Libraries

In [1]:

```
# Install pycocotools
#!pip install pycocotools

import os
import json
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, losses
from tensorflow.keras.optimizers import Adam
from pycocotools.coco import COCO
from sklearn.model_selection import train_test_split
import cv2
import matplotlib.pyplot as plt
from google.colab import drive
from PIL import Image
import tensorflow as tf
from tensorflow.keras.metrics import BinaryAccuracy, MeanSquaredError
import random
from tensorflow.keras.metrics import Metric

from tensorflow.keras import Model
from tensorflow.keras.layers import (
    Input, Conv2D, BatchNormalization, LeakyReLU, MaxPooling2D, Flatten, Dense, GlobalAv
eragePooling2D
)

from tensorflow.keras import mixed_precision
mixed_precision.set_global_policy('mixed_float16') # all operations during training will
be done with this precision (matrix mul...)
# speeds up training time and interference time
```

In [24]:

```
names_of_class= ["FLY", "NOPALM", "PAUSE", "UNDEFINED"]

def coco_to_center(coco_bbox):
    """
    Convert COCO format bbox [x, y, width, height] to center-based format [center_x, cent
er_y, width, height].

    Args:
        coco_bbox (list or tuple): Bounding box in COCO format [x, y, width, height].

    Returns:
        list: Bounding box in center-based format [center_x, center_y, width, height].
    """
    x, y, width, height = coco_bbox
    center_x = x + width / 2
    center_y = y + height / 2
    return [center_x, center_y, width, height]

def center_to_coco(center_bbox):
    """
    Convert center-based format bbox [center_x, center_y, width, height] to COCO format [
x, y, width, height].

    Args:
        center_bbox (list or tuple): Bounding box in center-based format [center_x, cente
r_y, width, height].

    Returns:
        list: Bounding box in COCO format [x, y, width, height].
    """
    center_x, center_y, width, height = center_bbox
```

```
    x = center_x - width / 2
    y = center_y - height / 2
    return [x, y, width, height]

def plot_image_with_bbox(image, true_bbox, true_class, pred_bbox, pred_class, bbox_format
="coco"):

    plt.figure(figsize=(6, 6))

    # Multiply by 255 to rescale the image to [0, 255] range
    image = image * 255  # Undo normalization

    # Ensure image is in the correct dtype (uint8) for display
    image = image.astype('uint8')

    plt.imshow(image, cmap='gray')
    plt.axis('off')

    # Convert bounding boxes based on the format
    if bbox_format == "center":
        true_bbox = center_to_coco(true_bbox)
        true_bbox = np.array(true_bbox, dtype=np.float16)
        pred_bbox = center_to_coco(pred_bbox)
        pred_bbox = np.array(pred_bbox, dtype=np.float16)

    # Draw the true bounding box if it exists
    if not (true_bbox == [0, 0, 0, 0]).all():
        x, y, w, h = true_bbox
        plt.gca().add_patch(plt.Rectangle((x, y), w, h, linewidth=2, edgecolor='r', face
color='none'))
        #plt.gca().add_patch(plt.Rectangle((x * 360, y * 240), w * 360, h * 240, linewid
th=2, edgecolor='r', facecolor='none'))

    # Draw the predicted bounding box if it exists
    if not (pred_bbox == [0, 0, 0, 0]).all():
        x, y, w, h = pred_bbox
        plt.gca().add_patch(plt.Rectangle((x, y), w, h, linewidth=2, edgecolor='g', face
color='none'))

    # Add title with both true and predicted class
    plt.title(f"True Class: {true_class} [{names_of_class[np.argmax(true_class)]}], Predi
cted Class: {pred_class}[{names_of_class[np.argmax(pred_class)]}]")
    print("IoU:", 1 - iou_loss(true_bbox, pred_bbox, "coco"))
    plt.show()
```

In [3]:

```
# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [4]:

```
# HYPERPARAMETERS

include_MIRROR = True
# - double the dataset by adding mirror images

bbox_format = "center" # "coco" - top left, "center" - center
# - do you want to use default COCO bbox format, or switch to Center bbox format that cou
ld potentially imrpove object detecting

include_MOVM = False
```

In [5]:

```
# Ova skripta učitava podatke i pretvara ih u format pogodan za treniranje

# Path to your COCO dataset files
annotations_path = "/content/drive/MyDrive/Colab Notebooks/FINAL_PALM/result.json"
```

```
images_path = "/content/drive/MyDrive/Colab Notebooks/FINAL_PALM"
if not os.path.exists(images_path):
    print(f"Image {images_path} does not exist!")

# Load COCO dataset
coco = COCO(annotations_path)

# Extract categories
categories = coco.loadCats(coco.getCatIds())
category_names = [cat['name'] for cat in categories]
print("Categories:", category_names)
print()

# Load image and annotation data
image_ids = coco.getImgIds()
images = coco.loadImgs(image_ids)

# Prepare data for Palm and No Palm
data = []
for image_info in images: #[:int(len(images)/2)]:
    img_id = image_info['id']
    img_path = os.path.join(images_path, image_info['file_name'].replace("\\", "/"))
    movm_path2 = "movement\\" + image_info['file_name'].replace("images", "movement").sp
lit("-", 1)[1].replace(".png", "_movement.png")
    movm_path = os.path.join(images_path, movm_path2.replace("\\", "/"))

    # print("Image file name:", image_info['file_name'])
    # print("MOVM file name:", movm_path2)
    # print(f"Image path: {img_path}")
    # print(f"MOVM path: {movm_path}")
    # print()
    anns = coco.loadAnns(coco.getAnnIds(imgIds=[img_id]))

    # Check if Palm exists
    palm_annotations = [ann for ann in anns if (ann['category_id'] != 1)]
    if palm_annotations:
        for ann in palm_annotations:
            bbox = ann['bbox']  # [x, y, width, height]
            if bbox_format == "center":
              bbox = coco_to_center(bbox)
            width, height = Image.open(img_path).size

            # Add original data
            data.append({
                "image_path": img_path,
                "movement_path": movm_path,
                "label": ann['category_id'],  # FLY=0 or PAUSE=2 or UNDEFINED=3
                "bbox": bbox
            })

    else:
        # Add original "No Palm" data
        data.append({
            "image_path": img_path,
            "movement_path": movm_path,
            "label": 1,  # NOPALM=1
            "bbox": [0, 0, 0, 0]
        })


def preprocess_image(image_path, movm_path, bbox, img_size=(240, 360), flip_or_not=False)
: #(480, 720)

    # Load image in grayscale
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    movm_img = None

    if movm_path is not None:
      movm_img = cv2.imread(movm_path, cv2.IMREAD_GRAYSCALE)

      if movm_img is None:
        print(f"Warning: Image at {movm_path} could not be loaded.")
```

```
        return None, None, None

      # Resize and normalize the image to [0, 1]
      movm_img = cv2.resize(movm_img, (img_size[1], img_size[0])) / 255.0

      if flip_or_not:
        movm_img = cv2.flip(movm_img, 1)




    if img is None:
        print(f"Warning: Image at {image_path} could not be loaded.")
        return None, None, None

    # Resize and normalize the image to [0, 1]
    img = cv2.resize(img, (img_size[1], img_size[0])) / 255.0  # Note: width comes first
in OpenCV resize

    # If Palm, normalize bounding box
    if bbox:
        h, w = img_size
        x, y, bw, bh = bbox
        #bbox = [x / w, y / h, bw / w, bh / h]  # Normalize to [0, 1]
    else:
        bbox = [0, 0, 0, 0]  # No bounding box for "No Palm"

    # Apply horizontal flip if requested
    if flip_or_not:
        img = cv2.flip(img, 1)  # Horizontal flip

        # Flip bounding box coordinates
        if bbox != [0, 0, 0, 0]:  # Only flip if there's a valid bounding box
            if bbox_format == "coco":
                #FLIP FOR COCO
                bbox = [
                    img_size[1] - (bbox[0] + bbox[2]),  # Flip x by mirroring x + width
                    bbox[1],                   # y remains the same
                    bbox[2],                   # Width remains the same
                    bbox[3]                    # Height remains the same
                ]
            elif bbox_format == "center":
                # FLIP FOR CENTER BASED BBOX
                bbox = [
                    img_size[1] - bbox[0],  # Flip x by mirroring x + width
                    bbox[1],                   # y remains the same
                    bbox[2],                   # Width remains the same
                    bbox[3]                    # Height remains the same
                ]
    # Cast bbox and label to float16
    bbox = np.array(bbox, dtype=np.float16)

    return img, movm_img, bbox
```

```
loading annotations into memory...
Done (t=0.94s)
creating index...
index created!
Categories: ['FLY', 'NOPALM', 'PAUSE', 'UNDEFINED']
```

In [26]:

```
# Učitavanje podataka, dijeljenje u train/test data, te na kraju definiranje arhitekture
NN. (Nikako ne koristiti batchnormalization layer)


def data_generator(data, img_size=(240, 360), is_include_MOVM=False):
    images, movements, labels, bboxes = [], [], [], []
    for entry in data:
        img, movm_img, bbox = preprocess_image(entry['image_path'], entry['movement_path
```

```
'] if include_MOVM else None, entry['bbox'], img_size, flip_or_not = False)
        if img is not None:
            images.append(img)
            movements.append(movm_img)
            labels.append(entry['label'])
            bboxes.append(bbox)

        if include_MIRROR:
          f_img, f_movm_img, f_bbox = preprocess_image(entry['image_path'], entry['movem
ent_path'] if include_MOVM else None, entry['bbox'], img_size, flip_or_not = True)
          if img is not None:
              images.append(f_img)
              movements.append(f_movm_img)
              labels.append(entry['label'])
              bboxes.append(f_bbox)

    # Convert to numpy arrays and add channel dimension
    images = np.array(images, dtype=np.float16)[..., np.newaxis]

    if is_include_MOVM:
        movements = np.array(movements, dtype=np.float16)[..., np.newaxis]
        images = np.concatenate([images, movements], axis=-1)

    # Combine label and bbox for the output (first element is label, next 4 are bbox)
    outputs = []
    for label, bbox in zip(labels, bboxes):
        # Ensure bbox is a 4-element vector (x, y, width, height)
        assert len(bbox) == 4, f"Expected bbox to have 4 elements, got {len(bbox)}"

        # One-hot encode the label (for 4 classes)
        one_hot_label = np.eye(4)[label]

        # Combine the one-hot label with the bounding box
        output = np.concatenate([one_hot_label, bbox]).astype(np.float16)
        outputs.append(output)

    # Convert outputs to numpy array with dtype float16
    return images, np.array(outputs, dtype=np.float16)


#####################################################

# Prepare training and testing data
IMG_SIZE = (240, 360)
X_all, y_all = data_generator(data, IMG_SIZE, is_include_MOVM = include_MOVM)
#split train/test 70-30
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, test_size=0.3, random_
state=42, shuffle=True)

# Extract labels from y_all
labels = y_all[:, 0]  # Assuming the first element is the label

# Count the occurrences of each label
unique_labels, label_counts = np.unique(labels, return_counts=True)

# Print the results
for label, count in zip(unique_labels, label_counts):
    print(f"Label {label}: {count} samples")



print()
print(f"Training: {len(X_train)} instances, {len(y_train)} labels | Testing: {len(X_test)
} instances, {len(y_test)} labels")
random_indices = random.sample(range(len(y_train)), 10)

# Print y_train and bbox_train for the selected instances
for idx in random_indices:
    print(f"Instance {idx}: Label = {y_train[idx]}")

print("\nX_train shape:", X_train.shape)  # Should be (N, 480, 720, 1)
print("y_train shape:", y_train.shape)  # Should be (N, 5), where 5 is [label, bbox_x, bb
```

```
ox_y, bbox_w, bbox_h]
print("X_test shape:", X_test.shape)  # Should be (N, 480, 720, 1)
print("y_test shape:", y_test.shape)  # Should be (N, 5), where 5 is [label, bbox_x, bbox
_y, bbox_w, bbox_h]


# Define the combined model
# def build_model(input_shape):
#     input_img = layers.Input(shape=input_shape)

#     # Feature extraction
#     # x = layers.Conv2D(8, (5, 5), activation="relu")(input_img)
#     # x = layers.MaxPooling2D((6, 6))(x)
#     # x = layers.Conv2D(24, (3, 3), activation="relu")(x)
#     # x = layers.MaxPooling2D((2, 2))(x)
#     # x = layers.Flatten()(x)
#     # x = layers.Dense(8, activation="relu")(x)

#     x = layers.Conv2D(2, (4, 4), activation="relu")(input_img)
#     x = layers.MaxPooling2D((2, 2))(x)
#     x = layers.Conv2D(4, (3, 3), activation="relu")(input_img)
#     x = layers.MaxPooling2D((6, 6))(x)
#     x = layers.Conv2D(16, (3, 3), activation="relu")(x)
#     x = layers.Flatten()(x)
#     x = layers.Dense(8, activation="relu")(x)


#     # Single output layer for both classification and bounding box
#     output = layers.Dense(5, activation="sigmoid", name="output")(x)

#     model = models.Model(inputs=input_img, outputs=output)
#     model.summary()

#     return model

#CUSTOM NN
#def build_model(input_shape):
    #input_img = layers.Input(shape=input_shape)

    # x = layers.Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    # x = layers.MaxPooling2D((2, 2))(x)

    # # 2nd Conv Block
    # x = layers.Conv2D(32, (2, 2), activation='relu', padding='same')(x)
    # x = layers.MaxPooling2D((2, 2))(x)

    # # 3rd Conv Block
    # x = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    # x = layers.MaxPooling2D((2, 2))(x)

    # # 4th Conv Block
    # x = layers.Conv2D(128, (2, 2), activation='relu', padding='same')(x)
    # x = layers.MaxPooling2D((3, 3))(x)

    # # Flatten the output for fully connected layers
    # x = layers.Flatten()(x)

    # # Fully Connected Layer
    # x = layers.Dense(32, activation='relu')(x)
    # x = layers.Dropout(0.15)(x)  # Dropout to prevent overfitting

##################################
# YOLO V1 NN
def build_model(input_shape, grid_size=7, num_classes=4, isMOVM = False):

    # without MOVM images
    if isMOVM == False:

        input_img = Input(shape=input_shape)

        # Convolutional Backbone
        x = Conv2D(64, (7, 7), strides=2, padding='same', activation=None)(input_img)
```

```
        #x = BatchNormalization()(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)

        x = Conv2D(128, (3, 3), strides=1, padding='same', activation=None)(x)
        #x = BatchNormalization()(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)

        x = Conv2D(64, (1, 1), strides=1, padding='same')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (3, 3), strides=1, padding='same')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (1, 1), strides=1, padding='same')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (3, 3), strides=1, padding='same')(x)
        #x = BatchNormalization()(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)

        # Concatination of convolutional info
        x = Flatten()(x)
        #x = GlobalAveragePooling2D()(x) ILI PROBAT global max pool!!!!

