it]/content/yolov5/train.py:412: FutureWarning
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G     0.1007    0.02606     0.1208        124        640:  54%|████▍     | 33/61 [00:56<00:44,  1.61s/it]/content/yolov5/train.py:412: FutureWarning
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G    0.09992    0.02595     0.1204        105        640:  56%|████▍     | 34/61 [00:57<00:42,  1.57s/it]/content/yolov5/train.py:412: FutureWarning:
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G    0.09906     0.0259     0.1201        117        640:  57%|█████     | 35/61 [00:58<00:37,  1.43s/it]/content/yolov5/train.py:412: FutureWarning
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G     0.0982    0.02592     0.1198        133        640:  59%|█████▍    | 36/61 [01:00<00:36,  1.44s/it]/content/yolov5/train.py:412: FutureWarning
      with torch.cuda.amp.autocast(amp):
           0/99      8.66G    0.09739    0.02585     0.1195        111        640:  61%|█████     | 37/61 [01:01<00:33,  1.39s/it]/content/yolov5/train.py:412: FutureWarning:
```

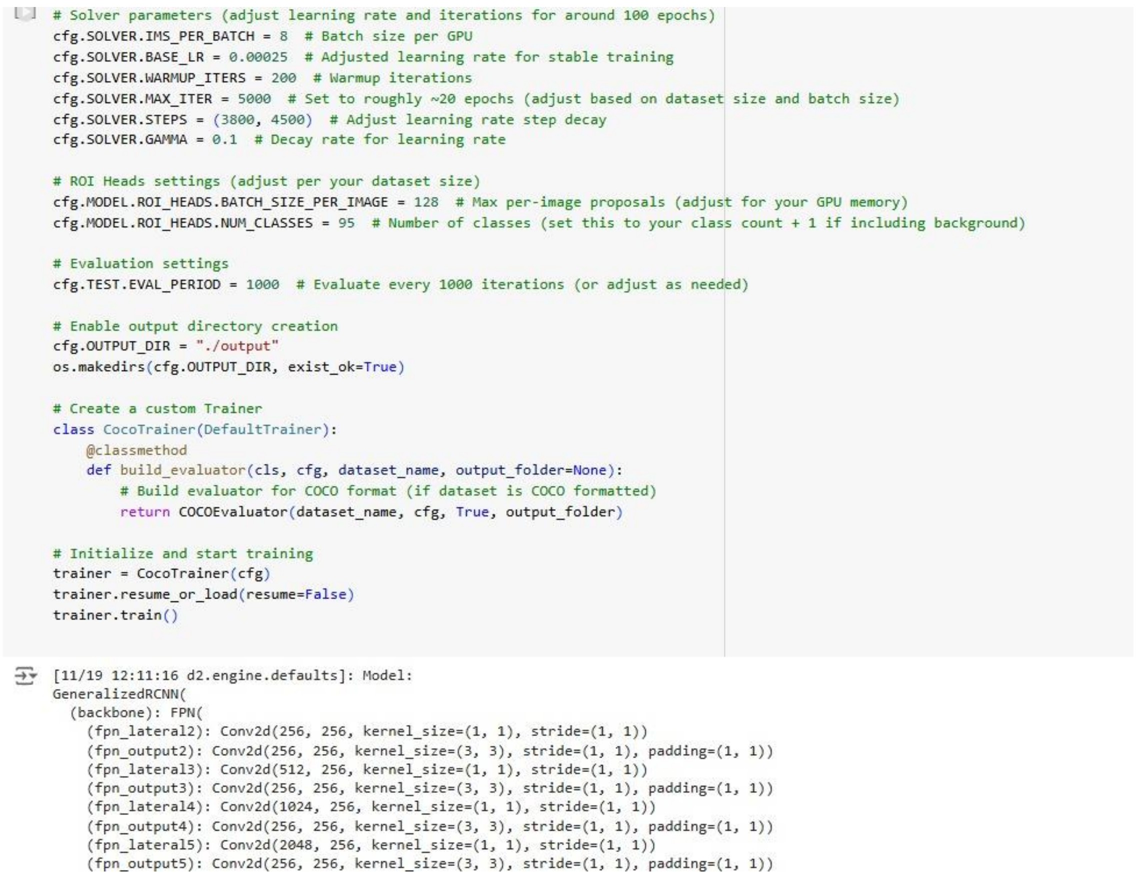
Figure 18: code for YOLOv5 training epochs

III. Faster R-CNN Training Model:

Faster R-CNN model was qualified on the hieroglyph dataset for 5,000 steps.



