St. Francis
College for Women
Begumpet, Hyderabad-500016
(Autonomous & Affiliated to Osmania University)

IMAGE SEGMENTATION

MSc (Data Science) Dissertation

Submitted to St. Francis College for Women in partial fulfillment of the requirements for the award of the Degree of

Masters of Science

By

NAGILLA SAI PRATHYUSHA
121322030019

Under the guidance of Ms.

B. JYOTHI REDDY

Department of Computer Science
St. Francis College for Women
Begumpet, Hyderabad – 500 016
(Autonomous and affiliated to Osmania University)

MARCH – 2024

St. Francis College for Women

Begumpet, Hyderabad-500016

(Autonomous & Affiliated to Osmania University)

CERTIFICATE

This is to certify that this bonafide project work titled "**Image Segmentation**", has been carried out by **Nagilla Sai Prathyusha** bearing Roll No: **121322030019**, towards partial fulfillment of the requirements for the award of Degree of **Masters in Data Science** from St. Francis College for Women, Begumpet for the academic year **2023- 2024.**

Supervisor **PG Coordinator**

External Examiner **Controller of Examinations**

DECLARATION

The current study "**Image Segmentation**" has been carried out under the supervision of **Ms. KHALIDA TABASSUM**, Assistant Professor, Department of Computer Science, St. Francis College for Women. We hereby declare that the present study that has been carried out by me, **Nagilla Sai Prathyusha**, during March 2023-2024 is original and no part of it has been carried out prior to this date.

Date:

Signature of Candidate:

ABSTRACT

This project focuses on image segmentation using deep learning and computer vision techniques, leveraging the COCO 2014 dataset. The COCO dataset, comprising over 330,000 diverse images across 80 object categories, serves as a rich source for training and evaluation. The workflow encompasses key steps from data setup to model training and visualization.

Initial steps involve installing essential tools such as pycocotools and importing requisite libraries. Subsequently, the COCO dataset is initialized, and categories, images, and annotations are loaded. Filtering and processing steps follow, including the extraction of relevant category information and the generation of image masks for object segmentation.

The project then proceeds to showcase the exploration and analysis of the dataset, visualizing category distributions through images and pie charts. Utilizing filtered images with annotations, the study demonstrates the practical aspects of working with the COCO dataset. The core of the project lies in the implementation of a U-Net model for image segmentation. This involves training the model on the generated image and mask datasets, and subsequently, predicting and visualizing segmentation masks.

Overall, this project provides a comprehensive guide for individuals interested in image segmentation using the COCO 2014 dataset, covering essential steps from data preprocessing to model deployment. The results and insights gained from this project contribute to the broader understanding of deep learning applications in computer vision and image analysis.

ACKNOWLEDGEMENT

The success and outcome of this project required a lot of guidance and assistance from many people and we are extremely privileged to have received this all along the completion of our project.

We respect and thank the Head of the Department, Ms. D. Sowjenya for giving us an opportunity to do the project work and for all the support.

We extend our gratitude to our project guide Ms. B. Jyothi Reddy for providing us with such great support and guidance that made us complete the project duly. We are also thankful to them for providing us with all the necessary information which was vital to develop this digitized system.

Furthermore, we would also like to acknowledge with much appreciation the crucial role of our family, friends, and teachers who were our constant motivators throughout the project. Also, we are grateful to all the computer lab assistants who permitted us to use all the required equipment and materials to complete the project.

CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

In the realm of computer vision and deep learning, image segmentation stands out as a pivotal task, enabling the precise identification and delineation of objects within images. This project embarks on a comprehensive exploration of image segmentation, leveraging the extensive COCO 2014 dataset—a repository of over 330,000 images capturing diverse scenes and objects across 80 categories. The intricate nature of this dataset provides a fertile ground for training and evaluating deep learning models.

The workflow unfolds systematically across 18 steps, encompassing critical phases such as data setup, exploratory data analysis, and model implementation. From the initial installation of pycocotools to the final visualization of segmentation masks, each step contributes to a cohesive narrative aimed at guiding practitioners through the complexities of image segmentation using the COCO dataset.

The project's focal point lies in the implementation of a U-Net model, a proven architecture for image segmentation tasks. The journey involves not only loading and filtering data but also extracting meaningful insights through visualizations, including category distribution analyses and image annotations. By following this project, enthusiasts and practitioners alike will gain a deep understanding of the practical aspects of utilizing the COCO 2014 dataset for training and deploying image segmentation models in the realm of deep learning and computer vision.

1.1 OBJECTIVES

- **Dataset Integration:** Install and configure pycocotools for seamless interaction with the COCO 2014 dataset.
- **Data Exploration:** Analyze the extensive COCO dataset, comprising over 330,000 images across 80 diverse categories, to comprehend its complexity.
- **API Initialization:** Initialize the COCO dataset API to efficiently retrieve image, annotation, and category information.
- **Data Filtering:** Systematically filter and categorize relevant object classes to streamline the dataset for image segmentation tasks.
- **Exploratory Visualization:** Develop visualizations to gain insights into category distribution, uncovering dataset diversity and potential biases.
- **Image and Annotation Display:** Present images with annotations, offering a visual representation of dataset content and segmentation challenges.
- **Mask Generation:** Implement algorithms for generating masks, a crucial step in image segmentation for precise object identification.
- **Dataset Preparation:** Create formatted image and mask datasets, ensuring compatibility for effective U-Net model training.
- **U-Net Model Training:** Deploy and train a U-Net model, a proven architecture for image segmentation, using prepared datasets.

- **Prediction and Visualization:** Apply the trained U-Net model to predict segmentation masks, visually assessing its performance in object delineation.
- **Insightful Analyses:** Conduct analyses on category distribution and image annotations, extracting meaningful insights from segmented data.
- **Documentation:** Compile comprehensive documentation outlining the project's methodology, results, and insights for knowledge transfer and replication.

1.2 PURPOSE

- **Accurate Object Delineation:** The primary goal is to achieve precise identification and delineation of objects within images through advanced image segmentation techniques.
- **U-Net Model Implementation:** Train and implement a U-Net model, a proven architecture for image segmentation, utilizing the diverse COCO 2014 dataset for reliable results.
- **Exploratory Data Analysis:** Gain deep insights into the COCO dataset's complexities, exploring category distribution and potential biases to enhance understanding and model performance.
- **Practical Application of Deep Learning:** Apply state-of-the-art deep learning techniques to solve real-world challenges in computer vision, emphasizing practical applications and tangible outcomes.
- **Knowledge Transfer and Documentation:** Compile comprehensive documentation to facilitate knowledge transfer, providing a valuable resource for practitioners to replicate and understand the image segmentation process.
- **Effective Visualization Techniques:** Develop and utilize visualization techniques to enhance the interpretability of segmentation results, aiding in the practical application and communication of findings.
- **Contribution to Research and Development:** Contribute to the broader field of computer vision by applying sophisticated methods to a substantial dataset, advancing the understanding and capabilities of image segmentation.

1.3 SCOPE AND APPLICABILITY

- Dataset Exploration and Diversity: Explore the vast and diverse COCO 2014 dataset, containing over 330,000 images across 80 categories, offering a broad scope for understanding and applying image segmentation techniques.
- Algorithmic Experimentation: Experiment with various image segmentation algorithms, with a particular focus on the implementation of a U-Net model, to assess their effectiveness in accurately delineating objects within the dataset.
- Adaptability and Scalability: Design the project with adaptability and scalability in mind, allowing for potential future integration with other datasets and scenarios, ensuring the model's broader applicability.
- Medical Imaging Precision: Apply the developed segmentation model to medical imaging scenarios, where precise identification and delineation of organs or anomalies within images are crucial for diagnostic purposes.

- Autonomous Vehicles Object Recognition: Extend the applicability to autonomous vehicles, using the segmentation model to enhance object recognition for improved path planning, obstacle avoidance, and overall safety.
- Environmental Monitoring and Analysis: Apply the segmentation techniques to satellite or drone imagery for environmental monitoring, enabling the analysis of land use, vegetation, and ecological changes for research and conservation purposes.

CHAPTER 2
SURVEY OF TECHNOLOGIES

CHAPTER-2

SURVEY OF TECHNOLOGIES

Certainly, completing a project on image segmentation using the COCO 2014 dataset involves a combination of surveys and utilization of specific technologies.

Here are detailed insights into the surveys and technologies relevant to the project:

2.1 PROGRAMMING LANGUAGES

The programming languages commonly used in deep learning and computer vision projects, including image segmentation, are:

Python:

- Python is a predominant language for deep learning due to its extensive libraries and frameworks, such as TensorFlow and PyTorch, which provide tools for building and training neural networks.

TensorFlow:

- TensorFlow is an open-source machine learning library developed by Google. It offers high-level APIs for building and training neural networks, making it widely used in image segmentation projects.

Keras:

- An open-source deep learning library that is integrated with TensorFlow. Keras simplifies the process of building and training neural networks, and it is extensively used in this code for defining model architectures, layers, and compiling the model.

2.2 FRAMEWORKS AND LIBRARIES

The code utilizes the following frameworks and libraries:

- **TensorFlow:** TensorFlow is a popular open-source machine learning framework developed by the Google Brain team. It provides tools and libraries for building and training deep learning models.
- **Keras:** Keras is a high-level neural networks API written in Python and capable of running on top of TensorFlow. In this code, Keras is used for building and configuring the neural network architecture.
- **NumPy:** NumPy is a fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with mathematical functions to operate on these arrays.
- **Matplotlib:** Matplotlib is a 2D plotting library for Python. It is used for creating visualizations, such as displaying sample images, ground truth masks, and predicted masks in this project.

- **os:** The `os` module provides a way to interact with the operating system, and in this code, it's likely used for file path manipulations.
- **sys:** The `sys` module provides access to some variables used or maintained by the Python interpreter and functions that interact with the interpreter. It's typically used for system-related functionalities.

These frameworks and libraries collectively enable the implementation of a U-Net model for image segmentation, including data processing, model training, and result visualization.

2.3 DEVELOPMENT TOOLS

The development tools used in this project are:

- **Kaggle Environment:** The code seems to be developed and executed in a Kaggle environment, as indicated by the paths to input directories (e.g., '/10aggle/input') and the use of Kaggle-specific commands. Kaggle provides a platform with pre-installed data science and machine learning tools, making it convenient for collaborative projects.
- **GitHub or Version Control System:** Version control systems like Git, and platforms like GitHub, may have been used for collaborative development, code sharing, and version tracking.
- **TensorBoard:** TensorBoard, a visualization tool provided by TensorFlow, may have been used for monitoring and visualizing the training process, model architecture, and performance metrics.
- **Command-Line Interface (CLI):** The use of the command line for running scripts or managing project-related tasks is common in machine learning projects. The code might be executed and managed using commands in a terminal or command prompt.
- **Data Visualization Tools:** Matplotlib is used within the code for creating visualizations. Additionally, tools like Seaborn or Plotly could be employed for more advanced data visualization if needed.
- **Google Colab:** Google Colab, a cloud-based Jupyter Notebook environment provided by Google, might have been used for training the model using GPU resources.

These development tools collectively facilitate the coding, testing, and visualization processes in the development of the image segmentation project.