        # This Fully connected layer will be used only by bbox pred head
        x = Dense(128, activation='relu')(x)

        # This Fully connected layer will be used only by classification head
        x_class = Dense(64, activation='relu')(x)
        #x_class = Dense(32, activation='relu')(x_class)

        # Bounding box predictions
        bbox_output = Dense(4, activation='relu', name='bbox_output')(x)

        # Class probabilities
        class_output = Dense(4, activation='softmax', name='class_output')(x_class)


        # Concatenate bbox and class probabilities into a single 8-element output
        output = tf.keras.layers.Concatenate(name='output')([class_output, bbox_output])

        model = models.Model(inputs=input_img, outputs=output)
        # model = Model(inputs=input_img, outputs=[class_output, combined_output])

    #with MOVM images
    else:
        input_img1 = Input(shape=input_shape, name="input_image1")  # Black-and-white im
age
        input_img2 = Input(shape=input_shape, name="input_image2")  # Difference image

        # Branch for input image 1
        x1 = Conv2D(64, (7, 7), strides=2, padding='same', activation=None)(input_img1)
        x1 = LeakyReLU(alpha=0.1)(x1)
        x1 = MaxPooling2D(pool_size=(2, 2), strides=2)(x1)

        x1 = Conv2D(128, (3, 3), strides=1, padding='same', activation=None)(x1)
        x1 = LeakyReLU(alpha=0.1)(x1)
        x1 = MaxPooling2D(pool_size=(2, 2), strides=2)(x1)

        # Branch for input image 2
        x2 = Conv2D(64, (7, 7), strides=2, padding='same', activation=None)(input_img2)
        x2 = LeakyReLU(alpha=0.1)(x2)
        x2 = MaxPooling2D(pool_size=(2, 2), strides=2)(x2)

        x2 = Conv2D(128, (3, 3), strides=1, padding='same', activation=None)(x2)
        x2 = LeakyReLU(alpha=0.1)(x2)
        x2 = MaxPooling2D(pool_size=(2, 2), strides=2)(x2)

        # Merge the features from both branches
        merged = tf.keras.layers.Concatenate()([x1, x2])

        x = Conv2D(64, (1, 1), strides=1, padding='same')(merged)
```

```
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (3, 3), strides=1, padding='same')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (1, 1), strides=1, padding='same')(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = Conv2D(128, (3, 3), strides=1, padding='same')(x)
        #x = BatchNormalization()(x)
        x = LeakyReLU(alpha=0.1)(x)
        x = MaxPooling2D(pool_size=(2, 2), strides=2)(x)

        # Concatination of convolutional info
        x = Flatten()(x)

        # This Fully connected layer will be used only by bbox pred head
        x_local = Dense(128, activation='relu')(x)

        # This Fully connected layer will be used only by classification head
        x_class = Dense(64, activation='relu')(x)
        #x_class = Dense(32, activation='relu')(x_class)

        # Class probabilities
        class_output = Dense(num_classes, activation='softmax', name='class_output')(x_c
lass)


        # Bounding box predictions
        bbox_output = Dense(4, activation='relu', name='bbox_output')(x_local)


        # Concatenate bbox and class probabilities into a single 8-element output
        output = tf.keras.layers.Concatenate(name='output')([class_output, bbox_output])

        model = models.Model(inputs=[input_img1, input_img2], outputs=output)

    model.summary()
    return model
```

```
Label 0.0: 696 samples
Label 1.0: 742 samples

Training: 1006 instances, 1006 labels | Testing: 432 instances, 432 labels
Instance 154: Label = [ 1.    0.    0.    0.   98.5  63.44 31.44 52.53]
Instance 678: Label = [  1.     0.     0.     0.    91.56 134.4   88.7   68.4 ]
Instance 696: Label = [  1.     0.     0.     0.   250.2  147.6   47.78  22.55]
Instance 172: Label = [  0.     0.     1.     0.   187.1   69.7   37.47  57.06]
Instance 313: Label = [  0.     0.     1.     0.   148.4  102.    90.5  104.25]
Instance 775: Label = [  1.     0.     0.     0.    73.3  120.75  49.16  42.72]
Instance 693: Label = [  1.     0.     0.     0.   301.8  137.2  106.75  42.34]
Instance 74: Label = [  0.     0.     0.     1.   109.6   99.8   30.44  45.94]
Instance 434: Label = [  0.     0.    0.    1.   255.8 181.1  48.3   36.4]
Instance 33: Label = [  1.     0.     0.     0.   247.6  160.9   51.38  31.05]

X_train shape: (1006, 240, 360, 1)
y_train shape: (1006, 8)
X_test shape: (432, 240, 360, 1)
y_test shape: (432, 8)
```

In [15]:

```
# Ovo je najvažniji dio, odnosi se na treniranje neuronske mreže, definiranje njene optim
izacije (loss funct.) i metrika da pratimo napredak kroz treniranje
# Nebi trebalo ništa mijenjati, eventualno epochs (koliko rundi treniranja) i learning ra
te (lr), ili izmjeniti samu arhitektutu NN u prethodnim dijelovima
# Loss function je kombinacija mse * iou
# TIP: ako u prvih 10-15 epocha IoU metric bude i dalje 0.0, onda prekini izvođenje koda
i pokreni iznova, dok se ne pojavi bar neki broj u prvih 10 epocha

from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
#tf.config.run_functions_eagerly(True)
#tf.data.experimental.enable_debug_mode()

classification_loss_fn = tf.keras.losses.SparseCategoricalCrossentropy()
```

```
# num_classes
def iou_loss(y_true, y_pred, bbox_format_rn=None):
    """
    Compute IoU Loss between true and predicted bounding boxes.
    y_true and y_pred are expected to be in the format [x, y, w, h].
    """
    if bbox_format_rn is None:
      bbox_format_rn=bbox_format

    y_true = tf.cast(y_true, tf.float32)
    y_pred = tf.cast(y_pred, tf.float32)

    if bbox_format_rn == "center":
        # Convert center-based bbox [cx, cy, w, h] to corner-based bbox [x_min, y_min, x_
max, y_max]
        y_true_x_min = y_true[..., 0] - y_true[..., 2] / 2
        y_true_y_min = y_true[..., 1] - y_true[..., 3] / 2
        y_true_x_max = y_true[..., 0] + y_true[..., 2] / 2
        y_true_y_max = y_true[..., 1] + y_true[..., 3] / 2

        y_pred_x_min = y_pred[..., 0] - y_pred[..., 2] / 2
        y_pred_y_min = y_pred[..., 1] - y_pred[..., 3] / 2
        y_pred_x_max = y_pred[..., 0] + y_pred[..., 2] / 2
        y_pred_y_max = y_pred[..., 1] + y_pred[..., 3] / 2

    elif bbox_format_rn == "coco":
        # For COCO format, use the bbox directly
        y_true_x_min = y_true[..., 0]
        y_true_y_min = y_true[..., 1]
        y_true_x_max = y_true[..., 0] + y_true[..., 2]
        y_true_y_max = y_true[..., 1] + y_true[..., 3]

        y_pred_x_min = y_pred[..., 0]
        y_pred_y_min = y_pred[..., 1]
        y_pred_x_max = y_pred[..., 0] + y_pred[..., 2]
        y_pred_y_max = y_pred[..., 1] + y_pred[..., 3]
    else:
        raise ValueError("Invalid bbox_format. Use 'coco' or 'center'.")

    # Intersection coordinates
    inter_x_min = tf.maximum(y_true_x_min, y_pred_x_min)
    inter_y_min = tf.maximum(y_true_y_min, y_pred_y_min)
    inter_x_max = tf.minimum(y_true_x_max, y_pred_x_max)
    inter_y_max = tf.minimum(y_true_y_max, y_pred_y_max)

    # Intersection area
    inter_area = tf.maximum(0.0, inter_x_max - inter_x_min) * tf.maximum(0.0, inter_y_ma
x - inter_y_min)

    # Union area
    true_area = (y_true_x_max - y_true_x_min) * (y_true_y_max - y_true_y_min)
    pred_area = (y_pred_x_max - y_pred_x_min) * (y_pred_y_max - y_pred_y_min)
    union_area = true_area + pred_area - inter_area

    # IoU
    iou = inter_area / tf.maximum(union_area, 1e-10)  # Avoid division by zero

    # IoU Loss
    return 1 - iou

def total_loss(y_true, y_pred):


    # print("y_true shape:", y_true.shape)
    # print("y_pred shape:", y_pred.shape)

    # # # Print the first few elements of y_true and y_pred to inspect
    # tf.print("y_true values:", y_true)  # Only the first 5 elements
    # tf.print("y_pred values:", y_pred)  # Only the first 5 elements

    class_true = y_true[..., 0:4]
```

```
    bbox_true = y_true[..., 4:]

    class_pred = y_pred[..., 0:4]
    bbox_pred = y_pred[..., 4:]

    # Create a mask to separate Palm and No Palm
    class_mask = tf.cast(tf.logical_or( tf.argmax(class_true, axis=-1) == 0, tf.argmax(c
lass_true, axis=-1) == 2), tf.float32)

    squared_errors = tf.square(bbox_true - bbox_pred)  # Shape: (batch_size, 4)

    # Per-sample loss (mean across bbox coordinates)
    per_sample_loss = tf.reduce_mean(squared_errors, axis=-1)  # Shape: (batch_size,)

    # Batch-level loss (mean across the batch)
    batch_loss = tf.reduce_mean(per_sample_loss)

    total_bbox_loss = batch_loss



    # For No Palm (label=0), only calculate class loss
    # For Palm (label=1), calculate both class and bbox loss
    abs_bbox_loss = total_bbox_loss * class_mask  # Apply mask to bbox loss (ignore bbox
loss if NOPALM or UNDEFINED)


    class_true_labels = tf.argmax(class_true, axis=-1)
    classification_loss = classification_loss_fn(class_true_labels, class_pred)
    # classification_loss = tf.equal(class_true, class_pred)
    # classification_loss = tf.cast(classification_loss, tf.float32)

    # tf.print("class_true:", class_true)
    # tf.print("class_pred:", class_pred)
    # tf.print("classification_loss:", classification_loss, "\n")

    # IoU loss
    bbox_iou_loss = iou_loss(bbox_true, bbox_pred, bbox_format)
    bbox_iou_loss = bbox_iou_loss * class_mask

    # Total loss
    #return 0 * classification_loss + 0 * bbox_iou_loss * 100 + 1 * abs_bbox_loss
    #return 0 * classification_loss + bbox_iou_loss * abs_bbox_loss * 10/7 #OVA JE GOAT
    return 2 * classification_loss**2 + 1 * (bbox_iou_loss+1)**2 * abs_bbox_loss #1000 *
classification_loss**4 +

# Define the custom Sparse Categorical Crossentropy loss function
# def custom_scc_loss(y_true, y_pred):
#     """
#     Custom Sparse Categorical Crossentropy Loss.
#     y_true: One-hot encoded true labels (batch_size, num_classes).
#     y_pred: Predicted probabilities (batch_size, num_classes).
#     """

#     class_true = y_true[..., 0:4]
#     #bbox_true = y_true[..., 4:]

#     class_pred = y_pred[..., 0:4]
#     #bbox_pred = y_pred[..., 4:]

#     # Compute SCCE
#     # class_true_labels = tf.argmax(class_true, axis=-1)
#     # loss = classification_loss_fn(class_true_labels, class_pred)
#     loss = -tf.reduce_sum(class_true * tf.math.log(class_pred + 1e-7), axis=-1)  # Add
epsilon for numerical stability
#     return tf.reduce_mean(loss)


def iou_metric(y_true, y_pred):
    class_true = y_true[..., 0:4]
    bbox_true = y_true[..., 4:]
    bbox_pred = y_pred[..., 4:]
```

```
    iou = 1 - iou_loss(bbox_true, bbox_pred)  # Since `iou_loss` returns 1 - IoU