```
[ ] import os
    from detectron2.config import get_cfg
    from detectron2.engine import DefaultTrainer
    from detectron2 import model_zoo
    from detectron2.data import DatasetCatalog, MetadataCatalog
    from detectron2.evaluation import COCOEvaluator, inference_on_dataset

    # Define your dataset registration function (assuming it's already registered in Detectron2 format)
    # This is where you register your train and val datasets if you haven't already done so.
    # For example:

    # def register_my_dataset():
    #     # Dataset registration code goes here (refer to Detectron2 docs for dataset registration)

    # Register the datasets
    # register_my_dataset()

    # Set up config object
    cfg = get_cfg()

    # Use a pre-trained model from model zoo (Faster R-CNN with ResNet-101 and FPN)
    cfg.merge_from_file(model_zoo.get_config_file("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml"))


    # Set your dataset names for training and validation
    cfg.DATASETS.TRAIN = ("my_dataset_train",)
    cfg.DATASETS.TEST = ("my_dataset_val",)

    # Set up dataloader parameters
    cfg.DATALOADER.NUM_WORKERS = 4  # Number of workers for data loading

    # Initialize model weights from model zoo
    cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
    # cfg.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-Detection/faster_rcnn_R_101_FPN_3x.yaml")
```



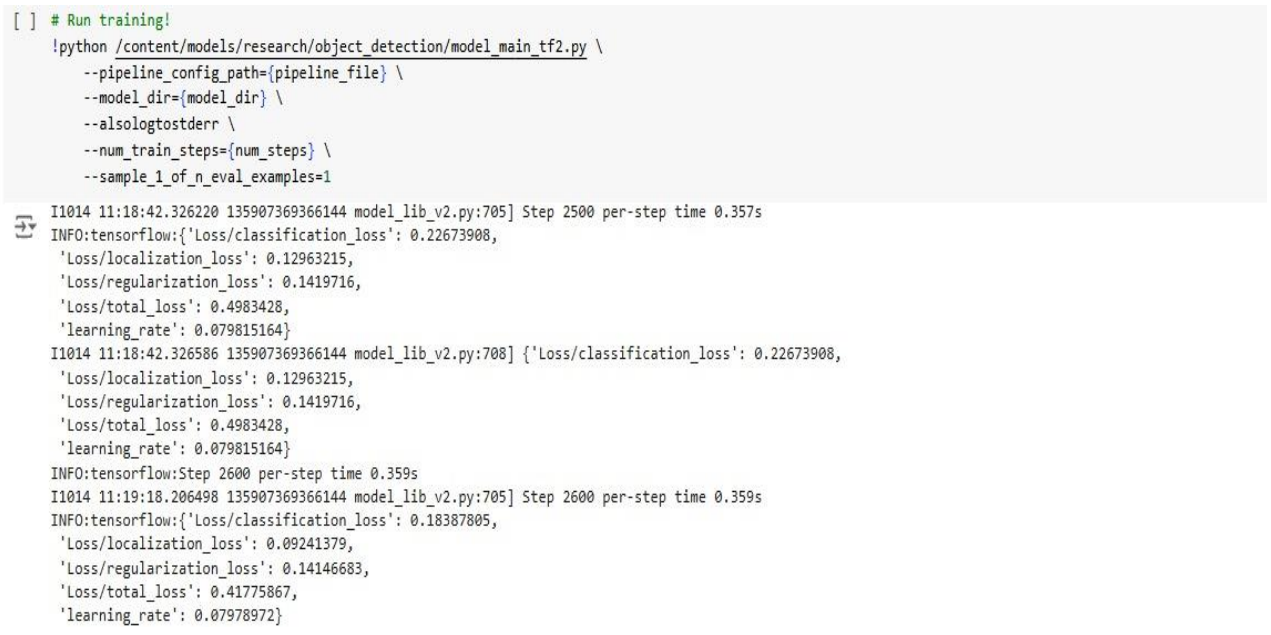
```
# Solver parameters (adjust learning rate and iterations for around 100 epochs)
cfg.SOLVER.IMS_PER_BATCH = 8  # Batch size per GPU
cfg.SOLVER.BASE_LR = 0.00025  # Adjusted learning rate for stable training
cfg.SOLVER.WARMUP_ITERS = 200  # Warmup iterations
cfg.SOLVER.MAX_ITER = 5000  # Set to roughly ~20 epochs (adjust based on dataset size and batch size)
cfg.SOLVER.STEPS = (3800, 4500)  # Adjust learning rate step decay
cfg.SOLVER.GAMMA = 0.1  # Decay rate for learning rate

