DEMOCRATIC PEOPOLE'S REPUBLIC OF ALGERIA

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

UNIVERSITY OF LAGHOUAT AMAR TELIDJI

FACULTY OF TECHNOLOGY

DEPARTEMENT OF ELECTRONIC

MASTER DISSERTATION

By:

HORMA TAHER

ZEMMIT MUSTAPHA

DOMAIN: Sciences and Technics

FIELD: Electronic

OPTION: Embedded Systems

Theme

Design and Implementation of Smart Vision System for Detection of Pedestrian on Jetson Nano Development Board

Jury members:

Name	Grade	Quality
Mr. KIOUS Mecheri	Pr	President
Mr. REGUIGUE Mourad	MCB	Reviewer
Mr. SEHAIRI Kamal	MCB	Supervisor

Promotion: 2019-2020

ACKNOWLEDGMENT

Thanks and praises be to Allah the Almighty who gave us health strength patience and willpower to complete this thesis.

We are truly indebted and thankful to our supervisor Dr. Kamal Sehairi who accepted to lead this work and for his encouragement, supervising and supporting us to understand and completing this thesis from the preliminary to the concluding level.

We would also like to thank our professors and all the teachers whose contributed to our training and the administrators and all the members of the jury for honoring us by accepting to judge our thesis and help reviewing and improving it.

Finally, we would like to express our gratitude to our families and friends who have welcomed and supported us during our etude.

To those that we have unintentionally forgotten and who participated in any way in the preparation of this work, find here the expression of our gratitude.

This work is supported by Nvidia Corporation, for providing the Jetson Nano and letting us access to teaching materials and white papers.

DEDICATION

Thank you Allah for giving me the ability to write and to think, the strength to believe in it, the patience to go through with the dream and the happiness to raise my hands to the sky and say "Ya Kayoum"

I dedicate this modest work to the one who gave me life, the symbol of tenderness, who sacrificed herself for my happiness and my success, to my dear

♥ MOTHER ♥

To my FATHER, the school of my childhood, the one who was my shadow throughout all my years of study, and who made sure to encourage me throughout all my life, which give me help and protect me.

May God protect them and bless them.

To my brothers and my sisters, (Yacine, Khedidja, Fatima Zahraa, Ilham Rokia)

To all my family.

To my best-friends: (Ayoub, Taher, Mohammed, Islam, Abdeldjalil, Azzou, Toufik, Belkacem, Alilo, Mustapha)

To the fragrant flower and my dear best friend Benallou.Amel

To all those I love and whose love me.

To all my teachers from the primary school to the university level

To all, I say Thank you.

Zemmit Mustapha

DEDICATION

To the fragrance of heaven and the source of tenderness, to the light that shines my life, to my beloved ♥ Mother ♥.

To the one who taught me the meaning of life and my solid bond to the owner of the big heart, my dear ♥ Father ♥.

To the owner of the warm embrace and the quiet heart, my beloved ♥Grandmother ♥.

To my dear brothers (Ali, Bilal)

To all my family and my best friends

To all my teachers from the primary school to the university level

To all, I say Thank you.

Horma Taher

CONTENTS

CONTENTS

CONTENTS

CONTENTS

LIST OF FIGURES

LIST OF FIGURES

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
CAFFE	Convolutional Architecture for Fast Feature Embedding
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CoM	Computer on Module
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
cuDNN	CUDA Deep Neural Network Library
CVAT	Computer Vision Annotation Tool
DL	Deep Learning
DNN	Deep Neural Network
DSPs	Digital Signal Processors
EULA	End User License Agreement
FPGAs	Faithful-programmable gate arrays
GLEW	OpenGL Extension Wrangler
GPUs	Graphics Processing Units
GTC	GPU Technology Conference
I/O	Input/Output
ILSVRC	the ImageNet Large Scale Visual Recognition Challenge
IoT	Internet of things
IOU	Intersection over Union
mAP	mean Average Precision
ML	Machine Learning
NLP	Natural Language Processing
NVRs	Network Video Recorders
OS	Operating System
PoP	Package-On-A-Package

ABBREVIATIONS AND ACRONYMS

R-CNNs	Region-Based Convolutional Neural Networks
ReLU	Rectified Linear Unit
RNNs	Recurrent neural networks
RPN	Region Proposal Network
SBCs	Single Board Computers
SiP	System-In-A-Package
SoC	System-On-A-Chip
SoM	System-On-Module
SSD	Single Shot Detector
SVM	Support Vector Machine
YOLO	You Only Look Once

Introduction

Introduction

Computer vision systems have undergone a great development in the field of artificial intelligence in recent years. One of the aspects to consider for this growth has been to not limit itself only to niches as robotics and manufacturing, but also to other areas such as home automation, intelligent detection, medical image analysis, autonomous driving …

Object detection, one of the most fundamental and challenging problems in computer vision, which aims to detect and position objects in the images such as traffic signs, vehicles, buildings, and people…etc. Nowadays some dedicated embedded systems have emerged as a powerful strategy for deliver high processing capabilities including the NVIDIA Jetson family.

In this thesis, we will try to design and implemented a smart vision system for detection of objects and pedestrians through an embedded system. For this purpose, a low power embedded Graphics Processing Unit (Jetson Nano) has been selected, which allows multiple neural networks to be run in simultaneous and a computer vision algorithm and OpenCV library to be applied for image recognition. As well, the performance of these deep learning neural networks such as ssd-mobilenet and YOLO v3 has been tested.

This thesis is organized as follows: in chapter I, we give general definitions of artificial intelligence and Machine learning, we present two techniques of machine learning (supervised & unsupervised) and ML steps, We also have defined deep learning with mentioning the stages of the process, network algorithms, types of training, and the most important libraries and models used for training, we providing a simplified definition of computer vision and object detection and some of its techniques, then we enumerate different datasets used object detection.

In chapter II, we present the materials and methods used in this study, starting by defining the embedded and the internet of things applications, then we define GPUs and a simplified comparison between CPUs and GPUs and we give some embedded acronyms, then we show the different characteristics of our development board the Jetson Nano, we detailed also step by step how to configure and install the jetpack SDK, and OpenCV with CUDA supports on this board.

In the last chapter III, we we'll discuss Single Shot Detectors and MobileNets and how to combining them together. And then a simple define to the used datasets and the Caffe model. We present also the code used for training this algorithm and deployed it on our development board the Jetson Nano and show results.

Chapter I:
Object Detection

I.1 Introduction

One of God's greatest blessings on humans is the fast and the complex visual system that can perform complex tasks such as identifying multiple objects and discovering obstacles with little conscious thinking. Nowadays, with the large amounts of data, faster GPUs, better algorithms, and the massive development in the field of artificial intelligence, Machine learning and Deep Learning, we can now easily train computers to detect, identify and classify multiple objects within an image with high accuracy.

So what is object detection and how we can do that?

I.2 Artificial intelligence

Artificial intelligence (AI) is a branch of computer science, it represents the behavior and specific characteristics of computer programs that make them mimic human mental capabilities and working patterns. Among the most important of these characteristics is the ability to learn, infer and react to situations not programmed in the machine. AI systems rely on learning algorithms, such as machine learning and deep learning, along with large of datasets well-labeled.

John McCarthy at the Massachusetts Institute of Technology coined the term in 1956, and he defined it as "the science and engineering of making smart machines". [1]

Andreas Kaplan and Michael Heinleen define artificial intelligence as "the ability of a system to properly interpret external data, learn from this data, and use that knowledge to achieve specific goals and tasks through flexible adaptation." [2]

I.3 Machine learning

Machine learning teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to "learn" information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases. [3]

We use **machine learning** as shorthand for "**traditional machine learning**"—the workflow in which we manually **select features** and then **train the model**. When we refer to machine learning we exclude deep learning. Common machine learning techniques include decision trees, support vector machines, and ensemble methods. [4]

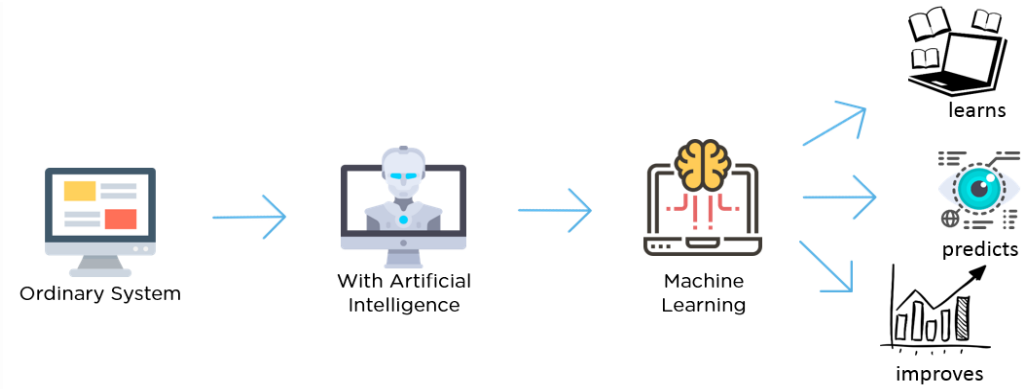

Figure I-1: Importance of Machine Learning

I.3.1 Real world applications of machine learning

With the rise in big data, machine learning has become particularly important for solving problems in areas like these: [3]

- Computational finance, for credit scoring and algorithmic trading
- Image processing and computer vision, for face recognition, motion detection, and object detection
- Computational biology, for tumor detection, drug discovery, and DNA sequencing
- Energy production, for price and load forecasting
- Automotive, aerospace, and manufacturing, for predictive maintenance
- Natural language processing

Figure I-2: Applications of Machine Learning

I.3.2 Techniques of machine learning

Machine learning uses two types of techniques: supervised learning, which trains a model on known input and output data (labeled data) so that it can predict future outputs, and unsupervised learning, which finds hidden patterns or intrinsic structures in input data (unlabeled data). We can find also a third technique, called semi-supervised methods. In this technique, we do training on a small-labeled data with a large unlabeled data. [3]

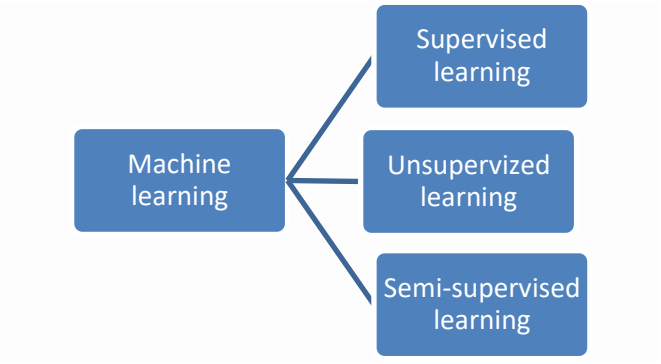

Figure I-3: Machine learning techniques

I.3.2.1 Supervised learning

The aim of supervised machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data. [3]

Supervised learning uses classification and regression techniques to develop predictive models.

- **Classification** techniques predict **discrete responses,** for example, whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring.
- **Regression** techniques predict **continuous responses**, for example, changes in temperature or fluctuations in power demand. Typical applications include electricity load forecasting and algorithmic trading.

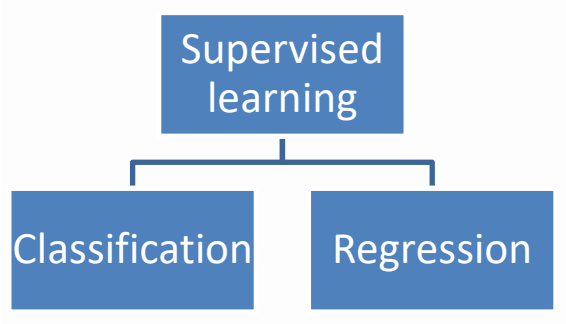

Figure I-4: Supervised learning techniques

I.3.2.1.1 Common classification algorithms

I.3.2.1.1.1 Logistic Regression

Logistic regression is named for the function used at the core of the method, the logistic function.

The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It’s an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits. [5]

$$1/\left(1+e^{-value}\right) \quad \text{(I.1)}$$

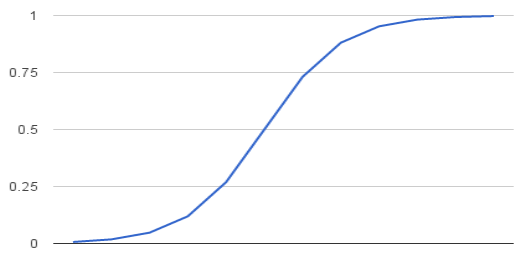

Figure I-5: Example of Logistic function

It is best used:

- When data can be clearly separated by a single, linear boundary
- As a baseline for evaluating more complex classification methods

I.3.2.1.1.2 *k*-Nearest *Neighbor (kNN)*

kNN categorizes objects based on the classes of their nearest neighbors in the dataset. kNN predictions assume that objects near each other are similar. Distance metrics, such as Euclidean, city block, cosine, and Chebychev, are used to find the nearest neighbor. [5]

It is best used:

- When we need a simple algorithm to establish benchmark-learning rules.
- When memory usage of the trained model is a lesser concern.
- When prediction speed of the trained model is a lesser concern.

