Get started Open in app

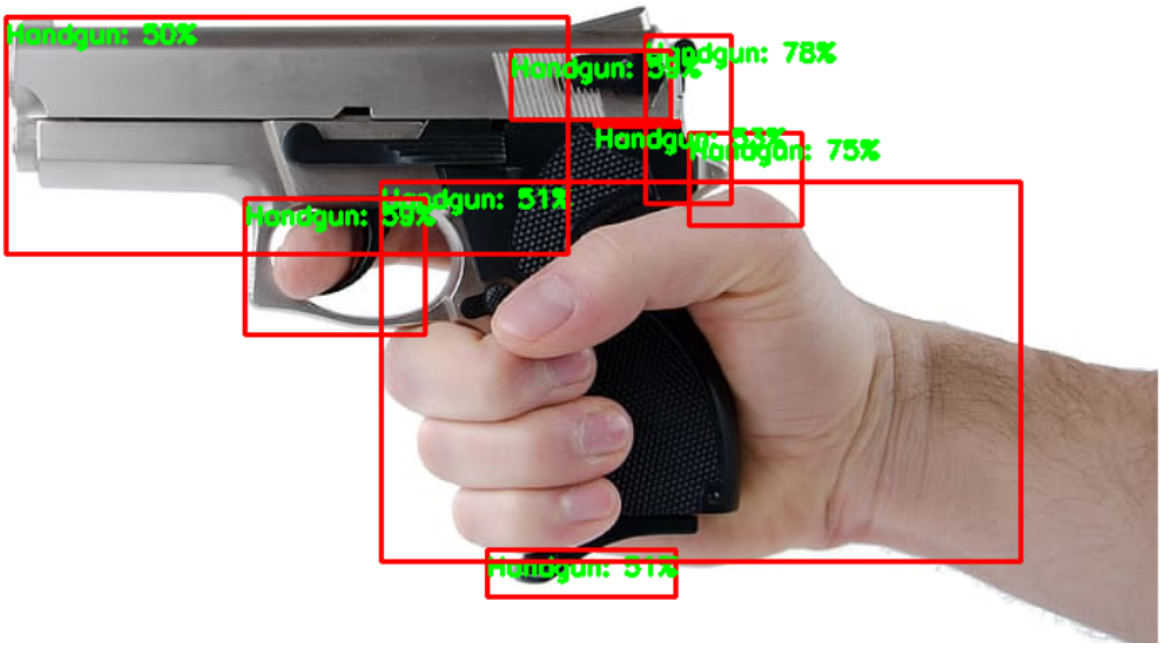
Follow 555K Followers

You have **2** free member-only stories left this month. Sign up for Medium and get an extra one

Custom Object Detection Using Keras and OpenCV

Build a System That Can Identify a Weapon Within a Given Image or Frame

Samuel Mohebban Sep 9, 2020 · 9 min read ★



Get started Open in app

I recently completed a project I am very proud of and figured I should share it in case anyone else is interested in implementing something similar to their specific needs. Before I get started in the tutorial, I want to give a HEFTY thanks to Adrian Rosebrock, PhD, creator of PyImageSearch. I am a self-taught programmer, so without his resources, much of this project would not be possible. He is the epitome of a *mensch*- I could not be more appreciative of the resources he puts on his website. If you want to learn advanced deep learning techniques but find textbooks and research papers dull, I highly recommend visiting his website linked above.

In most projects related to weapon classification, I was only able to find a dataset of 100–200 images maximum. This posed an issue because, from my experience, it is hard to get a working model with so little images. To gather images, I rigged my raspberry pi to scrape IMFDB.com- a website where gun enthusiasts post pictures where a model gun is featured in a frame or clip from a movie. If you visit the website, this will be more clear. To access the images that I used, you can visit my Google Drive. In this zip file, you will find all the images that were used in this project and the corresponding .xml files for the bounding boxes. If you are in need of bounding boxes for a large dataset, I highly recommend ScaleOps.AI, a company that specializes in data labeling for machine learning algorithms. Currently, I have 120,000 images from the IMFDB website, but for this project, I only used ~5000 due to time and money constraints.

Now, let's get to the logic. The architecture of this project follows the logic shown on this website. Although we implement the logic here, there are many areas for which it is different so that it can be useful for our specific problem — detecting weapons. The project uses 6 basic steps:

1. Build a dataset using OpenCV Selective search segmentation
2. Build a CNN for detecting the objects you wish to classify (in our case this will be 0 = No Weapon, 1 = Handgun, and 2 = Rifle)

Get started Open in app

4. When creating a bounding box for a new image, run the image through the selective search segmentation, then grab every piece of the picture.
5. Run each piece of an image through the algorithm, and whenever the algorithm predicts the object you are looking for mark the locations with a bounding box
6. If multiple bounding boxes are marked, apply Non-Maxima suppression to include only the box with the high confidence/region of interest (this part I am still figuring out… you will see my issue below)

Below is a gif showing how the algorithm works. For a given image, each square will be fed into the neural network. If a square is predicted as positive (handgun or rifle), we will mark the area that we fed onto the original image.

Sliding Window Approach: Object Detection (Image by Author)

If you want to see the entire code for the project, visit my GitHub Repo where I explain the steps in greater depth.

