Lab_02_output1

June 4, 2025

[22]:
```python
import os
import time
import random
import xml.etree.ElementTree as ET
from glob import glob
from PIL import Image
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras import layers, models, applications
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.models import load_model
```

[2]:
```python
# ================================
# Task 1: Dataset Exploration and Preprocessing
# ================================

# ================================
# Step 1: Extract Manually Downloaded Data
# ================================

# Define paths
data_root = './data'
images_tar_path = os.path.join(data_root, 'images.tar.gz')
annotations_tar_path = os.path.join(data_root, 'annotations.tar.gz')
images_folder = os.path.join(data_root, 'images')
annotations_folder = os.path.join(data_root, 'annotations')
trimaps_folder = os.path.join(annotations_folder, 'trimaps')

# Create data root directory if it doesn't exist
if not os.path.exists(data_root):
    os.makedirs(data_root)

# Extract tar.gz files if necessary
```

```
def extract_if_needed(tar_path, extract_to):
    if not os.path.exists(extract_to):
        print(f"Extracting {tar_path} to {extract_to}...")
        with tarfile.open(tar_path, 'r:gz') as tar:
            tar.extractall(path=data_root)
        print("Done.")
    else:
        print(f"{extract_to} already exists. Skipping extraction.")

# Ensure tar.gz files exist
assert os.path.exists(images_tar_path), f"{images_tar_path} not found"
assert os.path.exists(annotations_tar_path), f"{annotations_tar_path} not found"

# Extract
extract_if_needed(images_tar_path, images_folder)
extract_if_needed(annotations_tar_path, annotations_folder)
```

```
./data\images already exists. Skipping extraction.
./data\annotations already exists. Skipping extraction.
```

[3]:

```
# ================================
# Step 2: Load Filepaths and Split Dataset
# ================================

# List all image filenames (without extension)
image_paths = sorted(glob(os.path.join(images_folder, '*.jpg')))
all_ids = [os.path.splitext(os.path.basename(p))[0] for p in image_paths]

# Filter out IDs without corresponding XML
valid_ids = [id_ for id_ in all_ids if os.path.exists(os.path.
 ↪join(annotations_folder, 'xmls', id_ + '.xml'))]

# Shuffle and split
random.seed(42)
random.shuffle(valid_ids)
val_ratio = 0.2  # 20% for validation
split_idx = int(len(valid_ids) * (1 - val_ratio))
train_ids = valid_ids[:split_idx]
val_ids = valid_ids[split_idx:]

# Print dataset sizes
print("Total valid images with XML:", len(valid_ids))
print("Train:", len(train_ids))
print("Val:", len(val_ids))
```

```
Total valid images with XML: 3686
Train: 2948
Val: 738
```

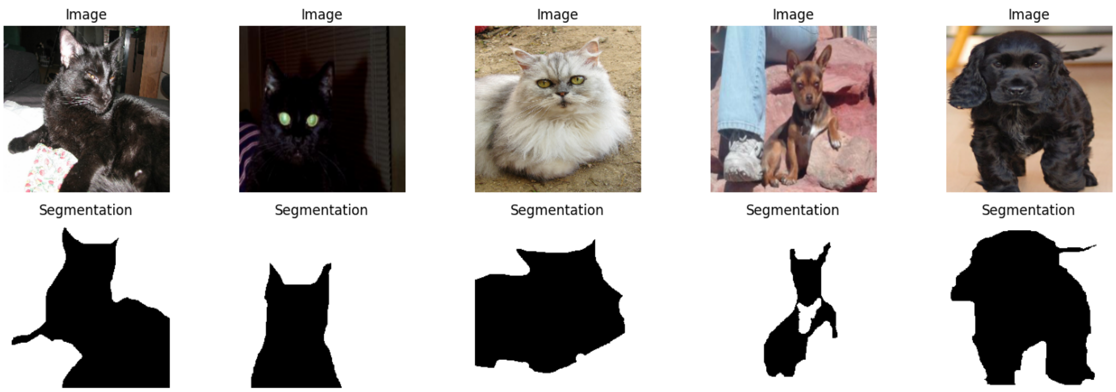
[4]:
```python
# ================================
# Step 3: Define Preprocessing Function
# ================================

IMG_SIZE = (224, 224)

def load_and_preprocess(id_):
    # Load image
    img_path = os.path.join(images_folder, id_ + '.jpg')
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, IMG_SIZE)
    img = tf.cast(img, tf.float32) / 255.0

    # Load segmentation mask
    mask_path = os.path.join(trimaps_folder, id_ + '.png')
    mask = tf.io.read_file(mask_path)
    mask = tf.image.decode_png(mask, channels=1)
    mask = tf.image.resize(mask, IMG_SIZE, method='nearest')
    mask = tf.cast(mask > 1, tf.float32)  # Convert trimap to binary mask␣
↪(foreground = 1)

    return img, mask
```

[5]:
```python
# ================================
# Step 4: Create tf.data.Datasets
# ================================

def load_and_preprocess_py(id_):
    # Convert bytes tensor to string
    id_str = id_.numpy().decode('utf-8')

    # Load image
    img_path = os.path.join(images_folder, id_str + '.jpg')
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, IMG_SIZE)
    img = tf.cast(img, tf.float32) / 255.0

    # Load mask
    mask_path = os.path.join(trimaps_folder, id_str + '.png')
    mask = tf.io.read_file(mask_path)
    mask = tf.image.decode_png(mask, channels=1)
    mask = tf.image.resize(mask, IMG_SIZE, method='nearest')
    mask = tf.cast(mask > 1, tf.float32)

    return img, mask
```

```python
def tf_wrapper(id_):
    img, mask = tf.py_function(
        func=load_and_preprocess_py,
        inp=[id_],
        Tout=[tf.float32, tf.float32]
    )
    img.set_shape([*IMG_SIZE, 3])
    mask.set_shape([*IMG_SIZE, 1])
    return img, mask

def create_dataset(id_list):
    dataset = tf.data.Dataset.from_tensor_slices(id_list)
    dataset = dataset.map(tf_wrapper, num_parallel_calls=tf.data.AUTOTUNE)
    dataset = dataset.batch(32).prefetch(tf.data.AUTOTUNE)
    return dataset

train_ds = create_dataset(train_ids)
val_ds = create_dataset(val_ids)
```

