Object detection with Vision Transformers

Author: Karan V. Dave
Date created: 2022/03/27
Last modified: 2022/03/27
Description: A simple Keras implementation of object detection using Vision Transformers.

Introduction

The article Vision Transformer (ViT) architecture by Alexey Dosovitskiy et al. demonstrates that a pure transformer applied directly to sequences of image patches can perform well on object detection tasks.

In this Keras example, we implement an object detection ViT and we train it on the Caltech 101 dataset to detect an airplane in the given image.

This example requires TensorFlow 2.4 or higher, and TensorFlow Addons, from which we import the `AdamW` optimizer.

TensorFlow Addons can be installed via the following command:

```
pip install -U tensorflow-addons
```

In [ ]:
```
# pip install -U tensorflow-addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.19.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 15.8 MB/s eta 0:00:00
Collecting typeguard>=2.7
  Downloading typeguard-3.0.2-py3-none-any.whl (30 kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from tensorflow-addons) (23.0)
Requirement already satisfied: importlib-metadata>=3.6 in /usr/local/lib/python3.9/dist-packages (from typeguard>=2.7->tensorflow-addons) (6.1.0)
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.9/dist-packages (from typeguard>=2.7->tensorflow-addons) (4.5.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.9/dist-packages (from importlib-metadata>=3.6->typeguard>=2.7->tensorflow-addons) (3.15.0)
Installing collected packages: typeguard, tensorflow-addons
Successfully installed tensorflow-addons-0.19.0 typeguard-3.0.2
```

Imports and setup

In [ ]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_addons as tfa
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
import scipy.io
import shutil
```

```
/usr/local/lib/python3.9/dist-packages/tensorflow_addons/utils/ensure_tf_in
stall.py:53: UserWarning: Tensorflow Addons supports using Python ops for a
ll Tensorflow versions above or equal to 2.9.0 and strictly below 2.12.0 (n
ightly versions are not supported).
 The versions of TensorFlow you are currently using is 2.12.0 and is not su
pported.
Some things might work, some things might not.
If you were to encounter a bug, do not file an issue.
If you want to make sure you're using a tested and supported configuration,
either change the TensorFlow version or the TensorFlow Addons's version.
You can find the compatibility matrix in TensorFlow Addon's readme:
https://github.com/tensorflow/addons
  warnings.warn(
```

Prepare dataset

We use the Caltech 101 Dataset.

In [ ]:
```
# Path to images and annotations
path_images = "/101_ObjectCategories/airplanes/"
path_annot = "/Annotations/Airplanes_Side_2/"

path_to_downloaded_file = keras.utils.get_file(
    fname="caltech_101_zipped",
    origin="https://data.caltech.edu/records/mzrjq-6wc02/files/caltech-101.z
    extract=True,
    archive_format="zip",  # downloaded file format
    cache_dir="/",  # cache and extract in current directory
)

# Extracting tar files found inside main zip file
shutil.unpack_archive("/datasets/caltech-101/101_ObjectCategories.tar.gz", "
shutil.unpack_archive("/datasets/caltech-101/Annotations.tar", "/")

# list of paths to images and annotations
image_paths = [
    f for f in os.listdir(path_images) if os.path.isfile(os.path.join(path_i
]
```

```
annot_paths = [
    f for f in os.listdir(path_annot) if os.path.isfile(os.path.join(path_ar
]

image_paths.sort()
annot_paths.sort()

image_size = 224  # resize input images to this size

images, targets = [], []

# loop over the annotations and images, preprocess them and store in lists
for i in range(0, len(annot_paths)):
    # Access bounding box coordinates
    annot = scipy.io.loadmat(path_annot + annot_paths[i])["box_coord"][0]

    top_left_x, top_left_y = annot[2], annot[0]
    bottom_right_x, bottom_right_y = annot[3], annot[1]

    image = keras.utils.load_img(
        path_images + image_paths[i],
    )
    (w, h) = image.size[:2]

    # resize train set images
    if i < int(len(annot_paths) * 0.8):
        # resize image if it is for training dataset
        image = image.resize((image_size, image_size))

    # convert image to array and append to list
    images.append(keras.utils.img_to_array(image))

    # apply relative scaling to bounding boxes as per given image and append
    targets.append(
        (
            float(top_left_x) / w,
            float(top_left_y) / h,
            float(bottom_right_x) / w,
            float(bottom_right_y) / h,
        )
    )

# Convert the list to numpy array, split to train and test dataset
(x_train), (y_train) = (
    np.asarray(images[: int(len(images) * 0.8)]),
    np.asarray(targets[: int(len(targets) * 0.8)]),
)
(x_test), (y_test) = (
    np.asarray(images[int(len(images) * 0.8) :]),
    np.asarray(targets[int(len(targets) * 0.8) :]),
)
```

```
Downloading data from https://data.caltech.edu/records/mzrjq-6wc02/files/ca
ltech-101.zip
137414764/137414764 [==============================] - 9s 0us/step
```

