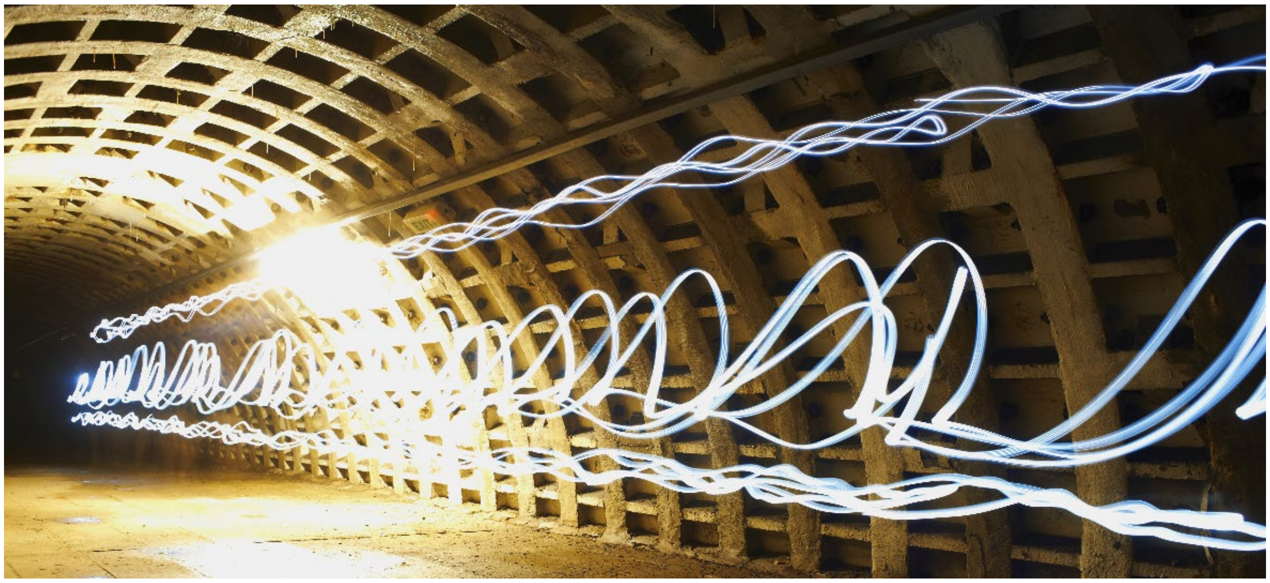


VISION ANALYTICS

AI Computer Vision Solutions Architecture

By Vishal Kapur, Douglas Bourgeois, Amina Jackson, Julie Kim, Taylor Jones, and Zachary Zweig

Rapid development and implementation of Artificial Intelligence (AI) has enabled many organizations to act faster, improve mission outcomes, and reduce costs. As part of this adoption trend, clients are increasingly requiring automated analytics solutions that can generate accurate and meaningful insights to meet new business trends and customer demands. One such example, Computer Vision (CV), utilizes AI to automatically process and analyze digital images or video files. This capability is derived from advanced machine learning (ML) techniques that can train algorithms to parse and perceive information contained within images or videos in much the same way humans 'see' – though CV can utilize inputs outside of the confines of our visible spectrum. As advancements in hardware, software, and machine learning toolkits make it easier to leverage real-time data and reduce latency, companies are focusing on computer vision applications to solve real-world problems in various fields. A few examples of CV capabilities include:

Common Computer Vision Applications

- Security and Surveillance – Collecting and analyzing security footage at scale (e.g. security at airports, malls, and military bases)
- Asset Management – Remotely monitoring assets reduces costs, optimizes business processes and increases public safety (e.g., utility management, pothole detection, traffic monitoring)
- Autonomous Vehicles – Unmanned Aerial Vehicles, Unmanned Underwater Vehicles, and Unmanned ground vehicles rely on computer vision to navigate their environments
- Banking, Insurance, and Finance –ATM facial recognition access and services and home loan evaluation
- Healthcare and Medical Research – Detecting and diagnosing diseases from radiology and pathology image results complement patient care and medical services
- Retail and Advertising – Augmenting a shopper's experience (e.g., targeted advertising) increases sales/revenues

The remainder of this paper will focus on the development of a Computer Vision Solutions Architecture (CVSA) designed to accelerate the development of CV-based solutions. With an in-depth understanding of the requirements involved, this solution architecture can be used as a sample guide for getting started in developing or tailoring CV solutions to your client's specific use case needs.

This architecture addresses typical machine learning challenges:

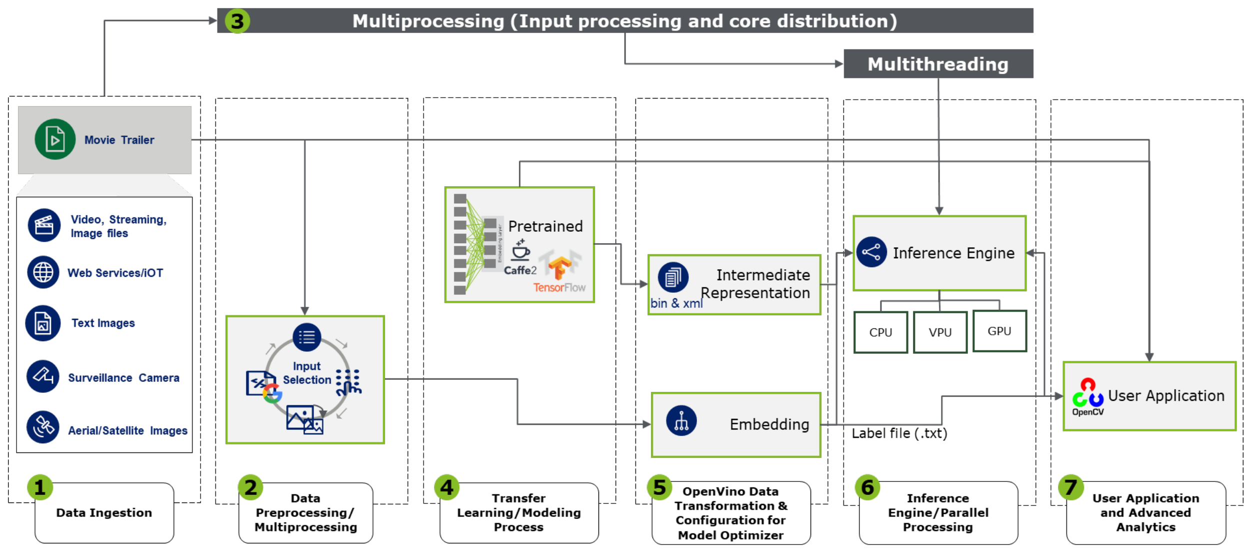
- Real-time predictions can be limited due to processing time
- Model underfitting: Can signify low accuracy in predictions, large data losses, or lack of training data
- Model overfitting: Misleading results with high accuracy can be produced by mixing validation testing data with training data
- In general, training the model for deep learning can be expensive in terms of time and cost
- As data volumes increase processing time needs to be evaluated for feasibility

Accompanying the solutions architecture are Facial Recognition, Age and Gender Detection, and Pose Estimation proofs-of-concepts. In addition to demonstrating the solutions architecture's value as an accelerator, this proof of concept facilitates a discussion around some of the technical considerations that must be adjudicated such as software and hardware configurations, performance optimization, and scalability.

Computer Vision Solutions Architecture (CVSA)

The Computer Vision Solutions Architecture (See Exhibit 1) is designed to build a simple computer vision solution and depicts different approaches taken to compare hardware performance. Because CV is a specific implementation of machine learning, execution requires typical ML pipelines e.g. data ingestion, data preprocessing, training, transfer-learning, feature engineering, hyper-parameter tuning, and model evaluation.

EXHIBIT 1 | COMPUTER VISION SOLUTIONS ARCHITECTURE



3 Multiprocessing (Input processing and core distribution)
Multithreading
Movie Trailer
Video, Streaming, Image files
Web Services/IOT
Text Images
Surveillance Camera
Aerial/Satellite Images
Input Selection
Pretrained
Caffe2
TensorFlow
Intermediate Representation
bin & xml
Embedding
Inference Engine
CPU
VPU
GPU
Label file (.txt)
User Application
OpenCV
1 Data Ingestion
2 Data Preprocessing/ Multiprocessing
4 Transfer Learning/Modeling Process
5 OpenVino Data Transformation & Configuration for Model Optimizer
6 Inference Engine/Parallel Processing
7 User Application and Advanced Analytics

This CVSA relies primarily upon Convolutional Neural Networks (CNNs), a Deep Learning (DL) technique within the family of Recurrent Neural Networks (RNNs).

Diverging pathways represent potential model configurations that vary in relation to the software, hardware, and ML techniques deployed depending on the specific requirements for which one must optimize. Details on the tradeoffs for each configuration will be included as the operationalized models are developed below.

CVSA Operationalization: Facial Recognition, Age & Gender Detection, and Pose Estimation

In this paper, three different use cases are operationalized to demonstrate the effectiveness of the solutions architecture: 1) Facial Recognition, 2) Age & Gender Detection, and 3) Pose Estimation (a CV technique to identify body language through detecting the positions of key body joints in an image).

Different motion picture files are used as input, with the simultaneous streaming of multiple movie trailers used as an analog to emulate a multi-device surveillance system (Exhibit 2). To demonstrate the effects of different combinations of hardware and software on performance, three different model configurations (Exhibit 3) are deployed and subjected to three different test cases (Exhibit 4).

EXHIBIT 2 | DATA INPUTS AND SOURCE FORMATS FOR TEST CASES

Movie Trailer	Length	Size	Computer Vision Usage
Troy	1 minute 28 seconds	16.1 MB	Facial Recognition
Iron Lady	2 minutes 19 seconds	12.9 MB	Gender and Age Detection
Kramer vs. Kramer	2 minute 32 seconds	11.9 MB	Gender and Age Detection
Webcam	1 minute	N/A	Real Time
Troy (Streaming Online)	1 minute 30 seconds	N/A	Streaming Real-Time

EXHIBIT 3 | CVSA CONFIGURATIONS

Configuration	Definition
Intel OpenVINO	Intel's Open Visual Inference & Neural Network Optimization (OpenVINO) is an open-source product that utilizes Convolutional Deep Neural Networks to enable and enhance deep learning and computer vision workloads. It supports heterogeneous execution across computer vision hardware accelerators (CPUs, GPUs, FPGAs, VPUs,) and maximizes performance for deep-learning workloads.
Multi-Threading	A method of program execution that reduces the overhead of the operating system by sharing memory across threads (units of program execution) resulting in a faster execution, thereby maximizing computer resources through application programming techniques.
Standard - No Acceleration	Using computing resources without applying performance optimization at the code level nor at the hardware level. Execution can be performed either on-premise or with a large cloud platform provider such as Google, Amazon, and Microsoft which offer access to their respective computer vision APIs to create specialized processes/applications.

EXHIBIT 4 | CVSA TEST CASES

Test Case	Definition
Single File Processing	Age and Gender Detection for each of the three configurations.
Multiple Model Processing	Simultaneous processing of combinations of use case models (Facial Recognition, Age & Gender Detection, Pose Estimation) for each of three configurations.
Multiple File Processing	Simultaneous processing of multiple files with the same use case model (Age & Gender Detection) for each of the three configurations.

For each configuration and test case, the host system is an on-premise Linux OS. However, clients have flexibility in choosing whether to use their own on-premise system or a major cloud service provider (CSP) such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure.

CVSA Model Use Case Breakdown

1. Data Ingestion

The data ingestion phase consists of configuring a predetermined data set(s) to serve as input(s) to the CV model. These data sets, in the form of digital images or video files, can be collected from various sources and include both static files as well as real-time streaming from surveillance devices, aerial/satellite systems, or edge/IoT devices.[1] For the purposes of this paper, the data that will be ingested are from the data inputs show in exhibit 2.

2. Data Preprocessing / Multiprocessing

Data preprocessing is the process of collecting training data, cleaning data, and then preparing data to load into a model.

Collecting Training Data

Before a model can be trained, training data sets must first be acquired. Such data sets can be sourced from a variety of places or created from scratch using a developer's own image repositories. Google Image API was utilized for the tests conducted in this solutions architecture because of the ease of image retrieval from Google's image repository. For this proof of concept, images of actors were automatically collected and stored in a designated file system. These actors correspond to those who appeared in the movies being tested. Alternatively, in a real-world surveillance example, a repository of permitted personnel and/or known bad actors can be used as a training set to recognize persons of interest.

Input Selection for Modeling

After the training data is collected, it is inspected for appropriate actor images to be used as model inputs. For the purposes of this study, any images containing multiple persons are deleted from the input collection. The selected images are then called from the program (using Python or C++), realign for facial landmark localization, and resized to fit the requirements of the CNN (Frame x 750 pixels - width) used. Others may use any of a variety of algorithms for realignment and can set up a separate file server or a cloud instance for storing the input and output result files.

3. Multiprocessing and Multithreading Configuration

Multithreading and multiprocessing enable efficient loading and resource distribution by integrating streaming data with historical data to make real-time predictions. Multithreading is a method of program execution that reduces the overhead of the operating system by sharing memory across threads (units of program execution) to enable concurrent application or task execution. Generally, this process differs from multiprocessing, which can be defined as either the use of multiple CPUs within a system or the ability of a system to support more than one processor and allocate tasks between them. However, these definitions can vary in relation to how a CPU is defined.

Both multithreading and multiprocessing are deployed in the model used with the addition of Message Passing Interface (MPI) – a program that allows users to specify their own multiprocessing steps. By using MPI function calls, multithreading and multiprocessing were applied to capture movie frames in multiple video files which enabled the effective use of computing resources.

4. Transfer Learning & Modeling Process

Transfer learning refers to a process whereby previously trained models are applied to new datasets. In addition to reducing the training data requirements, transfer learning provides time-savings and more efficient options for modeling.

Some deep learning platforms like TensorFlow and Caffe make it easy to implement transfer learning out of the box by offering common pre-trained models for computer vision including ResNet, FaceNet, and SqueezeNet. Additionally, companies like Intel provide pretrained, downloadable models for different deep learning frameworks such as Caffe, TensorFlow, MxNet, and Open Neural Network Exchange (ONYX) via Model Zoo. Intel also supports multiple image classification networks such as AlexNet, GoogLeNet, VGG and ResNet families of networks, fully convolutional networks like FCN8 used for image segmentation, and object detection networks like Faster R-CNN.

The facial recognition, pose estimation, Gender & Age Detection pre-trained models (ResNet & FaceNet) were leveraged and optimized using Intel's OpenVINO software. These optimized models were further customized through transfer learning for the proof of concept.

[1] Edge detection that requires matching to a reference or structure from motion, may call for specific hardware and computational resources and domain expertise to understand how to implement ML and DL algorithms due to insufficient quantity of training data.

5. OpenVINO Transformation & Configuration for the Model Optimizer

For many years DL algorithms for CV were impractical due to their computationally intensive nature and the hardware limitations of the time. More recently, specialized hardware has been introduced designed specifically for CV applications like Vision Processing Units (VPUs), Graphics Processing Units (GPUs), and Field Programmable Gate Arrays (FPGAs) which dramatically improve the performance of such algorithms by utilizing a concept known as parallel processing, or the simultaneous execution of tasks on multiple processors. Even with specialized hardware in place, it is critical to optimize the implementation of your model on your physical processors to maximize modeling performance and avoid processing bottlenecks.

Our approach for the implementation of CNNs utilizes Intel's OpenVINO software to maximize performance by automatically optimizing around hardware and compute parameters, including management of software dependencies, for CPU, GPU, and VPU execution. OpenVINO achieved performance improvements by reviewing how each CNN layer is represented in the memory, computing the redundant layer processing cost, and removing redundancies. The resulting model is better at utilizing allocated resources.

Using Model Optimizer to Generate Intermediate Representation

The Model Optimizer receives input models from TensorFlow, Caffe, etc. and outputs the Intermediate Representation (IR) that is used as an input to the Inference Engine (IE). In standard data science modeling techniques, a trained model is directly used for inference without being optimized for performance. OpenVINO Model Optimizer, on the other hand, transforms the model into IR where the code is transformed into .XML and .Bin files. The XML file contains the network architecture; the binary file contains the weights and biases. The Model Optimizer removes parts of the network that are not required for inference and reshuffles the neural network operations for efficiency. In this process, some operations are reduced into a single operation.

6. Inference Engine / Parallel Processing in SMT

Inference Engine (IE) is a run-time API software layer that handles the execution of an optimized model on a target Intel hardware platform. IE optimizes the model by making a blob representation of the model in the memory. IE handles the execution of optimized models on selected hardware.

Parallel Processing was implemented using Message Passing Interface (MPI). MPI is part of parallel programming, is standardized and is designed to work on multiple platforms and multiple architectures (common ones being High-Performance Computing such as Cray systems, IBM, SGI, Cloud and other clusters). It can also be implemented in various programming languages (common ones being C++, Python and Java).

As the job becomes bigger, MPI effectively distributes the running job or process across the cores. Because of this, MPI becomes more efficient with less overhead compared to shared memory approaches like OpenMPI (also another parallel programming approach). MPI is not as effective on small workloads.

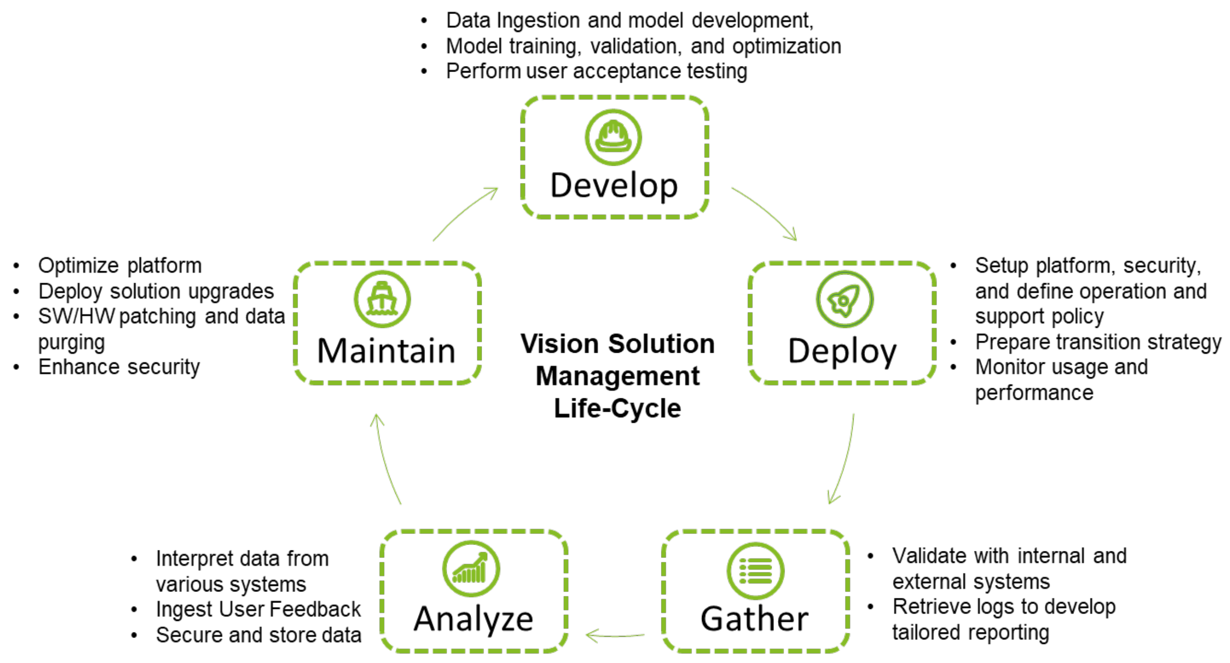
We implemented MPI to distribute the workload consisting of multiple video files using function calls of Scatter, Allgather, and Gather. OpenVINO demonstrated effective multiprocessing of the files using python libraries like the python multiprocessing module with Pool.

MPI was also utilized to process multiple input sources. Live input like webcam/camera and video streaming is broadcasted and displayed on an assigned root process while video files are processed in parallel and remaining processes are in the MPI communicator.

7. Application Deployment and Analytics

Deploying a CV solution requires a solid understanding of lifecycle management requirements as shown in Exhibit 5. A key factor for successful application deployment is to identify stakeholders and users who will be supporting the system. The platform needs to be architected for security and scalability, and the implementation can be sequenced to enable future-state business processes, all contributing to a successful CV application. It is suggested that having a phased deployment approach to deliver a solution in an orderly and incremental manner, without impacting current business activities, will result in a low-risk solution implementation. Having a solution support team who are well trained and well versed in analytics will likely contribute to high user satisfaction and opportunities for feedback and ongoing improvements.

EXHIBIT 5 | Management Lifecycle


• Data Ingestion and model development,
• Model training, validation, and optimization
• Perform user acceptance testing
Develop
• Optimize platform
• Deploy solution upgrades
• SW/HW patching and data purging
• Enhance security
Maintain
Vision Solution Management Life-Cycle
Deploy
• Setup platform, security, and define operation and support policy
• Prepare transition strategy
• Monitor usage and performance
• Interpret data from various systems
• Ingest User Feedback
• Secure and store data
Analyze
Gather
• Validate with internal and external systems
• Retrieve logs to develop tailored reporting

CVSA Deployment Considerations

When deploying a computer vision solution, it is important to understand there are other factors that must be considered outside of the technical choices, such as what algorithms to run or which sets of data to use. These factors impact the business and must be considered for a successful deployment. Several of these factors that implement the solution are described in Exhibit 6.

EXHIBIT 6 | Considerations for Vision Analytics Deployment

Criteria	Key Considerations
Business strategic goals	Establish regular touchpoints with stakeholders to align the solution to business needs. The refactor model needs to be dynamic enough so that it can be adjusted to the changing business goals and needs without requiring new development from scratch. The solution should be flexible enough to work with available resources to the business and offer more insight for future investment if desired by the business.
Readiness of certain end-user groups and training data.	The performance and accuracy of CV model predictions largely depend on the training data. The training data must be representative of most, if not all the case scenarios and structures of the data. It requires decisions on what is the right data to collect, how to filter and process the data before inputting it in the model, whether to use pre-trained models, and how to adjust these models to better fit your data and needs.
Readiness and availability of user groups for design, testing and validation	Avoid underfitting, where the model has very low accuracy, which is normally caused by a lack of training data or poor selection of hyperparameters. Avoid overfitting where the model is only memorizing with nearly perfect prediction, which could be misleading in future uses. This is usually caused by the mixing of training data and validation data.
Analytics and ongoing operations	The model's performance and accuracy should be able to improve over time and such metrics should be captured. Variations between training and validation data should be constantly measured for continuous adjustments. Use analytics to monitor trends and apply machine learning for predictive analytics.

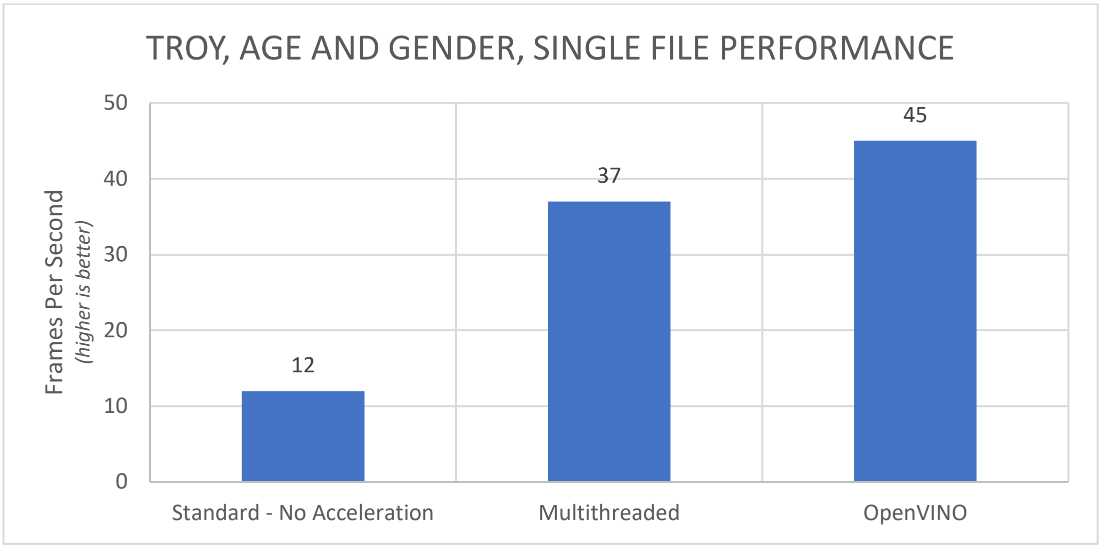
Results and Analysis

The following results show deep learning model implementations of CV using the OpenVINO™ toolkit and other comparison methods on CPU. This use case is focused and evaluated on the hardware optimization of DL model processing, not enhancing the accuracy of the DL models.

Test Case #1: Single file processing results for Age & Gender Detection Model on CPU

We ran the Age & Gender Detection model on the promotional movie trailer of Troy in three configurations: OpenVINO™, Multithreading, and Other (standard, no hardware acceleration). The OpenVINO™ configuration captured more frames per second (FPS) than the other two methods with a faster run time. Exhibit 7 illustrates OpenVINO's performance advantage in single file processing operations.

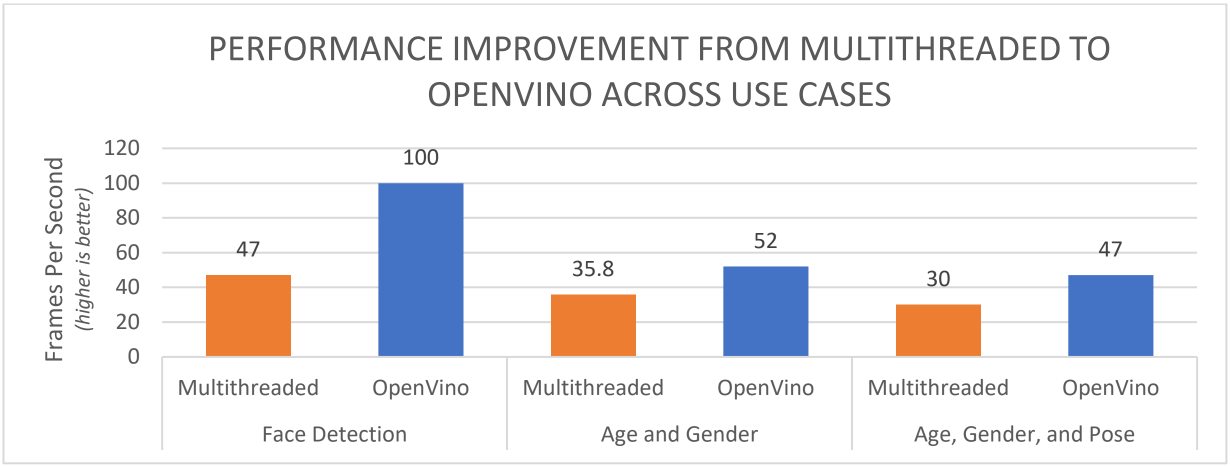
EXHIBIT 7 | TROY, AGE AND GENDER, SINGLE FILE PERFORMANCE



Test Case #2: Multiple Model Processing on CPU

The results of simultaneously running different combinations of multiple models (Facial Recognition only, Age & Gender Detection, and Age & Gender Detection plus Pose Estimation) for the OpenVino and Multithreading configurations demonstrate that using a toolkit like OpenVINO can enhance vision processing (See Exhibit 8.).

EXHIBIT 8 | PERFORMANCE IMPROVEMENT FROM MULTITHREADED TO OPENVINO ACROSS USE CASES

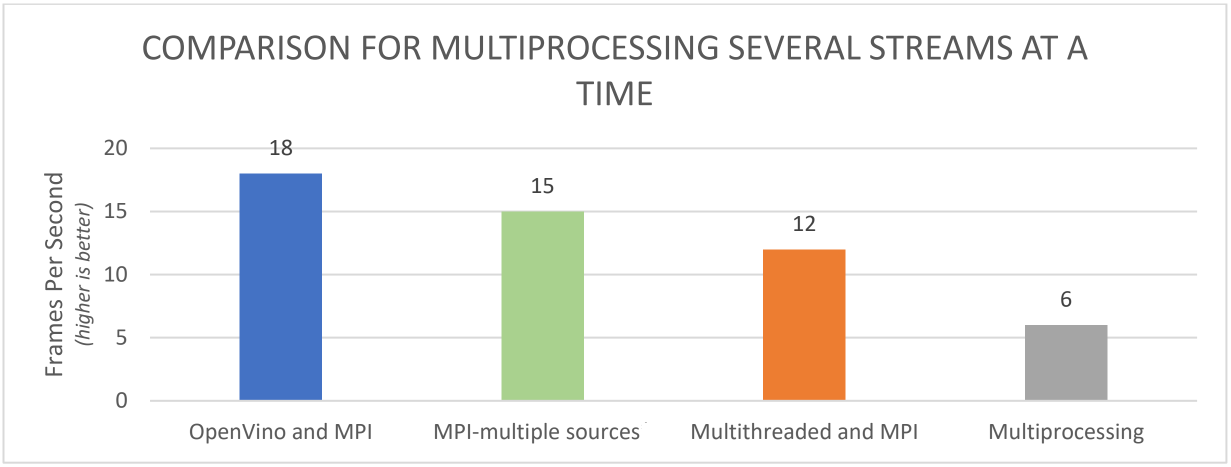


The number of models applied simultaneously influenced the number of frames captured. Comparing the two configurations, OpenVINO' s performance surpassed the Multithreading by capturing more frames per seconds (FPS) for all models.

Test Case #3: Multiple Files Processing with Age & Gender Detection on CPU

The following table shows the results of running the Age & Gender Detection model on multiple input files concurrently. The three input videos are trailers for the movies Troy, Iron Lady, Kramer vs. Kramer. Results are compared below with different approaches: OpenVINO™ and MPI, MPI, Multithreading and MPI, and Multiprocessing without OpenVINO/Multithreading. This test was conducted with multiprocessing one online video of Troy and one webcam streaming.

EXHIBIT 9 | COMPARISON FOR MULTIPROCESSING SEVERAL STREAMS AT A TIME



OpenVINO running 3 processes simultaneously captured more FPS as shown above. OpenVINO was also tested in heterogeneous environments: CPU, and CPU/VPU. Both environments used face detection, age and gender, and pose estimation models. For this trial, the hardware performance showed varied results based on the hardware assignment of each combination of models.

Conclusion

The Computer Vision Solutions Architecture mitigates some of the typical machine learning challenges through utilizing Intel's OpenVINO software, which both accelerated development time due to its library of pre-trained models and optimized performance hardware and compute parameters, including management of software dependencies, for CPU, GPU, and VPU execution. OpenVINO demonstrated superior performance in Frames per Second across the three different use cases utilized for benchmarking: 1) Facial Recognition, 2) Age & Gender Detection, and 3) Pose Estimation. Each phase of the architecture can be tailored to meet the client's needs and business process as organizations advanced towards advanced analytics.

This architecture describes an approach that was designed to accelerate the development of Computer Vision based solutions. It is believed that the use of computer vision will continue to grow as private and public institutions increasingly rely on vision solutions to automatically evaluate images and videos (sometimes in real-time) in the support of operations, customer services, or public safety. As the number of devices proliferates and size and quality of image/video feeds increase, having an optimized high-performance architecture can potentially help organizations scale faster, make quicker decisions, and more effectively actualize an organization's mission.

Let's Talk

Reach out to our team to request a demo and learn more about the computer vision solution architecture and how it can help you transform your organization.

Contacts:

Vishal Kapur
Principal
Deloitte Consulting LLP
vkapur@deloitte.com
+1.571.814.7510

Doug Bourgeois
Managing Director
Deloitte Consulting LLP
dbourgeois@deloitte.com
+1.571.814.7157

Appendix A: Hardware, Software, Computer Vision and Deep Learning Configuration for Use Cases:

The following table demonstrates the hardware and software used in the testing and development of this solutions architecture.

Type	Component	Product / Version	Description
Hardware	CPU	Linux OS - Lambda Laptop	A laptop designed for running deep learning models Intel i7-8750H (6 Cores, 2.20GHz) Ubuntu 16.04 LTS
Hardware	Visual Processing Unit (VPU)	Intel Movidius Compute Stick 2	The Myriad X VPU is a high-speed and power-efficient solution that brings advanced vision and artificial intelligence applications to devices Class-leading performance in computer vision and deep neural network inferencing applications
Software	OpenVINO™ Toolkit	2019	OpenVINO™ ™ toolkit, short for Open Visual Inference and Neural Network Optimization toolkit, provides developers with improved neural network performance on a variety of Intel® processors and helps them further unlock cost-effective, real-time vision applications. The toolkit enables deep learning inference and easy heterogeneous execution across multiple Intel® platforms (CPU, Intel® Processor Graphics)—providing implementations across cloud architectures to edge devices.
Software	Visual Studio	2019	Microsoft Visual Studio is an integrated development environment from Microsoft that is used to build applications. MS build is included in the configuration for OpenVINO™ and is a free and open-source build toolset for managed code such as native C++ code
Software	CMake	3.4	CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files and generate native makefiles and workspaces that can be used in the compiler environment. (source: https://cmake.org/)
Language	Python	3.6.5	Python is an open-source general-purpose programing language. The 3.6.5 version is compatible with the OpenVINO™ toolkit.
Language	C++		C++ is an open-source general-purpose programming language that was developed by Bjarne Stroustrup and offered as part of the OpenVINO™ use cases.
Programming Library	OpenCV		OpenCV is an open-source library of programming functions developed by Intel that aims at real-time computer vision using deep learning frameworks such as TensorFlow, PyTorch, and Caffe.
Programming Process and Library	MPICH and MPI4py		Multi_scatter and MPI_Allgather Packages needed to run MPI MPICH: https://www.mpich.org/downloads/ Python MPI4py: https://mpi4py.readthedocs.io/en/stable/