

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"

Document Index

| | |
|-------------------|----|
| INTRODUCTION | 2 |
| ABSTRACT | 4 |
| STATE OF THE ART | 4 |
| DEFINITIONS | 6 |
| EVALUATION METRIC | 11 |
| DATASET | 12 |
| ALGORITHM | 12 |
| IMPLEMENTATION | 14 |
| code | 14 |
| preprocessing | 15 |
| problems faced | 23 |
| RESULTS | 24 |
| CONCLUSIONS | 26 |
| REFERENCES | 28 |
| SITOGRAPHY | 30 |

1. INTRODUCTION

In recent years the analysis of the crowds is increasingly relevant in inter-disciplinary areas:

Customer analysis: The analysis of customer behaviour within stores is crucial to maximize profit and increase the competitiveness of retail stores [1][2] .

Security management: In public places with high crowd densities, statistics about human traffic flow can facilitate security management as well as urban planning. For security-critical areas such as airports and railway stations this is of extreme importance.

Design of public space: with the increase of populations in urban areas it is increasingly important to create environments such as airports, train stations and other public places organized to manage large influxes of people

There is also the possibility of military applications.

In addition, the technologies developed to analyze crowds can be used in other similar environments

The main purpose of this project is to understand how neural networks work a particular attention will be given to the convolutional layer

To develop this knowledge I will develop a paper-based neural network [24]

This project consists of an algorithm based on a Convolutional neural network to count the actual number of people inside a region of interest through detected images.

At the conclusion I expect an understanding of Convolutional Neural Network-based algorithms.

I choose datasets with a large number of people in outdoor and indoor environments. Thus increasing complexity by having a low resolution of the individual person and having many occlusions.

2. STATE OF THE ART

The study of crowds continues for over ten years and have developed different approaches

2.1. Detection-based approaches

These approaches usually apply a person or head detector where a sliding window detector is used to detect people in the scene on an image. Recently many extraordinary object detectors such as YOLO [16][17], have been presented, which may perform dramatic detection accuracy in the sparse scenes.

However, they will present unsatisfactory results when encountered in the situation of occlusion and background clutter in extremely dense crowds.

2.2. Regression-based approaches

regression-based methods which directly learn the mapping from an image patch to the count. They usually first extract global features (texture, gradient, edge features), or local features.

Then some regression techniques such as linear regression and Gaussian mixture regression are used to learn a mapping function to the crowd counting.

These methods are successful in dealing with the problems of occlusion and background clutter, but they always ignore spatial information

.

2.3. Density estimation based approaches.

While the earlier methods were successful in addressing the issues of occlusion and clutter, most of them ignored important spatial information as they were regressing on the global count[11].

First approach was a density estimation based method by learning a linear mapping between local features and corresponding density maps. For reducing the difficulty of learning a linear mapping, proposes a non-linear mapping, random forest regression, which

obtains satisfactory performance by introducing a crowdedness prior and using it to train two different forests. Besides, this method needs less memory to store the forest.

These methods consider the spatial information, but they only use traditional hand-crafted features to extract low level information, which cannot guide the high-quality density map to estimate more accurate counting.

2.4. CNN-based methods

Excellent results of CNNs in other areas of computer vision have led to the development of new techniques[19].

Earlier heuristic models typically leverage basic CNNs to predict the density of the crowds which obtain significant improvement compared with traditional hand-crafted features.

Lately, more effective and efficient models based on Fully Convolution Network (FCN), which has become the mainstream network architecture for the density estimation and crowd counting. Different supervised level and learning paradigms for different models, also there are some models designed in cross scene and multiple domains.

2.5. Challenge of crowd analysis

Like any other computer vision problem, crowd analysis comes with many challenges such as occlusions, high clutter, non-uniform distribution of people, non-uniform illumination, intra-scene and inter-scene variations in appearance, scale and perspective making the problem extremely difficult. The complexity of the problem together with the wide range of applications for crowd analysis has led to an increased focus by researchers in the recent past.

3. DEFINITIONS

to approach the topic and understand how it works I followed the course (I) offered by professor Andrew Ng on coursera and I referred to these material for better understanding (II)(III)

3.1. Neural Network

to understand how to build a CNN is good to clarify the basic about how it is structured and works an Neural Network

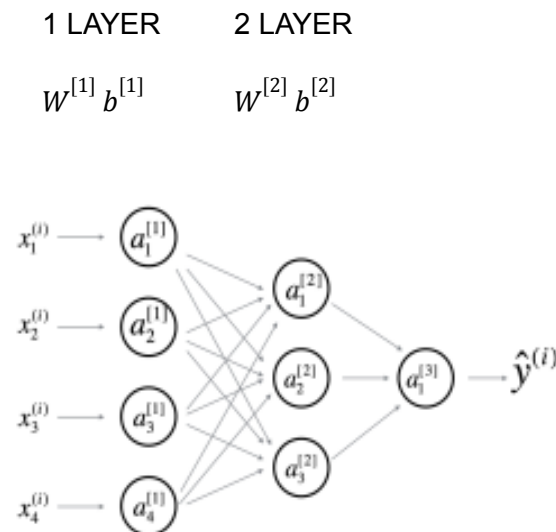


Fig.(1) Simple example of neural network

A neural Network is a computational model composed of artificial "neurons", loosely inspired by the simplification of a biological neural network.

