RCNN (Region-based Convolutional Neural Networks)

a family of deep learning models developed for object detection tasks. RCNNs generate region proposals and classify them into object classes, localizing objects within an image. Breakdown of RCNN and its evolution

Implementing our object detection dataset builder script

Launch Jupyter Nootebook on Google Colab

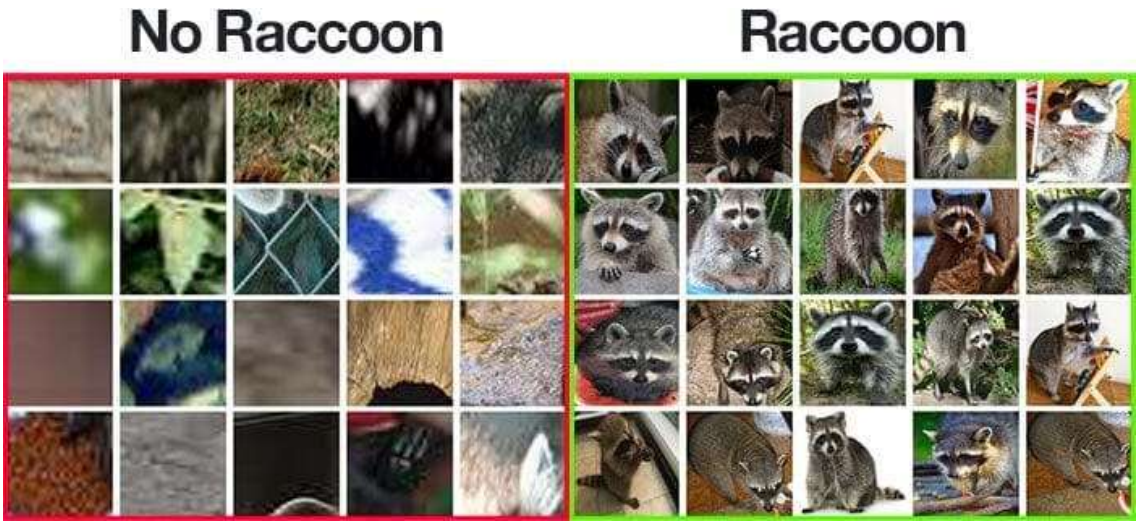
R-CNN object detection with Keras, TensorFlow, and Deep Learning

```
# import the necessary packages
from pyimagesearch.iou import compute_iou
from pyimagesearch import config
from bs4 import BeautifulSoup
from imutils import paths
import cv2
import os

# loop over the output positive and negative directories
for dirPath in (config.POSITVE_PATH, config.NEGATIVE_PATH):
# if the output directory does not exist yet, create it
if not os.path.exists(dirPath):
os.makedirs(dirPath)
# grab all image paths in the input images directory
imagePaths = list(paths.list_images(config.ORIG_IMAGES))
# initialize the total number of positive and negative images we have
# saved to disk so far
totalPositive = 0
totalNegative = 0

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
# show a progress report
print("[INFO] processing image {}/{}...".format(i + 1,
len(imagePaths)))
# extract the filename from the file path and use it to derive
# the path to the XML annotation file
filename = imagePath.split(os.path.sep)[-1]
filename = filename[:filename.rfind(".")]
annotPath = os.path.sep.join([config.ORIG_ANNOTS,
"{}.xml".format(filename)])
# load the annotation file, build the soup, and initialize our
# list of ground-truth bounding boxes
contents = open(annotPath).read()
soup = BeautifulSoup(contents, "html.parser")
gtBoxes = []
# extract the image dimensions
w = int(soup.find("width").string)
```

```
h = int(soup.find("height").string)

# loop over all 'object' elements
for o in soup.find_all("object"):
# extract the label and bounding box coordinates
label = o.find("name").string
xMin = int(o.find("xmin").string)
yMin = int(o.find("ymin").string)
xMax = int(o.find("xmax").string)
yMax = int(o.find("ymax").string)
# truncate any bounding box coordinates that may fall
# outside the boundaries of the image
xMin = max(0, xMin)
yMin = max(0, yMin)
xMax = min(w, xMax)
yMax = min(h, yMax)
# update our list of ground-truth bounding boxes
gtBoxes.append((xMin, yMin, xMax, yMax))
# load the input image from disk
image = cv2.imread(imagePath)
# run selective search on the image and initialize our list of
# proposed boxes
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(image)
ss.switchToSelectiveSearchFast()
rects = ss.process()
proposedRects= []
# loop over the rectangles generated by selective search
for (x, y, w, h) in rects:
# convert our bounding boxes from (x, y, w, h) to (startX,
# startY, startX, endY)
proposedRects.append((x, y, x + w, y + h))

# initialize counters used to count the number of positive and
# negative ROIs saved thus far
positiveROIs = 0
negativeROIs = 0
# loop over the maximum number of region proposals
for proposedRect in proposedRects[:config.MAX_PROPOSALS]:
# unpack the proposed rectangle bounding box
(propStartX, propStartY, propEndX, propEndY) = proposedRect
# loop over the ground-truth bounding boxes
for gtBox in gtBoxes:
# compute the intersection over union between the two
# boxes and unpack the ground-truth bounding box
iou = compute_iou(gtBox, proposedRect)
(gtStartX, gtStartY, gtEndX, gtEndY) = gtBox
# initialize the ROI and output path
```

```
roi = None
outputPath = None

# check to see if the IOU is greater than 70% *and* that
# we have not hit our positive count limit
if iou > 0.7 and positiveROIs <= config.MAX_POSITIVE:
# extract the ROI and then derive the output path to
# the positive instance
roi = image[propStartY:propEndY, propStartX:propEndX]
filename = "{}.png".format(totalPositive)
outputPath = os.path.sep.join([config.POSITVE_PATH,
filename])
# increment the positive counters
positiveROIs += 1
totalPositive += 1

# check to see if the IOU is greater than 70% *and* that
# we have not hit our positive count limit
if iou > 0.7 and positiveROIs <= config.MAX_POSITIVE:
# extract the ROI and then derive the output path to
# the positive instance
roi = image[propStartY:propEndY, propStartX:propEndX]
filename = "{}.png".format(totalPositive)
outputPath = os.path.sep.join([config.POSITVE_PATH,
filename])
# increment the positive counters
positiveROIs += 1
totalPositive += 1

# check to see if there is not full overlap *and* the IoU
# is less than 5% *and* we have not hit our negative
# count limit
if not fullOverlap and iou < 0.05 and \
negativeROIs <= config.MAX_NEGATIVE:
# extract the ROI and then derive the output path to
# the negative instance
roi = image[propStartY:propEndY, propStartX:propEndX]
filename = "{}.png".format(totalNegative)
outputPath = os.path.sep.join([config.NEGATIVE_PATH,
filename])
# increment the negative counters
negativeROIs += 1
totalNegative += 1

# check to see if both the ROI and output path are valid
if roi is not None and outputPath is not None:
# resize the ROI to the input dimensions of the CNN
# that we'll be fine-tuning, then write the ROI to
```

```
# disk
roi = cv2.resize(roi, config.INPUT_DIMS,
interpolation=cv2.INTER_CUBIC)
cv2.imwrite(outputPath, roi)

$ time python build_dataset.py
[INFO] processing image 1/200...
[INFO] processing image 2/200...
[INFO] processing image 3/200...
...
[INFO] processing image 198/200...
[INFO] processing image 199/200...
[INFO] processing image 200/200...
real 5m42.453s
user 6m50.769s
sys 1m23.245s

$ ls -l dataset/raccoon/*.png | wc -l
1560
$ ls -l dataset/no_raccoon/*.png | wc -l
2200
```



