IMAGE SEGMENTATION OF DATASET-2017(COCO)

## Student name: Jaya Navya Earni Student Id:22084778

Contents

1. [Introduction and Literature Review 3](#_bookmark0)
2. [Exploratory Data Analysis (EDA) 3](#_bookmark1)
3. [Methodology 6](#_bookmark2)
4. [Results and Discussion 7](#_bookmark3)
5. Conclusion ….9

[References 10](#_bookmark4)

[Appendix 11](#_bookmark5)

# Introduction and Literature Review

### Each pixel in an image is labeled by semantic segmentation, and it is crucial in such areas as

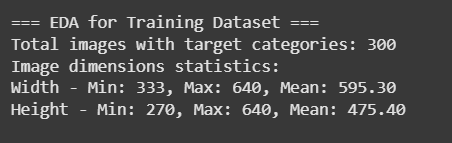
### medical imaging and autonomous driving. The conventional approaches were based on features which are crafted by hand and were limited in generalization. Segmentation is significantly

### enhanced using hierarchical features and multi-scale context implemented with deep learning and CNNs mostly and models like U-Net, DeepLabV3+, and Mask R-CNN are leading in this segmentation (Sarker, 2021).

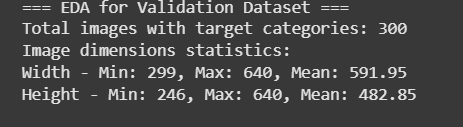
### The COCO dataset has good quality annotations and various images to use in segmentation tasks but its size and complexity may prove to be difficult (Liu et al., 2020). To cope with this, a lightweight model was trained on four classes taken: cake, car, dog and person. The work was concentrated on data preprocessing, model design and the evaluation to have an efficient and accurate solution that will be computationally economical. U-Net was adopted due to the efficiency and skip connections to maintain the spatial information (Dasgupta et al., 2020).

# Exploratory Data Analysis (EDA)

### The data consists of 600 images, whose ratio in the training and validation sets is equal and corresponds to 300 images each, in COCO-style format, with one.json file describing the position of objects and their semantics (Seo et al., 2019).The dataset was filtered during the initial exploratory data analysis to discard annotations of every class except four of them (while keeping noise and limiting the model to relevant information) (Seo et al., 2019).

