МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”

КАФЕДРА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ

ЗВІТ

про виконання лабораторної роботи №4

з дисципліни: «Обробка зображень методами штучного інтелекту»

на тему: «Детекція об’єктів»

Виконав:

студент групи КН-410

Шиманський П.С.

Прийняв:

Пелешко Д.Д.

Львів – 2021

Мета: навчитись вирішувати задачу детекутвання об’єктів.

Теоретичні відомості

Region-CNN [5] (R-CNN, Region-based Convolutional Network) - алгоритм, заснований на свёрточних нейронних мережах. Замість того, щоб використовувати для пошуку зображень ковзаючі вікна фіксованого розміру, на першому кроці алгоритм намагається знайти селективним пошуком "регіони" - прямокутні рамки різних розмірів, які, імовірно, містять об'єкт. Це забезпечує більш швидке і ефективне знаходження об'єктів незалежно від розміру об'єкта, відстані до камери, кута зору.

Сумарна кількість регіонів для кожного зображення, згенерованих на першому кроці, приблизно дорівнює двом тисячам. Знайдені регіони за допомогою афінних перетворень набувають розмір, який потрібно подати на вхід CNN. Також замість афінних перетворень можна використовувати паддінгі, або розширювати обмежують рамки до розмірів, необхідних для входу CNN. Як CNN часто використовується архітектура CaffeNet [6], витягають для кожного регіону порядку 4096 ознак. На останньому етапі вектора ознак регіонів обробляються SVM, які проводять класифікацію об'єктів, по одній SVM на кожен домен.

Селективний пошук, в свою чергу, теж можна навчати за допомогою лінійної регресії параметрів регіону - ширини, висоти, центру. Цей метод, названий bounding-box regression, дозволяє більш точно виділити об'єкт. В якості даних для регресії використовуються ознаки, отримані в результаті роботи CNN.

Код програми

```
class TDMModule(layers.Layer):
  def __init__(self, filters, bottom_up_kernel_size=3,
```

```
            top_down_kernel_size=3, activation='relu'):
        super(TDMModule, self).__init__()
        self.bottom_up_conv = layers.Conv2D(filters, bottom_up_kernel_size,
                                            padding='same', activation=activation)
        self.top_down_conv = layers.Conv2D(filters, top_down_kernel_size,
                                           padding='same', activation=activation)
        self.res_conv = layers.Conv2D(filters*2, 1, activation=activation)
    def call(self, bottom_up_features, top_down_features):
        new_bottom_up_features = self.bottom_up_conv(bottom_up_features)
        new_top_down_features = self.top_down_conv(top_down_features)
        new_top_down_features = layers.UpSampling2D((2, 2))(new_top_down_features)
        concatenated_features = layers.Concatenate()([new_top_down_features, new_botto
m_up_features])
        res = self.res_conv(concatenated_features)
        return res
class TDMModule(layers.Layer):
    def __init__(self, filters, bottom_up_kernel_size=3,
            top_down_kernel_size=3, activation='relu'):
        super(TDMModule, self).__init__()
        self.bottom_up_conv = layers.Conv2D(filters, bottom_up_kernel_size,
                                            padding='same', activation=activation)
        self.top_down_conv = layers.Conv2D(filters, top_down_kernel_size,
                                           padding='same', activation=activation)
        self.res_conv = layers.Conv2D(filters*2, 1, activation=activation)
    def call(self, bottom_up_features, top_down_features):
        new_bottom_up_features = self.bottom_up_conv(bottom_up_features)
        new_top_down_features = self.top_down_conv(top_down_features)
        new_top_down_features = layers.UpSampling2D((2, 2))(new_top_down_features)
```

```
        concatenated_features = layers.Concatenate()([new_top_down_features, new_botto
m_up_features])
        res = self.res_conv(concatenated_features)
        return res
```