НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра прикладної математики

Звіт

із переддипломної практики

на тему:  
Створення математичного та програмного забезпечення для системи приховання облич та реклами у реальному часі

|  |  |
| --- | --- |
| Виконав: | Керівник від університету: |
| студент групи КМ-73 | доцент кафедри ПМА |
| Тищенко Б.Ю. | Олефір О.С. |
|  | Керівник від підприємства: |
|  | доцент кафедри ПМА |
|  | Олефір О.С. |

Київ – 2021

Анотація

Даний звіт присвячено результатам проходження переддипломної практики на тему «Створення математичного та програмного забезпечення для системи приховання облич та реклами у реальному часі», яка відбувалася з 12 квітня по 16 травня 2021 року на базі моєї квартири. У рамках практики поставлено задачу на дипломне проектування, сформульовано критерії вибору методу розв’язання поставленої задачі.

Розглянуто ряд методів для детекції об’єктів побудованих на основі згорткових нейронних мереж: Мask-RCNN, MTCNN та методи родини YOLO: YOLO, YOLOv3, YOLOv5, YOLOFace, YOLACT. На основі сформульованих критеріїв для розв’язання поставленої задачі вибрано метод MТCNN для детекції облич та метод YOLOv3 для детекції рекламних банерів.

Abstract

The report is devoted to the results of the Pre-diploma Practice titled “Creating the mathematical model and the software for the real-time face and advertising obscuring system” that took place from April 12 to May 16, 2021 at my apartment. During the practice, the task for the thesis is set; criteria for choosing the method for solving this task are formulated.

A number of convolutional neural networks based object detection methods are discussed. They are Мask-RCNN, MTCNN and several methods from the YOLO family: YOLO, YOLOv3, YOLOv5, YOLOFace, YOLACT. According to the formulated criteria, MTCNN is chosen for solving the face detection task, and the YOLOv3 is chosen for solving the advertising detection task.

ЗМІСТ

[List of abbreviations 2](#_Toc73327617)

[Motivation 3](#_Toc73327618)

[1 Task description 4](#_Toc73327619)

[2 Аналіз існуючих методів розв’язання поставленої задачі 5](#_Toc73327620)

[3 Description of the implemented methods 7](#_Toc73327621)

[Neural network 7](#_Toc73327622)

[Convolutional neural network 8](#_Toc73327623)

[Convolutional layer 8](#_Toc73327624)

[Pooling layer 9](#_Toc73327625)

[Повнозв’язний шар 10](#_Toc73327626)

[Dropout layer 10](#_Toc73327627)

[3 YOLO algorithm 11](#_Toc73327628)

[3 MTCNN algorithm 11](#_Toc73327629)

[4 Програмна реалізація 14](#_Toc73327630)

[Software structure model 14](#_Toc73327631)

[ВИСНОВКИ 19](#_Toc73327632)

[ПЕРЕЛІК ПОСИЛАНЬ 20](#_Toc73327633)

[Додаток А Лістинг програмних рішень 21](#_Toc73327634)

# List of abbreviations

MTCNN – multi-task convolutional neural network.

YOLO – you only look once.

YOLACT – you only look at coefficients.

RCNN – residual convolutional neural network.

CNN – convolutional neural network.

GPU

IOU – intersection over union.

SOTA - state-of-the-art.

NN – neural network.

# Motivation

As both the augmented reality and image processing technologies become more and more advanced and ubiquitous we can finally start to think about solving tasks that require merging them together. This kind of combination will further help in making great tools for humans to become happier, hopefully. An example of such tool can be one that provides an environment for human to rest, by means of reducing the amount of distractions.

It is known that modern human has been around for millions of years. They relied on their pack the size of which was gradually increasing from 10 to 200 people over those years. It is natural for a human to be around the same circle of people through their whole life. Back then, and not too long ago, everyone in the group knew each other. If you meet someone - this is an opportunity to improve your relationship. Behave yourself well and try to be friendly, because in cooperation is the power of the "naked ape".

Should I really talk about how annoying the ads are? There are adblocks for internet browsers, we should have one for real life.

In big cities, people talk every day about this trait of the city - to tire you down. The unnatural number of people and the advertising posters anywhere you look play a large part in it. This leads you to put a lot of thought in behaving properly with every new person you see in addition to constant distraction from catchy advertisings of products you probably don’t need in your life.

Successful augmented reality glasses software will be able to hide advertisements and human faces, which will hopefully reduce stress on the inhabitants of the modern city.

# 1 Task description

The goal of this thesis project is to develop mathematical and software solutions for hiding faces and advertising banners on video in real time.

Functional requirements to the software:

1. The system should be able to read data from a webcam of a user.
2. The system should be able to detect faces.
3. The system should be able to detect advertising.
4. The detection results should be presented to the user graphically as a live video stream with hidden faces and ads.

The developed software running on modern PC with the Nvidia GeForce GTX 1650 GPU or better should meet the following non-functional requirements:

1. Face and advertising detection should be real time.
2. Face and advertising detection should run 24+ frames per second (counting separately).
3. The delay between the processed video and reality should not exceed 50 msec.

# 2 The analysis of existing methods

## 2.0.1 On non-machine learning based approaches

Past generation methods of object detection often rely on hand-crafted pixel maps. These maps are supposed to represent the distinctive features of the object we want to detect. Maps for the detection of objects like faces or advertising should get really sophisticated and numerous in order to do the job. But how sophisticated can we make them and can we come up with all the necessary ones for detection?