This model consists of a group of interconnections consisting of artificial neurons and processes that use a computational connectivity approach. In most cases an artificial neural network is an adaptive system that changes its structure based on external or internal information that flows through the network itself during the learning phase.

the resolution of the a values is given by the following procedure called forward propagation:

$$a = g^{[l]}(W_x x^{(i)} + b_1) = g^{[l]}(z_1) \text{ where } g^{[l]} \text{ denotes the } l^{th} \text{ layer activation function}$$

g is called activation function the more simple case is a Sigmoid function

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

an better function is the tanh function that have a codomain [-1,+1]

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

This function works better in the inner layers, but for the output layer we need to use the sigmoid function because the codomain of this function is [0,+1] so it can express the probability of the possible output.

One other popular activation function is the Rectified Linear Unit function (ReLU)

$$f(x) = x^+ = \max(0, x)$$

to generate the values of weights and b must be accomplished the back propagation.

The first formula we need is the loss function. It is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with those values. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network.

two possible cost function are:

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

$$J_1(\hat{y}, y) = \sum_{i=0}^m |y^{(i)} - \hat{y}^{(i)}|$$

in this way we can determine weights and bias

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

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

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

the next steps represent how the single passage from one neuron to another works and how during the training the weights are modified with the back propagation

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

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

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

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]'}(z^{[1]})$$

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

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

3.2. Convolutional Neural Network

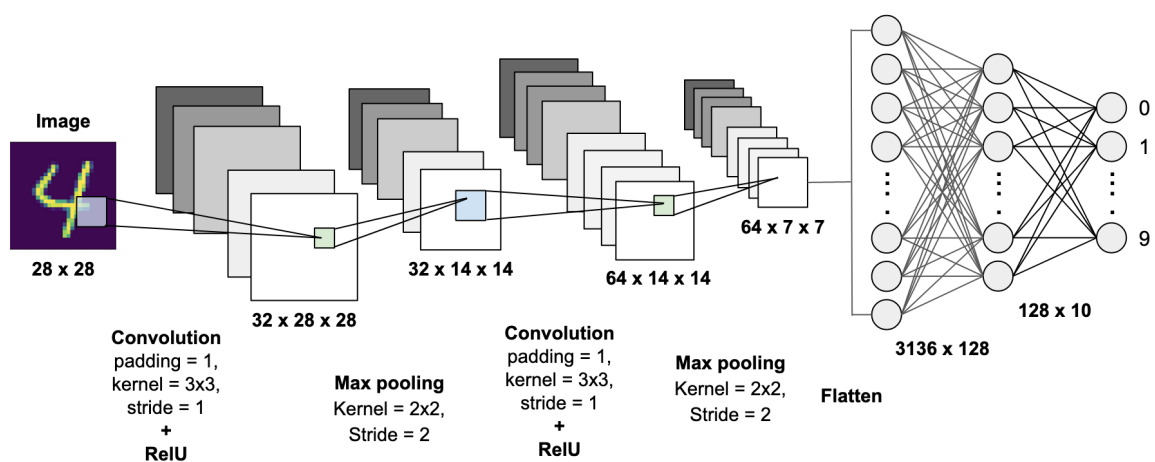


Fig.(2) Example of a Convolutional Neural Network

the simplest composition of a convolutional neural network is composed of three types of layers

3.3. Fully Convolutional Neural Network

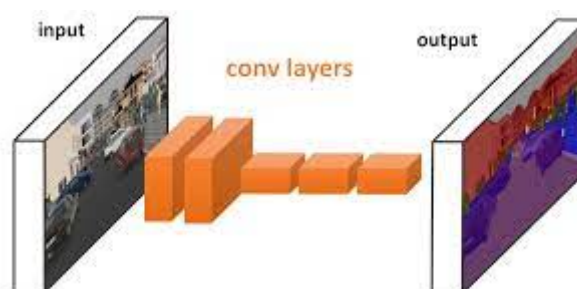


Fig.(3) Fully Convolutional Neural Network

A Fully Convolutional Neural Network (FCN) is a neural network that only performs convolution (and subsampling or upsampling) operations. Equivalently, an FCN is a CNN without fully connected layers.

often it is divided into encoders and decoders, in this case I will only develop the encoder while the decoder will consist of only two levels of upsampling layers

3.3.1. Convolutional layer(CONV)

The convolutional layer is the initial part of our network, in this layer happens what is called "Convolution" from which the CNN takes its name:

In the convolution the characteristics of the image are extracted.

Let's take an image: this is divided into squares

After performing this step, the image is multiplied to a matrix (called a 'filter' or 'kernel' or 'feature detector' - the yellow square) thus obtaining the convolution

the convolutional layer is composed placing as weights convolutional filters.

Every layer is composed of multiple filter.