[6]:

```python
# ===============================
# Step 5: Visualization
# ===============================

def visualize_batch(dataset, n=5):
    for images, masks in dataset.take(1):
        plt.figure(figsize=(15, 5))
        for i in range(n):
            ax1 = plt.subplot(2, n, i + 1)
            plt.imshow(images[i])
            plt.title("Image")
            plt.axis("off")

            ax2 = plt.subplot(2, n, n + i + 1)
            plt.imshow(masks[i, :, :, 0], cmap='gray')
            plt.title("Segmentation")
            plt.axis("off")
        plt.tight_layout()
        plt.show()

visualize_batch(train_ds)
```



[7]:
```
# ================================
# Task 2: Object Detection
# ================================

# ================================
# Step 1: Read bounding boxes from XML files
# ================================
# Parse XML annotation to extract bounding box in [cx, cy, w, h] format, normalized to image size.

def parse_bounding_box(xml_path, image_width, image_height):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    obj = root.find("object")
    bbox = obj.find("bndbox")

    xmin = int(bbox.find("xmin").text)
    ymin = int(bbox.find("ymin").text)
    xmax = int(bbox.find("xmax").text)
    ymax = int(bbox.find("ymax").text)

    cx = (xmin + xmax) / 2 / image_width
    cy = (ymin + ymax) / 2 / image_height
    w = (xmax - xmin) / image_width
    h = (ymax - ymin) / image_height

    return [cx, cy, w, h]
```

[8]:
```
# ================================
# Step 2: Load and preprocess image with bounding box
# ================================
# Read image and its corresponding bounding box, resize image and normalize bbox coordinates.
```

```python
def load_and_preprocess_detection_py(id_):
    try:
        id_str = id_.numpy().decode('utf-8')

        img_path = os.path.join(images_folder, id_str + '.jpg')
        xml_path = os.path.join(annotations_folder, 'xmls', id_str + '.xml')

        if not os.path.exists(img_path):
            raise FileNotFoundError(f"Image file not found: {img_path}")
        if not os.path.exists(xml_path):
            raise FileNotFoundError(f"XML file not found: {xml_path}")

        # Use PIL to get original image size
        with Image.open(img_path) as img_pil:
            orig_width, orig_height = img_pil.size

        # Load image with TensorFlow
        img_raw = tf.io.read_file(img_path)
        img = tf.image.decode_jpeg(img_raw, channels=3)
        img = tf.image.resize(img, IMG_SIZE)
        img = tf.cast(img, tf.float32) / 255.0

        # Parse bbox based on original image size
        bbox = parse_bounding_box(xml_path, orig_width, orig_height)
        bbox = tf.convert_to_tensor(bbox, dtype=tf.float32)

        return img, bbox

    except Exception as e:
        print(f"Error processing ID: {id_str}")
        raise e
```

[9]:
```python
# ===============================
# Step 3: Create tf.data.Dataset for object detection
# ===============================
# Build a dataset pipeline using tf.py_function to map image IDs to image and␣
 ↪bbox pairs.

def tf_wrapper_detection(id_):
    img, bbox = tf.py_function(
        func=load_and_preprocess_detection_py,
        inp=[id_],
        Tout=[tf.float32, tf.float32]
    )
    img.set_shape([*IMG_SIZE, 3])
    bbox.set_shape([4])
```

```
    return img, bbox

def create_detection_dataset(id_list):
    dataset = tf.data.Dataset.from_tensor_slices(id_list)
    dataset = dataset.map(tf_wrapper_detection, num_parallel_calls=tf.data.
↪AUTOTUNE)
    dataset = dataset.batch(32).prefetch(tf.data.AUTOTUNE)
    return dataset

train_det_ds = create_detection_dataset(train_ids)
val_det_ds = create_detection_dataset(val_ids)
```

[10]:
```
# ===============================
# Step 4: Build detection model with backbone
# ===============================
# Build a CNN model with a backbone (e.g., MobileNetV2) and a dense regression␣
↪head for bounding boxes.

def build_detection_model(pretrained=True):
    base_model = applications.MobileNetV2(
        input_shape=(*IMG_SIZE, 3),
        include_top=False,
        weights='imagenet' if pretrained else None
    )
    base_model.trainable = True

    x = layers.GlobalAveragePooling2D()(base_model.output)
    x = layers.Dense(128, activation='relu')(x)
    bbox_output = layers.Dense(4, activation='sigmoid')(x)

    model = models.Model(inputs=base_model.input, outputs=bbox_output)
    return model
```

[ ]:
```
# ===============================
# Step 4.1: Show sample shape
# ===============================

for img, bbox in train_det_ds.take(1):
    print("Sample shape:", img.shape, bbox.shape)
```

```
Sample shape: (32, 224, 224, 3) (32, 4)
```