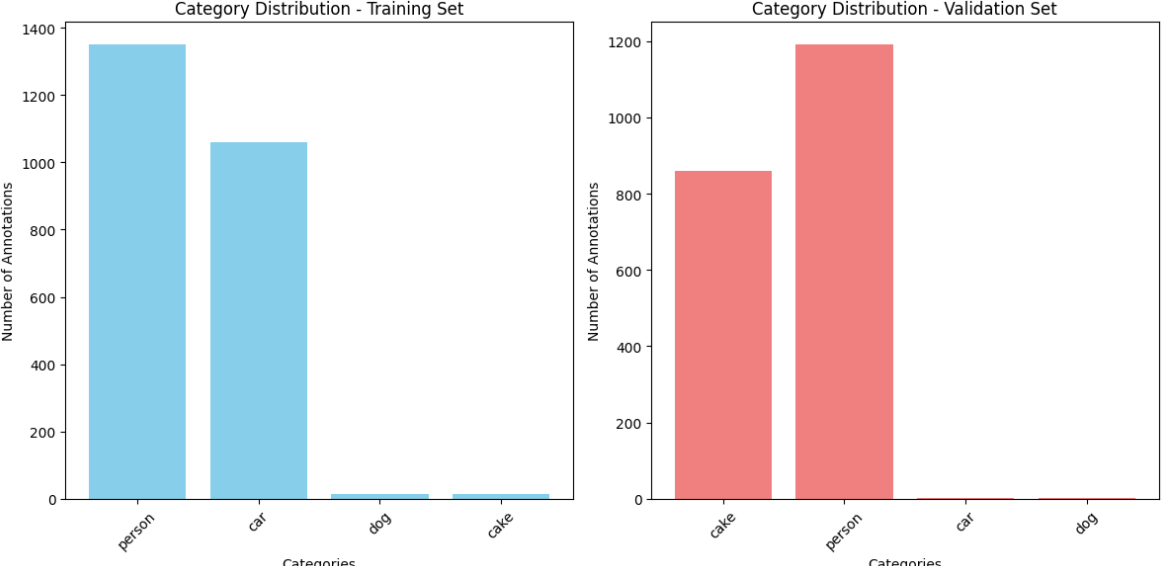


**Figure 1: Statistical summary of images in Training dataset**

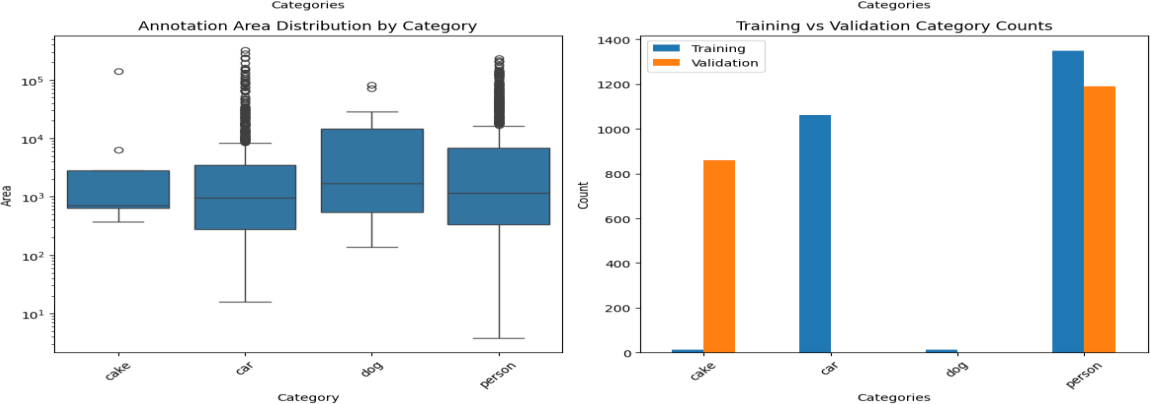


***Figure 2: Statistical summary of images in Validation dataset***

### Class distribution breakdown caused an object frequency imbalance wherein, person was the most frequent class, whereas cake and car were less frequent. This imbalance may lead to biasedness of the model in favor of the overrepresented classes and was taken into an adequate account during the assessment of the model and interpretation of findings.