CHAPTER 3
REQUIREMENTS

CHAPTER 3

REQUIREMENTS

3.1 PROBLEM DEFINITION

This project entails the development of an advanced image segmentation system utilizing deep learning and computer vision techniques, with a primary focus on the COCO 2014 dataset. Image segmentation, the process of partitioning images into meaningful segments, is critical for accurate object identification. Leveraging the extensive COCO dataset, comprising over 330,000 diverse images across 80 object categories, the project aims to guide participants through a systematic 18-step workflow.

This includes data setup, exploration, and the implementation of a U-Net model—a proven architecture for image segmentation tasks. The project further emphasizes the practical application of acquired skills in domains like object recognition, medical image analysis, and scene understanding.

By the project's conclusion, participants are expected to gain a comprehensive understanding of image segmentation, proficiency in working with the COCO dataset, and the practical ability to implement and deploy segmentation models using deep learning and computer vision techniques. This project definition sets the stage for a hands-on exploration of the intricacies involved in image segmentation within the specified framework.

3.2 DESCRIPTION

This image segmentation project is centered on implementing advanced techniques in deep learning and computer vision to partition images effectively, aiming for precise object delineation. The primary dataset employed is the COCO 2014 dataset, renowned for its diversity with over 330,000 images spanning 80 object categories.

The project follows a systematic 18-step workflow, beginning with the installation of critical tools such as pycocotools and the exploration of relevant libraries. Participants will navigate through key phases, including data setup, exploratory data analysis, and the pivotal implementation of a U-Net model—a well-established architecture for image segmentation tasks.

The COCO dataset's versatility allows for targeted model training by filtering specific object categories. Visualization techniques, encompassing category distribution analyses and annotated images, enhance the project's interpretability. The acquired skills find practical application across domains like object recognition, medical image analysis, and scene understanding.

By the project's conclusion, participants will emerge with comprehensive insights into image segmentation, hands-on experience with the COCO dataset, and the ability to implement and

deploy segmentation models using cutting-edge techniques in deep learning and computer vision.

3.3 SYSTEM SPECIFICATION:

Here are the general specifications relevant to your project:

3.3.1 HARDWARE SPECIFICATIONS

Google Colab provides virtual machines (VMs) with varying configurations based on the type of hardware accelerator selected.

The primary hardware components that users can leverage in Google Colab are Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs).

- **Graphics Processing Units (GPUs):** Google Colab offers free access to GPU resources, typically NVIDIA Tesla K80, T4, P4, or P100 GPUs. These GPUs accelerate deep learning tasks, particularly model training, by parallelizing computations.
- **Tensor Processing Units (TPUs):** For users interested in even more powerful acceleration, Google Colab provides access to TPUs. TPUs are custom-designed hardware by Google specifically for machine learning workloads, offering significant speedups for certain types of tasks.
- **CPU:** Google Colab instances also come with multi-core CPUs. While the CPU is less relevant for deep learning tasks (which are primarily offloaded to GPUs or TPUs), it still plays a role in general computing tasks within the Colab environment.
- **Memory (RAM):** The amount of available RAM in Google Colab instances may vary. Users are allocated a specific amount of RAM for their sessions. For large datasets or memory-intensive operations, it's crucial to be mindful of available RAM.

3.3.2 SOFTWARE SPECIFICATIONS

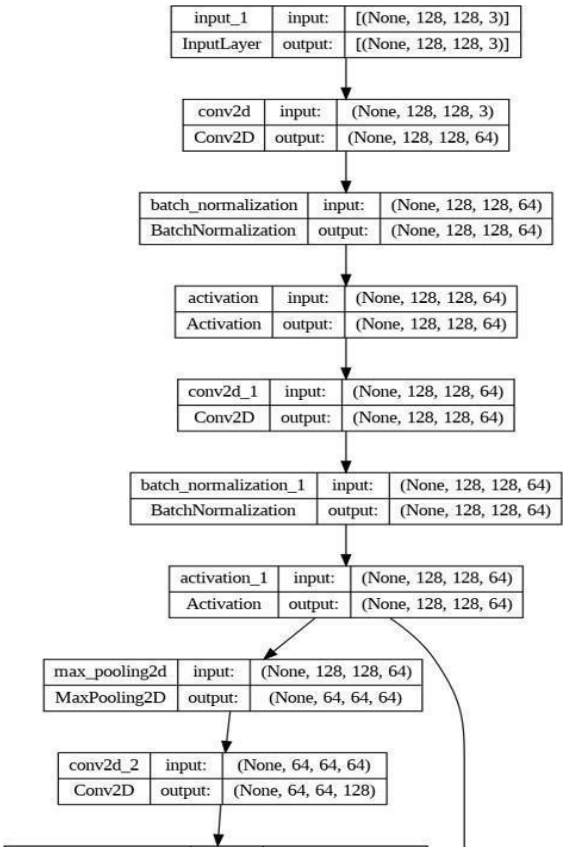
The software specifications for your project, developed in a Google Colab environment, include the following:

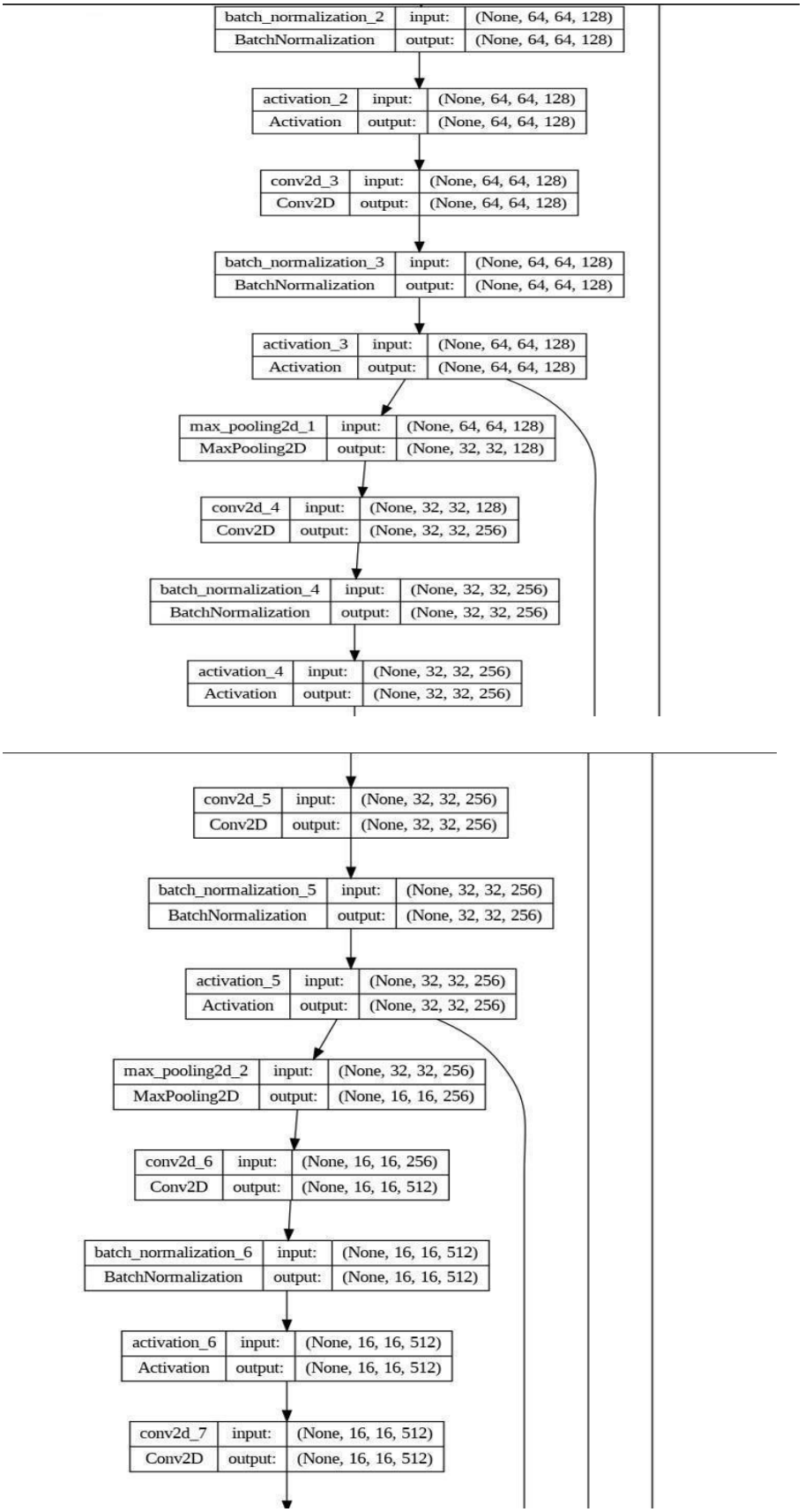
- **Python:** The primary programming language used for developing machine learning models and implementing the project.
- **TensorFlow:** An open-source machine learning framework developed by Google. TensorFlow is used for building and training deep learning models, and it comes pre-installed in Google Colab.
- **Keras:** A high-level neural networks API that runs on top of TensorFlow. Keras simplifies the process of building and training neural networks, and it is integrated with TensorFlow in the provided code.
- **NumPy:** A fundamental package for scientific computing with Python. NumPy provides support for large, multi-dimensional arrays and matrices, and it is used for numerical operations in the project.

- **Matplotlib:** A 2D plotting library for Python used for creating visualizations within the Google Colab environment. In the provided code, Matplotlib is likely used for displaying sample images and visualizing results.
- **Google Colab Environment:** An online platform provided by Google that allows users to run Python code in a Jupyter Notebook environment with access to GPU and TPU resources. It facilitates collaborative coding and provides a seamless integration with Google Drive.
- **Operating System (OS):** Google Colab is accessed through a web browser and is OS-agnostic. The code can be developed and executed on any operating system, such as Windows, macOS, or Linux.