# ROI Heads settings (adjust per your dataset size)
cfg.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128  # Max per-image proposals (adjust for your GPU memory)
cfg.MODEL.ROI_HEADS.NUM_CLASSES = 95  # Number of classes (set this to your class count + 1 if including background)

# Evaluation settings
cfg.TEST.EVAL_PERIOD = 1000  # Evaluate every 1000 iterations (or adjust as needed)

# Enable output directory creation
cfg.OUTPUT_DIR = "./output"
os.makedirs(cfg.OUTPUT_DIR, exist_ok=True)

# Create a custom Trainer
class CocoTrainer(DefaultTrainer):
    @classmethod
    def build_evaluator(cls, cfg, dataset_name, output_folder=None):
        # Build evaluator for COCO format (if dataset is COCO formatted)
        return COCOEvaluator(dataset_name, cfg, True, output_folder)

# Initialize and start training
trainer = CocoTrainer(cfg)
trainer.resume_or_load(resume=False)
trainer.train()
```

```
[11/19 12:11:16 d2.engine.defaults]: Model:
GeneralizedRCNN(
  (backbone): FPN(
    (fpn_lateral2): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral3): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral4): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output4): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (fpn_lateral5): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
    (fpn_output5): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

Figure 19: code for Fast R-CNN training steps

IV. SSD Model:

Prepared an SSD model using the hieroglyph dataset for 20,000 steps using the Tensor-Flow framework.



```
[ ] # Run training!
    !python /content/models/research/object_detection/model_main_tf2.py \
        --pipeline_config_path={pipeline_file} \
        --model_dir={model_dir} \
        --alsologtostderr \
        --num_train_steps={num_steps} \
        --sample_1_of_n_eval_examples=1
```

```
I1014 11:18:42.326220 135907369366144 model_lib_v2.py:705] Step 2500 per-step time 0.357s
INFO:tensorflow:{'Loss/classification_loss': 0.22673908,
 'Loss/localization_loss': 0.12963215,
 'Loss/regularization_loss': 0.1419716,
 'Loss/total_loss': 0.4983428,
 'learning_rate': 0.079815164}
I1014 11:18:42.326586 135907369366144 model_lib_v2.py:708] {'Loss/classification_loss': 0.22673908,
 'Loss/localization_loss': 0.12963215,
 'Loss/regularization_loss': 0.1419716,
 'Loss/total_loss': 0.4983428,
 'learning_rate': 0.079815164}
INFO:tensorflow:Step 2600 per-step time 0.359s
I1014 11:19:18.206498 135907369366144 model_lib_v2.py:705] Step 2600 per-step time 0.359s
INFO:tensorflow:{'Loss/classification_loss': 0.18387805,
 'Loss/localization_loss': 0.09241379,
 'Loss/regularization_loss': 0.14146683,
 'Loss/total_loss': 0.41775867,
 'learning_rate': 0.07978972}
```

Figure 20: code for SSD training steps

4.7 Model Evaluation

The evaluation of models is done to know the robustness and effectiveness of all four models. This was done on the basis of dividing the dataset into training and validation set by the ratio of 80:20. All four models were evaluated by using the evaluation metric which includes mAP (mean Average Precision), recall and F1Score.