[ ]:
```
# ===============================
# Step 5: Train the object detection model
# ===============================
# Compile and train the model with Mean Squared Error loss and early stopping␣
↪callbacks.
```

```python
model = build_detection_model(pretrained=True)

model.compile(
    optimizer=Adam(learning_rate=1e-4),
    loss=MeanSquaredError(),
    metrics=['mae']
)

callbacks = [
    EarlyStopping(patience=5, restore_best_weights=True),
    ReduceLROnPlateau(patience=3, factor=0.5)
]

history = model.fit(
    train_det_ds,
    validation_data=val_det_ds,
    epochs=50,
    callbacks=callbacks
)
```

```
Epoch 1/50
93/93 [==============================] - 22s 174ms/step - loss: 0.0177 - mae:
0.1022 - val_loss: 0.0315 - val_mae: 0.1407 - lr: 1.0000e-04
Epoch 2/50
93/93 [==============================] - 16s 168ms/step - loss: 0.0048 - mae:
0.0534 - val_loss: 0.0244 - val_mae: 0.1236 - lr: 1.0000e-04
Epoch 3/50
93/93 [==============================] - 15s 166ms/step - loss: 0.0017 - mae:
0.0317 - val_loss: 0.0200 - val_mae: 0.1108 - lr: 1.0000e-04
Epoch 4/50
93/93 [==============================] - 16s 171ms/step - loss: 8.5466e-04 -
mae: 0.0226 - val_loss: 0.0181 - val_mae: 0.1047 - lr: 1.0000e-04
Epoch 5/50
93/93 [==============================] - 15s 165ms/step - loss: 8.6575e-04 -
mae: 0.0229 - val_loss: 0.0142 - val_mae: 0.0926 - lr: 1.0000e-04
Epoch 6/50
93/93 [==============================] - 15s 163ms/step - loss: 0.0015 - mae:
0.0301 - val_loss: 0.0151 - val_mae: 0.0963 - lr: 1.0000e-04
Epoch 7/50
93/93 [==============================] - 15s 164ms/step - loss: 0.0024 - mae:
0.0386 - val_loss: 0.0147 - val_mae: 0.0948 - lr: 1.0000e-04
Epoch 8/50
93/93 [==============================] - 15s 164ms/step - loss: 0.0028 - mae:
0.0423 - val_loss: 0.0115 - val_mae: 0.0835 - lr: 1.0000e-04
Epoch 9/50
93/93 [==============================] - 16s 167ms/step - loss: 0.0023 - mae:
0.0385 - val_loss: 0.0104 - val_mae: 0.0803 - lr: 1.0000e-04
```

```
Epoch 10/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0016 - mae:
0.0318 - val_loss: 0.0104 - val_mae: 0.0798 - lr: 1.0000e-04
Epoch 11/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0011 - mae:
0.0265 - val_loss: 0.0101 - val_mae: 0.0786 - lr: 1.0000e-04
Epoch 12/50
93/93 [==============================] - 16s 167ms/step - loss: 9.0587e-04 -
mae: 0.0238 - val_loss: 0.0097 - val_mae: 0.0762 - lr: 1.0000e-04
Epoch 13/50
93/93 [==============================] - 16s 167ms/step - loss: 8.7284e-04 -
mae: 0.0227 - val_loss: 0.0095 - val_mae: 0.0748 - lr: 1.0000e-04
Epoch 14/50
93/93 [==============================] - 16s 177ms/step - loss: 9.8645e-04 -
mae: 0.0244 - val_loss: 0.0093 - val_mae: 0.0743 - lr: 1.0000e-04
Epoch 15/50
93/93 [==============================] - 16s 169ms/step - loss: 9.6446e-04 -
mae: 0.0242 - val_loss: 0.0093 - val_mae: 0.0738 - lr: 1.0000e-04
Epoch 16/50
93/93 [==============================] - 16s 167ms/step - loss: 9.5756e-04 -
mae: 0.0243 - val_loss: 0.0092 - val_mae: 0.0742 - lr: 1.0000e-04
Epoch 17/50
93/93 [==============================] - 16s 167ms/step - loss: 0.0011 - mae:
0.0264 - val_loss: 0.0090 - val_mae: 0.0731 - lr: 1.0000e-04
Epoch 18/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0012 - mae:
0.0280 - val_loss: 0.0085 - val_mae: 0.0711 - lr: 1.0000e-04
Epoch 19/50
93/93 [==============================] - 16s 177ms/step - loss: 0.0011 - mae:
0.0259 - val_loss: 0.0076 - val_mae: 0.0669 - lr: 1.0000e-04
Epoch 20/50
93/93 [==============================] - 16s 173ms/step - loss: 9.2943e-04 -
mae: 0.0241 - val_loss: 0.0076 - val_mae: 0.0668 - lr: 1.0000e-04
Epoch 21/50
93/93 [==============================] - 16s 170ms/step - loss: 9.8326e-04 -
mae: 0.0242 - val_loss: 0.0073 - val_mae: 0.0652 - lr: 1.0000e-04
Epoch 22/50
93/93 [==============================] - 16s 169ms/step - loss: 9.6216e-04 -
mae: 0.0239 - val_loss: 0.0074 - val_mae: 0.0660 - lr: 1.0000e-04
Epoch 23/50
93/93 [==============================] - 16s 169ms/step - loss: 8.2540e-04 -
mae: 0.0224 - val_loss: 0.0079 - val_mae: 0.0687 - lr: 1.0000e-04
Epoch 24/50
93/93 [==============================] - 16s 177ms/step - loss: 6.7384e-04 -
mae: 0.0203 - val_loss: 0.0071 - val_mae: 0.0643 - lr: 1.0000e-04
Epoch 25/50
93/93 [==============================] - 16s 172ms/step - loss: 5.6069e-04 -
mae: 0.0186 - val_loss: 0.0067 - val_mae: 0.0629 - lr: 1.0000e-04
```

```
Epoch 26/50
93/93 [==============================] - 16s 169ms/step - loss: 4.8712e-04 -
mae: 0.0174 - val_loss: 0.0067 - val_mae: 0.0621 - lr: 1.0000e-04
Epoch 27/50
93/93 [==============================] - 16s 171ms/step - loss: 4.7946e-04 -
mae: 0.0173 - val_loss: 0.0068 - val_mae: 0.0626 - lr: 1.0000e-04
Epoch 28/50
93/93 [==============================] - 16s 173ms/step - loss: 4.5511e-04 -
mae: 0.0168 - val_loss: 0.0067 - val_mae: 0.0624 - lr: 1.0000e-04
Epoch 29/50
93/93 [==============================] - 16s 176ms/step - loss: 5.2424e-04 -
mae: 0.0178 - val_loss: 0.0063 - val_mae: 0.0596 - lr: 5.0000e-05
Epoch 30/50
93/93 [==============================] - 17s 180ms/step - loss: 2.1098e-04 -
mae: 0.0113 - val_loss: 0.0063 - val_mae: 0.0593 - lr: 5.0000e-05
Epoch 31/50
93/93 [==============================] - 17s 179ms/step - loss: 9.8553e-05 -
mae: 0.0077 - val_loss: 0.0062 - val_mae: 0.0590 - lr: 5.0000e-05
Epoch 32/50
93/93 [==============================] - 17s 180ms/step - loss: 5.5914e-05 -
mae: 0.0057 - val_loss: 0.0061 - val_mae: 0.0586 - lr: 5.0000e-05
Epoch 33/50
93/93 [==============================] - 16s 177ms/step - loss: 3.6998e-05 -
mae: 0.0046 - val_loss: 0.0061 - val_mae: 0.0583 - lr: 5.0000e-05
Epoch 34/50
93/93 [==============================] - 16s 177ms/step - loss: 2.8636e-05 -
mae: 0.0040 - val_loss: 0.0060 - val_mae: 0.0581 - lr: 5.0000e-05
Epoch 35/50
93/93 [==============================] - 17s 178ms/step - loss: 2.4968e-05 -
mae: 0.0037 - val_loss: 0.0060 - val_mae: 0.0579 - lr: 5.0000e-05
Epoch 36/50
93/93 [==============================] - 16s 172ms/step - loss: 2.2257e-05 -
mae: 0.0035 - val_loss: 0.0060 - val_mae: 0.0578 - lr: 5.0000e-05
Epoch 37/50
93/93 [==============================] - 16s 176ms/step - loss: 1.9911e-05 -
mae: 0.0033 - val_loss: 0.0059 - val_mae: 0.0578 - lr: 5.0000e-05
Epoch 38/50
93/93 [==============================] - 16s 175ms/step - loss: 3.9477e-05 -
mae: 0.0047 - val_loss: 0.0059 - val_mae: 0.0577 - lr: 2.5000e-05
Epoch 39/50
93/93 [==============================] - 16s 177ms/step - loss: 3.3044e-05 -
mae: 0.0043 - val_loss: 0.0059 - val_mae: 0.0577 - lr: 2.5000e-05
Epoch 40/50
93/93 [==============================] - 16s 175ms/step - loss: 2.5851e-05 -
mae: 0.0038 - val_loss: 0.0059 - val_mae: 0.0578 - lr: 2.5000e-05
Epoch 41/50
93/93 [==============================] - 16s 175ms/step - loss: 2.9603e-05 -
mae: 0.0040 - val_loss: 0.0060 - val_mae: 0.0579 - lr: 1.2500e-05
```

```
Epoch 42/50
93/93 [==============================] - 16s 175ms/step - loss: 2.4977e-05 -
mae: 0.0036 - val_loss: 0.0060 - val_mae: 0.0580 - lr: 1.2500e-05
Epoch 43/50
93/93 [==============================] - 17s 183ms/step - loss: 1.7724e-05 -
mae: 0.0031 - val_loss: 0.0060 - val_mae: 0.0581 - lr: 1.2500e-05
Model saved to 'object_detector.h5'
Training history saved to 'training_log.csv'
```