    # Apply the condition: if true class is 1 or 3, set IoU to 0
    condition = tf.logical_or(tf.argmax(class_true, axis=-1) == 1, tf.argmax(class_true,
axis=-1) == 3)
    iou = tf.where(condition, 0.0, iou)

    return iou * (10/7) # -> *(10/7) zato jer 30% podataka class(NOPALM ili UNDEFINED) p
a kvare statistiku za cca ovaj omjer

def class_accuracy(y_true, y_pred):

    class_true = y_true[..., 0:4]
    class_pred = y_pred[..., 0:4]


    # Compare the true class with the predicted class
    class_true_index = tf.cast(tf.argmax(class_true, axis=-1), tf.float32)
    class_pred_index = tf.cast(tf.argmax(class_pred, axis=-1), tf.float32)

    # Compare the predicted class index with the true class index
    correct_predictions = tf.equal(class_true_index, class_pred_index)

    # Calculate accuracy: the number of correct predictions divided by the total number o
f samples
    accuracy = tf.cast(correct_predictions, tf.float32)
    #accuracy = tf.reduce_mean(tf.cast(correct_predictions, tf.float32))

    return accuracy


def mse_metric(y_true, y_pred):
    """
    Custom MSE metric with conditional logic.
    If y_true[..., 0] is 1 or 3, the MSE for those samples is set to 0.
    """
    # Extract bounding box coordinates and cast to float32
    bbox_true = tf.cast(y_true[..., 4:], tf.float32)
    bbox_pred = tf.cast(y_pred[..., 4:], tf.float32)
    # bbox_true = tf.cast(y_true[..., 4:], tf.float32)
    # bbox_pred = tf.cast(y_pred[..., 4:], tf.float32)
    class_true = y_true[..., 0:4]

    # Compute squared differences
    mse_per_sample = tf.reduce_mean(tf.square(bbox_true - bbox_pred), axis=-1)

    # Condition: Set MSE to 0 for specific class labels
    condition = tf.logical_or(tf.argmax(class_true, axis=-1) == 1, tf.argmax(class_true,
axis=-1) == 3)
    mse_per_sample = tf.where(condition, 0.0, mse_per_sample)

    # Return mean MSE across the batch
    return tf.reduce_mean(mse_per_sample) * (10/7) # -> *(10/7) zato jer 30% podataka cl
ass(NOPALM ili UNDEFINED) pa kvare statistiku za cca ovaj omjer



early_stopping = EarlyStopping(
    monitor='val_loss',         # Monitor validation loss
    patience=10,                # Stop after 10 epochs of no improvement
    restore_best_weights=True, # Restore weights from the epoch with the best validation
loss
    verbose=1                   # Print message when stopping
)

### GLOBAL ###############
# lr_scheduler = ReduceLROnPlateau(
#     monitor='val_loss',  # You can change this to 'val_iou' or another metric
#     factor=2/3,          # Reduce learning rate by half
#     patience=5,          # Number of epochs to wait for improvement before reducing th
e LR
#     min_lr= 0.00015, #1e-7          # Minimum learning rate
```

```
#     verbose=1
# )

# optimizer = Adam(learning_rate=0.001)
# # Glob avg pool: start lr from: 0.0015, stop lr: 0.00015, patience=5, factor=2/3, just
one: 0.001
# # Flatten: start lr: 0.000032, stop lr: 0.000002, just_one: 0.000008
############################



### FLATTEN ################
lr_scheduler = ReduceLROnPlateau(
    monitor='val_loss',  # You can change this to 'val_iou' or another metric
    factor=0.5,          # Reduce learning rate by half
    patience=3,          # Number of epochs to wait for improvement before reducing the
LR
    min_lr= 0.00000002, #1e-7          # Minimum learning rate
    verbose=1
)

optimizer = Adam(learning_rate=0.000052) #optimizer = Adam(learning_rate=0.000032)
# Glob avg pool: start lr from: 0.0015, stop lr: 0.00015, patience=5, factor=2/3, just on
e: 0.001
# Flatten: start lr: 0.000032, stop lr: 0.000002, just_one: 0.000008
############################

# BUILD MODEL
model = build_model((IMG_SIZE[0], IMG_SIZE[1], 1), isMOVM = include_MOVM)

model.compile(
    optimizer=optimizer,
    loss=total_loss,  # Using custom loss
    metrics=[
            class_accuracy,       # Tracks % of the total data that was correctly predic
ted
            iou_metric,           # Track average IoU across the epoch
            mse_metric           # Track Mean Squared Error (MSE)
    ]
)


# model.compile(
#     optimizer=optimizer,
#     #loss=total_loss,  # Using custom loss
#     loss={
#         'class_output': custom_scc_loss,
#         'comb_output': total_loss,
#     },
#     loss_weights={
#         'class_output': 0.5,
#         'comb_output': 1.0,
#     },
#     # metrics=[
#     #     #BinaryAccuracy(name="class_accuracy"),  # Track binary classification accur
acy
#     #     class_accuracy,       # Tracks % of the total data that was correctly predic
ted
#     #     iou_metric,           # Track average IoU across the epoch
#     #     mse_metric           # Track Mean Squared Error (MSE)

#     # ]
#         metrics={
#         'class_output': [class_accuracy],  # Metrics for the class_output (classificati
on)
#         'comb_output': [iou_metric, mse_metric],  # Metrics for the bbox_output (boundi
ng box)
#     }
# )

# Train the model
history = model.fit(
```

```
    [X_train[..., 0], X_train[..., 1]] if include_MOVM else X_train,
    y_train,
    validation_data=([X_test[..., 0], X_test[..., 1]] if include_MOVM else X_test, y_tes
t),  # y_test also contains the same structure
    epochs=150, #150, #50,
    batch_size=1,
    callbacks=[lr_scheduler, early_stopping]
)
```

Model: "functional_7"

Layer (type)	Output Shape	Param #	Connected to
input_layer_7 (InputLayer)	(None, 240, 360, 1)	0	-
cast_7 (Cast)	(None, 240, 360, 1)	0	input_layer_7[0][0]
conv2d_42 (Conv2D)	(None, 120, 180, 64)	3,200	cast_7[0][0]
leaky_re_lu_42 (LeakyReLU)	(None, 120, 180, 64)	0	conv2d_42[0][0]
max_pooling2d_21 (MaxPooling2D)	(None, 60, 90, 64)	0	leaky_re_lu_42[0][0]
conv2d_43 (Conv2D)	(None, 60, 90, 128)	73,856	max_pooling2d_21[0][0]
leaky_re_lu_43 (LeakyReLU)	(None, 60, 90, 128)	0	conv2d_43[0][0]
max_pooling2d_22 (MaxPooling2D)	(None, 30, 45, 128)	0	leaky_re_lu_43[0][0]
conv2d_44 (Conv2D)	(None, 30, 45, 64)	8,256	max_pooling2d_22[0][0]
leaky_re_lu_44	(None, 30, 45, 64)	0	conv2d_44[0][0]

```
│ (LeakyReLU)                │                    │               │
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ conv2d_45 (Conv2D)         │ (None, 30, 45, 128) │        73,856 │ leaky_re_lu_44[0]
[0]     │
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ leaky_re_lu_45             │ (None, 30, 45, 128) │             0 │ conv2d_45[0][0]
│
│ (LeakyReLU)                │                    │               │
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ conv2d_46 (Conv2D)         │ (None, 30, 45, 128) │        16,512 │ leaky_re_lu_45[0]
[0]     │
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ leaky_re_lu_46             │ (None, 30, 45, 128) │             0 │ conv2d_46[0][0]
│
│ (LeakyReLU)                │                    │               │
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ conv2d_47 (Conv2D)         │ (None, 30, 45, 128) │       147,584 │ leaky_re_lu_46[0]
[0]     │
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ leaky_re_lu_47             │ (None, 30, 45, 128) │             0 │ conv2d_47[0][0]
│
│ (LeakyReLU)                │                    │               │
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ max_pooling2d_23           │ (None, 15, 22, 128) │             0 │ leaky_re_lu_47[0]
[0]     │
│ (MaxPooling2D)             │                    │               │
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ flatten_7 (Flatten)        │ (None, 42240)      │             0 │ max_pooling2d_23[
0][0] │
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ dense_7 (Dense)            │ (None, 128)        │     5,406,848 │ flatten_7[0][0]
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ dense_8 (Dense)            │ (None, 64)         │         8,256 │ dense_7[0][0]
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ class_output (Dense)       │ (None, 4)          │           260 │ dense_8[0][0]
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ bbox_output (Dense)        │ (None, 4)          │           516 │ dense_7[0][0]
│
├────────────────────────────┼────────────────────┼───────────────┼──────────────────
────────┤
│ output (Concatenate)       │ (None, 8)          │             0 │ class_output[0][0
],      │
│                            │                    │               │ bbox_output[0][0]
│
```

Total params: 5,739,144 (21.89 MB)

Trainable params: 5,739,144 (21.89 MB)

Non-trainable params: 0 (0.00 B)

```
Epoch 1/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 16s 8ms/step - class_accuracy: 0.3761 - iou_metric: 0.0000e+00 - loss: 20627.0156 - mse_metric: 7322.3569 - val_class_accuracy: 0.3611 - val_iou_metric: 0.0000e+00 - val_loss: 8531.4385 - val_mse_metric: 3014.0027 - learning_rate: 5.2000e-05
Epoch 2/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 13s 5ms/step - class_accuracy: 0.4077 - iou_metric: 0.0000e+00 - loss: 9413.2627 - mse_metric: 3328.8225 - val_class_accuracy: 0.2616 - val_iou_metric: 0.0000e+00 - val_loss: 8088.9614 - val_mse_metric: 2855.9639 - learning_rate: 5.2000e-05
Epoch 3/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - class_accuracy: 0.4030 - iou_metric: 0.0000e+00 - loss: 8968.0938 - mse_metric: 3170.2717 - val_class_accuracy: 0.4514 - val_iou_metric: 0.0000e+00 - val_loss: 7371.6133 - val_mse_metric: 2599.5134 - learning_rate: 5.2000e-05
Epoch 4/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.4102 - iou_metric: 0.0000e+00 - loss: 7607.3794 - mse_metric: 2683.3328 - val_class_accuracy: 0.4144 - val_iou_metric: 0.0000e+00 - val_loss: 6171.8535 - val_mse_metric: 2170.2578 - learning_rate: 5.2000e-05
Epoch 5/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.4040 - iou_metric: 0.0378 - loss: 5560.7803 - mse_metric: 1974.3972 - val_class_accuracy: 0.3611 - val_iou_metric: 0.1504 - val_loss: 2600.7688 - val_mse_metric: 989.4973 - learning_rate: 5.2000e-05
Epoch 6/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.3889 - iou_metric: 0.1571 - loss: 2707.0737 - mse_metric: 1028.4290 - val_class_accuracy: 0.5069 - val_iou_metric: 0.1575 - val_loss: 2440.6172 - val_mse_metric: 917.4199 - learning_rate: 5.2000e-05
Epoch 7/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.4755 - iou_metric: 0.1881 - loss: 2186.7153 - mse_metric: 847.2201 - val_class_accuracy: 0.4653 - val_iou_metric: 0.2226 - val_loss: 1740.8560 - val_mse_metric: 682.5159 - learning_rate: 5.2000e-05
Epoch 8/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - class_accuracy: 0.5090 - iou_metric: 0.2151 - loss: 1695.0743 - mse_metric: 664.6793 - val_class_accuracy: 0.4190 - val_iou_metric: 0.2201 - val_loss: 1666.9321 - val_mse_metric: 650.2409 - learning_rate: 5.2000e-05
Epoch 9/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.5594 - iou_metric: 0.2436 - loss: 1519.1278 - mse_metric: 604.9079 - val_class_accuracy: 0.4907 - val_iou_metric: 0.2230 - val_loss: 1752.8549 - val_mse_metric: 673.2213 - learning_rate: 5.2000e-05
Epoch 10/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - class_accuracy: 0.5674 - iou_metric: 0.2419 - loss: 1420.2056 - mse_metric: 558.7230 - val_class_accuracy: 0.4630 - val_iou_metric: 0.2324 - val_loss: 1634.2407 - val_mse_metric: 633.3495 - learning_rate: 5.2000e-05
Epoch 11/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.6629 - iou_metric: 0.2483 - loss: 1132.8005 - mse_metric: 458.8701 - val_class_accuracy: 0.5995 - val_iou_metric: 0.2367 - val_loss: 1427.0640 - val_mse_metric: 560.1278 - learning_rate: 5.2000e-05
Epoch 12/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.6386 - iou_metric: 0.3030 - loss: 978.1494 - mse_metric: 407.9620 - val_class_accuracy: 0.6111 - val_iou_metric: 0.2764 - val_loss: 1233.8810 - val_mse_metric: 482.3761 - learning_rate: 5.2000e-05
Epoch 13/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - class_accuracy: 0.6952 - iou_metric: 0.2971 - loss: 1000.9975 - mse_metric: 411.6629 - val_class_accuracy: 0.6227 - val_iou_metric: 0.2817 - val_loss: 1139.6327 - val_mse_metric: 450.3763 - learning_rate: 5.2000e-05
Epoch 14/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.6811 - iou_metric: 0.3159 - loss: 877.7830 - mse_metric: 365.0973 - val_class_accuracy: 0.6088 - val_iou_metric: 0.2995 - val_loss: 1057.1931 - val_mse_metric: 421.8191 - learning_rate: 5.2000e-05
Epoch 15/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.7060 - iou_metric: 0.3174 - loss: 726.0701 - mse_metric: 306.4318 - val_class_accuracy: 0.6528 - val_iou_metric: 0.
```