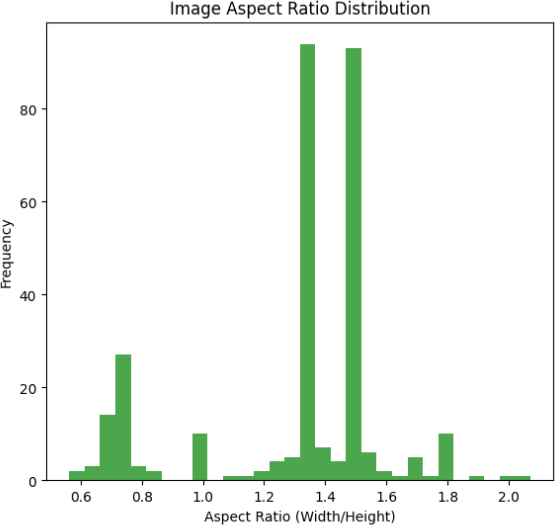


***Figure 3: Distribution of categories in training and validation datasets***

****

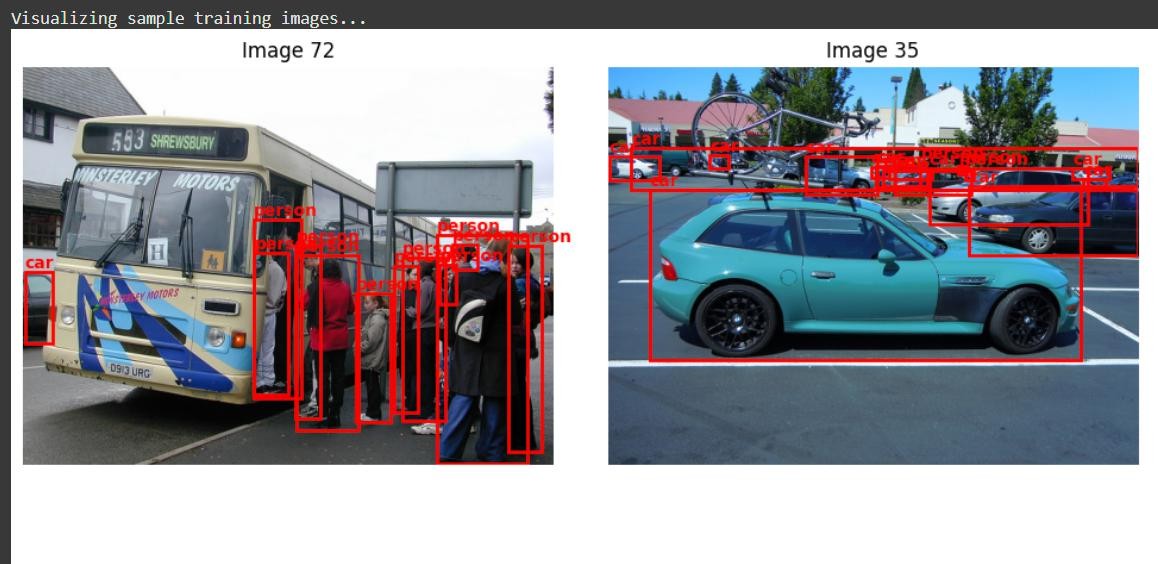
***Figure 4: Area distribution by category***

### A deeper inspection of the image size revealed that the majority of images were comparatively evenly sized, with average resolutions of 595×475 pixels for training and 591×482 for validation. This evenness provided a basis for uniform preprocessing, like resizing with little distortion. The aspect ratios, which were predominantly between 1.2 and 1.6, upheld the visual consistency within the dataset (Kreshuk, et al., 2019).

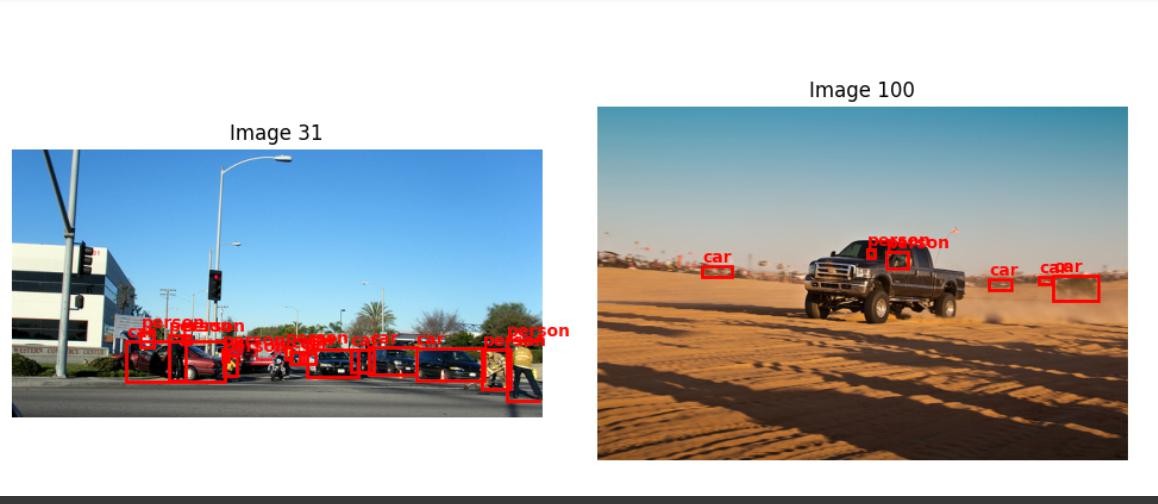


***Figure 5: Image aspect ratio distribution***

### Annotation regions were diverse in size, with the bigger and higher-level objects like 'dog' and 'person' covering much of the image, while 'cake' annotations were smaller and localized. This led to the speculation that the model would have more difficulty detecting and segmenting tiny objects because there was less pixel-level context. The average annotation count per image was moderate, with some images having a maximum of seven instances of objects, so the model would have to deal with sparse and moderately dense scenes.