Today’s convolutional neural networks can find these distinctive features automatically. Not only this makes it is easier to program, but also gives greater accuracy. This makes non-ANN based methods look obsolete. Maybe, the only competition older methods can still give CNNs is in speed, but with even a modest GPU speed is neither a problem for CNNs.

2.0.2 Obtaining GPU results

In this chapter we will present fps results of performing inference of a model considered on various GPUs. These results are mostly found in the original articles covering the method or on Github repositories. Some of the results were inferred from others by multiplying them on GPU coefficients. These coefficients are calculated using fps data from a GPU benchmark website [www.gpucheck.com](http://www.gpucheck.com).

GPU coefficients are calculated as

where fx is the fps metric of the GPU, is the fps of the Nvidia GeForce GTX 1650.

We use the following coefficients:

1. Nvidia TESLA k80, c = .
2. Nvidia 1050Ti, c = 0.69.
3. Nvidia 1080Ti, c = .
4. Nvidia GeForce GTX 1650, c = 1.

2.1 Advertising detection

2.1.2 Mask R-CNN

Mask R-CNN is a modification of another object detection architecture called Faster R-CNN. The title of the article to the Faster R-CNN reads “Towards Real-Time Object Detection with Region Proposal Networks”. But this model does not actually run in real time, that is on regular computers.

Mask R-CNN performance:

|  |  |  |
| --- | --- | --- |
| GPU | fps | Image size |
| Nvidia TESLA k80 | 5 |  |
| Nvidia 1050 Ti | 7 |  |
| Nvidia GTX 1650 | ~9 |  |

2.1.3 YOLACT

YOLACT is a YOLO based neural network architecture for solving instance segmentation task. Instance segmentation is a computer vision task the goal of which is to create a pixel perfect mask of a recognized object in a given image.

YOLACT performance:

|  |  |  |  |
| --- | --- | --- | --- |
| GPU | fps | Image size | mAP |
| Nvidia TESLA k80 | ~14 | 550x550 | 26.8 |
| Nvidia 1080 Ti | 37 |  |  |
| Nvidia GTX 1650 | ~ |  |  |

1)  yolact\_resnet50\_54\_800000.pth  
-> I am giving 550x550 images

TESLA K80 : 13-15 fps

2) 26.8 mAP for B0, 29.6mAP for B3.  
GTX 1080Ti, base YOLACT 37 FPS  
 EfficientNet-B0 41FPS

2.1.4 YOLO

2.1.5 YOLOv3

YOLOv3 is a 3rd iteration of the state-of-the-art YOLO algorithm.

2.1.6 YOLOv5

2.2 Face detection

2.2.2 MTCNN

Модель достатню швидкість детекції, що дозволить нам використовувати

2.2.3 YOLO

2.3.4 YOLOFace

2.3.5 Viola-Jones framework

2.4.1 Comparison

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | mAP | Fps | Segmentation |  |  |  |  |
| YOLACT |  |  |  |  |  |  |  |
| Mask R-CNN |  |  |  |  |  |  |  |
| YOLO |  |  |  |  |  |  |  |
| YOLOv3 |  |  | No |  |  |  |  |
| YOLOv5 |  |  | No |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Model | Accuracy | Fps |  |  |  |  |  |
| YOLOFace |  |  |  |  |  |  |  |
| MTCNN |  | 22 |  |  |  |  |  |
| YOLO |  | 50 |  |  |  |  |  |
| Viola-Jones |  | 40 |  |  |  |  |  |

2.4.2 Conclusions

2.5 Датасет

Реклама

Обличчя

WIDER FACE is a benchmark dataset for evaluating the accuracy of face detection models. Its images are selected from the WIDER dataset. The benchmark quality of the dataset emerges from the fact that owners of the WIDER FACE do not release the ground truth label values for the test set.

There are 32 203 images with 393 703 labeled faces with high degree of variability in scale, pose, and occlusion.

# Description of the implemented methods

## Neural network

Нейронна мережа - це модель яку використовують для класифікації даних у системах із багатьма вимірами. Модель вчиться ставити у відповідність вхідним (у модель) масивам даних правильні вихідні масиви.

Помилка

Помилка нейронної мережі показує наскільки результат роботи моделі відрізняється від значення яке модель мала б повертати. Чим менша помилка моделі, тим правильніший результат моделі на цих вхідних даних.

Навчання

Навчання нейронних мереж відбувається шляхом корекції параметрів які входять у функцію що повертає результат. Ці параметри називаються вагами. Ваги змінюються шляхом мінімізації функції помилки моделі методом градієнтного спуску.

Тренувальні дані

Тренувальні дані (навчальні приклади) для моделей навчання із вчителем виглядають як «вхідні дані» - «правильний результат». Наприклад «2+2» - 4. При цьому і вхідні дані і результат проходять попередню обробку, щоб моделі було зручно із ними працювати. Наприклад «[2., 2.]» - «0.004». Набір із багатьох навчальних прикладів називають датасетом.

Тренування

Під час тренування, усі приклади з датасету проходять через нейронну мережу, для кожного прикладу обчислюється помилка моделі, корегуються ваги. Для отримання потрібної точності, датасет подають моделі не один раз. Кожне використання того ж датасету називають епохою.