```
<ipython-input-3-6860251810f7>:69: VisibleDeprecationWarning: Creating an n
darray from ragged nested sequences (which is a list-or-tuple of lists-or-t
uples-or ndarrays with different lengths or shapes) is deprecated. If you m
eant to do this, you must specify 'dtype=object' when creating the ndarray.
  np.asarray(images[int(len(images) * 0.8) :]),
```

Implement multilayer-perceptron (MLP)

We use the code from the Keras example Image classification with Vision Transformer as a reference.

In [ ]:
```python
def mlp(x, hidden_units, dropout_rate):
    for units in hidden_units:
        x = layers.Dense(units, activation=tf.nn.gelu)(x)
        x = layers.Dropout(dropout_rate)(x)
    return x
```

Implement the patch creation layer

In [ ]:
```
class Patches(layers.Layer):
    def __init__(self, patch_size):
        super().__init__()
        self.patch_size = patch_size

    #     Override function to avoid error while saving model
    def get_config(self):
        config = super().get_config().copy()
        config.update(
            {
                "input_shape": input_shape,
                "patch_size": patch_size,
                "num_patches": num_patches,
                "projection_dim": projection_dim,
                "num_heads": num_heads,
                "transformer_units": transformer_units,
                "transformer_layers": transformer_layers,
                "mlp_head_units": mlp_head_units,
            }
        )
        return config

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images=images,
            sizes=[1, self.patch_size, self.patch_size, 1],
            strides=[1, self.patch_size, self.patch_size, 1],
            rates=[1, 1, 1, 1],
            padding="VALID",
        )
        # return patches
        return tf.reshape(patches, [batch_size, -1, patches.shape[-1]])
```

Display patches for an input image

In [ ]:
```
patch_size = 32  # Size of the patches to be extracted from the input images

plt.figure(figsize=(4, 4))
plt.imshow(x_train[0].astype("uint8"))
plt.axis("off")

patches = Patches(patch_size)(tf.convert_to_tensor([x_train[0]]))
print(f"Image size: {image_size} X {image_size}")
print(f"Patch size: {patch_size} X {patch_size}")
print(f"{patches.shape[1]} patches per image \n{patches.shape[-1]} elements


n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))
for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
```

```
    plt.imshow(patch_img.numpy().astype("uint8"))
    plt.axis("off")
```

```
Image size: 224 X 224
Patch size: 32 X 32
49 patches per image
3072 elements per patch
```

Implement the patch encoding layer

The `PatchEncoder` layer linearly transforms a patch by projecting it into a vector of size `projection_dim`. It also adds a learnable position embedding to the projected vector.

In [ ]:
```
class PatchEncoder(layers.Layer):
    def __init__(self, num_patches, projection_dim):
        super().__init__()
        self.num_patches = num_patches
        self.projection = layers.Dense(units=projection_dim)
        self.position_embedding = layers.Embedding(
            input_dim=num_patches, output_dim=projection_dim
        )

    # Override function to avoid error while saving model
    def get_config(self):
        config = super().get_config().copy()
        config.update(
            {
                "input_shape": input_shape,
                "patch_size": patch_size,
                "num_patches": num_patches,
                "projection_dim": projection_dim,
                "num_heads": num_heads,
                "transformer_units": transformer_units,
                "transformer_layers": transformer_layers,
                "mlp_head_units": mlp_head_units,
            }
        )
        return config

    def call(self, patch):
        positions = tf.range(start=0, limit=self.num_patches, delta=1)
        encoded = self.projection(patch) + self.position_embedding(positions
        return encoded
```

Build the ViT model

The ViT model has multiple Transformer blocks. The `MultiHeadAttention` layer is used for self-attention, applied to the sequence of image patches. The encoded patches (skip connection) and self-attention layer outputs are normalized and fed into a multilayer perceptron (MLP). The model outputs four dimensions representing the bounding box coordinates of an object.

In [ ]:
```
def create_vit_object_detector(
    input_shape,
    patch_size,
    num_patches,
    projection_dim,
    num_heads,
    transformer_units,
    transformer_layers,
    mlp_head_units,
):
    inputs = layers.Input(shape=input_shape)
    # Create patches
    patches = Patches(patch_size)(inputs)
```

```
    # Encode patches
    encoded_patches = PatchEncoder(num_patches, projection_dim)(patches)

    # Create multiple layers of the Transformer block.
    for _ in range(transformer_layers):
        # Layer normalization 1.
        x1 = layers.LayerNormalization(epsilon=1e-6)(encoded_patches)
        # Create a multi-head attention layer.
        attention_output = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=projection_dim, dropout=0.1
        )(x1, x1)
        # Skip connection 1.
        x2 = layers.Add()([attention_output, encoded_patches])
        # Layer normalization 2.
        x3 = layers.LayerNormalization(epsilon=1e-6)(x2)
        # MLP
        x3 = mlp(x3, hidden_units=transformer_units, dropout_rate=0.1)
        # Skip connection 2.
        encoded_patches = layers.Add()([x3, x2])