Fine-tuning a network for object detection with Keras and TensorFlow

```
# import the necessary packages
from pyimagesearch import config
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
```

```
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import pickle

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--plot", type=str, default="plot.png",
help="path to output loss/accuracy plot")
args = vars(ap.parse_args())
# initialize the initial learning rate, number of epochs to train for,
# and batch size
INIT_LR = 1e-4
EPOCHS = 5
BS = 32

# grab the list of images in our dataset directory, then initialize
# the list of data (i.e., images) and class labels
print("[INFO] loading images...")
imagePaths = list(paths.list_images(config.BASE_PATH))
data = []
labels = []
# loop over the image paths
for imagePath in imagePaths:
# extract the class label from the filename
label = imagePath.split(os.path.sep)[-2]
# load the input image (224x224) and preprocess it
image = load_img(imagePath, target_size=config.INPUT_DIMS)
image = img_to_array(image)
image = preprocess_input(image)
# update the data and labels lists, respectively
data.append(image)
labels.append(label)

# convert the data and labels to NumPy arrays
data = np.array(data, dtype="float32")
labels = np.array(labels)
# perform one-hot encoding on the labels
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
```

```
labels = to_categorical(labels)
# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.20, stratify=labels, random_state=42)
# construct the training image generator for data augmentation
aug = ImageDataGenerator(
rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")

# load the MobileNetV2 network, ensuring the head FC layer sets are
# left off
baseModel = MobileNetV2(weights="imagenet", include_top=False,
input_tensor=Input(shape=(224, 224, 3)))
# construct the head of the model that will be placed on top of the
# the base model
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
# place the head FC model on top of the base model (this will become
# the actual model we will train)
model = Model(inputs=baseModel.input, outputs=headModel)
# loop over all layers in the base model and freeze them so they will
# *not* be updated during the first training process
for layer in baseModel.layers:
layer.trainable = False

# compile our model
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])
# train the head of the network
print("[INFO] training head...")
H = model.fit(
aug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)
```

```
# make predictions on the testing set
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)
# for each image in the testing set we need to find the index of the
# label with corresponding largest predicted probability
predIdxs = np.argmax(predIdxs, axis=1)
# show a nicely formatted classification report
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

# serialize the model to disk
print("[INFO] saving mask detector model...")
model.save(config.MODEL_PATH, save_format="h5")
# serialize the label encoder to disk
print("[INFO] saving label encoder...")
f = open(config.ENCODER_PATH, "wb")
f.write(pickle.dumps(lb))
f.close()

# plot the training loss and accuracy
N = EPOCHS
plt.style.use("ggplot")
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Training our R-CNN object detection network with Keras and TensorFlow