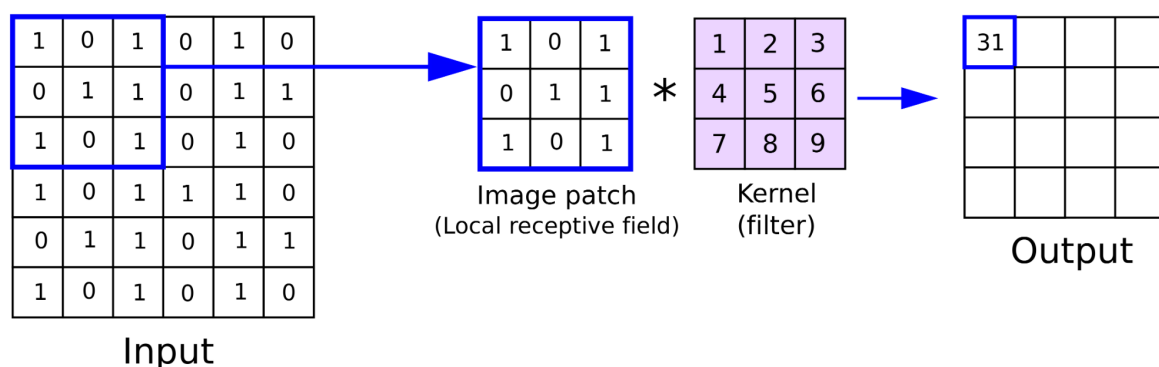


Fig.(4) Convolution layer

3.3.2. Max pooling(POOL)

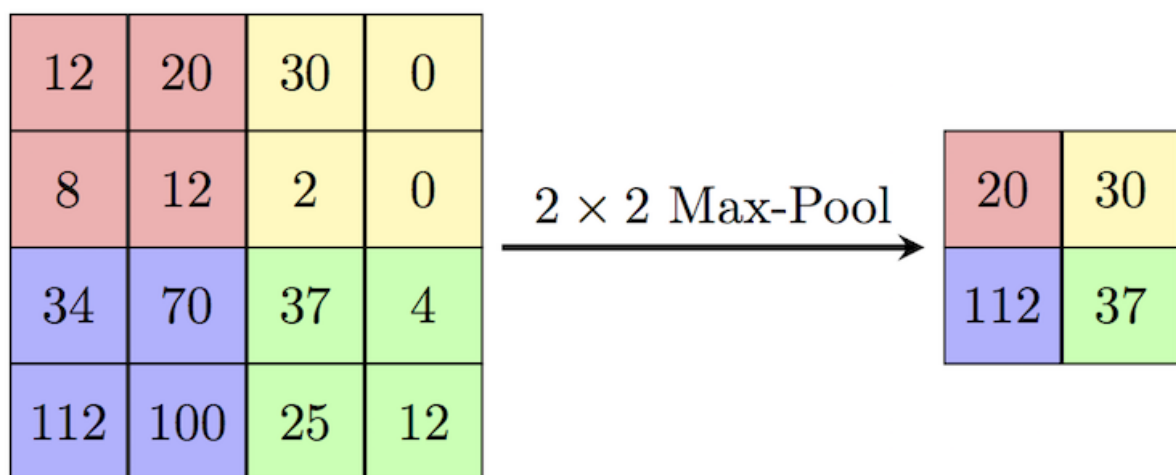


Fig.(5) demonstration of a max pooling layer

To reduce the number of parameters in a network you use a pooling layer; this layer takes only the maximum number from a region.

By convention since this level has no parameters nor weights is coupled with the convolution layer to create a layer

A possible substitute is the average pooling that takes the average of a region.

3.3.3. Fully connected layer (FC)

compose of single connected layer as we saw in the explanation of neural network in this layer every pixel or data is connect to the next layer

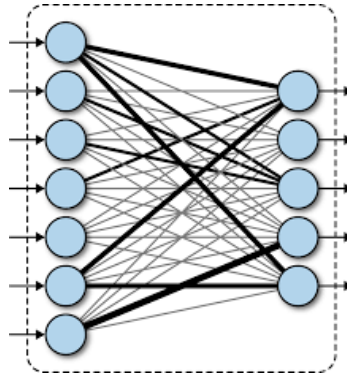


Fig.(6) fully connected layer

3.3.4. UpSampling

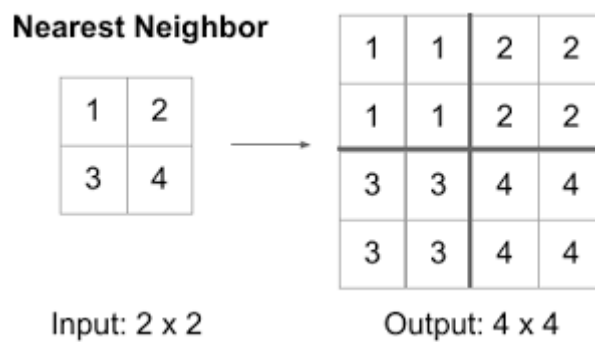


Fig.(7) Upsampling

upsampling is not a real level as it has no weights and bias but to make the image the same size as the original, in this case we use the simplest configuration with the algorithm of nearest neighbor

3.4. Evaluation Metrics

By following the convention of existing works for crowd counting[19], I evaluate the method with both the absolute error (MAE) and the mean squared error (MSE), which are defined as follows:

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

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

where N is the number of test images, z_i is the actual number of people in the i th image, and \hat{z}_i is the estimated number of people in the i th image. Roughly speaking, MAE indicates the accuracy of the estimates, and MSE indicates the robustness of the estimates

3.5. DATASET

3.5.1. UCF-QNRF_ECCV18

Is one of the largest dataset to-date (in terms of number of annotations) for training and evaluating crowd counting and localization methods [21]. It contains 1535 images which are divided into train and test sets of 1201 and 334 images respectively. The dataset is most suitable for training very deep Convolutional Neural Networks (CNNs) since it contains order of magnitude more annotated humans in dense crowd scenes than any other available crowd counting dataset.

The UCF-QNRF dataset has the most number of high-count crowd images and annotations, and a wider variety of scenes containing the most diverse set of viewpoints, densities and lighting variations. The resolution is large compared to WorldExpo10 and ShanghaiTech.

The average density, i.e., the number of people per pixel over all images is also the lowest, signifying high-quality large images. Lower per-pixel density is partly due to inclusion of background regions, where there are many high-density regions as well as zero-density regions. Part A of Shanghai dataset has high-count crowd images as well, however, they are severely cropped to contain crowds only. On the other hand, the new UCF-QNRF dataset contains buildings, vegetation, sky and roads as they are present in realistic scenarios captured in the wild. This makes this dataset more realistic as well as difficult.