Fully connected layer

Activation function

Loss function

## Convolutional neural network

Наведемо загальну характеристику згорткової нейронної мереіжі: в даній моделі навчання реалізується за рахунок автоматичного знаходження характерних ознак у навчальних прикладах. Процес навчання ітераційний, використовуються марковані навчальні приклади, очікуваний вихідний сигнал – назва класу, застосування – розпізнавання образів.

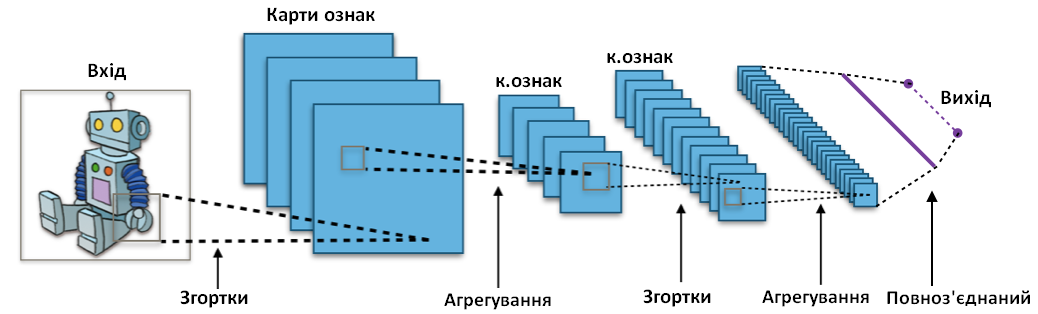


Рис. 1 – Архітектура згорткової нейронної мережі

Класична згорткова мережа може складатися із декількох різних шарів. Розглянемо будову такої моделі для задачі класифікації зображень.

## Convolutional layer

Шар згортки складається із так званих фільтрів, 2-во вимірних масивів. Процес згортки полягає у покроковому переміщенні фільтру по клітинкам вхідного масиву щоразу знаходячи суми добутків значень клітинок масиву фільтра і вхідного масива, що перекривають одна одну. Після повного проходження фільтра по вхідному масиву маємо карту ознак. Вона показує успішність застосування окремого фільтра на ділянках вхідного масиву. Після тренування, такі фільтри відповідають за розпізнавання патернів починаючи з ліній, точок у перших шарах згортки, закінчуючи патернами будь-якої складності у подальших шарах, для багатошарових згорткових моделей. Прикладом фільтра є такий двовимірний масив 3х3, що може відповідати за розпізнавання хрестиків

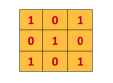


Рис 2. Filter

Згорткові шари мають ряд параметрів:

*Size* – length of the side of a square filter.

*Padding* – піксельна товщина рамки. Рамки додають до вхідних масивів для покращення розпізнавання патернів біля границь зображень.

*Stride* – the number of pixels filter shifts.

## Pooling layer

Призначення шару агрегування виділяти найбільш значущі частини двовимірного масиву таким чином зменшувати його розміри. Шар агрегування допоможе зменшити кількість параметрів моделі якщо вхідне зображення дуже велике.

Так само як фільтр у шарі згорток, маска шару агрегування проходиться по масиву. На кожному кроці знаходиться найбільше значення під маскою і передається у комірку вихідного масиву. Тобто, наприклад, з маскою розміру 2х2 і кроком 2, після агрегування вхідний масив зменшиться у 2 рази як в ширину, так і в довжину.

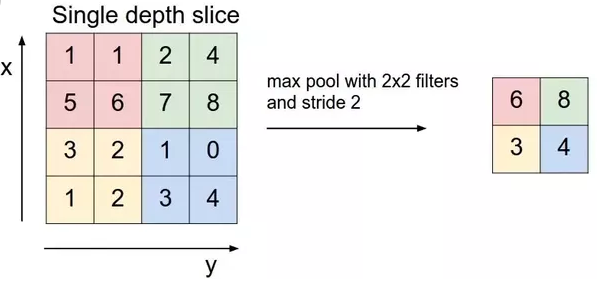


Рис 3. Pooling layer in action

## Повнозв’язний шар

Повнозв’язний шар ставиться після шарів згортки. Так він приймає на вхід знайдені згортковими шарами ознаки і виконує класифікацію.

## Dropout layer

Шар виключення кожну ітерацію випадково відключає частину вхідних у нього зв’язків (випадково n%, наприкад 50%). Таким чином нейронна мережа вимушена підбирати такі параметри, щоб обходитися без частини цих зв’язків.

Використання шару виключення набагато зменшить ризик перенавчання, якщо поставити його після повнозв’зного шару, адже такі шари, маючи велику кількість параметрів, схильні запам’ятовувати навчальні приклади, що призводить до неможливості узагальнення назовні навчальних даних.

## 3 YOLO algorithm

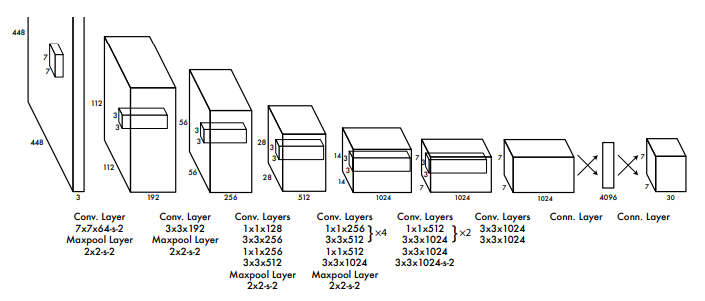


Figure. YOLO architecture.

Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers.

