SCIENCES
SORBONNE
UNIVERSITÉ

People Counting with Fully Convolutional Neural Network

"study of the bases of the fully convolutional neural network and application on the counting of the crowds"

INTRODUCTION

Security management
Customer analysis

Military applications
Design of public space

STATE OF THE ART

CNN-based methods

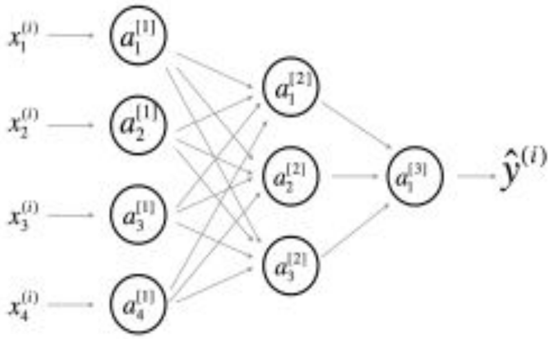
Regression-based approaches

Detection-based approaches

Density estimation based approaches.

DEFINITION

NEURAL NETWORK



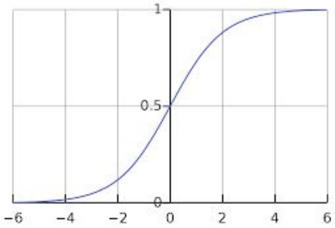
$$a = g^{[l]}\left(W_x x^{(i)} + b_1\right) = g^{[l]}(z_1)$$

$$a^{[L]} = g(w^{[L]} a^{[L-1]} + b^{[L]})$$

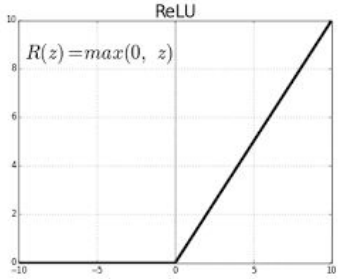
$$z^{[L]} = w^{[L]} a^{[L-1]} + b^{[L]}$$

$$J_0 = (a^{[L]} - y)$$

$$S(x) = \frac{1}{1+e^{-x}}$$



f(x)=x+=max(0,x)