The data I linked above contains a lot of folders that I need to explain in order to understand whats going on. After unzipping the folder, these are the files & folders that

Get started Open in app

pertaining to the folder name. So for the AR folder, you will find images of Assault rifles inside. Inside the Labels folder, you will see the .xml labels for all the images inside the class folders. Lastly, the PATHS.csv will point to every single image that will be used in the algorithm. For the purpose of this tutorial these are the only folders/files you need to worry about:

1. FinalImages/NoWeapon
2. FinalImages/Pistol
3. FinalImages/Rifle

The way the images within these folders were made is the following.

- For every image with a bounding box, extract the bounding box and put it into its corresponding class folder. So for an image where a person is holding a pistol, the bounding box around the pistol will become positive, while every part outside the bounding box will become the negative (no weapon)
- In the image below, imagine a bounding box around the image on the left. After extracting the pixels inside the bounding box (image on the right), we place that image to another folder (FinalImages/Pistol), while we place all the white space around the bounding box in the NoWeapons folder.
- Although the image on the right looks like a resized version of the one on the left, it is really a segmented image. Picture a bounding box around the gun on the left. The image on the right is *just* the bounding box and nothing else (removing everything outside the box). This technique is called region of interest (ROI).

ROI Extraction (Image by Author)

Get started Open in app

have a list of RGB values and the corresponding label (0= No Weapon, 1 = Pistol, 2 = Rifle)

```
import pandas as pd
import os
import numpy as np
import cv2
from keras.preprocessing import image
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from keras.utils import to_categorical
from skimage.segmentation import mark_boundaries

def get_image_value(path, dim):
    '''This function will read an image and convert to a specified version and resize depending c
    img = image.load_img(path, target_size = dim)
    img = image.img_to_array(img)
    return img/255

def get_img_array(img_paths, dim):
    '''This fucntion takes a list of image paths and returns the np array corresponding to each i
    final_array = []
    from tqdm import tqdm
    for path in tqdm(img_paths):
        img = get_image_value(path, dim)
        final_array.append(img)
    final_array = np.array(final_array)
    return final_array

def get_tts():
    '''This function will create a train test split'''
   '''
    DIM =  (150,150)
    np.random.seed(10)
    pistol_paths = [f'../Separated/FinalImages/Pistol/{i}' for i in os.listdir('../Separated/Fina
    pistol_labels = [1 for i in range(len(pistol_paths))]
    rifle_paths = [f'../Separated/FinalImages/Rifle/{i}' for i in os.listdir('../Separated/Final
    rifle_labels = [2 for i in range(len(rifle_paths))]
    neg_paths = [f'../Separated/FinalImages/NoWeapon/{i}' for i in os.listdir('../Separated/Final
    np.random.shuffle(neg_paths)
    neg_paths = neg_paths[:len(pistol_paths)- 500]
```

Get started Open in app

```
    pistol_paths = pistol_paths[:len(rifle_paths)+150]
    neg_paths = neg_paths[:len(rifle_paths)+150]

    pistol_labels = [1 for i in range(len(pistol_paths))]
    rifle_labels = [2 for i in range(len(rifle_paths))]
    neg_labels = [0 for i in range(len(neg_paths))]
    paths = pistol_paths + rifle_paths + neg_paths
    labels = pistol_labels + rifle_labels + neg_labels
    x_train, x_test, y_train, y_test = train_test_split(paths, labels, stratify = labels, train_s

    new_x_train = get_img_array(x_train, DIM)
    new_x_test = get_img_array(x_test, DIM)

    print('Train Value Counts')
    print(pd.Series(y_train).value_counts())
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    print('Test Value Counts')
    print(pd.Series(y_test).value_counts())
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    print('X Train Shape')
    print(new_x_train.shape)
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    print('X Test Shape')
    print(new_x_test.shape)
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')

    y_train = np.array(y_train)
    y_test = np.array(y_test)
    y_test = to_categorical(y_test)
    y_train = to_categorical(y_train)
    tts = (new_x_train, new_x_test, y_train, y_test)
    return tts

x_train, x_test, y_train, y_test = get_tts()

#uncomment the code below to see what the images look like
#cv2.imshow('test', x_train[25])
# cv2.waitKey(0)
# cv2.destroyAllWindows()
```