Detection often requires fine-grained visual information so we increase the input resolution of the network from 224 × 224 to 448 × 448.

For the final layer we use linear activation function and for all other layers the following leaky rectified linear activation is used:

Loss function:

where denotes if object appears in cell and denotes that the th bounding box predictor in cell is “responsible” for that prediction.

Оцінка якості локалізації об'єкта

Intersection Over Union (IOU)

Non-max suppression

Для результуючого вектора [pc, bx, by, bw, bh].

Алгоритм:

Відкинути усі детекції деймовірність pc < 0.6

Доки ще залишилися детекції

Вибрати детекцію із найбільшим рс. Повернути її як результат.

Відкинути усі детекції що мають IOU > 0.5 із детекцією з попереднього кроку.

Ящик якір

Результуючий вектор [pc, bx, by, bw, bh, c1, …, cn, ppc, ax, ay, aw, ah, c1, …, cn].

Figure. Learning rate graph

## 3 MTCNN algorithm

Отримавши зображення, ми спочатку змінюємо його розмір до різних масштабів створюючи піраміду зображень. Піраміда є вкладом наступного триступеневого каскадного каркасу:

Етап 1: Ми використовуємо «повністю згорткову мережу», що називаємо Пропозиційна мережа (P-Net), для отримання баундинг боксів та регресійних векторів кандидатів. Потім ми використовуємо регресійні вектори баундинг боксів для калібрування кандидатів. Після цього ми виконуємо не максимальне придушення (non-max-suppression) для злиття кандидатів, що перекриваються іншими.

Етап 2: усіх кандидатів подають до іншої CNN, яка називається Мережа Уточнення (R-Net). Вона відкидає велику кількість помилкових кандидатів, виконує калібрування з регресією баундинг боксу, та робить злиття кандидатів на не максимальне придушення.

Етап 3: Цей етап схожий на другий, але на цьому етапі ми прагнемо описати обличчя більш детально, тому на виході мережі - п’ять нейронів - місць на обличчі.

При тренуванні використовуємо такі функції помилок

1. Класифікація облич



1. Регресія баундинг боксів



1. Локалізація ключових точок обличчя



1. Загальна функція помилки для багатоджерельного тренування

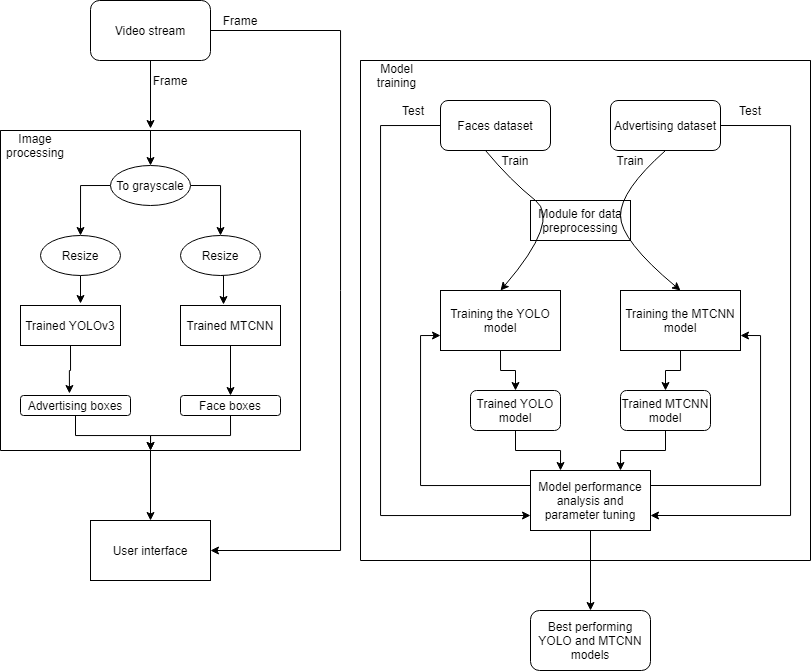


де – N - кількість навчальних зразків.  відповідає за важливість завдання. Ми використовуємо   у P-Net та R-Net, тоді як   в O-Net для точнішої локалізації орієнтирів обличчя.  - показник типу вибірки. В цьому методі природно застосовувати стохастичний градієнтний спуск до навчати CNN

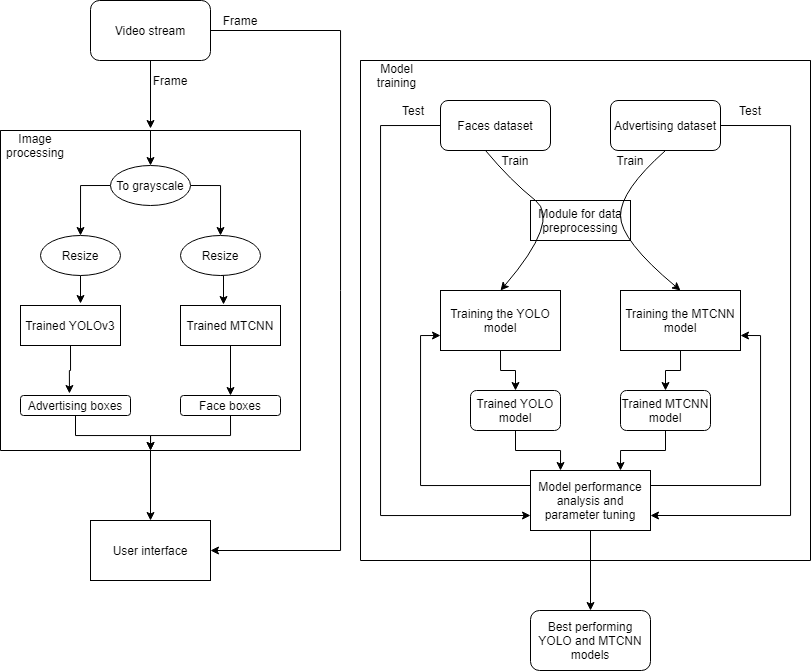
# Програмна реалізація

## Software structure model

The software was written using Python language.