DEFINITION

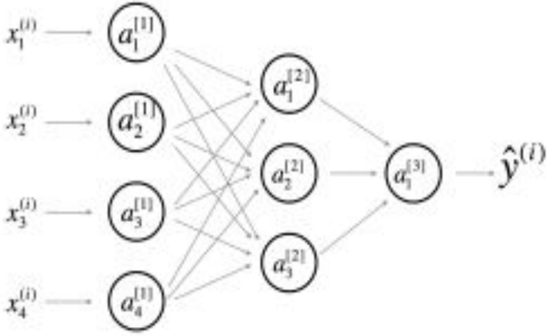
NEURAL NETWORK

$$J_{CE}(\hat{y}, y) = -\sum_{i=0}^{m} y^{(i)} log\ \hat{y}^{(i)}$$

$$J_{1}(\hat{y}, y) = \sum_{i=0}^{m} |y^{(i)} - \hat{y}^{(i)}|$$

DEFINITION

NEURAL NETWORK



$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

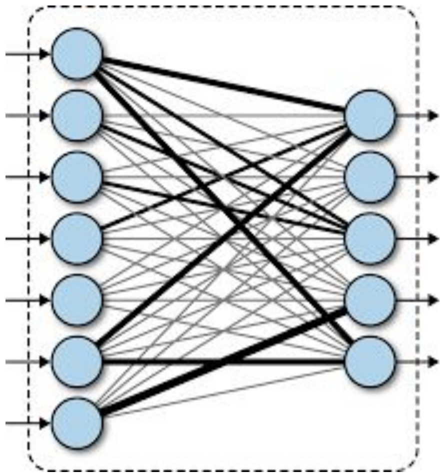
$$dW^{[1]} = dz^{[1]} x^T$$

$$db^{[1]} = dz^{[1]}$$

SCIENCES SORBONNE UNIVERSITÉ

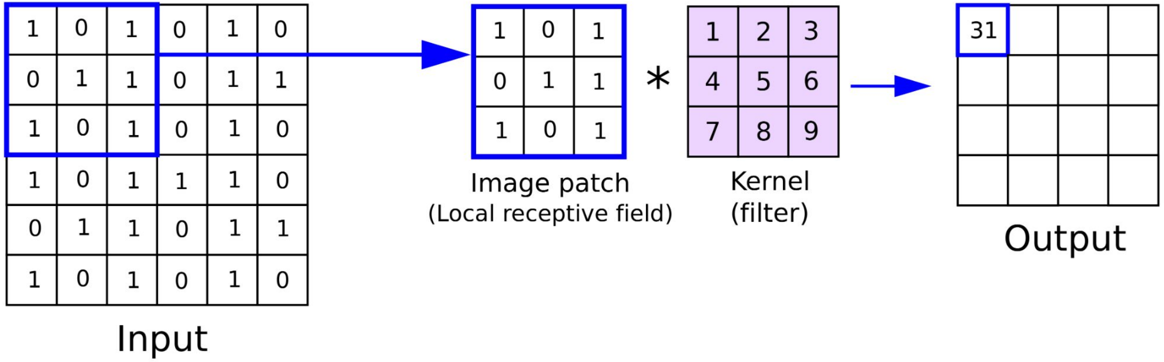
DEFINITION

FULLY CONNECTED LAYER



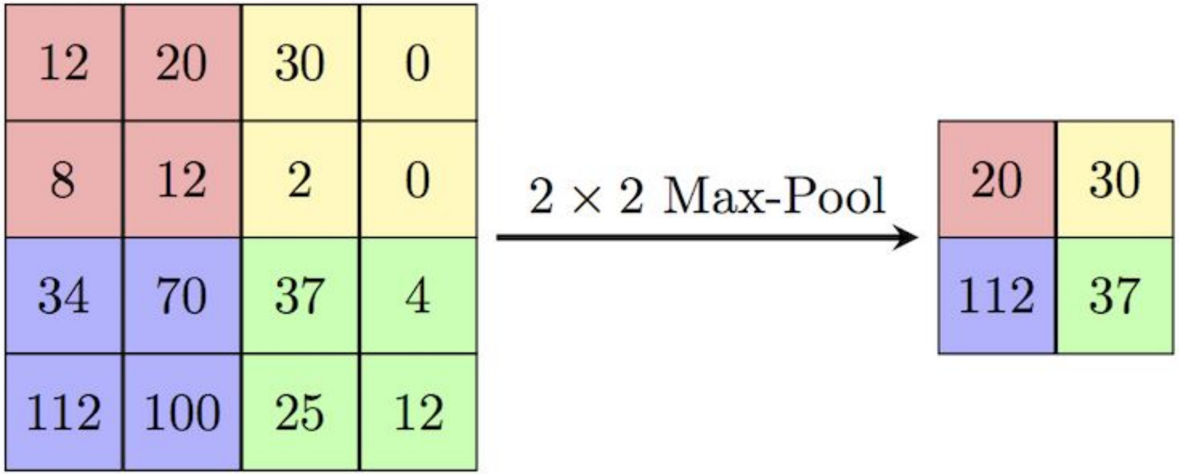