I.3.2.1.1.3 *Support Vector Machine (SVM)*

A SVM is a discriminative classifier formally defined by a separating hyperplane. Given labeled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. [5]

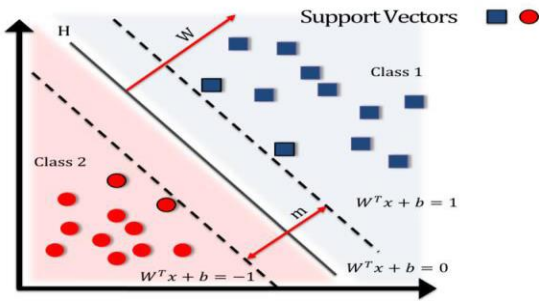

Figure I-6: classification scheme

It is best used:

- For data that has exactly two classes (we can also use it for multiclass classification with a technique called error-correcting output codes)
- For high-dimensional, nonlinearly separable data
- When we need a classifier that's simple, easy to interpret, and accurate

I.3.2.1.1.4 *Neural Network*

Inspired by the human brain, a neural network consists of highly connected networks of neurons that relate the inputs to the desired outputs. The network is trained by iteratively modifying the strengths of the connections so that given inputs map to the correct response. [5]

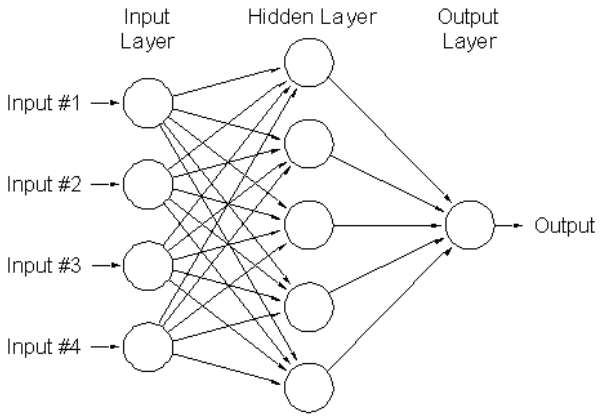

Figure I-7: The neural network is an interconnected network of functioning nodes

It is best used:

- For modeling highly nonlinear systems
- When data is available incrementally and we wish to constantly update the model
- When there could be unexpected changes in our input data
- When model interpretability is not a key concern

I.3.2.1.1.5 *Decision Tree*

A decision tree lets us predict responses to data by following the decisions in the tree from the root (beginning) down to a leaf node. A tree consists of branching conditions where the value of a predictor is compared to a trained weight. The number of branches and the values of weights are determined in the training process. Additional modification, or pruning, may be used to simplify the model. [5]

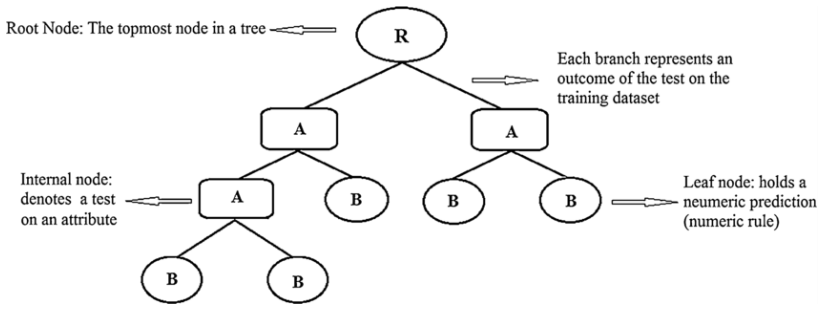

Figure I-8: decision tree scheme

It is best used:

- When we need an algorithm that is easy to interpret and fast to fit
- To minimize memory usage
- When high predictive accuracy is not a requirement

I.3.2.1.2 Common classification algorithms

I.3.2.1.2.1 *Linear Regression*

Linear regression is a statistical modeling technique used to describe a continuous response variable as a linear function of one or more predictor variables. Because linear regression models are simple to interpret and easy to train, they are often the first model to be fitted to a new dataset. [5]

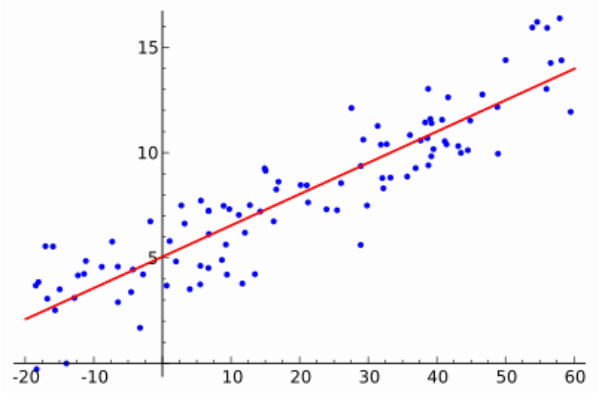

Figure I-9: Example of Linear Regression

It is best used:

- When we need an algorithm that is easy to interpret and fast to fit
- As a baseline for evaluating other, more complex, regression models

I.3.2.1.2.2 *Nonlinear Regression*

Nonlinear regression is a statistical modeling technique that helps describe nonlinear relationships in experimental data. Nonlinear regression models are generally assumed to be parametric, where the model is described as a nonlinear equation. [5]

"Nonlinear" refers to a fit function that is a nonlinear function of the parameters. For example, if the fitting parameters are b_0, b_1, and b_2:

The equation: $y = b0 + b1.x + b2.x^2$: is a linear function of the fitting parameters,

Whereas: $y = (b0.x^{b1})/(x + b2)$: is a nonlinear function of the fitting parameters.

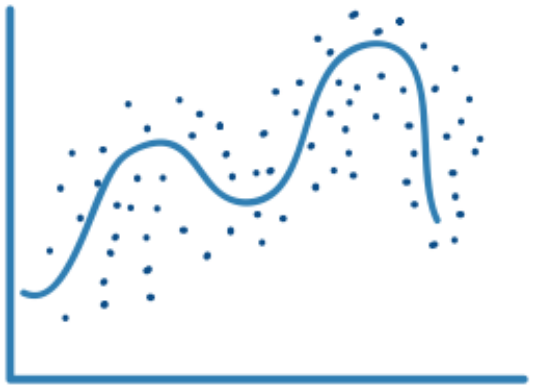

Figure I-10: Nonlinear regression scheme

It is best used:

- When data has strong nonlinear trends and cannot be easily transformed into a linear space
- For fitting custom models to data

I.3.2.1.2.3 *SVM Regression*

SVM regression algorithms work like SVM classification algorithms, but are modified to be able to predict a continuous response. Instead of finding a hyperplane that separates data, SVM regression algorithms find a model that deviates from the measured data by a value no greater than a small amount, with parameter values that are as small as possible (to minimize sensitivity to error). [5]

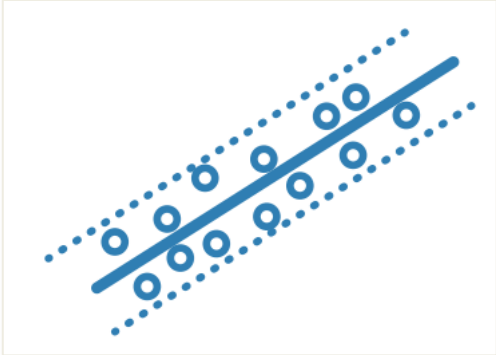

Figure I-11: SVM regression scheme

It is best used:

- For high-dimensional data (where there will be a large number of predictor variables)

I.3.2.2 Unsupervised learning

Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data **without labeled responses**. [6]

Clustering is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data.

Applications for clustering include gene sequence analysis, market research, and object recognition.

Figure I-12: example of Clustering technique [6]

I.3.2.2.1 Common Unsupervised learning Algorithms

I.3.2.2.1.1 *k-Means*

Partitions data into k number of mutually exclusive clusters. How well a point fits into a cluster is determined by the distance from that point to the cluster's center. [7]

It is best used:

- When the number of clusters is known
- For fast clustering of large data sets

I.3.2.2.1.2 *k-Medoids*

Similar to k-means, but with the requirement that the cluster centers coincide with points in the data. [7]

It is best used:

- When the number of clusters is known
- For fast clustering of categorical data
- To scale to large data sets

I.3.2.2.1.3 *Self-Organizing Map*

Neural-network based clustering that transforms a dataset into a topology-preserving 2D map. [7]

It is best used:

- To visualize high-dimensional data in 2D or 3D
- To deduce the dimensionality of data by preserving its topology (shape)

I.3.3 Machine learning steps

Feature selection: Identifying the most relevant features, or variables, that provides the best predictive power in modeling our data. This could mean adding variables to the model or removing variables that do not improve model performance.

Feature transformation: Turning existing features into new features using techniques such as principal component analysis, nonnegative matrix factorization, and factor analysis.

Hyperparameter tuning: The process of identifying the set of parameters that provides the best model. Hyperparameter control how a machine learning algorithm fits the model to the data. [3]

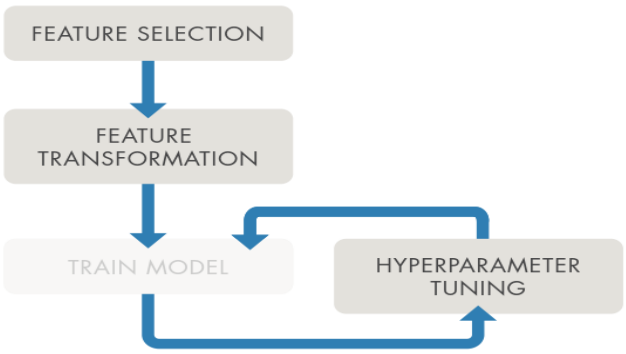

Figure I-13: Steps to Improve a Model

Selecting the Classification Algorithm: is a process of trial and error. It's also a trade-off between specific characteristics of the algorithms, such as:

- Speed of training
- Memory usage
- Predictive accuracy on new data
- Transparency or interpretability (how easily we can understand the reasons an algorithm makes its predictions)

Improving Models: means increasing its accuracy and predictive power and preventing overfitting (when the model cannot distinguish between data and noise). Model improvement involves feature engineering (select more relevant features, select the best classification method) and hyperparameter tuning.

I.4 Deep Learning

I.4.1 What is Deep Learning?

Deep learning is a type of machine learning in which a model learns to perform classification tasks directly from images, text, or sound. Deep learning is usually implemented using a neural network architecture. The term "deep" refers to the number of layers in the network—the more layers, the deeper in the network. Traditional neural networks or shallow neural networks contain only 2 or 3 layers, while deep networks can have hundreds. [8]

Deep learning algorithms are constructed with connected layers.

- The first layer is called the Input Layer
- The last layer is called the Output Layer
- All layers in between are called Hidden Layers. The word deep means the network join neurons in more than two layers.

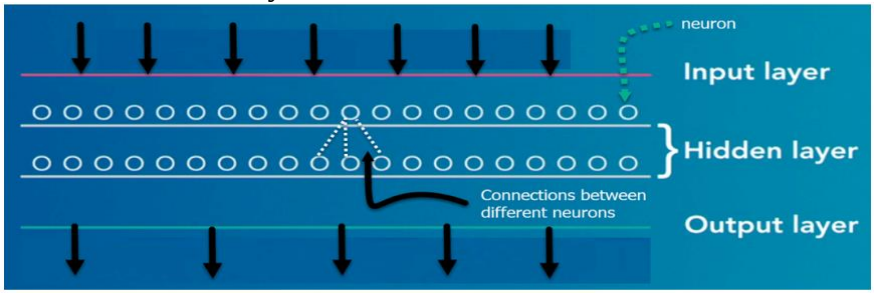

Figure I-14: layers of Deep learning algorithms sheme [8]

Each Hidden layer is composed of neurons. The neurons are connected to each other. The neuron will process and then propagate the input signal it receives the layer above it. The strength of the signal given the neuron in the next layer depends on **the weight, bias and activation function.**

I.4.1.1 Weights

Weights are the coefficients of the equation which we are trying to resolve. Negative weights reduce the value of an output. When a neural network is trained on the training set, it is initialized with a set of weights. These weights are then optimized during the training period and the optimum weights are produced. [9]

A neuron first computes the weighted sum of the inputs.

$$Y = \sum (weight * input) + bias \quad \textbf{(I.2)}$$

I.4.1.2 Bias

Bias is simply a constant value (or a constant vector) that is added to the product of inputs and weights. Bias is utilized to offset the result. It is used to shift the result of activation function towards the positive or negative side. [9]

The addition of bias reduces the variance and hence introduces flexibility and better generalization to the neural network.

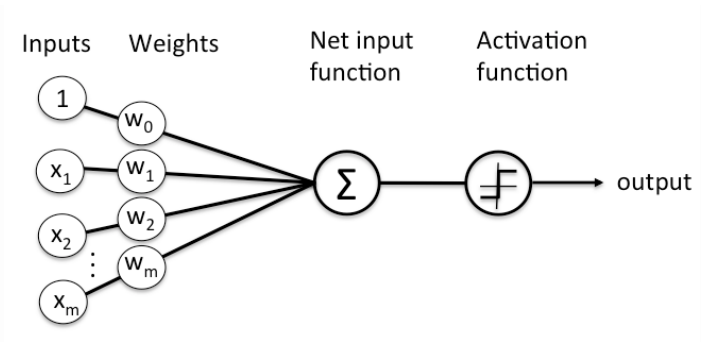

Figure I-15: Neural network architecture

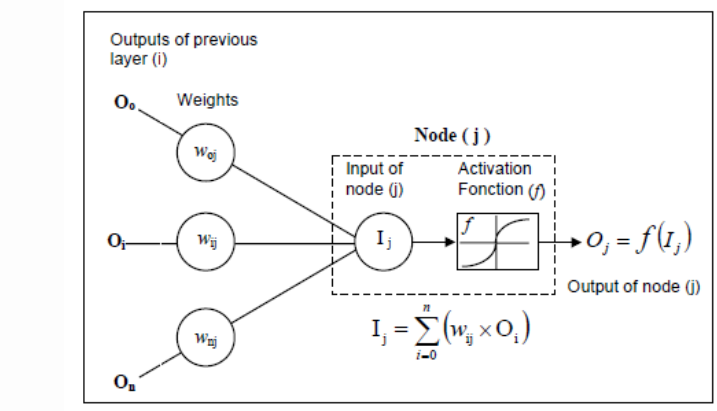

Figure I-16: Neural network architecture (2)

The network consumes large amounts of input data and operates them through multiple layers; the network can learn increasingly complex features of the data at each layer.

Deep learning can outperform traditional method. For instance, deep learning algorithms are 41% more accurate than machine learning algorithm in image classification, 27% more accurate in facial recognition and 25% in voice recognition.

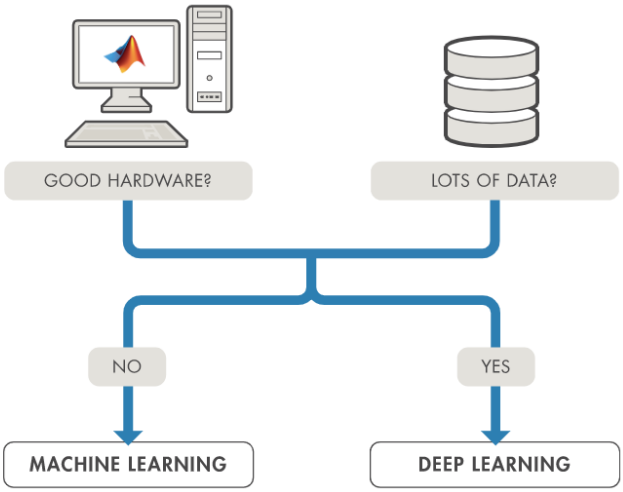

Figure I-17: Key factors for choosing between deep learning and machine learning

I.4.2 Deep learning Process

A deep neural network provides state-of-the-art accuracy in many tasks, from object detection to speech recognition. They can learn automatically, without predefined knowledge explicitly coded by the programmers. [10]

Figure I-18: Deep Learning Process [10]

In neural network each layer represents a deeper level of knowledge, i.e., the hierarchy of knowledge. A neural network with four layers will learn more complex feature than with that with two layers.

The learning occurs in two phases.

- The first phase consists of applying a nonlinear transformation of the input and creates a statistical model as output.
- The second phase aims at improving the model with a mathematical method known as derivative.