***Figure 6: Sample training images segmentation***



***Figure 7: Sample training images segmentation***

### Visual checking of annotations from sample plots attested that the segmentation masks and labels were properly aligned to the objects on the images.

# Methodology

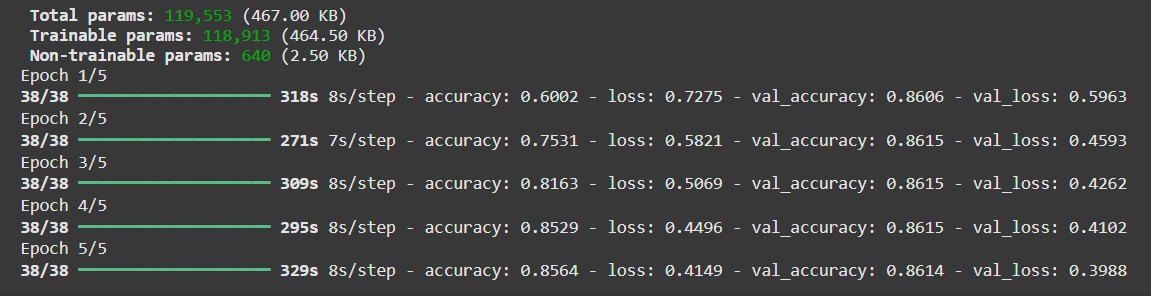
### *Data preprocessing:* joint decoding of the annotations in the COCO category, selection of images with at least one instance of a target object, resizing and normalization of images, encoding pycocotools masks to one-hot encoded tensors toward categorizing pixels, all of it mentioned in the initial passage.

### *Model architecture:* trained in TensorFlow/Keras, an encoder-decoder consisting of roughly 119,000 parameters, where convolution was used in the down sampling and feature extraction of the encoder and up sampling in the decoder that reconstructs masks, and where skip connections are used to maintain the spatial detail and semantic segmentation particularly of small objects as mentioned in the second passage.

# Results and Discussion

### It was possible to obtain the accuracy of training of 85.6 per cent within five epochs and the accuracy of validation of 86.1 per cent, which means that there was a good generalization with a small overfitting (Kreshuk et al., 2021). The validation loss decreased steadily 0.5963 to 0.3988, which was a constant training (Liu et al., 2020). Frequent classes such as cars and persons were divided in case of the frequent classes in the sets of test-30. Some less frequent classes (cake, dogs) were identified and occasionally with blurred or moved boundaries. Partial or indistinct

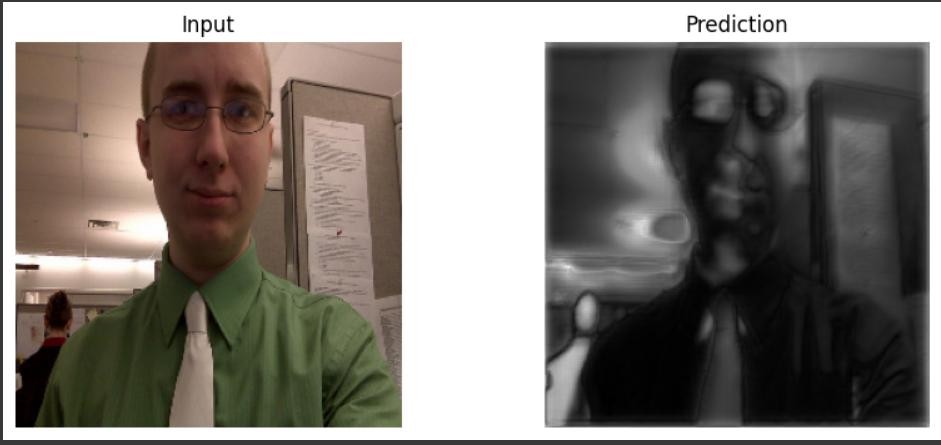
### boundaries were exhibited in case of small or overlapping objects.



***Figure 8: Model performance on training and validation datasets with loss function***

### The model was good at capturing the general shapes but it had a difficulty in the small or less frequent objects details- as it was seen with EDA in the scale and class imbalance. The accuracy of underrepresented classes was lower implying that there should be weighting of classes or data augmentation. Although U-Net had an accuracy of ~86 percent, its small resolution recovery

### reduced precision. The results may be enhanced by future work which incorporates U-Net ++ or pre-trained encoders when doing fine-grained segmentation.



