Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем і технологій

Лабораторна робота №6

Проектування та реалізація програмних систем з нейронними мережами

Згорткові нейронні мережі типу Xception

Київ 2023

Завдання: написати програму що реалізує згорткову нейронну мережу Xception для розпізнавання об'єктів на відео. Створити власний дата сет з папки на диску, навчити нейронну мережу на цьому датасеті розпізнавати логотип вашого улюбленого бренду, скажімо Apple чи BMW. Навчену нейронну мережу зберегти на комп'ютер написати програму, що відкриває та аналізує відео, результат – час на якому з'являвся логотип.

Програмний код:

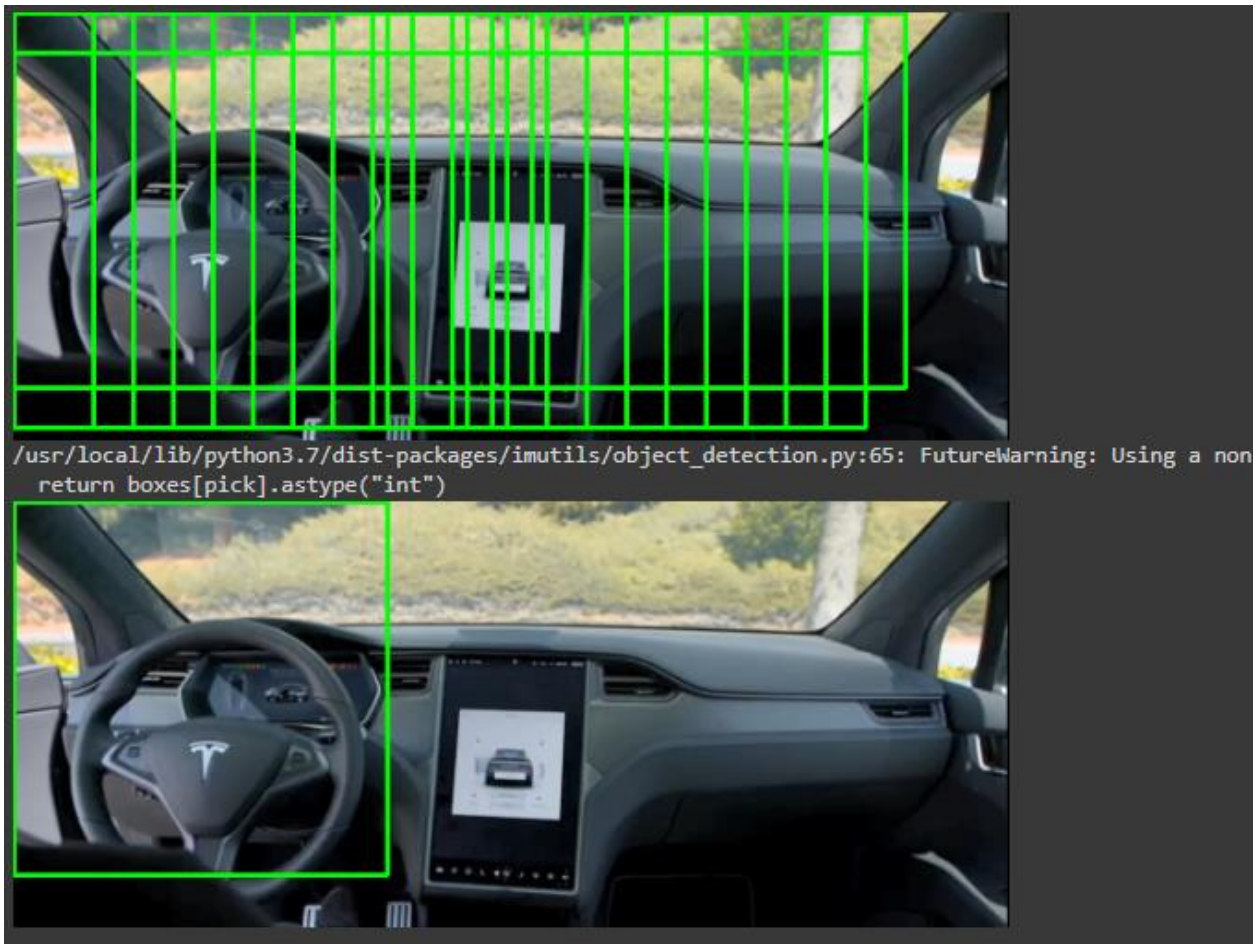
```
import tensorflow as tf
tf.test.gpu_device_name()

from tensorflow.python.client import device_lib
device_lib.list_local_devices()

from google.colab import drive
drive.mount('/content/gdrive', force_remount=True)

NN_name = "NNetwork_Lab_6.h5"
from keras.models import load_model
modelSaved =
load_model("/content/gdrive/MyDrive/ColabNotebooks/"+NN_name)

from tensorflow.keras.utils import img_to_array
import numpy as np
from imutils.object_detection import non_max_suppression
from google.colab.patches import cv2_imshow
import cv2
def processImageFunc(frame):
  flag = False
  img = frame
  img = np.resize(img, (1,120,120,3))
  image = np.array([item[0] for item in img])
  image = img.astype('float32')
  image /= 255
  '''imgarray = np.asarray(img)
  trainD = []
  trainD.append((img,1))
  image = np.array([item[0] for item in trainD])
  image = image.astype('float32')
  image /= 255'''
  prediction = modelSaved.predict(image)
  if(prediction>0.4):
    print("Cool!")
    flag = True
  return flag
```

```
video_capture =
cv2.VideoCapture('/content/gdrive/MyDrive/ColabNotebooks/tesla.m
p4')
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
file_size = (1024, 640)
frame_count = 0
while (video_capture.isOpened()):
  frame_count+=1
  ret, frame = video_capture.read()
  if ret:
   ismers = processImageFunc(frame)
   if ismers:
     print(frame_count)
     print(frame_count/30)
   if cv2.waitKey(1) & 0xFF == ord('q'):
     break
  else:
   break
print(frame_count)
video_capture.release()

from PIL import Image
orig =
Image.open('/content/gdrive/MyDrive/ColabNotebooks/tesla.jpg').c
onvert('RGB')
orig = orig.resize((120, 120))
imgarray = np.asarray(orig)
trainD = []
trainD.append((imgarray,1))
image = np.array([item[0] for item in trainD])
image = image.astype('float32')
image /= 255
print(image.shape)
prediction = modelSaved.predict(image)

import imutils
def sliding_window(image, step, ws):
  # slide a window across the image
  for y in range(0, image.shape[0] - ws[1], step):
    for x in range(0, image.shape[1] - ws[0], step):