    # Create a [batch_size, projection_dim] tensor.
    representation = layers.LayerNormalization(epsilon=1e-6)(encoded_patches
    representation = layers.Flatten()(representation)
    representation = layers.Dropout(0.3)(representation)
    # Add MLP.
    features = mlp(representation, hidden_units=mlp_head_units, dropout_rate

    bounding_box = layers.Dense(4)(
        features
    )  # Final four neurons that output bounding box

    # return Keras model.
    return keras.Model(inputs=inputs, outputs=bounding_box)
```

Run the experiment

In [ ]:

```
def run_experiment(model, learning_rate, weight_decay, batch_size, num_epoch

    optimizer = tfa.optimizers.AdamW(
        learning_rate=learning_rate, weight_decay=weight_decay
    )

    # Compile model.
    model.compile(optimizer=optimizer, loss=keras.losses.MeanSquaredError())

    checkpoint_filepath = "logs/"
    checkpoint_callback = keras.callbacks.ModelCheckpoint(
        checkpoint_filepath,
        monitor="val_loss",
        save_best_only=True,
        save_weights_only=True,
    )

    history = model.fit(
```

```
        x=x_train,
        y=y_train,
        batch_size=batch_size,
        epochs=num_epochs,
        validation_split=0.1,
        callbacks=[
            checkpoint_callback,
            keras.callbacks.EarlyStopping(monitor="val_loss", patience=10),
        ],
    )

    return history


input_shape = (image_size, image_size, 3)  # input image shape
learning_rate = 0.001
weight_decay = 0.0001
batch_size = 32
num_epochs = 100
num_patches = (image_size // patch_size) ** 2
projection_dim = 64
num_heads = 4
# Size of the transformer layers
transformer_units = [
    projection_dim * 2,
    projection_dim,
]
transformer_layers = 4
mlp_head_units = [2048, 1024, 512, 64, 32]  # Size of the dense layers


history = []
num_patches = (image_size // patch_size) ** 2

vit_object_detector = create_vit_object_detector(
    input_shape,
    patch_size,
    num_patches,
    projection_dim,
    num_heads,
    transformer_units,
    transformer_layers,
    mlp_head_units,
)

# Train model
history = run_experiment(
    vit_object_detector, learning_rate, weight_decay, batch_size, num_epochs
)
```

```
Epoch 1/100
18/18 [==============================] - 10s 104ms/step - loss: 0.9097 - va
l_loss: 0.3615
Epoch 2/100
18/18 [==============================] - 1s 59ms/step - loss: 0.3516 - val_
loss: 0.3112
Epoch 3/100
18/18 [==============================] - 1s 55ms/step - loss: 0.2791 - val_
loss: 0.2288
Epoch 4/100
18/18 [==============================] - 1s 57ms/step - loss: 0.2010 - val_
loss: 0.1066
Epoch 5/100
18/18 [==============================] - 1s 55ms/step - loss: 0.1312 - val_
loss: 0.0355
Epoch 6/100
18/18 [==============================] - 1s 69ms/step - loss: 0.0911 - val_
loss: 0.0110
Epoch 7/100
18/18 [==============================] - 1s 76ms/step - loss: 0.0709 - val_
loss: 0.0066
Epoch 8/100
18/18 [==============================] - 1s 65ms/step - loss: 0.0545 - val_
loss: 0.0027
Epoch 9/100
18/18 [==============================] - 1s 54ms/step - loss: 0.0419 - val_
loss: 0.0018
Epoch 10/100
18/18 [==============================] - 0s 25ms/step - loss: 0.0337 - val_
loss: 0.0020
Epoch 11/100
18/18 [==============================] - 1s 28ms/step - loss: 0.0284 - val_
loss: 0.0037
Epoch 12/100
18/18 [==============================] - 0s 27ms/step - loss: 0.0277 - val_
loss: 0.0026
Epoch 13/100
18/18 [==============================] - 0s 26ms/step - loss: 0.0240 - val_
loss: 0.0032
Epoch 14/100
18/18 [==============================] - 0s 27ms/step - loss: 0.0234 - val_
loss: 0.0050
Epoch 15/100
18/18 [==============================] - 0s 25ms/step - loss: 0.0233 - val_
loss: 0.0019
Epoch 16/100
18/18 [==============================] - 0s 26ms/step - loss: 0.0236 - val_
loss: 0.0042
Epoch 17/100
18/18 [==============================] - 0s 25ms/step - loss: 0.0225 - val_
loss: 0.0036
Epoch 18/100
18/18 [==============================] - 1s 55ms/step - loss: 0.0198 - val_
loss: 0.0016
Epoch 19/100
18/18 [==============================] - 0s 26ms/step - loss: 0.0195 - val_
```