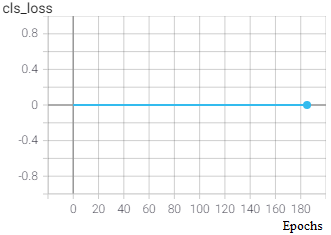
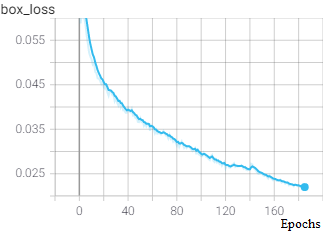


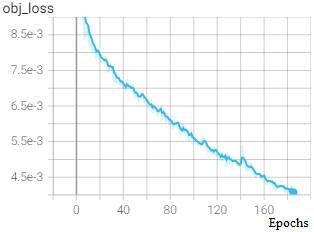
Model evaluation diagram



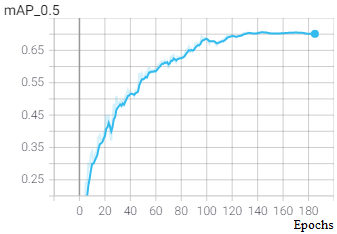
Model training diagram

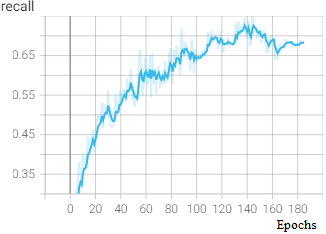
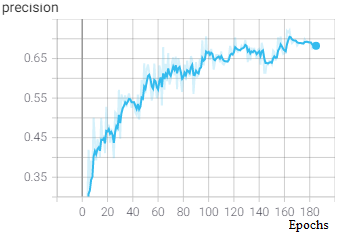
Training the YOLO model





Validation results





4.2 Experimental results

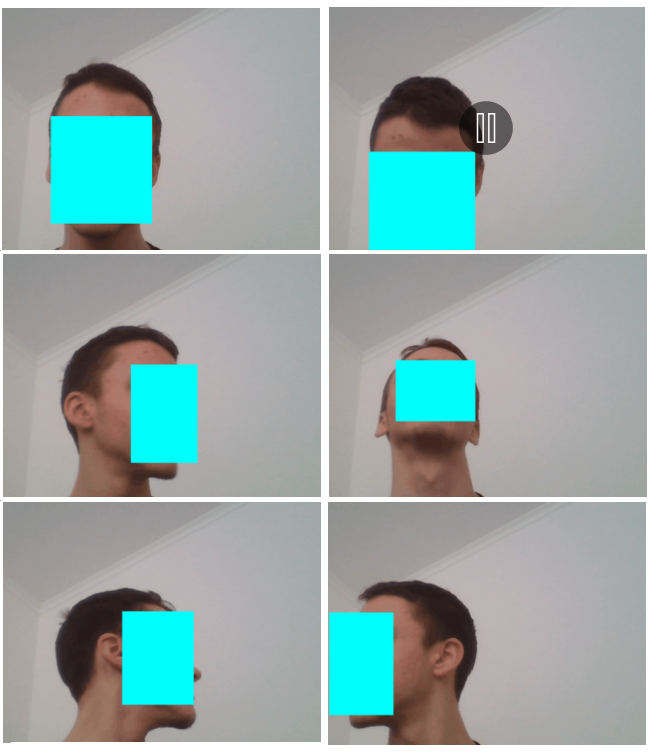


Figure. Frames with marked detections of faces from a webcam in real time.

While running it on our CPU we got detection speed of 2.2 fps.

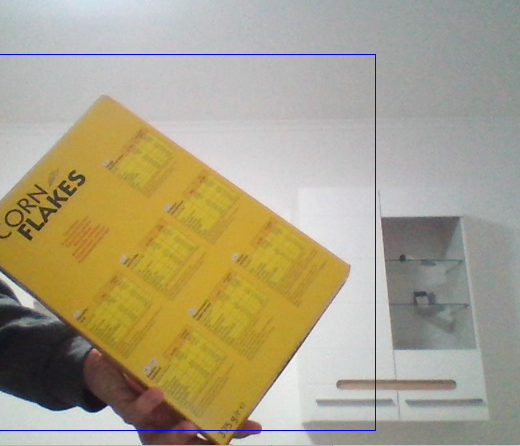


Figure. Frames with marked detections of advertising from a webcam in real time.

Figure. Frames with detections of advertising and faces simultaneously from a webcam.

Ways to improve the project

Tracking algorithms

# ВИСНОВКИ

We have analysed a number of object detection CNN based methods. As a result, we have chosen MTCNN method for face detection, and YOLOv3 for the detection of advertising. Also, the software architecture of the project has been introduced. It covers the training of ANN’s as well as a way to use them and evaluate their performance.

