Convolutional Neural Network for Object Recognition and Detection

---

By : Antonius Robotsoft – www.robotsoft.co.id

Basically computer vision has 4 main tasks :

1. Object Recognition/Classification

Classify the object in the image.

2.Object Detection

Are there any object that we want to detect in the image? If yes, draw the bounding box around the image

3. Object Localization

Are there any object that we want to detect in the image? If yes, draw the bounding box around the image and show the coordinate of the bounding box.

The (x1, y1) would be the top left corner and the (x2, y2) the bottom right.

And finally … the latest one ：

4. Object Segmentation

By accommodating mask rcnn, we can get the exact pixel position for each object. This kind of development is very important for robotic vision.

Suppose you have a small robot

And we need to instruct the robot to get passes through this woman between her tiny legs. By using mask rcnn, the robot knows the exact position of her legs.

This kind of trick can not be accomplished by object localization which uses bounding box since we need to know exact position of her leg.

Currently, The most suitable type of neural network to perform those 4 tasks is **“convolutional neural network”**.

Previously on my post, I wrote about "Cardboard Box Detection using Retinanet (Keras)", it's about train a custom model on keras retinanet for cardboard localization in the image. **RetinaNet is a convolutional neural network architecture**.

Convolutional neural network is commonly used in computer vision for object detections, object localizations, object recognitions, analyzing depth of image regions, etc…

This post will cover about convolutional neural network in general, including some math of convnet, convnet architecture and then continue with RetinaNet architecture.

Convolutional Neural Network

"A convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNN is an improved version of multilayer perceptron". It's a class of deep neural network inspired by human's visual cortex.

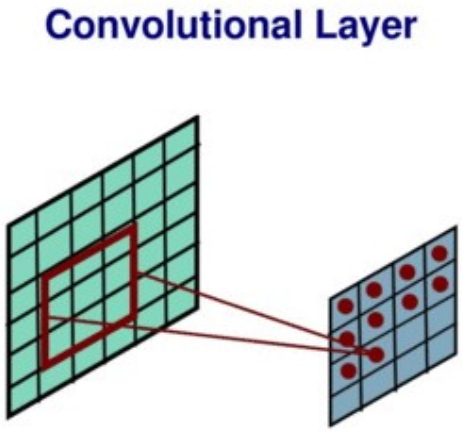
Basically CNN works by collecting matrix of features then predicting whether this image contains a class or another class based on these features using softmax probabilities.

Convolutional Neural Network Architecture

Commonly, a convolutional neural network architecture consists of these layers :

1.Convolution Layer

The core idea between convolutional operation is for feature extractions or we can say filtering. Later, the network will be trying every possible matching features from the input image compared to the class's image (class is an object name that we want to recognize, e.g : a car).



In order to get enlightenment of how convolutional layer operate, have a look at above image. Based from the above picture we have an input image of 6×6 px, and we have a 3×3 px 2d convolution kernel. The kernel will do 1 **stride** from top left pixel of the image until the bottom of the image. The kernel is a 3×3 matrix of weight (each component of the matrix is a weight). This convolutional operation is used to extract features from image. The most frequently used kernel for convolutional is 2d convolution kernel.

Suppose we have a 6×6 pixel image with RGB color channel.

Suppose we are going to do a convolutional operation using 3×3 matrix as kernel and stride = 1 (the number of strides defines how many pixels the kernel will step).