```
loss: 0.0025
Epoch 20/100
18/18 [==============================] - 0s 27ms/step - loss: 0.0190 - val_
loss: 0.0026
Epoch 21/100
18/18 [==============================] - 0s 28ms/step - loss: 0.0182 - val_
loss: 0.0023
Epoch 22/100
18/18 [==============================] - 0s 25ms/step - loss: 0.0178 - val_
loss: 0.0040
Epoch 23/100
18/18 [==============================] - 0s 26ms/step - loss: 0.0169 - val_
loss: 0.0026
Epoch 24/100
18/18 [==============================] - 0s 26ms/step - loss: 0.0160 - val_
loss: 0.0024
Epoch 25/100
18/18 [==============================] - 0s 25ms/step - loss: 0.0154 - val_
loss: 0.0024
Epoch 26/100
18/18 [==============================] - 1s 30ms/step - loss: 0.0153 - val_
loss: 0.0024
Epoch 27/100
18/18 [==============================] - 1s 32ms/step - loss: 0.0152 - val_
loss: 0.0017
Epoch 28/100
18/18 [==============================] - 1s 34ms/step - loss: 0.0146 - val_
loss: 0.0026
```

Evaluate the model

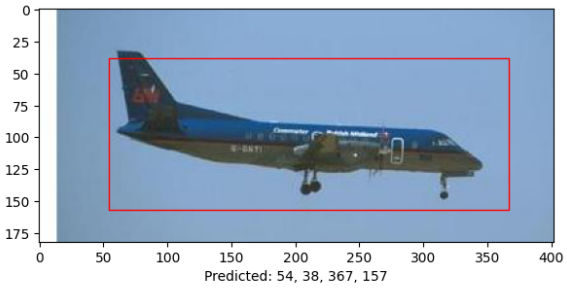
In [ ]:

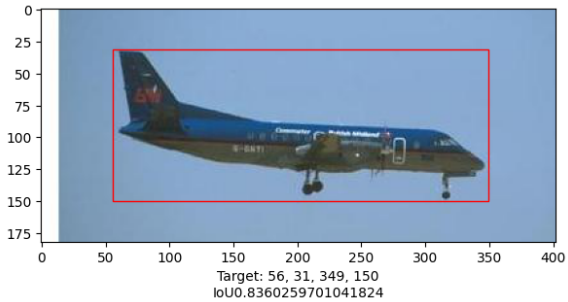
```
import matplotlib.patches as patches

# Saves the model in current path
vit_object_detector.save("vit_object_detector.h5", save_format="h5")

# To calculate IoU (intersection over union, given two bounding boxes)
def bounding_box_intersection_over_union(box_predicted, box_truth):
    # get (x, y) coordinates of intersection of bounding boxes
    top_x_intersect = max(box_predicted[0], box_truth[0])
    top_y_intersect = max(box_predicted[1], box_truth[1])
    bottom_x_intersect = min(box_predicted[2], box_truth[2])
    bottom_y_intersect = min(box_predicted[3], box_truth[3])

    # calculate area of the intersection bb (bounding box)
    intersection_area = max(0, bottom_x_intersect - top_x_intersect + 1) * m
        0, bottom_y_intersect - top_y_intersect + 1
    )

    # calculate area of the prediction bb and ground-truth bb
    box_predicted_area = (box_predicted[2] - box_predicted[0] + 1) * (
        box_predicted[3] - box_predicted[1] + 1
    )
    box_truth_area = (box_truth[2] - box_truth[0] + 1) * (
        box_truth[3] - box_truth[1] + 1
```

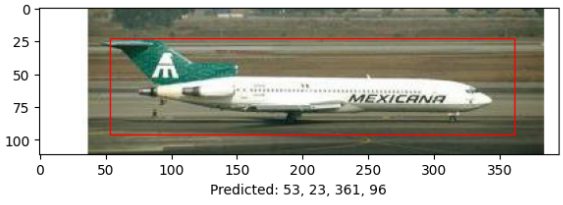
```
    )

    # calculate intersection over union by taking intersection
    # area and dividing it by the sum of predicted bb and ground truth
    # bb areas subtracted by  the interesection area

    # return ioU
    return intersection_area / float(
        box_predicted_area + box_truth_area - intersection_area
    )


i, mean_iou = 0, 0

# Compare results for 10 images in the test set
# for input_image in x_test[:10]:

# Compare results for 20 images in the test set
#for input_image in x_test[:10]:
np.random.seed(4567)
index = np.random.choice(x_test.shape[0],
                                  size=20,
                                  replace=False)
for input_image in x_test[index]:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 15))
    im = input_image

    # Display the image
    ax1.imshow(im.astype("uint8"))
    ax2.imshow(im.astype("uint8"))

    input_image = cv2.resize(
        input_image, (image_size, image_size), interpolation=cv2.INTER_AREA
    )
    input_image = np.expand_dims(input_image, axis=0)
    preds = vit_object_detector.predict(input_image)[0]

