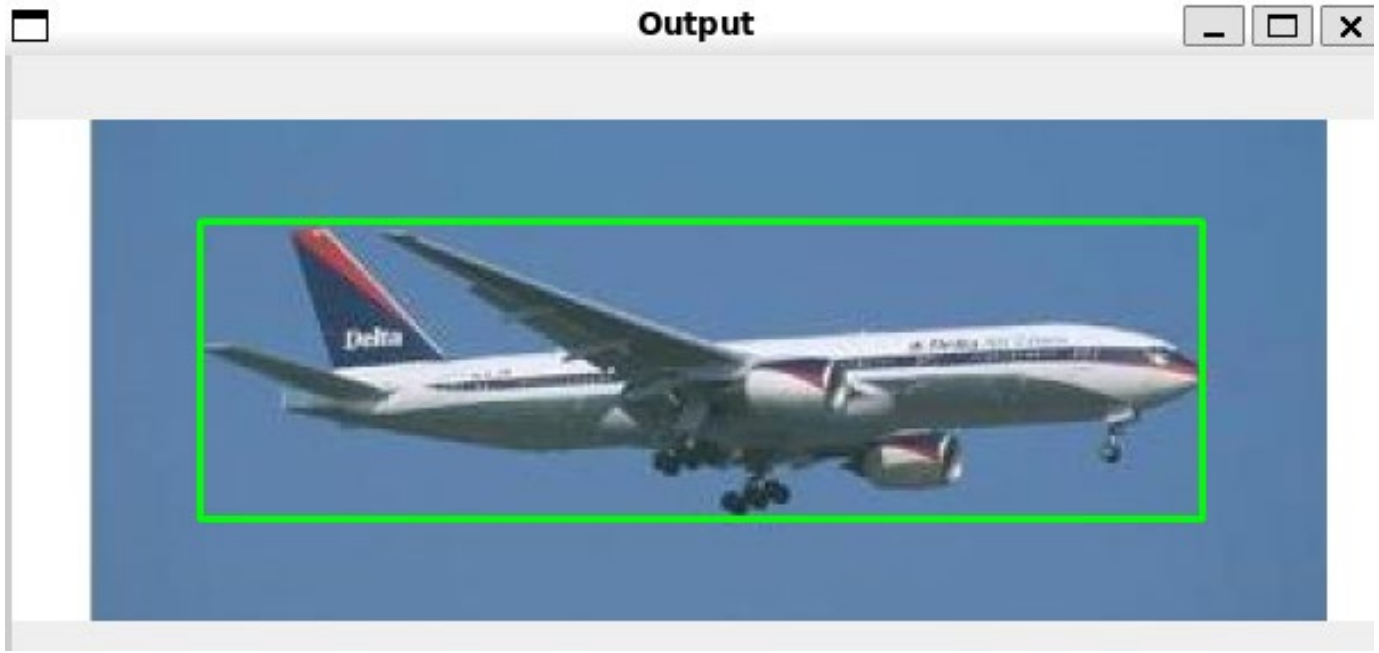
Bounding Box Detection

Noah Pritchard

What is a bounding box?

- Outline objects of interest
- Serve as point of reference
- Used in image processing



Implementation

- Given set of images
- Given bounding box coordinates for images
- Split into train and test images
- Use model to predict bounding box for a given image

```
54,27,347,100,image_0424.jpg
52,26,349,115,image_0764.jpg
50,24,349,121,image_0502.jpg
56,27,352,103,image_0601.jpg
56,29,352,113,image_0611.jpg
58,34,352,154,image_0791.jpg
76,28,340,138,image_0478.jpg
54,25,350,129,image_0501.jpg
62,29,350,115,image_0527.jpg
48,25,345,125,image_0767.jpg
```

Annotation

- Used the Caltech-101 dataset
- Annotations in MATLAB files
- Convert from MATLAB files to CSV format

```
with open(os.getcwd() + "/dataset/airplanes.csv", "w") as csv_file:
    for filename in os.listdir(dir):
        try:
            mat = loadmat(dir + "/" + filename)
            coords = [x for x in mat["box_coord"][0]]
            annotationIndex = filename.split("_")[1].split(".")[0]
            annotationTitle = f"image_{annotationIndex}.jpg"
            coords.append(annotationTitle)
            vals = np.asarray(coords)
            line = f"{vals[2]},{vals[0]},{vals[3]},{vals[1]},{vals[4]}\n"
            csv_file.write(line)
```

Training the Model

- Read in each image along with coordinates
- Create a train/test split
- Use keras VGG16 model
- Add layers and optimizer
- Train the model

Images and Coordinates

- Create two lists
- Read CSV file line by line
- Use OpenCV to save images
- Reshape images so they are consistent

```
print("loading dataset...")
rows = open(config.ANNOTATIONS_PATH).read().strip().split("\n")

images = []
coords = []
filenames = []

for row in rows:
    # for each row, parse the csv input
    # and get the coords/matching image
    row = row.split(",")
    (startX, startY, endX, endY, filename) = row
    imagePath = os.path.sep.join([config.IMAGES_PATH, filename.strip()])
    image = cv2.imread(imagePath)

    # scale bounding box coords to the image
    (h, w) = image.shape[:2]
    startX = float(startX) / w      Noah Pritchard, 20 hours ago • fixed coordinate er
    startY = float(startY) / h
    endX = float(endX) / w
    endY = float(endY) / h

    # load each image through keras and add image/coords/filename to arrays
    image = tf.keras.preprocessing.image.load_img(imagePath, target_size=(224, 224))
    image = tf.keras.preprocessing.image.img_to_array(image)

    images.append(image)
    coords.append((startX, startY, endX, endY))
    filenames.append(filename)

# scale the images by 255 (for pixels), leave coords untouched
images = np.array(images, dtype="float32") / 255.0
coords = np.array(coords, dtype="float32")
```