tts hosted with ❤️ by GitHub view raw

Get started Open in app

you should see this:

```
100%|| 3969/3969 [00:45<00:00, 87.60it/s]
100%|| 441/441 [00:04<00:00, 96.42it/s]
Train Value Counts
1    1368
0    1368
2    1233
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Test Value Counts
1    152
0    152
2    137
dtype: int64
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
X Train Shape
(3969, 150, 150, 3)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
X Test Shape
(441, 150, 150, 3)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

(Image by Author)

Now its time for the neural network. In the code below, the function will return a model given a dimension size. If you noticed in the code above, the dimensions for the photos were resized to (150, 150, 3). If you wish to use different dimensions just make sure you change the variable DIM above, as well as the dim in the function below

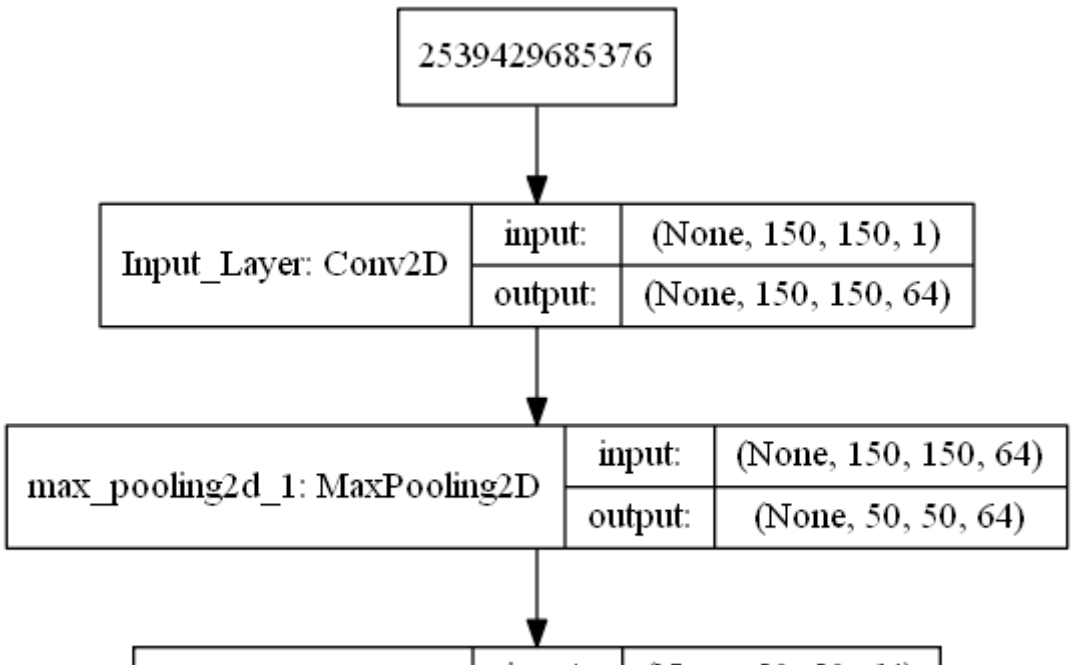
```
import numpy as np
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, AveragePooling2D, Dense, Drop
from keras.optimizers import Adam
from keras import regularizers
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
import pandas as pd
import matplotlib.pyplot as plt
import os
import pickle
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
import cv2

def get_conv_model(dim = (150,150, 3)):
    '''This function will create and compile a CNN given the input dimension'''
    inp_shape = dim
    act = 'relu'
    drop = .25
```

Get started Open in app