***Figure 9: Prediction on test image 1***

****

***Figure 10: Prediction on test image 2***

****

***Figure 11: Prediction on test image 3***

***Conclusion:***

In this project, segmentation in binary images concerning four chosen classes of COCO 2017 (person, car, dog, and cake) was used. The ~86% struck a lightweight encoder-decoder model. Students showed better performance on popular classes and performed lesser on unpopular classes in a class imbalanced style. The reliability of the model was confirmed with the help of EDA and the visual checks. Future enhancements may include transfer learning, balanced sampling of data, as well as further improved architecture.

# References

Dasgupta, A., Manuel, M., Mansur, R.S., Nowak, N. and Gračanin, D., 2020, March. Machined learning approach towards real time object recognition a context based mixed reality. Accorded to IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW), 2020; abstracts on page 262 and articles on page 268 (pp. 262-268). IEEE. <https://ieeexplore.ieee.org/document/9123220/>

Haamied, R.D., Al-Abudi, B.Q. and Hassan, R.N., 2020. Applying machine learning algorithms to classification coco dataset. Turkish Journal of Physiotherapy and Rehabilitation, 32, p.3. <https://www.researchgate.net/publication/358229248>

Kreshuk, A., 2019; Kreshuk, A. and Zhang, C., 2019. By ilastik: advanced image segmentation by means of machine learning. Computer Optimized Microscopy, Methods and Protocols, pp.449-463. <https://www.researchgate.net/publication/335281559_Machine_Learning_Advanced_Image_Segmentation_Using_ilastik>

Liu, L., Ouyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X. and Pietikainen, M., 2020. Generic object detection using deep learning: A survey. International journal of computer vision, 128, 261-318. <https://link.springer.com/article/10.1007/s11263-019-01247-4>

Sarker, I. H., 2021. Machine learning: Algorithms, practical applications and research lines. SN computer science 2(3) p.160. <https://www.scirp.org/reference/referencespapers?referenceid=3574576>

### Seo, H., Badiei Khuzani, M., Vasudevan, V., Huang, C., Ren, H., Xiao, R., Jia, X. and Xing, L., 2020. Overview of technical aspects of machine learning based biomedical image segmentation: introduction to state-of-art applications. Medical physics, 47(5), p.e148-167.<https://ijisae.org/index.php/IJISAE/article/view/5202>

# Appendix

## Python Code

# Install required packages

!pip install pycocotools albumentations --quiet

# Import necessary libraries

import os, cv2, json, random

import numpy as np

import matplotlib.pyplot as plt

from pycocotools.coco import COCO

from PIL import Image

from tqdm import tqdm  
  
# Google Drive mount

from google.colab import drive

drive.mount("/content/drive")

# Dataset paths

root\_dir = "/content/drive/MyDrive/coco2017"

train\_path = os.path.join(root\_dir, "train-300")

val\_path = os.path.join(root\_dir, "validation-300")

test\_path = os.path.join(root\_dir, "test-30")

train\_images = os.path.join(train\_path, "data")

val\_images = os.path.join(val\_path, "data")

train\_ann = os.path.join(train\_path, "labels.json")

val\_ann = os.path.join(val\_path, "labels.json")

# Loading COCO

coco\_train = COCO(train\_ann)

coco\_val = COCO(val\_ann)

# Define target object categories

target\_names = ['cake', 'car', 'dog', 'person']

target\_ids = coco\_train.getCatIds(catNms=target\_names)

# Geting category IDs for our target categories

def get\_category\_ids(coco, target\_categories):

    cats = coco.loadCats(coco.getCatIds())

    category\_dict = {cat['name']: cat['id'] for cat in cats}

    target\_cat\_ids = []

    for cat in target\_categories:

        if cat in category\_dict:

            target\_cat\_ids.append(category\_dict[cat])

        else:

            print(f"Warning: Category '{cat}' not found in COCO categories")

    return target\_cat\_ids, category\_dict

train\_cat\_ids, train\_cat\_dict = get\_category\_ids(coco\_train, target\_names)

val\_cat\_ids, val\_cat\_dict = get\_category\_ids(coco\_val, target\_names)

print(f"Target category IDs: {train\_cat\_ids}")

print(f"Target category IDs: {val\_cat\_ids}")

def perform\_eda(coco, cat\_ids, dataset\_name):

    print(f"\n=== EDA for {dataset\_name} Dataset ===")