```
2988 - val_loss: 997.8541 - val_mse_metric: 403.2990 - learning_rate: 5.2000e-05
Epoch 16/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.7245 - iou_metric: 0.3517
- loss: 682.0455 - mse_metric: 293.0174 - val_class_accuracy: 0.6134 - val_iou_metric: 0.
2696 - val_loss: 1121.5245 - val_mse_metric: 441.1426 - learning_rate: 5.2000e-05
Epoch 17/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - class_accuracy: 0.7146 - iou_metric: 0.3131
- loss: 616.7276 - mse_metric: 261.6057 - val_class_accuracy: 0.6204 - val_iou_metric: 0.
3088 - val_loss: 888.9110 - val_mse_metric: 361.7423 - learning_rate: 5.2000e-05
Epoch 18/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.7683 - iou_metric: 0.3742
- loss: 459.8280 - mse_metric: 209.5753 - val_class_accuracy: 0.6319 - val_iou_metric: 0.
2523 - val_loss: 1087.8544 - val_mse_metric: 425.5473 - learning_rate: 5.2000e-05
Epoch 19/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.7636 - iou_metric: 0.3461
- loss: 518.3761 - mse_metric: 224.3706 - val_class_accuracy: 0.6736 - val_iou_metric: 0.
2589 - val_loss: 1036.5045 - val_mse_metric: 408.7679 - learning_rate: 5.2000e-05
Epoch 20/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.7464 - iou_metric: 0.3935
- loss: 422.3975 - mse_metric: 185.3858 - val_class_accuracy: 0.6620 - val_iou_metric: 0.
3261 - val_loss: 809.9001 - val_mse_metric: 328.3138 - learning_rate: 5.2000e-05
Epoch 21/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.8008 - iou_metric: 0.4319
- loss: 344.5787 - mse_metric: 162.2257 - val_class_accuracy: 0.6806 - val_iou_metric: 0.
3458 - val_loss: 749.7704 - val_mse_metric: 304.4341 - learning_rate: 5.2000e-05
Epoch 22/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.8095 - iou_metric: 0.4709
- loss: 282.0826 - mse_metric: 133.5281 - val_class_accuracy: 0.6250 - val_iou_metric: 0.
3454 - val_loss: 734.8336 - val_mse_metric: 297.0541 - learning_rate: 5.2000e-05
Epoch 23/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.8027 - iou_metric: 0.4607
- loss: 285.9562 - mse_metric: 138.0040 - val_class_accuracy: 0.6736 - val_iou_metric: 0.
3555 - val_loss: 702.4413 - val_mse_metric: 286.1792 - learning_rate: 5.2000e-05
Epoch 24/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.8171 - iou_metric: 0.4449
- loss: 233.0018 - mse_metric: 115.1150 - val_class_accuracy: 0.4352 - val_iou_metric: 0.
3303 - val_loss: 866.2593 - val_mse_metric: 330.8800 - learning_rate: 5.2000e-05
Epoch 25/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.8318 - iou_metric: 0.4920
- loss: 231.5166 - mse_metric: 112.5189 - val_class_accuracy: 0.6343 - val_iou_metric: 0.
3317 - val_loss: 753.8298 - val_mse_metric: 307.3468 - learning_rate: 5.2000e-05
Epoch 26/150
1003/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.8191 - iou_metric: 0.4657
- loss: 220.6829 - mse_metric: 108.6716
Epoch 26: ReduceLROnPlateau reducing learning rate to 2.5999999706982635e-05.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.8191 - iou_metric: 0.4657
- loss: 220.7547 - mse_metric: 108.7026 - val_class_accuracy: 0.7060 - val_iou_metric: 0.
3465 - val_loss: 710.0319 - val_mse_metric: 292.0042 - learning_rate: 5.2000e-05
Epoch 27/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.8766 - iou_metric: 0.5321
- loss: 132.4313 - mse_metric: 71.3822 - val_class_accuracy: 0.6782 - val_iou_metric: 0.3
695 - val_loss: 669.4116 - val_mse_metric: 272.6316 - learning_rate: 2.6000e-05
Epoch 28/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.8880 - iou_metric: 0.5540
- loss: 124.4042 - mse_metric: 67.3281 - val_class_accuracy: 0.7014 - val_iou_metric: 0.3
889 - val_loss: 626.9677 - val_mse_metric: 257.9487 - learning_rate: 2.6000e-05
Epoch 29/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9073 - iou_metric: 0.5923
- loss: 97.7967 - mse_metric: 56.2162 - val_class_accuracy: 0.7037 - val_iou_metric: 0.38
47 - val_loss: 610.2598 - val_mse_metric: 250.0912 - learning_rate: 2.6000e-05
Epoch 30/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 11s 6ms/step - class_accuracy: 0.8904 - iou_metric: 0.5895
- loss: 91.0143 - mse_metric: 53.1640 - val_class_accuracy: 0.6968 - val_iou_metric: 0.38
77 - val_loss: 601.9846 - val_mse_metric: 247.6160 - learning_rate: 2.6000e-05
Epoch 31/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.9198 - iou_metric: 0.5735
- loss: 96.8888 - mse_metric: 56.1842 - val_class_accuracy: 0.6620 - val_iou_metric: 0.40
44 - val_loss: 575.3612 - val_mse_metric: 238.4911 - learning_rate: 2.6000e-05
Epoch 32/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9151 - iou_metric: 0.5962
- loss: 75.9031 - mse_metric: 46.0760 - val_class_accuracy: 0.6481 - val_iou_metric: 0.39
28 - val_loss: 586.6464 - val_mse_metric: 242.4448 - learning_rate: 2.6000e-05
```