The neural network repeats these two phases hundreds to thousands of time until it has reached a tolerable level of accuracy. The repeat of this two-phase is called iteration. [10]

The deep neural network development passes by several steps:

I.4.2.1 Data collection

Most of Deep Learning applications will require a lot of data which need to be labeled. Time will be mostly consumed in this process. Depending upon the domain of our problem, we may either use standard datasets collected by others, but maybe that labeled dataset needed for our project is not available publicly, so we should starting collecting our own data. [11]

I.4.2.2 Data Labeling

Generating labeled training data requires a great deal of time, effort, and investment. If we are building a machine learning model, chances are we're going to need data labeling tools to quickly put together datasets and ensure high-quality data production. [11]

Data labeling tools vary in the features they offer, file types they support, data security practices, storage options, and more. There are many labeling tools available on the web; we can cite some of them:

I.4.2.2.1 Lionbridge AI

Lionbridge AI offers an end-to-end data labeling and annotation platform for data scientists looking to train machine learning models. With over 20 years of hands-on experience creating custom data for the world's largest technology companies, Lionbridge AI has built the most intuitive data annotation platform on the market. [12]

This all-in-one platform allows us to build custom training datasets quickly and cost effectively while maintaining data quality. Furthermore, the tool works for all major file types, with unique features to handle text, audio, image & video data.

Figure I-19: Lionbridge AI Interface

The platform gives us maximum control and flexibility to customize our task, workflow and quality checks. Furthermore, we are also given the option to invite our own annotators onto the platform, or hire from Lionbridge's network of over 500,000 qualified contributors.

I.4.2.2.2 Amazon Mechanical Turk

Also known as MTurk, Amazon Mechanical Turk is a popular crowdsourcing marketplace commonly used for data labeling. As a requester on Amazon Mechanical Turk, we can design, publish, and coordinate a wide range of human intelligence tasks (known as HITs), such as text classification, transcriptions, or surveys. The MTurk platform provides useful tools to describe our task, specify consensus rules, and define the amount we are willing to spend for each item. [12]

I.4.2.2.3 Computer Vision Annotation Tool (CVAT)

The Computer Vision Annotation Tool (CVAT) is a web-based tool for annotating digital images and videos. The tool supports tasks like object detection, image segmentation and image classification. Although the tool itself requires some time to learn and master, CVAT boasts a wide range of features for labeling computer vision data. [12]

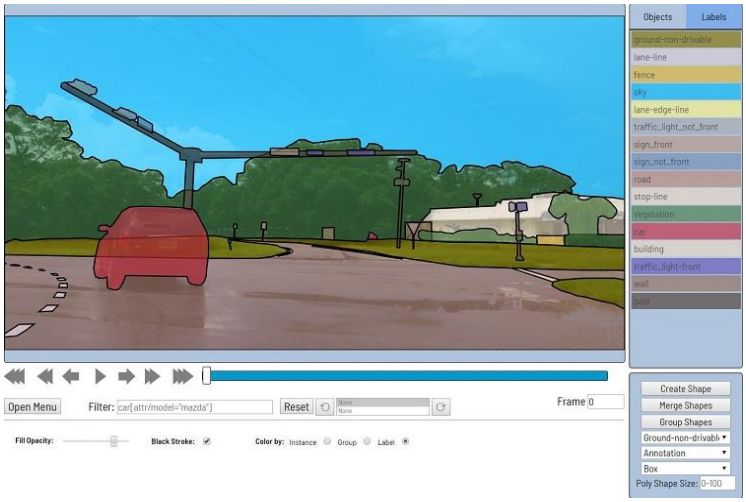

Figure I-20: Computer Vision Annotation Tool (CVAT) Interface

I.4.2.2.4 LabelIMG

LabelIMG is an open source image labeling tool that has pre-built binaries for Windows so it's extremely easy to install. LabelIMG supports only bounding boxes. The format is PascalVoc XML and annotation files are saved separately for each image in the source folder. [13]

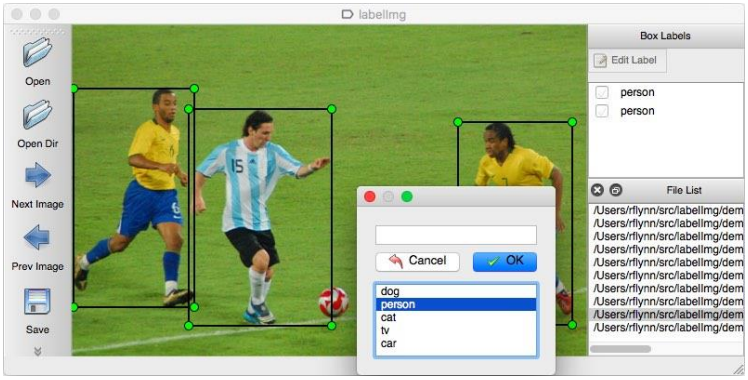

Figure I-21: LabelIMG Interface

I.4.2.2.5 VGG Image Annotator

VGG is an open-source tool that, just like LabelIMG, can do an amazing job for straightforward tasks that do not require project management. It is available as an online interface and can also be used offline as an HTML file. In its most recent version, it also offers a wide variety of video labeling tools.

VGG Image Annotator offers a lot more tools, including dots, lines, polygons, circles and ellipses. Also has the option of adding object and image attributes/tags. The annotations can be downloaded as one JSON file containing all annotations, or as one CSV file, and can be uploaded afterwards if there is a need to review them. [13]

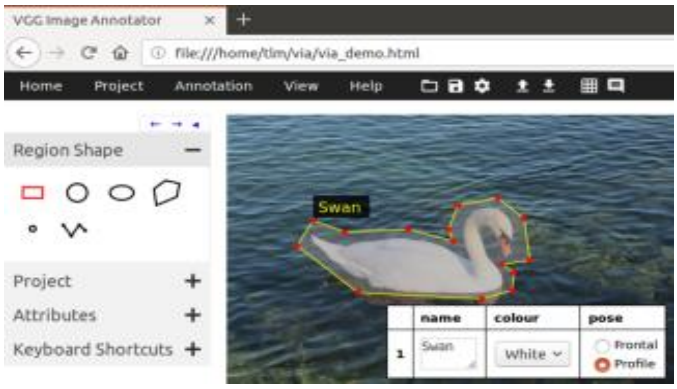

Figure I-22: VGG Image Annotator Interface

I.4.2.2.6 Supervisely

Supervisely is an awesome web-based platform that offers an advanced annotation interface but also covers the entire process of computer vision training, including a deep learning models library that can be directly trained, tested, and improved within the platform.

Supervisely has a great array of tools, including dots, lines, boxes, polygons, and a bitmap brush for semantic segmentation. Also includes the possibility to draw holes in polygons, which has been incredibly valuable. Another very useful feature is the option to add image and object tags and to order Figures in layers. Output is in JSON files for each image or PNG masks and the platform also allows us to upload formats such as Cityscapes and COCO. In addition, there is an option to do data transformation directly on the platform. [13]

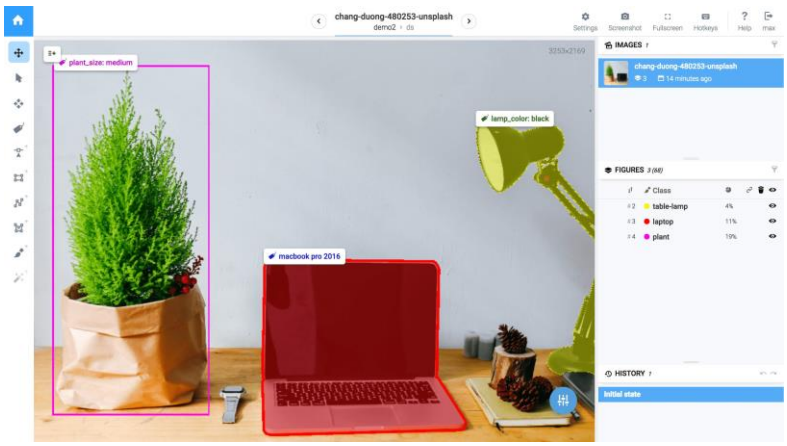

Figure I-23: Supervisely Interface

I.4.2.2.7 Labelbox

Labelbox is another great web-based platform that launched in early 2018 and ever since then has been constantly updating and improving its functionalities. It also offers the possibility to integrate a human-in-the-loop by importing model predictions and seeing the consensus between the labelers and the model.

Labelbox offers a complete array of tools for annotation, such as points, lines, boxes and polygons, and has recently added an awesome new feature for their semantic segmentation brush—a super pixel coloring option that makes life so much easier when boundaries are clear. Output is as one JSON or CSV file containing all annotations or as PNG masks. [14]

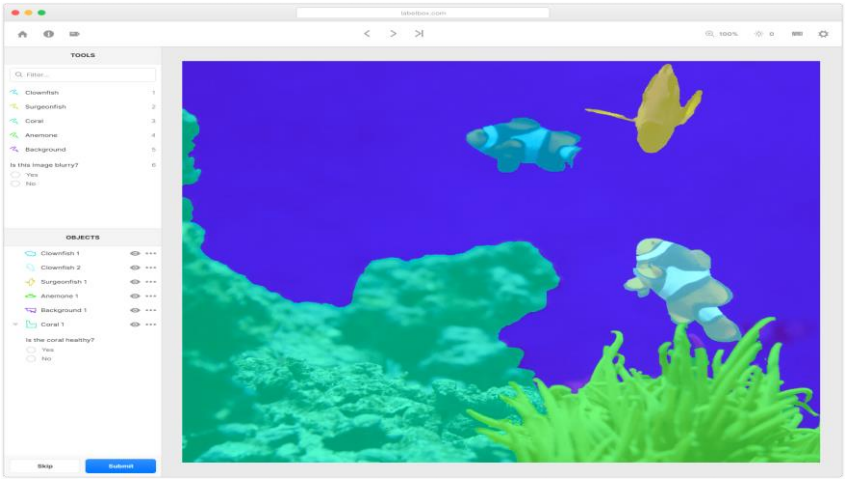

Figure I-24: Labelbox Interface

Select deep learning algorithm it depends on **the application**, it depends all on **the available data**, **available hardware**.

I.4.2.3 Training the model

Training a deep learning model can take hours, days, or weeks, depending on the size of the data and the amount of processing power available. Selecting a computational resource is a critical consideration when set up a workflow.

Currently, there are three computation options: CPU-based, GPU-based, and cloud-based.

1. **CPU-based computation:** is the simplest and most readily available option. It recommended using a CPU-based computation only for simple examples using a pre-trained network.
2. **Using a GPU-based** reduces network training time from days to hours. It can use a GPU in MATLAB without doing any additional programming. An NVidia® 3.0 compute-capable GPU is recommended. Multiple GPUs can speed up processing even more.
3. **Cloud-based GPU computation:** means that we don't have to buy and set up the hardware ourselves. The MATLAB code we write for using a local GPU can be extended to use cloud resources with just a few settings changes.

I.4.2.4 Test and deploying the model

Deploy trained model on embedded systems, enterprise systems, or the cloud. MATLAB supports automatic CUDA® code generation for the trained network as well as for preprocessing and post-processing to specifically target the latest NVIDIA GPUs, including Jetson Xavier and Nano.

When performance matters, it possible to generate code that leverages optimized libraries from Intel® (MKL-DNN), NVIDIA (TensorRT, cuDNN), and ARM® (ARM Compute Library) to create deployable models with high-performance inference speed. [15]

I.4.3 Types of Deep Learning Networks

I.4.3.1 Recurrent neural networks (RNNs)

RNN is a multi-layered neural network that can store information in context nodes, allowing it to learn data sequences and output a number or another sequence. In simple words it an artificial neural networks whose connections between neurons include loops. RNNs are well suited for processing sequences of inputs. [10]

I.4.3.1.1 Common uses of RNN

- Help securities traders to generate analytic reports
- Detect abnormalities in the contract of financial statement
- Detect fraudulent credit-card transaction
- Provide a caption for images
- Power chatbots
- The standard uses of RNN occur when the practitioners are working with time-series data or sequences (e.g., audio recordings or text).

I.4.3.2 Reinforcement learning

Reinforcement learning is a subfield of machine learning in which systems are trained by receiving virtual "rewards" or "punishments," essentially learning by trial and error. Google's DeepMind has used reinforcement learning to beat a human champion in the Go games. Reinforcement learning is also used in video games to improve the gaming experience by providing smarter bot. [10]

I.4.3.3 Convolutional neural networks (CNN)

A convolutional neural network (CNN or ConvNet) is one of the most popular algorithms for deep learning, a type of machine learning in which a model learns to perform classification tasks directly from images, video, text, or sound.

CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They learn directly from image data, using patterns to classify images and eliminating the need for manual feature extraction.

Applications that call for object recognition and computer vision — such as self-driving vehicles and face-recognition applications — rely heavily on CNNs. Depending on our application; we can build a CNN from scratch, or use a pre-trained model with our dataset. [16]

I.4.3.3.1 Importance of CNN

Using CNNs for DL has become increasingly popular due to three important factors:

- CNNs eliminate the need for manual feature extraction—the features are learned directly by the CNN.
- CNNs produce state-of-the-art recognition results.
- CNNs can be retrained for new recognition tasks, enabling us to build on pre-existing networks.

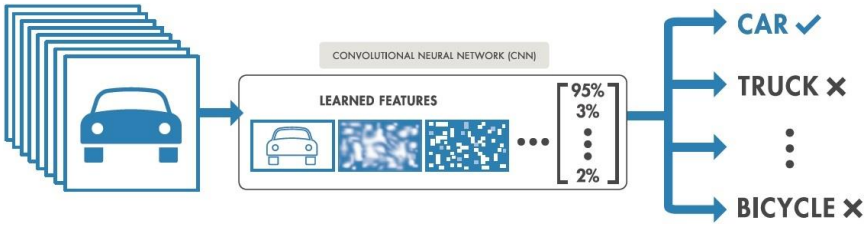

Figure I-25: Convolutional Neural Network (CNN)

I.4.3.3.2 How CNNs Work

A convolutional neural network can have tens or hundreds of layers that each learn to detect different features of an image. Filters are applied to each training image at different resolutions, and the output of each convolved image is used as the input to the next layer. The filters can start as very simple features, such as brightness and edges, and increase in complexity to features that uniquely define the object.

CNNs perform feature identification and classification of images, text, sound, and video. [16]

I.4.3.4 Feature Learning, Layers, and Classification

Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between.

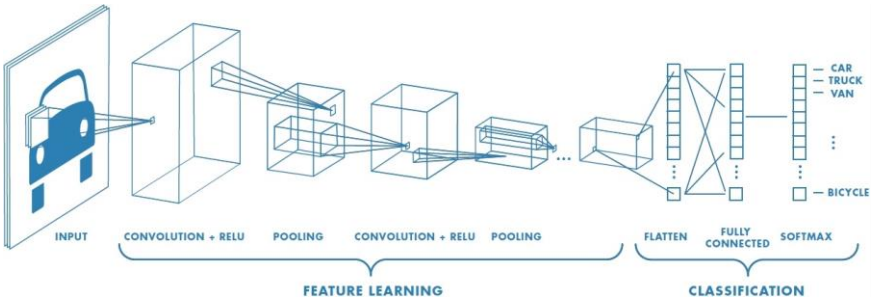

Figure I-26: Example of a network with many convolutional layers

These layers perform operations that alter the data with the intent of learning features specific to the data. Three of the most common layers are: convolution, activation or ReLU, and pooling. [16]

Convolution puts the input images through a set of convolutional filters, each of which activates certain features from the images. Note that, after the convolution, the size of the image is reduced.