[13]:

```
# ===============================
# Step 5.1: Save the trained model to HDF5 (.h5) format in models directory
# ===============================

# Create models directory if it doesn't exist
models_dir = "models"
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

model.save("models/object_detector.h5")
print("Model saved to 'models/object_detector.h5'")

# ===============================
# Step 5.2: Save training history to CSV
# ===============================

pd.DataFrame(history.history).to_csv("models/training_log.csv", index=False)
print("Training history saved to 'models/training_log.csv'")
```

```
Model saved to 'models/object_detector.h5'
Training history saved to 'models/training_log.csv'
```

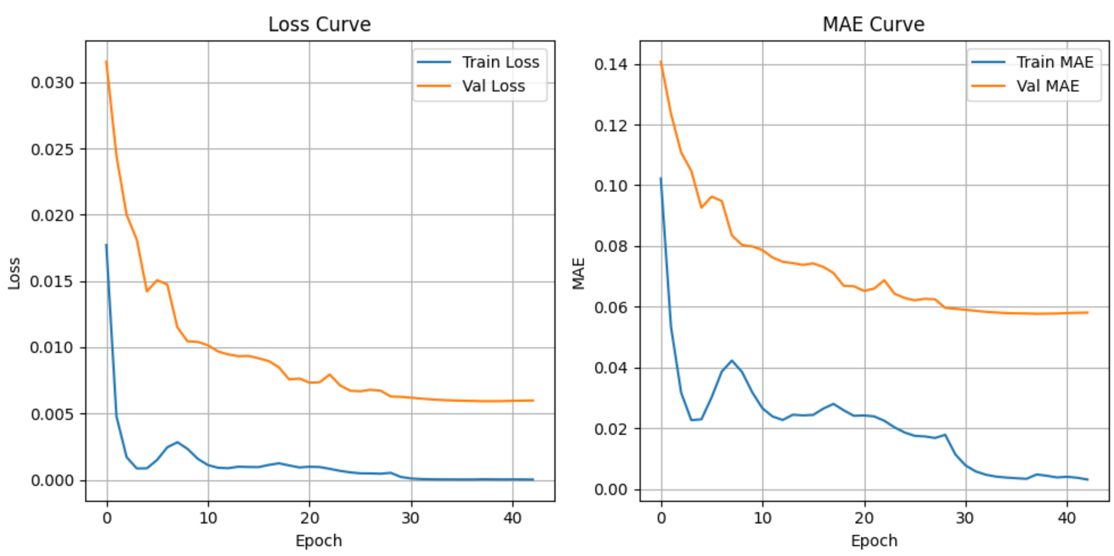
[ ]:

```
# ===============================
# Step 5.3: Load a trained model from HDF5 file (optional)
# ===============================

model = load_model("models/object_detector.h5")
print("Model loaded from 'models/object_detector.h5'")
```

[ ]:

```
# ===============================
# Step 5.4: Plot training and validation curves
# ===============================

plt.figure(figsize=(10, 5))

# Loss
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
```

```python
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Curve")
plt.legend()
plt.grid(True)

# MAE
plt.subplot(1, 2, 2)
plt.plot(history.history['mae'], label='Train MAE')
plt.plot(history.history['val_mae'], label='Val MAE')
plt.xlabel("Epoch")
plt.ylabel("MAE")
plt.title("MAE Curve")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```



[15]:

```python
# ================================
# Step 6: Visualize predicted and true bounding boxes
# ================================
# Draw predicted and true bounding boxes on images for visual evaluation.

def display_bbox(image, true_bbox, pred_bbox):
    h, w = IMG_SIZE

    def denormalize(bbox):
```

```
        cx, cy, bw, bh = bbox
        xmin = int((cx - bw / 2) * w)
        ymin = int((cy - bh / 2) * h)
        xmax = int((cx + bw / 2) * w)
        ymax = int((cy + bh / 2) * h)
        return xmin, ymin, xmax, ymax

    fig, ax = plt.subplots()
    ax.imshow(image)
    xmin, ymin, xmax, ymax = denormalize(true_bbox)
    ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,␣
 ↪edgecolor='green', fill=False, lw=2, label='Ground Truth'))

    xmin, ymin, xmax, ymax = denormalize(pred_bbox)
    ax.add_patch(plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,␣
 ↪edgecolor='red', fill=False, lw=2, label='Prediction'))
    plt.legend()
    plt.show()
```

[ ]:

