ДОДАТОК А.

Програмний код

А.1 OpenGLHelper.h

```
OpenGLHelper.h
#ifndef OpenGLHelper_h
#define OpenGLHelper_h
#ifdef ASSERT_OPENGL
    #define WT_GL_ASSERT( __gl_code__ ) { \
        __gl_code__; \
        GLenum __wt_gl_error_code__ = glGetError(); \
        if ( __wt_gl_error_code__ != GL_NO_ERROR ) { \
            printf("OpenGL error '%x' occured at line %d inside
function %s\n",
__wt_gl_error_code__, __LINE__, __PRETTY_FUNCTION__); \
        } \
    }
    #define WT_GL_ASSERT_AND_RETURN( __assign_to__, __gl_code__
) { \
        __assign_to__ = __gl_code__; \
        GLenum __wt_gl_error_code__ = glGetError(); \
        if ( __wt_gl_error_code__ != GL_NO_ERROR ) { \
            printf("OpenGL error '%x' occured at line %d inside
function %s\n",
__wt_gl_error_code__, __LINE__, __PRETTY_FUNCTION__); \
        } \
    }
#else
    #define WT_GL_ASSERT( __gl_code__ ) __gl_code__
    #define WT_GL_ASSERT_AND_RETURN( __assign_to__, __gl_code__
) __assign_to__ =
__gl_code__
#endif
#endif
```