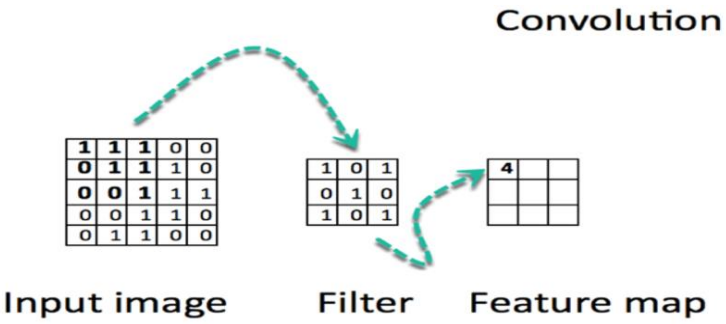

Figure I-27: Convolution

Rectified linear unit (ReLU) stands for rectified linear unit, allows for faster and more effective training by mapping negative values to zero and maintaining positive values. ReLU is a type of activation function, because only the activated features are carried forward into the next layer. Mathematically, it is defined as y = max(0, x). Visually, it looks like the following:

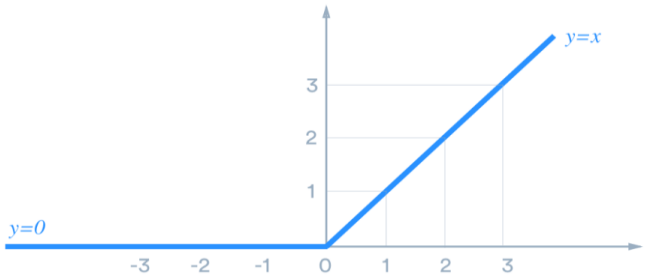

Figure I-28: Rectified linear unit (ReLU)

Pooling simplifies the output by performing nonlinear down sampling, reducing the number of parameters that the network needs to learn.

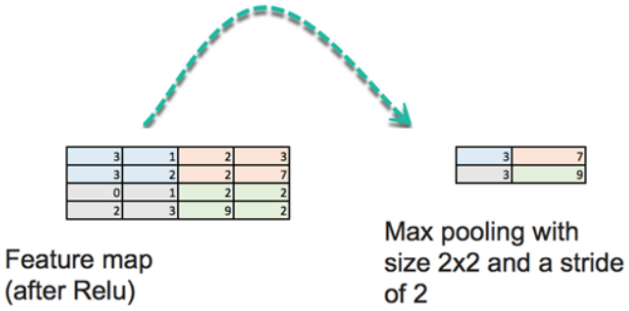

Figure I-29: example of a Max pooling operation

These operations are repeated over tens or hundreds of layers, with each layer learning to identify different features.

I.4.4 Types of training models

I.4.4.1 Training from Scratch

Creating a network from scratch means we determine the network configuration. It requires an understanding of the structure of a neural network and the many options for layer types and configuration [16]. This method tends to require more images for training, as the

new network needs many examples of the object to understand the variation of features. Training times are often longer.

I.4.4.2 Pre-trained models

A pre-trained model is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. Accordingly, due to the computational cost of training such models, it is common practice to import and use models from published literature (e.g. VGG, Inception, and MobileNet). [16]

I.4.4.3 Transfer Learning using pre-trained Models

Fine-tuning a pre-trained network with transfer learning is typically much faster and easier than training from scratch. It requires the least amount of data and computational resources. Transfer learning uses knowledge from one type of problem to solve similar problems. We start with a pretrained network and use it to learn a new task. One advantage of transfer learning is that the pretrained network has already learned a rich set of features. These features can be applied to a wide range of other similar tasks. [16]

With Deep Learning Toolbox, we can perform transfer learning with pretrained CNN models (such as GoogLeNet, AlexNet, vgg16, vgg19) and models from Caffe and TensorFlow-Keras.

Training from Scratch results can sometimes **exceed transfer learning**

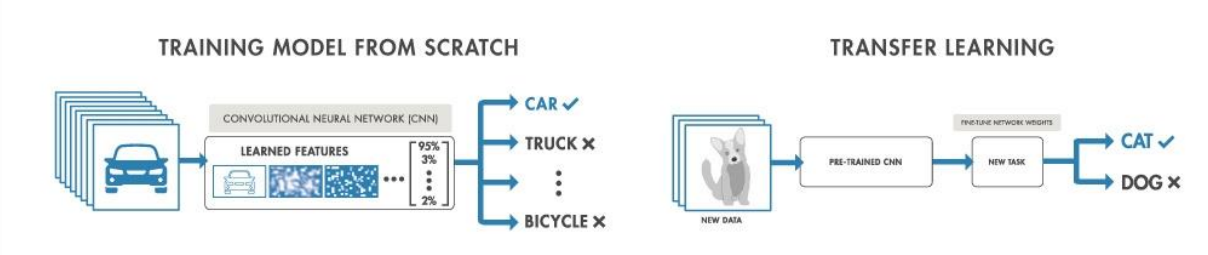

Figure I-30: Comparison of training a model from scratch and transfer learning. [16]

I.4.5 Deep Learning Libraries and Models

The Jetson Nano NVIDIA device can run a wide variety of advanced neural networks, including full native versions of popular Machine Learning (ML) and deep learning (DL) frameworks such as TensorFlow, PyTorch, Caffe/Caffe2, Keras, MXNet and others. The device utilizes the NVIDIA TensorRT accelerator library included with JetPack 4.2. In addition, Jetson Nano is capable of real-time performance in many scenarios and is able to process multiple high-definition video streams. The main Python libraries used for machine learning are:

I.4.5.1 Numpy

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow use NumPy internally for manipulation of Tensors. [17]

I.4.5.2 Theano

We all know that Machine Learning is basically mathematics and statistics. Theano is a popular python library that is used to define, evaluate and optimize mathematical expressions involving multi-dimensional arrays in an efficient manner. It is achieved by optimizing the utilization of CPU and GPU. It is extensively used for unit-testing and self-verification to detect and diagnose different types of errors. Theano is a very powerful library that has been used in large-scale computationally intensive scientific projects for a long time but is simple and approachable enough to be used by individuals for their own projects. [17]

I.4.5.3 TensorFlow

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application. [17]

I.4.5.4 Keras

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best things about Keras is that it allows for easy and fast prototyping.

I.4.5.5 PyTorch

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library which is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that supports on Computer Vision, Natural Language Processing (NLP) and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs. [17]

I.4.5.6 Apache MXNet (incubating)

Apache MXNet is a deep learning framework designed for both efficiency and flexibility. It allows you to mix symbolic and imperative programming to maximize efficiency and productivity. At its core, MXNet contains a dynamic dependency scheduler that automatically parallelizes both symbolic and imperative operations on the fly. A graph optimization layer on top of that makes symbolic execution fast and memory efficient. MXNet is portable and lightweight, scalable to many GPUs and machines.

MXNet is more than a deep learning project. It is a community on a mission of democratizing AI. It is a collection of blue prints and guidelines for building deep learning systems, and interesting insights of DL systems for hackers. [17]

I.4.5.7 CAFFE

CAFFE (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework, originally developed at University of California, Berkeley. It is open source, under a BSD license. It is written in C++, with a Python interface. [18]

Caffe supports many different types of deep learning architectures geared towards image classification and image segmentation. It supports CNN, RCNN and LSTM and fully connected neural network designs. Caffe supports GPU- and CPU-based acceleration computational kernel libraries such as NVIDIA cuDNN and Intel MKL.

Caffe is being used in academic research projects, startup prototypes, and even large-scale industrial applications in vision, speech, and multimedia. Yahoo! has also integrated Caffe with Apache Spark to create CaffeOnSpark, a distributed deep learning framework.

Caffe2

In April 2017, Facebook announced Caffe2, which included new features such as Recurrent Neural Networks. At the end of March 2018, Caffe2 was merged into PyTorch

I.5 Object Detection

Object detection is a technique used in the field of computer vision, Object detection is the process of finding and classifying objects in an image. Image classification involves predicting the class of one object in an image. Object localization refers to identifying the location of one or more objects in an image and drawing abounding box around their extent. Object detection combines these two tasks and localizes and classifies one or more objects in an image. [19]

We can classify three computer vision tasks:

Image Classification: Predict the type or class of an object in an image. So that the input is an image with a single object (such as a photograph), and the output is a class label (e.g. one or more integers that are mapped to class labels). [19]

Object Localization: Locate the presence of objects in an image and indicate their location with a bounding box. So that the input is an image with a single object (such as a photograph), and the output is one or more bounding boxes (e.g. defined by a point, width, and height). [19]

Object Detection: Locate the presence of objects with a bounding box and types or classes of the located objects in an image. So that the input is an image with a single object (such as a photograph), and the output is one or more bounding boxes and a class label for each bounding box. [19]

I.5.1 Object detection techniques:

Region-Based Convolutional Neural Networks, or R-CNNs, are a family of techniques for addressing object localization and recognition tasks, designed for model performance.

You Only Look Once, or YOLO, is a second family of techniques for object detection designed for speed and real-time use. [19]

I.5.1.1 R-CNN Model Family

The R-CNN family of methods refers to the R-CNN, which may stand for “Regions with CNN Features” or “Region-Based Convolutional Neural Network,” developed by Ross Girshick, et al. [19]

This includes the techniques R-CNN, Fast R-CNN, and Faster-RCNN designed and demonstrated for object localization and object recognition.

Models for object detection using regions with CNNs are based on the following three processes:

- Find regions in the image that might contain an object. These regions are called region proposals.
- Extract CNN features from the region proposals.
- Classify the objects using the extracted features.

There are three variants of an R-CNN. Each variant attempts to optimize, speed up, or enhance the results of one or more of these processes.

I.5.1.1.1 R-CNN

The R-CNN detector first generates region proposals using an algorithm. The proposal regions are cropped out of the image and resized. Then, the CNN classifies the cropped and resized regions. Finally, the region proposal bounding boxes are refined by a support vector machine (SVM) that is trained using CNN features. [19]

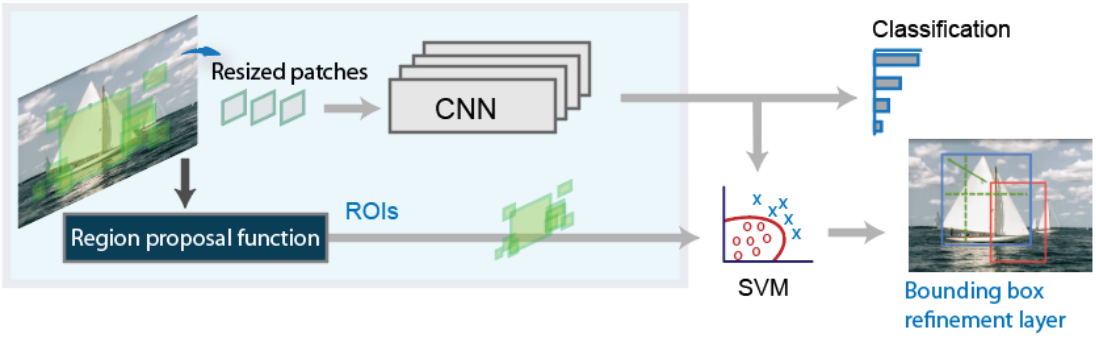

Figure I-31: The R-CNN detector model

I.5.1.1.2 Fast-R-CNN

As in the R-CNN detector, the Fast R-CNN detector also uses an algorithm like Edge Boxes to generate region proposals. Unlike the R-CNN detector, which crops and resizes region proposals, the Fast R-CNN detector processes the entire image. Whereas an R-CNN detector must classify each region, Fast R-CNN pools CNN features corresponding to each region proposal. Fast R-CNN is more efficient than R-CNN, because in the Fast R-CNN detector, the computations for overlapping regions are shared. [19]

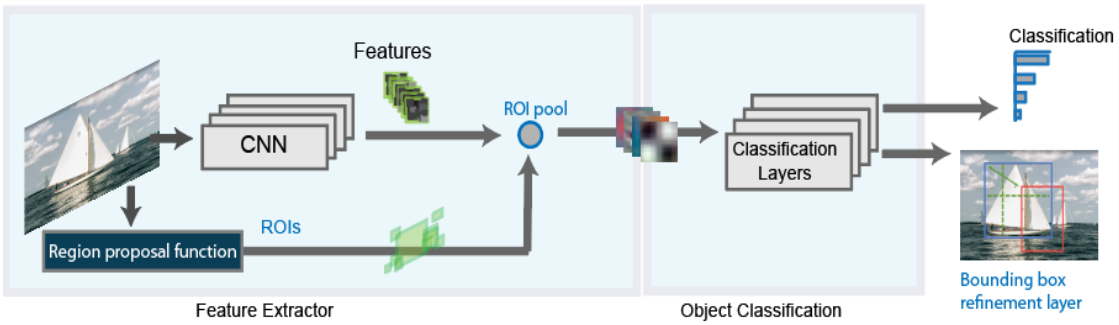

Figure I-32: the Fast R-CNN detector model

I.5.1.1.3 Faster-R-CNN

The Faster R-CNN detector adds a region proposal network (RPN) to generate region proposals directly in the network instead of using an external algorithm like Edge Boxes. The RPN uses Anchor Boxes for Object Detection. Generating region proposals in the network is faster and better tuned to our data. [19]

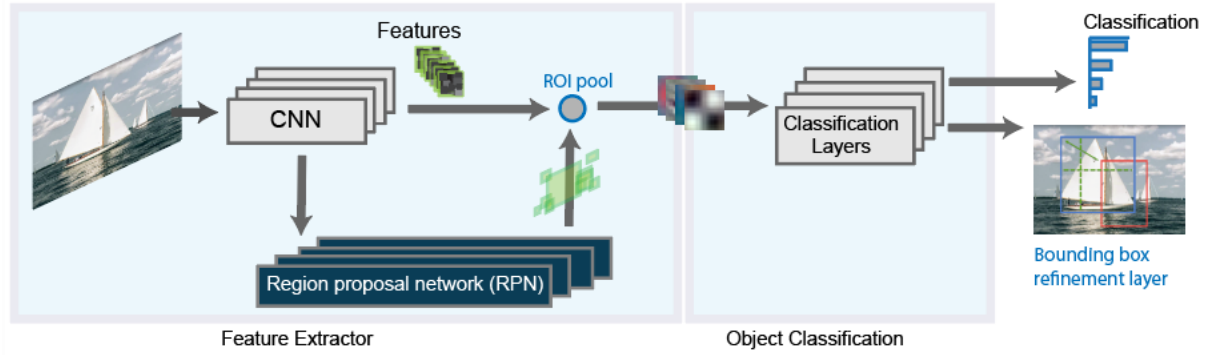

Figure I-33: The Faster R-CNN detector model

I.5.1.2 YOLO Model Family

Another popular family of object detection models is referred to collectively as YOLO or "You Only Look Once," developed by Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi. The R-CNN models may be generally more accurate, yet the YOLO family of models is fast, much faster than R-CNN. This means that we can achieve real-time object detection. [19]

YOLO involves a single neural network trained end-to-end that takes an image as input and predicts class labels and bounding boxes for each bounding box directly. This technique offers lower accuracy but operates at 45 frames per second and up to 155 frames per second for a speed-optimized version of the model.