    # Get images containing our target categories

    img\_ids = []

    for cat\_id in cat\_ids:

        img\_ids.extend(coco.getImgIds(catIds=cat\_id))

    # Remove duplicates

    img\_ids = list(set(img\_ids))

    print(f"Total images with target categories: {len(img\_ids)}")

    # Load image information

    imgs = coco.loadImgs(img\_ids)

    # Analyze image dimensions

    widths = [img['width'] for img in imgs]

    heights = [img['height'] for img in imgs]

    print(f"Image dimensions statistics:")

    print(f"Width - Min: {min(widths)}, Max: {max(widths)}, Mean: {np.mean(widths):.2f}")

    print(f"Height - Min: {min(heights)}, Max: {max(heights)}, Mean: {np.mean(heights):.2f}")

    # Analyze annotations per category

    category\_counts = defaultdict(int)

    annotation\_areas = defaultdict(list)

    for img\_id in img\_ids:

      # Get annotation IDs for current image and categories

        ann\_ids = coco.getAnnIds(imgIds=img\_id, catIds=cat\_ids)

        anns = coco.loadAnns(ann\_ids)

# Count annotations and record areas by category

        for ann in anns:

            cat\_name = [k for k, v in train\_cat\_dict.items() if v == ann['category\_id']][0]

            category\_counts[cat\_name] += 1

            annotation\_areas[cat\_name].append(ann['area'])

    print(f"\nAnnotation counts per category:")

    for cat, count in category\_counts.items():

        print(f"{cat}: {count}")

    return img\_ids, imgs, category\_counts, annotation\_areas, widths, heights

# Performing EDA on training and validation sets

train\_img\_ids, train\_imgs, train\_cat\_counts, train\_areas, train\_widths, train\_heights = perform\_eda(coco\_train, train\_cat\_ids, "Training")

val\_img\_ids, val\_imgs, val\_cat\_counts, val\_areas, val\_widths, val\_heights = perform\_eda(coco\_val, train\_cat\_ids, "Validation")

#Import required libraries

import pandas as pd

import seaborn as sns

#  visualizations

fig, axes = plt.subplots(2, 3, figsize=(18, 12))

#  Category distribution - Training

axes[0, 0].bar(train\_cat\_counts.keys(), train\_cat\_counts.values(), color='skyblue')

axes[0, 0].set\_title('Category Distribution - Training Set')

axes[0, 0].set\_xlabel('Categories')

axes[0, 0].set\_ylabel('Number of Annotations')

axes[0, 0].tick\_params(axis='x', rotation=45)

#  Category distribution - Validation

axes[0, 1].bar(val\_cat\_counts.keys(), val\_cat\_counts.values(), color='lightcoral')

axes[0, 1].set\_title('Category Distribution - Validation Set')

axes[0, 1].set\_xlabel('Categories')

axes[0, 1].set\_ylabel('Number of Annotations')

axes[0, 1].tick\_params(axis='x', rotation=45)

#  Image dimensions distribution

axes[0, 2].scatter(train\_widths, train\_heights, alpha=0.5, s=10)

axes[0, 2].set\_title('Image Dimensions Distribution - Training')

axes[0, 2].set\_xlabel('Width')

axes[0, 2].set\_ylabel('Height')

#  Annotation area distribution

all\_areas = []

all\_categories = []

for cat in target\_names:

    if cat in train\_areas:

        all\_areas.extend(train\_areas[cat])

        all\_categories.extend([cat] \* len(train\_areas[cat]))

area\_df = pd.DataFrame({'Category': all\_categories, 'Area': all\_areas})