```
Epoch 33/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9171 - iou_metric: 0.5825
- loss: 77.4031 - mse_metric: 46.3078 - val_class_accuracy: 0.6875 - val_iou_metric: 0.25
94 - val_loss: 936.6805 - val_mse_metric: 370.9520 - learning_rate: 2.6000e-05
Epoch 34/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9203 - iou_metric: 0.5490
- loss: 100.7263 - mse_metric: 57.6677 - val_class_accuracy: 0.6782 - val_iou_metric: 0.4
025 - val_loss: 563.9953 - val_mse_metric: 233.9018 - learning_rate: 2.6000e-05
Epoch 35/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9462 - iou_metric: 0.6233
- loss: 69.3015 - mse_metric: 42.6301 - val_class_accuracy: 0.7014 - val_iou_metric: 0.39
47 - val_loss: 574.9741 - val_mse_metric: 237.9975 - learning_rate: 2.6000e-05
Epoch 36/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - class_accuracy: 0.9289 - iou_metric: 0.5991
- loss: 61.4953 - mse_metric: 38.5578 - val_class_accuracy: 0.6435 - val_iou_metric: 0.39
87 - val_loss: 560.0239 - val_mse_metric: 233.3848 - learning_rate: 2.6000e-05
Epoch 37/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.9278 - iou_metric: 0.6658
- loss: 53.7378 - mse_metric: 35.3933 - val_class_accuracy: 0.6921 - val_iou_metric: 0.40
01 - val_loss: 578.7515 - val_mse_metric: 238.8418 - learning_rate: 2.6000e-05
Epoch 38/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9417 - iou_metric: 0.6692
- loss: 55.9984 - mse_metric: 35.9875 - val_class_accuracy: 0.6991 - val_iou_metric: 0.41
09 - val_loss: 590.1279 - val_mse_metric: 241.1058 - learning_rate: 2.6000e-05
Epoch 39/150
 996/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.9350 - iou_metric: 0.6405
- loss: 51.8708 - mse_metric: 34.0935
Epoch 39: ReduceLROnPlateau reducing learning rate to 1.2999999853491317e-05.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9349 - iou_metric: 0.6405
- loss: 51.8899 - mse_metric: 34.1030 - val_class_accuracy: 0.6944 - val_iou_metric: 0.39
76 - val_loss: 584.3257 - val_mse_metric: 238.3082 - learning_rate: 2.6000e-05
Epoch 40/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9412 - iou_metric: 0.6855
- loss: 33.2144 - mse_metric: 23.1098 - val_class_accuracy: 0.6968 - val_iou_metric: 0.41
60 - val_loss: 547.9374 - val_mse_metric: 226.5429 - learning_rate: 1.3000e-05
Epoch 41/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9724 - iou_metric: 0.7323
- loss: 31.4277 - mse_metric: 22.3261 - val_class_accuracy: 0.6991 - val_iou_metric: 0.41
74 - val_loss: 538.9340 - val_mse_metric: 223.2766 - learning_rate: 1.3000e-05
Epoch 42/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9465 - iou_metric: 0.7254
- loss: 26.0226 - mse_metric: 19.0376 - val_class_accuracy: 0.6991 - val_iou_metric: 0.41
16 - val_loss: 553.3980 - val_mse_metric: 229.4954 - learning_rate: 1.3000e-05
Epoch 43/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.9631 - iou_metric: 0.7087
- loss: 26.7892 - mse_metric: 19.7950 - val_class_accuracy: 0.6458 - val_iou_metric: 0.41
56 - val_loss: 546.3019 - val_mse_metric: 224.8112 - learning_rate: 1.3000e-05
Epoch 44/150
 999/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - class_accuracy: 0.9571 - iou_metric: 0.7341
- loss: 25.0281 - mse_metric: 18.6892
Epoch 44: ReduceLROnPlateau reducing learning rate to 6.499999926745659e-06.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.9571 - iou_metric: 0.7339
- loss: 25.0439 - mse_metric: 18.6987 - val_class_accuracy: 0.6852 - val_iou_metric: 0.41
94 - val_loss: 547.8863 - val_mse_metric: 225.5279 - learning_rate: 1.3000e-05
Epoch 45/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9712 - iou_metric: 0.7331
- loss: 18.4967 - mse_metric: 14.5300 - val_class_accuracy: 0.6968 - val_iou_metric: 0.42
40 - val_loss: 535.9304 - val_mse_metric: 222.0641 - learning_rate: 6.5000e-06
Epoch 46/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9803 - iou_metric: 0.7550
- loss: 17.4729 - mse_metric: 13.6407 - val_class_accuracy: 0.6829 - val_iou_metric: 0.42
80 - val_loss: 538.5540 - val_mse_metric: 223.0716 - learning_rate: 6.5000e-06
Epoch 47/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - class_accuracy: 0.9865 - iou_metric: 0.7665
- loss: 18.5896 - mse_metric: 15.0692 - val_class_accuracy: 0.6921 - val_iou_metric: 0.42
44 - val_loss: 532.3173 - val_mse_metric: 220.4612 - learning_rate: 6.5000e-06
Epoch 48/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.9825 - iou_metric: 0.7651
- loss: 16.0539 - mse_metric: 13.0456 - val_class_accuracy: 0.7060 - val_iou_metric: 0.42
74 - val_loss: 533.2076 - val_mse_metric: 220.7447 - learning_rate: 6.5000e-06
Epoch 49/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9853 - iou_metric: 0.7377
```

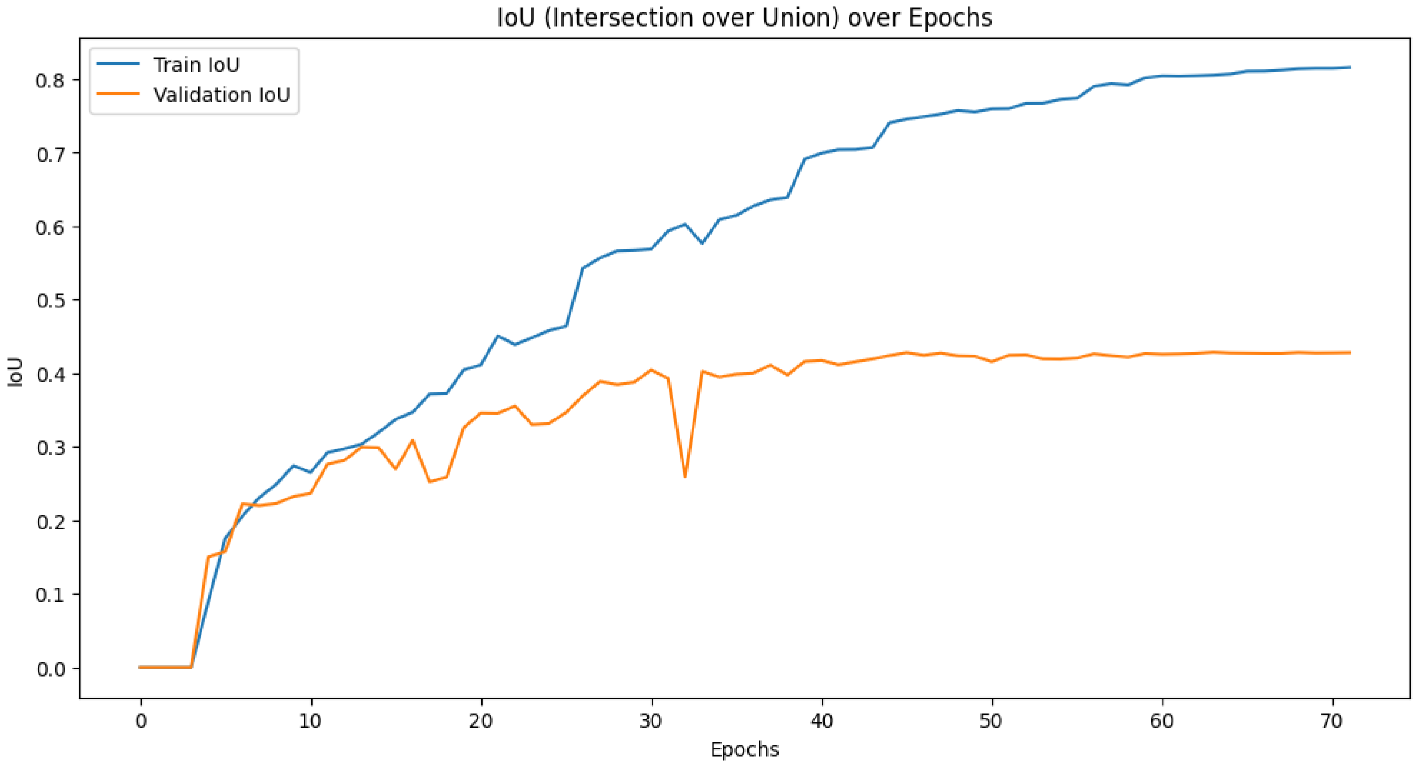
```
- loss: 16.0919 - mse_metric: 13.2210 - val_class_accuracy: 0.6991 - val_iou_metric: 0.42
36 - val_loss: 534.7366 - val_mse_metric: 221.6014 - learning_rate: 6.5000e-06
Epoch 50/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9788 - iou_metric: 0.7505
- loss: 16.0819 - mse_metric: 13.1877 - val_class_accuracy: 0.7014 - val_iou_metric: 0.42
30 - val_loss: 530.6917 - val_mse_metric: 219.5492 - learning_rate: 6.5000e-06
Epoch 51/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9847 - iou_metric: 0.7555
- loss: 15.8945 - mse_metric: 13.0344 - val_class_accuracy: 0.6875 - val_iou_metric: 0.41
57 - val_loss: 537.7115 - val_mse_metric: 222.7533 - learning_rate: 6.5000e-06
Epoch 52/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.9866 - iou_metric: 0.7649
- loss: 15.3199 - mse_metric: 12.5530 - val_class_accuracy: 0.6944 - val_iou_metric: 0.42
45 - val_loss: 533.1596 - val_mse_metric: 220.6167 - learning_rate: 6.5000e-06
Epoch 53/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9862 - iou_metric: 0.7611
- loss: 12.5356 - mse_metric: 10.6450 - val_class_accuracy: 0.6991 - val_iou_metric: 0.42
49 - val_loss: 528.9001 - val_mse_metric: 218.8921 - learning_rate: 6.5000e-06
Epoch 54/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9807 - iou_metric: 0.7597
- loss: 14.4797 - mse_metric: 11.5935 - val_class_accuracy: 0.6991 - val_iou_metric: 0.41
95 - val_loss: 537.5649 - val_mse_metric: 221.7848 - learning_rate: 6.5000e-06
Epoch 55/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9814 - iou_metric: 0.7822
- loss: 13.3445 - mse_metric: 11.2500 - val_class_accuracy: 0.7060 - val_iou_metric: 0.41
91 - val_loss: 535.7968 - val_mse_metric: 221.8777 - learning_rate: 6.5000e-06
Epoch 56/150
1002/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.9892 - iou_metric: 0.8056
- loss: 13.4284 - mse_metric: 11.4169
Epoch 56: ReduceLROnPlateau reducing learning rate to 3.2499999633728294e-06.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9891 - iou_metric: 0.8054
- loss: 13.4281 - mse_metric: 11.4164 - val_class_accuracy: 0.7037 - val_iou_metric: 0.42
07 - val_loss: 534.8378 - val_mse_metric: 220.9633 - learning_rate: 6.5000e-06
Epoch 57/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.9946 - iou_metric: 0.7923
- loss: 10.1104 - mse_metric: 8.8584 - val_class_accuracy: 0.6991 - val_iou_metric: 0.426
4 - val_loss: 530.1531 - val_mse_metric: 219.4250 - learning_rate: 3.2500e-06
Epoch 58/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9914 - iou_metric: 0.7819
- loss: 11.0332 - mse_metric: 9.5416 - val_class_accuracy: 0.7037 - val_iou_metric: 0.423
8 - val_loss: 530.3414 - val_mse_metric: 219.4827 - learning_rate: 3.2500e-06
Epoch 59/150
1003/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - class_accuracy: 0.9917 - iou_metric: 0.7716
- loss: 10.6909 - mse_metric: 9.2749
Epoch 59: ReduceLROnPlateau reducing learning rate to 1.6249999816864147e-06.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9916 - iou_metric: 0.7717
- loss: 10.6916 - mse_metric: 9.2756 - val_class_accuracy: 0.7037 - val_iou_metric: 0.421
8 - val_loss: 531.4850 - val_mse_metric: 219.6875 - learning_rate: 3.2500e-06
Epoch 60/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - class_accuracy: 0.9928 - iou_metric: 0.8150
- loss: 9.1175 - mse_metric: 8.2592 - val_class_accuracy: 0.7060 - val_iou_metric: 0.4268
- val_loss: 528.0565 - val_mse_metric: 218.3153 - learning_rate: 1.6250e-06
Epoch 61/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9931 - iou_metric: 0.7880
- loss: 9.5410 - mse_metric: 8.4967 - val_class_accuracy: 0.6968 - val_iou_metric: 0.4259
- val_loss: 530.1737 - val_mse_metric: 219.2425 - learning_rate: 1.6250e-06
Epoch 62/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9952 - iou_metric: 0.8022
- loss: 10.1751 - mse_metric: 8.9312 - val_class_accuracy: 0.6944 - val_iou_metric: 0.426
3 - val_loss: 527.3990 - val_mse_metric: 218.0893 - learning_rate: 1.6250e-06
Epoch 63/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9941 - iou_metric: 0.8169
- loss: 9.3246 - mse_metric: 8.4248 - val_class_accuracy: 0.6944 - val_iou_metric: 0.4270
- val_loss: 528.9360 - val_mse_metric: 218.6078 - learning_rate: 1.6250e-06
Epoch 64/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.9905 - iou_metric: 0.8108
- loss: 8.6568 - mse_metric: 7.9662 - val_class_accuracy: 0.6921 - val_iou_metric: 0.4286
- val_loss: 530.2296 - val_mse_metric: 219.2297 - learning_rate: 1.6250e-06
Epoch 65/150
1001/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.9930 - iou_metric: 0.7801
- loss: 8.8411 - mse_metric: 7.9210
Epoch 65: ReduceLROnPlateau reducing learning rate to 8.124999908432073e-07.
```

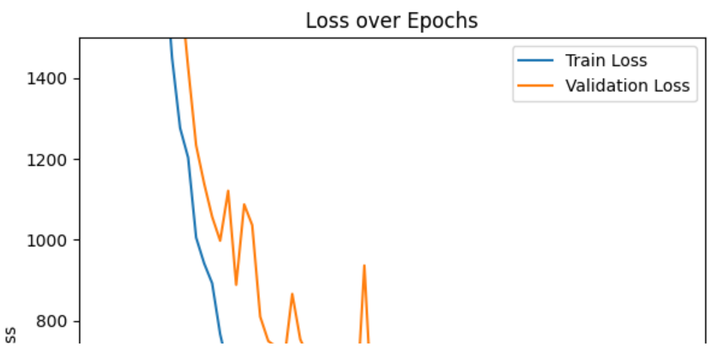
```
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9930 - iou_metric: 0.7803
- loss: 8.8440 - mse_metric: 7.9233 - val_class_accuracy: 0.6944 - val_iou_metric: 0.4275
- val_loss: 527.9388 - val_mse_metric: 218.3549 - learning_rate: 1.6250e-06
Epoch 66/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9991 - iou_metric: 0.8393
- loss: 8.9136 - mse_metric: 7.9937 - val_class_accuracy: 0.6944 - val_iou_metric: 0.4272
- val_loss: 528.8261 - val_mse_metric: 218.3917 - learning_rate: 8.1250e-07
Epoch 67/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9932 - iou_metric: 0.8300
- loss: 8.2230 - mse_metric: 7.5977 - val_class_accuracy: 0.7014 - val_iou_metric: 0.4270
- val_loss: 528.0361 - val_mse_metric: 218.2209 - learning_rate: 8.1250e-07
Epoch 68/150
1002/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.9948 - iou_metric: 0.7873
- loss: 8.3555 - mse_metric: 7.6161
Epoch 68: ReduceLROnPlateau reducing learning rate to 4.0624999542160367e-07.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - class_accuracy: 0.9948 - iou_metric: 0.7874
- loss: 8.3570 - mse_metric: 7.6171 - val_class_accuracy: 0.6991 - val_iou_metric: 0.4270
- val_loss: 529.5509 - val_mse_metric: 218.6602 - learning_rate: 8.1250e-07
Epoch 69/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9954 - iou_metric: 0.8207
- loss: 8.9603 - mse_metric: 7.8578 - val_class_accuracy: 0.6968 - val_iou_metric: 0.4282
- val_loss: 528.0030 - val_mse_metric: 218.2516 - learning_rate: 4.0625e-07
Epoch 70/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - class_accuracy: 0.9939 - iou_metric: 0.8138
- loss: 8.8107 - mse_metric: 7.9107 - val_class_accuracy: 0.6991 - val_iou_metric: 0.4275
- val_loss: 527.8857 - val_mse_metric: 218.2142 - learning_rate: 4.0625e-07
Epoch 71/150
1002/1006 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - class_accuracy: 0.9919 - iou_metric: 0.8188
- loss: 7.2018 - mse_metric: 6.6233
Epoch 71: ReduceLROnPlateau reducing learning rate to 2.0312499771080184e-07.
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - class_accuracy: 0.9919 - iou_metric: 0.8188
- loss: 7.2075 - mse_metric: 6.6281 - val_class_accuracy: 0.6968 - val_iou_metric: 0.4276
- val_loss: 528.0751 - val_mse_metric: 218.1803 - learning_rate: 4.0625e-07
Epoch 72/150
1006/1006 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - class_accuracy: 0.9940 - iou_metric: 0.7890
- loss: 9.6437 - mse_metric: 8.3651 - val_class_accuracy: 0.6991 - val_iou_metric: 0.4279
- val_loss: 528.0644 - val_mse_metric: 218.2196 - learning_rate: 2.0312e-07
Epoch 72: early stopping
Restoring model weights from the end of the best epoch: 62.
```

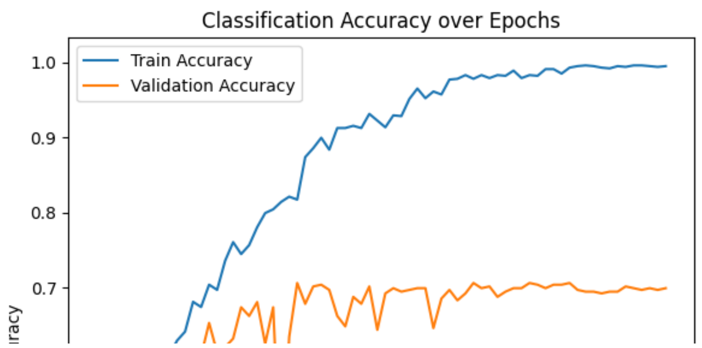
In [16]:

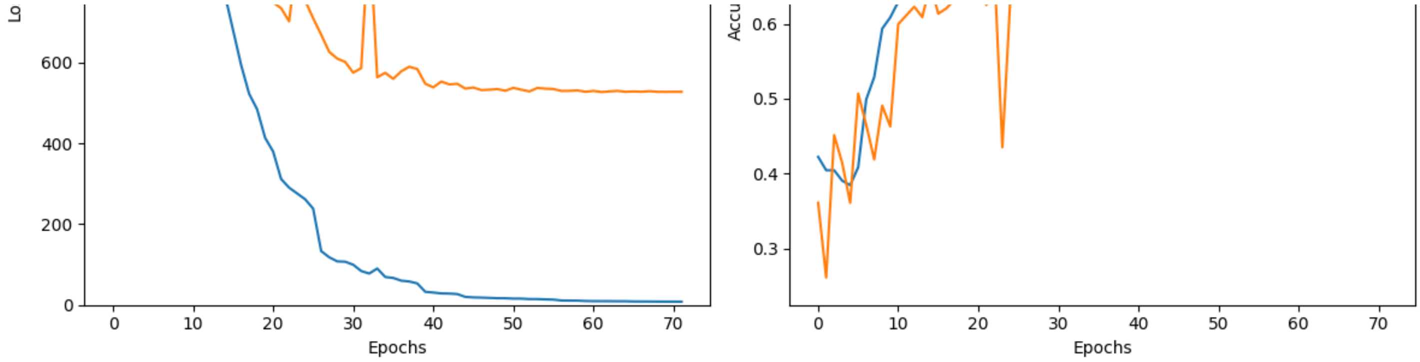
```python
# Vizualizacija procesa treniranja

# Get the training and validation metrics from the history object
train_iou = history.history['iou_metric']
val_iou = history.history['val_iou_metric']