```
$ time python fine_tune_rcnn.py
[INFO] loading images...
[INFO] compiling model...
[INFO] training head...
Train for 94 steps, validate on 752 samples
Train for 94 steps, validate on 752 samples
Epoch 1/5
94/94 [==============================] - 77s 817ms/step - loss: 0.3072 - accuracy: 0.8647 -
val_loss: 0.1015 - val_accuracy: 0.9728
Epoch 2/5
94/94 [==============================] - 74s 789ms/step - loss: 0.1083 - accuracy: 0.9641 -
val_loss: 0.0534 - val_accuracy: 0.9837
Epoch 3/5
```

94/94 [==============================] - 71s 756ms/step - loss: 0.0774 - accuracy: 0.9784 - val_loss: 0.0433 - val_accuracy: 0.9864
Epoch 4/5
94/94 [==============================] - 74s 784ms/step - loss: 0.0624 - accuracy: 0.9781 - val_loss: 0.0367 - val_accuracy: 0.9878
Epoch 5/5
94/94 [==============================] - 74s 791ms/step - loss: 0.0590 - accuracy: 0.9801 - val_loss: 0.0340 - val_accuracy: 0.9891
[INFO] evaluating network...

	precision	recall	f1-score	support
no_raccoon	1.00	0.98	0.99	440
raccoon	0.97	1.00	0.99	312
accuracy			0.99	752
macro avg	0.99	0.99	0.99	752
weighted avg	0.99	0.99	0.99	752

[INFO] saving mask detector model...
[INFO] saving label encoder...
real 6m37.851s
user 31m43.701s
sys 33m53.058s

Putting the pieces together: Implementing our R-CNN object detection inference script

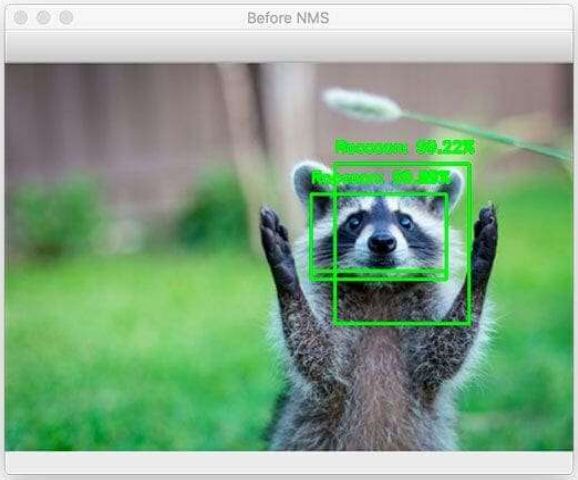
```
# import the necessary packages
from pyimagesearch.nms import non_max_suppression
from pyimagesearch import config
```

```
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import numpy as np
import argparse
import imutils
import pickle
import cv2
# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
help="path to input image")
args = vars(ap.parse_args())

# load the our fine-tuned model and label binarizer from disk
print("[INFO] loading model and label binarizer...")
model = load_model(config.MODEL_PATH)
lb = pickle.loads(open(config.ENCODER_PATH, "rb").read())
# load the input image from disk
image = cv2.imread(args["image"])
image = imutils.resize(image, width=500)
# run selective search on the image to generate bounding box proposal
# regions
print("[INFO] running selective search...")
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(image)
ss.switchToSelectiveSearchFast()
rects = ss.process()