А.2 Клас FaceDetactor

```
typedef void(^FlushBlock)();
@interface SSFaceDetector : NSObject
- (instancetype)initWithCameraView:(UIView *)view
                             scale:(CGFloat)scale
                        flushBlock:(FlushBlock)block;
@property (nonatomic, readonly) BOOL isCapturing;
@property (nonatomic, readonly) float *modelView;
@property (nonatomic, readonly) float *projection;
- (void)startCapture;
- (void)stopCapture;
```

```
- (void)calibrate;
@end
#ifdef __cplusplus
#import <opencv2/opencv.hpp>
#import <opencv2/highgui/cap_ios.h>
#endif
#import <math.h>
#import <libkern/OSAtomic.h>
#import "SSFaceDetector.h"
#import "SSModelBuilder.h"
using namespace cv;
static int      const kMaxPointsCount = 100;
static CFIndex  const kCascadeNameLen = 2048.l;
static cv::Size const kSubPixWinSize(10, 10);//,
kWinSize(21,21);
@interface SSFaceDetector () <CvVideoCameraDelegate> {

    volatile OSSpinLock _flushLock;

    CascadeClassifier _faceDetector;
    CascadeClassifier _rightEyeDetector;
    CascadeClassifier _leftEyeDetector;
    CascadeClassifier _mouthDetector;
    CascadeClassifier _noseDetector;

    /*
     We use two sets of points
     in order to swap pointers.
     */
    Mat _gray, _prevGray;
    vector<uchar> _status;
    vector<Point2f> _points, _prevPoints;
    vector<Point3f> _modelProjection;

    vector<cv::Rect> _faceRects; //do we need property?

    cv::Mat _modelView;
    float _projection[16]; //TODO: change to cv::Mat
}
@property (nonatomic, copy) FlushBlock flushBlock;
@property (nonatomic, strong) CvVideoCamera* videoCamera;
@property (nonatomic, assign) CGFloat scale;
@end
@implementation SSFaceDetector
- (instancetype)initWithCameraView:(UIView *)view
scale:(CGFloat)scale flushBlock:
(FlushBlock)block {
    self = [super init];
    if (self) {
        memset(&_projection, 0, 16 * sizeof(float));
        _flushLock = OS_SPINLOCK_INIT;
        _modelView = cv::Mat::zeros(4, 4, CV_64F);
```

```
            self.scale = scale;
            self.flushBlock = block;
            [self setupCameraWithView:view];
            [self setupClassifiers];
        }

        return self;
    }
    - (void)setupCameraWithView:(UIView *)view {
        self.videoCamera = [[CvVideoCamera alloc]
initWithParentView:view];
        self.videoCamera.defaultAVCaptureDevicePosition   =
AVCaptureDevicePositionFront;
        self.videoCamera.defaultAVCaptureSessionPreset    =
AVCaptureSessionPreset640x480;
        self.videoCamera.defaultAVCaptureVideoOrientation =
AVCaptureVideoOrientationPortrait;
        self.videoCamera.defaultFPS = 60;
        self.videoCamera.grayscaleMode = NO;
        self.videoCamera.delegate = self;
    }
    - (void)setupClassifiers {
        [self setupFaceRecognition];
    //    [self setupEyeRecognition];
        //[self setupMouthRecognition];
        //[self setupNoseRecognition];
    }
    - (void)setupFaceRecognition {
        [self setupClassifier:_faceDetector
             withResourceName:@"haarcascade_frontalface_alt2"];
    }
    - (void)setupEyeRecognition {
        [self setupClassifier:_leftEyeDetector
             withResourceName:@"haarcascade_lefteye_2splits"];
        [self setupClassifier:_rightEyeDetector
             withResourceName:@"haarcascade_righteye_2splits"];
    }
    - (void)setupMouthRecognition {
        [self setupClassifier:_mouthDetector
             withResourceName:@"haarcascade_mcs_mouth"];
    }
    - (void)setupNoseRecognition {
        [self setupClassifier:_noseDetector
             withResourceName:@"haarcascade_mcs_nose"];
    }
    - (void)setupClassifier:(CascadeClassifier &)classifier
           withResourceName:(NSString *)name {
        char *cascadeName = (char *)malloc(kCascadeNameLen);
        NSString *eyesCascadePath = [[NSBundle mainBundle]
pathForResource:name

 ofType:@"xml"];
```

```
        CFStringGetFileSystemRepresentation((CFStringRef)eyesCascad
ePath,
                                            cascadeName,
                                            kCascadeNameLen);
        classifier.load(cascadeName);
        free(cascadeName);
    }
    #pragma mark - Public
    - (void)startCapture {
        [self.videoCamera start];
    }
    - (void)stopCapture; {
        [self.videoCamera stop];
    }
    - (BOOL)isCapturing {
        return self.videoCamera.running;
    }
    - (float *)modelView {
        return &_modelView.at<float>(0, 0);
    }
    - (float *)projection {
        return &_projection[0];
    }
    - (void)calibrate {
        OSSpinLockLock(&_flushLock);
        [self freeResources];
        OSSpinLockUnlock(&_flushLock);
    }
    #pragma mark - CvVideoCameraDelegate
    - (void)processImage:(cv::Mat &)image {

        vector<cv::Rect> rects = [self detectFacesOn:image
withScale:self.scale multiple:NO];
        if (rects.empty()) {
            [self freeResources];
        }

        int i = 0;
        for(vector<cv::Rect>::const_iterator rect = rects.begin();
rect != rects.end(); rect++,
    i++ )
        {
            NSAssert(i < 1, @"We handle only one face.");

            cv::Rect faceRect = *rect;
            faceRect.x *= self.scale;
            faceRect.y *= self.scale;
            faceRect.width *= self.scale;
            faceRect.height *= self.scale;

            float resize = 0.8;
            float dx = faceRect.height * (1 - resize) / 2;
            float dy = faceRect.width * (1 - resize) / 2;
```

```
            faceRect.x += dx;
            faceRect.y -= dy;
            faceRect.width *= resize;
    //        faceRect.height *= resize;

    #ifdef DEBUG
            rectangle(image, faceRect, Scalar(0,0,255), 1, 8, 0);
    #endif

            cvtColor(image, _gray, COLOR_BGR2GRAY);
            _points.clear();
            _status.clear();
            OSSpinLockLock(&_flushLock);
            [self performOpticalFlowCalculationWithGray:_gray
                                               prevGray:_prevGray
                                                 points:_points
                                             prevPoints:_prevPoints
                                                 status:_status
                                                 inRect:faceRect
                                                  scale:self.scale]
;

            if (!_status.empty() && ![self
isTrackingAvailableWithStatus:_status]) {
                _status.clear();
                _prevPoints.clear();
                _modelProjection.clear();
            }
            if (_status.empty() || _points.empty()) {
                OSSpinLockUnlock(&_flushLock);
                return;
            }

    #ifdef DEBUG
            for (size_t i = 0; i < _points.size(); i++) {
                if (_status[i]) {
                    line(image, _prevPoints[i], _points[i],
Scalar(0,0,255), 4);
                } else {
                    line(image, _prevPoints[i], _points[i],
Scalar(255,0,0), 2);
                }
            }
    #endif

            [self updateProjectionIfNeededWithRect:faceRect];

            vector<Point3f> activeModelPoints;
            vector<Point2f> activeImagePoints;

            int i = 0;
            for (vector<uchar>::iterator active = _status.begin();
```

```
                 active != _status.end() || i <
_modelProjection.size() || i < _points.size();
                 active++, i++) {

                if (*active) {
                    activeModelPoints.push_back(_modelProjection[i]
);
                    activeImagePoints.push_back(_points[i]);
                }
            }

            OSSpinLockUnlock(&_flushLock);

            if (activeImagePoints.size() !=
activeModelPoints.size() || activeModelPoints.size()
    < 4)
                return;

            [self calculateMatrixWithModelPoints:activeModelPoints
                                     imagePoints:activeImagePoints
                                         inImage:image];
        }

        self.flushBlock();
    }
    #pragma mark - CV
    - (void)findChessCorners:(cv::Mat &)image {
        std::vector<cv::Point2f> chessPoints;
        findChessboardCorners(image, cv::Size(7, 9), chessPoints);

        if (chessPoints.empty()) return;
        std::vector<cv::Point2f> imagePoints(4);
        imagePoints[0] = chessPoints[0];
        imagePoints[1] = chessPoints[6];
        imagePoints[2] = chessPoints[chessPoints.size() - 7];
        imagePoints[3] = chessPoints[chessPoints.size() - 1];
    //    std::vector<cv::Point2f> imagePoints =
Generate2DPoints(image);
        std::vector<cv::Point3f> objectPoints = Generate3DPoints();

        [self calculateMatrixWithModelPoints:objectPoints
                                 imagePoints:imagePoints
                                     inImage:image];
    }
    - (vector<cv::Rect>)detectFacesOn:(Mat &)image
                           withScale:(double)scale
                            multiple:(BOOL)multiple {
        Mat gray, smallImg(cvRound(image.rows/scale),
cvRound(image.cols/scale), CV_8UC1);
        cvtColor(image, gray, COLOR_BGR2GRAY);
        resize(gray, smallImg, smallImg.size(), 0, 0,
INTER_LINEAR);
        equalizeHist(smallImg, smallImg);
```

```
        double scalingFactor = 1.1;
        int minRects = 2;
        cv::Size minSize(30, 30);

        _faceDetector.detectMultiScale(smallImg, _faceRects,
                                       scalingFactor, minRects,
(multiple ? 0 :
    CASCADE_FIND_BIGGEST_OBJECT),
                                       minSize);

        return _faceRects;
    }
    - (void)performOpticalFlowCalculationWithGray:(cv::Mat &)gray
                                         prevGray:(cv::Mat
&)prevGray
                                           points:(vector<Point2f>
&)points
                                       prevPoints:(vector<Point2f>
&)prevPoints
                                           status:(vector<uchar>
&)status
                                           inRect:(cv::Rect)rect
                                            scale:(float)scale {

        TermCriteria termcrit(CV_TERMCRIT_ITER | CV_TERMCRIT_EPS,
20, 0.3);

        if (prevPoints.empty()) {
            //Feature detection is performed here...
            goodFeaturesToTrack(gray, points, kMaxPointsCount,
                                0.01, 45, Mat(), 3, 0, 0.04);
            cornerSubPix(gray, points, kSubPixWinSize,
                         cv::Size(-1, -1), termcrit);
            points = [self points:points byFiltringInRect:rect];
        } else {
            Mat err;
            calcOpticalFlowPyrLK(prevGray, gray,
                                 prevPoints, points,
                                 status, err);
            cornerSubPix(gray, points, kSubPixWinSize,
                         cv::Size(-1, -1), termcrit);
        }

        std::swap(points, prevPoints);
        cv::swap(prevGray, gray);
    }
    #pragma mark - Helper
    - (BOOL)isTrackingAvailableWithStatus:(vector<uchar>)status {
        int features = 0;
        for (vector<uchar>::iterator iterator = status.begin();
iterator != status.end();
    iterator++) {
```

```
            features += *iterator;
        }
        return features > 2;
    }
    - (void)updateProjectionIfNeededWithRect:(cv::Rect)faceRect {
        if (_modelProjection.empty()) {
            _modelProjection = [self
updateModelProjectionWithRect:faceRect
                                                                points:
_points];
        }
    }
    -
(vector<cv::Point3f>)updateModelProjectionWithRect:(cv::Rect)faceRec
t
                                                    points:(vector<cv
::Point2f>)points {
        std::vector<Point2f> origin (points.size());
        std::transform(points.begin(), points.end(),
                       origin.begin(), [&faceRect](Point2f point) {
                           Point2f topLeft = faceRect.tl();
                           return point - topLeft;
                       });
        return [SSModelBuilder projectPoints:origin
                                    withSize:faceRect.size()
                                  targetSize:cv::Size(1000, 1000)];
    }
    -
(void)calculateMatrixWithModelPoints:(vector<Point3f>)modelPoints
                               imagePoints:(vector<Point2f>)imagePo
ints
                                   inImage:(cv::Mat &)image {

        float focal_length = MAX(image.cols, image.rows); //
Approximate focal length.
        Point2f center = cv::Point2f(image.cols / 2, image.rows /
2);
        cv::Mat cameraMatrix = (cv::Mat_<double>(3,3) <<
focal_length, 0, center.x,
                                0, focal_length, center.y,
                                0, 0, 1);
        cv::Mat
distCoeffs(cv::Mat::zeros(4,1,cv::DataType<double>::type));
        cv::Mat
rvec(cv::Mat::zeros(3,1,cv::DataType<double>::type));
        cv::Mat
tvec(cv::Mat::zeros(3,1,cv::DataType<double>::type));

        cv::solvePnP(modelPoints, imagePoints, cameraMatrix,
distCoeffs, rvec, tvec, false,
    CV_ITERATIVE);

    #ifdef DEBUG
```

```
        vector<Point2f> projection;
        projectPoints(modelPoints, rvec, tvec, cameraMatrix,
distCoeffs, projection);
        for (int i = 0; i < projection.size(); i++) {
            cv::line(image, projection[i], imagePoints[i],
cv::Scalar(255,0,0), 2);
        }
    #endif

        [self calcModelviewMat:_modelView
                    fromRotVec:rvec
                      transVec:tvec];

        [self calcProjectionMat:self.projection
                     fromCameraMat:cameraMatrix
                    withScreenSize:CGSizeMake(image.cols, image.rows)
                             znear:50
                              zfar:5000];
    }
    - (std::vector<Point2f>)points:(vector<Point2f>)points
byFiltringInRect:(cv::Rect)rect {
        size_t size = 0;
        std::vector<Point2f> pointsInRect (points.size());
        std::copy_if(points.begin(), points.end(),
                     pointsInRect.begin(), [&rect, &size](Point2f
point)
                     {
                         if (rect.contains(point)) {
                             size++;
                             return YES;
                         }
                         return NO;
                     });
        pointsInRect.resize(size);
        return pointsInRect;
    }
    - (void)calcModelviewMat:(cv::Mat &)modelView
                  fromRotVec:(cv::Mat const &)rotVec
                    transVec:(cv::Mat const &)tVec {

        cv::Mat rotation;
        cv::Mat viewMatrix(cv::Mat::zeros(4, 4,
cv::DataType<double>::type));
        cv::Rodrigues(rotVec, rotation);

        for(unsigned int row=0; row<3; ++row)
        {
            for(unsigned int col=0; col<3; ++col)
            {
                viewMatrix.at<double>(row, col) =
rotation.at<double>(row, col);
            }
```

```
            viewMatrix.at<double>(row, 3) = tVec.at<double>(row,
0);
        }
        viewMatrix.at<double>(3, 3) = 1.0f;

        cv::Mat cvToGl = cv::Mat::zeros(4, 4,
cv::DataType<double>::type);
        cvToGl.at<double>(0, 0) = 1.0f;
        cvToGl.at<double>(1, 1) = -1.0f; // Invert the y axis
        cvToGl.at<double>(2, 2) = -1.0f; // invert the z axis
        cvToGl.at<double>(3, 3) = 1.0f;
        viewMatrix = cvToGl * viewMatrix;

        cv::transpose(viewMatrix , viewMatrix);
        viewMatrix.convertTo(modelView, cv::DataType<float>::type);
    P72
    }
    /*
     For the simple common case where the OpenCV camera matrix has
the form:

     |fx  0 cx|
     |0  fy cy|
     |0   0  1|
     */
    - (void)calcProjectionMat:(float *)projMat
                fromCameraMat:(cv::Mat const &)camMat
               withScreenSize:(CGSize)screen
                        znear:(float)znear
                         zfar:(float)zfar {

        memset((void *)projMat, 0, 16 * sizeof(float));

        float fx = cvRound(camMat.at<double>(0, 0));
        float fy = cvRound(camMat.at<double>(1, 1));

        projMat[0] = 2.0 * fx / screen.width;
        projMat[5] = 2.0 * fy / screen.height;
        projMat[10] = -(zfar + znear) / (znear - zfar);
        projMat[11] = -1.f;
        projMat[14] = - 2.0 * zfar * znear / (znear - zfar);
    }
    - (void)freeResources {
        _status.clear();
        _points.clear();
        _prevPoints.clear();
        _modelProjection.clear();
        _prevGray = Mat();
        _gray = Mat();
    }
    static float counter = 1;
    std::vector<cv::Point2f> Generate2DPoints(cv::Mat &image)
    {
```

```
    std::vector<cv::Point2f> points;
    counter -= 0.01;
    if (counter < 0) counter = 1;

    float x,y;

    x=image.cols * counter;y=image.rows * 0/3;
    points.push_back(cv::Point2f(x,y));

    x=image.cols * counter;y=image.rows * counter;
    points.push_back(cv::Point2f(x,y));

    x=image.cols * 0/3;y=image.rows * counter;
    points.push_back(cv::Point2f(x,y));

    x=image.cols * 0/3;y=image.rows * 0/3;
    points.push_back(cv::Point2f(x,y));

    x=image.cols * counter / 2;y=image.rows * counter / 2;
    points.push_back(cv::Point2f(x,y));

    return points;
}
std::vector<cv::Point3f> Generate3DPoints()
{
    std::vector<cv::Point3f> points;

    float x,y,z;

P73
    x=-50.0;y=-50.0;z=50;
    points.push_back(cv::Point3f(x,y,z));

    x=50.0;y=-50.0;z=50;
    points.push_back(cv::Point3f(x,y,z));

    x=-50.0;y=50.0;z=50;
    points.push_back(cv::Point3f(x,y,z));

    x=50.0;y=50.0;z=50;
    points.push_back(cv::Point3f(x,y,z));

    return points;
}
@end
```