Під час дипломної практики було натреновано та протестовано алгоритм детекції обличя. Також було протестовано графічний інтерфейс для захоплення відео з вебкамери та режим детекції обличчя у реальному часі.

На нашому слабкому ПК отримали швидкість детекції 2.2 кадри/с. Цей результат є достатньо перспективним на даному етапі.

Помістивши систему на ПК з зазначеною у вимогах відеокартою отримаємо в рази швидшу детекцію облич. Це ми згодом і зробимо.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* doi:10.1109/cvpr.2016.91, 2016.
2. Redmon, J., & Farhadi, A. YOLO9000: Better, Faster, Stronger. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2017.690 , 2016.
3. YOLOv3: An Incremental Improvement. arXiv:1804.02767, 2018.
4. Chen, W., Huang, H., Peng, S., Zhou, C., & Zhang, C. YOLO-face: a real-time face detector. The Visual Computer. *doi:10.1007/s00371-020-01831-7*, 2020.
5. Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. In [IEEE Signal Processing Letters](https://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber=97)*,* 23(10), 1499–1503 *doi:* [*10.1109/LSP.2016.2603342*](https://doi.org/10.1109/LSP.2016.2603342)*,* 2016.
6. Yang, S., Luo, P., Loy, C. C., & Tang, X. WIDER FACE: A Face Detection Benchmark. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).  WIDER FACE: A Face Detection Benchmark. doi:10.1109/cvpr.2016.596, 2016.
7. YOLACT: Real-time Instance Segmentation. *arXiv:1904.02689*, 2019.
8. Mask R-CNN. *arXiv:1703.06870*, 2018.
9. Yinda Xu, Zeyu Wang, Zuoxin Li, Ye Yuan, Gang Yu. SiamFC++: Towards Robust and Accurate Visual Tracking with Target Estimation Guidelines. Conference on Artificial Intelligence (AAAI-20), 10.1609/aaai.v34i07.6944, 2020
10. Erik-Linder Noren, PyTorch YOLOv3, (2018), Github repository.

https://github.com/eriklindernoren/PyTorch-YOLOv3

1. Tim Esler, Face Recognition using PyTorch, (2020), Github repository.

<https://github.com/timesler/facenet-pytorch>

1. Датасет зображень реклами

<http://www.cs.ucf.edu/~aroshan/index_files/Dataset_PitOrlManh/zipped%20images/part1.zip>

Анотації:

WannaFly, Mask\_AD, Github repository.

https://github.com/WannaFIy/mask\_AD/releases/download/AD/regions.csv

# Додаток А Лістинг програмних рішень

qtapp.py Графічний інтерфейс для обробки зображення з камери

**import** sys

**from** PyQt5 **import** QtGui, QtCore

**from** PyQt5.QtWidgets **import** QWidget, QMainWindow, QApplication, QPushButton, QLabel, QVBoxLayout, QFileDialog

**import** cv2

**import** threading

**from** PIL **import** Image

**import** time

**from** mtcnn\_model **import** extract\_faces, draw\_rectangles

def **to\_bytes**(non\_bytes\_img, type**=**'jpeg'):

**from** io **import** BytesIO

    buf **=** BytesIO()

    non\_bytes\_img.save(non\_bytes\_img, type)

    buf.seek(0)

    image\_bytes **=** buf.read()

    buf.close()

**return** image\_bytes

class QtCapture(QWidget):

    def \_\_init\_\_(self, **\***args):

        super(QWidget, **self**).\_\_init\_\_()

**self**.fps **=** 24

**self**.cap **=** cv2.VideoCapture(**\***args)

**self**.video\_frame **=** QLabel()

        lay **=** QVBoxLayout()

        lay.setContentsMargins(0,0,0,0)

        lay.addWidget(**self**.video\_frame)

**self**.setLayout(lay)

        # ------ Modification ------ #

**self**.isCapturing **=** False

**self**.ith\_frame **=** 1

        # ------ Modification ------ #

    def **setFPS**(self, fps):

**self**.fps **=** fps

    def **nextFrameSlot**(self):

        ret, frame **=** **self**.cap.read()

        # ------ Modification ------ #

        # Save images if isCapturing

**if** **self**.isCapturing:

            cv2.imwrite('img\_%05d.jpg'**%self**.ith\_frame, frame)

**self**.ith\_frame **+=** 1

        # ------ Modification ------ #

        # My webcam yields frames in BGR format

        frame **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        # Face detection

        frame **=** draw\_rectangles(frame, extract\_faces(frame), enlarge**=**True)

        # t1 = threading.Thread(target=QtGui.QImage, args=(frame, frame.shape[1], frame.shape[0], QtGui.QImage.Format\_RGB888))

        # t1.start()

        img **=** QtGui.QImage(frame, frame.shape[1], frame.shape[0], QtGui.QImage.Format\_RGB888)

        pix **=** QtGui.QPixmap.fromImage(img)

**self**.video\_frame.setPixmap(pix)

**self**.count **+=** 1

        print((time.time()**-self**.t1)**/self**.count)

    # def streamThread(self, frame, frame.shape[1], frame.shape[0], QtGui.QImage.Format\_RGB888):

    #     while

    def **start**(self):

**self**.t1 **=** time.time()

**self**.count **=** 0

**self**.timer **=** QtCore.QTimer()

**self**.timer.timeout.connect(**self**.nextFrameSlot)

**self**.timer.start(1000.**/self**.fps)

    def **stop**(self):