Here's the RGB channel in 6×6 matrix extracted from numpy array :

```
ringlayer@ringlayer-Inspiron-3442:~/Desktop/tutor/np$ ./png2.py
R channel
[[ 46  67 161 250 223 169]
 [ 48  41 114  65 159 104]
 [101 165 216 231 230 196]
 [ 54 255 145  87 106 111]
 [233 138 206 233 134 145]
 [148 255  75  43 139 176]]
------------------------------
G channel
[[ 48  55 120 175 113  40]
 [ 33  20  80  19  96  36]
 [ 53 126 195 234 255 248]
 [  0 238 118 105 185 219]
 [151  67 157 230 202 250]
 [ 60 191  13  29 203 255]]
------------------------------
B channel
[[ 60  55 102 152 100  34]
 [ 40  15  53   0  78  23]
 [ 49 109 152 187 235 225]
 [  0 201  75  55 128 159]
 [ 69   1 125 187 103 121]
 [  0 112   0   0  81 116]]
```

```
#!/usr/bin/env python3
from PIL import Image
import numpy as np
im = Image.open("6px.png")
imgarr = np.array(im)
print("R channel")
print(imgarr[:,:,0])
print("_" * 30)
print("G channel")
print(imgarr[:,:,1])
print("_" * 30)
```

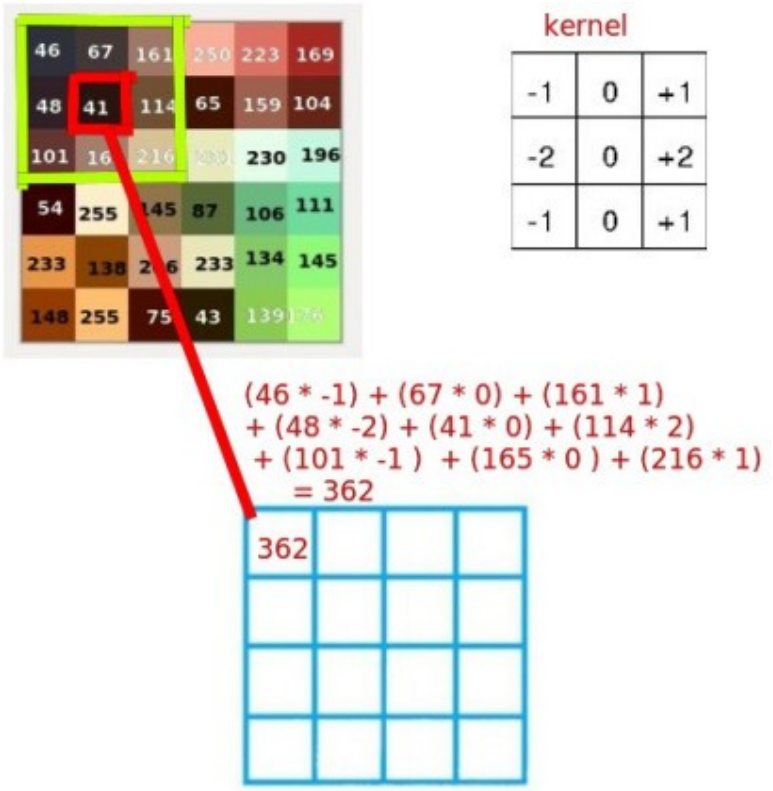
```
print("B channel")
print(imgarr[:,:,2])
print("_" * 30)
```

For this example, we are going to do a 1 stride convolutional operation on red channel using this 3×3 matrix of weight (sobel) :

-1	0	+1
-2	0	+2
-1	0	+1

As an example of convolutional operation, we are going to use "The Red Channel" matrix :

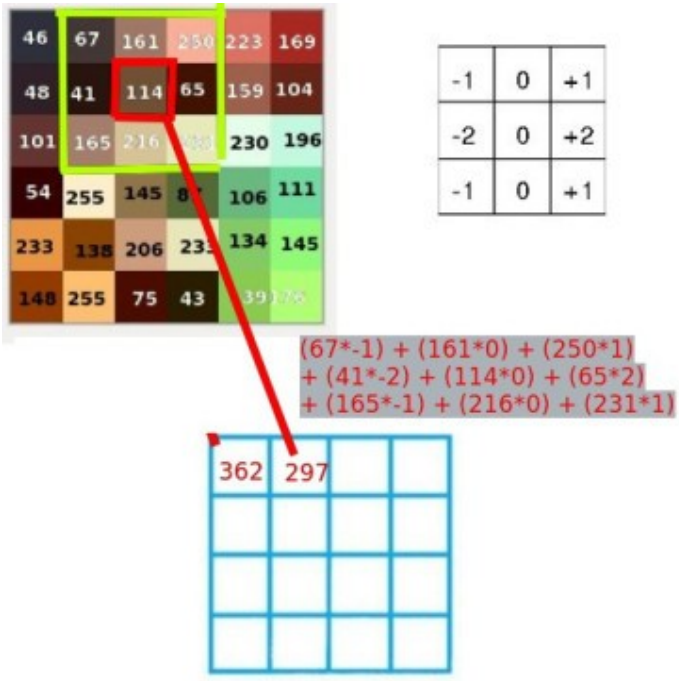
Here's the mathematical operation using convolutional operation :




```
#!/usr/bin/env python3
res = (46 * -1) + (67 * 0) + (161 * 1)+ (48 * -2) + (41 * 0) + (114 * 2) + (101 * -1 ) + (165 * 0 ) + (216 * 1)

print(res)
```

The next 1 pixel stride :



and so on, the stride will continue until the last pixel.

The result is called **convolved feature map matrix**.

Since the matrix is only 4×4 pixel, There will be 2 pixel **padding** for bottom, right, top and left.

The Linearity

Algebraically, a convolutional operation is a linear combination. We need to introduce non linearity hence an activation function is needed. Right after the convolutional operation, in order to introduce non linearity we the “ReLU” activation function is used. If we keep it linear, we do not need to use deep learning since it’s just a simple linear functions.

Mathematically, ReLU can be defined as

$$y = max(0, x)$$

After ReLU, all negative pixel value from the previous **convolved feature map matrix** with negative pixel value will be replaced by 0.

Why Non Linearity is Needed ?

In Math and statistic, a non linearity is commonly used to solve complex problem, meanwhile a linear equation is simple, if we define the input of a linear equation, the output can be found by simple algebra. Before we use activation function such as ReLU, basically the convolutional operation is only a linear function.

Consider an example of a simple this linear equation :

Y = a.x

No matter how many layers, the final activation function will always yield the exact same predicted output. In this condition we do not need to use deep learning with many layers, a simple one layer neural network is enough.

By using ReLU activation function right after a convolutional operation, we can introduce the non linearity hence the system can learn how to solve more complex problem.

Real-life image recognition is a complex problem which can’t be solved literally by computer.

For example we have trained our single layer neural network using dataset of cars and dataset of bat logos:

class 1 is honda civic

class 2 is bat logo

Then if we give an input image with something like this (the same image resolution with dataset)

The computer will be able to answer the correct prediction since it's just answering a literally just the same image with the same pixels arrangement.

Unfortunately when we give this input image :

The computer will not be able to **vote** correctly whether this one is a bat logo or a honda civic class.

In order to solve this kind of complex problem (since the object in image might be rotated slightly or having a different pose or a bit different form) the ideal neural network to solve this one need a non linearity.

By having a different pose or a slightly different form, this means that the prediction can not by simply solved by a linear regression, since

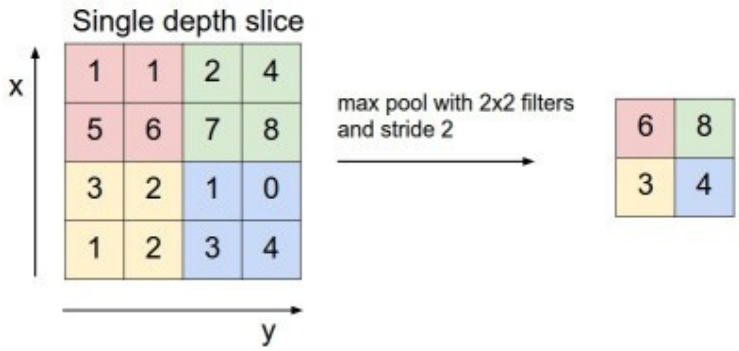
$$Y \neq a.X$$

Y is no longer a.X hence we need to solve this using a non linear equation.

In convolutional neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as **back-propagation**. **Activation functions** will introduce non linearity to the system thus making the **back-propagation** possible since the **gradients** are supplied along with the **error / loss** to update the **weights** and **biases**.

2. Pooling Layer

Right after the ReLU, the next layer is a pooling layer. The pooling layer basically is used to reduce the spatial size of the input hence reducing the number of parameters and computational complexity.



Commonly used pooling method is max pooling.

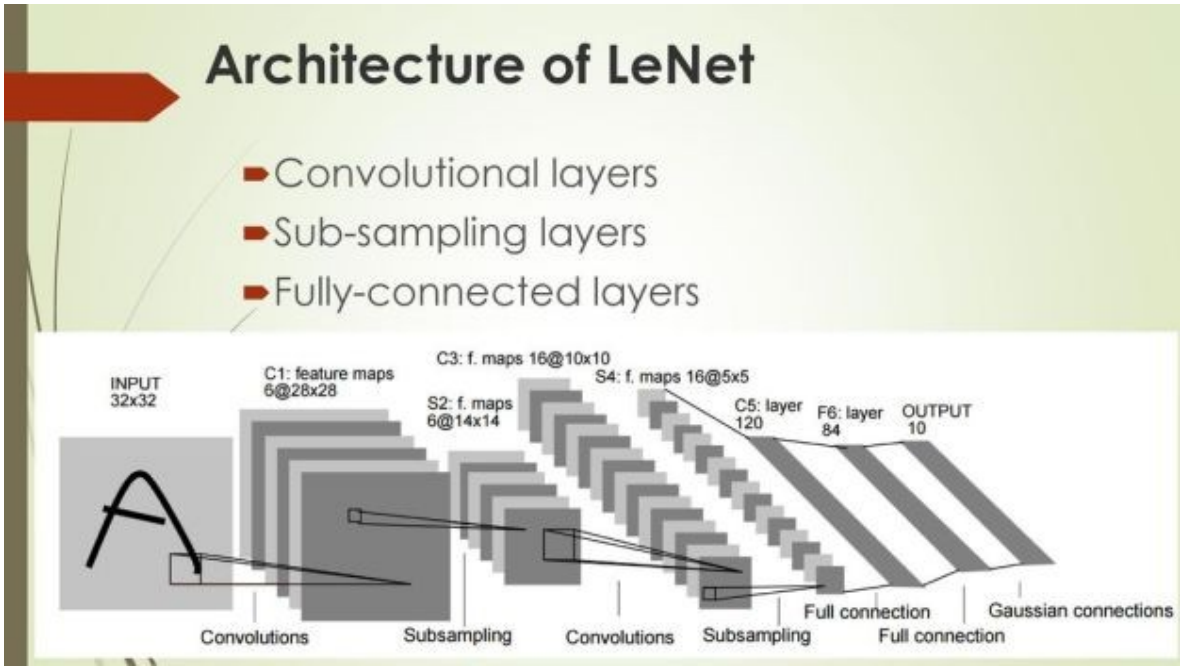
3. Fully Connected Layer

The Fully Connected layer holds composite and aggregate information from previous layers. Before given as input of fully connected layers, those previous multi dimensional inputs will be flatten into a single dimensional inputs.

And finally, the prediction (voting) will be accomplished using the activation function, e.g : softmax.

Some Examples of CNN Architectures

Lenet



The input image of lenet 5 is **32×32 px** image. Here's the summary of lenet 5 architecture :

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Other than using tanh activation function, we can use ReLU as activation function.

Here’s example of lenet implementation in keras :

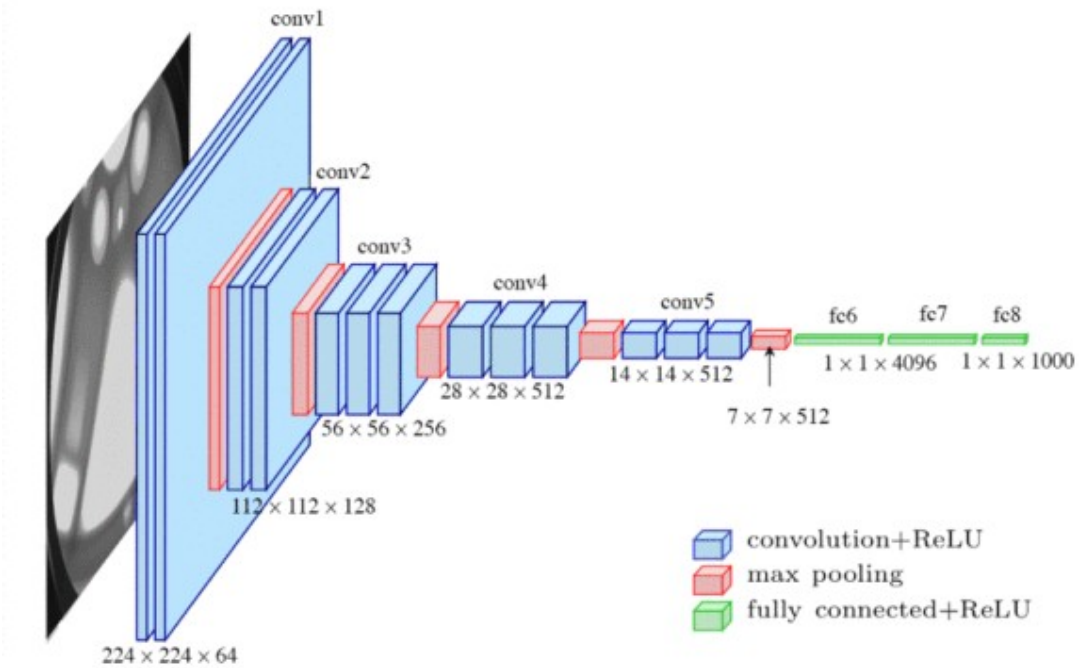
```
import keras
from keras.models import Sequential
from keras import models, layers
model = keras.Sequential()
model.add(layers.Conv2D(filters=6, kernel_size=(3, 3), activation='tanh',
input_shape=(32,32,1)))
model.add(layers.AveragePooling2D())
model.add(layers.Conv2D(filters=16, kernel_size=(3, 3), activation='tanh'))
model.add(layers.AveragePooling2D())
model.add(layers.Flatten())
model.add(layers.Dense(units=120, activation='tanh'))
model.add(layers.Dense(units=84, activation='tanh'))
model.add(layers.Dense(units=10, activation = 'softmax'))
model.summary()
```

```
ringlayer@ringlayer-Inspiron-3442:~/Downloads$ ./lenet.py
Using TensorFlow backend.
```

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 6)	60
average_pooling2d_1 (Average	(None, 15, 15, 6)	0
conv2d_2 (Conv2D)	(None, 13, 13, 16)	880
average_pooling2d_2 (Average	(None, 6, 6, 16)	0
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 120)	69240
dense_2 (Dense)	(None, 84)	10164
dense_3 (Dense)	(None, 10)	850

```
Total params: 81,194
Trainable params: 81,194
Non-trainable params: 0
```

VGG16



The input image of vgg16 is **224×224 px.** Here's the summary of vgg16 architecture :

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	224 x 224 x 3	-	-	-
1	2 X Convolution	64	224 x 224 x 64	3x3	1	relu
	Max Pooling	64	112 x 112 x 64	3x3	2	relu
3	2 X Convolution	128	112 x 112 x 128	3x3	1	relu
	Max Pooling	128	56 x 56 x 128	3x3	2	relu
5	2 X Convolution	256	56 x 56 x 256	3x3	1	relu
	Max Pooling	256	28 x 28 x 256	3x3	2	relu
7	3 X Convolution	512	28 x 28 x 512	3x3	1	relu
	Max Pooling	512	14 x 14 x 512	3x3	2	relu
10	3 X Convolution	512	14 x 14 x 512	3x3	1	relu
	Max Pooling	512	7 x 7 x 512	3x3	2	relu
13	FC	-	25088	-	-	relu
14	FC	-	4096	-	-	relu
15	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Example of implementation of vgg16 in keras :

```
#!/usr/bin/env python3

import keras
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout, Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras import models, layers

model = keras.Sequential()

model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu',
input_shape=(224,224,3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(layers.Conv2D(128, (3, 3),activation='relu',padding='same'))
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(256, (3, 3), activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Conv2D(512, (3, 3),activation='relu',padding='same'))
model.add(layers.Conv2D(512, (3, 3),activation='relu', padding='same'))
model.add(layers.Conv2D(512, (3, 3),activation='relu',padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
```

```
model.add(layers.Flatten())
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dense(1000, activation='softmax'))

model.summary()
```

Resnet

The main purpose of resnet architecture is to make a convolutional neural network with many layers to train effectively.

The problem of a deep convolutional neural network is that when we increase the network depth, there's a vanishing gradient problem. As the network goes deeper, its performance gets saturated or even starts degrading in accuracy.

Resnet splits a deeper network into three layer chunks and passing the input into each chunk straight through to the next chunk, along with the residual output of the chunk minus the input to the chunk that is reintroduced.

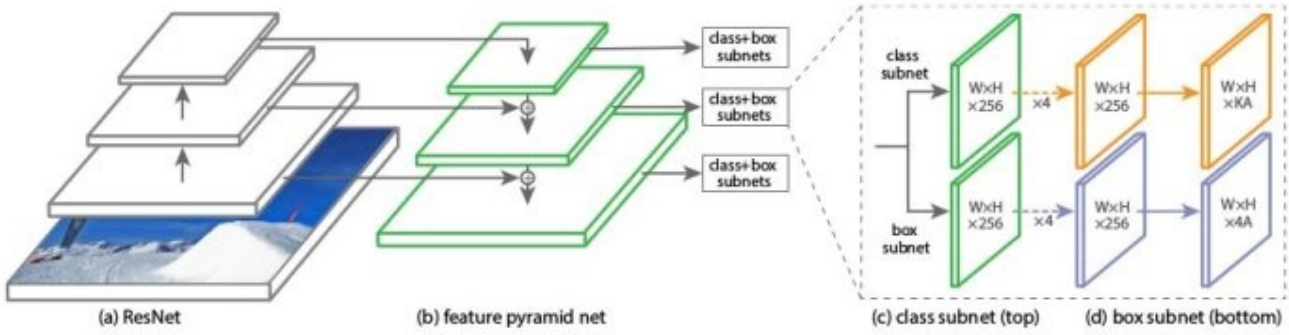
An implementation of resnet from keras :

https://github.com/raghakot/keras-resnet/blob/master/resnet.py

RetinaNet

The problem with a single shot detection model such as yolo is : "there is extreme foreground-background class imbalance problem in one-stage detector."