DEFINITION

CONVOLUTION LAYER



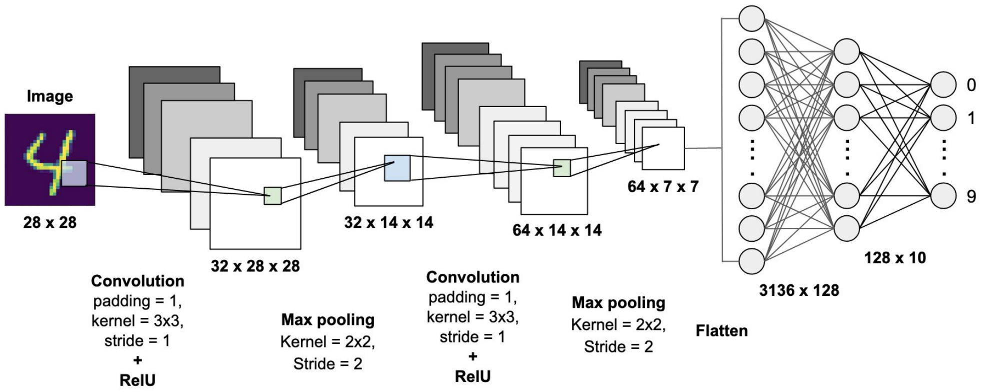
DEFINITION

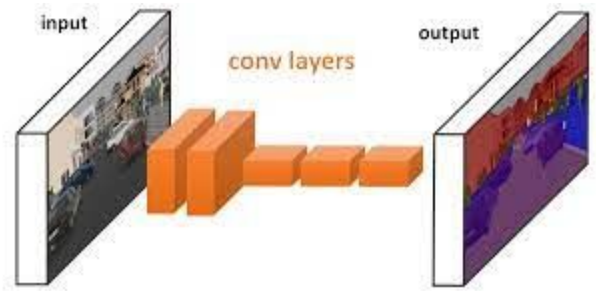
MAX POOLING LAYER



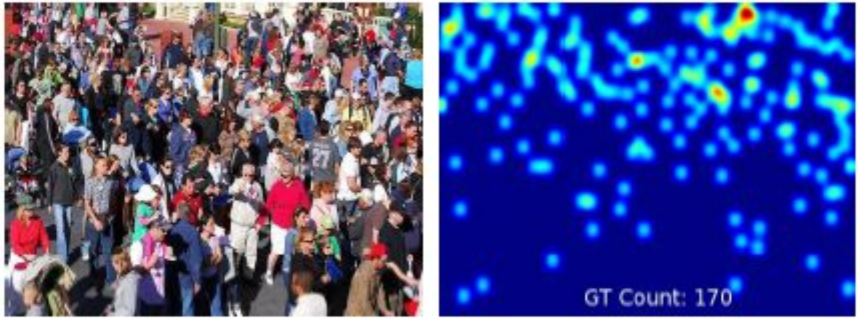
DEFINITION

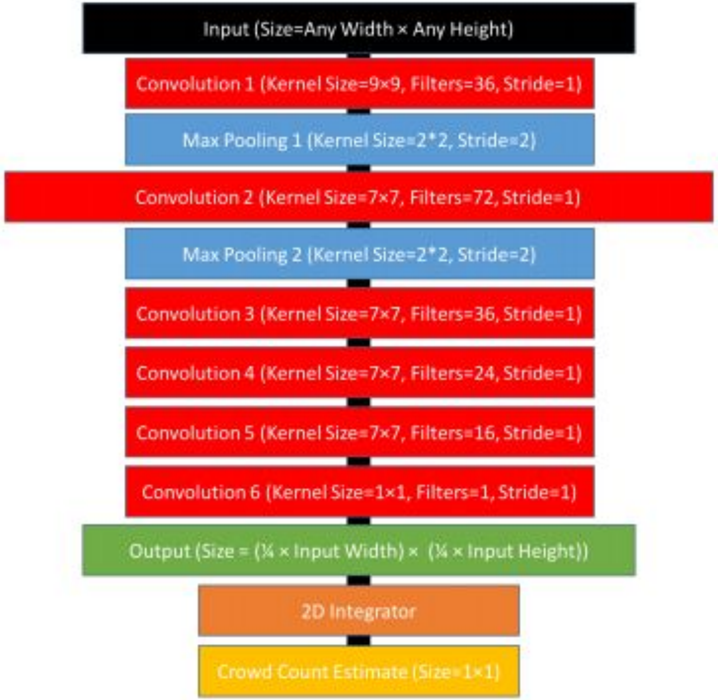
FULLY CONNECT LAYER





ALGORITHM


GT Count: 170

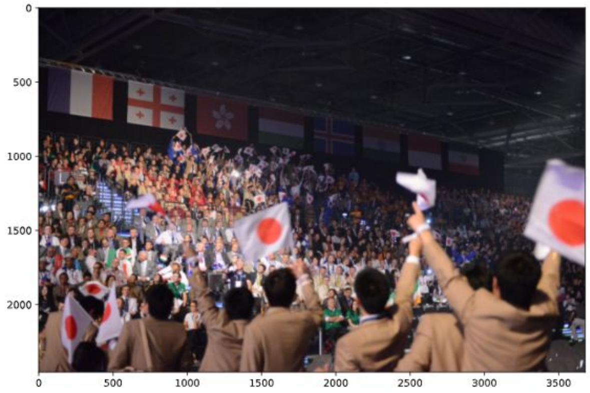

Input (Size=Any Width × Any Height)
Convolution 1 (Kernel Size=9×9, Filters=36, Stride=1)
Max Pooling 1 (Kernel Size=2*2, Stride=2)
Convolution 2 (Kernel Size=7×7, Filters=72, Stride=1)
