Assignment 1

Akshay Aralikatti (012624106)
Ricardo Sparks (012737947)

Task 1: Summary on Cloud Scaling: Explore Video analytics in the cloud

The article talks about challenges and solutions associated with processing and analyzing large volumes of video data in cloud environments. It emphasizes the importance of scalable architectures to handle the increasing demand for video analytics across various industries. One of the primary concerns discussed is the massive volume of video data, which requires efficient storage and processing mechanisms. Many applications demand real-time analysis, necessitating low-latency and high speed processing capabilities. To address these challenges, the article highlights cloud-based solutions as an effective approach. Cloud platforms offer elastic resources, allowing systems to scale dynamically based on demand. Distributed computing frameworks help process video data efficiently across multiple servers, while cloud storage solutions provide scalable and reliable options for handling vast amounts of data.

This article also discusses key architectural considerations for cloud-based video analytics. A modular design makes it easier to scale and maintain the system. Load balancing ensures that resources are efficiently utilized. Fault tolerance mechanisms enhance reliability, ensuring uninterrupted operation even during failures.

Observations on OpenCV 2.x, 3.x to 4.x: OpenCV has been constantly undergoing changes. There are always new modules bening introduced, API changes, performance improvements and better support for modern computing hardware.

Modularity and Structural Changes: In 2.x all functions were part of a monolithic library, making it heavy and difficult to maintain. But with modularization: splitting OpenCV into different modules such as core, imgproc, ml, objdetect in 3.x it has improved maintainability and allowed developers to include only required modules.

Deep Learning Integration: 2.x had very limited support for deep learning. 3.x introduced DNN (Deep Neural Network) module, inference using pre-trained models from caffe, TensorFlow, ONNX. 4.x enhanced the DNN module, adding GPU acceleration using CUDA, OpenVINO, Vulkan.

C++ API and Code Simplification: 2.x used older C-style API requiring manual memory management, 3.x and 4.x introduced cleaner C++ API removing the need for manual memory management, making code more readable and object oriented.

Feature Detection and Object Tracking*:* 2.x provided feature detection algorithms like SIFT, SURF, ORB, FAST. But 3.x and 4.x introduced improved feature detection and optimized methods such as KCF (Kernelized Correlation Filter), MIL(Multiple Instance Learning) and MOSSE. DeepSORT Tracking.

Improved Video Processing: 2.x had basic video input/output support. But 3.x improved VideoCapture and VideoWriter APIs and added support for H.264, HEVC Hardware encoding and VP9. 4.x optimized multi-threaded video processing and integrated advanced real-time processing frameworks.

Comparison of Early and Modern Methods for Pose Estimation in Video Analysis:
Pose estimation is a critical component in Video analysis used for tracking the movement of animals, aircraft, and humans. Over time, methodologies have evolved from traditional computer vision techniques to deep learning-based approaches such as MediaPipe, PoseNet, and OpenPose. Early pose estimation relied on handcrafted features, optical flow and statistical models, often struggling with lighting, occlusions and scalability. Techniques like HOG-SVM, kinematic skeletons and background subtraction required manual tuning and lacked generalization. With deep learning, modern methods like MediaPipe, PoseNet and OpenPose revolutionized pose tracking by leveraging CNNs for higher accuracy, robustness and real-time performance. MediaPipe offers fast, lightweight 2D tracking optimized for mobile use, while PoseNet enables browser-based applications with decent accuracy. OpenPose, though computationally expensive, provides high-precision multi-person 2D and 3D tracking using part affinity fields(PAFs). Unlike early methods that struggled with multip-le subjects and environmental changes, deep learning-based approaches learn directly from data, making them more adaptable and scalable. These advance have improved applications in sports analysis, human-computer interaction, surveillance and robotics, far surpassing traditional techniques in efficiency and accuracy.