    (h, w) = (im).shape[0:2]

    top_left_x, top_left_y = int(preds[0] * w), int(preds[1] * h)

    bottom_right_x, bottom_right_y = int(preds[2] * w), int(preds[3] * h)

    box_predicted = [top_left_x, top_left_y, bottom_right_x, bottom_right_y]
    # Create the bounding box
    rect = patches.Rectangle(
        (top_left_x, top_left_y),
        bottom_right_x - top_left_x,
        bottom_right_y - top_left_y,
        facecolor="none",
        edgecolor="red",
        linewidth=1,
    )
    # Add the bounding box to the image
    ax1.add_patch(rect)
    ax1.set_xlabel(
```

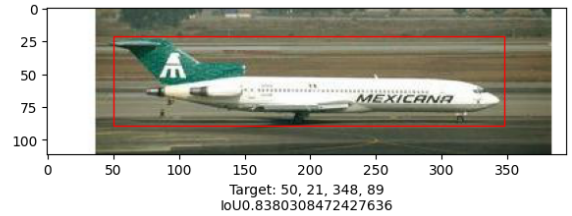
```
        "Predicted: "
        + str(top_left_x)
        + ", "
        + str(top_left_y)
        + ", "
        + str(bottom_right_x)
        + ", "
        + str(bottom_right_y)
    )

    top_left_x, top_left_y = int(y_test[i][0] * w), int(y_test[i][1] * h)

    bottom_right_x, bottom_right_y = int(y_test[i][2] * w), int(y_test[i][3]

    box_truth = top_left_x, top_left_y, bottom_right_x, bottom_right_y

    mean_iou += bounding_box_intersection_over_union(box_predicted, box_trut
    # Create the bounding box
    rect = patches.Rectangle(
        (top_left_x, top_left_y),
        bottom_right_x - top_left_x,
        bottom_right_y - top_left_y,
        facecolor="none",
        edgecolor="red",
        linewidth=1,
    )
    # Add the bounding box to the image
    ax2.add_patch(rect)
    ax2.set_xlabel(
        "Target: "
        + str(top_left_x)
        + ", "
        + str(top_left_y)
        + ", "
        + str(bottom_right_x)
        + ", "
        + str(bottom_right_y)
        + "\n"
        + "IoU"
        + str(bounding_box_intersection_over_union(box_predicted, box_truth)
    )
    i = i + 1