# Plot the IoU over epochs
plt.figure(figsize=(12, 6))
plt.plot(train_iou, label='Train IoU')
plt.plot(val_iou, label='Validation IoU')
plt.title('IoU (Intersection over Union) over Epochs')
plt.xlabel('Epochs')
plt.ylabel('IoU')
plt.legend()
plt.show()


train_loss = history.history['loss']
val_loss = history.history['val_loss']

train_class_accuracy = history.history['class_accuracy']
val_class_accuracy = history.history['val_class_accuracy']

train_bbox_mse = history.history['mse_metric']
val_bbox_mse = history.history['val_mse_metric']

# Plot the loss over epochs
plt.figure(figsize=(12, 6))
```

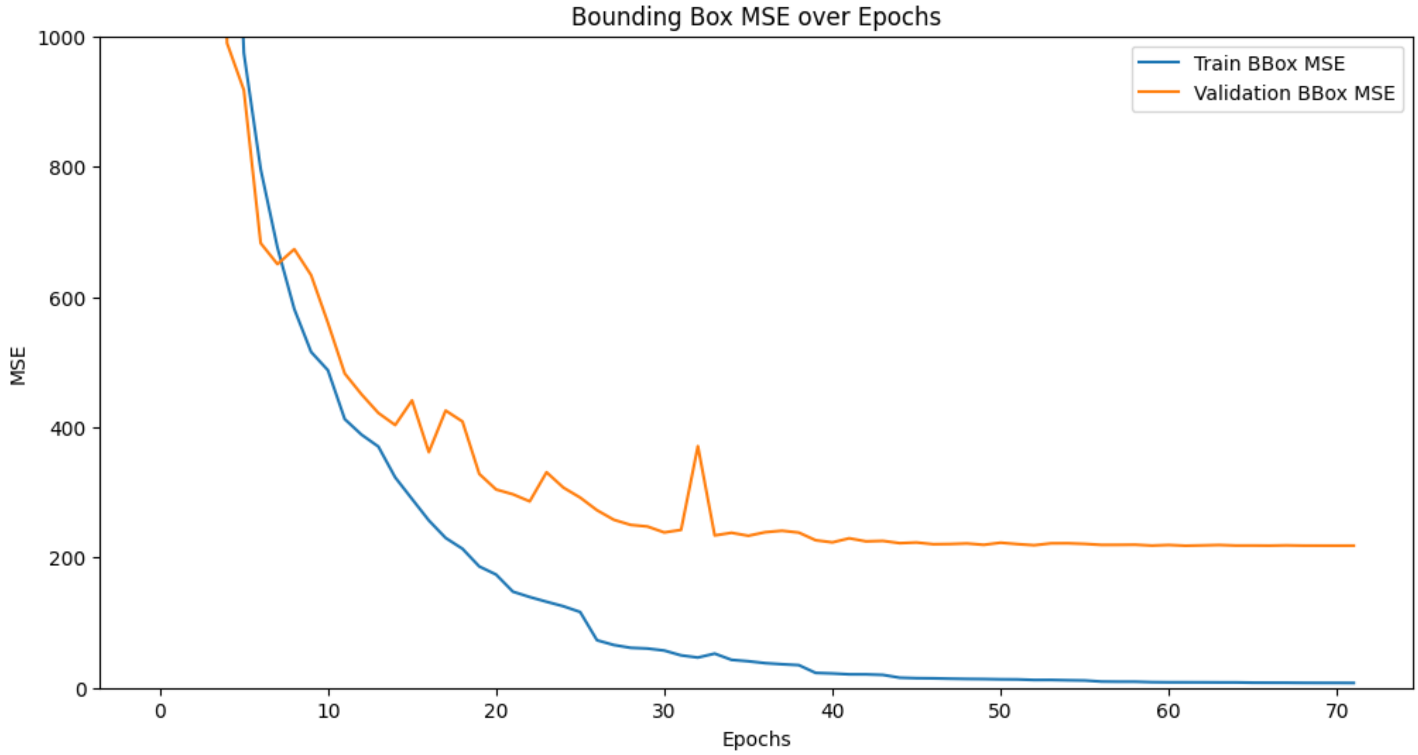
```
# Loss plot
plt.subplot(1, 2, 1)
plt.plot(train_loss, label='Train Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Loss over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.ylim(0, 1500)
plt.legend()

# Accuracy plot
plt.subplot(1, 2, 2)
plt.plot(train_class_accuracy, label='Train Accuracy')
plt.plot(val_class_accuracy, label='Validation Accuracy')
plt.title('Classification Accuracy over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()

# Optionally, you can also plot the bounding box MSE
plt.figure(figsize=(12, 6))
plt.plot(train_bbox_mse, label='Train BBox MSE')
plt.plot(val_bbox_mse, label='Validation BBox MSE')
plt.title('Bounding Box MSE over Epochs')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.ylim(0, 1000)
plt.show()

model.training = False
```











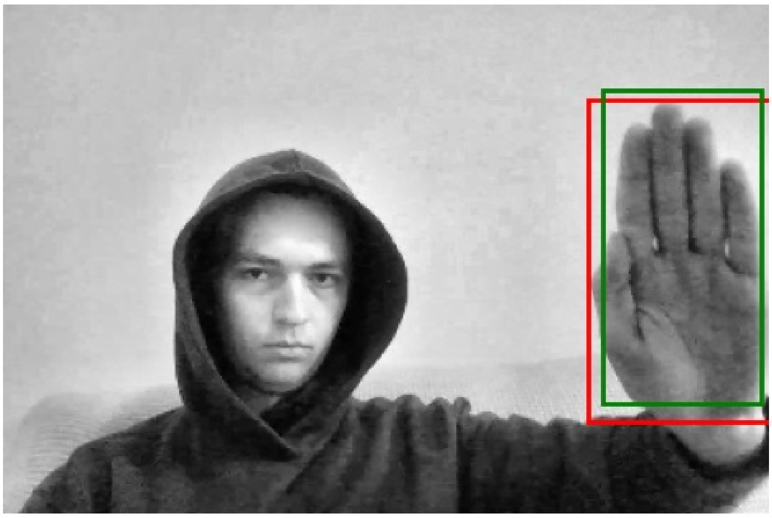
In [51]:

```
# Praktično testiranje kako radi trenirani model, predikcije su prave predikcije (našeg) treniranog modela
# Parametar which_one služi da se odabere želiš li testirati na treniranim podacima, ili na dosad neviđenim test podacima


which_one = "test" # "train" or "test"

if which_one == "test":
    print("TEST")

    # Select 8 random indices from the filtered indices
    valid_indices = [
        i for i, label in enumerate(y_test[..., 0:4])
        if np.argmax(label[0:4]) == 0 or np.argmax(label[0:4]) == 2
    ]
    print("valid_indices num:", len(valid_indices))
    random_indices = random.sample(valid_indices, 8)

    # Loop through the random samples
    for index in random_indices:
        # Get the image, true bounding box, and true class label
        image = X_test[index, ..., 0]  # Get grayscale image (remove channel dimension)
        true_bbox = y_test[index, 4:]  # Get bounding box (without label)
        true_class = y_test[index, 0:4]  # First element is the label (Palm or No Palm)

        # Get predicted bounding box and class from your model
        pred_output = model.predict(image[np.newaxis, ..., np.newaxis])  # Add batch and channel dimensions

        # Separate predicted class and bounding box from the model output
```

```
        pred_class = pred_output[0, 0:4]  # First element is the predicted class (Palm or No Palm)
        pred_bbox = pred_output[0, 4:]  # The remaining four elements are the predicted bounding box

        # Plot the image with both true and predicted bounding boxes
        plot_image_with_bbox(image, true_bbox, true_class, pred_bbox, pred_class, bbox_format=bbox_format)

elif which_one == "train":
    print("TRAIN")

    # Select 8 random indices from the filtered indices
    valid_indices = [
        i for i, label in enumerate(y_test[..., 0:4])
        if np.argmax(label[0:4]) == 0 or np.argmax(label[0:4]) == 2
    ]
    print("valid_indices num:", len(valid_indices))
    random_indices = random.sample(valid_indices, 8)

    # Loop through the random samples
    for index in random_indices:
        # Get the image, true bounding box, and true class label
        image = X_train[index, ..., 0]  # Get grayscale image (remove channel dimension)
        true_bbox = y_train[index, 4:]  # Get bounding box (without label)
        true_class = y_train[index, 0:4]  # First element is the label (Palm or No Palm)

        # Get predicted bounding box and class from your model
        pred_output = model.predict(image[np.newaxis, ..., np.newaxis])  # Add batch and channel dimensions

        # Separate predicted class and bounding box from the model output
        pred_class = pred_output[0, 0:4]  # First element is the predicted class (Palm or No Palm)
        pred_bbox = pred_output[0, 4:]  # The remaining four elements are the predicted bounding box

        # Plot the image with both true and predicted bounding boxes
        plot_image_with_bbox(image, true_bbox, true_class, pred_bbox, pred_class, bbox_format=bbox_format)
```

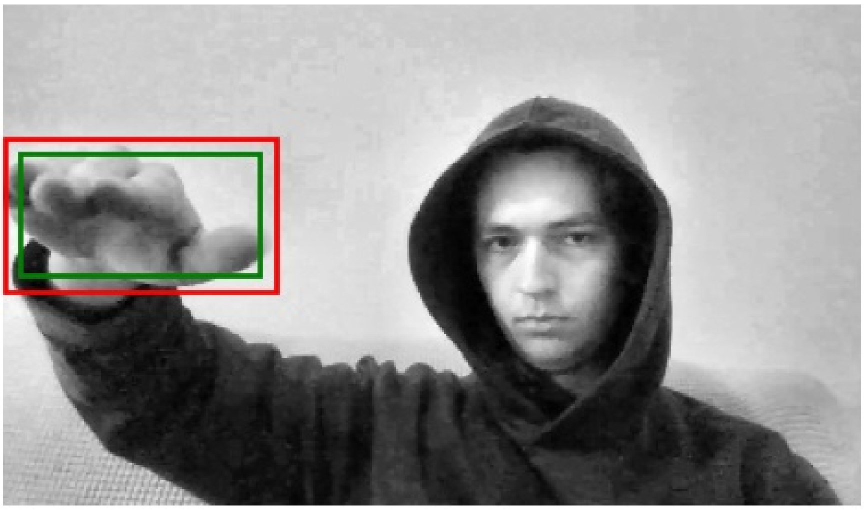
```
TEST
valid_indices num: 289
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
IoU: tf.Tensor(0.79221106, shape=(), dtype=float32)
```

True Class: [0. 0. 1. 0.] [PAUSE], Predicted Class: [5.035e-02 2.801e-06 9.497e-01 0.000e+00][PAUSE]



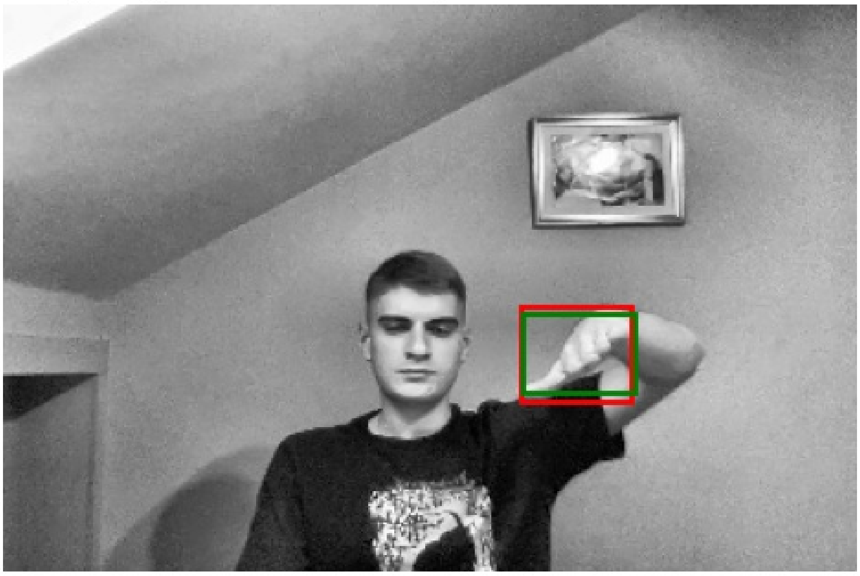
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
IoU: tf.Tensor(0.70008194, shape=(), dtype=float32)
```

True Class: [1. 0. 0. 0.] [FLY], Predicted Class: [1.0e+00 0.0e+00 4.5e-06 0.0e+00][FLY]



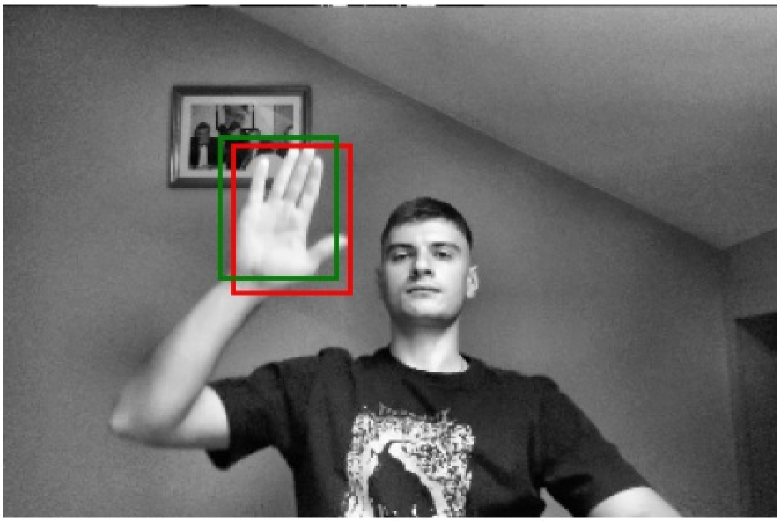
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
IoU: tf.Tensor(0.7675164, shape=(), dtype=float32)
```

True Class: [1. 0. 0. 0.] [FLY], Predicted Class: [1.00e+00 0.00e+00 5.96e-08 9.46e-05][FLY]



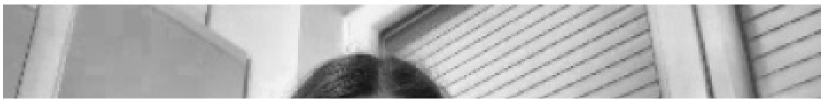
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
IoU: tf.Tensor(0.6748983, shape=(), dtype=float32)
```