A number of training and architectural changes were made to the model, such as the use of batch normalization and high-resolution input images.

Like Faster R-CNN, YOLOv2 model makes use of anchor boxes, pre-defined bounding boxes with useful shapes and sizes that are tailored during training. The choice of bounding boxes for the image is pre-processed using a k-means analysis on the training dataset.

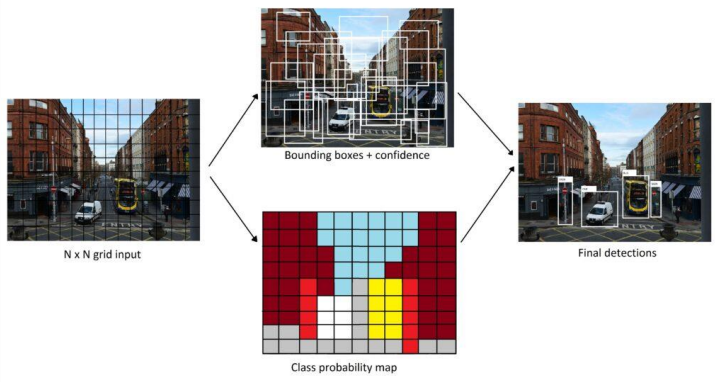

Figure I-34: YOLO Model

The Yolo model works by splitting the input image into grid cells, where each cell is associated with a bounding box that falls within it.

Each cell will predict the bounding box and confidence. A class prediction is also based on each cell.

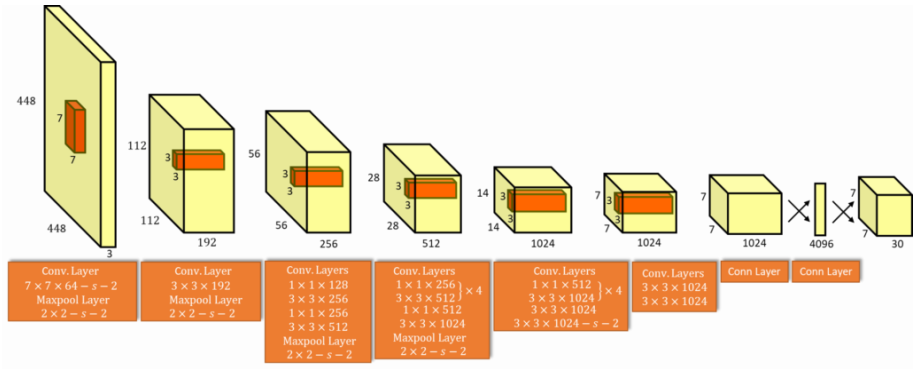

Figure I-35: The Yolo model Architecture

I.5.2 Datasets used for Object detection:

Datasets are an integral part of the field of machine learning. High-quality labeled training datasets for supervised and semi-supervised machine and deep learning algorithms are usually difficult and expensive to produce because of the large amount of time needed to label the data. Although they do not need to be labeled, high-quality datasets for unsupervised learning can also be difficult and costly to produce. [20]

Datasets for object detection task are consisting primarily of images or videos. There are many datasets for object detection such as:

Table I-1 Some Object detection datasets [20]

Dataset name	Preprocessing	Instances	Format	Default task	Creator
ImageNet	Labeled objects, bounding boxes, descriptive words, SIFT features	14,197,122	Images, text	Object recognition, scene recognition	J. Deng et al.
SIFT10M Dataset	Extensive SIFT feature extraction.	11,164,866	Text	Classification, object detection	X. Fu et al.
Open Images	Image-level labels, Bounding boxes	9,178,275	Images, text	Classification, Object recognition	
Microsoft Common Objects in Context (COCO)	Object highlighting, labeling, and classification into 91 object types.	2,500,000	Labeled images, text	Object recognition	T. Lin et al.
FieldSAFE	Classes labelled geographically	>400 GB of data	Images and 3D point clouds	Classification, object detection, object localization	M. Kragh et al.

I.6 Conclusion

We would like to summarize the salient findings of this chapter:

- ✓ We have given the definition of Artificial intelligence and machine learning.
- ✓ We have presented two techniques of machine learning (supervised & unsupervised).
- ✓ We have given the definition of deep learning and presented it's process, it's networks and it's libraries and models
- ✓ We also given a general overview about object detection and its techniques and used datasets.

Chapter II:
Materials and Methods

II.1 Introduction

NVIDIA Jetson Nano developer kit is an embedded system-on-module (SoM) based on the NVIDIA Tegra X1 processor, which combines CPU and GPU capabilities. As a result, it enables the development of millions of new small, low-power AI systems. It opens new worlds of embedded IoT applications, including entry-level Network Video Recorders (NVRs), home robots, and intelligent gateways with full analytics capabilities. In this work, we will use the Jetson Nano developer kit lunched by Nvidia in march 2019. The primary objective of this chapter is to give a general overview about embedded applications and IoT applications, and then a simplified comparison between CPUs and GPUs and embedded systems acronyms. We will detail the architecture of the Jetson Nano board, and we will also explain step by step how to install the jetpack and how to use the development tools such as Opencv, Cuda and TensorFlow.

II.2 Embedded Application

An embedded application is software that is placed permanently inside some kind of device to perform a very specific set of functions. Some small embedded applications like those in a microwave oven do not need an operating system (OS) to control them. Other, more complex, devices like entertainment systems that control hundreds of different channels and need to record and play videos from a variety of devices, will need an OS to manage the resources of the embedded system in real time. Real Time Embedded Systems include things like GPS Navigation systems, industrial robots, traffic monitoring systems, smart phones, and smart TVs. [21]

Embedded versions of popular operating systems like Linux, Windows and Mac are available, along with some specialized OSes. They will usually have reduced storage needs and will work with less RAM than a desktop OS. The program instructions for embedded systems are called firmware, or embedded software, and are stored in read-only memory, or flash memory chips. Embedded software is typically very easy on hardware resources – requiring little memory and often needing no keyboard or screen. The embedded software is not controlled by human interfaces, but rather by machine interfaces. [21]

II.3 The Internet of things (IoT)

II.3.1 Definition

The Internet of things (IoT) is essentially a platform where embedded devices are connected to the internet, so they can collect and exchange data with each other. It enables devices to interact, collaborate and, learn from each other’s experiences just like humans do.

Traditional fields of embedded systems, wireless sensor networks, control systems, automation, and others all contribute to enabling the Internet of things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", including devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common

ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers. [22]

II.3.2 IoT Applications

The applications of IoT technologies are multiple, because it is adjustable to almost any technology that is capable of providing relevant information about its own operation, about the performance of an activity and even about the environmental conditions that we need to monitor and control at a distance. [22]

Nowadays, many companies from different sectors are adopting this technology to simplify, improve, automate and control different processes.

IoT applications are expected to equip billions of everyday objects with connectivity and intelligence. It is already being deployed extensively, in various domains, namely:

- Wearables
- Smart Home Applications
- Health Care
- Smart Cities
- Agriculture
- Industrial Automation
- IoT Applications

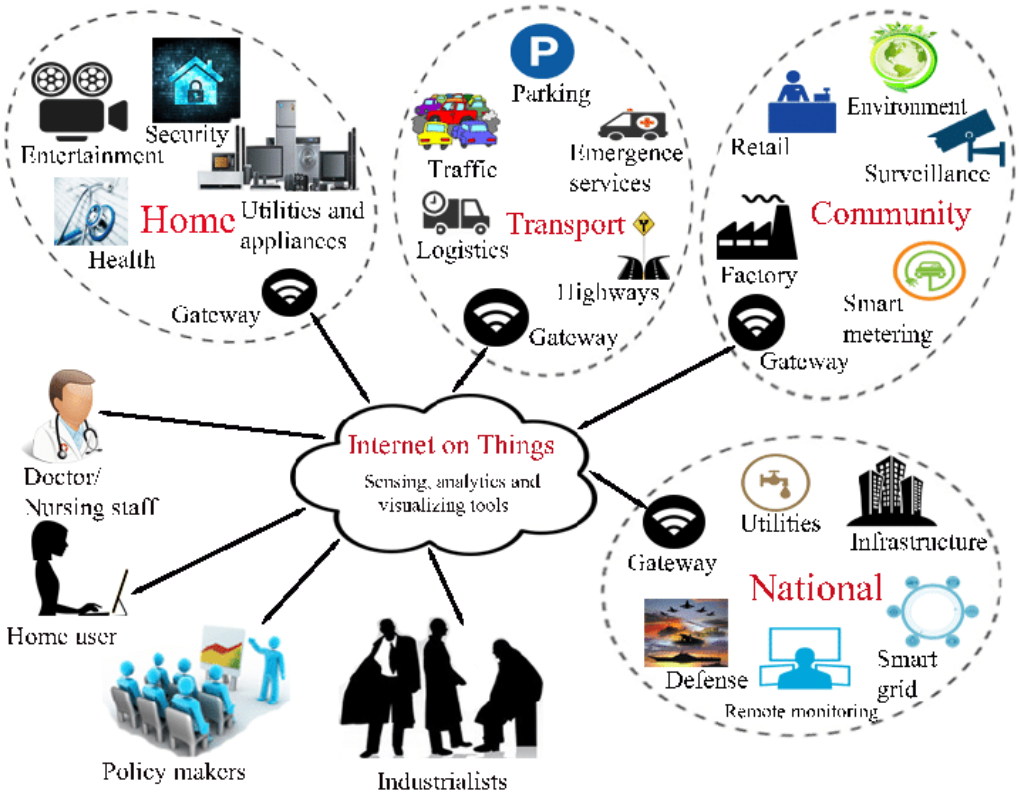

Figure II-1: architecture for Internet of IoT applications

II.4 Graphics Processing Units (GPUs)

GPUs consist of many processing cores, and are accelerators that are optimized for performing fast matrix calculations in parallel. These devices are typically very affordable, since their development is motivated by the gaming industry.

Feng et al. mentions that the advances in the development of computer vision algorithms are not only based on deep learning techniques and large data sets, but also relies on advanced parallel computing architectures that enable efficient training of multiple layers of neural networks. Furthermore, modern GPUs are not only a powerful graphics engine but also a highly parallelized computing processor featuring high throughput and high memory bandwidth for massive parallel algorithms. [23]

Although GPUs were initially intended for high-performance gaming as well as graphics rendering, new techniques in the NVIDIA-developed Compute Unified Device Architecture (CUDA) and CUDA Deep Neural Network Library (cuDNN) have enabled users to adapt them for particular purposes, enhancing system performance. CUDA cores or stream processors are the smallest processing units of NVidia GPUs, and each task can be assigned to one of them.

Zhang et al. states that GPUs are typically built with thousands of cores and operate exceptionally well on the rapid matrix multiplications that are required for neural network training. Therefore, higher memory bandwidth is provided to CPUs and the learning process is dramatically accelerated.

HajiRassouliha et al. provides suitable considerations for the selection of hardware in computer vision and image processing tasks, the devices assessed are Digital Signal Processors (DSPs), Faithful-programmable gate arrays (FPGAs) and Graphics Processing Units (GPUs). In this research, the advantages and disadvantages of development time, tools and utilities, and type of hardware accelerator are discussed, as well as the fact that GPUs are well suited for implementing deep neural network algorithms, because of the similarity between the mathematical basis of neural networks and image manipulation tasks of GPUs. Given their advantages, GPUs have been the most common choice of hardware implementation. [23]

Basulto-Lantova et al. showed the comparison between Jetson TX2 and Jetson Nano performance by using their development kits. Evidence from tests with different image sizes shows that Jetson TX2 is faster than Jetson Nano, which is expected due to the differences in hardware characteristics. However, both computers maintain a relatively low processing time when performing image processing tasks.

II.4.1 How CPU and GPU Work Together

A CPU (central processing unit) works together with a GPU (graphics processing unit) to increase the throughput of data and the number of concurrent calculations within an application. GPUs were originally designed to create images for computer graphics and video

game consoles, but since the early 2010's, GPUs can also be used to accelerate calculations involving massive amounts of data.

A CPU can never be fully replaced by a GPU: a GPU complements CPU architecture by allowing repetitive calculations within an application to be run in parallel while the main program continues to run on the CPU. The CPU can be thought of as the taskmaster of the entire system, coordinating a wide range of general-purpose computing tasks, with the GPU performing a narrower range of more specialized tasks (usually mathematical). Using the power of parallelism, a GPU can complete more work in the same amount of time as compared to a CPU.

II.4.2 Difference between CPU and GPU

The main difference between CPU and GPU architecture is that a CPU is designed to handle a wide-range of tasks quickly (as measured by CPU clock speed), but are limited in the concurrency of tasks that can be running. A GPU is designed to quickly render high-resolution images and video concurrently.

Because GPUs can perform parallel operations on multiple sets of data, they are also commonly used for non-graphical tasks such as machine learning and scientific computation. Designed with thousands of processor cores running simultaneously, GPUs enable massive parallelism where each core is focused on making efficient calculations.

II.5 Embedded system acronyms

II.5.1 System-In-A-Package (SiP)

A system in package (SiP) contains several ICs (chips) including a microprocessor on a single substrate such as ceramic or laminate. An example SiP can comprise several chips—such as a specialized processor, DRAM, flash memory—combined with passive components—resistors and capacitors—all mounted on the same substrate. This means that a complete functional unit can be built in a multi-chip package so that few external components need to be added to make it work. [24]

SiP dies can be stacked vertically or tiled horizontally, unlike slightly less dense multi-chip modules, which place dies horizontally on a carrier. SiP connects the dies with standard off-chip wire bonds or solder bumps.

The appeal of a SiP is that it can be compact an otherwise complex system into a very simple package, making it easier to integrate into larger systems. It also simplifies PCB layouts.

Unlike a SOC that is based on a single silicon die, SiP can be based on multiple dies in a single package. SiP is believed to provide more interconnection in the future and possibly face out SoCs.

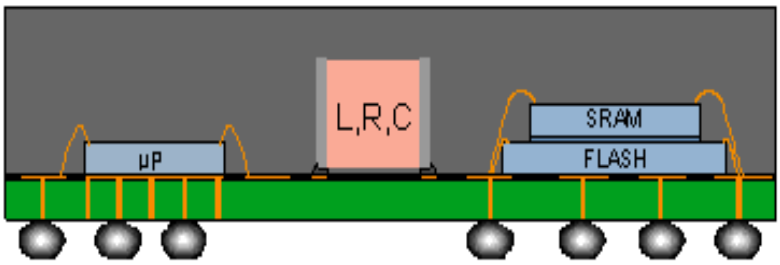

Figure II-2: system in package (SiP)

II.5.2 Package-On-A-Package (PoP)

A Package-on-a-Package stacks single-component packages vertically, connected via ball grid arrays. Packages can be discrete components (memory, CPU, other logic) or a System-in-a-Package stacked with another package for added or expanded functionality. [24]

PoP provides more component density and also simplifies PCB design. It can also improve signal propagation since the interconnects between components is much shorter.