4. ALGORITHM

The algorithm i want to try it is based on the paper [24] a fully connected convolutional neural network based on single-column

One of the key aspects of fully convolutional nets that makes the method particularly suited to crowd counting is the use of a variable size input, allowing the model to avoid the loss of detail and visual distortions typically encountered during image downsampling and reshaping.

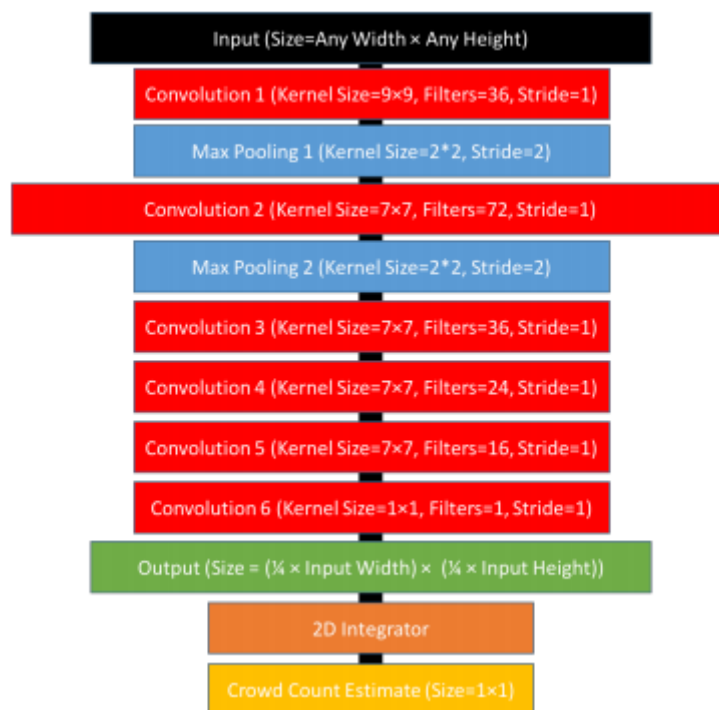


Fig.(8) Architecture of the Fully Convolutional Network [24]

the algorithm developed consist of: (1) A training set augmentation scheme that minimises redundancy among training samples to improve model generalisation and overall counting performance; (2) a deep, single column, fully convolutional network (FCN) architecture; (3) a multi-scale averaging step during inference[X]

The network designed is a 6 layer, single column FCN as illustrated in figure. This network contains just 315,000 parameters.

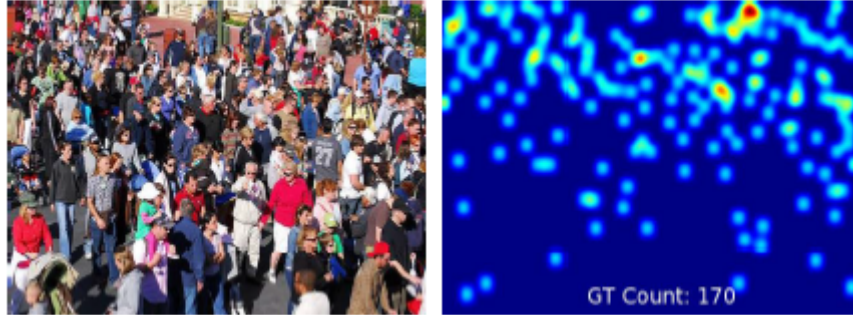


Fig.(9-10) Input image and expected output [24]

The expected output is a corresponding density map with the count of people .

5. IMPLEMENTATION

After analyzing the state of the art completed the basic learning of convolutional networks the next steps will be to realize the neural network and test its effectiveness realize the density map and assess the weaknesses of the algorithm.

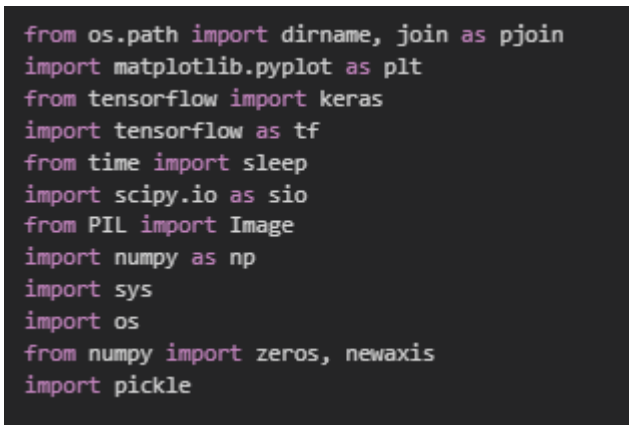
5.1. CODE

The first step was to figure out which library to use to develop the neural network.

the choice fell between pythorch or keras/tensorflow after careful consultation of libraries and on the advice of several colleagues I found that pythorch is currently the most popular library especially in research and development but given the recent increase in use lacks a community or external support to the official documentation so my choice fell on keras

As for the preprocessing of the data I used Numpy, Pickle, PIL.