```
# ===============================
# Step 7: Train and compare pretrained vs non-pretrained models
# ===============================
# Train two versions of the model and compare performance (MAE, loss, training␣
 ↪time, early stopping)

def train_and_evaluate(pretrained=True):
    print(f"Training model with pretrained={pretrained}")
    model = build_detection_model(pretrained=pretrained)
    model.compile(optimizer=Adam(1e-4), loss=MeanSquaredError(),␣
 ↪metrics=['mae'])

    callbacks = [
        EarlyStopping(patience=5, restore_best_weights=True),
        ReduceLROnPlateau(patience=3, factor=0.5)
    ]

    start_time = time.time()
    history = model.fit(
        train_det_ds,
        validation_data=val_det_ds,
        epochs=50,
        callbacks=callbacks,
        verbose=1
    )
    duration = time.time() - start_time

    print(f"Training time: {duration:.2f} seconds")
```

```
    return model, history, duration

# Train both models
model_pretrained, history_pre, time_pre = train_and_evaluate(pretrained=True)
model_scratch, history_scratch, time_scratch =␣
↪train_and_evaluate(pretrained=False)
```

```
Training model with pretrained=True
Epoch 1/50
93/93 [==============================] - 19s 174ms/step - loss: 0.0186 - mae:
0.1052 - val_loss: 0.0277 - val_mae: 0.1285 - lr: 1.0000e-04
Epoch 2/50
93/93 [==============================] - 16s 169ms/step - loss: 0.0053 - mae:
0.0562 - val_loss: 0.0207 - val_mae: 0.1102 - lr: 1.0000e-04
Epoch 3/50
93/93 [==============================] - 16s 168ms/step - loss: 0.0021 - mae:
0.0349 - val_loss: 0.0178 - val_mae: 0.1027 - lr: 1.0000e-04
Epoch 4/50
93/93 [==============================] - 16s 169ms/step - loss: 0.0011 - mae:
0.0254 - val_loss: 0.0157 - val_mae: 0.0964 - lr: 1.0000e-04
Epoch 5/50
93/93 [==============================] - 16s 171ms/step - loss: 8.5937e-04 -
mae: 0.0228 - val_loss: 0.0151 - val_mae: 0.0942 - lr: 1.0000e-04
Epoch 6/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0013 - mae:
0.0278 - val_loss: 0.0138 - val_mae: 0.0903 - lr: 1.0000e-04
Epoch 7/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0023 - mae:
0.0374 - val_loss: 0.0150 - val_mae: 0.0950 - lr: 1.0000e-04
Epoch 8/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0028 - mae:
0.0415 - val_loss: 0.0143 - val_mae: 0.0928 - lr: 1.0000e-04
Epoch 9/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0023 - mae:
0.0377 - val_loss: 0.0130 - val_mae: 0.0870 - lr: 1.0000e-04
Epoch 10/50
93/93 [==============================] - 16s 177ms/step - loss: 0.0018 - mae:
0.0326 - val_loss: 0.0110 - val_mae: 0.0803 - lr: 1.0000e-04
Epoch 11/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0012 - mae:
0.0274 - val_loss: 0.0107 - val_mae: 0.0792 - lr: 1.0000e-04
Epoch 12/50
93/93 [==============================] - 16s 173ms/step - loss: 9.0517e-04 -
mae: 0.0240 - val_loss: 0.0106 - val_mae: 0.0787 - lr: 1.0000e-04
Epoch 13/50
93/93 [==============================] - 16s 176ms/step - loss: 7.3333e-04 -
mae: 0.0213 - val_loss: 0.0091 - val_mae: 0.0730 - lr: 1.0000e-04
Epoch 14/50
```

```
93/93 [==============================] - 17s 179ms/step - loss: 7.1258e-04 -
mae: 0.0211 - val_loss: 0.0082 - val_mae: 0.0694 - lr: 1.0000e-04
Epoch 15/50
93/93 [==============================] - 16s 175ms/step - loss: 7.2782e-04 -
mae: 0.0215 - val_loss: 0.0085 - val_mae: 0.0708 - lr: 1.0000e-04
Epoch 16/50
93/93 [==============================] - 17s 178ms/step - loss: 7.7636e-04 -
mae: 0.0218 - val_loss: 0.0083 - val_mae: 0.0691 - lr: 1.0000e-04
Epoch 17/50
93/93 [==============================] - 17s 179ms/step - loss: 9.3953e-04 -
mae: 0.0241 - val_loss: 0.0079 - val_mae: 0.0671 - lr: 1.0000e-04
Epoch 18/50
93/93 [==============================] - 16s 177ms/step - loss: 9.9903e-04 -
mae: 0.0245 - val_loss: 0.0084 - val_mae: 0.0701 - lr: 1.0000e-04
Epoch 19/50
93/93 [==============================] - 17s 182ms/step - loss: 0.0010 - mae:
0.0247 - val_loss: 0.0080 - val_mae: 0.0681 - lr: 1.0000e-04
Epoch 20/50
93/93 [==============================] - 16s 176ms/step - loss: 9.8256e-04 -
mae: 0.0242 - val_loss: 0.0071 - val_mae: 0.0641 - lr: 1.0000e-04
Epoch 21/50
93/93 [==============================] - 16s 177ms/step - loss: 8.9387e-04 -
mae: 0.0232 - val_loss: 0.0078 - val_mae: 0.0674 - lr: 1.0000e-04
Epoch 22/50
93/93 [==============================] - 16s 174ms/step - loss: 8.8604e-04 -
mae: 0.0234 - val_loss: 0.0069 - val_mae: 0.0633 - lr: 1.0000e-04
Epoch 23/50
93/93 [==============================] - 16s 175ms/step - loss: 8.4579e-04 -
mae: 0.0227 - val_loss: 0.0068 - val_mae: 0.0631 - lr: 1.0000e-04
Epoch 24/50
93/93 [==============================] - 17s 178ms/step - loss: 8.5748e-04 -
mae: 0.0228 - val_loss: 0.0070 - val_mae: 0.0641 - lr: 1.0000e-04
Epoch 25/50
93/93 [==============================] - 16s 174ms/step - loss: 8.2900e-04 -
mae: 0.0228 - val_loss: 0.0065 - val_mae: 0.0613 - lr: 1.0000e-04
Epoch 26/50
93/93 [==============================] - 16s 171ms/step - loss: 7.5020e-04 -
mae: 0.0217 - val_loss: 0.0064 - val_mae: 0.0601 - lr: 1.0000e-04
Epoch 27/50
93/93 [==============================] - 16s 173ms/step - loss: 6.9852e-04 -
mae: 0.0207 - val_loss: 0.0063 - val_mae: 0.0597 - lr: 1.0000e-04
Epoch 28/50
93/93 [==============================] - 16s 175ms/step - loss: 7.2608e-04 -
mae: 0.0209 - val_loss: 0.0066 - val_mae: 0.0612 - lr: 1.0000e-04
Epoch 29/50
93/93 [==============================] - 17s 179ms/step - loss: 7.1501e-04 -
mae: 0.0209 - val_loss: 0.0067 - val_mae: 0.0624 - lr: 1.0000e-04
Epoch 30/50
```