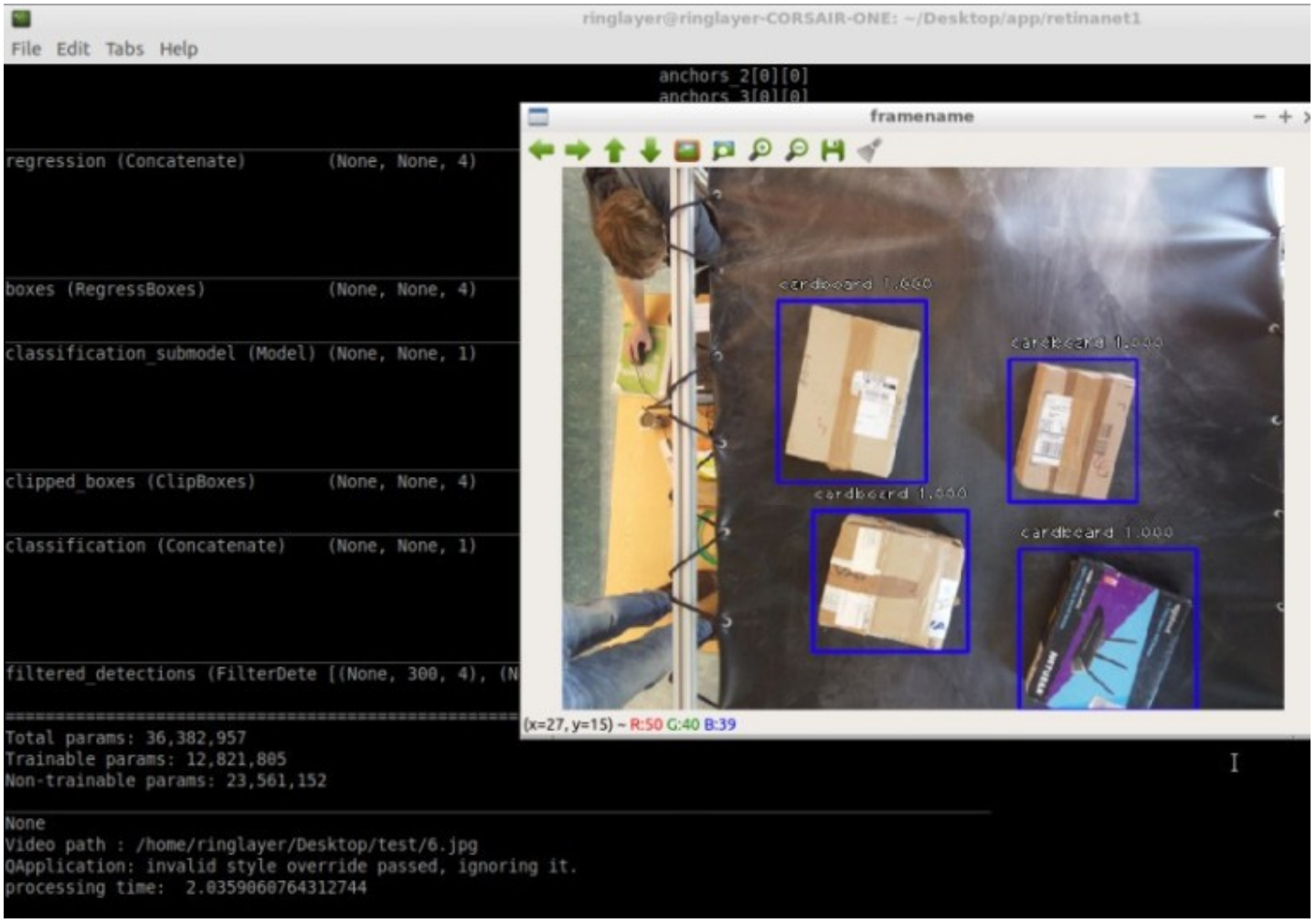
RetinaNet introduce "The Focal Loss" to cover for extreme foreground-background class imbalance problem in one-stage detector.



Retinanet is a single shot detection model just like Yolo. On RetinaNet, a commonly used backbone is resnet50, we add a FPN (Feature Pyramid Network) for feature extraction and later the network will use Focal lost to handle extreme foreground-background class imbalance problem.

Example implementation of RetinaNet using keras can be cloned from
https://github.com/fizyr/keras-retinanet

Example of custom object detection using Retinanet :



Reference :

https://arxiv.org/abs/1708.02002