# Boxplot showing distribution of annotation areas by category (log scale for better visualization)

sns.boxplot(data=area\_df, x='Category', y='Area', ax=axes[1, 0])

axes[1, 0].set\_title('Annotation Area Distribution by Category')

axes[1, 0].set\_yscale('log')

axes[1, 0].tick\_params(axis='x', rotation=45)

# Combined category counts comparison between training and validation sets

combined\_counts = pd.DataFrame({

    'Training': [train\_cat\_counts.get(cat, 0) for cat in target\_names],

    'Validation': [val\_cat\_counts.get(cat, 0) for cat in target\_names]

}, index=target\_names)

# Bar plot comparing category counts in train vs val datasets

combined\_counts.plot(kind='bar', ax=axes[1, 1])

axes[1, 1].set\_title('Training vs Validation Category Counts')

axes[1, 1].set\_xlabel('Categories')

axes[1, 1].set\_ylabel('Count')

axes[1, 1].tick\_params(axis='x', rotation=45)

axes[1, 1].legend()

#  Image aspect ratio distribution

aspect\_ratios = [w/h for w, h in zip(train\_widths, train\_heights)]

axes[1, 2].hist(aspect\_ratios, bins=30, alpha=0.7, color='green')

axes[1, 2].set\_title('Image Aspect Ratio Distribution')

axes[1, 2].set\_xlabel('Aspect Ratio (Width/Height)')

axes[1, 2].set\_ylabel('Frequency')

plt.tight\_layout()

plt.show()

# Randomly sample image IDs from the first 100 images (or fewer if not enough)

def visualize\_samples(coco, img\_ids, images\_path, num\_samples=6):

    """Visualize sample images with annotations"""

    sample\_ids = random.sample(img\_ids[:100], min(num\_samples, len(img\_ids)))

    fig, axes = plt.subplots(2, 3, figsize=(15, 10))

    axes = axes.flatten()

    for i, img\_id in enumerate(sample\_ids):

        # Load image metadata from COCO

        img\_info = coco.loadImgs(img\_id)[0]

        img\_path = os.path.join(images\_path, img\_info['file\_name'])

       # Skip if image file does not exist

       if not os.path.exists(img\_path):

            continue

        # Read and convert image from BGR to RGB for correct color display

        img = cv2.imread(img\_path)

        img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

        # Get annotation IDs for current image and target categories

        ann\_ids = coco.getAnnIds(imgIds=img\_id, catIds=train\_cat\_ids)

        anns = coco.loadAnns(ann\_ids)

        # Show the image on the subplot

        axes[i].imshow(img)

        # Draw bounding boxes and category names for each annotation

        for ann in anns:

            # Retrieve category name from category dictionary

            cat\_name = [k for k, v in train\_cat\_dict.items() if v == ann['category\_id']][0]

            # Extract bounding box coordinates (x, y, width, height)

            bbox = ann['bbox']

            x, y, w, h = bbox

            # Draw a red rectangle for the bounding box

            rect = plt.Rectangle((x, y), w, h, linewidth=2, edgecolor='red', facecolor='none')

            axes[i].add\_patch(rect)

            # Put category name text above the bounding box

            axes[i].text(x, y-5, cat\_name, color='red', fontsize=10, weight='bold')

            # Set title with image ID and remove axis ticks

            axes[i].set\_title(f'Image {img\_id}')c

            axes[i].axis('off')

    plt.tight\_layout()

    plt.show()

print("Visualizing sample training images...")

visualize\_samples(coco\_train, train\_img\_ids, train\_images)

# Gathering training image IDs

def get\_image\_ids(coco\_obj, cat\_ids):

    img\_ids = []

    for cid in cat\_ids:

        img\_ids += coco\_obj.getImgIds(catIds=cid)

    return list(set(img\_ids))

train\_ids = get\_image\_ids(coco\_train, target\_ids)

val\_ids = get\_image\_ids(coco\_val, target\_ids)