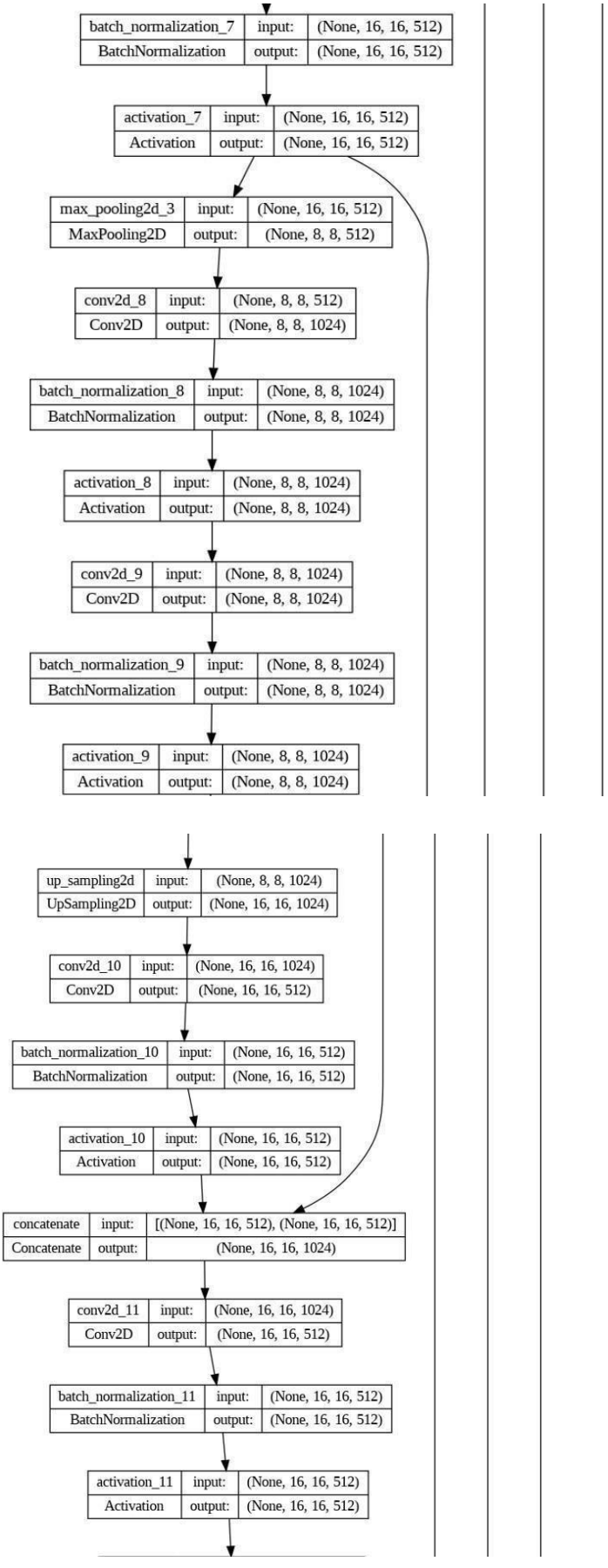
These software specifications collectively form the development and execution environment for your machine learning project within the Google Colab platform.

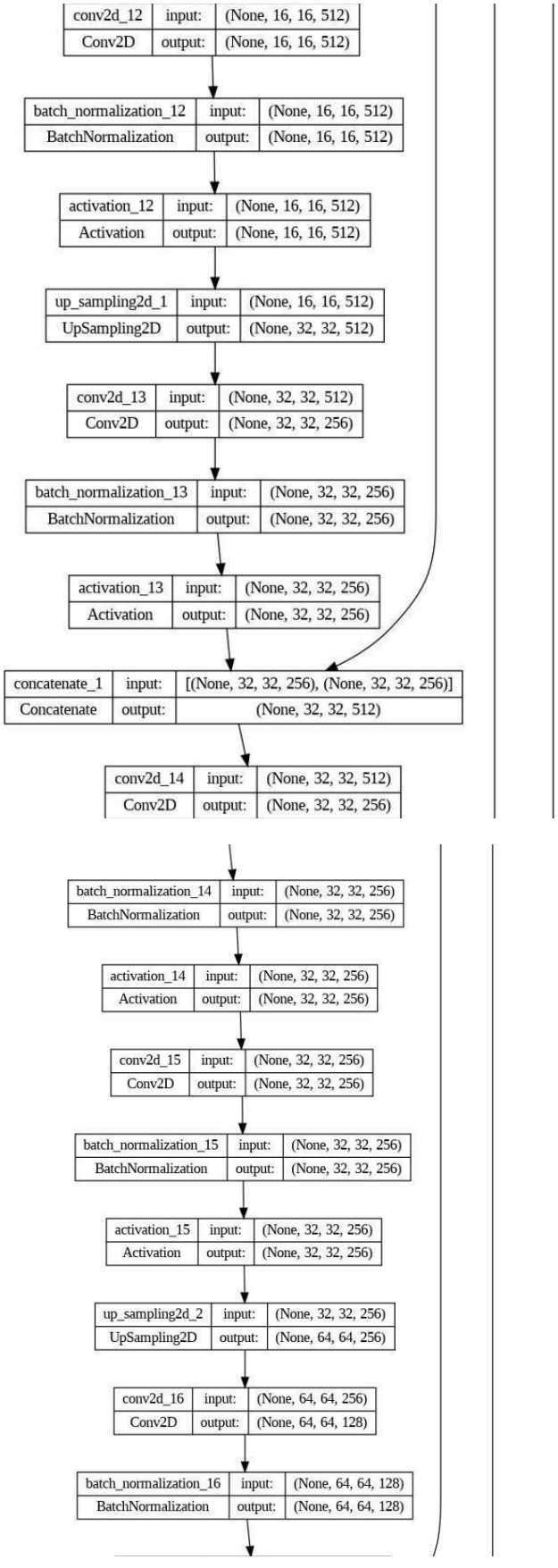
3. 4 MODELARCHITECTURE

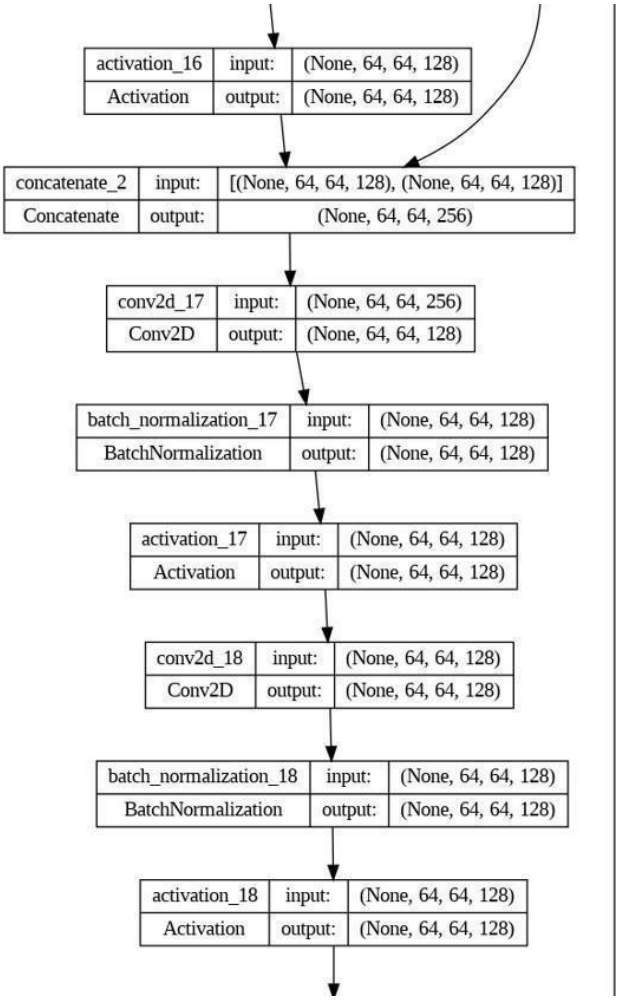


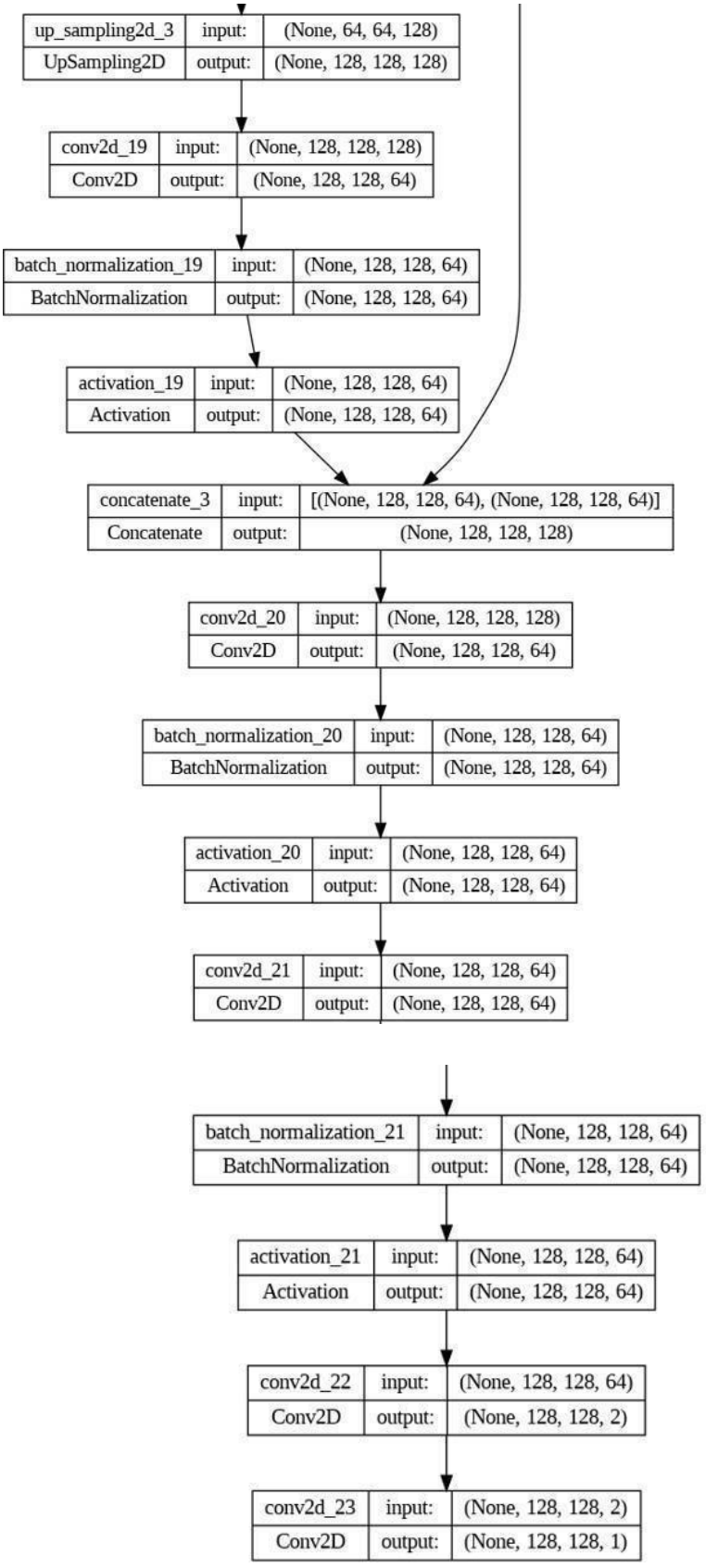

batch_normalization_2 | input: | (None, 64, 64, 128)
BatchNormalization | output: | (None, 64, 64, 128)
activation_2 | input: | (None, 64, 64, 128)
Activation | output: | (None, 64, 64, 128)
conv2d_3 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 128)
batch_normalization_3 | input: | (None, 64, 64, 128)
BatchNormalization | output: | (None, 64, 64, 128)
activation_3 | input: | (None, 64, 64, 128)
Activation | output: | (None, 64, 64, 128)
max_pooling2d_1 | input: | (None, 64, 64, 128)
MaxPooling2D | output: | (None, 32, 32, 128)
conv2d_4 | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_4 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
activation_4 | input: | (None, 32, 32, 256)
Activation | output: | (None, 32, 32, 256)