Train/Test Split

- sklearn built-in function
- Save testing filenames for later use
- Training data used in this file

```
# create training and testing splits for our model
split = train_test_split(images, coords, filenames, test_size=0.10, random_state=42)
(trainImages, testImages) = split[:2]
(trainTargets, testTargets) = split[2:4]
(trainFilenames, testFilenames) = split[4:]

# save the test filenames into our output for later use
print("saving testing filenames...")
f = open(config.TEST_FILENAMES, "w")
f.write("\n".join(testFilenames))
f.close()
```

VGG16 Model

- Premade image processing model
- Pass in shape to reflect size of image

```
# VGG16 is a premade model from keras that can be used for image classification
vgg = VGG16(
    weights="imagenet",
    include_top=False,
    input_tensor=tf.keras.layers.Input(shape=(224, 224, 3)),
)
```

Available models

Model	Size (MB)	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth	Time (ms) per inference step (CPU)	Time (ms) per inference step (GPU)
Xception	88	79.0%	94.5%	22.9M	81	109.4	8.1
VGG16	528	71.3%	90.1%	138.4M	16	69.5	4.2
VGG19	549	71.3%	90.0%	143.7M	19	84.8	4.4

Layers and Optimizer

- Multiple dense layers
  - 3 relu
  - 1 sigmoid
- Adam optimizer

```
# Flatten our data and add dense layers
flatten = tf.keras.layers.Flatten()(flatten)

bboxHead = tf.keras.layers.Dense(128, activation="relu")(flatten)
bboxHead = tf.keras.layers.Dense(64, activation="relu")(bboxHead)
bboxHead = tf.keras.layers.Dense(32, activation="relu")(bboxHead)
bboxHead = tf.keras.layers.Dense(4, activation="sigmoid")(bboxHead)

model = tf.keras.models.Model(inputs=vgg.input, outputs=bboxHead)

# Add an optimizer (Adam) to speed up our results
opt = tf.keras.optimizers.Adam(learning_rate=config.INIT_LR)
model.compile(loss="mse", optimizer=opt)
print(model.summary())
```

Training the Model

- Model.fit(…)
- Use predefined hyperparameters
- Save model so it can be used again later

```
# Train the model
print("training bounding box regressor...")
H = model.fit(
    trainImages,
    trainTargets,
    validation_data=(testImages, testTargets),
    batch_size=config.BATCH_SIZE,
    epochs=config.NUM_EPOCHS,
    verbose="auto",
)

# save the model to our output and plot the training data
print("saving object detector model...")
model.save(config.MODEL_PATH, save_format="h5")
```

Using Model for Predictions

- Take in input
- Load model
- Load images
- Collect and plot predicted bounding box

Inputs

- Pass in a single file or a list of files
- List of files → add every file name to a list

```
# add argument parsing so we can pass in a file or directory
# to run the model on
# Example: python3 predict.py --input image.jpg
ap = argparse.ArgumentParser()
ap.add_argument(
    "-i",
    "--input",
    required=True,
    help="path to input image/text file of image filenames",
)
```

```
if filetype == "text/plain":
    filenames = open(args["input"]).read().strip().split("\n")
    imagePaths = []

    for f in filenames:
        p = os.path.sep.join([config.IMAGES_PATH, f])
        imagePaths.append(p)
```

Loading the model

- Save file after training
- Load file when predicting

```
# save the model to our output and plot the training data
print("saving object detector model...")
model.save(config.MODEL_PATH, save_format="h5")
```

```
# load the model from memory
print("loading image detector...")
model = tf.keras.models.load_model(config.MODEL_PATH)
```

Loading Image

- Use keras to load and shape the images
- Same load_img and img_to_array as in training

```
# load each image, preprocess/scale them as our model expects
# this means they must match our training data
for ip in imagePaths:
    image = tf.keras.preprocessing.image.load_img(ip, target_size=(224, 224))
    image = tf.keras.preprocessing.image.img_to_array(image) / 255.0
    image = np.expand_dims(image, axis=0)
```

Collect and Plot bounding box predictions

- Model.predict(image)
- Use OpenCV again to load the image
- Resize bounding box coordinates to match image
- Use OpenCV to plot the image along with the bounding box

```
preds = model.predict(image)[0]
(startX, startY, endX, endY) = preds

image = cv2.imread(ip)
image = imutils.resize(image, width=600)
(h, w) = image.shape[:2]

# get the predited coordinates and plot them on top of the image
# when this file is executed, it will open an output dialogue
# showing the image and the bounding box

# if you pass in a single file, press any key to end the program
# if you pass in a text file containing multiple files,
# pressing any key will move you to the next image
# until all images have been seen, then the program will end
startX = int(startX * w)
startY = int(startY * h)
endX = int(endX * w)
endY = int(endY * h)

cv2.rectangle(image, (startX, startY), (endX, endY), (0, 255, 0), 2)

cv2.imshow("Output", image)
cv2.waitKey(0)
```