**self**.timer.stop()

    # ------ Modification ------ #

    def **capture**(self):

**if** **not** **self**.isCapturing:

**self**.isCapturing **=** True

**else**:

**self**.isCapturing **=** False

    # ------ Modification ------ #

    def **deleteLater**(self):

**self**.cap.release()

        super(QWidget, **self**).deleteLater()

class ControlWindow(QWidget):

    def \_\_init\_\_(self):

        QWidget.\_\_init\_\_(**self**)

**self**.capture **=** None

**self**.start\_button **=** QPushButton('Start')

**self**.start\_button.clicked.connect(**self**.startCapture)

**self**.picture\_button **=** QPushButton('Take picture')

**self**.picture\_button.clicked.connect(**self**.picture)

**self**.quit\_button **=** QPushButton('End')

**self**.quit\_button.clicked.connect(**self**.endCapture)

**self**.end\_button **=** QPushButton('Stop')

**self**.inference\_file\_button **=** QPushButton('Inference file')

**self**.inference\_file\_button.clicked.connect(**self**.inf\_file)

        # self.dialog = QFileDialog()

**self**.le **=** QLabel('')

        # ------ Modification ------ #

**self**.capture\_button **=** QPushButton('Capture')

**self**.capture\_button.clicked.connect(**self**.saveCapture)

        # ------ Modification ------ #

        vbox **=** QVBoxLayout(**self**)

        vbox.addWidget(**self**.start\_button)

        vbox.addWidget(**self**.picture\_button)

        vbox.addWidget(**self**.end\_button)

        vbox.addWidget(**self**.inference\_file\_button)

        vbox.addWidget(**self**.le)

        # vbox.addWidget(self.dialog)

        vbox.addWidget(**self**.quit\_button)

        # ------ Modification ------ #

        vbox.addWidget(**self**.capture\_button)

        # ------ Modification ------ #

**self**.setLayout(vbox)

**self**.setWindowTitle('Control Panel')

**self**.setGeometry(100,100,200,200)

**self**.show()

    def **startCapture**(self):

**if** **not** **self**.capture:

**self**.capture **=** QtCapture(0)

**self**.end\_button.clicked.connect(**self**.capture.stop)

            # self.capture.setFPS(1)

**self**.capture.setParent(**self**)

**self**.capture.setWindowFlags(QtCore.Qt.Tool)

**self**.capture.start()

**self**.capture.show()

    def **endCapture**(self):

**self**.capture.deleteLater()

**self**.capture **=** None

    # ------ Modification ------ #

    def **saveCapture**(self):

**if** **self**.capture:

**self**.capture.capture()

    # ------ Modification ------ #

    def **inf\_file**(self):

        filename **=** QFileDialog.getOpenFileName()[0]

**self**.le.setPixmap(QtGui.QPixmap(filename))

        im  **=** cv2.imread(filename)

        im **=** cv2.cvtColor(im, cv2.COLOR\_BGR2RGB)

        faces **=** extract\_faces(im)

        im\_mod **=** draw\_rectangles(im, faces, enlarge**=**True)

        cv2.imwrite('im\_mod.jpg', im\_mod)

        # except:

        #     print('wrong file')

    def **picture**(self):

**if** **not** **self**.capture:

**self**.capture **=** QtCapture(0)

**self**.end\_button.clicked.connect(**self**.capture.stop)

        ret, frame **=** **self**.capture.cap.read()

        cv2.imwrite('img.jpg', frame)

        # My webcam yields frames in BGR format

        # frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        # Face detection

        frame **=** draw\_rectangles(frame, extract\_faces(frame), enlarge**=**True)

        cv2.imwrite('img\_mod.jpg', frame)

**if** \_\_name\_\_ **==** '\_\_main\_\_':

**import** sys

    app **=** QApplication(sys.argv)

    window **=** ControlWindow()

    sys.exit(app.exec\_())

mtcnn\_model.py Модуль, що застосовує натренований детектор облич.

**import** cv2

**from** PIL **import** Image, ImageDraw

**import** csv

**import** numpy **as** np

**import** matplotlib.pyplot **as** plt

**import** matplotlib.patches **as** patches

**import** os

**import** time

**from** mtcnn **import** MTCNN

**from** skimage.transform **import** resize

"""

Important functions:

extract\_faces(image(np.array))

drawRectangles(image(np.array), rectangles(4), color(3))

"""

detector **=** MTCNN()

def **prep\_mtcnn**(pic)->np.array:

    t1 **=** time.time()

    image **=** Image.fromarray(pic.astype(np.uint8))

    image **=** image.resize((224, 224))

    face\_array **=** np.asarray(image)

    print('resize', time.time()**-**t1)

**return** face\_array

def **mapKeypointsToOriginal**(facesRatioed: list, origShape)->list:

    """

    changes mapping of keypoints of faces. not boxes!