# print("mean_iou: " + str(mean_iou / len(x_test[:10])))
print("mean_iou: " + str(mean_iou / len(x_test[index])))
plt.show()
```

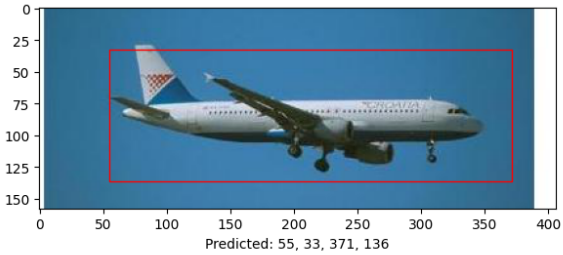
```
1/1 [==============================] - 1s 643ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
mean_iou: 0.8245216300683653
```

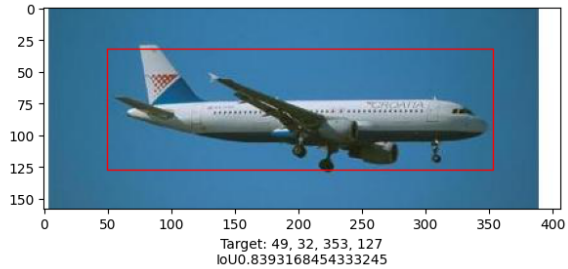


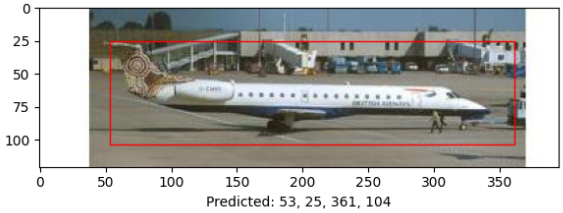


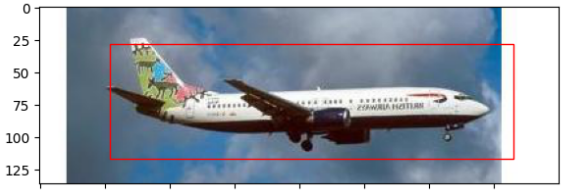


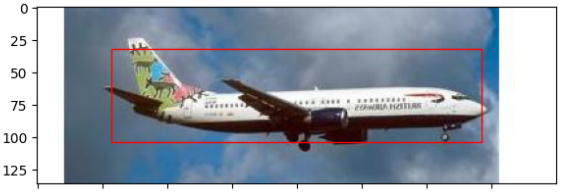






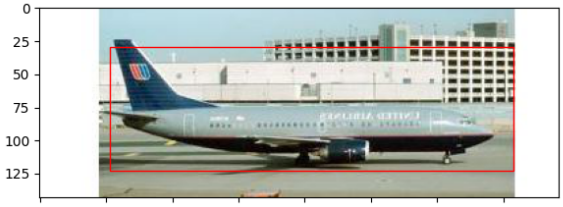


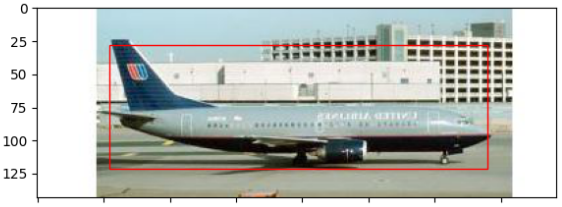

Predicted: 54, 28, 365, 117

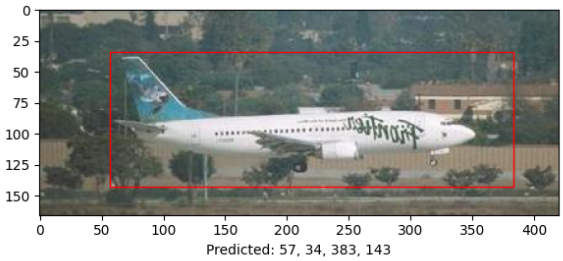

Target: 57, 32, 342, 104
IoU0.7435185185185185

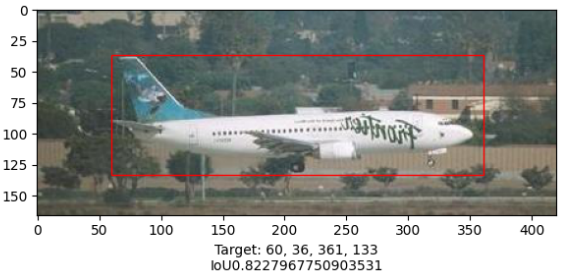

Predicted: 53, 32, 359, 134

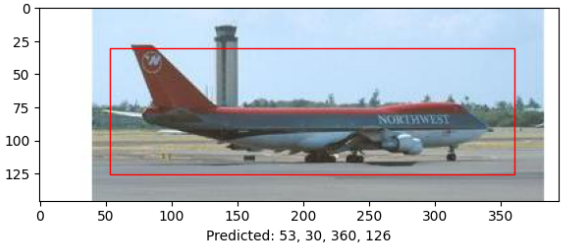

Target: 59, 28, 340, 128
IoU0.8352621454090201

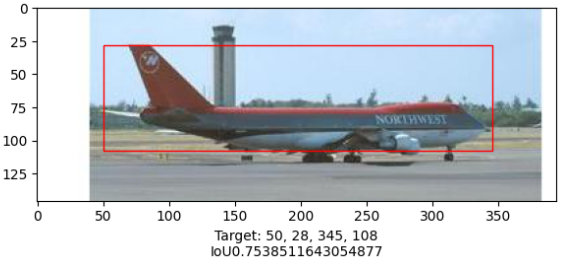