A.3 Клас ExternalRenderer

```
#import <Foundation/Foundation.h>
#import <QuartzCore/QuartzCore.h>
typedef void(^ExternalRenderBlock)(CADisplayLink *displayLink);
@interface ExternalRenderer : NSObject
```

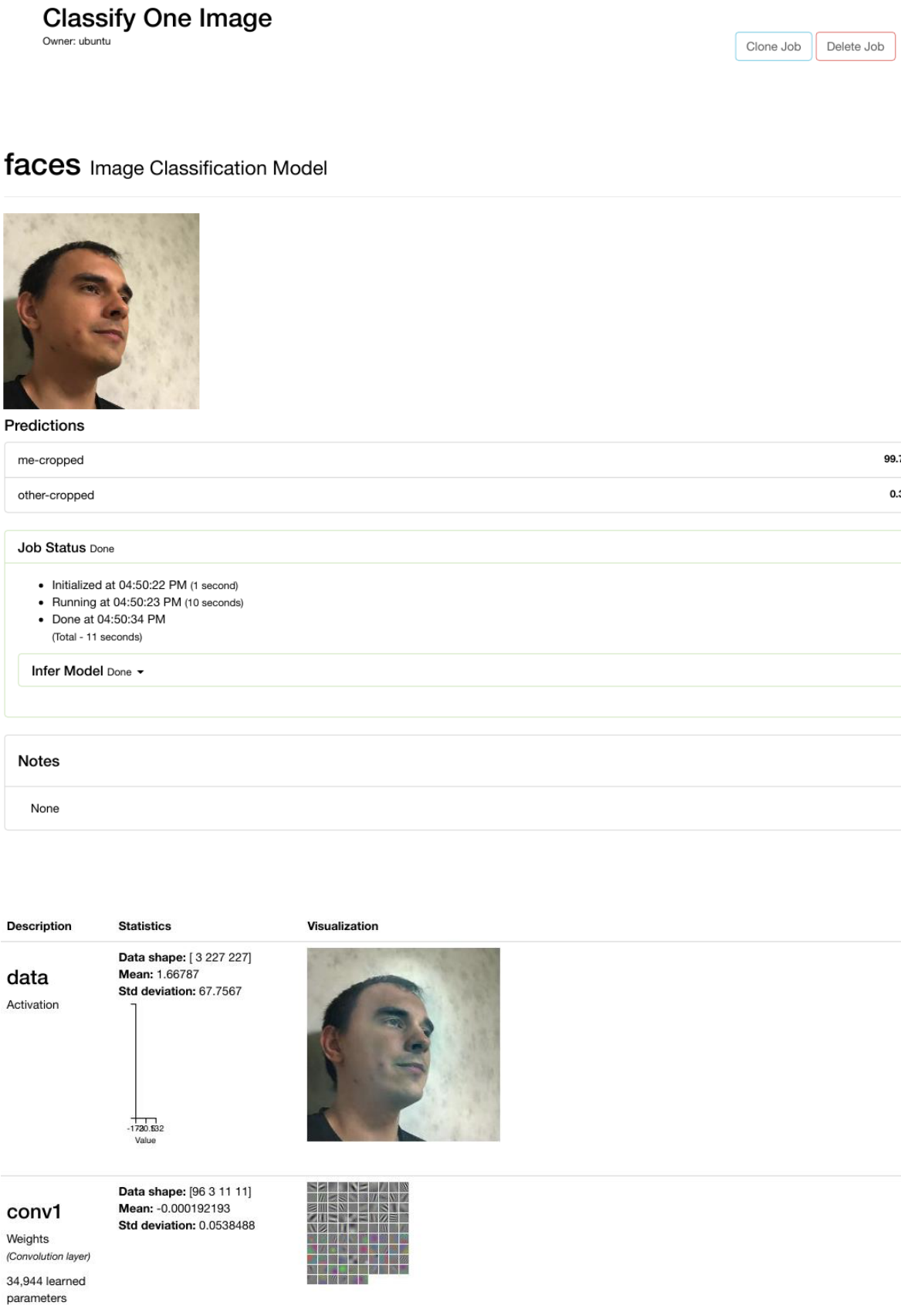
```
    @property (nonatomic, readonly) EAGLContext *context;
    @property (nonatomic, strong) CAEAGLLayer *eaglLayer;
    -
(void)startRenderLoopWithRenderBlock:(ExternalRenderBlock)renderBloc
k;
    - (void)stopRenderLoop;
    - (void)bindBuffer;
    - (void)teardownRendering;
    @end
    @protocol SSRenderable <NSObject>
    @property (nonatomic, assign) CGFloat scale;
    - (void)drawAt:(NSTimeInterval)time;
    - (void)setProjectionMatrix:(const float *)projectionMatrix;
    - (void)setModelViewMatrix:(const float *)modelViewMatrix;
    - (void)releaseProgram;
    @end
    #import "ExternalRenderer.h"
    #import "OpenGLHelper.h"
    #import <QuartzCore/QuartzCore.h>
    #include <OpenGLES/EAGL.h>
    #include <OpenGLES/ES2/gl.h>
    #include <OpenGLES/ES2/glext.h>
    @interface ExternalRenderer()
    @property (nonatomic, strong) EAGLContext *context;
    @property (nonatomic, strong) CADisplayLink *displayLink;
    @property (nonatomic, assign) GLuint colorRenderBuffer;
    @property (nonatomic, assign) GLuint depthRenderBuffer;
    @property (nonatomic, assign) GLuint framebuffer;
    @property (nonatomic, copy) ExternalRenderBlock renderBlock;
    @end
    @implementation ExternalRenderer
    - (void)setEaglLayer:(CAEAGLLayer *)eaglLayer
    {
        _eaglLayer = eaglLayer;

        [self setupContext];
        [self setupDepthBuffer];
        [self setupRenderBuffer];
        [self setupFrameBuffer];
    }
    -
(void)startRenderLoopWithRenderBlock:(ExternalRenderBlock)renderBloc
k
    {
        NSParameterAssert(renderBlock);
        self.renderBlock = renderBlock;
        [self setupDisplayLink];
    }
    - (void)stopRenderLoop
    {
        [self.displayLink invalidate];
        self.displayLink = nil;
    }
```

```
- (void)teardownRendering
{
    self.renderBlock = nil;
    self.context = nil;

    if (self.colorRenderBuffer) {
        glDeleteRenderbuffers(1, &_colorRenderBuffer);
    }

    if (self.framebuffer) {
        glDeleteFramebuffers(1, &_framebuffer);
    }
}
- (void)bindBuffer
{
    [EAGLContext setCurrentContext:self.context];

    glBindRenderbuffer(GL_RENDERBUFFER,
self.depthRenderBuffer);
    glBindRenderbuffer(GL_RENDERBUFFER,
self.colorRenderBuffer);
    glBindFramebuffer(GL_FRAMEBUFFER, self.framebuffer);
}
#pragma mark - Helper
- (void)setupContext
{
    self.context = [[EAGLContext alloc]
initWithAPI:kEAGLRenderingAPIOpenGLES2];

    if (!self.context) {
        NSLog(@"Failed to initialize OpenGLES 2.0 context");
        exit(1);
    }

    if (![EAGLContext setCurrentContext:_context]) {
        NSLog(@"Failed to set current OpenGL context");
        exit(1);
    }
}
- (void)setupDepthBuffer
{
    glGenRenderbuffers(1, &_depthRenderBuffer);
    glBindRenderbuffer(GL_RENDERBUFFER,
self.depthRenderBuffer);
    glRenderbufferStorage(GL_RENDERBUFFER,
GL_DEPTH_COMPONENT16,
                          self.eaglLayer.bounds.size.width,
self.eaglLayer.bounds.size.height);
}
- (void)setupRenderBuffer
{
    glGenRenderbuffers(1, &_colorRenderBuffer);
```

```
        glBindRenderbuffer(GL_RENDERBUFFER,
self.colorRenderBuffer);
        [self.context renderbufferStorage:GL_RENDERBUFFER
fromDrawable:self.eaglLayer];
    }
    - (void)setupFrameBuffer
    {
        glGenFramebuffers(1, &_framebuffer);
        glBindFramebuffer(GL_FRAMEBUFFER, self.framebuffer);
        glFramebufferRenderbuffer(GL_FRAMEBUFFER,
GL_COLOR_ATTACHMENT0,
                                  GL_RENDERBUFFER,
self.colorRenderBuffer);
        glFramebufferRenderbuffer(GL_FRAMEBUFFER,
GL_DEPTH_ATTACHMENT,
                                  GL_RENDERBUFFER,
self.depthRenderBuffer);
        glViewport(0, 0, self.eaglLayer.frame.size.width,
self.eaglLayer.frame.size.height);
    }
    - (void)setupDisplayLink
    {
        self.displayLink =
        [CADisplayLink displayLinkWithTarget:self
                                    selector:@selector(render:)];

        [self.displayLink addToRunLoop:[NSRunLoop currentRunLoop]
                               forMode:NSDefaultRunLoopMode];
    }
    - (void)render:(CADisplayLink *)link
    {
        [EAGLContext setCurrentContext:self.context];
        glDepthMask(true);
        glClearColor(0.f, 0.3f, 0.3f, 1.f);
        glClearDepthf(1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glEnable(GL_DEPTH_TEST);

        if (self.renderBlock) {
            self.renderBlock(link);
        }

        [self.context presentRenderbuffer:GL_RENDERBUFFER];

        [EAGLContext setCurrentContext:nil];
    }
@end
```