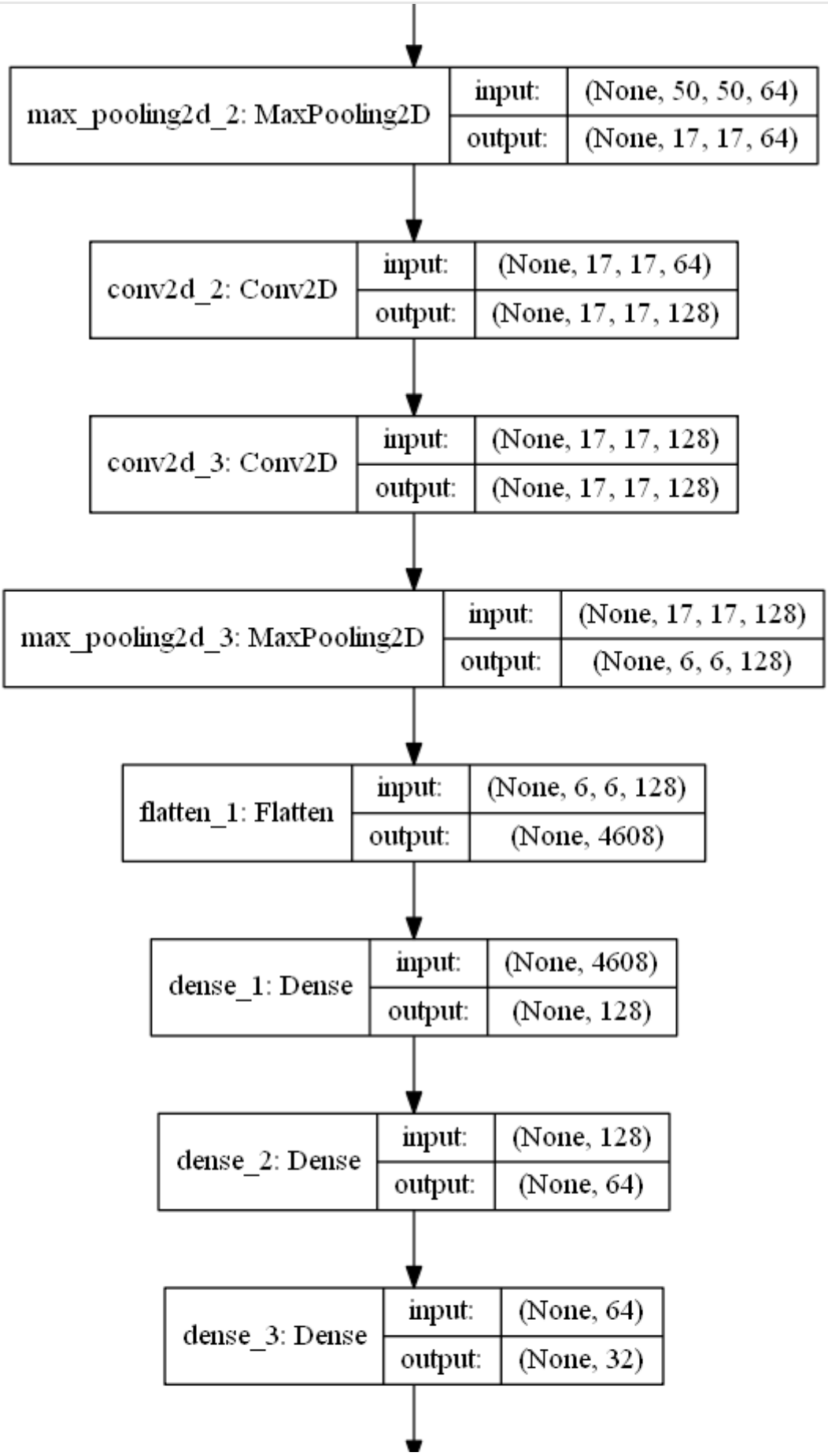
```
model = Sequential()
model.add(Conv2D(64, kernel_size=(3,3),activation=act, input_shape = inp_shape,
                 kernel_regularizer = kernal_reg,
                 kernel_initializer = 'he_uniform',  padding = 'same', name = 'Input_Layer')
model.add(MaxPooling2D(pool_size=(2, 2),  strides = (3,3)))
model.add(Conv2D(64, (3, 3), activation=act, kernel_regularizer = kernal_reg,
                 kernel_initializer = 'he_uniform',padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides = (3,3)))
model.add(Conv2D(128, (3, 3), activation=act, kernel_regularizer = kernal_reg,
                 kernel_initializer = 'he_uniform',padding = 'same'))
model.add(Conv2D(128, (3, 3), activation=act, kernel_regularizer = kernal_reg,
                 kernel_initializer = 'he_uniform',padding = 'same'))
model.add(MaxPooling2D(pool_size=(2, 2), strides = (3,3)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dropout(drop))
model.add(Dense(3, activation='softmax', name = 'Output_Layer'))
model.compile(loss = 'categorical_crossentropy', optimizer = optimizer, metrics = ['accuracy
return model
```

WeaponCNN hosted with ❤️ by GitHub view raw

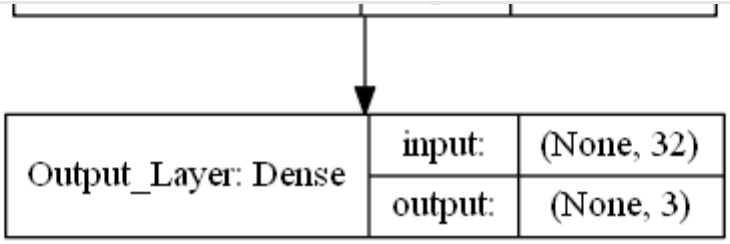
The model returned above will have the architecture shown below:



Get started Open in app



Get started Open in app



CNN Architecture (Image by Author)

Once we have our train and test sets, all we need to do is fit it onto our model. Running the code below will start the training process.

```
#prevents overfitting and saves models every time the validation loss improves
early_stopping = EarlyStopping(monitor='val_loss', verbose = 1, patience=10, min_delta = .00075)
model_checkpoint = ModelCheckpoint('ModelWeights.h5', verbose = 1, save_best_only=True,
                                   monitor = 'val_loss')
lr_plat = ReduceLROnPlateau(patience = 2, mode = 'min')
epochs = 1000
batch_size = 32
model = get_conv_model()
model_history = model.fit(x_train, y_train, batch_size = batch_size,
             epochs = epochs,
      callbacks = [early_stopping, model_checkpoint, lr_plat], validation_data = (x_test, y_test)
```

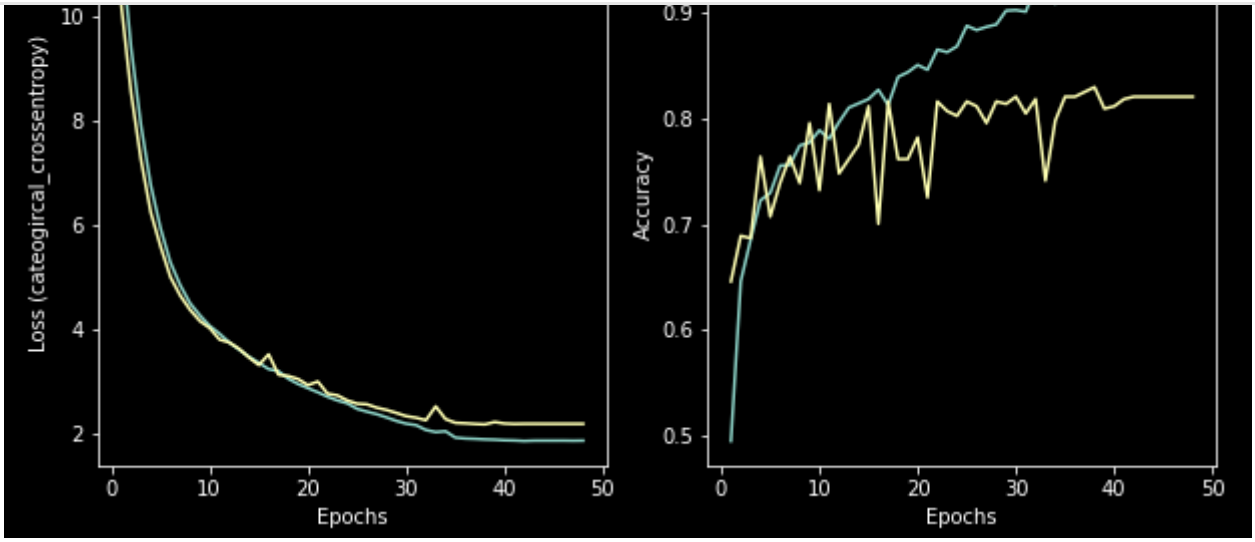
training hosted with ❤️ by GitHub view raw

If you run the code without any errors, you should see a window like this:

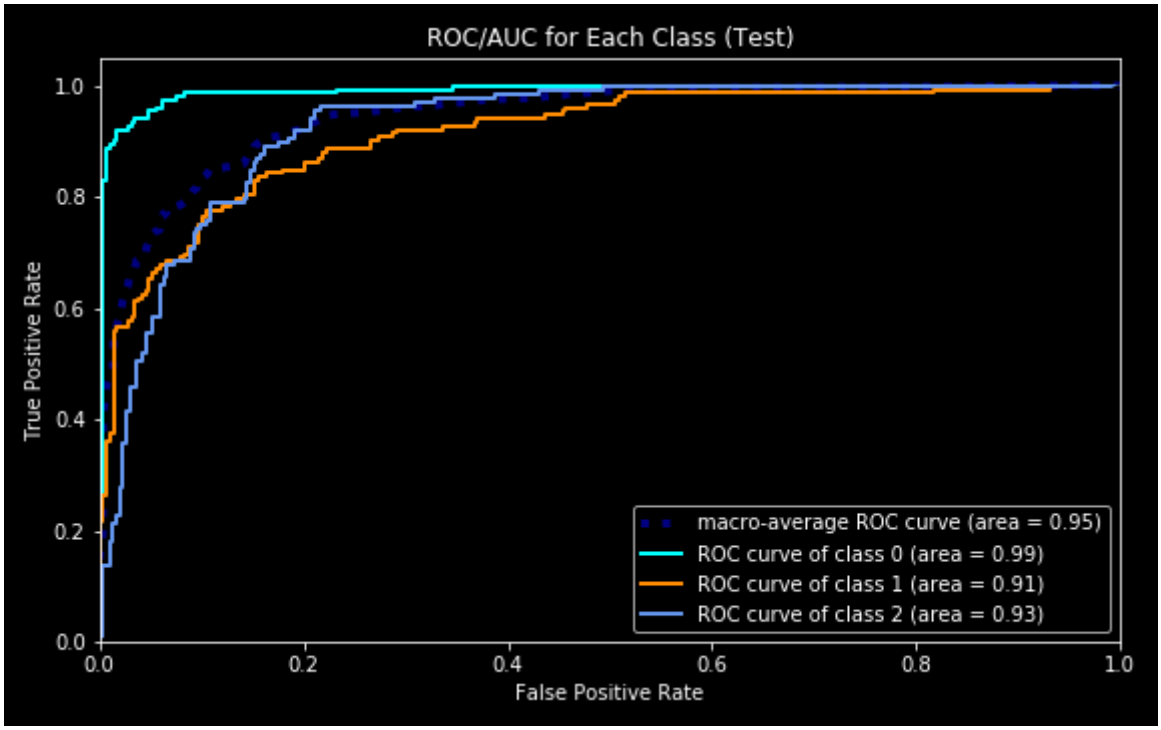
```
Train on 3969 samples, validate on 441 samples
Epoch 1/1000
INFO:plaidml:Analyzing Ops: 90 of 422 operations complete
2816/3969 [====================>.........] - ETA: 9s - loss: 11.6514 - acc: 0.4229
```

I want to note that I have the epochs set to 1000, but the EarlyStopping will prevent the algorithm from overfitting so it should not run for longer than 30–50 epochs. After the model is finished, you should see a .h5 file in your directory called ModelWeights.h5. This file is the weights that the model produced, so loading these into a model will load the model before it started to overfit.

Get started Open in app



Accuracy of the Model (Image by Author)



ROC For each Class (Image by Author)

The accuracy was pretty good considering a balanced data set. Looking at the ROC curve, we can also assume pretty good classification given that the area under each class is very close to 1.

Now time for object detection! The following logic is used to create the bounding boxes:

Get started Open in app

2. Apply selective search segmentation to create hundreds of thousands of bounding box propositions

3. Run each bounding box through the trained algorithm and retrieve the locations where the prediction is the same as the base predictions (in step 1)

4. After retrieving the locations where the algorithm predicted the same as the base prediction, mark a bounding box on the location that was run through the algorithm

5. If multiple bounding boxes are chosen, apply non-maxima suppression to suppress all but one box, leaving the box with the highest probability and best Region of Interest (ROI)

- Note: Non-maxima suppression is still a work in progress. In some instances, it can only detect features of the gun rather than the entire gun itself (see model comparisons below).

```
def non_max_suppression(boxes, overlapThresh= .5):
    '''This image was taken from PyImageSearch... again cannot thank that guy enough'''
    # if there are no boxes, return an empty list
    if len(boxes) == 0:
        return []
    # if the bounding boxes integers, convert them to floats --
    # this is important since we'll be doing a bunch of divisions
    if boxes.dtype.kind == "i":
        boxes = boxes.astype("float")
    # initialize the list of picked indexes
    pick = []
    # grab the coordinates of the bounding boxes
    x1, y1, x2, y2 = boxes[:,0], boxes[:,1], boxes[:,2], boxes[:,3]
    # compute the area of the bounding boxes and sort the bounding
    # boxes by the bottom-right y-coordinate of the bounding box
    area = (x2 - x1 + 1) * (y2 - y1 + 1)
    idxs = np.argsort(y2)
    # keep looping while some indexes still remain in the indexes
    # list
    while len(idxs) > 0:
        # grab the last index in the indexes list and add the
        # index value to the list of picked indexes
        last = len(idxs) - 1
```

Get started Open in app

```
        # the bounding box and the smallest (x, y) coordinates
        # for the end of the bounding box
        xx1, yy1, xx2, yy2 = np.maximum(x1[i], x1[idxs[:last]]), np.maximum(y1[i], y1[idxs[:last
        # compute the width and height of the bounding box
        w, h = np.maximum(0, xx2 - xx1 + 1), np.maximum(0, yy2 - yy1 + 1)
        # compute the ratio of overlap
        overlap = (w * h) / area[idxs[:last]]
        # delete all indexes from the index list that have
        idxs = np.delete(idxs, np.concatenate(([last],
            np.where(overlap > overlapThresh)[0])))
    # return only the bounding boxes that were picked using the
    # integer data type
    return pick

def get_img_prediction_bounding_box(path, model, dim):
    '''This function will create a bounding box over what it believes is a weapon given the imag
    img = get_image_value(path, dim)
    img = img.reshape(1, img.shape[0], img.shape[1], 3)
    pred = model.predict(img)[0]
    category_dict = {0: 'No Weapon', 1: 'Handgun', 2: 'Rifle'}
    cat_index = np.argmax(pred)
    cat = category_dict[cat_index]
    print(f'{path}\t\tPrediction: {cat}\t{int(pred.max()*100)}% Confident')

    #speed up cv2
    cv2.setUseOptimized(True)
    cv2.setNumThreads(10) #change depending on your computer
    img = cv2.imread(path)
    clone = img.copy()
    clone2 = img.copy()
    ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
    ss.setBaseImage(img)
    ss.switchToSelectiveSearchFast()

    rects = ss.process()
    windows = []
    locations = []
    print(f'Creating Bounding Boxes for {path}')
    for x, y, w,h in rects[:1001]:
        startx, starty, endx, endy = x, y, x+w, y+h
        roi = img[starty:endy, startx:endx]
```

Get started Open in app

```
    windows = np.array(windows)
    windows = windows.reshape(windows.shape[0], windows.shape[1], windows.shape[2], 3)
    windows = np.array(windows)
    locations = np.array(locations)
    predictions = model.predict(windows)
    nms = non_max_suppression(locations)
    bounding_cnt = 0
    for idx in nms:
        if np.argmax(predictions[idx]) != cat_index:
            continue
        startx, starty, endx, endy = locations[idx]
        cv2.rectangle(clone, (startx, starty), (endx, endy), (0,0,255), 2)
        text = f'{category_dict[np.argmax(predictions[idx])]}: {int(predictions[idx].max()*100)
        cv2.putText(clone, text, (startx, starty+15), cv2.FONT_HERSHEY_SIMPLEX, .5, (0,255,0),2
        bounding_cnt += 1

    if bounding_cnt == 0:
        pred_idx= [idx for idx, i in enumerate(predictions) if np.argmax(i) == cat_index]
        cat_locations = np.array([locations[i] for i in pred_idx])
        nms = non_max_suppression(cat_locations)
        if len(nms)==0:
            cat_predictions = predictions[:,cat_index]
            pred_max_idx = np.argmax(cat_predictions)
            pred_max = cat_predictions[pred_max_idx]
            pred_max_window = locations[pred_max_idx]
            startx, starty, endx, endy = pred_max_window
            cv2.rectangle(clone, (startx, starty), (endx, endy),  (0,0,255),2)
            text = f'{category_dict[cat_index]}: {int(pred_max*100)}%'
            cv2.putText(clone, text, (startx, starty+15), cv2.FONT_HERSHEY_SIMPLEX, .5, (0,255,0
        for idx in nms:
            startx, starty, endx, endy = cat_locations[idx]
            cv2.rectangle(clone, (startx, starty), (endx, endy), (0,0,255), 2)
            text = f'{category_dict[np.argmax(predictions[pred_idx[idx]])]}: {int(predictions[pr
            cv2.putText(clone, text, (startx, starty+15), cv2.FONT_HERSHEY_SIMPLEX, .5, (0,255,0
    print('~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~')
    cv2.imshow(f'Test', np.hstack([clone, clone2]))
    cv2.waitKey(0)
    ss.clear()
    return predictions

#NORMAL MODEL
dim = (150, 150, 3)
```

Get started Open in app

```
predictions = []
for idx, i in enumerate([i for i in os.listdir(test_folder) if i != 'ipynb_checkpoints']):
    img_path = f'{test_folder}/{i}'
    pred = get_img_prediction_bounding_box(img_path, normal_model, dim = (150,150))
    predictions.append(pred)

```

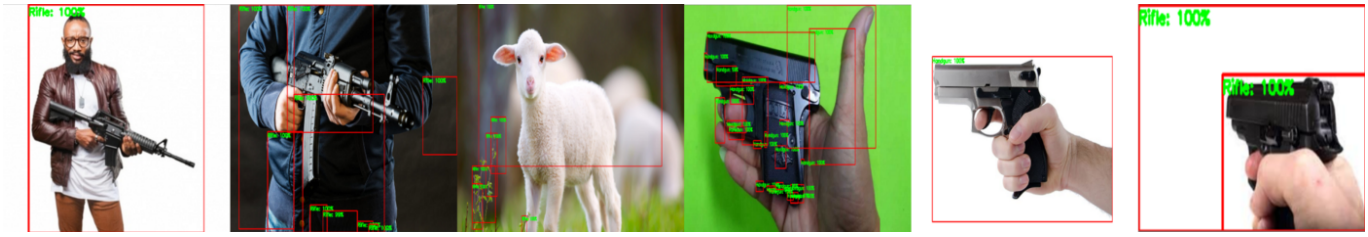
img_bounding_box hosted with ❤️ by GitHub view raw

Before you run the code above, create a folder Tests, and download any image from the internet and name the image the class for which you want to predict. Running the code above will search through every image inside the Tests folder and run that image through our object detection algorithm using the CNN we build above.

The images I tested on were the following:

Base Images

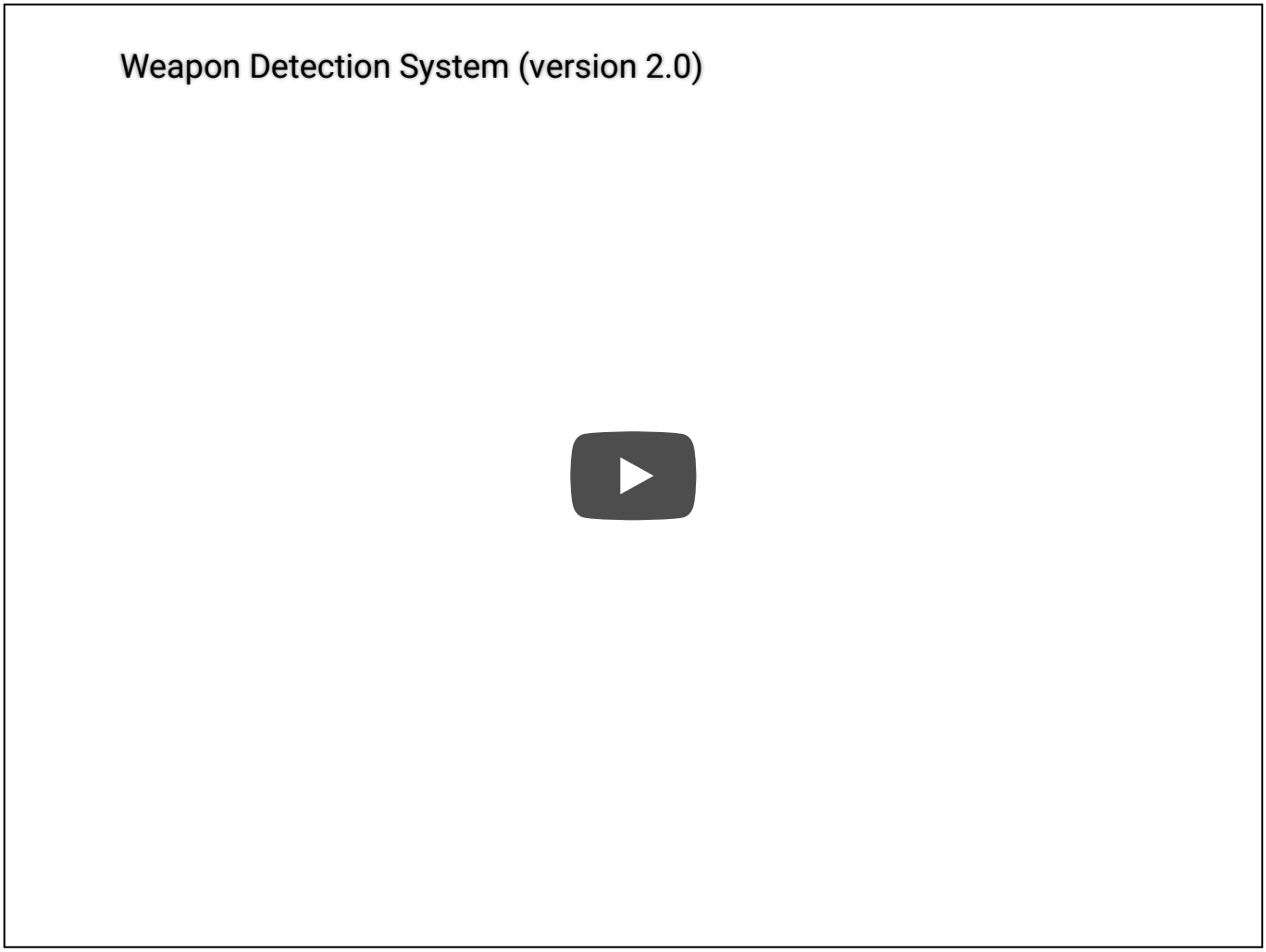
After running the code above, these are the predictions the algorithm gave as an output.



Model Results (Image by Author)[Click here for larger image]

As you can see above, Non-maxima suppression is not perfect, but it does work in some sense. The issue I have here is that there are multiple bounding boxes with 100% confidence so it is hard to pick which one is the *best*. Also, the algorithm is unable to detect non-weapon when there is no weapon in the frame (sheep image).

Get started Open in app



Demo Weapon Detection (Video by Author)

Based on the examples above, we see that the algorithm is faaaar from perfect. This is okay because we still created a pretty cool model that only used 5000 images. Like I said earlier, I have a total of 120,000 images that I scraped from IMFDB.com, so this can only get better with more images we pass in during training.

LIME: Feature Extraction

One of the difficult parts of building and testing a neural network is that the way it works is basically a black box, meaning that you don't understand why the weights are what they are or what within the image the algorithm is using to make its predictions.

Get started Open in app

```
def get_lime_predictions(base_folder, model, dim, iter = 3500):
    from lime import lime_image
    '''This function will take a base folder containing images that will be run through the LIME
    save_name is passed. '''
    lime_images = []
    original_images = []
    for file in os.listdir(base_folder):
        print(f'Creating Lime Image for {file}')
        path = f'{base_folder}/{file}'
        img = get_image_value(path, dim)
        original_images.append(img)
        explainer = lime_image.LimeImageExplainer()
        explanation = explainer.explain_instance(img, model.predict, top_labels = 5, hide_color
        temp, mask = explanation.get_image_and_mask(explanation.top_labels[0], positive_only = F
                                                    hide_rest = False)
        lime_img = mark_boundaries(temp/2 + .5, mask)
        lime_images.append(lime_img)
    lime_images = np.hstack(lime_images)
    original_images = np.hstack(original_images)
    joined_images = np.vstack([original_images, lime_images])
    plt.figure(figsize = (13,13))
    plt.imshow(joined_images)

model_dim = (150,150)
dim = (150, 150, 3)
model = get_conv_model(dim)
model.load_weights('ModelWeights.h5')
func.get_lime_predictions('Tests/Photos', model, dim)
```

lime hosted with ❤️ by GitHub view raw

Lime Predictions

Running the code above will create an image that looks like this:

Get started Open in app

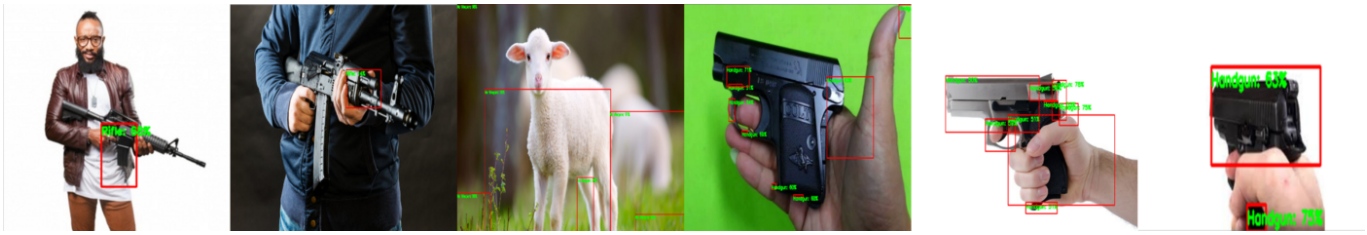
LIME (Image by Author)

The areas that are green are those that the algorithm deems “important”, while the opposite is true for the areas that are red. What we are seeing above is good considering we want the algorithm to detect features of the gun and not the hands or other portions of an image.

Now that we can say we created our very own sentient being… it is time to get real for a second. The model we made is nothing compared to the tools that are already out there. This leads me to Transfer Learning…. where we see some really cool results. For the sake of this tutorial, I will not post the code here but you can find it on my GitHub Repo

Mobilenet

- In the example below, mobilenet was better at predicting objects that were not weapons and had bounding boxes around correct areas.



Mobilenet (Image by Author)[Click here for larger image]

Get started Open in app

- In the example below, VGG16 was unable to distinguish non-weapons like the architecture we built ourselves. It incorrectly classified 1 out of 3 handgun images, while correctly classifying the rest as a handgun

- Although it incorrectly classified a handgun as no weapon (4th to the right), the bounding boxes were not on the gun whatsoever as it stayed on the hand holding the gun.

VGG16 (Image by Author)[Click here for larger image]

VGG16: LIME (Image by Author)

Conclusion

- The goal of this project was to create an algorithm that can integrate itself into traditional surveillance systems and prevent a bad situation faster than a person would (considering the unfortunate circumstances in today's society).

- Although this was cool, the hardware in my computer is not yet there. To segment an image and process each portion of the image takes about 10–45 seconds, which is too slow for live video.

Get started Open in app

- However, although live video is not feasible with an RX 580, using the new Nvidia GPU (3000 series) might have better results.
- Also, this technique can be used for retroactive examination of an event such as body cam footage or protests.

NOTE If you want to follow along with the full project, visit my GitHub **

Sign up for The Daily Pick
By Towards Data Science
Hands-on real-world examples, research, tutorials, and cutting-edge techniques delivered Monday to Thursday. Make learning your daily ritual. Take a look.
Your email
Get this newsletter
By signing up, you will create a Medium account if you don't already have one. Review our Privacy Policy for more information about our privacy practices.

Some rights reserved

Artificial Intelligence Machine Learning Keras Python Object Detection

About Help Legal
Get the Medium app
Download on the GET IT ON

Get started Open in app