```
93/93 [==============================] - 16s 171ms/step - loss: 8.2320e-04 -
mae: 0.0225 - val_loss: 0.0060 - val_mae: 0.0585 - lr: 5.0000e-05
Epoch 31/50
93/93 [==============================] - 16s 172ms/step - loss: 3.7404e-04 -
mae: 0.0152 - val_loss: 0.0058 - val_mae: 0.0572 - lr: 5.0000e-05
Epoch 32/50
93/93 [==============================] - 16s 175ms/step - loss: 1.5306e-04 -
mae: 0.0097 - val_loss: 0.0057 - val_mae: 0.0566 - lr: 5.0000e-05
Epoch 33/50
93/93 [==============================] - 16s 172ms/step - loss: 7.3780e-05 -
mae: 0.0066 - val_loss: 0.0057 - val_mae: 0.0564 - lr: 5.0000e-05
Epoch 34/50
93/93 [==============================] - 16s 169ms/step - loss: 4.3827e-05 -
mae: 0.0050 - val_loss: 0.0056 - val_mae: 0.0563 - lr: 5.0000e-05
Epoch 35/50
93/93 [==============================] - 16s 171ms/step - loss: 3.1021e-05 -
mae: 0.0043 - val_loss: 0.0056 - val_mae: 0.0561 - lr: 5.0000e-05
Epoch 36/50
93/93 [==============================] - 16s 170ms/step - loss: 2.4709e-05 -
mae: 0.0038 - val_loss: 0.0056 - val_mae: 0.0560 - lr: 5.0000e-05
Epoch 37/50
93/93 [==============================] - 16s 170ms/step - loss: 4.4209e-05 -
mae: 0.0052 - val_loss: 0.0056 - val_mae: 0.0557 - lr: 2.5000e-05
Epoch 38/50
93/93 [==============================] - 16s 170ms/step - loss: 4.0830e-05 -
mae: 0.0050 - val_loss: 0.0056 - val_mae: 0.0556 - lr: 2.5000e-05
Epoch 39/50
93/93 [==============================] - 16s 169ms/step - loss: 3.3737e-05 -
mae: 0.0045 - val_loss: 0.0056 - val_mae: 0.0556 - lr: 2.5000e-05
Epoch 40/50
93/93 [==============================] - 16s 169ms/step - loss: 4.1728e-05 -
mae: 0.0051 - val_loss: 0.0056 - val_mae: 0.0557 - lr: 1.2500e-05
Epoch 41/50
93/93 [==============================] - 16s 171ms/step - loss: 3.6228e-05 -
mae: 0.0048 - val_loss: 0.0056 - val_mae: 0.0557 - lr: 1.2500e-05
Epoch 42/50
93/93 [==============================] - 16s 170ms/step - loss: 2.7631e-05 -
mae: 0.0041 - val_loss: 0.0056 - val_mae: 0.0558 - lr: 1.2500e-05
Epoch 43/50
93/93 [==============================] - 16s 173ms/step - loss: 2.4289e-05 -
mae: 0.0039 - val_loss: 0.0056 - val_mae: 0.0559 - lr: 6.2500e-06
Training time: 696.00 seconds
Training model with pretrained=False
Epoch 1/50
93/93 [==============================] - 20s 176ms/step - loss: 0.0247 - mae:
0.1222 - val_loss: 0.0287 - val_mae: 0.1394 - lr: 1.0000e-04
Epoch 2/50
93/93 [==============================] - 16s 173ms/step - loss: 0.0217 - mae:
```

```
0.1141 - val_loss: 0.0284 - val_mae: 0.1383 - lr: 1.0000e-04
Epoch 3/50
93/93 [==============================] - 16s 174ms/step - loss: 0.0181 - mae:
0.1039 - val_loss: 0.0278 - val_mae: 0.1367 - lr: 1.0000e-04
Epoch 4/50
93/93 [==============================] - 16s 176ms/step - loss: 0.0149 - mae:
0.0948 - val_loss: 0.0273 - val_mae: 0.1352 - lr: 1.0000e-04
Epoch 5/50
93/93 [==============================] - 16s 177ms/step - loss: 0.0126 - mae:
0.0872 - val_loss: 0.0270 - val_mae: 0.1342 - lr: 1.0000e-04
Epoch 6/50
93/93 [==============================] - 16s 174ms/step - loss: 0.0109 - mae:
0.0814 - val_loss: 0.0264 - val_mae: 0.1323 - lr: 1.0000e-04
Epoch 7/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0103 - mae:
0.0787 - val_loss: 0.0268 - val_mae: 0.1332 - lr: 1.0000e-04
Epoch 8/50
93/93 [==============================] - 17s 179ms/step - loss: 0.0086 - mae:
0.0716 - val_loss: 0.0263 - val_mae: 0.1314 - lr: 1.0000e-04
Epoch 9/50
93/93 [==============================] - 16s 174ms/step - loss: 0.0072 - mae:
0.0662 - val_loss: 0.0265 - val_mae: 0.1314 - lr: 1.0000e-04
Epoch 10/50
93/93 [==============================] - 16s 172ms/step - loss: 0.0061 - mae:
0.0613 - val_loss: 0.0258 - val_mae: 0.1293 - lr: 1.0000e-04
Epoch 11/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0055 - mae:
0.0580 - val_loss: 0.0264 - val_mae: 0.1308 - lr: 1.0000e-04
Epoch 12/50
93/93 [==============================] - 16s 176ms/step - loss: 0.0046 - mae:
0.0531 - val_loss: 0.0291 - val_mae: 0.1372 - lr: 1.0000e-04
Epoch 13/50
93/93 [==============================] - 16s 173ms/step - loss: 0.0042 - mae:
0.0510 - val_loss: 0.0291 - val_mae: 0.1375 - lr: 1.0000e-04
Epoch 14/50
93/93 [==============================] - 16s 171ms/step - loss: 0.0037 - mae:
0.0475 - val_loss: 0.0282 - val_mae: 0.1347 - lr: 5.0000e-05
Epoch 15/50
93/93 [==============================] - 16s 175ms/step - loss: 0.0024 - mae:
0.0384 - val_loss: 0.0294 - val_mae: 0.1372 - lr: 5.0000e-05
Training time: 246.36 seconds
```

[17]:
```
# ===============================
# Step 7.1: Save both trained models to HDF5 (.h5) format
# ===============================

models_dir = "models"
```

```
if not os.path.exists(models_dir):
    os.makedirs(models_dir)

model_pretrained.save(os.path.join(models_dir, "object_detector_pretrained.h5"))
print("Saved: models/object_detector_pretrained.h5")

model_scratch.save(os.path.join(models_dir, "object_detector_scratch.h5"))
print("Saved: models/object_detector_scratch.h5")