ДОДАТОК Б

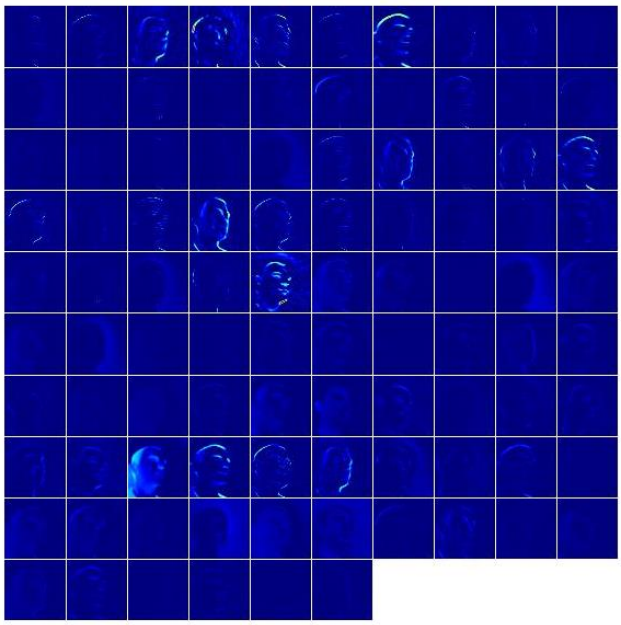
Результати тестових експериментів



conv1

Activation

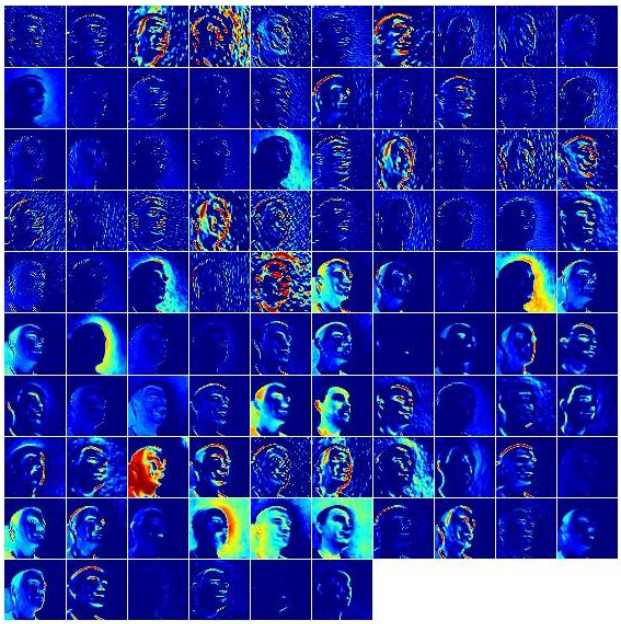
Data shape: [96 55 55]
Mean: 16.4927
Std deviation: 48.6786



norm1

Activation

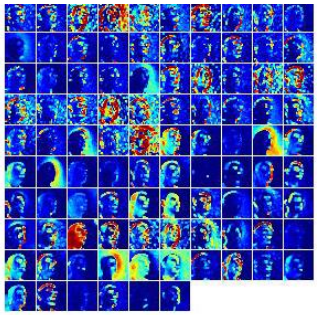
Data shape: [96 55 55]
Mean: 11.9165
Std deviation: 23.3132



pool1

Activation

Data shape: [96 27 27]
Mean: 26.1409
Std deviation: 32.5803

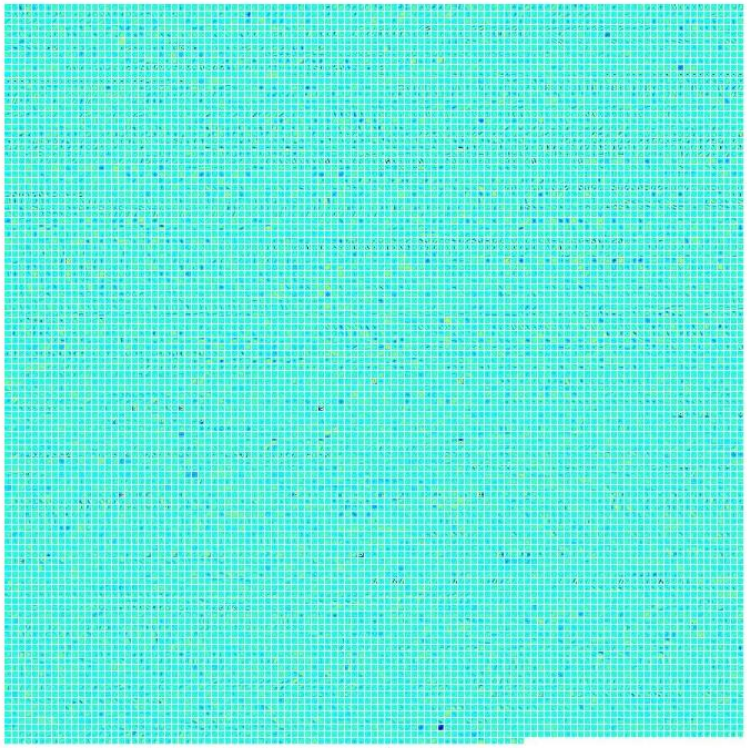


conv2

Weights
(Convolution layer)

307,456 learned parameters

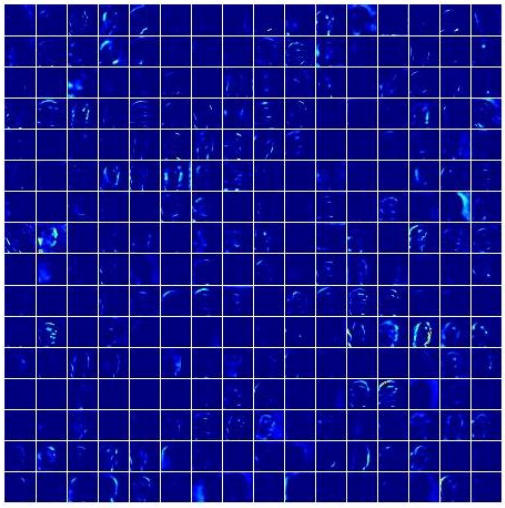
Data shape: [256 48 5 5]
Mean: -0.00138049
Std deviation: 0.0224899