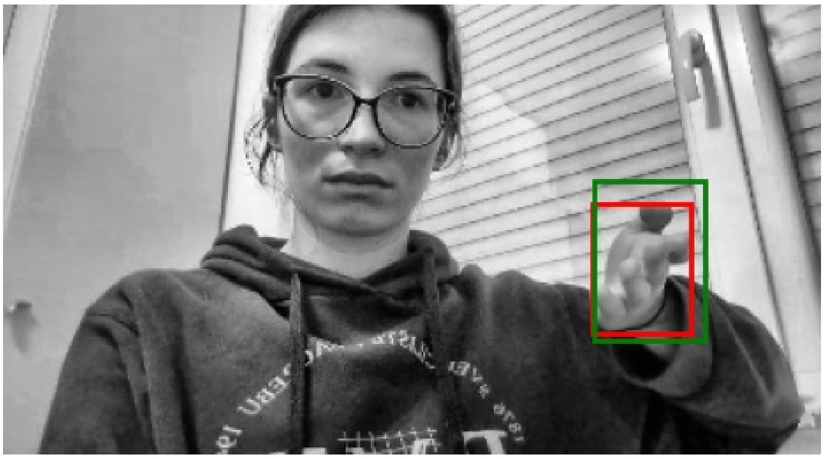
True Class: [0. 0. 1. 0.] [PAUSE], Predicted Class: [1.013e-06 1.994e-02 9.800e-01 5.901e-06][PAUSE]



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
IoU: tf.Tensor(0.6881364, shape=(), dtype=float32)
```

True Class: [1. 0. 0. 0.] [FLY], Predicted Class: [9.990e-01 0.000e+00 5.960e-08 5.927e-04][FLY]



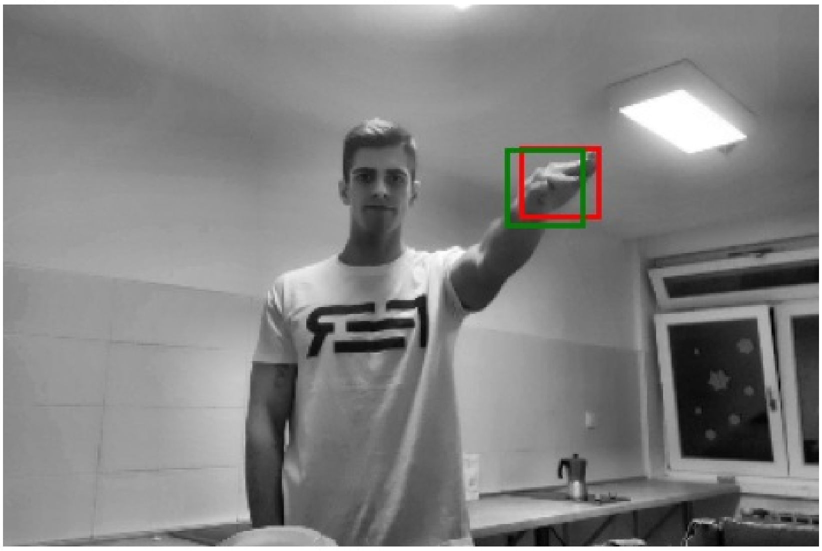


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
IoU: tf.Tensor(0.8511985, shape=(), dtype=float32)
```

True Class: [0. 0. 1. 0.] [PAUSE], Predicted Class: [0.03906 0.004814 0.822   0.1342][PAUSE]

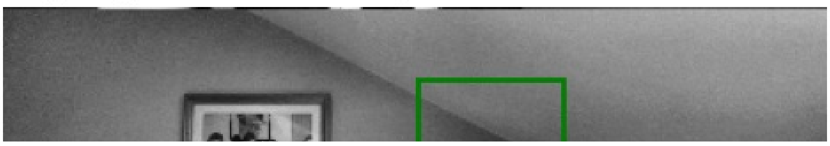
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
IoU: tf.Tensor(0.593128, shape=(), dtype=float32)
```

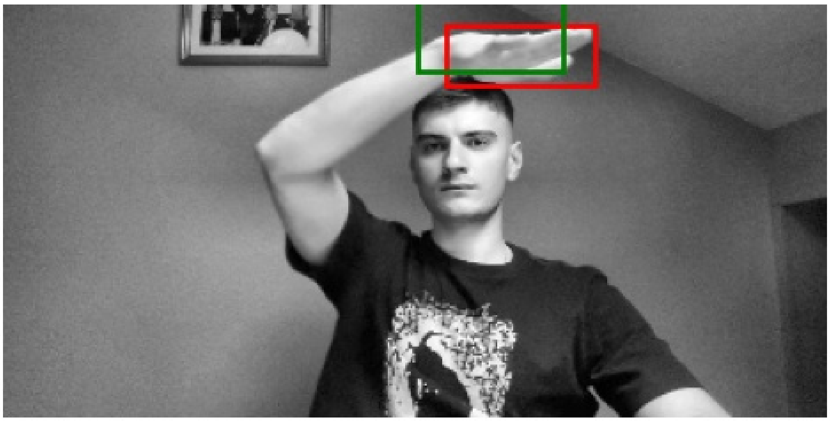
True Class: [1. 0. 0. 0.] [FLY], Predicted Class: [9.883e-01 0.000e+00 1.138e-05 1.169e-02][FLY]



```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
IoU: tf.Tensor(0.23467761, shape=(), dtype=float32)
```

True Class: [1. 0. 0. 0.] [FLY], Predicted Class: [6.85e-06 4.77e-07 1.00e+00 0.00e+00][PAUSE]



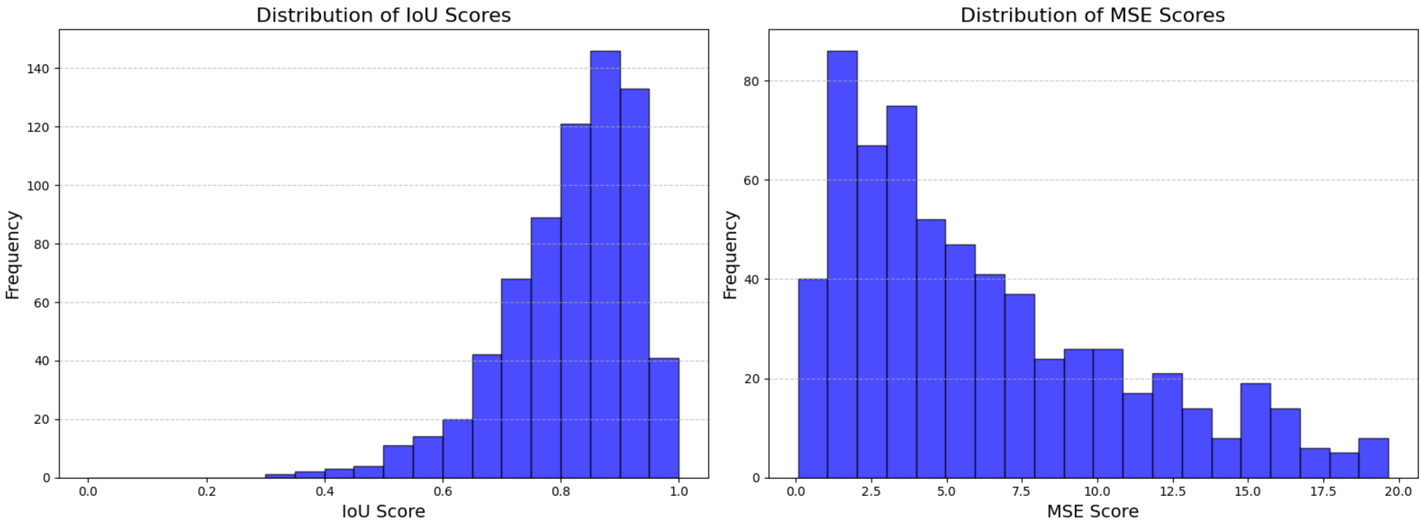


In [25]:

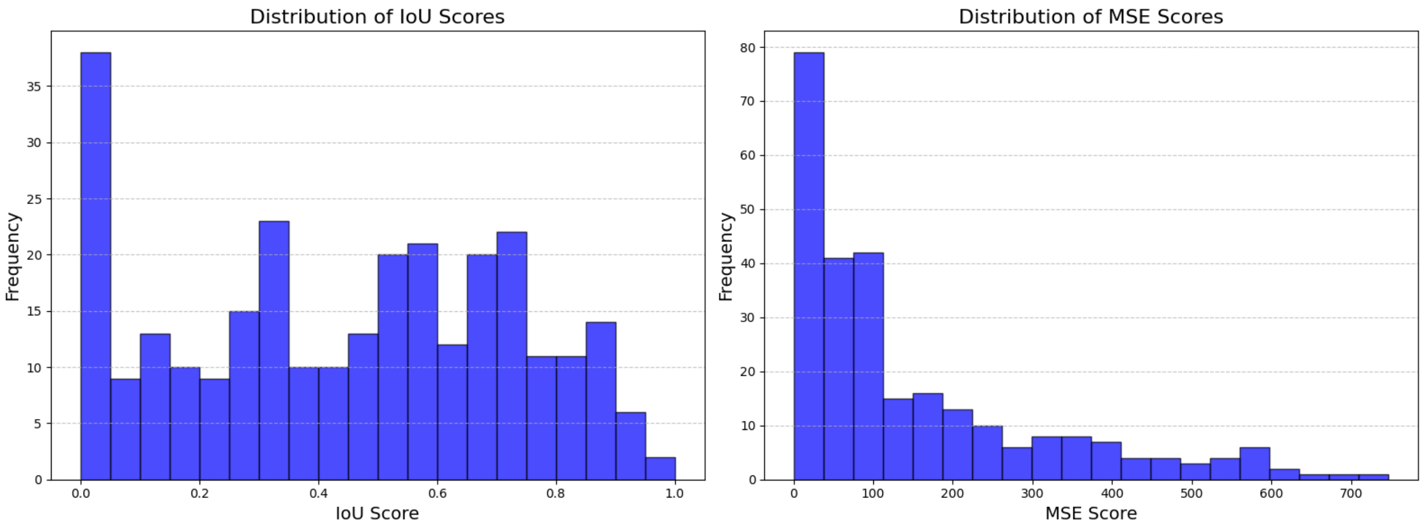
```
# Detaljna evaluacija IOU i MSE metrike (i TRAIN i TEST data)

def calculate_iou_mse_scores(dataset, model, class_filter, bbox_format_rn=None):
    """
    Calculate MSE scores for samples in the dataset where the class is in class_filter.

    Args:
    - dataset: Tuple (X, y) where X contains images and y contains labels and bounding bo
xes.
    - model: The trained model to make predictions.
    - class_filter: List of class labels to include in the MSE calculation (e.g., [0, 2])
.
    - bbox_format_rn: Bounding box format, e.g., 'center' or 'coco'. Default is None.