Predicted: 53, 29, 358, 123

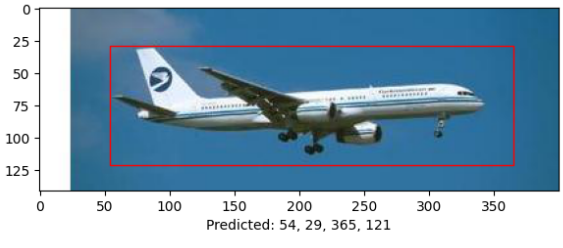

Target: 54, 28, 340, 121
IoU0.9091869060190074

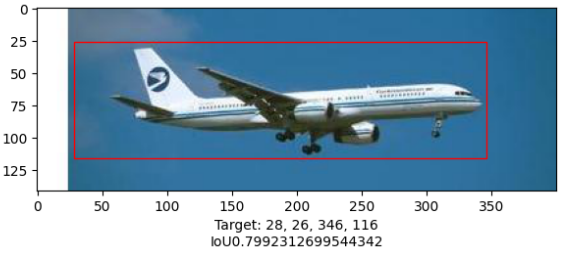

Predicted: 57, 34, 383, 143


Target: 60, 36, 361, 133
IoU0.8227967750903531


Predicted: 53, 30, 360, 126

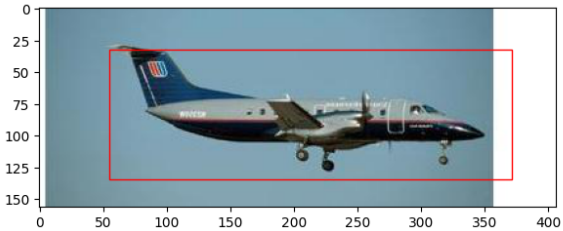

Target: 50, 28, 345, 108
IoU0.7538511643054877

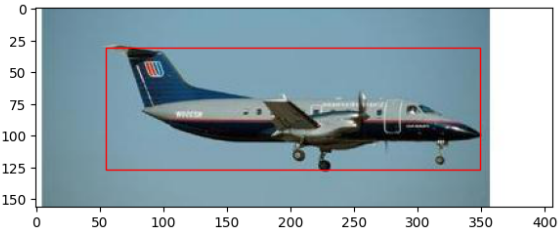

Predicted: 54, 29, 365, 121


Target: 28, 26, 346, 116
IoU0.7992312699544342


Predicted: 53, 33, 358, 136

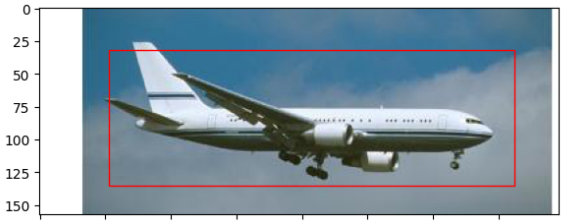

Target: 60, 32, 338, 124
IoU0.7995514437903

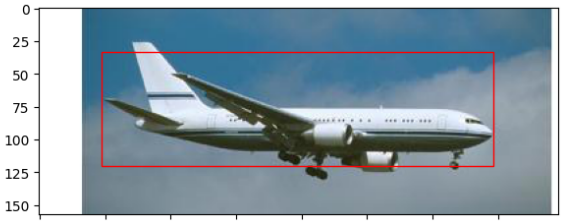

Predicted: 55, 32, 371, 134


Target: 55, 31, 349, 127
IoU0.8595884174103078


Predicted: 54, 34, 364, 143

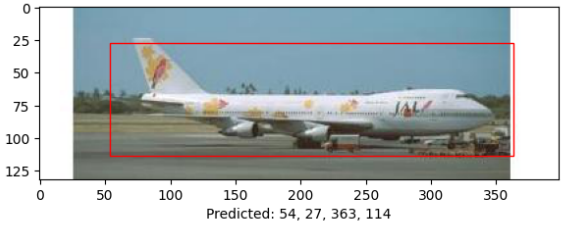

Target: 54, 36, 346, 128
IoU0.7965214849459222

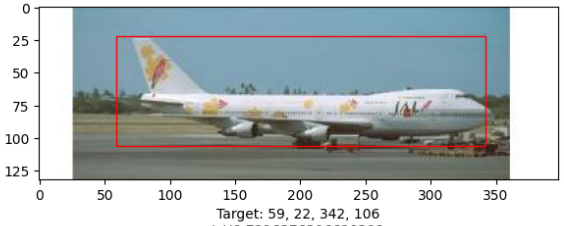

Predicted: 53, 32, 362, 135