      # yield the current window
      yield (x, y, image[y:y + ws[1], x:x + ws[0]])

def image_pyramid(image, scale=1.5, minSize=(120, 120)):
  # yield the original image
  yield image
  # keep looping over the image pyramid
  while True:
    # compute the dimensions of the next image in the pyramid
    w = int(image.shape[1] / scale)
    image = imutils.resize(image, width=w)
    # if the resized image does not meet the supplied minimum
```

```
    # size, then stop constructing the pyramid
    if image.shape[0] < minSize[1] or image.shape[1] <
minSize[0]:
      break
    # yield the next image in the pyramid
    yield image

from tensorflow.keras.utils import img_to_array
import numpy as np
from imutils.object_detection import non_max_suppression
from google.colab.patches import cv2_imshow
import cv2
def processImage(frame):
  WIDTH = 600
  PYR_SCALE = 1.5
  WIN_STEP = 16
  ROI_SIZE = (100, 100)
  INPUT_SIZE = (120, 120)
  orig = frame
  orig = imutils.resize(orig, width=WIDTH)
  (H, W) = orig.shape[:2]
  pyramid = image_pyramid(orig, scale=PYR_SCALE,
minSize=ROI_SIZE)
  rois = []
  locs = []
  for image in pyramid:
    # determine the scale factor between the *original* image
    # dimensions and the *current* layer of the pyramid
    scale = W / float(image.shape[1])
    # for each layer of the image pyramid, loop over the sliding
    # window locations
    for (x, y, roiOrig) in sliding_window(image, WIN_STEP,
ROI_SIZE):
      # scale the (x, y)-coordinates of the ROI with respect to
the
      # *original* image dimensions
      x = int(x * scale)
      y = int(y * scale)
      w = int(ROI_SIZE[0] * scale)
      h = int(ROI_SIZE[1] * scale)
      roi = cv2.resize(roiOrig, INPUT_SIZE)
      roi = roi.reshape((1,120,120,3))#(1,224,224,3)
      #roi = img_to_array(roi)
      roi = roi.astype('float32')/255
      # update our list of ROIs and associated coordinates
      rois.append(roi)
      locs.append((x, y, x + w, y + h))
    res = []

    for i in range(len(rois)):
      a = modelSaved.predict(rois[i])
      if a[0] > 0.95:
        res.append([locs[i],a[0]])
```

```
  return len(res)

video_capture =
cv2.VideoCapture('/content/gdrive/MyDrive/ColabNotebooks/tesla.m
p4')
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
file_size = (1024, 640)
frame_count = 0
while (video_capture.isOpened()):
 frame_count+=1
 if(frame_count%3 == 0):
  ret, frame = video_capture.read()
  print(frame_count)
  if ret:
   ismers = processImage(frame)
   if ismers>0:
     print(frame_count)
     print(frame_count/30)
   if cv2.waitKey(1) & 0xFF == ord('q'):
     break
  else:
   break
print(frame_count)
video_capture.release()

from tensorflow.keras.utils import img_to_array
import numpy as np