In retrospect ,I can see that the keras/tensor flow library has many elements that would have been more comfortable to use for preprocessing, This error of choice certainly made me lengthen my time but at the same time taught manual memory management and greater knowledge of python.



```
from os.path import dirname, join as pjoin
import matplotlib.pyplot as plt
from tensorflow import keras
import tensorflow as tf
from time import sleep
import scipy.io as sio
from PIL import Image
import numpy as np
import sys
import os
from numpy import zeros, newaxis
import pickle
```

Fig.(11) libraries used for the development

5.1.1. PREPROCESSING

the first step of the project is to load the datasets and view the data available

Upload images and related data with the location of each person within the image



Fig.(12) input image

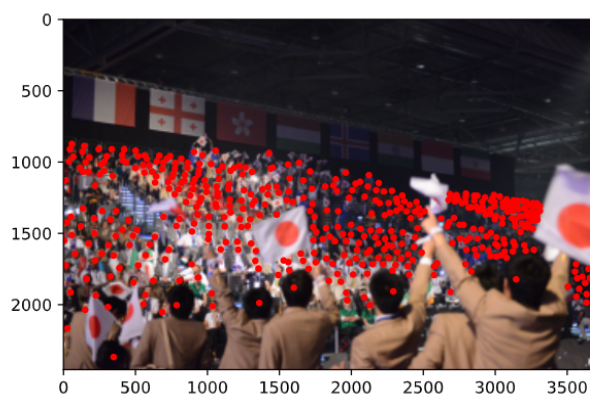


Fig.(13) representation of heads position on the image

In the next lines of code I load all the images progressively and for every image I create the relative heatmap.

each head is represented as a Gaussian function in 2d because with a simple 2d integration of the entire image will result in the number of heads as each Gaussian has sum 1 as its integral

to define the gaussian function we rely on the reference paper that to define the function establishes σ spread gaussian parameter as

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

where \overline{d}_i gave the average distance between the i^{th} and the 5 nearest heads, while the value 0,3 is empirically derived from the work[22]

```
#CREATE HEATMAPS
path_dataset=(ROOT_DIR + "\\dataset\\UCF-QNRF-ECCV18\\Train\\")#directory

mappa=[]
dataset_images_permentant=[]
NumberOfImage=1202 #1202
#ALL IMAGES PROCESSING
for imm in range(1,NumberOfImage):
    ProgressBar(imm,NumberOfImage,"heatmap create")
    num=""
    if imm<10:
        num='000'+str(imm)
    elif imm<100:
        num="00"+str(imm)
    elif imm<1000 :
        num="0"+str(imm)
    else:
        num=imm
    dataname= 'img_'+str(num)+'_ann.mat'
    imagenname= 'img_'+str(num)+'_jpg'
    heatmapname= 'heatmap_'+str(num)+'_pkl'
    mat_fname = pjoin(path_dataset, dataname)
    mat_contents = sio.loadmat(mat_fname)
    dataset_values = mat_contents
    dataset_images = np.array(Image.open(path_dataset+imagenname))
    dataset_images_permentant.append(dataset_images)
    corner=np.array(dataset_values['annPoints'])
    corner_shape=corner.shape
    #DISPLAY HEADS
    # imshow(dataset_images)
    # displaycorners(dataset_images,dataset_values['annPoints'])
    #FIND THE 5 HEAD MORE CLOSER AND CALCULATE AVERAGE DISTANCE
    dist = np.zeros((corner_shape[0],corner_shape[0]))
    for i in range(0,corner_shape[0]):
        for j in range(0,corner_shape[0]):
            dist[i][j] = np.linalg.norm(corner[i]-corner[j])
    distance = []
    for k in range(0,corner_shape[0]):
        distance.append(np.sum(np.sort(dist[k])[2:8])/6)
    #CREATE HEATMAP
    t=(createHeatMap(distance,corner,dataset_images))
    #SAVE HEATMAP
    imshow(t)
    save_path= pjoin('dataset\\UCF-QNRF-ECCV18\\heatmapencoded\\', heatmapname)
    with open(save_path,'ab') as f:
        pickle.dump(t,f)
    del distance,corner,t
    ProgressBar(NumberOfImage,NumberOfImage,"heatmap create")#Just a visual reference for the progress
```

Fig.(14) algorithm for distance calculation and store of the images and heatmaps

```

#CREATE HEAT IMAGE
def createHeatMap(dist,heads,data):
    shape=(data.shape[0],data.shape[1])
    image=np.zeros(shape)
    for k in range(0,len(heads)):
        sigma=dist[k]*0.3
        mux=int(heads[k][0])
        muy=int(heads[k][1])
        n2 = np.int(3*sigma)
        x,y = np.meshgrid(np.arange(-n2,n2+1),np.arange(-n2,n2+1))
        kern = (1/sigma*np.sqrt(2*np.pi))*np.exp(-(x**2+y**2)/(2*sigma*sigma))
        kern=kern/kern.sum()
        for cord_x in range(-n2+mux,n2+1+mux):
            for cord_y in range(-n2+muy,n2+1+muy):
                if(cord_x>0 and cord_x<image.shape[1] and cord_y>0 and cord_y<image.shape[0]):
                    image[cord_y][cord_x]+=kern[+n2+cord_x-mux][+n2+cord_y-muy]
    return image