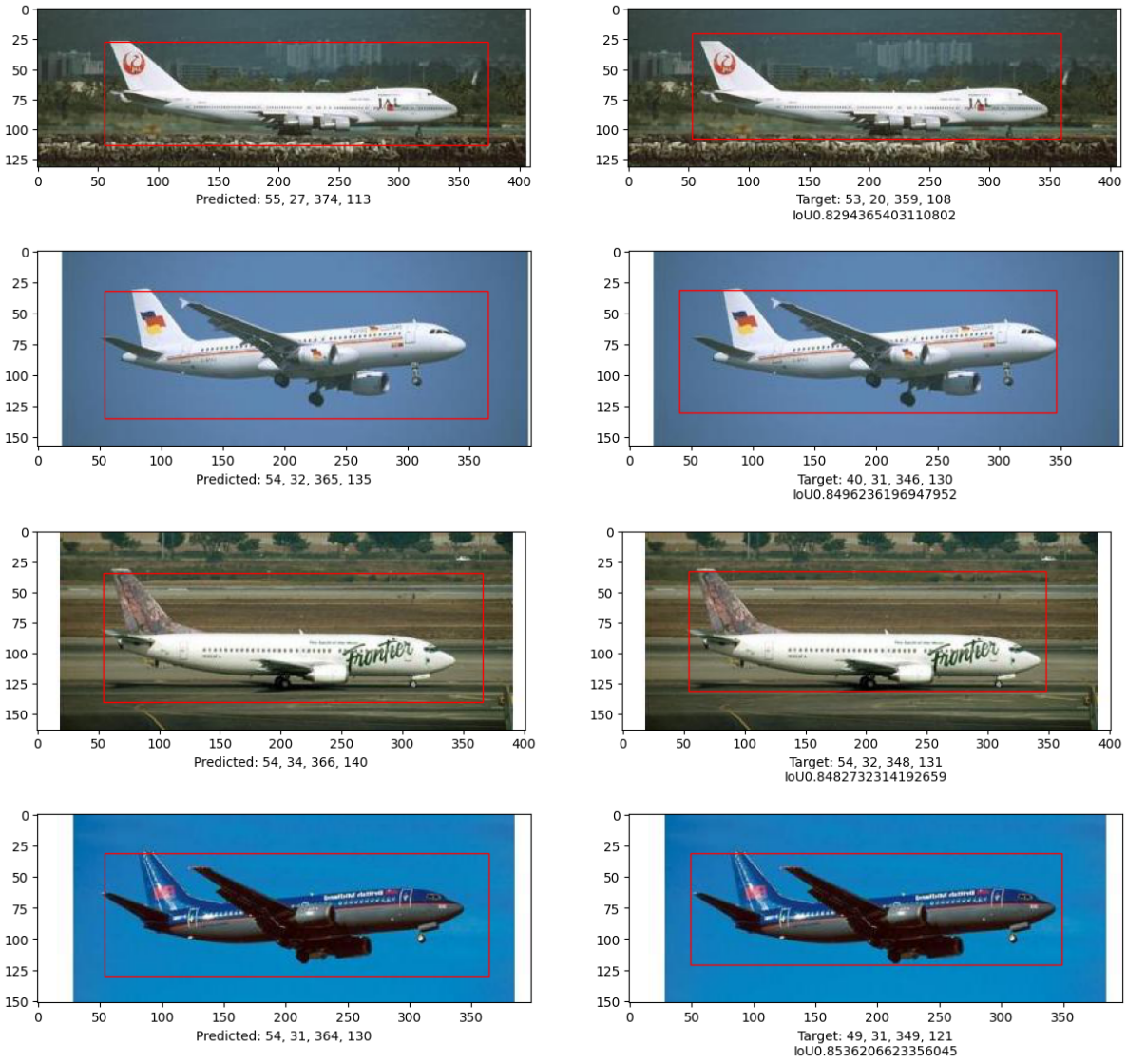

Target: 47, 33, 346, 120
IoU0.78955078125


Predicted: 56, 38, 378, 158

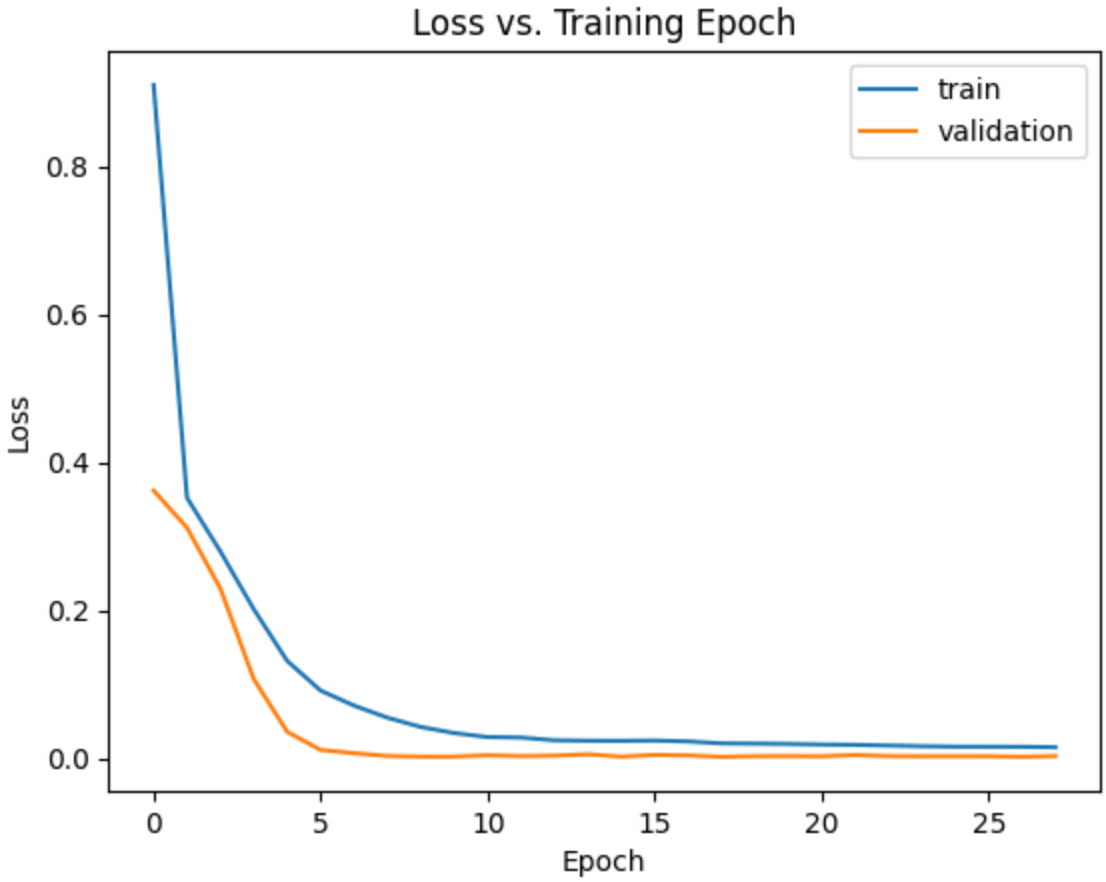

Target: 51, 33, 357, 152
IoU0.8431044109436069


Predicted: 54, 27, 363, 114


Target: 59, 22, 342, 106
IoU0.7916376306620209



In [ ]:
```python
import matplotlib.pyplot as plt

def plot_hist(hist):
    plt.plot(hist.history["loss"], label="train")
    plt.plot(hist.history["val_loss"], label="validation")
    plt.title("Loss vs. Training Epoch")
    plt.ylabel("Loss")
    plt.xlabel("Epoch")
    plt.legend(loc="upper right")
    plt.show()

plot_hist(history)
```



This example demonstrates that a pure Transformer can be trained to predict the bounding boxes of an object in a given image, thus extending the use of Transformers to object detection tasks. The model can be improved further by tuning hyper-parameters and pre-training.