conv2d_5 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_5 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
activation_5 | input: | (None, 32, 32, 256)
Activation | output: | (None, 32, 32, 256)
max_pooling2d_2 | input: | (None, 32, 32, 256)
MaxPooling2D | output: | (None, 16, 16, 256)
conv2d_6 | input: | (None, 16, 16, 256)
Conv2D | output: | (None, 16, 16, 512)
batch_normalization_6 | input: | (None, 16, 16, 512)
BatchNormalization | output: | (None, 16, 16, 512)
activation_6 | input: | (None, 16, 16, 512)
Activation | output: | (None, 16, 16, 512)
conv2d_7 | input: | (None, 16, 16, 512)
Conv2D | output: | (None, 16, 16, 512)


batch_normalization_7 | input: | (None, 16, 16, 512)
BatchNormalization | output: | (None, 16, 16, 512)
activation_7 | input: | (None, 16, 16, 512)
Activation | output: | (None, 16, 16, 512)
max_pooling2d_3 | input: | (None, 16, 16, 512)
MaxPooling2D | output: | (None, 8, 8, 512)
conv2d_8 | input: | (None, 8, 8, 512)
Conv2D | output: | (None, 8, 8, 1024)
batch_normalization_8 | input: | (None, 8, 8, 1024)
BatchNormalization | output: | (None, 8, 8, 1024)
activation_8 | input: | (None, 8, 8, 1024)
Activation | output: | (None, 8, 8, 1024)
conv2d_9 | input: | (None, 8, 8, 1024)
Conv2D | output: | (None, 8, 8, 1024)
batch_normalization_9 | input: | (None, 8, 8, 1024)
BatchNormalization | output: | (None, 8, 8, 1024)
activation_9 | input: | (None, 8, 8, 1024)
Activation | output: | (None, 8, 8, 1024)
up_sampling2d | input: | (None, 8, 8, 1024)
UpSampling2D | output: | (None, 16, 16, 1024)
conv2d_10 | input: | (None, 16, 16, 1024)
Conv2D | output: | (None, 16, 16, 512)
batch_normalization_10 | input: | (None, 16, 16, 512)
BatchNormalization | output: | (None, 16, 16, 512)
activation_10 | input: | (None, 16, 16, 512)
Activation | output: | (None, 16, 16, 512)
concatenate | input: | [(None, 16, 16, 512), (None, 16, 16, 512)]
Concatenate | output: | (None, 16, 16, 1024)
conv2d_11 | input: | (None, 16, 16, 1024)
Conv2D | output: | (None, 16, 16, 512)
batch_normalization_11 | input: | (None, 16, 16, 512)
BatchNormalization | output: | (None, 16, 16, 512)
activation_11 | input: | (None, 16, 16, 512)
Activation | output: | (None, 16, 16, 512)


conv2d_12 | input: | (None, 16, 16, 512)
Conv2D | output: | (None, 16, 16, 512)
batch_normalization_12 | input: | (None, 16, 16, 512)
BatchNormalization | output: | (None, 16, 16, 512)
activation_12 | input: | (None, 16, 16, 512)
Activation | output: | (None, 16, 16, 512)
up_sampling2d_1 | input: | (None, 16, 16, 512)
UpSampling2D | output: | (None, 32, 32, 512)
conv2d_13 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_13 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
activation_13 | input: | (None, 32, 32, 256)
Activation | output: | (None, 32, 32, 256)
concatenate_1 | input: | [(None, 32, 32, 256), (None, 32, 32, 256)]
Concatenate | output: | (None, 32, 32, 512)
conv2d_14 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_14 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
activation_14 | input: | (None, 32, 32, 256)
Activation | output: | (None, 32, 32, 256)
conv2d_15 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
batch_normalization_15 | input: | (None, 32, 32, 256)
BatchNormalization | output: | (None, 32, 32, 256)
activation_15 | input: | (None, 32, 32, 256)
Activation | output: | (None, 32, 32, 256)
up_sampling2d_2 | input: | (None, 32, 32, 256)
UpSampling2D | output: | (None, 64, 64, 256)
conv2d_16 | input: | (None, 64, 64, 256)
Conv2D | output: | (None, 64, 64, 128)
batch_normalization_16 | input: | (None, 64, 64, 128)
BatchNormalization | output: | (None, 64, 64, 128)


activation_16 input: (None, 64, 64, 128)
Activation output: (None, 64, 64, 128)
concatenate_2 input: [(None, 64, 64, 128), (None, 64, 64, 128)]
Concatenate output: (None, 64, 64, 256)
conv2d_17 input: (None, 64, 64, 256)
Conv2D output: (None, 64, 64, 128)
batch_normalization_17 input: (None, 64, 64, 128)
BatchNormalization output: (None, 64, 64, 128)
activation_17 input: (None, 64, 64, 128)
Activation output: (None, 64, 64, 128)
conv2d_18 input: (None, 64, 64, 128)
Conv2D output: (None, 64, 64, 128)
batch_normalization_18 input: (None, 64, 64, 128)
BatchNormalization output: (None, 64, 64, 128)
activation_18 input: (None, 64, 64, 128)
Activation output: (None, 64, 64, 128)


up_sampling2d_3 | input: | (None, 64, 64, 128)
UpSampling2D | output: | (None, 128, 128, 128)
conv2d_19 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 64)
batch_normalization_19 | input: | (None, 128, 128, 64)
BatchNormalization | output: | (None, 128, 128, 64)
activation_19 | input: | (None, 128, 128, 64)
Activation | output: | (None, 128, 128, 64)
concatenate_3 | input: | [(None, 128, 128, 64), (None, 128, 128, 64)]
Concatenate | output: | (None, 128, 128, 128)
conv2d_20 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 64)
batch_normalization_20 | input: | (None, 128, 128, 64)
BatchNormalization | output: | (None, 128, 128, 64)
activation_20 | input: | (None, 128, 128, 64)
Activation | output: | (None, 128, 128, 64)
conv2d_21 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 64)
batch_normalization_21 | input: | (None, 128, 128, 64)
BatchNormalization | output: | (None, 128, 128, 64)
activation_21 | input: | (None, 128, 128, 64)
Activation | output: | (None, 128, 128, 64)
conv2d_22 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 2)
conv2d_23 | input: | (None, 128, 128, 2)
Conv2D | output: | (None, 128, 128, 1)