conv2

Activation

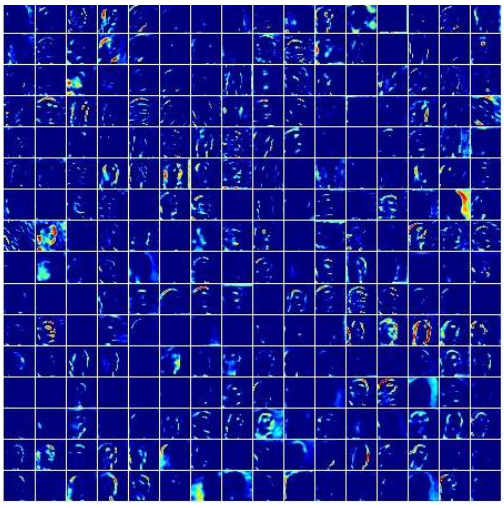
Data shape: [256 27 27]
Mean: 5.2726
Std deviation: 18.3718



norm2

Activation

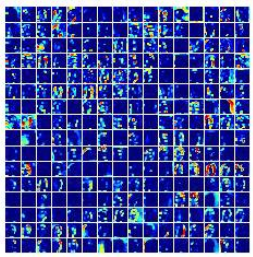
Data shape: [256 27 27]
Mean: 4.72547
Std deviation: 15.1745



pool2

Activation

Data shape: [256 13 13]
Mean: 13.0969
Std deviation: 25.1705

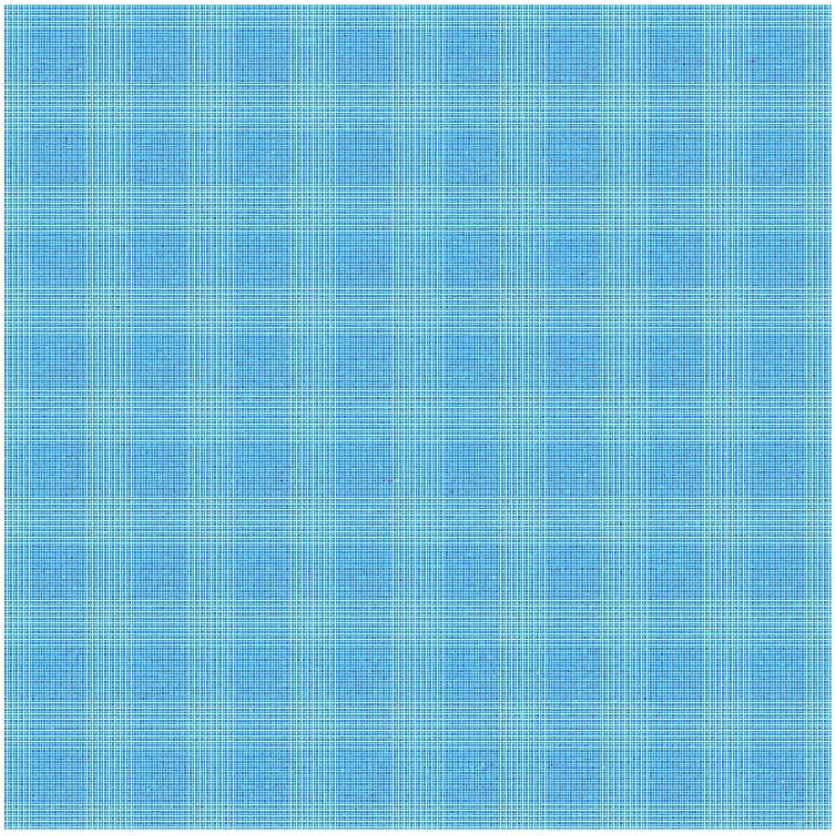


conv3

Weights
(Convolution layer)

885,120 learned parameters

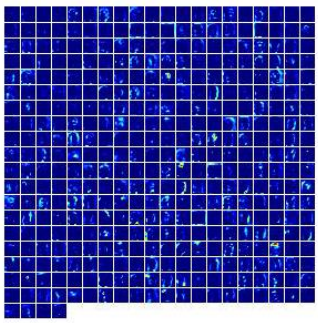
Data shape: [384 256 3 3]
Mean: -0.000862584
Std deviation: 0.0150457



conv3

Activation

Data shape: [384 13 13]
Mean: 6.49212
Std deviation: 16.2144



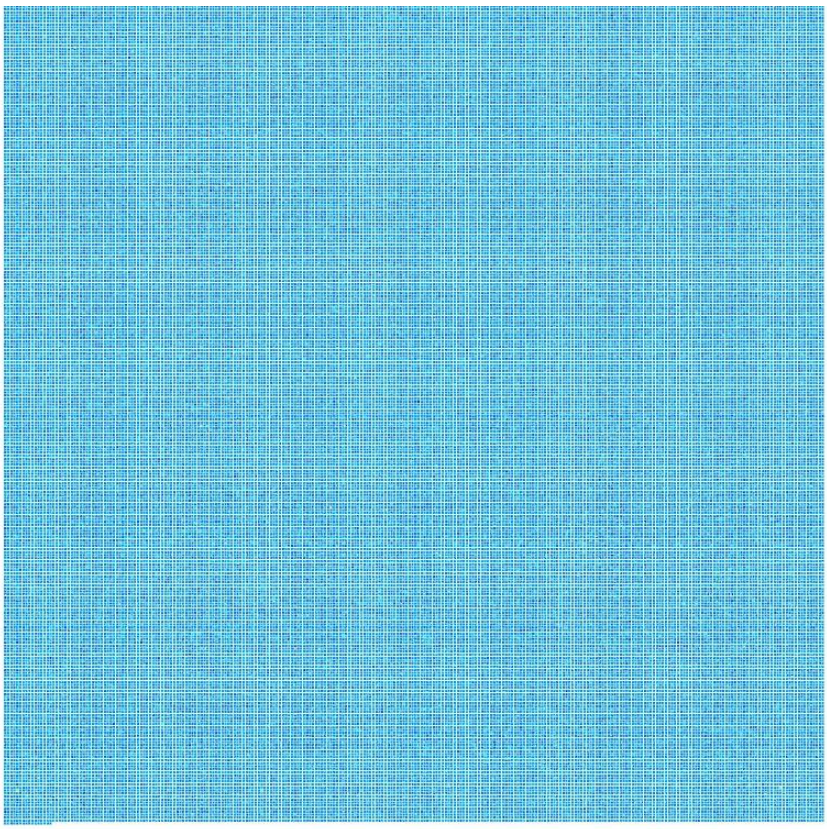
conv4

Weights

(Convolution layer)

663,936 learned parameters

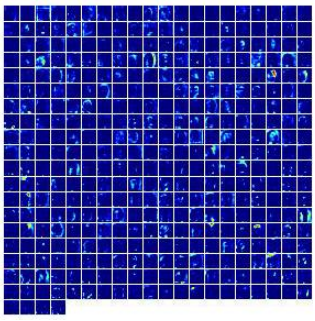
Data shape: [384 192 3 3]
Mean: -0.00129242
Std deviation: 0.0161757



conv4

Activation

Data shape: [384 13 13]
Mean: 3.30458
Std deviation: 8.11716

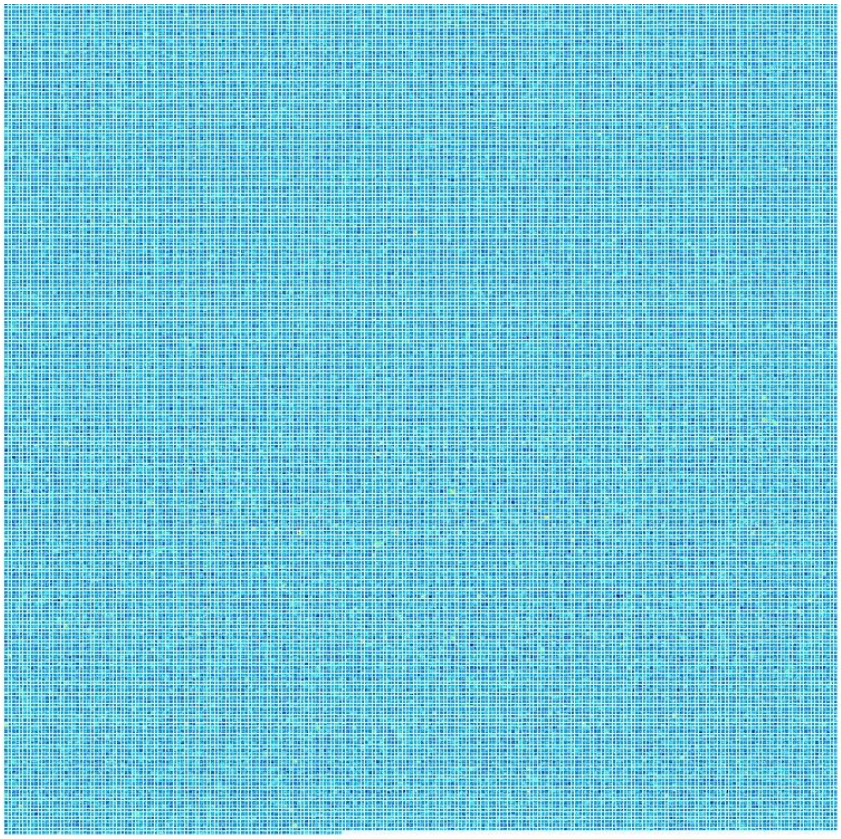


conv5

Weights
(Convolution layer)