    Returns:
    - mse_scores: List of MSE scores for the filtered dataset.
    """
    X, y = dataset
    iou_just_0_2 = []
    mse_just_0_2 = []

    for i in range(len(X)):
        true_class = y[i, 0:4]  # Get the class label
        true_class = np.argmax(true_class)
        if true_class in class_filter:

            # Get true bounding box
            true_bbox = y[i, 4:]  # Bounding box (without label)

            # Predict bounding box and class
            image = X[i, ..., 0][np.newaxis, ..., np.newaxis]  # Prepare image for predi
ction
            pred_output = model.predict(image, verbose=0)
            pred_bbox = pred_output[0, 4:]  # Extract predicted bounding box

            # Calculate MSE (Mean Squared Error)
            mse = np.mean((true_bbox - pred_bbox) ** 2)  # MSE between true and predicte
d bounding box
            if (mse > 1e6):
              print("pred output:", pred_output)
              print("mse0", mse)
              print()

            mse_just_0_2.append(float(mse))

            # Calculate IoU
            iou = 1 - iou_loss(true_bbox[np.newaxis, ...], pred_bbox[np.newaxis, ...], b
box_format_rn)
            iou_just_0_2.append(float(iou))

    return iou_just_0_2, mse_just_0_2


def plot_distributions(iou_scores, mse_scores):
```

```
    # Calculate min and max for MSE x-axis limits
    min_mse = min(mse_scores)
    max_mse = max(mse_scores)

    # Create subplots
    fig, axes = plt.subplots(1, 2, figsize=(16, 6))  # 1 row, 2 columns

    # IoU distribution plot
    axes[0].hist(iou_scores, bins=20, range=(0, 1), color='blue', alpha=0.7, edgecolor='
black')
    axes[0].set_title('Distribution of IoU Scores', fontsize=16)
    axes[0].set_xlabel('IoU Score', fontsize=14)
    axes[0].set_ylabel('Frequency', fontsize=14)
    axes[0].grid(axis='y', linestyle='--', alpha=0.7)

    # MSE distribution plot
    axes[1].hist(mse_scores, bins=20, range=(min_mse, max_mse*0.3), color='blue', alpha=
0.7, edgecolor='black')
    axes[1].set_title('Distribution of MSE Scores', fontsize=16)
    axes[1].set_xlabel('MSE Score', fontsize=14)
    axes[1].set_ylabel('Frequency', fontsize=14)
    axes[1].grid(axis='y', linestyle='--', alpha=0.7)

    # Adjust layout and show the plots
    plt.tight_layout()
    plt.show()




for which_one in ["train", "test"]:
    print("\n\n")
    print("-----", which_one,  "-----")

    if which_one == "test":
        dataset = (X_test, y_test)
    elif which_one == "train":
        dataset = (X_train, y_train)

    class_filter = [0, 2]  # Include only classes 0 and 2
    iou_just_0_2, mse_just_0_2 = calculate_iou_mse_scores(dataset, model, class_filter,
bbox_format_rn=bbox_format)


    # IOU
    avg_iou = np.mean(iou_just_0_2)
    std_iou = np.std(iou_just_0_2)  # Standard deviation of IoU
    total_samples = len(iou_just_0_2)
    print("IoU:")
    print("Total analyzed samples:", total_samples)
    print(f"Average IoU for JUST classes {class_filter} for dataset {which_one}: {avg_iou
:.4f}")
    print(f"Standard Deviation of IoU for JUST classes {class_filter}: {std_iou:.4f}")

    # MSE
    avg_mse = np.mean(mse_just_0_2)
    std_mse = np.std(mse_just_0_2)  # Standard deviation of MSE
    print("\nMSE:")
    print(f"Average MSE for JUST classes {class_filter} for dataset {which_one}: {avg_mse
:.4f}")
    print(f"Standard Deviation of MSE for JUST classes {class_filter}: {std_mse:.4f}")
    print()




    # Plot IoU and MSE distributions side by side
    plot_distributions(iou_just_0_2, mse_just_0_2)
```

```
----- train -----
IoU:
Total analyzed samples: 695
Average IoU for JUST classes [0, 2] for dataset train: 0.8166
Standard Deviation of IoU for JUST classes [0, 2]: 0.1108

MSE:
Average MSE for JUST classes [0, 2] for dataset train: 8.2508
Standard Deviation of MSE for JUST classes [0, 2]: 8.3508
```




```
----- test -----
IoU:
Total analyzed samples: 289
Average IoU for JUST classes [0, 2] for dataset test: 0.4460
Standard Deviation of IoU for JUST classes [0, 2]: 0.2803

MSE:
Average MSE for JUST classes [0, 2] for dataset test: 228.2007
Standard Deviation of MSE for JUST classes [0, 2]: 351.7076
```

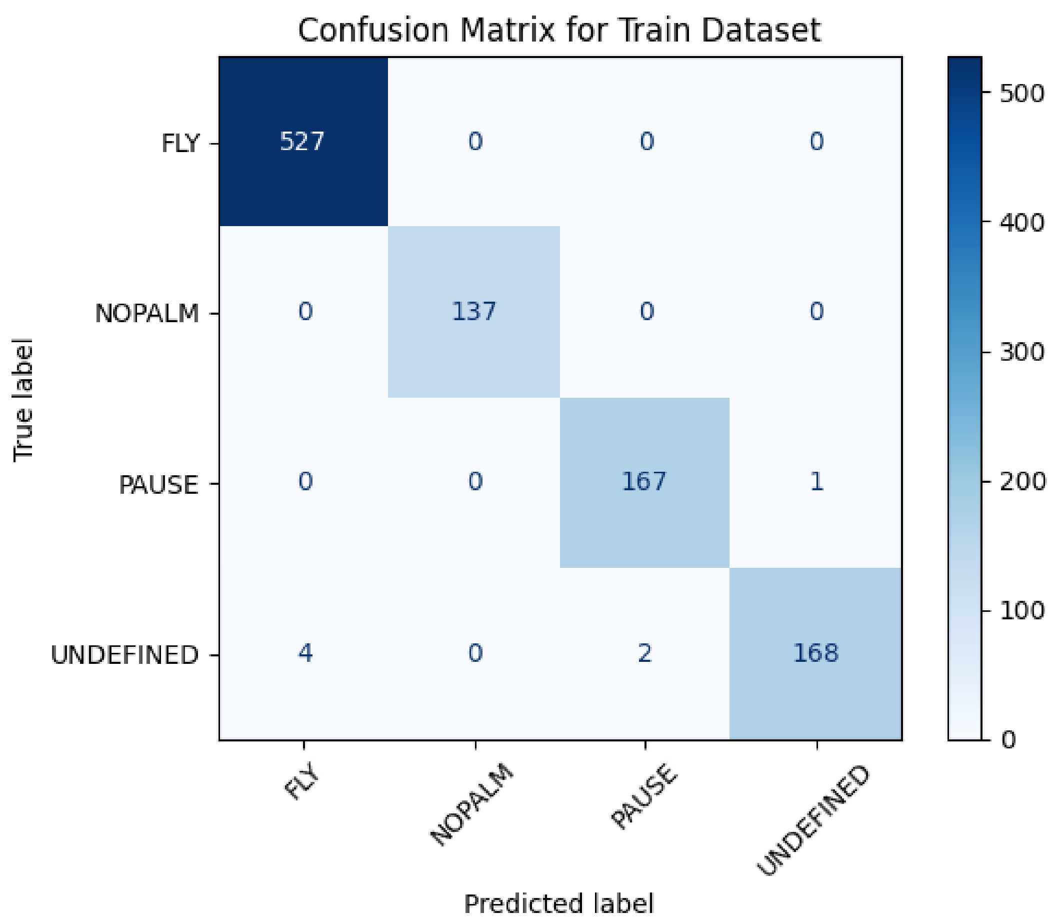


In [52]:

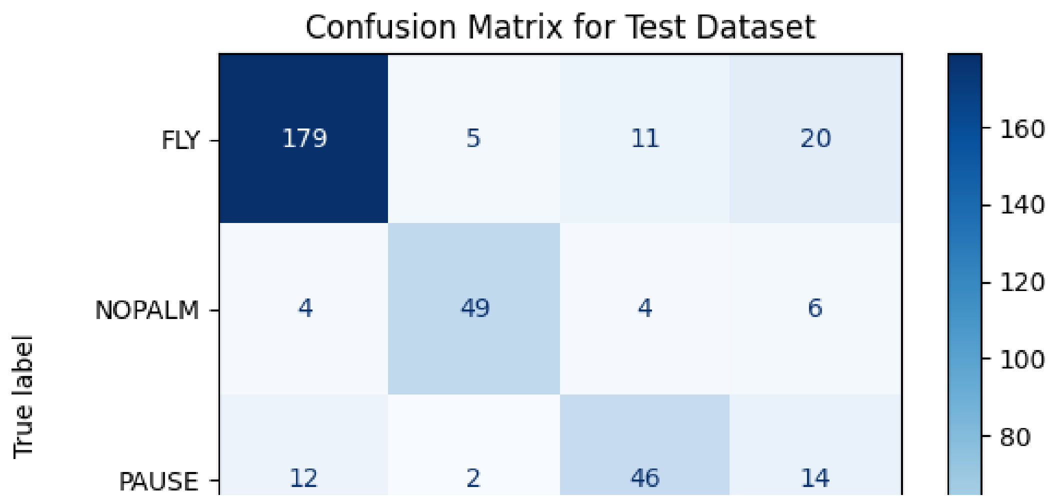
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_repo
rt, accuracy_score

def process_dataset(dataset, model):
    """
    Process the dataset to get true and predicted classes.
    Args:
        dataset: Tuple of (X, y), where X is the input data and y contains the labels.
```

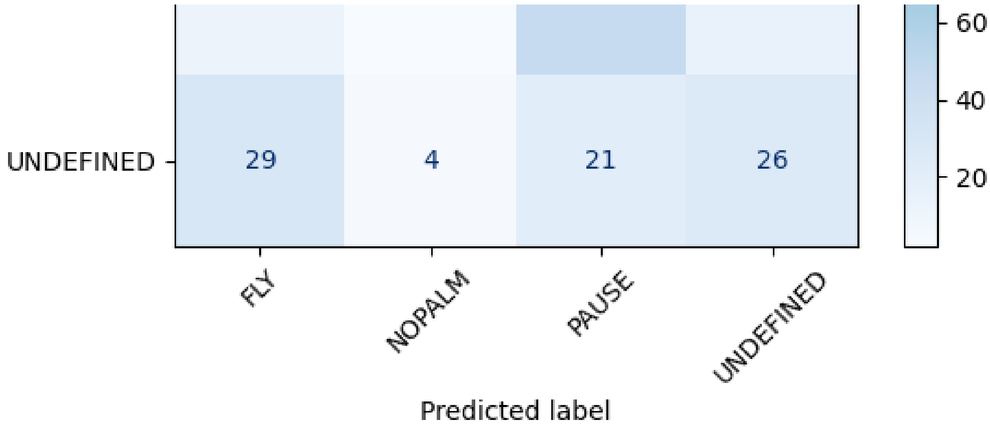
```python
        model: Trained model to predict outputs.
    Returns:
        all_true_classes: Array of true class indices.
        all_pred_classes: Array of predicted class indices.
    """
    X, y = dataset
    all_true_classes = []
    all_pred_classes = []

    for i in range(len(X)):
        true_class = y[i, 0:4]  # Get the class label
        true_class = np.argmax(true_class)
        all_true_classes.append(true_class)

        # Predict bounding box and class
        image = X[i, ..., 0][np.newaxis, ..., np.newaxis]  # Prepare image for predictio
n
        pred_output = model.predict(image, verbose=0)
        pred_class = pred_output[0, 0:4]  # Extract predicted class probabilities

        pred_class = np.argmax(pred_class)
        all_pred_classes.append(pred_class)

    return np.array(all_true_classes), np.array(all_pred_classes)

def plot_confusion_matrix(y_true, y_pred, names_of_class, dataset_name="Dataset"):
    """
    Computes and plots the confusion matrix with class names.
    Args:
        y_true: True class labels.
        y_pred: Predicted class labels.
        names_of_class: List of class names.
        dataset_name: Name of the dataset for labeling the plot.
    """
    # Compute confusion matrix
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=names_of_class)

    # Plot the confusion matrix
    disp.plot(cmap=plt.cm.Blues, xticks_rotation=45)
    plt.title(f"Confusion Matrix for {dataset_name}")
    plt.show()

def print_metrics(y_true, y_pred, names_of_class, dataset_name="Dataset"):
    """
    Prints additional metrics like classification report and accuracy.
    Args:
        y_true: True class labels.
        y_pred: Predicted class labels.
        names_of_class: List of class names.
        dataset_name: Name of the dataset for labeling metrics.
    """
    print(f"----- Metrics for {dataset_name} -----")
    print(f"Accuracy: {accuracy_score(y_true, y_pred):.4f}")
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred, target_names=names_of_class))

def process_and_plot(data, model, names_of_class, dataset_name="Dataset"):
    """
    Processes the dataset, plots the confusion matrix, and prints additional metrics.
    Args:
        data: Tuple of (X, y), where X is the input data and y contains the labels.
        model: Trained model to predict outputs.
        names_of_class: List of class names.
        dataset_name: Name of the dataset for labeling the outputs.
    """
    # Process the dataset to extract true and predicted labels
    y_true, y_pred = process_dataset(data, model)

    # Plot confusion matrix
    plot_confusion_matrix(y_true, y_pred, names_of_class, dataset_name)
```

```
    # Print additional metrics
    print_metrics(y_true, y_pred, names_of_class, dataset_name)

# Example usage
names_of_class = ["FLY", "NOPALM", "PAUSE", "UNDEFINED"]
process_and_plot((X_train, y_train), model, names_of_class, dataset_name="Train Dataset"
)
process_and_plot((X_test, y_test), model, names_of_class, dataset_name="Test Dataset")
```




```
----- Metrics for Train Dataset -----
Accuracy: 0.9930

Classification Report:
              precision    recall  f1-score   support

         FLY       0.99      1.00      1.00       527
      NOPALM       1.00      1.00      1.00       137
       PAUSE       0.99      0.99      0.99       168
   UNDEFINED       0.99      0.97      0.98       174

    accuracy                           0.99      1006
   macro avg       0.99      0.99      0.99      1006
weighted avg       0.99      0.99      0.99      1006
```






```
----- Metrics for Test Dataset -----
Accuracy: 0.6944

Classification Report:
              precision    recall  f1-score   support

         FLY       0.80      0.83      0.82       215
      NOPALM       0.82      0.78      0.80        63
       PAUSE       0.56      0.62      0.59        74
   UNDEFINED       0.39      0.33      0.36        80

    accuracy                           0.69       432
   macro avg       0.64      0.64      0.64       432
weighted avg       0.69      0.69      0.69       432
```

In [ ]:

```
# calculate mAP?
```

In [ ]:

```
# Save the model, kasnije se može importat
model.save("coco_palm_detection_model_val_0.46_233.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.savin
g.save_model(model)`. This file format is considered legacy. We recommend using instead t
he native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(m
odel, 'my_model.keras')`.
```