    """

    width\_ratio **=** 224.**/**origShape[0]

    height\_ratio**=** 224.**/**origShape[1]

**for** i **in** range(len(facesRatioed)):

        facesRatioed[i]['keypoints'] **=** {keypoint:(int(coords[0]**/**height\_ratio), int(coords[1]**/**width\_ratio))

**for** keypoint, coords **in** facesRatioed[i]['keypoints'].items()}

**return** facesRatioed

def **keypointsToBoundingBox**(faces)->list:

    bbs **=** []

**for** face **in** faces:

        face\_points **=** [i[1] **for** i **in** face['keypoints'].items()]

        # face\_points.append((face['box'][0],face['box'][1]))

        # face\_points.append((face['box'][2],face['box'][3]))

        # squeezed face boxes

        x1, y1 **=** min(face\_points, key**=**lambda x:x[0])[0], min(face\_points, key**=**lambda x:x[1])[1]

        x2, y2 **=** max(face\_points, key**=**lambda x:x[0])[0], max(face\_points, key**=**lambda x:x[1])[1]

        bbs.append([x1,y1, x2,y2])

**return** bbs

def **extract\_faces**(pic, detector**=**detector)->list:

    """

    extracts faces from image and returns a list of bounding boxes [x1,y1,x2,y2]

    """

    picResized **=** prep\_mtcnn(pic)

    facesRatioed **=** detector.detect\_faces(picResized) # detect faces in the image

    faces **=** mapKeypointsToOriginal(facesRatioed, pic.shape) # map to original sized image

    boundingBoxes **=** keypointsToBoundingBox(faces)

**return** boundingBoxes

def **extract\_faces\_multiproc**(frame\_queue)->list:

    """

    extracts faces from image and returns a list of bounding boxes [x1,y1,x2,y2]

    """

    picResized **=** prep\_mtcnn(pic)

    facesRatioed **=** detector.detect\_faces(picResized) # detect faces in the image

    t1 **=** time.time()

    faces **=** mapKeypointsToOriginal(facesRatioed, pic.shape) # map to original sized image

    boundingBoxes **=** keypointsToBoundingBox(faces)

    print('key',time.time()**-**t1)

**return** boundingBoxes

def **draw\_rectangles**(imPointer, rectangles, color**=**[0,255,255], enlarge**=**False):

    """

    image - numpy image - RGB format

    rectangles - (x1, y1, x2, y2) rectangle coords

            or a list of (x1, y1, x2, y2)

    returns changed image

    """

    image **=** imPointer.copy()

**if** enlarge:

**for** i **in** range(len(rectangles)):

            width\_add **=** int((rectangles[i][2] **-** rectangles[i][0])**/**2)

            length\_add **=** int((rectangles[i][3] **-** rectangles[i][1])**/**2)

            rectangles[i][0], rectangles[i][1] **=** rectangles[i][0] **-** width\_add, rectangles[i][1] **-** length\_add

            rectangles[i][2], rectangles[i][3] **=** rectangles[i][2] **+** width\_add, rectangles[i][3] **+** length\_add

**for** rectangle **in** rectangles:

**for** i **in** range(rectangle[1], min(rectangle[3], image.shape[0])):

**for** j **in** range(rectangle[0], min(rectangle[2], image.shape[1])):

                image[i][j] **=** color

**return** image

def **mark\_faces\_mtcnn**(im):

    """

    im - np.array

    returns im with drawn rectangles (np.array)

    """

    # old draw

    # if type(im)!="<class 'numpy.ndarray'>":

    # imarr = Image.fromarray(im)

    # print(type(imarr))

    # draw = ImageDraw.Draw(imarr)

    original\_size **=** im.shape

    faces **=** extract\_faces(im)

    rectangles **=** []

**for** face **in** faces:

        face\_points **=** [i[1] **for** i **in** face['keypoints'].items()]

        # face\_points.append((face['box'][0],face['box'][1]))

        # face\_points.append((face['box'][2],face['box'][3]))

        # squeezed face boxes

        x1, y1 **=** min(face\_points, key**=**lambda x:x[0])[0], min(face\_points, key**=**lambda x:x[1])[1]

        x2, y2 **=** max(face\_points, key**=**lambda x:x[0])[0], max(face\_points, key**=**lambda x:x[1])[1]

        x1, y1 **=** mapFacesToOriginal((x1,y1), original\_size)

        x2, y2 **=** mapFacesToOriginal((x2,y2), original\_size)

        rectangles.append((x1,y1,x2,y2))

        # draw.rectangle((x1, y1, x2, y2), fill=(0, 192, 192), outline=(255, 255, 255))

    # im\_changed = im

    im\_changed **=** draw\_rectangles(im, rectangles)

    # plt.imshow(im\_changed)

    # plt.show()

**return** im\_changed

def **plot\_res\_mtcnn**(pic, faces):

    fig, ax **=** plt.subplots()

    ax.imshow(pic)

**for** face **in** faces:

        length **=** face['box'][3] **-** face['box'][1]

        width **=** face['box'][2] **-** face['box'][0]

        # key points

**for** key, val **in** face['keypoints'].items():

            ax.scatter(val[0],  val[1])

        face\_points **=** [i[1] **for** i **in** face['keypoints'].items()]

        face\_points.append((face['box'][0],face['box'][1]))

        face\_points.append((face['box'][2],face['box'][3]))

        # face box

        x1, y1 **=** min(face\_points, key**=**lambda x:x[0])[0], min(face\_points, key**=**lambda x:x[1])[1]

        x2, y2 **=** max(face\_points, key**=**lambda x:x[0])[0], max(face\_points, key**=**lambda x:x[1])[1]

        ax.add\_patch(patches.Rectangle((x1, y1), x2**-**x1, y2**-**y1, linewidth**=**1, edgecolor**=**'r', facecolor**=**'none'))

        # ax.scatter(face['box'][0], face['box'][1])

        # ax.scatter(face['box'][2], face['box'][3])