442,624 learned parameters

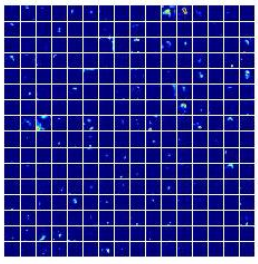
Data shape: [256 192 3 3]
Mean: -0.00263172
Std deviation: 0.0175113



conv5

Activation

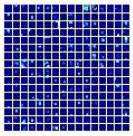
Data shape: [256 13 13]
Mean: 0.404745
Std deviation: 2.36035



pool5

Activation

Data shape: [256 6 6]
Mean: 1.48332
Std deviation: 4.63594



fc6

Weights
(InnerProduct layer)

37,752,832 learned parameters

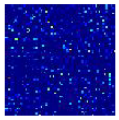
Data shape: [4096 9216]
Mean: -0.000306774
Std deviation: 0.00409036

Not shown

fc6

Activation

Data shape: [4096]
Mean: 0.545412
Std deviation: 1.59786



fc7

Weights
(InnerProduct layer)

16,781,312 learned parameters

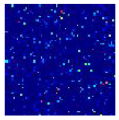
Data shape: [4096 4096]
Mean: -0.000799385
Std deviation: 0.00566364

Not shown

fc7

Activation

Data shape: [4096]
Mean: 0.0977236
Std deviation: 0.365498



fc8

Weights
(InnerProduct layer)

8,194 learned parameters

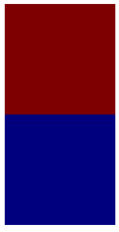
Data shape: [2 4096]
Mean: -3.30214e-05
Std deviation: 0.0104389