from imutils.object_detection import non_max_suppression
from google.colab.patches import cv2_imshow
import cv2
WIDTH = 600
PYR_SCALE = 1.5
WIN_STEP = 16
ROI_SIZE = (150, 150)
INPUT_SIZE = (120, 120)
orig =
cv2.imread('/content/gdrive/MyDrive/ColabNotebooks/PZPaRNN/tesla
.jpg')
orig = imutils.resize(orig, width=WIDTH)
(H, W) = orig.shape[:2]
pyramid = image_pyramid(orig, scale=PYR_SCALE, minSize=ROI_SIZE)

rois = []
locs = []
for image in pyramid:
# determine the scale factor between the *original* image
# dimensions and the *current* layer of the pyramid
  scale = W / float(image.shape[1])
# for each layer of the image pyramid, loop over the sliding
# window locations

for (x, y, roiOrig) in sliding_window(image, WIN_STEP,
ROI_SIZE):
```

```
      # scale the (x, y)-coordinates of the ROI with respect to
the
      # *original* image dimensions
      x = int(x * scale)
      y = int(y * scale)
      w = int(ROI_SIZE[0] * scale)
      h = int(ROI_SIZE[1] * scale)
      roi = cv2.resize(roiOrig, INPUT_SIZE)
      roi = roi.reshape((1,120,120,3))#(1,224,224,3)
      #roi = img_to_array(roi)
      roi = roi.astype('float32')/255
      # update our list of ROIs and associated coordinates
      rois.append(roi)
      locs.append((x, y, x + w, y + h))

res = []
for i in range(len(rois)):
  a = modelSaved.predict(rois[i])
  #print(a)
  if a[0] > 0.95:
    #print(locs[i])
    #print(f'a = {a}')
    res.append([locs[i],a[0]])

import cv2
from google.colab.patches import cv2_imshow
for label in range(1):
      clone = orig.copy()
      for (box, prob) in res:
        # draw the bounding box on the image
        (startX, startY, endX, endY) = box
        cv2.rectangle(clone, (startX, startY), (endX, endY), (0,
255, 0), 2)
      boxes = np.array([p[0] for p in res])
      proba = np.array([p[1] for p in res])
      # applying non-maxima suppression
      cv2_imshow( clone)
      clone = orig.copy()
      boxes = non_max_suppression(boxes, proba)
      # loop over all bounding boxes that were kept after
applying
      for (startX, startY, endX, endY) in boxes:
        # draw the bounding box and label on the image
        cv2.rectangle(clone, (startX, startY), (endX, endY), (0,
255, 0), 2)
        y = startY - 10 if startY - 10 > 10 else startY + 10
      cv2_imshow(clone)
```



Висновки

Таким чином, в наслідок виконання даної лабораторної роботи у середовищі розробки програмного забезпечення “Google Colaboratory” було проведено створення та навчання моделі згорткової нейронної мережі Xception для розпізнавання зображень логотипу компанії Tesla. Варто зазначити, що остаточна точність оцінки зображення для тестових даних є досить висока, що дає змогу досить точно визначати час появи логотипу на відео.