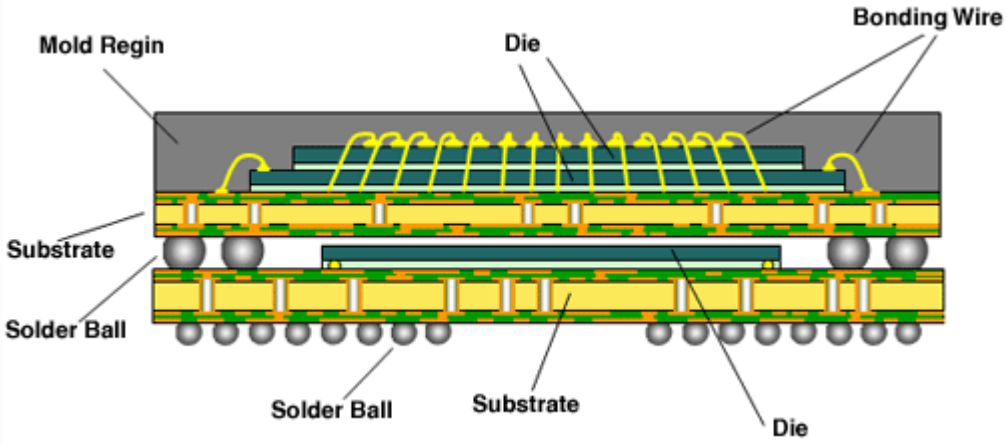

Figure II-3: Package-On-A-Package (PoP)

II.5.3 System-On-A-Chip (SoC)

A System-on-a-chip (SoC) is a microchip with all the necessary electronic circuits and parts for a given system, such as a smartphone or wearable computer, into a single integrated circuit (IC). [24]

A SoC integrates a microcontroller (or microprocessor) with advanced peripherals like graphic processing unit (GPU), Wi-Fi module, or coprocessor.

Think of a SoC as a computer package inside a chip. The SoC integrates all components of a system into one. It may contain digital, analog, mixed-signal, and often radio-frequency functions – all on a single substrate. A SoC can be based around either a microcontroller (includes CPU, RAM, ROM, and other peripherals) or a microprocessor (includes only a CPU). It is also possible for SoCs to be customized for a specific application, including

whatever components, memory, or peripherals necessary, ranging from digital/analog signal ICs, FPGAs, and IOs.

One of the major advantages of a SoC is that it is usually cheaper, smaller, easy to scale, and even more energy efficient. It is easier to build around a SoC for a product than to add several components individually. Despite its obvious advantages, SoC still has a significant disadvantage – we are going to be locked into that hardware configuration for life. This could be fine for consumer products, since we don't expect any hardware upgrade or so but would limit hacking for maker's related application.

A good example of a SoC is what we have in the Raspberry Pi; The Raspberry Pi uses a system on a chip as an almost fully-contained microcomputer. SoCs can help engineers speed up a product to market and even the adoption of new protocols, such as those Bluetooth 5 SoCs, that make it easier to integrate Bluetooth 5 into new products.

II.5.4 System on Module (SoM) / Computer on Module (CoM)

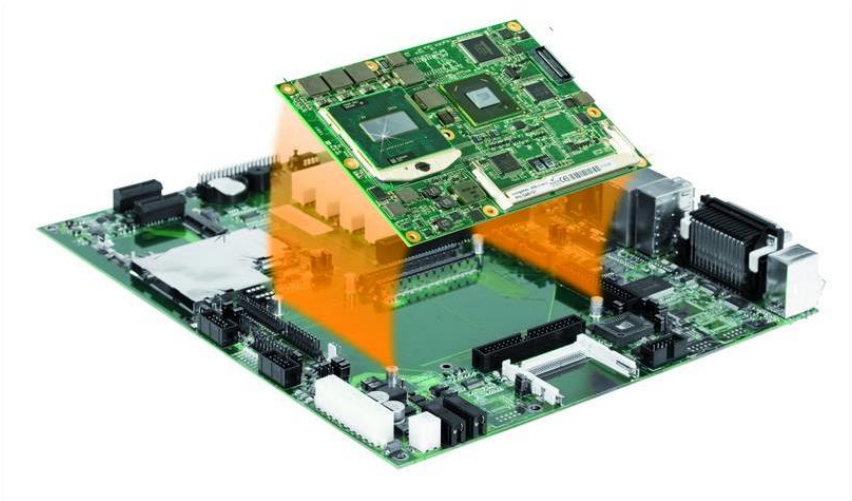

Figure II-4: Computer on a module

A System on a Module (SoM) and Computer on Module usually refers to the same thing. A Computer-on-a-module is a step above a SoC; it means a computer or system packaged in a single module. CoMs usually provide every piece we need to build a complete system; they incorporate a SoC (most of the time), connectivity, multimedia and display, GPIO, operating system, and others into one single module. [24]

SoM based designs are usually scalable. SoMs/CoMs are usually paired with a carrier board. These carrier boards are usually used to extend out the SoMs functionality or parts. A SoM helps system designers realize a fully customized electronics assembly, complete with custom interfaces and form factor without the effort of a ground-up electronics design. Customers can purchase an off-the-shelf SoM and marry it to an easy to develop custom baseboard to create a solution functionally identical to one that is fully custom-engineered.

CoMs provide a plug-and-play type advantage since a CoM can be replaced or upgraded within a carrier, without having to change the carrier. There are some benefits to the SoM

approach vs. ground-up development. These include cost savings, reduced risk, a variety of CPU choices, decreased customer design requirements, and a small footprint.

Unlike an SBC, a computer-on-module is a type of single-board computer made to plug into a carrier board, baseboard, or backplane for system expansion.

II.5.5 Single Board Computers (SBCs)

II.5.5.1 Definition

A single-board computer (SBC) is a complete computer built on a single circuit board, with a microprocessor(s), memory, input/output (I/O) and other features required for a functional computer. Single-board computers were made as demonstration or development systems, for educational systems, or for use as embedded computer controllers. [24]

Single-board computer builds on SoC to provide a full-fledged computer on small circuit board. Examples of popular SBCs are Raspberry Pi boards, Nvidia Jetson, Beaglebone, and several others.

Figure II-5: A Raspberry Pi SBC

II.5.5.2 Architecture of Single Board Computers

Single board computers emerged owing to the increase of integrated circuits density. Smaller overall size and costs of the system can be obtained by putting all the functionalities on one board. One way to do this is by reducing the number of circuit boards required, and by eliminating connectors and bus driver circuits.

Basic building blocks of SCB architecture are processor core, memory banks, buses and peripherals. Internal data bus is used to connect all the system's blocks. New multi core processor architectures over 1 GHz are now common feature of SCB platforms. General architecture of single board computer is shown on (Figure II-6) here; accent is put on a concept of SCB, not necessarily on an exact configuration of peripheral units.

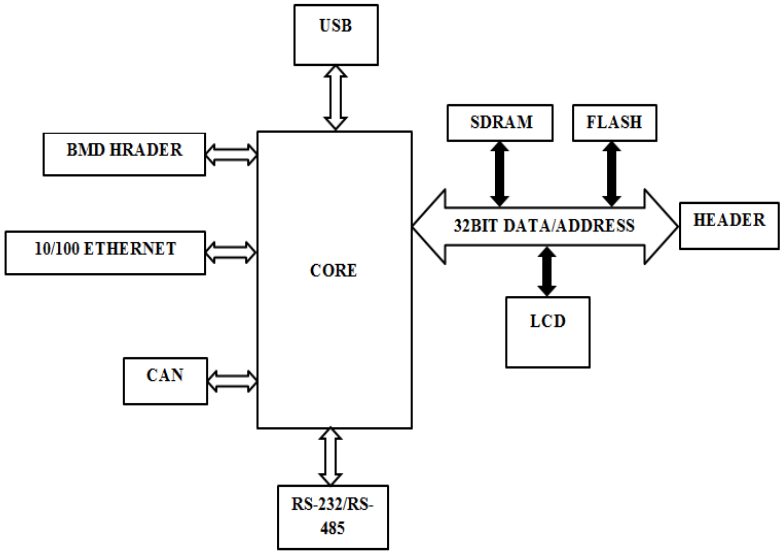

Figure II-6: Architecture of single computer board

II.5.5.3 Types of SBCs

While SBCs can be used for most any purpose, many have originally been designed for a specific purpose or application. A perfect example of this is the Raspberry Pi which was developed as an educational tool to help encourage and strengthen students programming skills. The BeagleBoard and BeagleBone were also developed to help educate and promote the benefits and usage of open source hardware and software in embedded computing.

Numerous other SBCs have been developed in the past few years including Atmel's SAMA5D3 Xplained, which was designed for rapid prototyping development, and the RIoTboard which focuses on Android development to enable the development of the Internet of Things. Other well-known boards include the PandaBoard, OlinuXino, as well as a whole host of Allwinner ARM SoC based SBCs.

The long term success of an SBC, like most other products, relies heavily on the performance/price ratio. But what also weighs in just as heavily is the amount of available support for a particular board or range of boards. While some SBCs rely on a dedicated supplier or secondary support entity, most of the open source SBCs are supported through a community of developers. These communities strengthen the board's proposition by providing software updates from individuals as well as projects that showcase the board's features as well as the many accessories that are often available to expand the board's functionality.

II.6 NVIDIA Jetson

II.6.1 What is Jetson?

NVIDIA® Jetson is the world's leading platform for AI at the edge. Its high-performance, low-power computing for deep learning and computer vision makes it the ideal platform for compute-intensive projects. [25]

II.6.2 Jetson Developer Kits and Jetson modules

Each Jetson developer kit includes a non-production specification Jetson module attached to a reference carrier board. Together with JetPack SDK, it is used to develop and test software for use case. Jetson developer kits are not intended for production use.

Jetson modules are suitable for deployment in a production environment throughout their operating lifetime. Each Jetson module ships with no software pre-installed. [25]

Table II-1: Some differences between Jetson developer kits and modules [25]

	Jetson developer kit	**Jetson module**
Operating Lifetime	None specified	5 or 10 year operating life in a production environment
Warranty	1 year warranty for development use only	1 year warranty
Availability	No guarantee of availability. Order quantities may be limited. No notice before EOL.	Available for at least 5 years (up to 10). Built to forecast. Last Time Buy notice before EOL.
BOM	Several components may be non-production quality. Components may change without notification.	Production rated components. Any changes are notified via PCN following JEDEC JESD-046 standard.
Validation	Basic functional validation in a constrained environment.	Full functionality and reliability validation across environmental specification. Tests are listed in datasheet.

Table II-2: Some NVIDIA Jetson Modules

Features	**Jetson Nano**	**Jetson TX1**	**Jetson Xavier NX**
			
CPU	ARM Cortex-A57 (quad-core) @ 1.43GHz	ARM Cortex-A57 (quad-core) @ 1.73GHz	NVIDIA Carmel ARMv8.2 (6-core) @ 1.4GHz (6MB L2 + 4MB L3)
GPU	128-core NVIDIA Maxwell @ 921MHz	256-core NVIDIA Maxwell @ 998MHz	384-core Volta @ 1100MHz + 48 Tensor Cores
DL	NVIDIA GPU support (CUDA, cuDNN, TensorRT)		dual NVIDIA Deep Learning Accelerators
Memory	4GB 64-bit LPDDR4 @ 1600MHz | 25.6 GB/s		8GB 128-bit LPDDR4x @ 1600MHz | 51.2GB/s
Storage	MicroSD card	16GB eMMC 5.1	16GB eMMC 5.1
Vision	NVIDIA GPU support (CUDA, VisionWorks,		7-way VLIW Vision

	OpenCV)		Accelerator
Encoder	4Kp30, (2x) 1080p60, (4x) 1080p30		(2x) 4Kp30, (6x) 1080p60, (12x) 1080p30
Decoder	4Kp60, (2x) 4Kp30, (4x) 1080p60, (8x) 1080p30		(2x) 4Kp60, (4x) 4Kp30, (12x) 1080p60
Camera	12 lanes MIPI CSI-2 | 1.5 Gbps per lane		14 lanes MIPI CSI-2 | 2.5 Gbps per lane
Display	2x HDMI 2.0 / DP 1.2 / eDP 1.2 | 2x MIPI DSI		(2x) DP 1.4 / eDP 1.4 / HDMI 2.0 @ 4Kp60
Wireless	M.2 Key-E site on carrier	802.11a/b/g/n/ac 2×2 867Mbps | Bluetooth 4.0	M.2 Key-E site on carrier
Ethernet	10/100/1000 BASE-T Ethernet		
USB	(4x) USB 3.0 + Micro-USB 2.0	USB 3.0 + USB 2.0	USB 3.1 + (3x) USB 2.0
CAN	Not Supported		Single CAN bus controller
Misc IO	UART, SPI, I2C, I2S, GPIOs		
Socket	260-pin edge connector, 45x70mm	400-pin board-to-board connector, 50x87mm	260-pin edge connector, 45x70mm
Thermals	-25°C to 80°C		
Power	5/10W	10W	10/15W
Perf	472 GFLOPS	1 TFLOPS	

II.6.3 Machine Learning on Jetson

Jetson is able to natively run the full versions of popular machine learning frameworks, including TensorFlow, PyTorch, Caffe2, Keras, and MXNet.

There are also helpful deep learning examples and tutorials available, created specifically for Jetson - like Hello AI World and JetBot. [26]

II.6.4 TensorFlow on Jetson Platform

TensorFlow™ is an open-source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that flow between them. This flexible architecture lets us deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device without rewriting code. [27]

II.6.4.1 Benefits of TensorFlow on Jetson Platform

Installing TensorFlow for Jetson Platform provides us the access to the latest version of the framework on a lightweight, mobile platform without being restricted to TensorFlow Lite.

II.6.4.2 Ecosystem Products & Cameras

The Jetson Ecosystem includes a diverse set of companies producing add-ons, accessories, sensors, and software for Jetson such as carrier boards, enclosures, cameras, and custom design services.

II.6.4.3 Software Support

NVIDIA Jetson production modules and developer kits are all supported by the same NVIDIA software stack, enabling to develop once and deploy everywhere. JetPack SDK includes the latest Jetson Linux Driver Package (L4T) with Linux operating system and CUDA-X accelerated libraries and APIs for AI Edge application development. It also includes samples, documentation, and developer tools for both host computer and developer kit, and supports higher level SDKs such as DeepStream for streaming video analytics and Isaac for robotics.