Not shown

fc8

Activation

Data shape: [2]
Mean: -0.0601876
Std deviation: 2.90158



softmax

Activation

Data shape: [2]
Mean: 0.5
Std deviation: 0.496991

Totals

Total learned parameters:
56,876,418

ДОДАТОК В

Слайди презентації

Рисунок В.1 – Титульний слайд

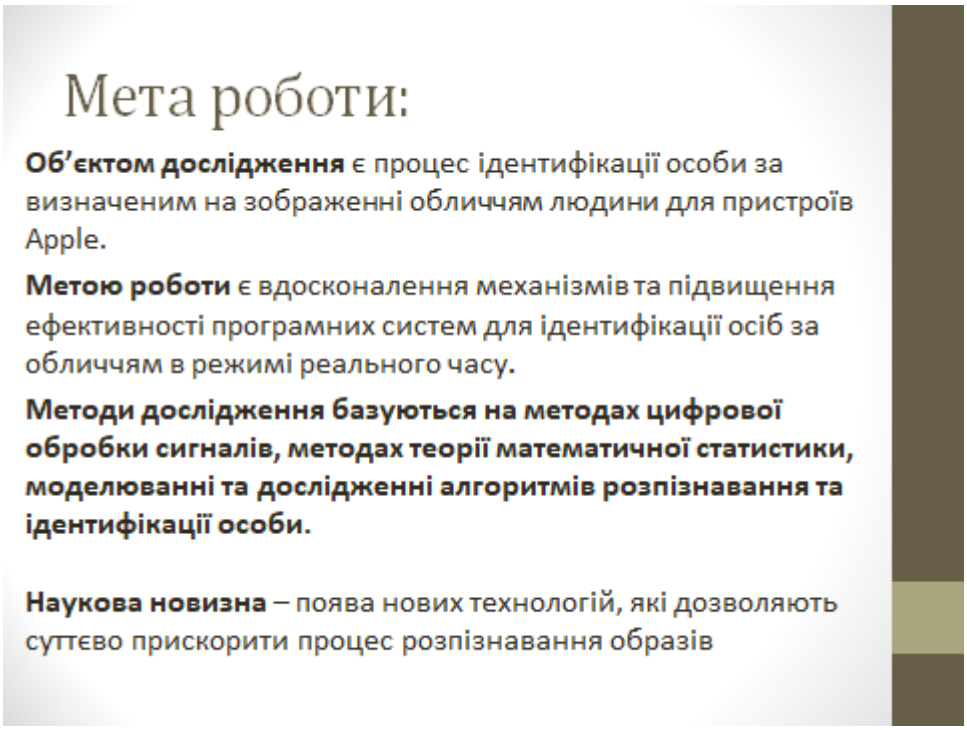


Рисунок В.2 – Мета роботи

Рисунок В.3 – Визначення розпізнавання образів

Рисунок В.4 – Система розпізнавання образів

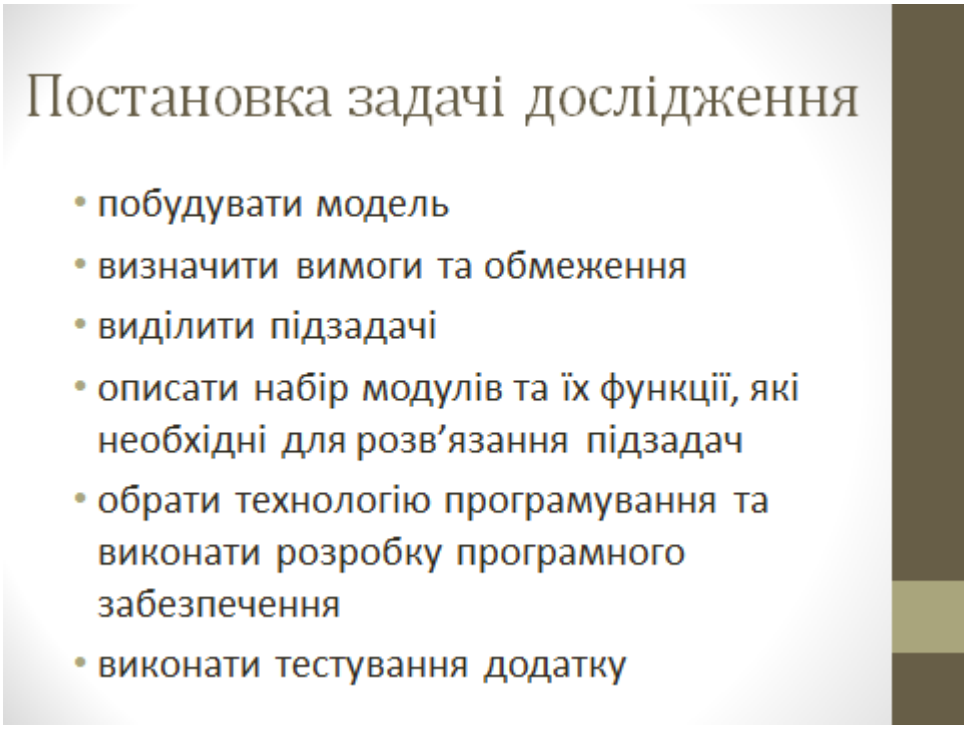


Рисунок В.5 – Постановка задачі дослідження

Рисунок В.6 – Алгоритми розпізнавання за обличчям

Рисунок В.7 – Прототипи-аналоги

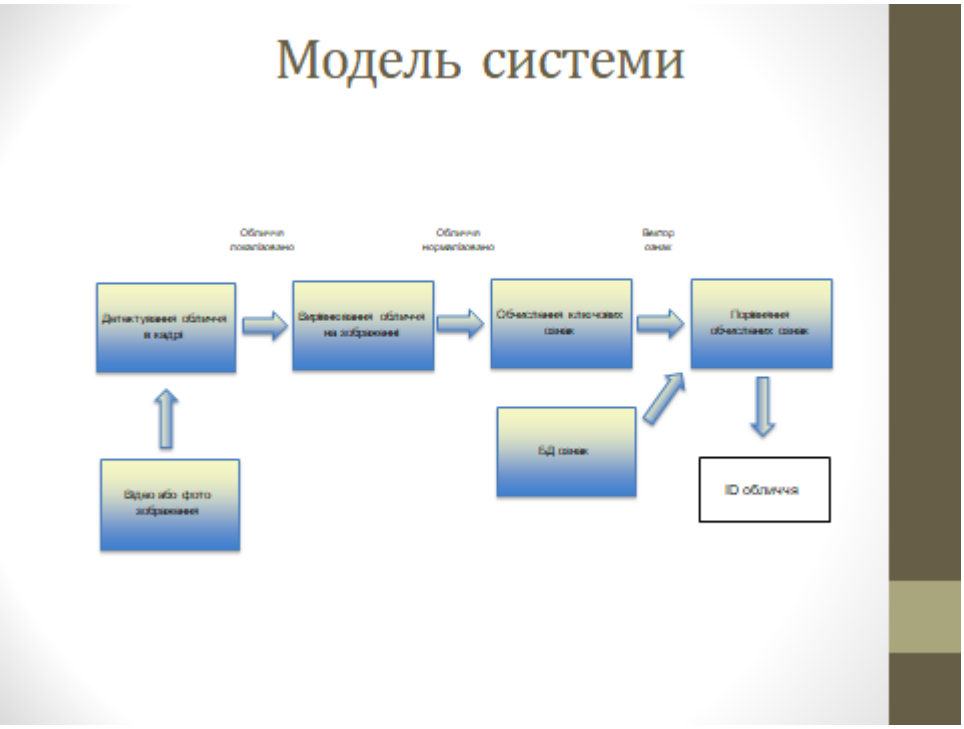


Рисунок В.8 – Модель системи

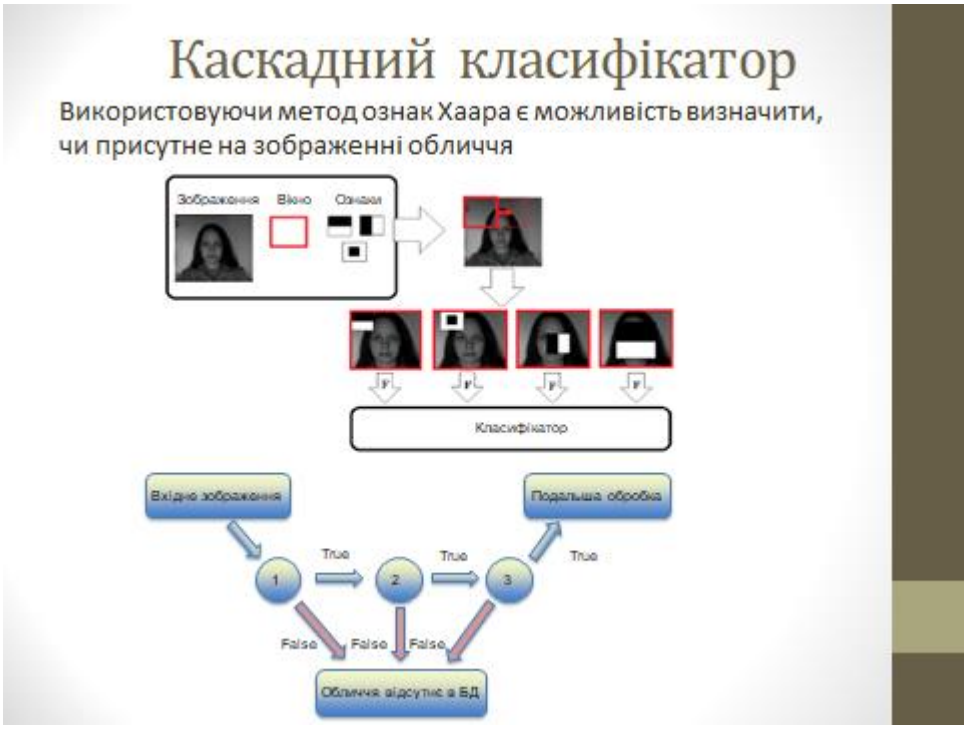