# initialize the list of region proposals that we'll be classifying
# along with their associated bounding boxes
proposals = []
boxes = []
# loop over the region proposal bounding box coordinates generated by
# running selective search
for (x, y, w, h) in rects[:config.MAX_PROPOSALS_INFER]:
# extract the region from the input image, convert it from BGR to
# RGB channel ordering, and then resize it to the required input
# dimensions of our trained CNN
roi = image[y:y + h, x:x + w]
roi = cv2.cvtColor(roi, cv2.COLOR_BGR2RGB)
roi = cv2.resize(roi, config.INPUT_DIMS,
interpolation=cv2.INTER_CUBIC)
# further preprocess the ROI
roi = img_to_array(roi)
roi = preprocess_input(roi)
# update our proposals and bounding boxes lists
proposals.append(roi)
```

```
boxes.append((x, y, x + w, y + h))

# convert the proposals and bounding boxes into NumPy arrays
proposals = np.array(proposals, dtype="float32")
boxes = np.array(boxes, dtype="int32")
print("[INFO] proposal shape: {}".format(proposals.shape))
# classify each of the proposal ROIs using fine-tuned model
print("[INFO] classifying proposals...")
proba = model.predict(proposals)

# clone the original image so that we can draw on it
clone = image.copy()
# loop over the bounding boxes and associated probabilities
for (box, prob) in zip(boxes, proba):
# draw the bounding box, label, and probability on the image
(startX, startY, endX, endY) = box
cv2.rectangle(clone, (startX, startY), (endX, endY),
(0, 255, 0), 2)
y = startY - 10 if startY - 10 > 10 else startY + 10
text= "Raccoon: {:.2f}%".format(prob * 100)
cv2.putText(clone, text, (startX, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
# show the output after *before* running NMS
cv2.imshow("Before NMS", clone)

# run non-maxima suppression on the bounding boxes
boxIdxs = non_max_suppression(boxes, proba)
# loop over the bounding box indexes
for i in boxIdxs:
# draw the bounding box, label, and probability on the image
(startX, startY, endX, endY) = boxes[i]
cv2.rectangle(image, (startX, startY), (endX, endY),
(0, 255, 0), 2)
y = startY - 10 if startY - 10 > 10 else startY + 10
text= "Raccoon: {:.2f}%".format(proba[i] * 100)
cv2.putText(image, text, (startX, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 255, 0), 2)
# show the output image *after* running NMS
cv2.imshow("After NMS", image)
cv2.waitKey(0)
```

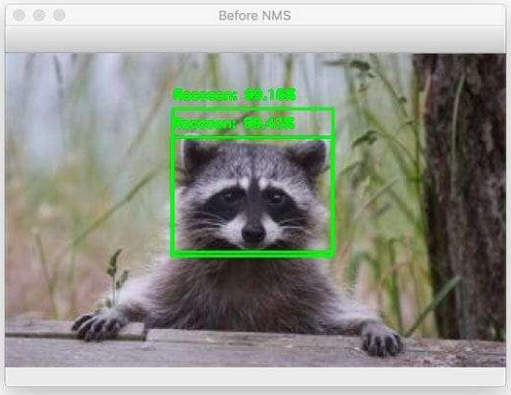
R-CNN object detection results using Keras and TensorFlow

```
$ python detect_object_rcnn.py --image images/raccoon_01.jpg
[INFO] loading model and label binarizer...
[INFO] running selective search...
[INFO] proposal shape: (200, 224, 224, 3)
[INFO] classifying proposals...
[INFO] applying NMS...
```

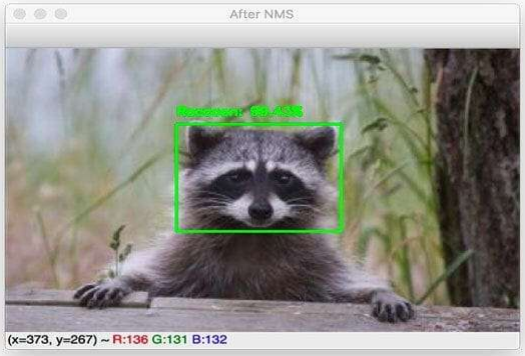


By applying non-maxima suppression, we can suppress the weaker one, leaving with the one correct bounding box:


```
$ python detect_object_rcnn.py --image images/raccoon_02.jpg
[INFO] loading model and label binarizer...
[INFO] running selective search...
[INFO] proposal shape: (200, 224, 224, 3)
[INFO] classifying proposals...
[INFO] applying NMS...
```



Applying non-maxima suppression to our R-CNN object detection output leaves us with the final object detection:



$ python detect_object_rcnn.py --image images/raccoon_03.jpg
[INFO] loading model and label binarizer...
[INFO] running selective search...
[INFO] proposal shape: (200, 224, 224, 3)
[INFO] classifying proposals...
[INFO] applying NMS...