CHAPTER 4

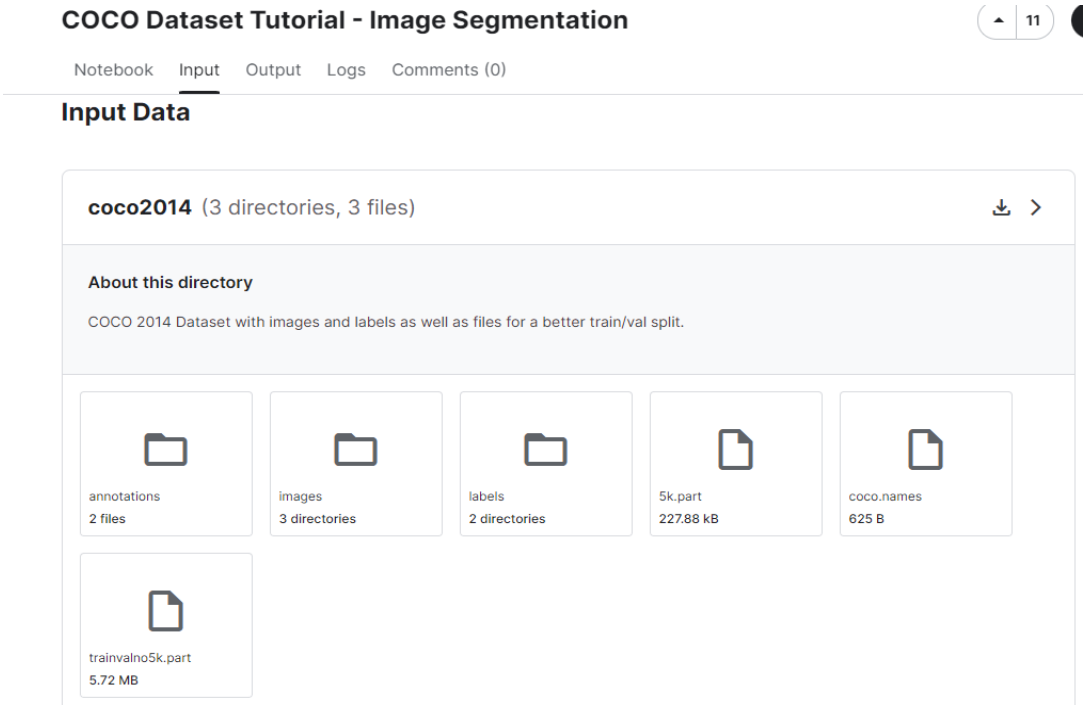
IMPLEMENTATION AND RESULTS

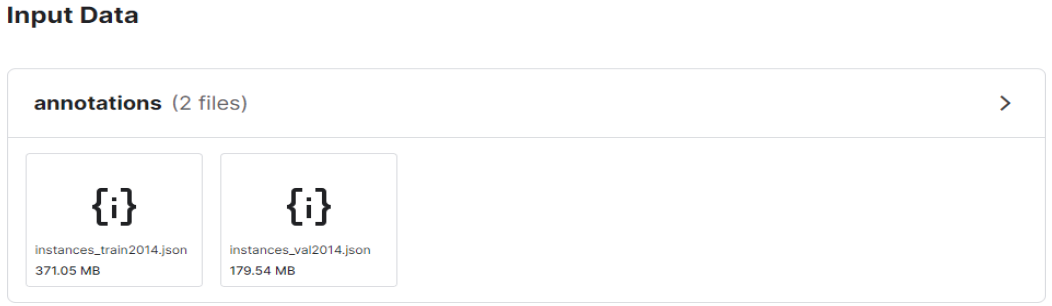
CHAPTER 4

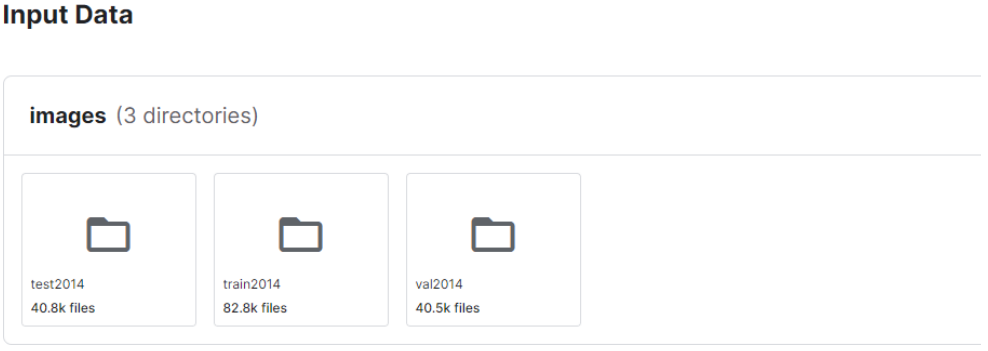
IMPLEMENTATION AND RESULTS

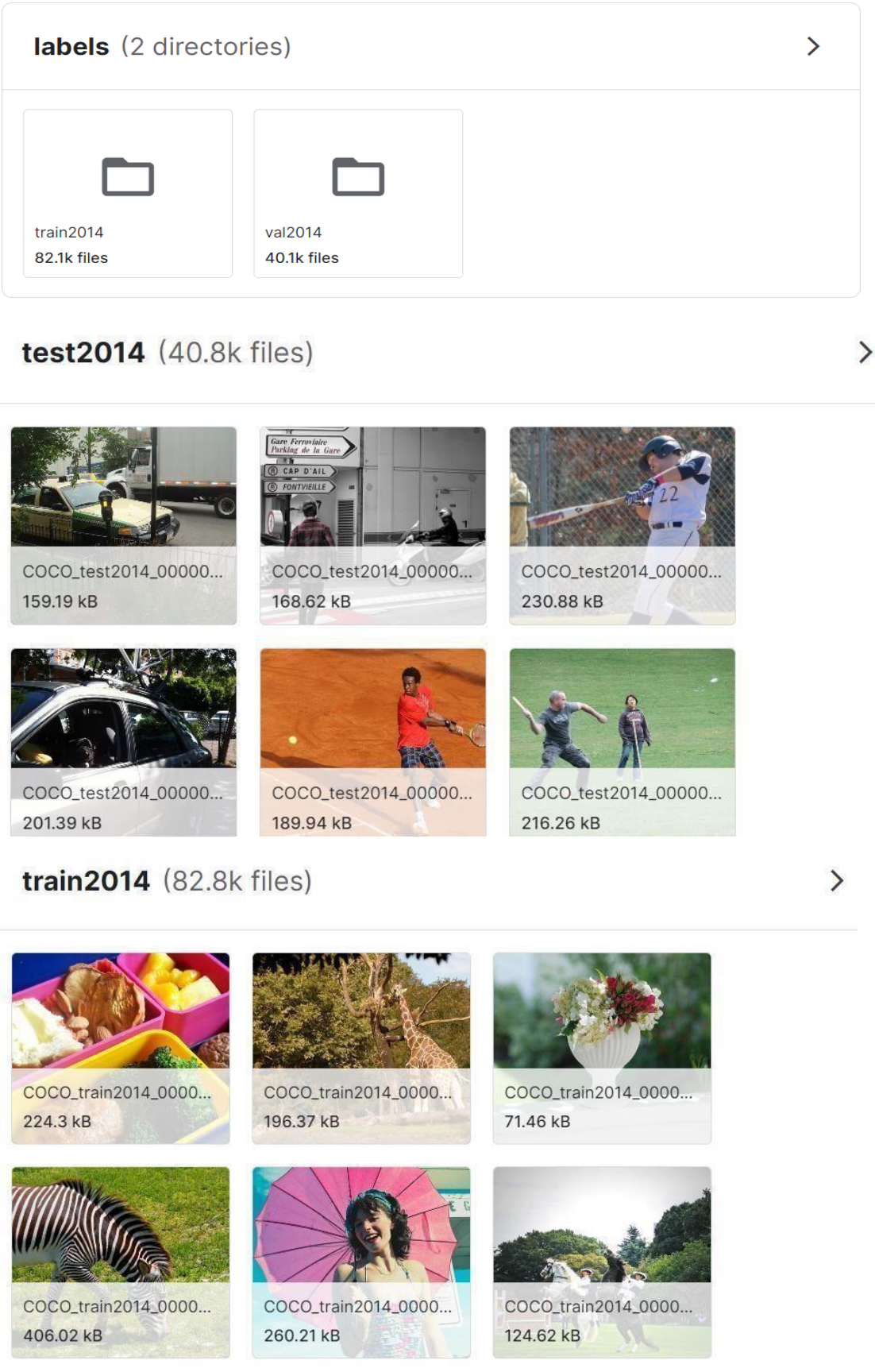
4. IMPLEMENTATION AND RESULTS

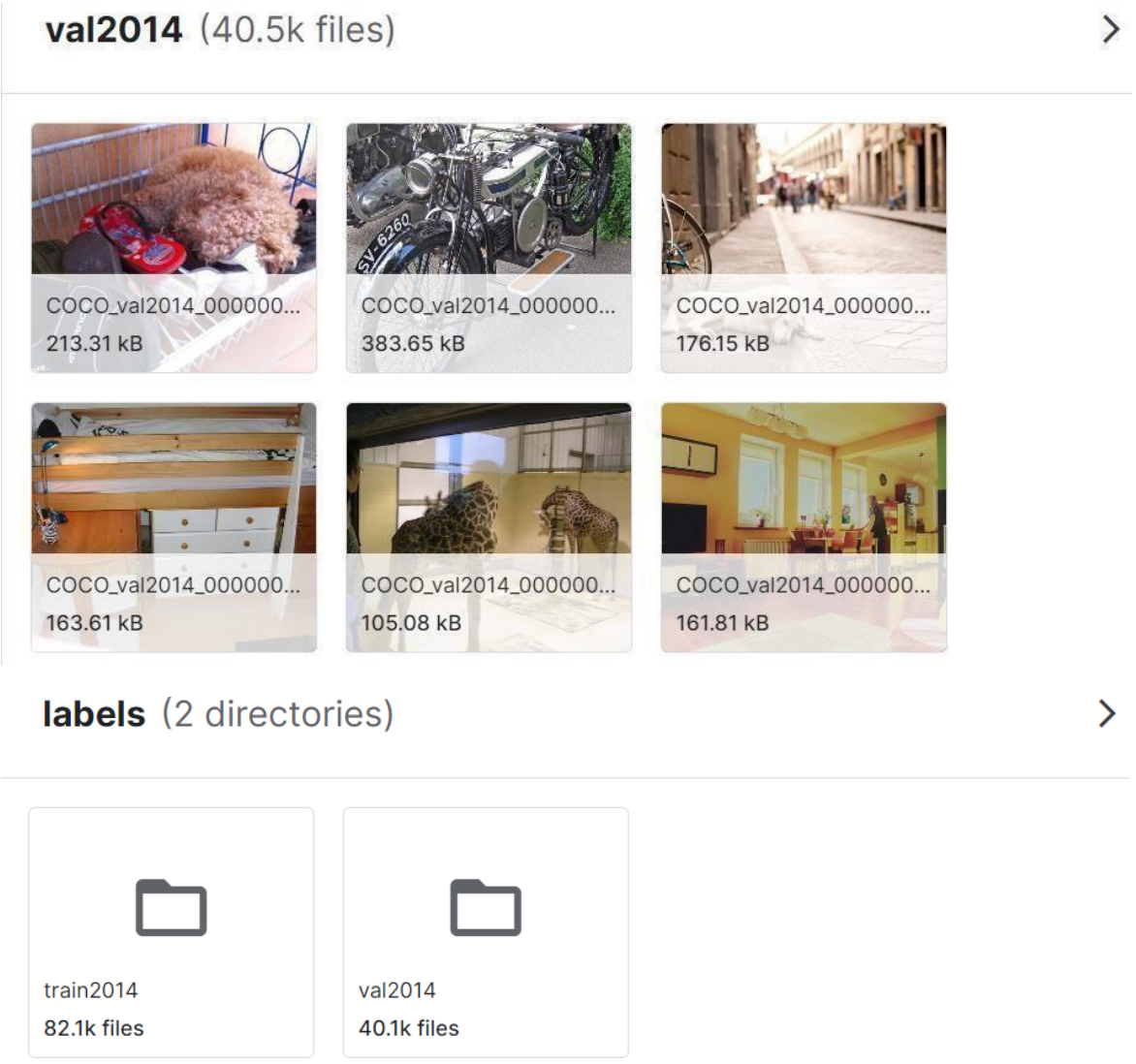
4.1 DATASET








labels (2 directories)
train2014
82.1k files
val2014
40.1k files
test2014 (40.8k files)
COCO_test2014_00000...
159.19 kB
COCO_test2014_00000...
168.62 kB
COCO_test2014_00000...
230.88 kB
COCO_test2014_00000...
201.39 kB
COCO_test2014_00000...
189.94 kB
COCO_test2014_00000...
216.26 kB
train2014 (82.8k files)
COCO_train2014_0000...
224.3 kB
COCO_train2014_0000...
196.37 kB
COCO_train2014_0000...
71.46 kB
COCO_train2014_0000...
406.02 kB
COCO_train2014_0000...
260.21 kB
COCO_train2014_0000...
124.62 kB



4.2 CODING DETAILS

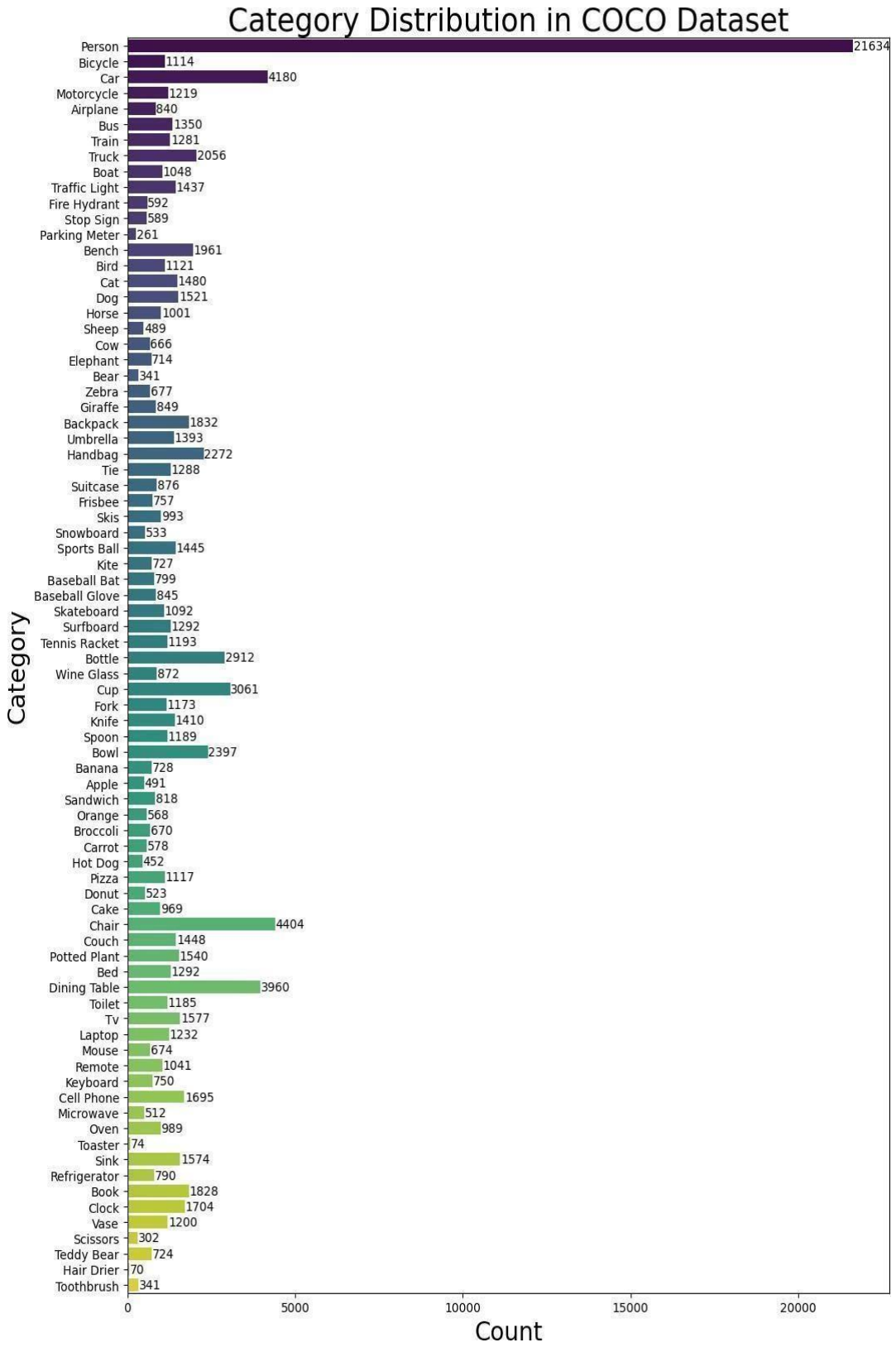
Displaying Image with Annotations

```
print(f"Annotations for Image ID {imgId}:")
anns = coco.loadAnns(ann_ids)

image_path = coco.loadImgs(imgId)[0]['file_name']
print(image_path)
image = plt.imread(imageDir + image_path)
plt.imshow(image)

# Display the specified annotations
coco.showAnns(anns, draw_bbox=True)

plt.axis('off')
plt.title('Annotations for Image ID: {}'.format(image_id))
plt.tight_layout()
plt.show()
```

```
Annotations for Image ID 262148:
COCO_val2014_000000262148.jpg
```