Task 2: Starter code example:

The given example code makes use of an in-built OpenCV library that manipulates video parameters, each file allowed to play around with the given image, Following are the files which could be used in providing API to edit images example: setting different contrast, brightness, sharpness levels.

a. Brighten.cpp:

What does it do: This program reads an image from a file and allows the user to adjust its brightness and contrast interactively. The user provides a brighten factor (alpha) and contrast increase value (beta):

new pixel = œ x original pixel + ß

Uses: Could be used in:
1. Photo editing software (adding more image adjustments like gamma correction, sharpening and noises reduction)
2. Real time image enhancement (GUI slider for adjusting brightness and contrast interactively)

Three OpenCV Functions used:

1. imread()
   Usage: Mat image = imread(argv[1]): reads an image from the specified file path into an OpenCV Mat Object.
2. imshow()
   Usage: imshow("Orignal Image", image): Displays an image in a window with a given name.
3. saturate_cast<uchar>()
   Usage: saturate_cast<uchar>(alpha * image.at<Vec3b>(y,x)[c] + beta): Ensures pixel values remain within the valid range [0,255] to avoid overflow and underflow.

b. Capture.cpp:

What does it do: This program captures live video from a webcam and displays it in a window called 'video_display'. It continuously reads frames from the camera and displays them in real time. The user can exit by pressing the ESC key.

Uses: Could be used in:
1. Real-time Object Detection - Integrate OpenCV's object detection models (like Haar cascades or YOLO) to detect objects live.
2. Motion tracking - Enhance the code by adding movement tracking using optical flow or background subtraction.

Three OpenCV Functions used:

4. VideoCaputre()
   Usage: VideoCapture cam0(0): Opens the default camera (index 0) for video capture.
5. imshow()
   Usage: imshow("video_display", frame): Displays captured frame in a window.
6. waitKey()
   Usage: if ((winInput = waitkey(1)) == ESC): Waits for a key press for a given time (1ms here); if ESC is pressed, it exits the loop.

c. Diffcapture.cpp:

What does it do: This program captures video frames, converts them to grayscale, and computes the difference between consecutive frames. The difference (mat_diff) represents the changes in the scene, useful for motion detection. If significant movement is detected, the program displays the percentage difference.

Uses: Could be used in:

1. Security Surveillance: Detect and alert when motion in recordings is detected.
2. Gesture recognition: Recognize specific movements and gestures.
3. Animal movement tracking: To identify and not wildlife footage.

Three OpenCV Functions used:

7. cvtColor()
   Usage: cv::cvtColor(mat_frame, mat_grey, COLOR_BGR2GRAY):Converts an image from color (BGR) to grayscale.
8. absdiff()
   Usage: absdiff(mat_gray_prev, mat_gray, mat_diff): Computes the absolute difference between two frames, highlighting changes.
9. putText()
   Usage: cv::putText(mat_diff, difftext, Point(30,30), FONT_HERSHEY_COMPLEX_SMALL, 0.8, Scalar(200,200,250), 1, LINE_AA: Displays text(difftext) on the image.

Task 3. Sharpen-psf example

There are two files namely sharpen.cpp and sharpen_grid.cpp. Both programs perform edge sharpening on an image using a convolution operation with a Point Spread Function (PSF). The key difference between them is that sharpen_grid.c is optimized for real-time processing using multi-threading, making it significantly faster than the simpler sharpen.c.

What is PSF and Why does it provide edge sharpening?

The point spread Function(PSF) is a mathematical model used to describe how an imaging system affects a point source of light. In image processing, PSF-based convolution is commonly used to apply sharpening filters. The PSF matrix is sharpen.c applies a 3x3 convolution kernel, where the central pixel (k+1) is weighted higher while surrounding pixels are slightly subtracted. This enhances edges by increasing contrast at transitions, making details appear sharper.