```

Fig.(15) algorithm for heatmap creation

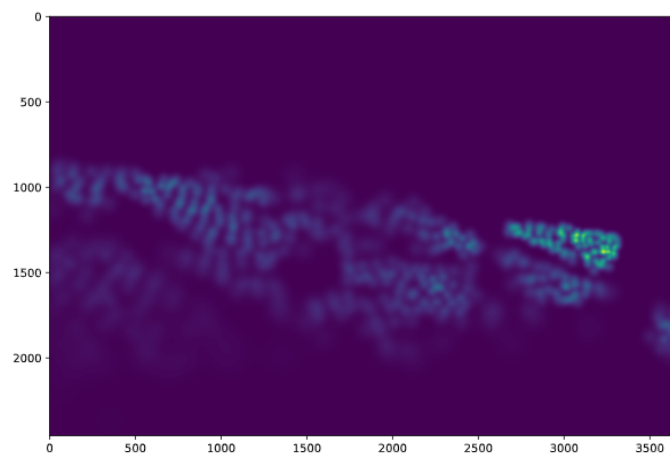


Fig.(16) heatmap created of the input image

after creating images you need a generator to load images and heatmaps to be inserted in a progressive way

```
def generator_(start,end,n,First):
    while 1:
        if(First):
            n=start
            First=False
        imm=n
        path_dataset=("G:\\PROGETTO IMAGE\\heatmapencoded\\")#directory
        # path_dataset=(ROOT_DIR + "\\dataset\\UCF-QNRF_ECCV18\\heatmapencoded\\")#directory
        num=""
        if imm<10:
            num='000'+str(imm)
        elif imm<100:
            num="00"+str(imm)
        elif imm<1000 :
            num="0"+str(imm)
        else:
            num=imm
        imagename= 'img_'+str(num)+'.jpg'
        name = 'img_'+str(num)+'.pkl'
        heatmapname = 'heatmap_'+str(num)+'.pkl'
        HEATMAP=read(heatmapname,True,path_dataset)
        IMAGE=read(name,False,path_dataset)
        IMAGE=np.expand_dims(IMAGE, axis=0)
        HEATMAP=np.expand_dims(HEATMAP, axis=0)
        while(HEATMAP.shape!=(1, 500, 500, 1) or IMAGE.shape!=(1, 500, 500, 3)or (n in list_of_redo)):
            n+=1
            imm=n
            num=""
            if imm<10:
                num='000'+str(imm)
            elif imm<100:
                num="00"+str(imm)
            elif imm<1000:
                num="0"+str(imm)
            else:
                num=imm
            imagename= 'img_'+str(num)+'.jpg'
            name = 'img_'+str(num)+'.pkl'
            heatmapname = 'heatmap_'+str(num)+'.pkl'
            HEATMAP=read(heatmapname,True,path_dataset)
            IMAGE=read(name,False,path_dataset)
            IMAGE=np.expand_dims(IMAGE, axis=0)
            HEATMAP=np.expand_dims(HEATMAP, axis=0)
            n+=1
        if(n==end):
            n=start
        yield IMAGE,HEATMAP
```

Fig.(17) generator for load image and heatmap during fit of the model

```
def read(path_image,heat,path_dataset):
    save_path= pjoin(path_dataset, path_image)
    with open(save_path,'rb') as f:
        try:
            imag=np.array(pickle.load(f))
            x,y,*z = imag.shape
            if(heat==False):
                imag=(crop(imag/255,500,500))
            else:
                imag=(crop(np.expand_dims(imag/255, axis=2),500,500))
        except EOFError:
            return 0
        return imag

def readOnly(path_image,path_dataset):
    save_path= pjoin(path_dataset, path_image)
    with open(save_path,'rb') as f:
        try:
            imag=np.array(pickle.load(f))
        except EOFError:
            return 0
        return imag

def write(name,imageToSave,path_dataset):
    save_path= pjoin(path_dataset, name)
    with open(save_path,'ab') as f:
        pickle.dump(imageToSave,f)
```

Fig.(18) function for read and save images from binary file

Here all the functions used for writing and reading data at the same time images are normalized to values between 0 and 1.

```
#NORMALIZE IMAGE
def normalizeImage(I,k1,k2):
    newArray = []
    for rows in I:
        row = []
        for pixel in rows:
            row.append(float(((255/(k2-k1))*(pixel-k1))/255))
        newArray.append(row)
    return np.array(newArray)

#NORMALIZE IMAGE COLOR
def normalizeImage3Color(I,k1,k2):
    image= np.array(I)
    image[:, :, :] = ((255/(k2-k1))*(image[:, :, :]-k1))
    return I
```

Fig.(19) function for Normalize images

In addition, images and heatmaps are cropped with size 500*500 with images divided into 3 RGB color channels while heatmaps on a single channel.

unlike the reference paper where the images are cropped 1000*1000 I chose a smaller size for memory problems

```
#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
```

Fig.(20) function for create a single image or multiple with predetermine dimension from dataset

Then I create the network model based on the image

```
#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 )
])
```

Fig.(21) keras model of the Network

in the paper model [24] no padding is inserted and each convolutional layer has as activation function Relu, also the weights of each level are preset with a Gaussian function with standard deviation =0,01

Finally, through two layers of upsampling return to an output with the same size of the input image to have no change in the position in the heatmaps

```
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'])
```

Fig.(22) loss function and optimizer for learning process of the network

as the paper suggests I use SGD stochastic gradient Descent as optimizer, with the relative learning rate momentum and decay, this particular optimizer has like characteristic that Stochastic gradient Descent maintains to single learning rate (termed alpha) for all weight updates and the learning rate does not change During training

Given the limited availability of time I processed the algorithm only on a limited number of images

```
trainGen = generator_(1,900,1,True)
valGen = generator_(900,1000,900,True)

print("start")
model.fit(
    x = trainGen,
    validation_data = valGen,
    validation_steps=1,
    batch_size=1,
    steps_per_epoch =1,
    epochs=900,
    verbose=1,
    shuffle=True,
)
print("end")
```

Fig.(23) start the learning process with batch size 1

for reasons of time and memory each batch consists of only one image

additional functions used to test and verify the code to display them reference to the code linked to the processed

5.2. PROBLEMS FACED

given the little experience I had to face many problems regarding data storing, memory management , python usage,compiling time.

the first problem faced was to save the heatmap created as images through the PIL library, as PIL reworks images using compression and thus distorting all values

Then for the learning of the neural network I loaded all the necessary images as variables causing the use of all the RAM of the server.