Max Pooling 2 (Kernel Size=2*2, Stride=2)
Convolution 3 (Kernel Size=7×7, Filters=36, Stride=1)
Convolution 4 (Kernel Size=7×7, Filters=24, Stride=1)
Convolution 5 (Kernel Size=7×7, Filters=16, Stride=1)
Convolution 6 (Kernel Size=1×1, Filters=1, Stride=1)
Output (Size = (¼ × Input Width) × (¼ × Input Height))
2D Integrator
Crowd Count Estimate (Size=1×1)

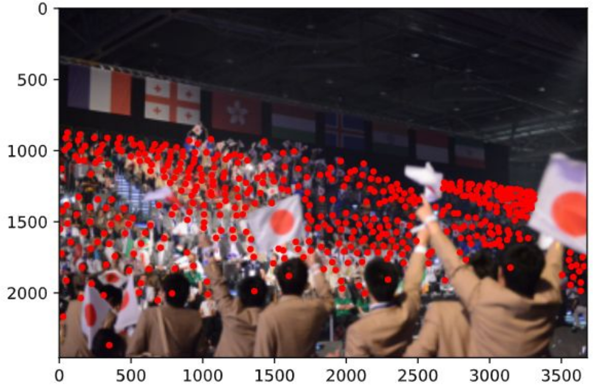
METRICS

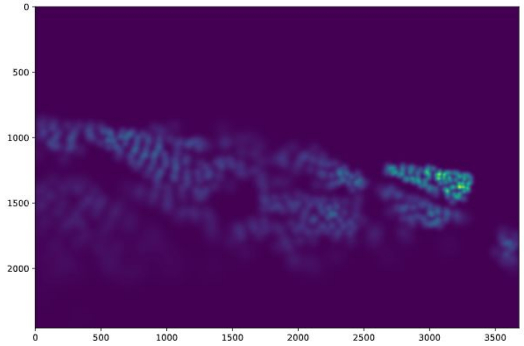
$$MAE = \frac{1}{N}\sum_{1}^{N}|z_i - \check{z}_i|$$