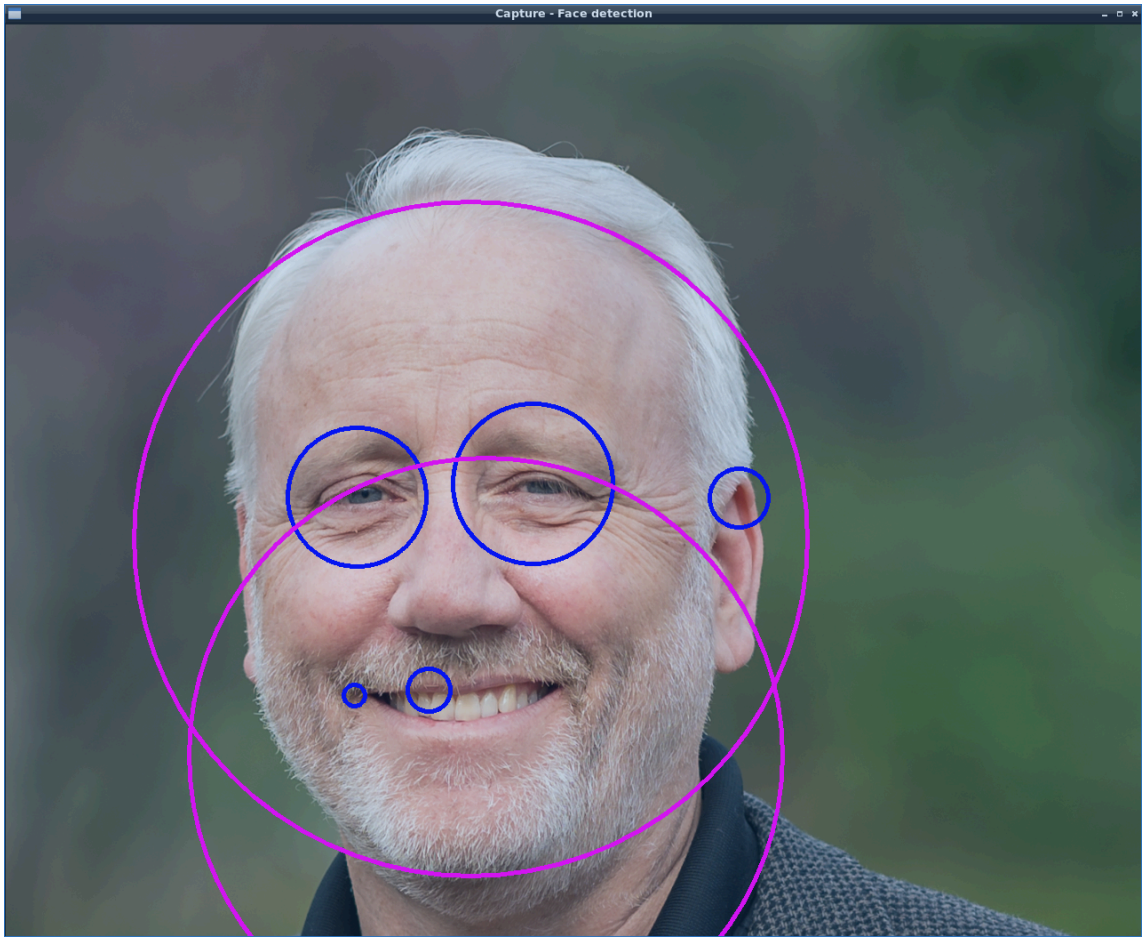
How is sharpen_grid.c different from `sharpen.c and why is it better?

Sharpen.c uses Single-Threaded Image Sharpening This code applies a convolution-based edge enhancement filter to PPM image. Sharpen.c processes the image sequentially. On the other hand sharpen_grid.c splits the image into multiple blocks and assigns them to different threads using POSIX(pthreads). Each thread independently applies PSF-based convolution to its region, significantly improving performance. Sharpen_grid.c is ideal for large images and real-time applications, such as video frame processing, where speed is crucial.

The following is the image whose sharpening was achieved with sharpen_grid.c:

Task 4: Building the code in faceDetect

The following picture demonstrates the face detection:



How does Haar cascade Object Detection work?
It is a machine learning based approach where cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.The algorithm needs a lot of positive images (images of faces) and negative images(images without faces) to train the classifier. Then the features are extracted similar to convolutional kernels. Each feature is a single value obtained by subtracting the sum of pixels under the white rectangle from the sum of pixels under rectangle. Includes Edge features, Line features and Four-rectangle features. Since each kernel takes a long time to compute ex: 24x24 would take 160000 features. This could be burden on computation. To solve this they introduced the integral image. Haar Cascade is a machine learning-based object detection method that identifies objects like faces and eyes in images. It uses Haar features, which are rectangular regions representing intensity differences. The classifier applies integral images for fast computation and employs AdaBoost to select the best features. The detection process uses a cascade of classifiers, starting with simple checks and progressively Haar Cascades for detecting faces, eyes and more. Although fast, Haar cascades are less accurate than modern deep learning approaches like CNNs and DNN models.

Task 5: own simple OpenCV code to open camera and produce cross hairs

The following is the image from the code: