

YOLO Convolutional Neural Network

Carlos Matheus Barros da Silva, *Computer Engineering Bachelor Student of ITA*

Prof. Marcos Ricardo Omena de Albuquerque Máximo

Abstract

This paper evaluates a Convolutional Neural Network known as YOLO (You Only Look Once), a very good Neural Network (NN) for computer vision.

In order to evaluate it, it was used as a data set of robot soccer in order to the NN identify the ball and the crossbars on the field.

It was observed that the YOLO NN worked as expected. The predictions to the position of the objects were accurate.

Index Terms

Keras, Neural Network, Convolutional Neural Network, YOLO

I. INTRODUCTION

Neural networks (NN) are computing systems vaguely inspired by the biological neural networks and astrocytes that constitute animal brains. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. Such systems "learn" to perform tasks by considering examples, generally without being programmed with any task-specific rules. For example, an image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the results to identify cats in other images. They do this without any prior knowledge about cats, for example, that they have fur, tails, whiskers and cat-like faces. Instead, they automatically generate identifying characteristics from the learning material that they process.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended

Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer are François Chollet, a Google engineer. Chollet also is the author of the Xception deep neural network model.

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery.

CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually refer to fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. However, CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme.

They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

II. NEURAL NETWORK IMPLEMENTATION

The implementation was based on the file *yolo_detector* and the file *make_detector_network*. The first one is where the logic of getting the image, treat the image, pass it to the NN, and treat the result lies. While the second is where the Neural Network is defined.

The code of *make_detector_network* can be seen in the Code 1 and the code of *yolo_detector* can be seen in the Code 2.

```
def make_detector_network(img_cols, img_rows):
    """
    Makes the convolutional neural network used in the object detector.

    :param img_cols: number of columns of the input image.
    :param img_rows: number of rows of the input image.
    :return: Keras' model of the neural network.
    """
    # Input layer
    input_image = Input(shape=(img_cols, img_rows, 3))

    # Layer 1
    layer = Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_1',
        use_bias=False)(input_image)
    layer = BatchNormalization(name='norm_1')(layer)
```

```
layer = LeakyReLU(alpha=0.1, name='leaky_relu_1')(layer)

# Layer 2
layer = Conv2D(filters=8, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_2',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_2')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_2')(layer)

# Layer 3
layer = Conv2D(filters=16, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_3',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_3')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_3')(layer)
layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', name='max_pool_3')(
layer)

# Layer 4
layer = Conv2D(filters=32, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_4',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_4')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_4')(layer)
layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', name='max_pool_4')(
layer)

# Layer 5
layer = Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_5',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_5')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_5')(layer)
layer = MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same', name='max_pool_5')(
layer)

# Layer 6
layer = Conv2D(filters=64, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_6',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_6')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_6')(layer)
layer = MaxPooling2D(pool_size=(2, 2), strides=(1, 1), padding='same', name='max_pool_6')(
layer)

skip_connection = layer

# Layer 7
layer = Conv2D(filters=128, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_7',
               use_bias=False)(layer)
layer = BatchNormalization(name='norm_7')(layer)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_7')(layer)
```

```

conv_skip = Conv2D(filters=128, kernel_size=(1, 1), strides=(1, 1), padding='same', name='
conv_skip', use_bias=False)(skip_connection)

# Layer 8
layer = Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), padding='same', name='conv_8'
, use_bias=False)(
    layer)
skip_connection = BatchNormalization(name='norm_skip')(conv_skip)
layer = BatchNormalization(name='norm_8')(layer)
skip_connection = LeakyReLU(alpha=0.1, name='leaky_relu_skip')(skip_connection)
layer = LeakyReLU(alpha=0.1, name='leaky_relu_8')(layer)

# Concatenating layers 7B and 8
layer = concatenate([skip_connection, layer], name='concat')

# Layer 9 (last layer)
layer = Conv2D(10, (1, 1), strides=(1, 1), padding='same', name='conv_9', use_bias=True)(
layer)

model = Model(inputs=input_image, outputs=layer, name='ITA_YOLO')

return model

```

Code 1. Definition of the Neural Network with Keras

```

class YoloDetector:
    """
    Represents an object detector for robot soccer based on the YOLO algorithm.
    """
    def __init__(self, model_name, anchor_box_ball=(5, 5), anchor_box_post=(2, 5)):
        """
        Constructs an object detector for robot soccer based on the YOLO algorithm.

        :param model_name: name of the neural network model which will be loaded.
        :type model_name: str.
        :param anchor_box_ball: dimensions of the anchor box used for the ball.
        :type anchor_box_ball: bidimensional tuple.
        :param anchor_box_post: dimensions of the anchor box used for the goal post.
        :type anchor_box_post: bidimensional tuple.
        """
        self.network = load_model(model_name + '.hdf5')
        self.network.summary() # prints the neural network summary
        self.anchor_box_ball = anchor_box_ball
        self.anchor_box_post = anchor_box_post

    def detect(self, image):
        """
        Detects robot soccer's objects given the robot's camera image.

```

```

        :param image: image from the robot camera in 640x480 resolution and RGB color space.
        :type image: OpenCV's image.
        :return: (ball_detection, post1_detection, post2_detection), where each detection is
given
        by a 5-dimensional tuple: (probability, x, y, width, height).
        :rtype: 3-dimensional tuple of 5-dimensional tuples.
        """
        image = self.preprocess_image(image)
        output = self.network.predict(image)
        return self.process_yolo_output(output)

def preprocess_image(self, image):
    """
    Preprocesses the camera image to adapt it to the neural network.

    :param image: image from the robot camera in 640x480 resolution and RGB color space.
    :type image: OpenCV's image.
    :return: image suitable for use in the neural network.
    :rtype: NumPy 4-dimensional array with dimensions (1, 120, 160, 3).
    """
    image = cv2.resize(image, (160, 120), interpolation=cv2.INTER_AREA)
    image = np.array(image) / 255.0
    image = np.reshape(image, (1, 120, 160, 3))
    return image

def process_yolo_output(self, output):
    """
    Processes the neural network's output to yield the detections.

    :param output: neural network's output.
    :type output: NumPy 4-dimensional array with dimensions (1, 15, 20, 10).
    :return: (ball_detection, post1_detection, post2_detection), where each detection is
given
    by a 5-dimensional tuple: (probability, x, y, width, height).
    :rtype: 3-dimensional tuple of 5-dimensional tuples.
    """
    coord_scale = 4 * 8 # coordinate scale used for computing the x and y coordinates of the
BB's center
    bb_scale = 640 # bounding box scale used for computing width and height

    output = np.reshape(output, (15, 20, 10)) # reshaping to remove the first dimension

def get_crossbar_params(row_idx, elm_idx, elm):
    x_cross = (elm_idx + sigmoid(elm[6])) * coord_scale
    y_cross = (row_idx + sigmoid(elm[7])) * coord_scale
    w_cross = bb_scale * 2 * np.exp(elm[8])
    h_cross = bb_scale * 5 * np.exp(elm[9])
    return x_cross, y_cross, w_cross, h_cross

```

```

ball_detection = (0.0, 0.0, 0.0, 0.0, 0.0)
post1_detection = (0.0, 0.0, 0.0, 0.0, 0.0)
post2_detection = (0.0, 0.0, 0.0, 0.0, 0.0)

for row_idx, row in enumerate(output):
    for elm_idx, elm in enumerate(row):

        # treat ball case:
        ball_prob = sigmoid(elm[0])
        if ball_prob > ball_detection[0]:
            x_ball = (elm_idx + sigmoid(elm[1])) * coord_scale
            y_ball = (row_idx + sigmoid(elm[2])) * coord_scale
            w_ball = bb_scale * 5 * np.exp(elm[3])
            h_ball = bb_scale * 5 * np.exp(elm[4])
            ball_detection = (ball_prob, x_ball, y_ball, w_ball, h_ball)

        # treat crossbar case:
        cross_prob = sigmoid(elm[5])
        if cross_prob > post1_detection[0]:
            x_cross, y_cross, w_cross, h_cross = get_crossbar_params(row_idx, elm_idx,
elm)

            post1_detection = (cross_prob, x_cross, y_cross, w_cross, h_cross)
        elif cross_prob > post2_detection[0]:
            x_cross, y_cross, w_cross, h_cross = get_crossbar_params(row_idx, elm_idx,
elm)

            post2_detection = (cross_prob, x_cross, y_cross, w_cross, h_cross)

    return ball_detection, post1_detection, post2_detection

```

Code 2. Logic to detect crossbars and ball using the NN

III. NEURAL NETWORK ANALYSIS

A. YOLO Neural Network creation analysis

Running the file *make_detector_network*, it is printed the NN defined in the code 1. The details of the Neural Network can be seen on the file 3. It is possible to see that it is very similar to the proposed.

Layer (type)	Output Shape	Param #	Connected to
<hr/>			
input_1 (InputLayer)	(None, 120, 160, 3)	0	

conv_1 (Conv2D)	(None, 120, 160, 8)	216	input_1[0][0]
<hr/>			
norm_1 (BatchNormalization)	(None, 120, 160, 8)	32	conv_1[0][0]
<hr/>			
leaky_relu_1 (LeakyReLU)	(None, 120, 160, 8)	0	norm_1[0][0]
<hr/>			
conv_2 (Conv2D)	(None, 120, 160, 8)	576	leaky_relu_1[0][0]
<hr/>			
norm_2 (BatchNormalization)	(None, 120, 160, 8)	32	conv_2[0][0]
<hr/>			
leaky_relu_2 (LeakyReLU)	(None, 120, 160, 8)	0	norm_2[0][0]
<hr/>			
conv_3 (Conv2D)	(None, 120, 160, 16)	1152	leaky_relu_2[0][0]
<hr/>			
norm_3 (BatchNormalization)	(None, 120, 160, 16)	64	conv_3[0][0]
<hr/>			
leaky_relu_3 (LeakyReLU)	(None, 120, 160, 16)	0	norm_3[0][0]
<hr/>			
max_pool_3 (MaxPooling2D)	(None, 60, 80, 16)	0	leaky_relu_3[0][0]
<hr/>			
conv_4 (Conv2D)	(None, 60, 80, 32)	4608	max_pool_3[0][0]
<hr/>			
norm_4 (BatchNormalization)	(None, 60, 80, 32)	128	conv_4[0][0]
<hr/>			
leaky_relu_4 (LeakyReLU)	(None, 60, 80, 32)	0	norm_4[0][0]
<hr/>			
max_pool_4 (MaxPooling2D)	(None, 30, 40, 32)	0	leaky_relu_4[0][0]
<hr/>			
conv_5 (Conv2D)	(None, 30, 40, 64)	18432	max_pool_4[0][0]
<hr/>			
norm_5 (BatchNormalization)	(None, 30, 40, 64)	256	conv_5[0][0]
<hr/>			

leaky_relu_5 (LeakyReLU)	(None, 30, 40, 64)	0	norm_5[0][0]
<hr/>			
max_pool_5 (MaxPooling2D)	(None, 15, 20, 64)	0	leaky_relu_5[0][0]
<hr/>			
conv_6 (Conv2D)	(None, 15, 20, 64)	36864	max_pool_5[0][0]
<hr/>			
norm_6 (BatchNormalization)	(None, 15, 20, 64)	256	conv_6[0][0]
<hr/>			
leaky_relu_6 (LeakyReLU)	(None, 15, 20, 64)	0	norm_6[0][0]
<hr/>			
max_pool_6 (MaxPooling2D)	(None, 15, 20, 64)	0	leaky_relu_6[0][0]
<hr/>			
conv_7 (Conv2D)	(None, 15, 20, 128)	73728	max_pool_6[0][0]
<hr/>			
norm_7 (BatchNormalization)	(None, 15, 20, 128)	512	conv_7[0][0]
<hr/>			
leaky_relu_7 (LeakyReLU)	(None, 15, 20, 128)	0	norm_7[0][0]
<hr/>			
conv_skip (Conv2D)	(None, 15, 20, 128)	8192	max_pool_6[0][0]
<hr/>			
conv_8 (Conv2D)	(None, 15, 20, 256)	294912	leaky_relu_7[0][0]
<hr/>			
norm_skip (BatchNormalization)	(None, 15, 20, 128)	512	conv_skip[0][0]
<hr/>			
norm_8 (BatchNormalization)	(None, 15, 20, 256)	1024	conv_8[0][0]
<hr/>			
leaky_relu_skip (LeakyReLU)	(None, 15, 20, 128)	0	norm_skip[0][0]
<hr/>			
leaky_relu_8 (LeakyReLU)	(None, 15, 20, 256)	0	norm_8[0][0]
<hr/>			
concat (Concatenate)	(None, 15, 20, 384)	0	leaky_relu_skip[0][0] leaky_relu_8[0][0]

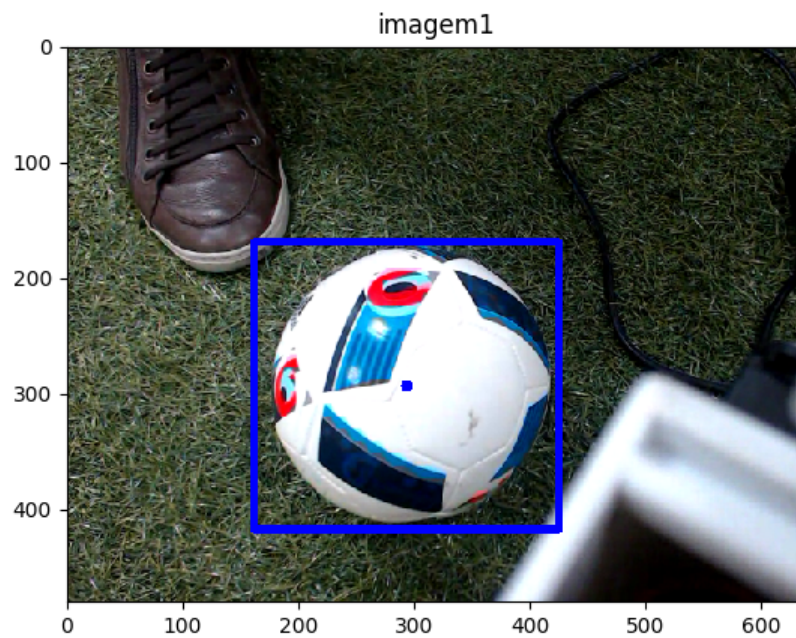


Fig. 1. Example of identification of ball by the YOLO.

```
conv_9 (Conv2D)          (None, 15, 20, 10)    3850      concat[0][0]
```

Total params: 445,346

Trainable params: 443,938

Non-trainable params: 1,408

Code 3. Details of the NN created with Keras

B. YOLO performance analysis

In order to evaluate the YOLO Neural Network, it was used as a model already trained to identify the ball and the crossbars in the soccer field.

The YOLO performed very well on the 10 cases having a 100% accuracy. On the figures from Image 1 to Image 5 it is possible its identification in some test cases.

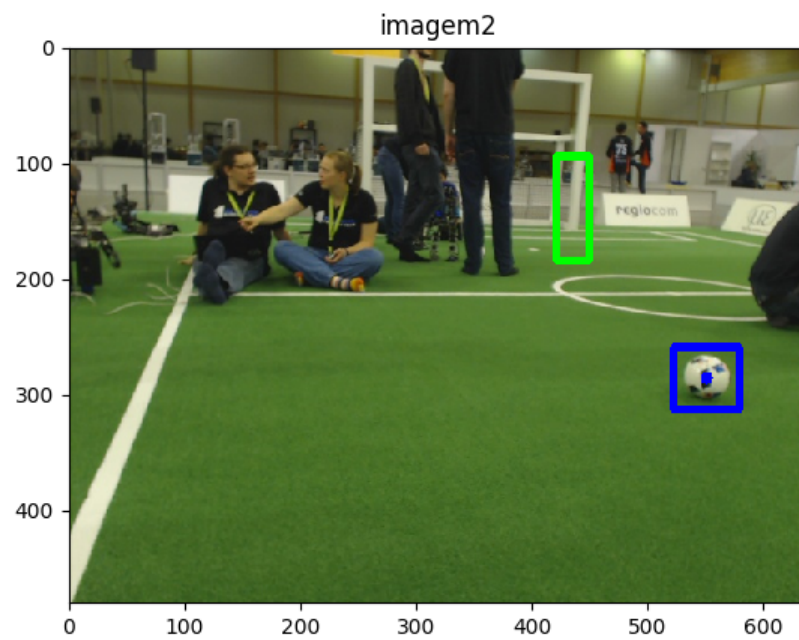


Fig. 2. Example of identification of ball and crossbar by the YOLO.

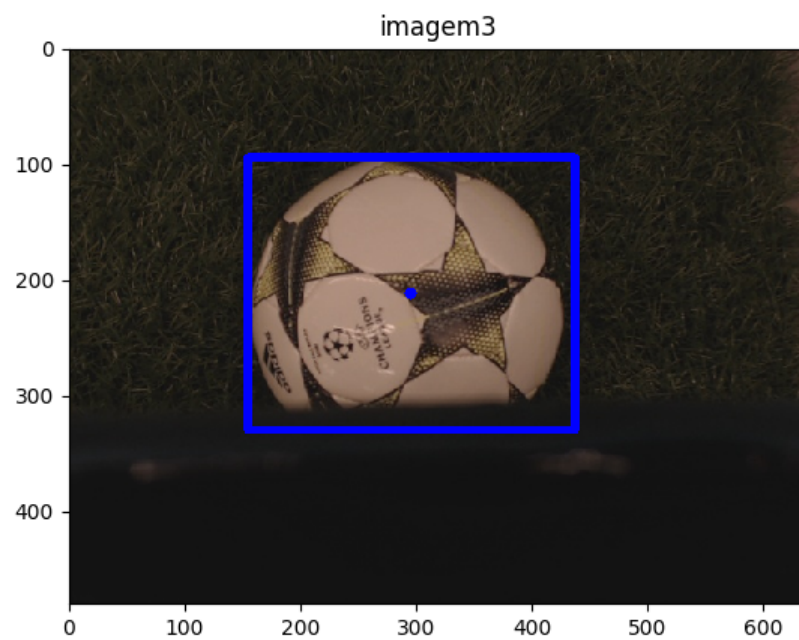


Fig. 3. Example of identification of ball by the YOLO.

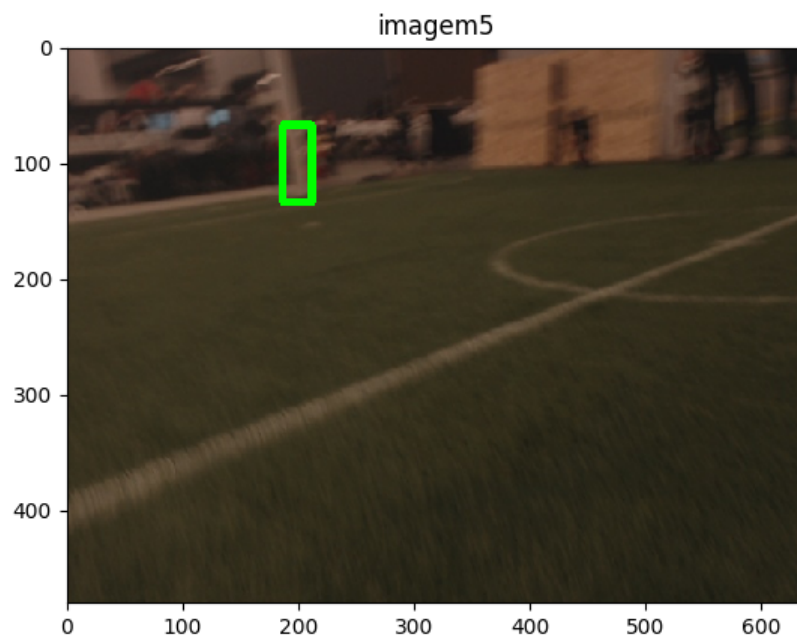


Fig. 4. Example of identification of crossbar by the YOLO.

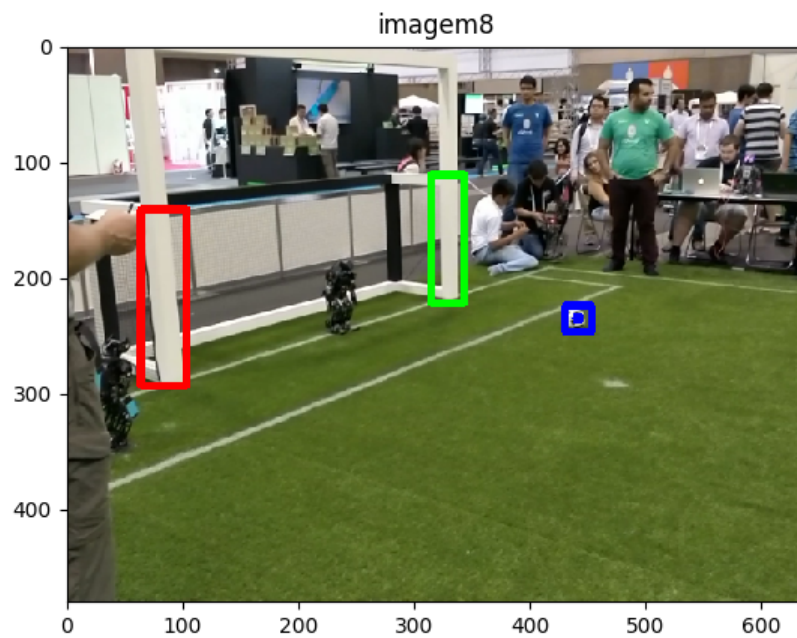


Fig. 5. Example of identification of ball and crossbars by the YOLO.

IV. CONCLUSION

It was clear, therefore, that the YOLO Neural Network worked as expected. The accuracy was 100% for the data set of 10 elements. Due to the small number of the elements on the data set is not possible to predict how it should perform for the very large data set in terms of the actual accuracy percentage.