Visualizing Category Distribution in the COCO Dataset

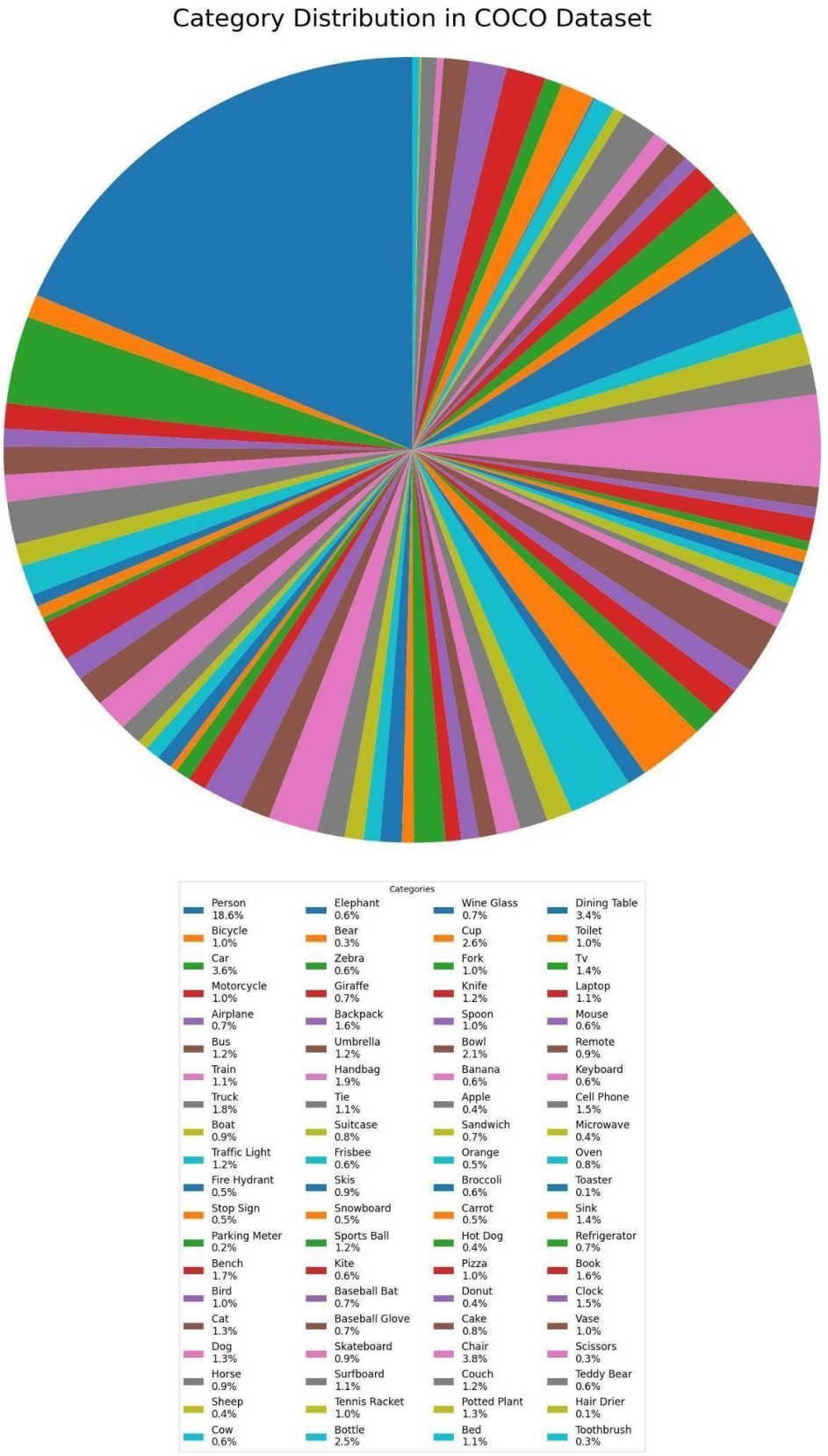
```
# Load the categories in a variable
catIDs = coco.getCatIds()
cats = coco.loadCats(catIDs)

# Get category names
category_names = [cat['name'].title() for cat in cats]

# Get category counts
category_counts = [coco.getImgIds(catIds=[cat['id']]) for cat in cats]
category_counts = [len(img_ids) for img_ids in category_counts]


# Create a color palette for the plot
colors = sns.color_palette('viridis', len(category_names))

# Create a horizontal bar plot to visualize the category counts
plt.figure(figsize=(11, 15))
sns.barplot(x=category_counts, y=category_names, palette=colors)

# Add value labels to the bars
for i, count in enumerate(category_counts):
    plt.text(count + 20, i, str(count), va='center')
plt.xlabel('Count',fontsize=20)
plt.ylabel('Category',fontsize=20)
plt.title('Category Distribution in COCO Dataset',fontsize=25)
plt.tight_layout()
plt.savefig('coco-cats.png',dpi=300)
plt.show()
```


Category Distribution in COCO Dataset
Category
Count
Person 21634
Bicycle 1114
Car 4180
Motorcycle 1219
Airplane 840
Bus 1350
Train 1281
Truck 2056
Boat 1048
Traffic Light 1437
Fire Hydrant 592
Stop Sign 589
Parking Meter 261
Bench 1961
Bird 1121
Cat 1480
Dog 1521
Horse 1001
Sheep 489
Cow 666
Elephant 714
Bear 341
Zebra 677
Giraffe 849
Backpack 1832
Umbrella 1393
Handbag 2272
Tie 1288
Suitcase 876
Frisbee 757
Skis 993
Snowboard 533
Sports Ball 1445
Kite 727
Baseball Bat 799
Baseball Glove 845
Skateboard 1092
Surfboard 1292
Tennis Racket 1193
Bottle 2912
Wine Glass 872
Cup 3061
Fork 1173
Knife 1410
Spoon 1189
Bowl 2397
Banana 728
Apple 491
Sandwich 818
Orange 568
Broccoli 670
Carrot 578
Hot Dog 452
Pizza 1117
Donut 523
Cake 969
Chair 4404
Couch 1448
Potted Plant 1540
Bed 1292
Dining Table 3960
Toilet 1185
Tv 1577
Laptop 1232
Mouse 674
Remote 1041
Keyboard 750
Cell Phone 1695
Microwave 512
Oven 989
Toaster 74
Sink 1574
Refrigerator 790
Book 1828
Clock 1704
Vase 1200
Scissors 302
Teddy Bear 724
Hair Drier 70
Toothbrush 341
0
5000
10000
15000
20000

Visualizing Category Distribution as a Pie Chart

```
# Calculate percentage for each category
total_count = sum(category_counts)
category_percentages = [(count / total_count) * 100 for count in category_counts]


# Create a pie chart to visualize the category distribution
plt.figure(figsize=(15, 24.9))


# Customize labels properties
labels = [f"{name} " for name, percentage in zip(category_names, category_percentages)]
label_props = {"fontsize": 25,
               "bbox": {"edgecolor": "white",
                        "facecolor": "white",
                        "alpha": 0.7,
                        "pad": 0.5}
              }

# Add percentage information to labels, and set labeldistance to remove labels from the pie
wedges, _, autotexts = plt.pie(category_counts,
                               autopct='',
                               startangle=90,
                               textprops=label_props,
                               pctdistance=0.85)

# Create the legend with percentages
legend_labels = [f"{label}\n{category_percentages[i]:.1f}%" for i, label in enumerate(labels)]
plt.legend(wedges, legend_labels, title="Categories", loc="upper center", bbox_to_anchor=(0.5, -0.01),
           ncol=4, fontsize=12)
```

```
plt.axis('equal')
plt.title('Category Distribution in COCO Dataset', fontsize=29)
plt.tight_layout()
plt.savefig('coco-dis.png', dpi=300)
plt.show()
```

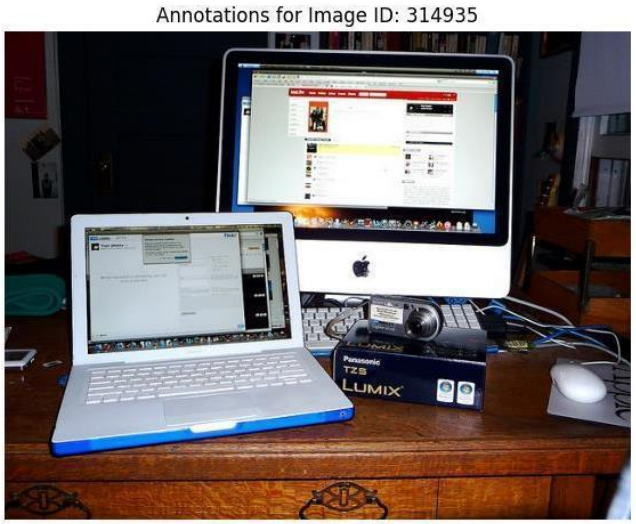

Category Distribution in COCO Dataset
Categories
Person 18.6%
Elephant 0.6%
Wine Glass 0.7%
Dining Table 3.4%
Bicycle 1.0%
Bear 0.3%
Cup 2.6%
Toilet 1.0%
Car 3.6%
Zebra 0.6%
Fork 1.0%
Tv 1.4%
Motorcycle 1.0%
Giraffe 0.7%
Knife 1.2%
Laptop 1.1%
Airplane 0.7%
Backpack 1.6%
Spoon 1.0%
Mouse 0.6%
Bus 1.2%
Umbrella 1.2%
Bowl 2.1%
Remote 0.9%
Train 1.1%
Handbag 1.9%
Banana 0.6%
Keyboard 0.6%
Truck 1.8%
Tie 1.1%
Apple 0.4%
Cell Phone 1.5%
Boat 0.9%
Suitcase 0.8%
Sandwich 0.7%
Microwave 0.4%
Traffic Light 1.2%
Frisbee 0.6%
Orange 0.5%
Oven 0.8%
Fire Hydrant 0.5%
Skis 0.9%
Broccoli 0.6%
Toaster 0.1%
Stop Sign 0.5%
Snowboard 0.5%
Carrot 0.5%
Sink 1.4%
Parking Meter 0.2%
Sports Ball 1.2%
Hot Dog 0.4%
Refrigerator 0.7%
Bench 1.7%
Kite 0.6%
Pizza 1.0%
Book 1.6%
Bird 1.0%
Baseball Bat 0.7%
Donut 0.4%
Clock 1.5%
Cat 1.3%
Baseball Glove 0.7%
Cake 0.8%
Vase 1.0%
Dog 1.3%
Skateboard 0.9%
Chair 3.8%
Scissors 0.3%
Horse 0.9%
Surfboard 1.1%
Couch 1.2%
Teddy Bear 0.6%
Sheep 0.4%
Tennis Racket 1.0%
Potted Plant 1.3%
Hair Drier 0.1%
Cow 0.6%
Bottle 2.5%
Bed 1.1%
Toothbrush 0.3%