I. YOLOv8 Model:

The YOLOv8, that is an enhanced model of YOLOv5 works in the same manner, but the anchor free mechanism is done by itself, and this model has got the highest mAP that is 0.975, the precision value is 0.943, the recall is 0.977 and F1 score value is 0.960.

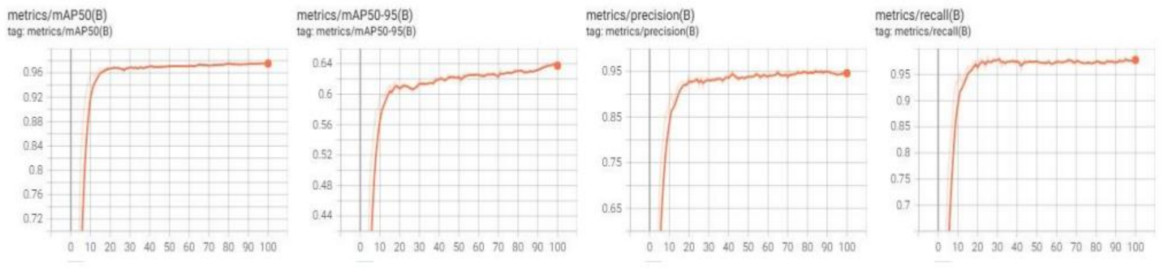


Figure 21: Performance metrics of model during training, including mAP@50, mAP@50-95, precision, and recall trends across epochs

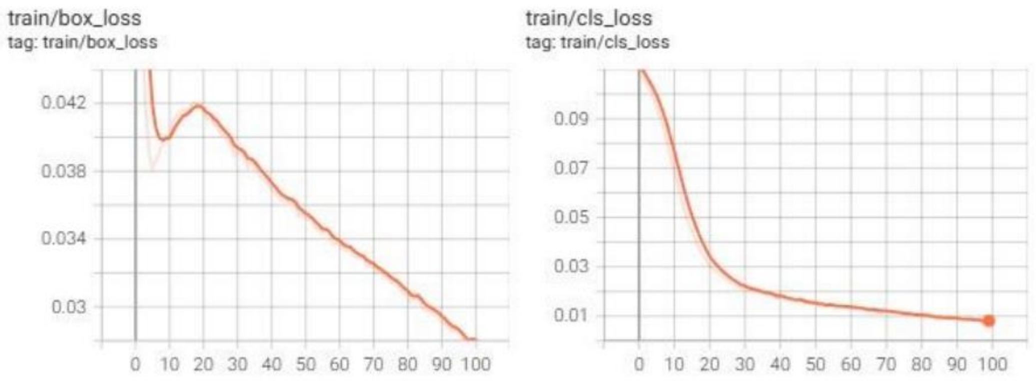


Figure 22: Training loss curves for YOLOv8 showing box loss and classification loss reduction over epochs

II. YOLOv5 Model:

This model uses features like fusion PANet that helps in object detection of various sizes and shapes. Here the anchor-based training is also done where sizes and dimensions are adjusted accordingly. The values here include as mAP achieved is 0.970, the precision is 0.93, the recall is 0.970 and F1 score is 0.960