So I used the generators, and here the error was not using the keras functions for image acquisition thus complicating the code

although I was able to find a solution for all the previous problems the time to create heatmaps, the processing and learning process of the neural network are very long and I could not do many tests and not even use the whole dataset

Given the above problems it was not possible to tow the neural network with a lot of data, an image of 500*500 size was taken as a sample from each image of the dataset and its heatmap.

6. RESULTS

Despite the few training iterations the results are encouraging for the identification of people within the images, but not very precise.

It is possible to distinguish especially in images with very isolated people that the network can distinguish the position of people

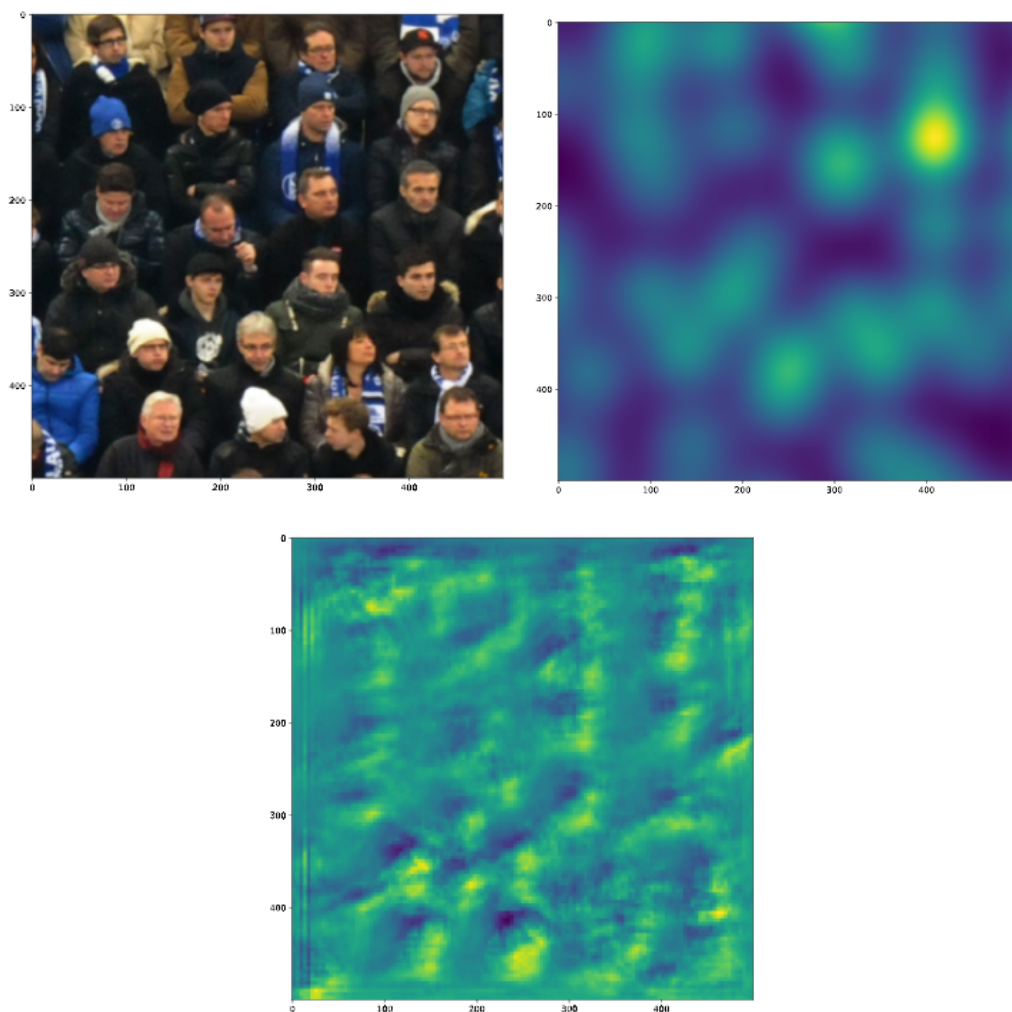


Fig.(24-25-26) input expected output and result

the values of mae and mse are not comparable with the paper because the very noisy images in fact it can be notice that there are no zones with zero values as it may be and this results in metrics not comparable

also creates outputs with very high values and this means that after the integral of the resulting heat map has values of different scales of magnitude different from those expected

7. CONCLUSIONS

The whole project proved to be very complex due to the difficulty of algorithms and data processing.

This took longer than I thought.

The difficulties encountered did not lead to an algorithm able to count people with the low number of iterations but came very close to predictions very close to the groundtruth as shape but not as values

The detail that seems invalid to me is that the predicted heat maps have negative values.

Another detail to increase the efficiency of the algorithm is in the creation of the heatmaps to tow the network as having a circular shape around the position of the person also introduces the overlaps as verified data going therefore to introduce noise.

a simple correction for reduce this problem is enter different values for σ_x and σ_y factor when creating the heatmaps

The most important aspect of this project was the ability to learn how to use neural networks and learn the basics of the algorithms on which they are based and the various layers

also I was able to strengthen my knowledge of python and being the first project also understand the importance of using short functions and to test its behavior to avoid problems in the continuation of projects

8. REFERENCES

- [1] D. I. Hawkins and D. L. Mothersbaugh, Consumer behavior: Building marketing strategy. McGraw-Hill Education, 2015.