$$MSE = \sqrt{\frac{1}{N}\sum_{1}^{N}(z_i - \check{z}_i)^2}$$

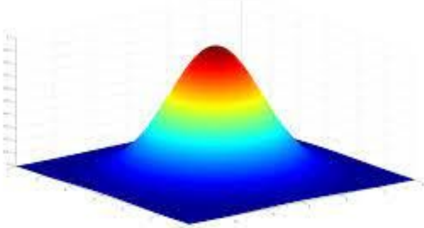
IMPLEMENTATION





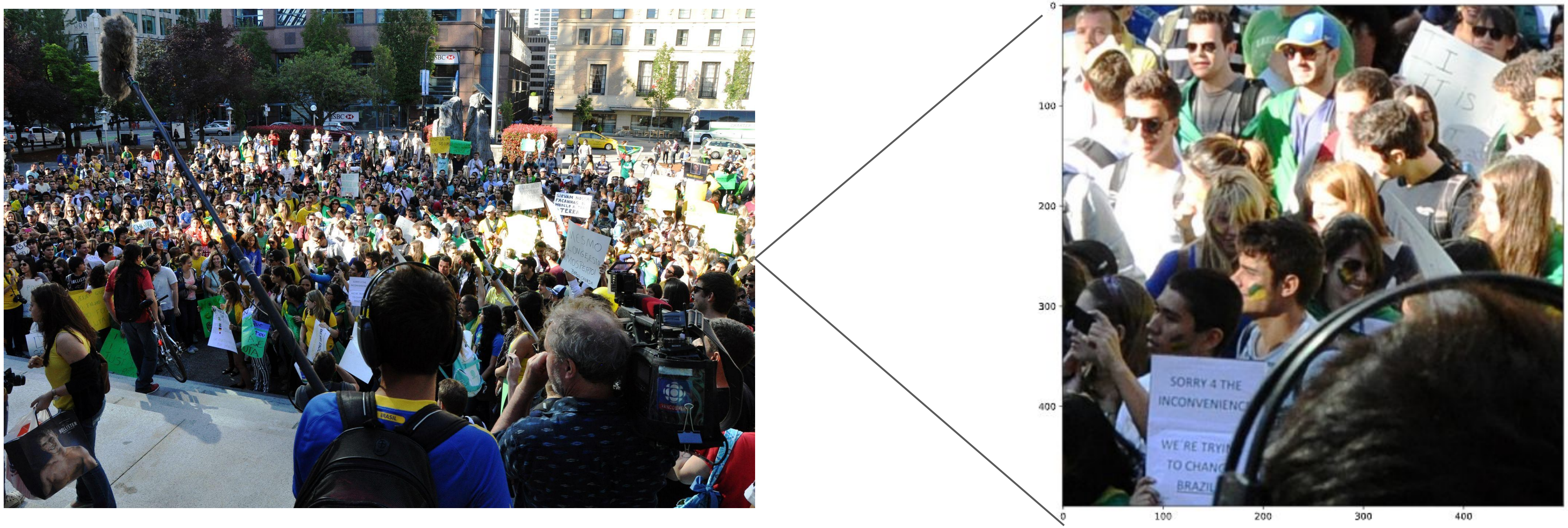


$$G_{\sigma i} = 0,3 * \overline{d_i}$$



$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

IMPLEMENTATION



IMPLEMENTATION

```
#CROP IMAGE
def crop(I,width,height):
    x,y,*z =I.shape
    image= I[int(x/2-width/2):int(x/2+width/2), int(y/2-height/2):int(y/2+height/2)]
    return image
def MultipleCrop(I,width,height):
    x,y,z =I.shape
    images=[]
    numberImageRow=int(x/width)
    numberImageColumn=int(y/height)
    for i in range(0,numberImageColumn):
        for j in range(0,numberImageRow):
            images.append(I[int(j*width):int(j*width+width), int(i*height):int(i*height+height)])
    return images

#INTEGRATION
```

SCIENCES SORBONNE UNIVERSITÉ

IMPLEMENTATION

```
#create model
model=Sequential([
    #encoder
Convolution2D(36,(9,9),input_shape=(None,None,3), padding="same", activation="relu",
kernel_initializer=initializers.random_normal( stddev=0.01)),
MaxPool2D(pool_size=(2,2), strides=(2,2),padding="same"),
Convolution2D(72,(7,7), padding="same", activation="relu",kernel_initializer=initializers.random_normal(stddev=0.01)),
MaxPool2D(pool_size=(2, 2),strides=(2,2)),
Convolution2D(36,(7,7), padding="same", activation="relu",kernel_initializer=initializers.random_normal(stddev=0.01)),
Convolution2D(24,(7,7), padding="same", activation="relu",kernel_initializer=initializers.random_normal(stddev=0.01)),
Convolution2D(16,(7,7), padding="same", activation="relu",kernel_initializer=initializers.random_normal(stddev=0.01)),
Convolution2D(1,(1,1), padding="same", kernel_initializer=initializers.random_normal(stddev=0.01)),#activation="relu"
UpSampling2D((2, 2)),
UpSampling2D((2, 2)),
# Conv2DTranspose(1,kernel_size=(4,4),strides=(4,4),use_bias=False )
])
```

```
from keras.optimizers import SGD

loss_function=keras.losses.MeanSquaredError(reduction='auto')
opt=SGD(learning_rate=0.00001, momentum=0.9, decay=0.0005 , nesterov= True)
model.compile(loss=loss_function, optimizer=opt,metrics=['accuracy'])
```

RESULTS