II.6.5 NVIDIA JetPack SDK

NVIDIA JetPack SDK is the most comprehensive solution for building AI applications. It bundles all the Jetson platform software, including TensorRT, cuDNN, CUDA Toolkit, VisionWorks, GStreamer, and OpenCV, all built on top of L4T with LTS Linux kernel. [28]

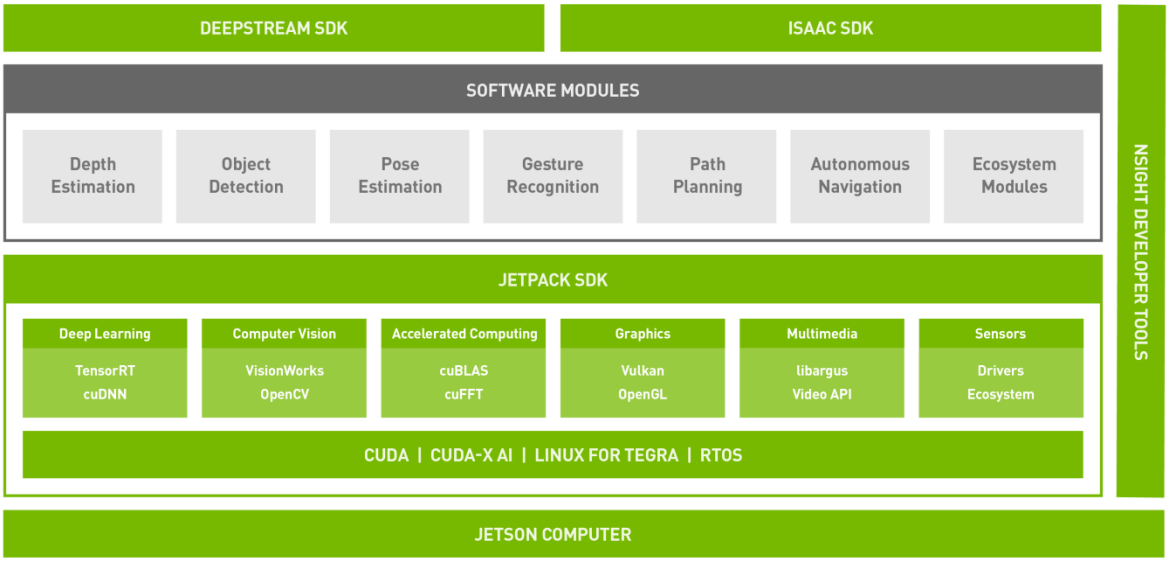

Figure II-7: NVIDIA JetPack SDK

II.6.5.1 JetPack includes:

JetPack provides the Jetson Developer Kit with the following:

- Full desktop Linux with NVIDIA drivers
- AI and Computer Vision libraries and APIs
- Developer tools
- Documentation and sample code

II.6.5.2 Summary of Jetpack Components

II.6.5.2.1 OS Image

JetPack includes a reference file system derived from Ubuntu.

II.6.5.2.2 Libraries and APIs

JetPack libraries and APIs include:

- TensorRT and cuDNN for high-performance deep learning applications
- CUDA for GPU accelerated applications across multiple domains
- NVIDIA Container Runtime for containerized GPU accelerated applications
- Multimedia API package for camera applications and sensor driver development
- VisionWorks, OpenCV, and VPI (Developer Preview) for visual computing applications
- Sample applications

II.6.5.2.3 Sample Applications

JetPack includes several samples which demonstrate the use of JetPack components. These are stored in the reference filesystem and can be compiled on the developer kit.

Table II-3: The reference filesystem for the JetPack component

JetPack component	Sample locations on reference filesystem
TensorRT	/usr/src/tensorrt/samples/
cuDNN	/usr/src/cudnn_samples_<version>/
CUDA	/usr/local/cuda-<version>/samples/
Multimedia API	/usr/src/tegra_multimedia_api/
VisionWorks	/usr/share/visionworks/sources/samples/ /usr/share/visionworks-tracking/sources/samples/ /usr/share/visionworks-sfm/sources/samples/
OpenCV	/usr/share/OpenCV/samples/
VPI	/opt/nvidia/vpi/vpi-0.0/samples

II.6.5.2.4 Developer Tools

JetPack includes the following developer tools. Some are used directly on a Jetson system, and others run on a Linux host computer connected to a Jetson system.

II.6.5.2.4.1 Tools for application development and debugging:

- Nsight Eclipse Edition for development of GPU accelerated applications: Runs on Linux host computer. Supports all Jetson products.
- CUDA-GDB for application debugging: Runs on the Jetson system or the Linux host computer. Supports all Jetson products.
- CUDA-MEMCHECK for debugging application memory errors: Runs on the Jetson system. Supports all Jetson products.

II.6.5.2.4.2 Tools for application profiling and optimization:

- Nsight Systems for application profiling across GPU and CPU: Runs on the Linux host computer. Supports all Jetson products.

- nvprof for application profiling across GPU and CPU: Runs on the Jetson system. Supports all Jetson products.
- Visual Profiler for application profiling across GPU and CPU: Runs on Linux host computer. Supports all Jetson products.
- Nsight Graphics for graphics application debugging and profiling: Runs on the Linux host computer. Supports all Jetson products.

II.7 Jetson Nano Developer Kit

II.7.1 Description

NVIDIA® Jetson Nano™ Developer Kit is a small, powerful computer that lets us run multiple neural networks in parallel for applications like image classification, object detection, segmentation, and speech processing. [29]

NVIDIA announced the Jetson Nano Developer Kit at the 2019 NVIDIA GPU Technology Conference (GTC), a powerful computer available now for embedded designers, and researchers, delivering the power of modern AI in a compact, easy-to-use platform with full software programmability. Jetson Nano delivers 472 GFLOPS of compute performance with a quad-core 64-bit ARM CPU and a 128-core integrated NVIDIA GPU. It also includes 4GB LPDDR4 memory in an efficient, low-power package with 5W/10W power modes and 5V DC input. [29]

Figure II-8: Jetson Nano Developer Kit (80x100mm)

Jetson Nano is supported by the comprehensive NVIDIA® JetPack™ SDK, This JetPack is compatible with NVIDIA's world-leading AI platform for training and deploying AI software, reducing complexity and effort for developers. [29]

II.7.2 Included in the Box

- A Jetson non-production Jetson Nano module (P3448-0000) with heatsink
- A reference carrier board (P3449-0000)
- A small paper card with quick start and support information
- A folded paper stand for the developer kit

The newly released JetPack 4.2 SDK provides a complete desktop Linux environment for Jetson Nano based on Ubuntu 18.04 with accelerated graphics, support for NVIDIA CUDA Toolkit 10.0, and libraries such as cuDNN 7.3 and TensorRT 5.The SDK also includes the ability to natively install popular open source Machine Learning (ML) frameworks such as TensorFlow, PyTorch, Caffe, Keras, and MXNet, along with frameworks for computer vision and robotics development like OpenCV.

Full compatibility with these frameworks and NVIDIA's leading AI platform makes it easier than ever to deploy AI-based inference workloads to Jetson. Jetson Nano brings real-time computer vision and inferencing across a wide variety of complex Deep Neural Network (DNN) models. These capabilities enable multi-sensor autonomous robots, IoT devices with intelligent edge analytics, and advanced AI systems. Even transfer learning is possible for re-training networks locally onboard Jetson Nano using the ML frameworks.

The Jetson Nano Developer Kit fits in a footprint of just 80x100mm and features four high-speed USB 3.0 ports, MIPI CSI-2 camera connector, HDMI 2.0 and DisplayPort 1.3, Gigabit Ethernet, M.2 Key-E module, MicroSD card slot, and 40-pin GPIO header. The ports and GPIO header works out-of-the-box with a variety of popular peripherals, sensors, and ready-to-use projects.

The devkit boots from a removable MicroSD card which can be formatted and imaged from any PC with an SD card adapter. The devkit can be conveniently powered via either the Micro USB port or a 5V DC barrel jack adapter. The camera connector is compatible with affordable MIPI CSI sensors including modules based on the 8MP IMX219, available from Jetson ecosystem partners. Also supported is the Raspberry Pi Camera Module v2, which includes driver support in JetPack. Table 1 shows key specifications.

II.7.3 Adding a Camera

The Raspberry Pi Camera Module V2 (Figure II-9) can work with Jetson Nano well, it will be a perfect camera in AI project.

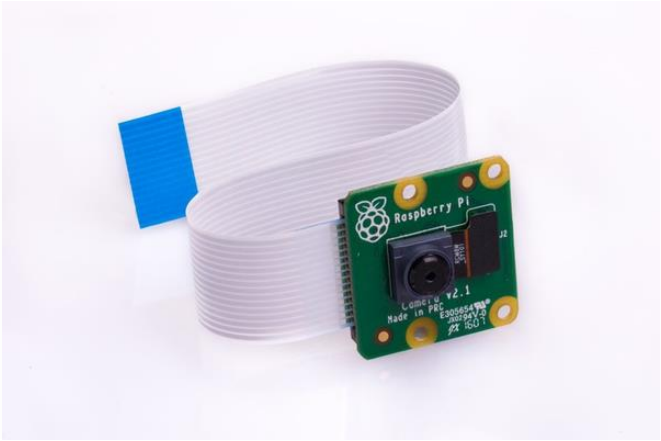

Figure II-9: The Raspberry Pi Camera Module V2

The Raspberry Pi Camera Module v2 Camera Module has a Sony IMX219 8-megapixel sensor.

The Camera Module can be used to take high-definition video, as well as stills photographs. It's easy to use for beginners, but has plenty to offer advanced users. There are lots of examples online of people using it for time-lapse, slow-motion, and other video cleverness. We can also use the libraries bundle with the camera to create effects.

It's a leap forward in image quality, color fidelity, and low-light performance. It supports 1080p30, 720p60 and VGA90 video modes, as well as still capture. It attaches via a 15cm ribbon cable to the CSI port on the Jetson Nano. [30]

The camera module is very popular in home security applications, and in wildlife camera traps.

II.7.4 Technical Specifications

Table II-4: Jetson Nano Developer Kit technical specifications [29]

GPU	128-core Maxwell
CPU	**Quad-core ARM A57 @ 1.43 GHz**
Memory	**4 GB 64-bit LPDDR4 25.6 GB/s**
Storage	**microSD (not included)**
Video Encode	**4K @ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)**
Video Decode	**4K @ 60 | 2x 4K @ 30 | 8x 1080p @ 30 | 18x 720p @ 30 (H.264/H.265)**
Camera	**2x MIPI CSI-2 DPHY lanes**
Connectivity	**Gigabit Ethernet, M.2 Key E**
Display	**HDMI and display port**
USB	**4x USB 3.0, USB 2.0 Micro-B**
Others	**GPIO, I²C, I²S, SPI, UART**
Mechanical	**69 mm x 45 mm, 260-pin edge connector**

II.7.5 Developer Kit Interfaces

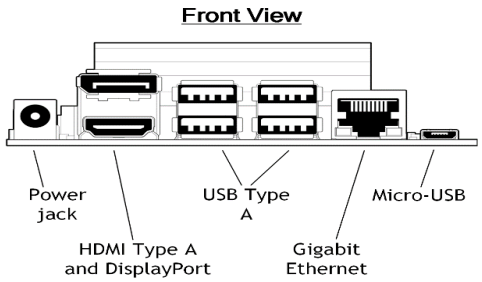

Figure II-10: Developer kit module and carrier boards: front views [31]

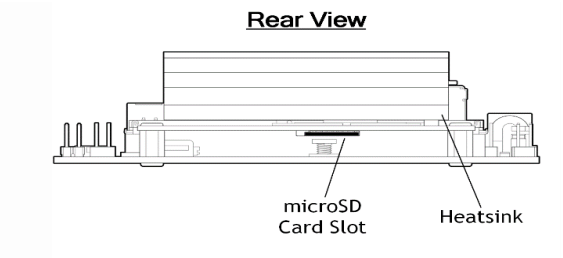

Figure II-11: Developer kit module and carrier boards: rear views [31]

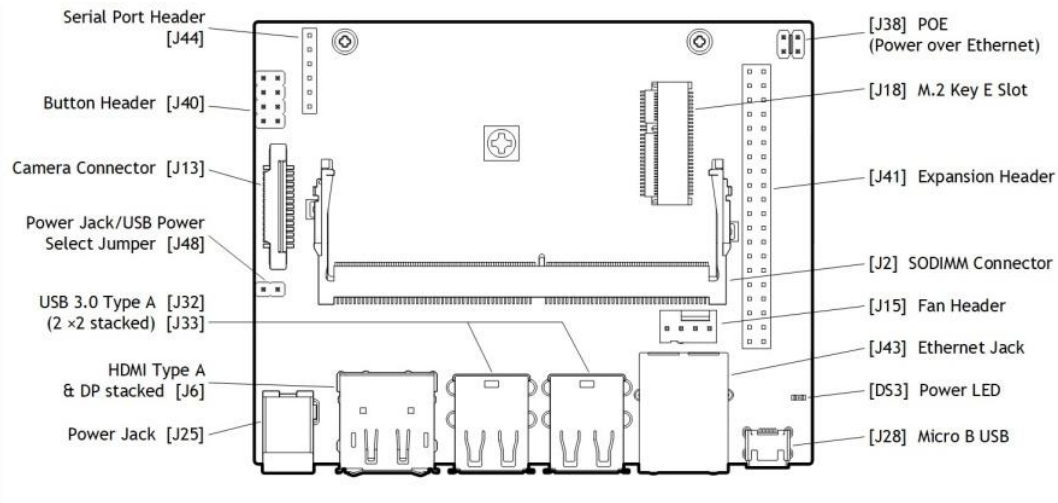

Figure II-12: Developer kit carrier boards: top view [31]

II.7.6 Ports & Interfaces

- 4x USB 3.0 A (Host)
- USB 2.0 Micro B (Device)
- MIPI CSI-2 x2 (15-position Camera Flex Connector)
- HDMI 2.0
- DisplayPort
- Gigabit Ethernet (RJ45)
- M.2 Key-E with PCIe x1
- MicroSD card slot
- (3x) I2C, (2x) SPI, UART, I2S, GPIOs

The devkit is built around a 260-pin SODIMM-style System-on-Module (SoM), shown in (Figure II-13). The SoM contains the processor, memory, and power management circuitry. The Jetson Nano compute module is 45x70mm. The production compute module will include 16GB eMMC onboard storage and enhanced I/O with PCIe Gen2 x4/x2/x1, MIPI DSI, additional GPIO, and 12 lanes of MIPI CSI-2 for connecting up to three x4 cameras or up to four cameras in x4/x2 configurations. Jetson's unified memory subsystem, which is shared between CPU, GPU, and multimedia engines, provides streamlined ZeroCopy sensor ingest and efficient processing pipelines. [31]

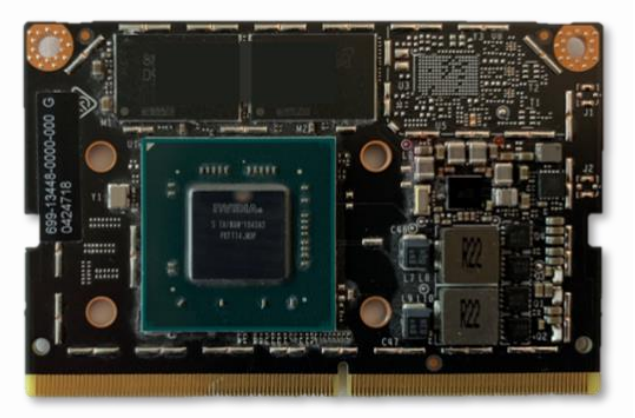

Figure II-13: The Jetson Nano compute module with 260-pin edge connector

II.7.7 Multi-Stream Video Analytics

Jetson Nano processes up to eight HD full-motion video streams in real-time and can be deployed as a low-power edge intelligent video analytics platform for Network Video Recorders (NVR), smart cameras, and IoT gateways. NVIDIA's DeepStream SDK optimizes the end-to-end inferencing pipeline with ZeroCopy and TensorRT to achieve ultimate performance at the edge and for on-premises servers. [31]