# extract image + mask

def get\_img\_mask(coco\_obj, image\_id, img\_dir):

    info = coco\_obj.loadImgs(image\_id)[0]

    path = os.path.join(img\_dir, info['file\_name'])

    image = cv2.imread(path)

    image = cv2.resize(image, (256, 256))

    ann\_ids = coco\_obj.getAnnIds(imgIds=image\_id, catIds=target\_ids)

    anns = coco\_obj.loadAnns(ann\_ids)

    mask = np.zeros((info['height'], info['width']), dtype=np.uint8)

    for ann in anns:

        mask = np.maximum(mask, coco\_obj.annToMask(ann))

    mask = cv2.resize(mask, (256, 256))

    return image, mask

# Load data (train and val)

def prepare\_dataset(ids, coco\_obj, img\_dir):

    X, Y = [], []

    for img\_id in tqdm(ids):

        img, msk = get\_img\_mask(coco\_obj, img\_dir=img\_dir, image\_id=img\_id)

        X.append(img)

        Y.append(msk)

    X = np.array(X) / 255.0

    Y = np.expand\_dims(np.array(Y), -1)

    return X, Y

# Prepare training and validation datasets (first 300 images each)

X\_train, Y\_train = prepare\_dataset(train\_ids[:300], coco\_train, train\_images)

X\_val, Y\_val = prepare\_dataset(val\_ids[:300], coco\_val, val\_images)

# U-Net

from tensorflow.keras import layers, models

def build\_unet(input\_shape=(256, 256, 3)):

    inputs = layers.Input(input\_shape)

    def conv\_block(x, filters):

        x = layers.Conv2D(filters, 3, activation='relu', padding='same')(x)

        x = layers.BatchNormalization()(x)

        x = layers.Conv2D(filters, 3, activation='relu', padding='same')(x)

        x = layers.BatchNormalization()(x)

        return x

  # Encoder path

    s1, p1 = encoder\_block(inputs, 16)  # First encoder block

    s2, p2 = encoder\_block(p1, 32)      # Second encoder block

    # Bottleneck

    b = conv\_block(p2, 64)

    # Decoder path

    d1 = decoder\_block(b, s2, 32)  # First decoder block with skip connection from s2

    d2 = decoder\_block(d1, s1, 16) # Second decoder block with skip connection from s1

    # Output layer with sigmoid activation for binary segmentation mask

    outputs = layers.Conv2D(1, 1, activation='sigmoid')(d2)

    model = models.Model(inputs, outputs)

    return model

# Build the U-Net model for segmentation

model = build\_unet()

# Compile the model with Adam optimizer and binary cross-entropy loss

# 'accuracy' metric is used to monitor training progress

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Print a summary of the model architecture

model.summary()

# Train the model on the training data (X\_train, Y\_train)

# Validate performance on the validation set (X\_val, Y\_val)

# Training runs for 5 epochs with batch size of 8

history = model.fit(X\_train, Y\_train, validation\_data=(X\_val, Y\_val), epochs=5, batch\_size=8)

# Visualise prediction

def show\_prediction(img, mask\_pred):

    plt.figure(figsize=(10, 4))

    plt.subplot(1, 2, 1)

    plt.imshow(img)

    plt.title("Input")

    plt.axis('off')

    plt.subplot(1, 2, 2)

    plt.imshow(mask\_pred.squeeze(), cmap='gray')

    plt.title("Prediction")

    plt.axis('off')

    plt.show()

# Select a random index from the validation dataset

idx = random.randint(0, len(X\_val)-1)

# Predict the mask for the selected validation image

pred = model.predict(np.expand\_dims(X\_val[idx], axis=0))

# Visualize the input image and its predicted mask

show\_prediction(X\_val[idx], pred[0])

# List all test image filenames in the test directory

test\_imgs = os.listdir(test\_path)

# Loop through the first 20 test images

for file in test\_imgs[:20]:

    img\_path = os.path.join(test\_path, file)  # Full path to the image

    img = cv2.imread(img\_path)  # Read the image (BGR format)

    img\_resized = cv2.resize(img, (256, 256))  # Resize image to model input size

    input\_img = img\_resized / 255.0  # Normalize pixel values to [0, 1]

    # Predict the segmentation mask for the image

    pred\_mask = model.predict(np.expand\_dims(input\_img, 0))[0]

    # Visualize the resized input image (converted to RGB) and predicted mask side by side

    show\_prediction(cv2.cvtColor(img\_resized, cv2.COLOR\_BGR2RGB), pred\_mask)