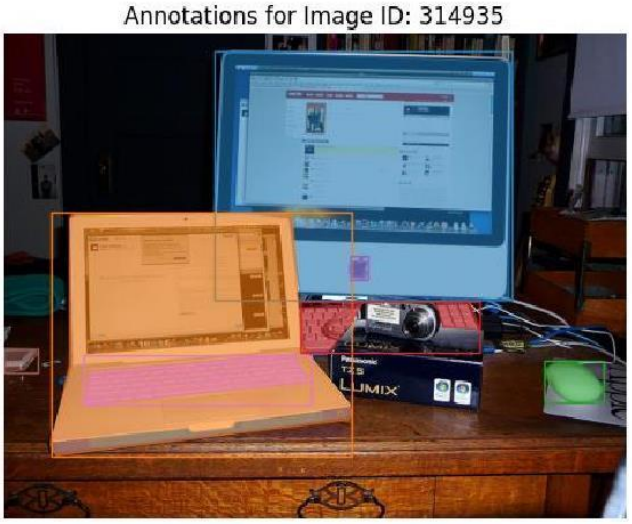
Displaying Filtered Images with Annotations

```
# Define the classes (out of the 80) which you want to see. Others will not be shown.
filterClasses = ['laptop', 'tv', 'cell phone']

# Fetch class IDs only corresponding to the filterClasses
catIds = coco.getCatIds(catNms=filterClasses)

# Get all images containing the above Category IDs
imgIds = coco.getImgIds(catIds=catIds)

# Load a random image from the filtered list
if len(imgIds) > 0:
    image_id = imgIds[np.random.randint(len(imgIds))]  # Select a random image ID
    image_info = coco.loadImgs(image_id)

    if image_info is not None and len(image_info) > 0:
        image_info = image_info[0]
        image_path = imageDir + image_info['file_name']

        # Load the annotations for the image
        annotation_ids = coco.getAnnIds(imgIds=image_id)
        annotations = coco.loadAnns(annotation_ids)

        # Get category names and assign colors for annotations
        category_names = [coco.loadCats(ann['category_id'])[0]['name'].capitalize() for ann in annotations]
        category_colors = list(matplotlib.colors.TABLEAU_COLORS.values())

        # Load the image and plot it
        image = plt.imread(image_path)
        plt.imshow(image)
```

```
        plt.axis('off')
        plt.title('Annotations for Image ID: {}'.format(image_id))
        plt.tight_layout()
        plt.savefig('Img.png',dpi=350)
        plt.show()

        plt.imshow(image)
        plt.axis('off')

        # Display bounding boxes and segmented colors for each annotation
        for ann, color in zip(annotations, category_colors):
            bbox = ann['bbox']
            segmentation = ann['segmentation']

            # Display bounding box
            rect = patches.Rectangle((bbox[0], bbox[1]), bbox[2], bbox[3], linewidth=1,
                                     edgecolor=color, facecolor='none')
            plt.gca().add_patch(rect)

            # Display segmentation masks with assigned colors
            for seg in segmentation:
                poly = np.array(seg).reshape((len(seg) // 2, 2))
                plt.fill(poly[:, 0], poly[:, 1], color=color, alpha=0.6)

        # Create a legend with category names and colors
        legend_patches = [patches.Patch(color=color, label=name) for color, name in zip(category_colors, category_names)]
        plt.legend(handles=legend_patches, loc="lower center", ncol=4, bbox_to_anchor=(0.5, -0.2), fontsize='small')
```

```
        # Show the image with legend
        plt.title('Annotations for Image ID: {}'.format(image_id))
        plt.tight_layout()
        plt.savefig('annImg.png',dpi=350)
        plt.show()
    else:
        print("No image information found for the selected image ID.")
else:
    print("No images found for the desired classes.")
```

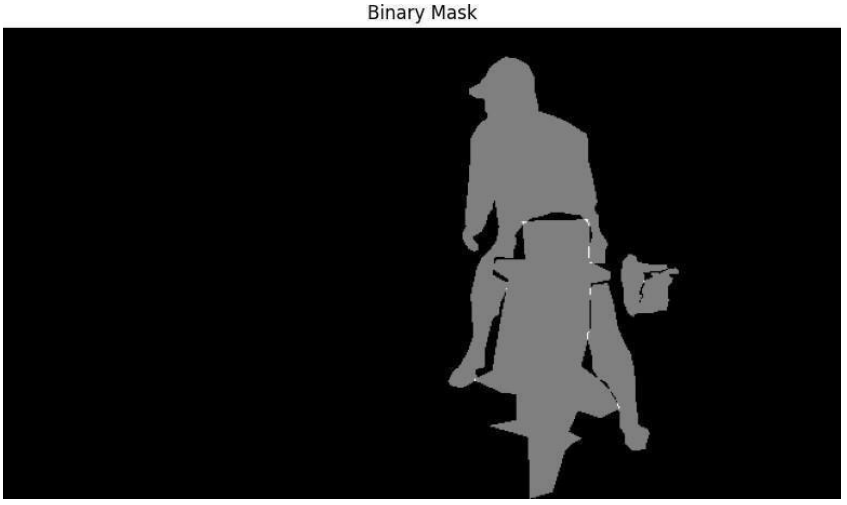




Generating Masks for Object Segmentation

Generating Binary Masks

```
# Retrieve image dimensions
image_info = coco.loadImgs(image_id)[0]
height, width = image_info['height'], image_info['width']

# Create an empty binary mask with the same dimensions as the image
binary_mask = np.zeros((height, width), dtype=np.uint8)

# Iterate through the annotations and draw the binary masks
for annotation in annotations:
    segmentation = annotation['segmentation']
    mask = coco.annToMask(annotation)

    # Add the mask to the binary mask
    binary_mask += mask

# Display the binary mask
plt.figure(figsize=(10,10))
plt.imshow(binary_mask, cmap='gray')
plt.axis('off')
plt.title('Binary Mask')
plt.savefig('binary_mask.png', dpi=300)
plt.show()
```



Generating RGB Mask

```
# Retrieve image dimensions
image_info = coco.loadImgs(image_id)[0]
height, width = image_info['height'], image_info['width']

# Create an empty RGB mask with the same dimensions as the image
rgb_mask = np.zeros((height, width, 3), dtype=np.uint8)

# Define a color map for different object classes
color_map = {cat['id']: (np.random.randint(0, 256), np.random.randint(0, 256), np.random.randint(0, 256))
             for cat in coco.loadCats(catIDs)}

# Iterate through the annotations and assign unique colors to each class/object
for annotation in annotations:
    category_id = annotation['category_id']
    color = color_map[category_id]

    # Draw the mask on the RGB mask
    mask = coco.annToMask(annotation)
    rgb_mask[mask == 1] = color

# Display the RGB mask
plt.figure(figsize=(10,10))
plt.imshow(rgb_mask)
plt.axis('off')
plt.title('RGB Mask')
plt.savefig('rgb_mask.png', dpi=300)
plt.show()
```

RGB Mask

Generating Instance Segmentation Mask

```
# Retrieve image dimensions
image_info = coco.loadImgs(image_id)[0]
height, width = image_info['height'], image_info['width']

# Create an empty mask with the same dimensions as the image
instance_mask = np.zeros((height, width), dtype=np.uint8)

# Iterate through the annotations and draw the instance segmentation masks
for annotation in annotations:
    segmentation = annotation['segmentation']
    mask = coco.annToMask(annotation)
    category_id = annotation['category_id']

    # Assign a unique value to each instance mask
    instance_mask[mask == 1] = category_id

# Display the instance segmentation mask
plt.figure(figsize=(10,10))
plt.imshow(instance_mask, cmap='viridis')
plt.axis('off')
plt.title('Instance Segmentation Mask')
plt.savefig('instance_mask.png', dpi=300)
plt.show()
```

Instance Segmentation Mask

Generating Object Detection Bounding Boxes

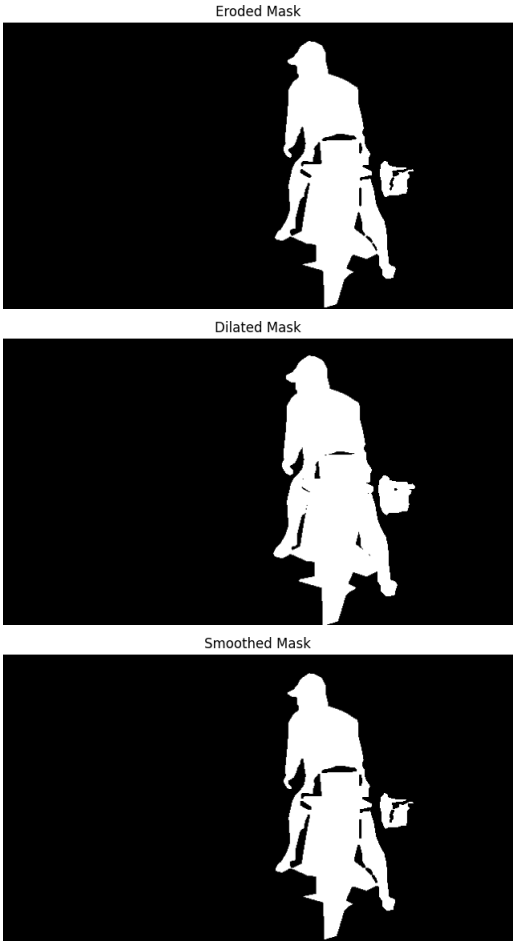
```
# Retrieve image dimensions
image_info = coco.loadImgs(image_id)[0]
height, width = image_info['height'], image_info['width']

# Create a new figure with the same dimensions as the image
fig, ax = plt.subplots(figsize=(10,10), dpi=100)

# Display the original image
ax.imshow(main_image)
ax.axis('off')
ax.set_title('Original Image')

# Draw bounding boxes on the original image
for annotation in annotations:
    bbox = annotation['bbox']
    category_id = annotation['category_id']
    category_name = coco.loadCats(category_id)[0]['name']

    # Convert COCO bounding box format (x, y, width, height) to matplotlib format (xmin, ymin, xmax, ymax)
    xmin, ymin, width, height = bbox
    xmax = xmin + width
    ymax = ymin + height

    # Draw the bounding box rectangle
    rect = patches.Rectangle((xmin, ymin), width, height, linewidth=1, edgecolor='red', facecolor='none')
    ax.add_patch(rect)

    # Add the category name as a label above the bounding box
    ax.text(xmin, ymin - 5, category_name, fontsize=8, color='red', weight='bold')
```

```
# Save the figure with adjusted dimensions
plt.savefig('bounding_boxes.png', bbox_inches='tight')

# Show the plot
plt.show()
```

Post-Processing Techniques

```
import numpy as np
from scipy.ndimage import binary_erosion, binary_dilation
from scipy.ndimage.filters import gaussian_filter

# Apply erosion to the binary mask
eroded_mask = binary_erosion(binary_mask)

# Apply dilation to the binary mask
dilated_mask = binary_dilation(binary_mask)

# Apply Gaussian blur to the binary mask
smoothed_mask = gaussian_filter(binary_mask, sigma=.2)

# Display the post-processed masks
fig, axes = plt.subplots(3, 1, figsize=(12, 12))

axes[0].imshow(eroded_mask, cmap='gray')
axes[0].set_title('Eroded Mask')
axes[0].axis('off')

axes[1].imshow(dilated_mask, cmap='gray')
axes[1].set_title('Dilated Mask')
axes[1].axis('off')

axes[2].imshow(smoothed_mask, cmap='gray')
axes[2].set_title('Smoothed Mask')
axes[2].axis('off')
```

```
plt.tight_layout()
plt.savefig('post_processed_masks.png', dpi=300)
plt.show()
```



Visualization and Qualitative Assessment

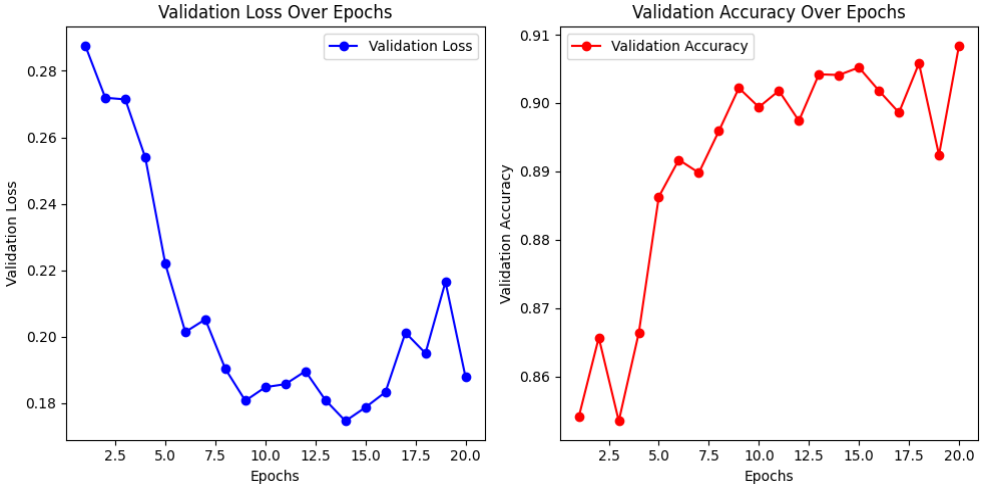
```
# Select an image ID for visualization
image_id = image_ids[0]

# Load the image
image_info = coco.loadImgs(image_id)[0]
image_path = os.path.join(imageDir, image_info['file_name'])
image = plt.imread(image_path)

# Get the ground truth annotations for the image
annotation_ids = coco.getAnnIds(imgIds=image_id)
annotations = coco.loadAnns(annotation_ids)

# Create a blank image for overlaying the masks
overlay = image.copy()

# Iterate over the annotations and draw the masks on the overlay image
for annotation in annotations:
    # Get the segmentation mask
    mask = coco.annToMask(annotation)

    # Choose a random color for the mask
    color = np.random.randint(0, 256, size=(3,), dtype=np.uint8)

    # Apply the mask to the overlay image
    overlay[mask == 1] = color

# Create a figure and subplot for visualization
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
```

```
# Plot the original image
ax1.imshow(image)
ax1.set_title('Original Image')
ax1.axis('off')

# Plot the image with overlay masks
ax2.imshow(overlay)
ax2.set_title('Masks Overlay')
ax2.axis('off')

# Adjust the spacing between subplots
plt.tight_layout()

# Save the visualization as an image file
plt.savefig('mask_visualization.png', dpi=300)

# Show the plot
plt.show()
```

Training a U-Net Model for Image Segmentation

```
import matplotlib.pyplot as plt

# Your data
epochs = range(1, 21)  # Assuming you have 20 epochs based on the provided data

# Loss values
val_loss = [0.2876, 0.2719, 0.2714, 0.2539, 0.2219, 0.2014, 0.2052, 0.1903, 0.1807, 0.1848, 0.1857, 0.1896, 0.1809, 0.1746, 0.1787, 0.1833, 0.2012, 0.1950, 0.2165, 0.1879]

# Accuracy values
val_accuracy = [0.8541, 0.8657, 0.8535, 0.8664, 0.8863, 0.8917, 0.8898, 0.8959, 0.9022, 0.8994, 0.9018, 0.8974, 0.9042, 0.9041, 0.9052, 0.9018, 0.8986, 0.9058, 0.8924, 0.9084]

# Plotting validation loss
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.plot(epochs, val_loss, label='Validation Loss', marker='o', linestyle='-', color='b')
plt.title('Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Validation Loss')
plt.legend()

# Plotting validation accuracy
plt.subplot(1, 2, 2)
plt.plot(epochs, val_accuracy, label='Validation Accuracy', marker='o', linestyle='-', color='r')
plt.title('Validation Accuracy Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Validation Accuracy')
plt.legend()
```

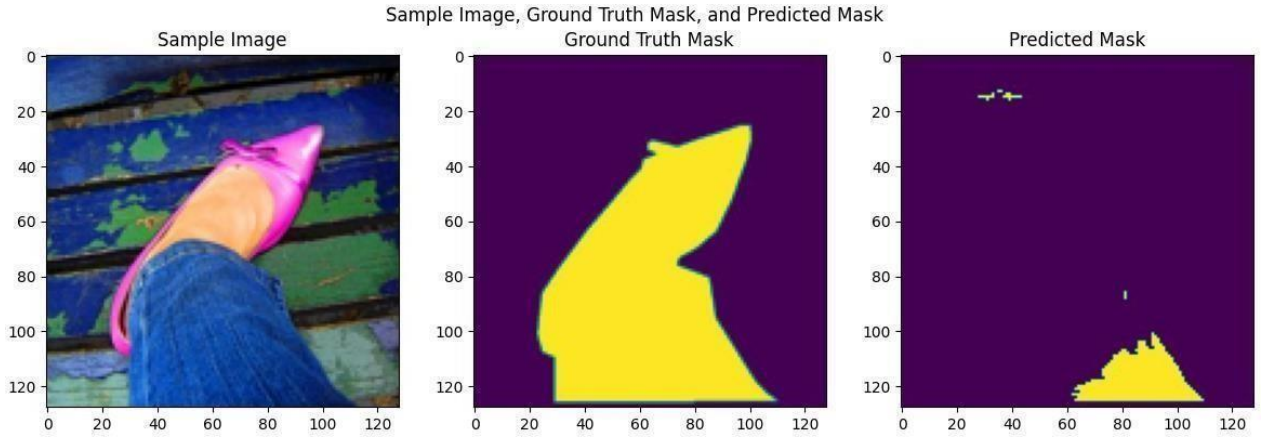
```
# Display the plots
plt.tight_layout()
plt.show()
```



Predicting and Visualizing Segmentation Masks

```
# Get a sample batch from the validation data generator
sample_images, sample_masks = val_generator[0]

# Generate predictions on the sample batch
predictions = model.predict(sample_images)

# Threshold the predictions (if needed)
threshold = 0.5  # Adjust the threshold as per your requirement
thresholded_predictions = (predictions > threshold).astype(np.uint8)

# Select a random index from the batch
idx = np.random.randint(0, sample_images.shape[0])

# Plot the sample image, ground truth mask, and predicted mask
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

# Plot sample image
axes[0].imshow(sample_images[idx])
axes[0].set_title('Sample Image')

# Plot ground truth mask
axes[1].imshow(sample_masks[idx])
axes[1].set_title('Ground Truth Mask')

# Plot predicted mask
axes[2].imshow(thresholded_predictions[idx])
axes[2].set_title('Predicted Mask')
```

```
# Set common title for the figure
fig.suptitle('Sample Image, Ground Truth Mask, and Predicted Mask')

# Adjust the spacing between subplots
plt.tight_layout()

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



CHAPTER 5
CONCLUSION

5. CONCLUSION

In conclusion, this project navigated the intricacies of image segmentation by harnessing the diverse COCO 2014 dataset and implementing a U-Net model. With over 330,000 images across 80 categories, the dataset served as a robust training ground. The U-Net model, developed using TensorFlow and Keras within the Google Colab environment, showcased adept object delineation. The systematic journey through 18 steps covered data setup, exploratory analysis, and model implementation, offering a comprehensive guide for practitioners in computer vision.

Visualizations, including category distribution analyses and mask comparisons, enriched the project's exploration. Beyond technical skills, the endeavor provided practical insights into deploying deep learning for real-world challenges in computer vision. This project not only advances understanding of image segmentation but also contributes valuable insights to the evolving landscape of artificial intelligence, empowering practitioners for future advancements in image analysis and segmentation tasks.

5.1 LIMITATIONS

1. **Dataset Size and Diversity:** While the COCO 2014 dataset is extensive, it may not cover all possible scenarios or domains. Limitations in dataset diversity could affect the model's generalization to specific objects or scenes not well-represented in the dataset.
2. **Computational Resources in Google Colab:** Google Colab provides free GPU and TPU resources, but there are limitations on usage time and available memory. Larger datasets or more complex models may face constraints, impacting the scalability of the project.
3. **Model Complexity and Training Time:** The U-Net model's depth and complexity, while beneficial for intricate segmentation tasks, could lead to longer training times. This may hinder rapid experimentation and iteration during the model development phase.
4. **Hyperparameter Tuning Challenges:** Fine-tuning hyperparameters for optimal performance might be challenging due to constraints in time and resources. Exhaustive hyperparameter searches were not explicitly addressed in the project, potentially limiting model optimization.
5. **Deployment Considerations:** The project focuses on model development and training but lacks a detailed exploration of deployment considerations. Real-world applications may require additional steps, such as model conversion, optimization, and integration into production systems, which are not fully covered.

5.2 FUTURE SCOPE

The project lays a robust foundation for several future enhancements and applications in the realm of image segmentation and deep learning:

1. Semantic Segmentation Improvements
2. Transfer Learning and Fine-Tuning
3. Real-Time Segmentation Applications
4. Ensemble Methods
5. Interactive User Interfaces
6. Integration with Web Technologies
7. Multi-Modal Segmentation
8. Adversarial Robustness and Security

CHAPTER 6
REFERENCES

6. REFERENCES

- https://arxiv.org/pdf/1405.0312.pdf
- https://link.springer.com/chapter/10.1007/978-3-319-10602-1_48
- https://encord.com/blog/image-segmentation-for-computer-vision-best-practice-guide/
- https://www.geeksforgeeks.org/introduction-to-tensorflow/
- https://www.geeksforgeeks.org/u-net-architecture-explained/