Рисунок В.9 – Класифікатор за ознаками Хаара

Рисунок В.10 – Перешкоди при розпізнаванні

Рисунок В.11 – Алгоритми корекції зображень

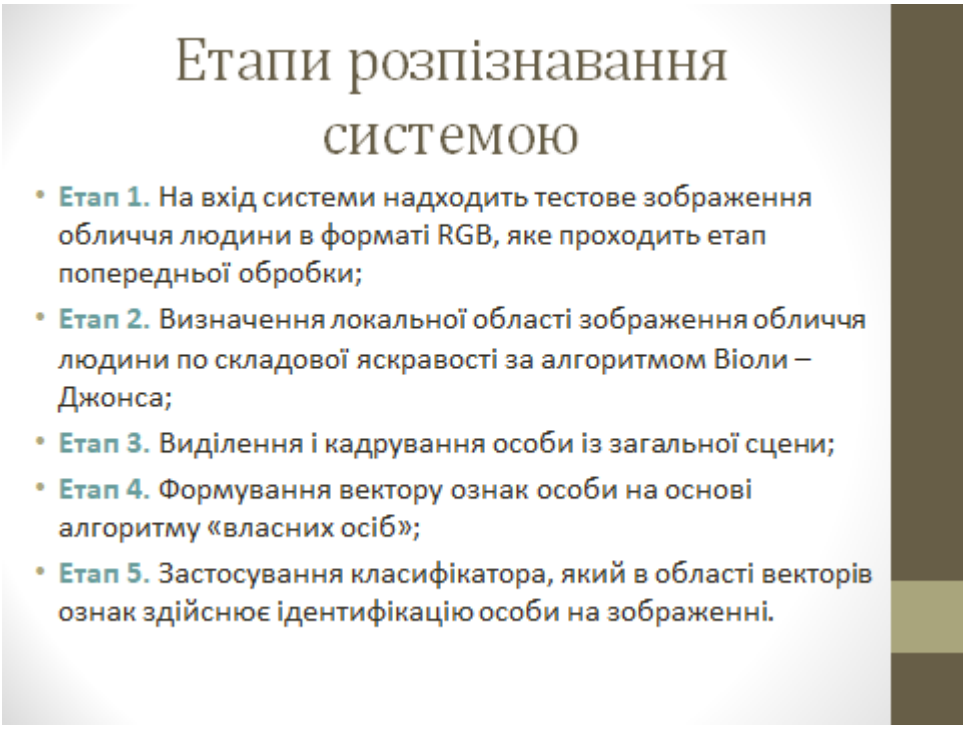


Рисунок В.12 – Алгоритми корекції зображень

Рисунок В.13 – Архітектура системи

Рисунок В.14 – Діаграма послідовностей

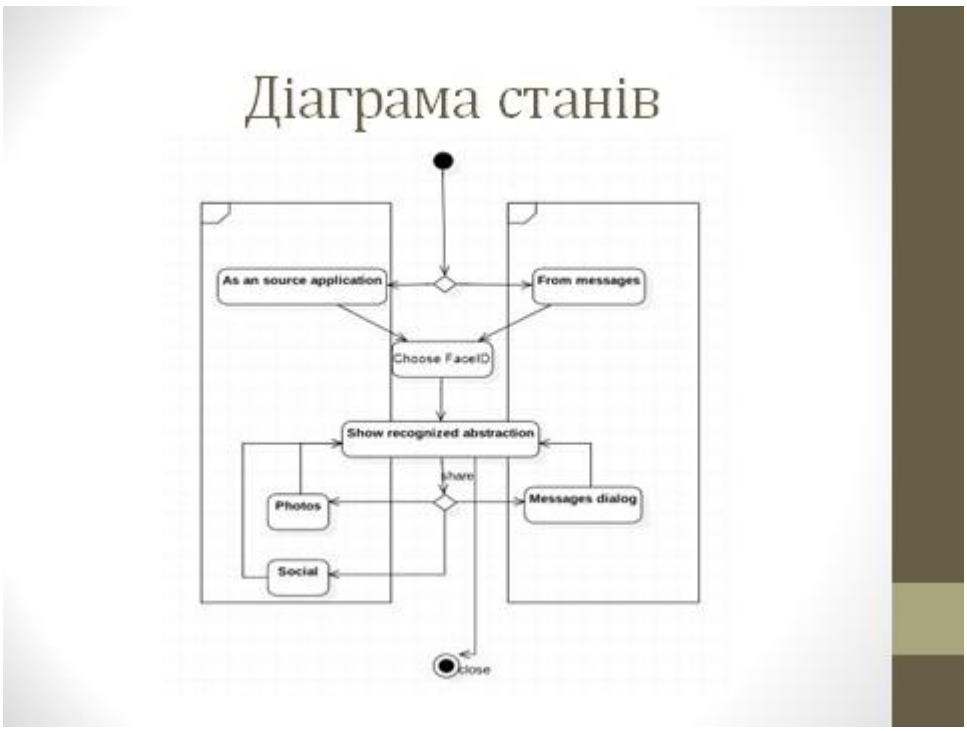


Рисунок В.15 – Діаграма станів

Рисунок В.16 – Ієрархія модулів програми

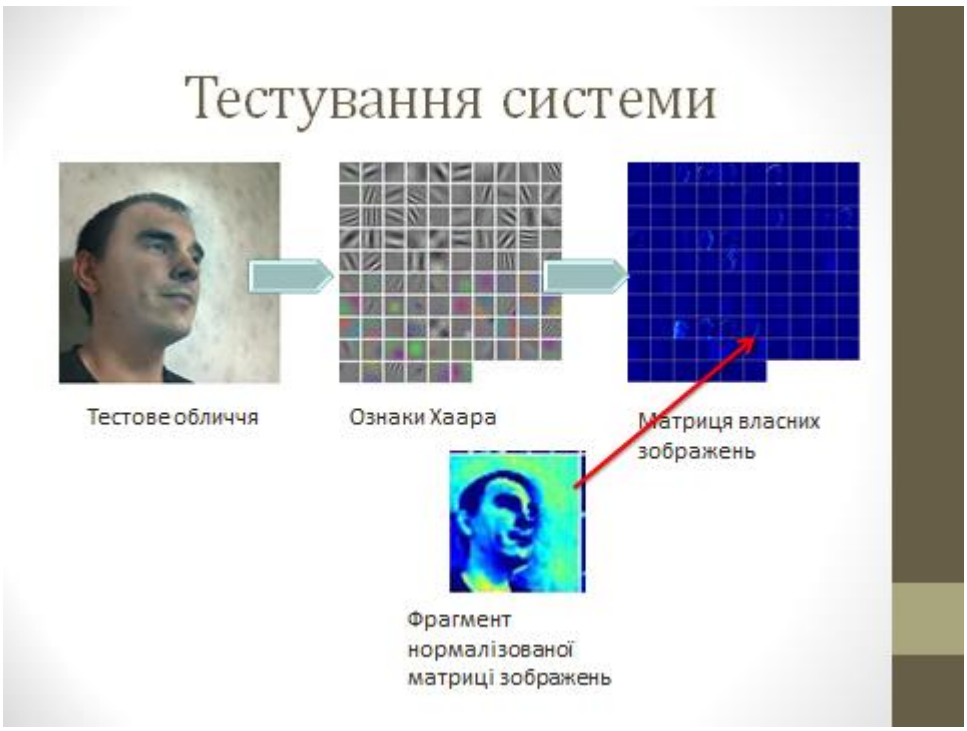


Рисунок В.17 – Тестування системи

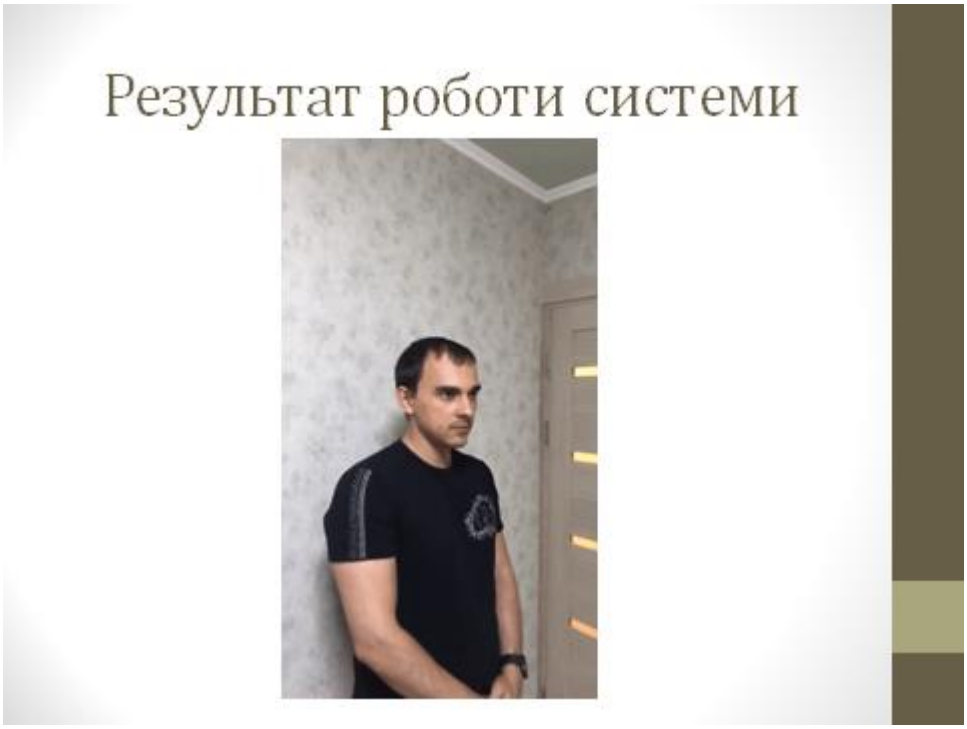


Рисунок В.18 – Результат роботи системи

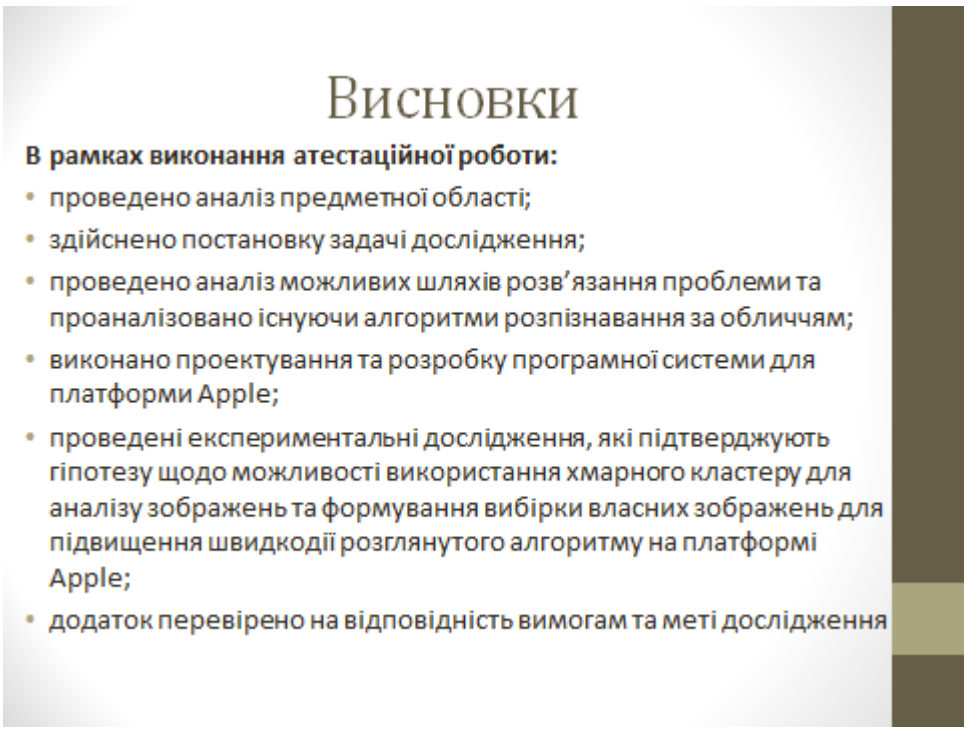


Рисунок В.19 – Висновки по роботі

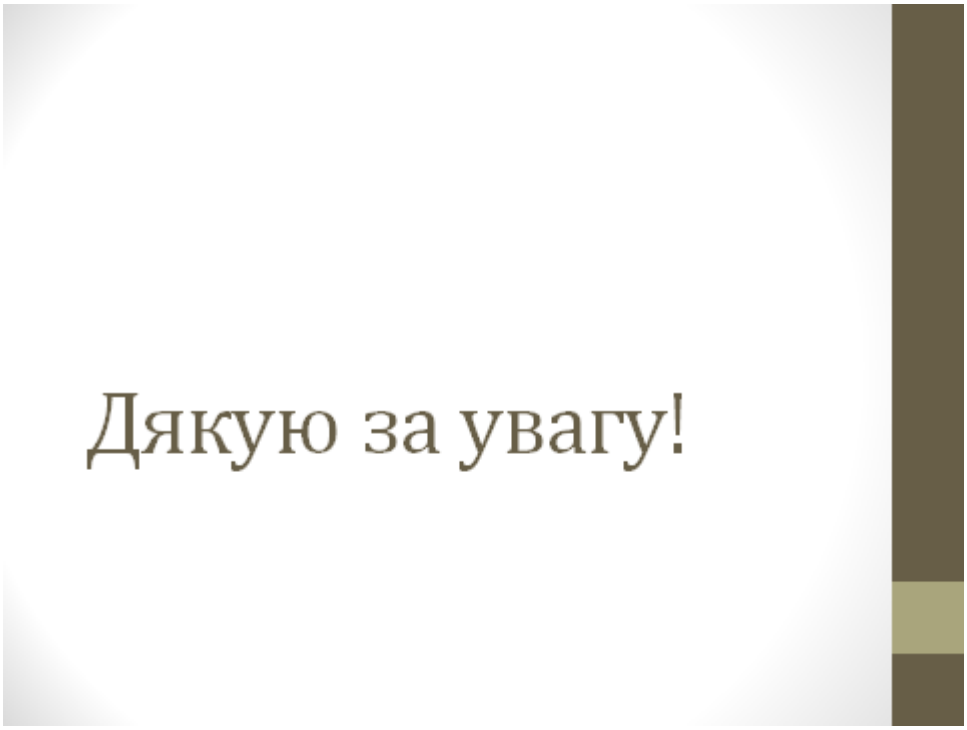


Рисунок В.20 – Останній слайд