# ================================
# Step 7.2: Save training histories to CSV
# ================================

import pandas as pd

pd.DataFrame(history_pre.history).to_csv(os.path.join(models_dir,␣
 ↪"training_log_pretrained.csv"), index=False)
print("Saved: models/training_log_pretrained.csv")

pd.DataFrame(history_scratch.history).to_csv(os.path.join(models_dir,␣
 ↪"training_log_scratch.csv"), index=False)
print("Saved: models/training_log_scratch.csv")
```

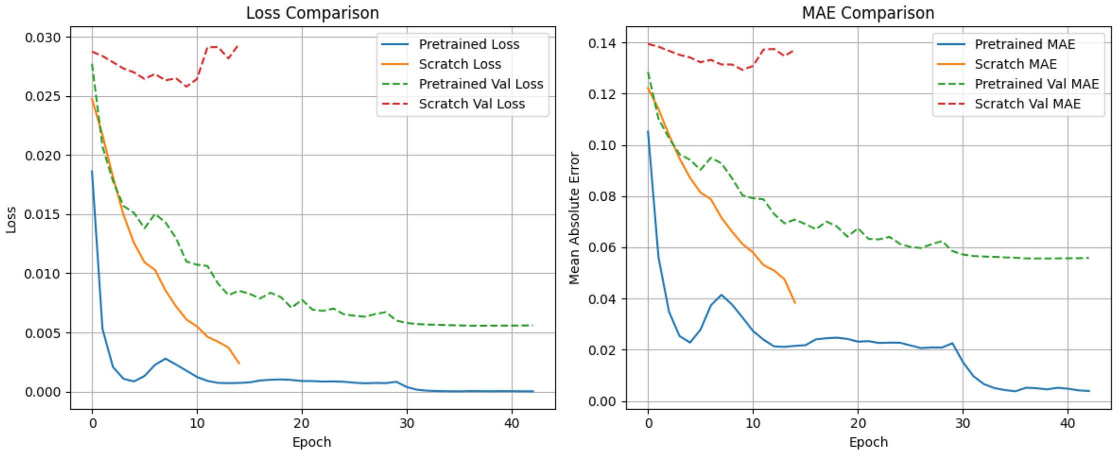
```
Saved: models/object_detector_pretrained.h5
Saved: models/object_detector_scratch.h5
Saved: models/training_log_pretrained.csv
Saved: models/training_log_scratch.csv
```

[23]:
```
# ================================
# Step 7.3 : Load both trained models from HDF5 files
# ================================

model_pretrained = load_model(os.path.join(models_dir,␣
 ↪"object_detector_pretrained.h5"))
model_scratch = load_model(os.path.join(models_dir, "object_detector_scratch.
 ↪h5"))
print("Loaded pretrained model from 'models/object_detector_pretrained.h5'")
print("Loaded scratch model from 'models/object_detector_scratch.h5'")
```

```
Loaded pretrained model from 'models/object_detector_pretrained.h5'
Loaded scratch model from 'models/object_detector_scratch.h5'
```

[24]:
```
# ================================
# Step 7.4: Visualize training and validation loss and MAE for both models
# ================================

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12, 5))

# Plot loss
plt.subplot(1, 2, 1)
plt.plot(history_pre.history['loss'], label='Pretrained Loss')
plt.plot(history_scratch.history['loss'], label='Scratch Loss')
plt.plot(history_pre.history['val_loss'], label='Pretrained Val Loss',␣
↪linestyle='--')
plt.plot(history_scratch.history['val_loss'], label='Scratch Val Loss',␣
↪linestyle='--')
plt.title("Loss Comparison")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

# Plot MAE
plt.subplot(1, 2, 2)
plt.plot(history_pre.history['mae'], label='Pretrained MAE')
plt.plot(history_scratch.history['mae'], label='Scratch MAE')
plt.plot(history_pre.history['val_mae'], label='Pretrained Val MAE',␣
↪linestyle='--')
plt.plot(history_scratch.history['val_mae'], label='Scratch Val MAE',␣
↪linestyle='--')
plt.title("MAE Comparison")
plt.xlabel("Epoch")
plt.ylabel("Mean Absolute Error")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()
```

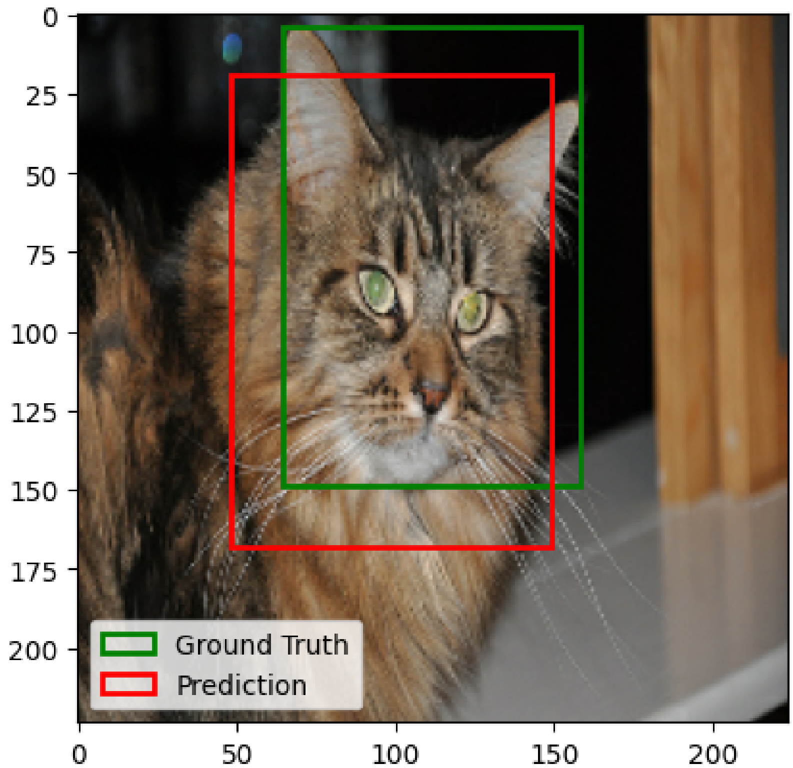


[25]:
```python
# ================================
# Step 8: Compute Intersection over Union (IoU)
# ================================
# Compute IoU between predicted and true bounding boxes for visual examples