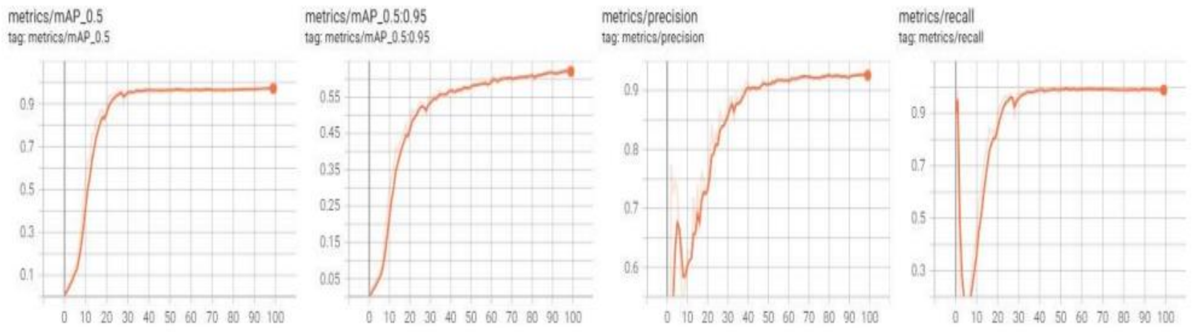


Figure 23: code for SSD training steps Performance metrics of YOLOv5 during training, including mAP@0.5, mAP@0.5:0.results in the form of accuracy, precision, recall over epochs

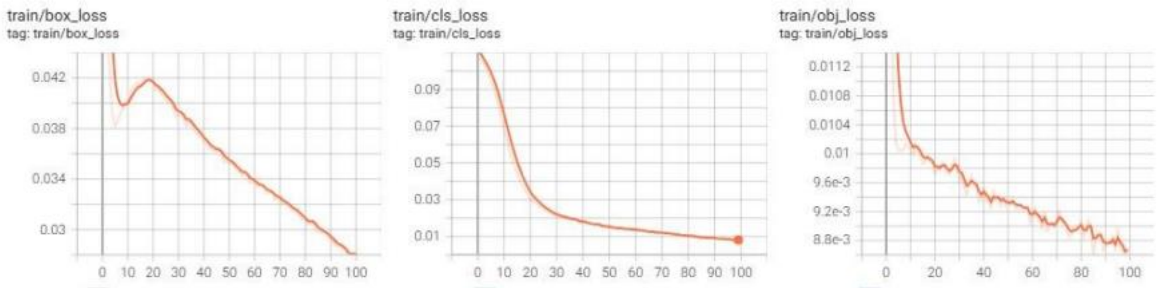
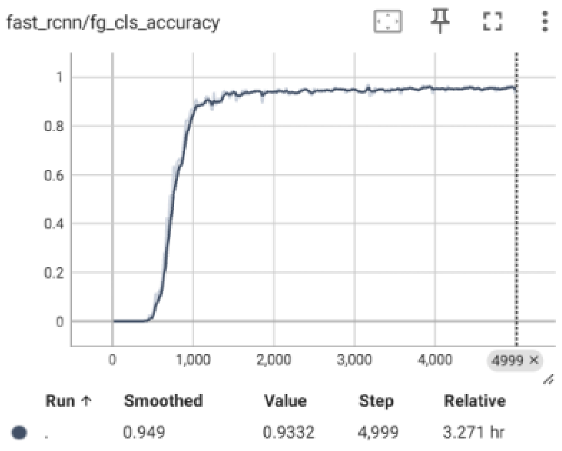


Figure 24: Training loss trends for YOLOv5 showing box loss, classification loss, and abjectness loss over epochs

III. Faster R-CNN Training Model:

The Approach that used for this model was region-based where the first region-based candidates originated and are combined for further alignment. The key metrics values that are achieved include the mAP that is 0.937 and recall value is 0.650 followed by null values of precision and F1 score.



Run ↑	Smoothed	Value	Step	Relative
● .	0.949	0.9332	4,999	3.271 hr

Figure 25: Classification accuracy curve of Fast R-CNN showing foreground class accuracy over training steps

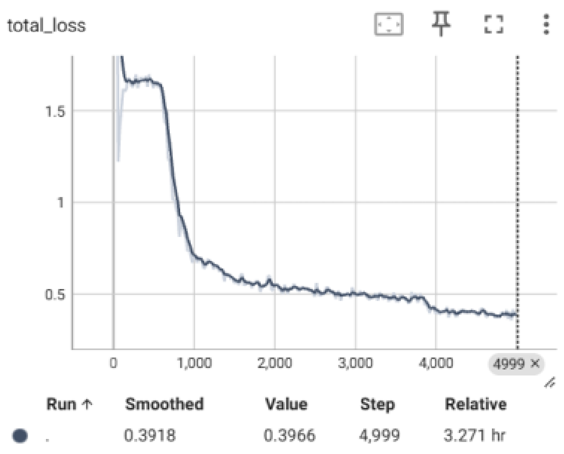


Figure 26: Total loss curve of Fast R-CNN showing loss reduction over training steps

IV. SSD Model:

The model showed exceptional result because of its real time detection capability. The evaluated key metrics shows result as accuracy being 67.54% and null values of precision, recall and F1 score.

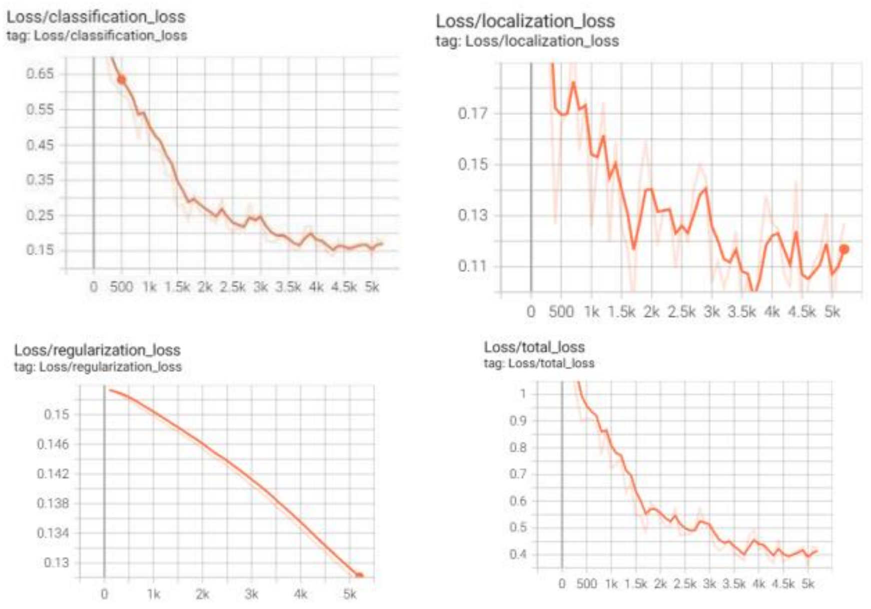
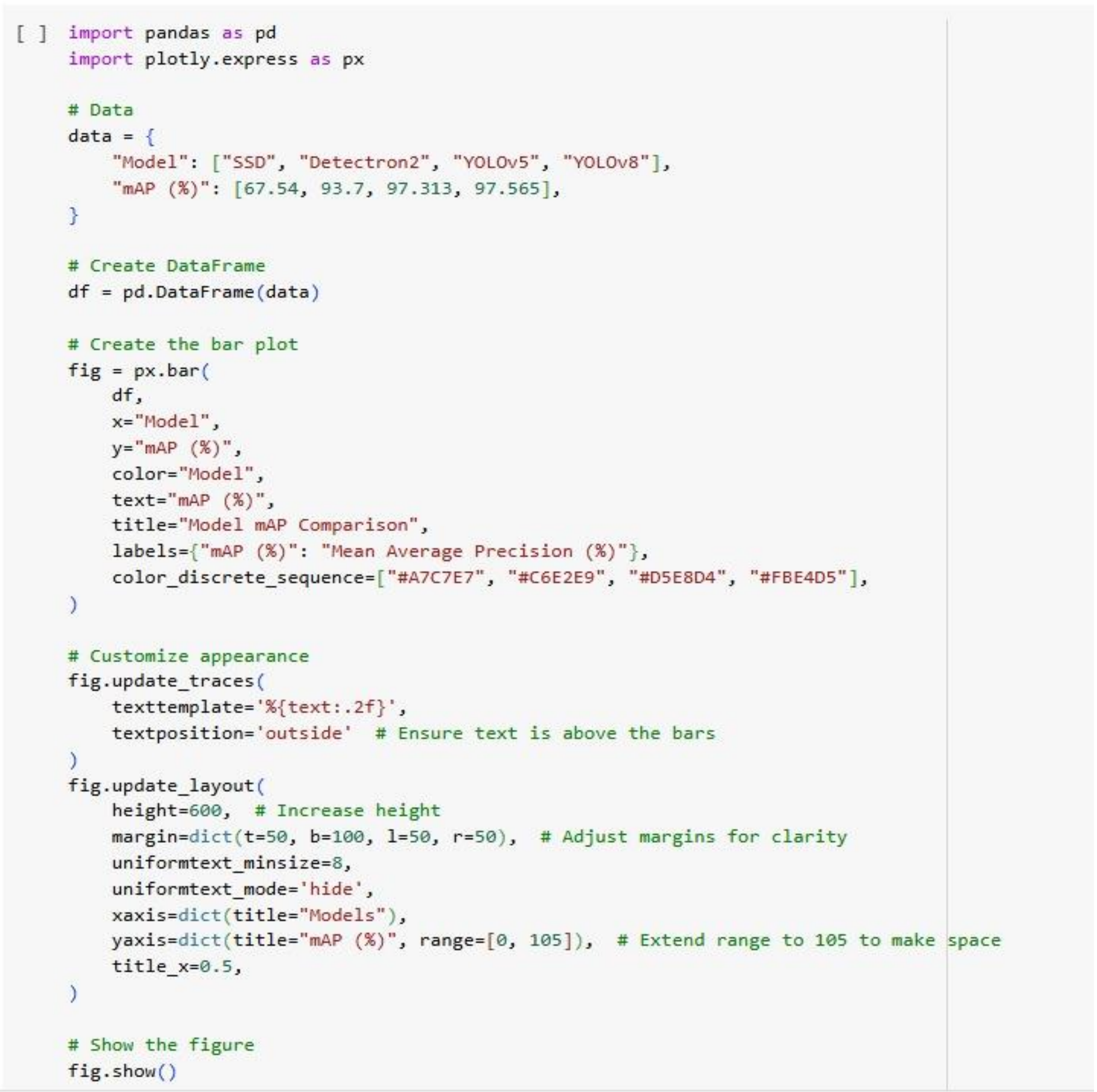


Figure 27: Training loss curves for SSD model showing classification loss, localization loss, regularization loss, and total loss over training iterations.

V. Model Comparision:

The comparison of all models has shown that YOLOv8 has the highest mAP of 0.975 among all the models that has done anchor free techniques by itself. Whereas the Single Shot Detection model has got the lowest mAP that is 67.46 as it focused on larger objects and did not focus on smaller objects. Although the Faster R-CNN got the average precision of 0.937 and YOLOv5 achieved 0.970 but showed great results overall.



```
[ ] import pandas as pd
    import plotly.express as px

    # Data
    data = {
        "Model": ["SSD", "Detectron2", "YOLOv5", "YOLOv8"],
        "mAP (%)": [67.54, 93.7, 97.313, 97.565],
    }

    # Create DataFrame
    df = pd.DataFrame(data)

    # Create the bar plot
    fig = px.bar(
        df,
        x="Model",
        y="mAP (%)",
        color="Model",
        text="mAP (%)",
        title="Model mAP Comparison",
        labels={"mAP (%)": "Mean Average Precision (%)"},
        color_discrete_sequence=["#A7C7E7", "#C6E2E9", "#D5E8D4", "#FBE4D5"],
    )

    # Customize appearance
    fig.update_traces(
        texttemplate='%{text:.2f}',
        textposition='outside'  # Ensure text is above the bars
    )
    fig.update_layout(
        height=600,  # Increase height
        margin=dict(t=50, b=100, l=50, r=50),  # Adjust margins for clarity
        uniformtext_minsize=8,
        uniformtext_mode='hide',
        xaxis=dict(title="Models"),
        yaxis=dict(title="mAP (%)", range=[0, 105]),  # Extend range to 105 to make space
        title_x=0.5,
    )

    # Show the figure
    fig.show()
```

Figure 28: code for model comparison

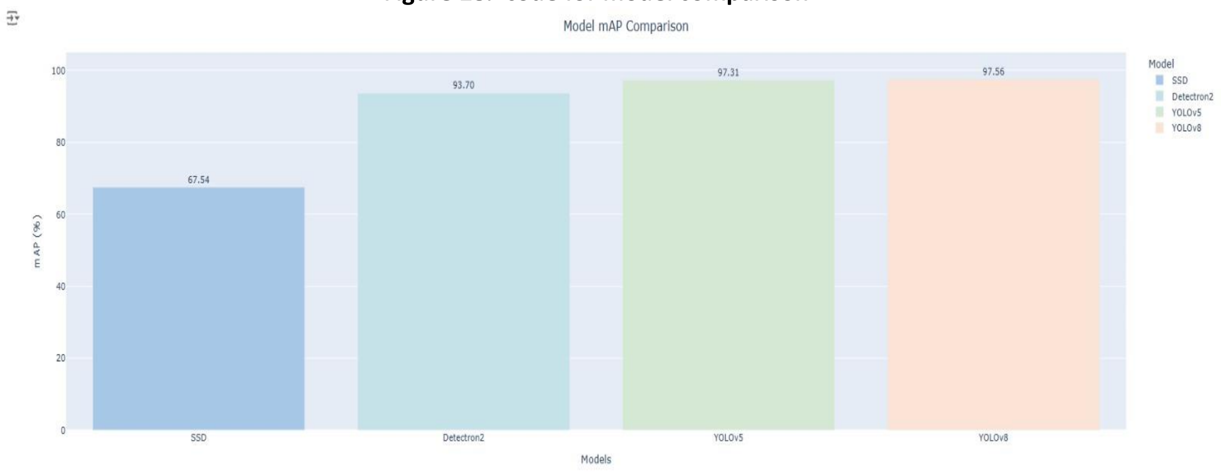


Figure 29: Comparison of mAP performance across SSD, Fast R-CNN, YOLOv5, and YOLOv8 models.

References

Barucci, A., Cucci, C., Franci, M., Loschiavo, M. & Argenti, F. (2021). A deep learning approach to ancient Egyptian hieroglyphs classification. IEEE Access, 9, pp. 123438-123447.

Corsini, M., Ferdani, D., Kuijper, A. & Kutlu, H. (2024). Unsupervised detection and localization of Egyptian hieroglyphs.

Elnabawy, R., Elias, R., Salem, M.A.M. & Abdennadher, S. (2021). Extending Gardiner's code for hieroglyphic recognition and English mapping. Multimedia Tools and Applications, 80, pp. 3391-3408.

Franken, M. & van Gemert, J.C. (2013). Automatic Egyptian hieroglyph recognition by retrieving images as texts. In Proceedings of the 21st ACM International Conference on Multimedia, pp. 765-768.

Cucci, C., Guidi, T., Picollo, M., Stefani, L., Python, L., Argenti, F. & Barucci, A. (2024). Hyperspectral imaging and convolutional neural networks for augmented documentation of ancient Egyptian artefacts. Heritage Science, 12(1), pp. 75.