-
- [2] V. Nogueira, H. Oliveira, J. Augusto Silva, T. Vieira and K. Oliveira, "RetailNet: A Deep Learning Approach for People Counting and Hot Spots Detection in Retail Stores," *2019 32nd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, Rio de Janeiro, Brazil, 2019, pp. 155-162, doi: 10.1109/SIBGRAPI.2019.00029.
- [3] G. J. Brostow and R. Cipolla, "Unsupervised Bayesian detection of independent motion in crowds," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, Jun. 2006, pp. 594–601.
- [4] J. García, A. Gardel, I. Bravo, J. L. Lázaro, M. Martínez, and D. Rodríguez, "Directional people counter based on head tracking," *IEEE Trans. Ind. Electron.*, vol. 60, no. 9, pp. 3991–4000, Sep. 2013.
- [5] V. Rabaud and S. Belongie, "Counting crowded moving objects," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2006, pp. 705–711.
- [6] A. G. Vicente, I. B. Munoz, P. J. Molina, and J. L. L. Galilea, "Embedded vision modules for tracking and counting people," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 9, pp. 3004–3011, Sep. 2009.
- [7] J. Zhu, F. Feng and B. Shen, "People counting and pedestrian flow statistics based on convolutional neural network and recurrent neural network," *2018 33rd Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, Nanjing, China, 2018, pp. 993-998, doi: 10.1109/YAC.2018.8406516.
- [8] S. Liu, S. Zhai, C. Li and J. Tang, "An effective approach to crowd counting with CNN-based statistical features," *2017 International Smart Cities Conference (ISC2)*, Wuxi, 2017, pp. 1-5, doi: 10.1109/ISC2.2017.8090827.
- [9] D. Fehr, R. Sivalingam, V. Morellas, N. Papanikolopoulos, O. Lotfallah and Y. Park, "Counting People in Groups," *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, Genova, Italy, 2009, pp. 152-157, doi: 10.1109/AVSS.2009.55.
- [10] C. Lien, Y. Huang and C. Han, "People Counting Using Multi-Mode Multi-Target Tracking Scheme," *2009 Fifth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, Kyoto, Japan, 2009, pp. 1018-1021, doi: 10.1109/IIH-MSP.2009.239.
- [11] D. Conte, P. Foggia, G. Percannella, F. Tufano and M. Vento, "Counting Moving People in Videos by Salient Points Detection," *2010 20th International Conference on Pattern Recognition*, Istanbul, Turkey, 2010, pp. 1743-1746, doi: 10.1109/ICPR.2010.431.

-
- [12] S. In Cho, "Vision-Based People Counter Using CNN-Based Event Classification," in *IEEE Transactions on Instrumentation and Measurement*, vol. 69, no. 8, pp. 5308-5315, Aug. 2020, doi: 10.1109/TIM.2019.2959853.
- [13] J. Fu, H. Yang, P. Liu and Y. Hu, "A CNN-RNN Neural Network Join Long Short-Term Memory For Crowd Counting and Density Estimation," *2018 IEEE International Conference on Advanced Manufacturing (ICAM)*, Yunlin, Taiwan, 2018, pp. 471-474, doi: 10.1109/AMCON.2018.8614939.
- [14] H. H. Lin and K. Thi Win, "People Counting System with C-Deep Feature in Dense Crowd Views," *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, Singapore, 2018, pp. 451-456, doi: 10.1109/ICIS.2018.8466440.
- [15] H. H. Lin and K. Thi Win, "People Counting System with C-Deep Feature in Dense Crowd Views," *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, Singapore, 2018, pp. 451-456, doi: 10.1109/ICIS.2018.8466440.
- [16] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.
- [17] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, USA, 2017, pp. 6517-6525, doi: 10.1109/CVPR.2017.690.
- [18] H. Idrees, M. Tayyab, K. Athrey, D. Zhang, S. Al-Maddeed, N. Rajpoot, M. Shah, *Composition Loss for Counting, Density Map Estimation and Localization in Dense Crowds*, in *Proceedings of IEEE European Conference on Computer Vision (ECCV 2018)*, Munich, Germany, September 8-14, 2018.
- [19] Gao, G., Gao, J., Liu, Q., Wang, Q., & Wang, Y. (2020). *CNN-based Density Estimation and Crowd Counting: A Survey*. ArXiv, abs/2003.12783.
- [20] H. Idrees, I. Saleemi, C. Seibert and M. Shah, "Multi-source Multi-scale Counting in Extremely Dense Crowd Images," *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Portland, OR, USA, 2013, pp. 2547-2554, doi: 10.1109/CVPR.2013.329.
- [21] S. Ali and M. Shah, "A Lagrangian Particle Dynamics Approach for Crowd Flow Segmentation and Stability Analysis," *2007 IEEE Conference on Computer Vision and Pattern Recognition*, Minneapolis, MN, USA, 2007, pp. 1-6, doi: 10.1109/CVPR.2007.382977.

-
- [22] Y. Zhang, D. Zhou, S. Chen, S. Gao and Y. Ma, "Single-Image Crowd Counting via Multi-Column Convolutional Neural Network," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 589-597, doi: 10.1109/CVPR.2016.70
- [23] Victor Lempitsky and Andrew Zisserman. 2010. Learning To count objects in images. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 1 (NIPS'10)*. Curran Associates Inc., Red Hook, NY, USA, 1324–1332.
- [24] Marsden, M., McGuinness, K., Little, S., and O'Connor, N. E., "Fully Convolutional Crowd Counting On Highly Congested Scenes", arXiv:1612.00220, 2016.

9. SITOGRAPHY

- I) <http://introtodeeplearning.com/>
- II) <https://www.coursera.org/specializations/deep-learning>
- III) <https://www.deeplearningbook.org/>
- IV) <https://stackoverflow.com/> infinite thanks