def compute_iou(box1, box2):
    def denorm(bbox):
        cx, cy, w, h = bbox
        xmin = cx - w / 2
        ymin = cy - h / 2
        xmax = cx + w / 2
        ymax = cy + h / 2
        return [xmin, ymin, xmax, ymax]

    box1 = denorm(box1)
    box2 = denorm(box2)

    xA = max(box1[0], box2[0])
    yA = max(box1[1], box2[1])
    xB = min(box1[2], box2[2])
    yB = min(box1[3], box2[3])

    inter_area = max(0, xB - xA) * max(0, yB - yA)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union_area = box1_area + box2_area - inter_area

    iou = inter_area / union_area if union_area > 0 else 0
    return iou
```

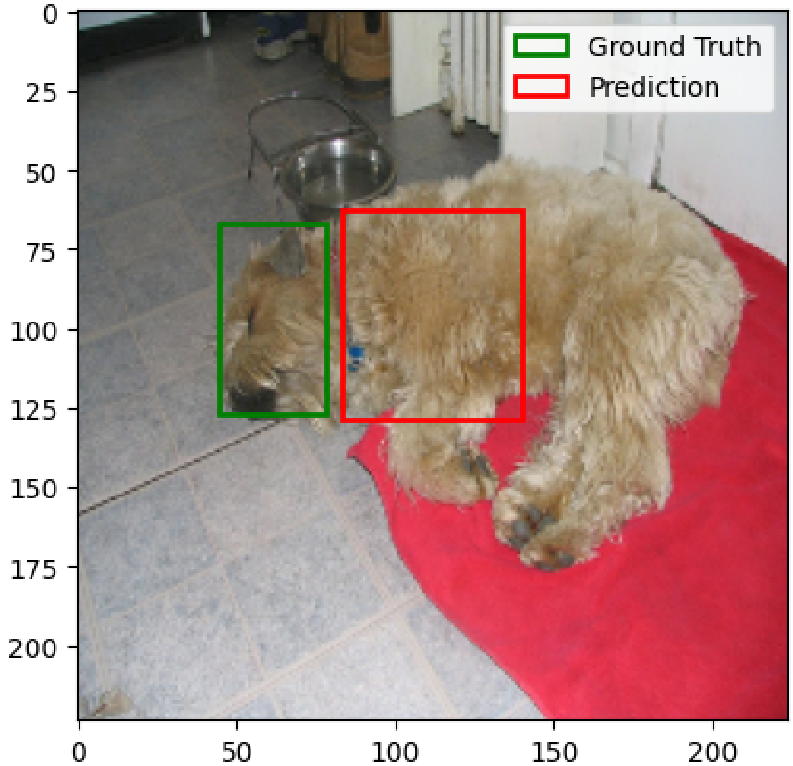
[26]:
```python
# ================================
# Step 9: Visualize predictions and IoU
# ================================
# Visualize a few predictions from the validation set and display the IoU

def visualize_predictions(model, dataset, n=3):
    for images, bboxes in dataset.take(1):
        preds = model.predict(images)
        for i in range(n):
            img_np = images[i].numpy()
            true_bbox = bboxes[i].numpy()
            pred_bbox = preds[i]

            iou_score = compute_iou(true_bbox, pred_bbox)
            print(f"IoU Score: {iou_score:.3f}")
```

```
            display_bbox(img_np, true_bbox, pred_bbox)

# Show prediction results from pretrained model
visualize_predictions(model_pretrained, val_det_ds)
```

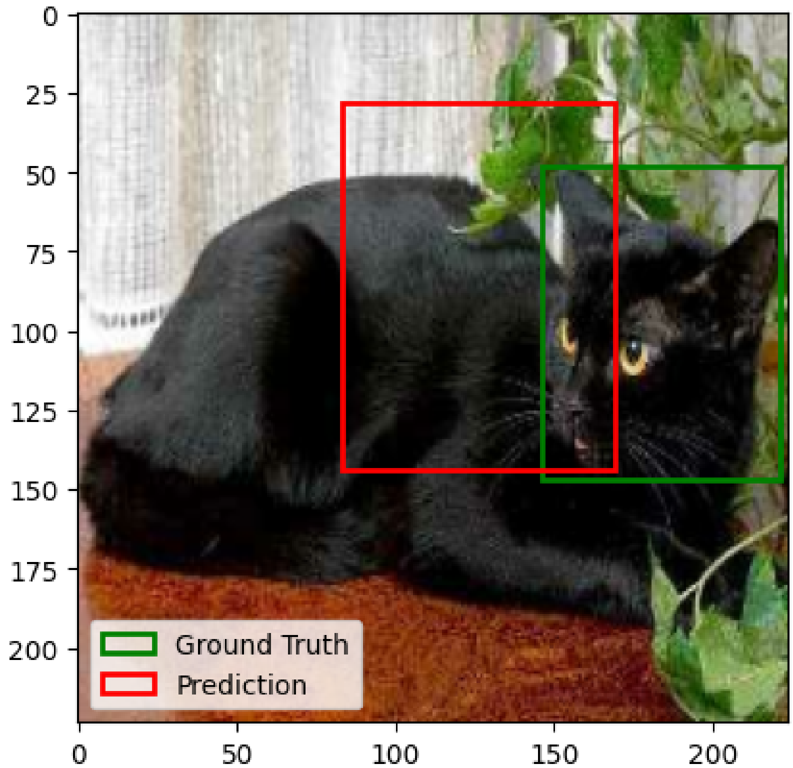
```
1/1 [==============================] - 1s 571ms/step
IoU Score: 0.622
```




```
IoU Score: 0.000
```



IoU Score: 0.147



[ ]:
```
# ================================
# Step 10: Run inference on test images (optional)
# ================================
#
# If test images are prepared with bbox, use this step for evaluation

# Example:
# test_det_ds = create_detection_dataset(test_ids)
# visualize_predictions(model_pretrained, test_det_ds)
```