The block diagram in (Figure II-14) shows an example NVR architecture using Jetson Nano for ingesting and processing up to eight digital streams over Gigabit Ethernet with deep learning analytics. The system can decode 500 MP/s of H.264/H.265 and encode 250 MP/s of H.264/H.265 video. [31]

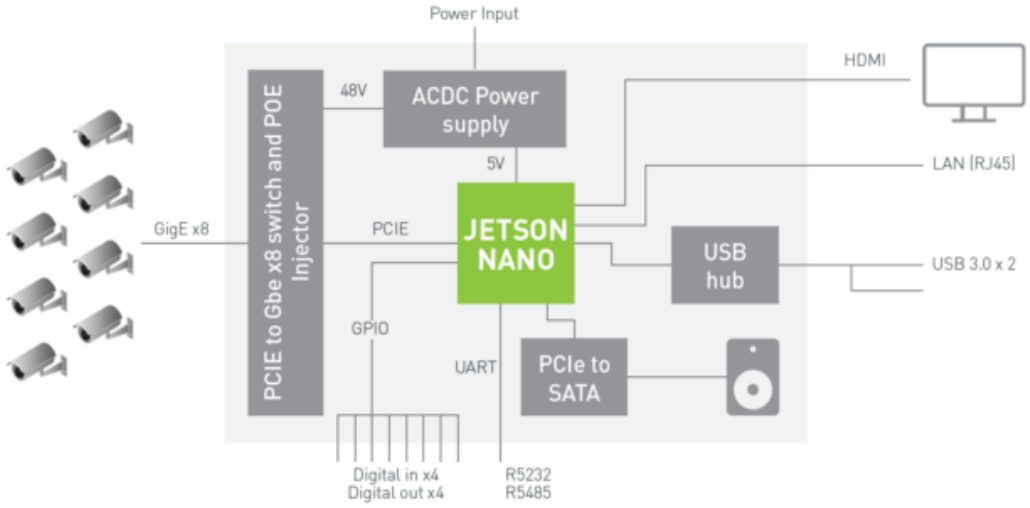

Figure II-14: NVR system architecture with Jetson Nano & 8xHD camera inputs [31]

II.7.8 Prepare for Setup

II.7.8.1 Items for Getting Started

- MicroSD Card: The Jetson Nano Developer Kit uses a microSD card as a boot device and for main storage. It's important to have a card that's fast and large enough for our

projects; the minimum recommended is a 16GB UHS-1 card. we will use the **Lexar High-Performance 633x 64GB MicroSDXC UHS-I Card with SD Adapter (LSDMI64GBBNL633A)**

- Micro-USB Power Supply: To power the developer kit we will need a good quality power supply that can deliver 5V⎓2A at the developer kit's Micro-USB port. As an example of a good power supply, the Raspberry Pi DSA-13PFC-05 FCA 051250 (5.1V⎓2.5A Universal MicroUSB Power Supply)

II.7.8.2 Lexar High-Performance 633x 64GB MicroSDXC

II.7.8.2.1 MicroSD Card Specification

- Premium memory solution for smartphones, tablets, or action cameras
- Quickly captures, plays back, and transfers media files, including 1080p full-HD, 3D, and 4K video
- Leverages UHS-I technology for a transfer speed up to 95MB/s
- Includes high-speed, Class 10 card and SD adapter
- Limited lifetime product support for card and one-year limited support for adapter

Figure II-15: Lexar 64GB MicroSDXC UHS-I Card with SD Adapter

II.7.8.3 Raspberry Pi Universal Power Supply

The official universal micro USB power supply recommended for Raspberry Pi 1, 2 and 3. It'll keep feeding our Jetson nano the steady 2.5A it needs for proper performance.

II.7.8.3.1 Power Supply Specification

The official and recommended universal micro USB power supply for Raspberry Pi1.5m lead

- Interchangeable heads for different countries
- Short circuit, over current and over voltage protection
- 50,000 hours MTBF

Figure II-16: Raspberry Pi Universal Power Supply

- To prepare our microSD card, we'll need a computer (with windows) with Internet connection and the ability to read and write SD cards, either via a built-in SD card slot or adapter.

II.7.9 How to install JetPack on the Jetson Nano

There are two ways to install JetPack on our developer kit:

- Use an SD Card image (hat what we will use): downloading the system image and use SD Card writing software to flash it to a microSD card. And boot the developer kit using the microSD card.
- Use NVIDIA SDK Manager: using Linux host computer with a working Internet connection to run SDK Manager and flash the developer kit. Supported host operating systems are:
- Ubuntu Linux x64 Version 18.04 or 16.04

II.7.9.1 Download and installing JetPack Nano

II.7.9.1.1 Write Image to the microSD Card

- First we download the Jetson Nano Developer Kit SD Card Image from the web site (https://developer.nvidia.com/jetson-nano-sd-card-image) and extract the files (Figure II-17)

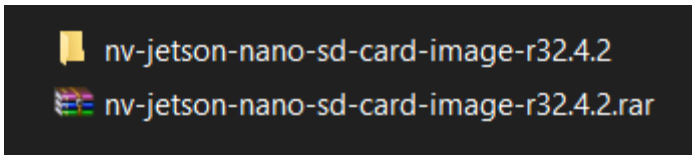

Figure II-17: The Jetson Nano Developer Kit SD Card Image

- We Format our microSD card using SD Card Formatter from the SD Association.
- We Download SD Memory Card Formatter for Windows from the web site (https://www.sdcard.org/downloads/formatter/eula_windows/).

We Install SD Card Formatter for Windows, and launch it.

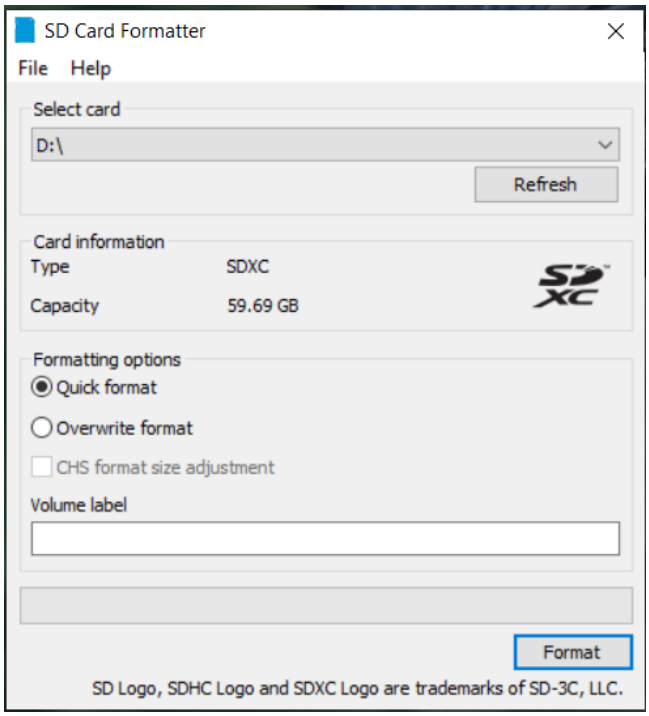

Figure II-18: SD Card Formatter

- We Select the card drive
- We Select "Quick format"
- We Leave "Volume label" blank
- We click "Format" to start formatting, and "Yes" on the warning dialog

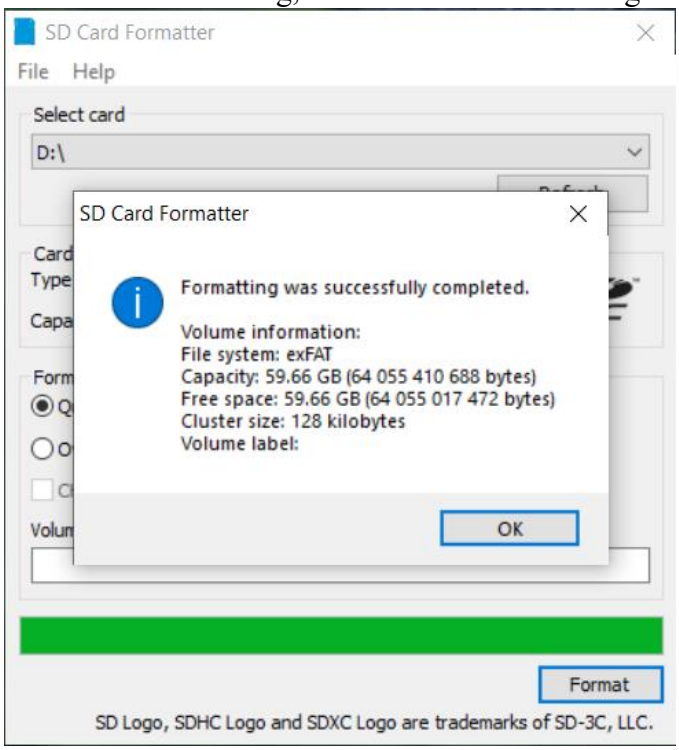

Figure II-19: Formatting theSD Card

- We use Etcher to write the Jetson Nano Developer Kit SD Card Image to our microSD card
- We download Etcher from web site (https://www.balena.io/etcher/)

- We install, and launch Etcher.

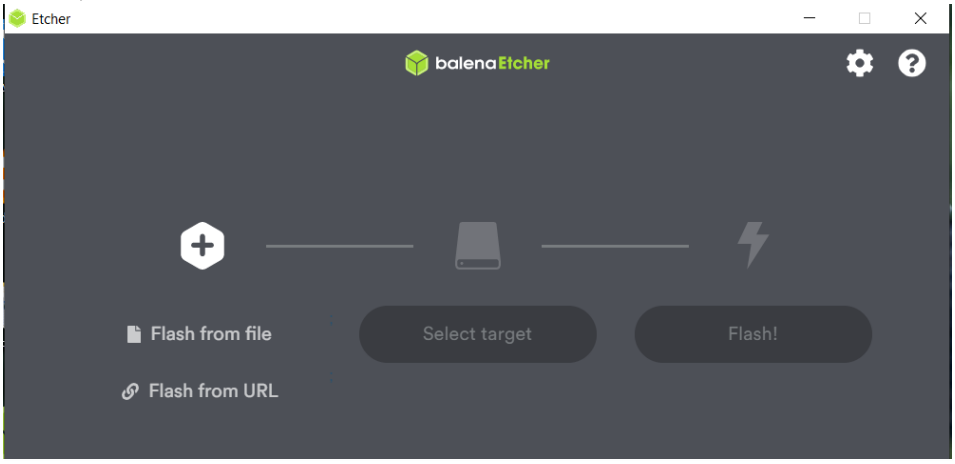

Figure II-20: Balena Etcher application

- We click “Select image” and choose the zipped image file downloaded earlier.

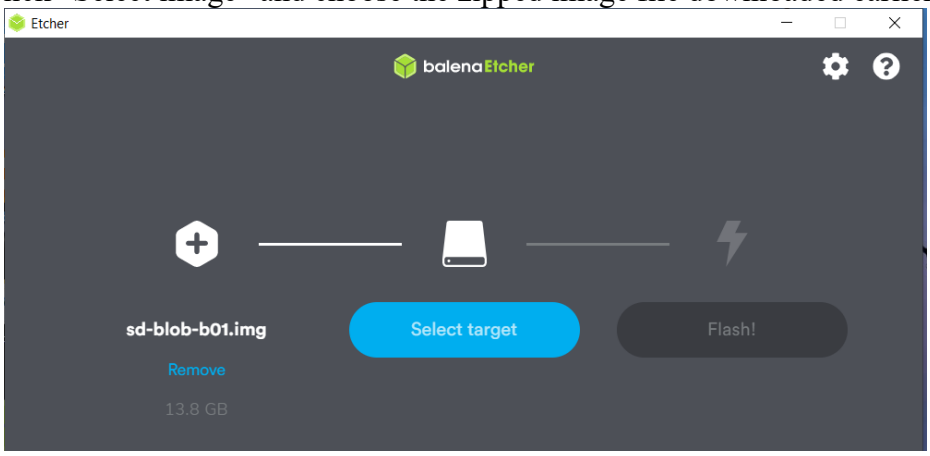

Figure II-21: Select the Image file

- We click “Select drive” and we choose our SD card.

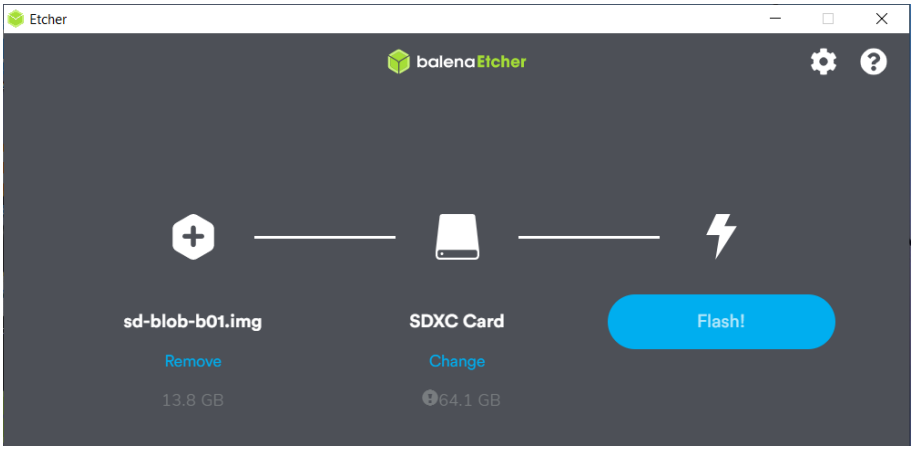

Figure II-22: Select the drive and flash it

- We click “Flash!”
  After our microSD card is ready, we proceed to set up our developer kit.

II.7.10 Setup and First Boot

II.7.10.1 Setup Steps

- Insertion of the microSD card (with system image already written to it) into the slot on the underside of the Jetson Nano module.
- Power on our computer display and connect it via HDMI cable.
- Connect the USB keyboard and mouse.
- Connect our Micro-USB power supply (5V⎓2A). The Jetson Nano Developer Kit will power on and boot automatically.

II.7.10.2 First Boot

A green LED next to the Micro-USB connector will light as soon as the developer kit powers on. When we boot the first time, the Jetson Nano Developer Kit will take us through some initial setup, including:

- We Review and accept NVIDIA Jetson software EULA

II.7.10.3 Initial Configuration

The basic configuration is menu driven and pretty straightforward. We will be choosing the language we speak, accepting a license agreement, selecting a time zone and keyboard style, and other routine system-setup tasks.

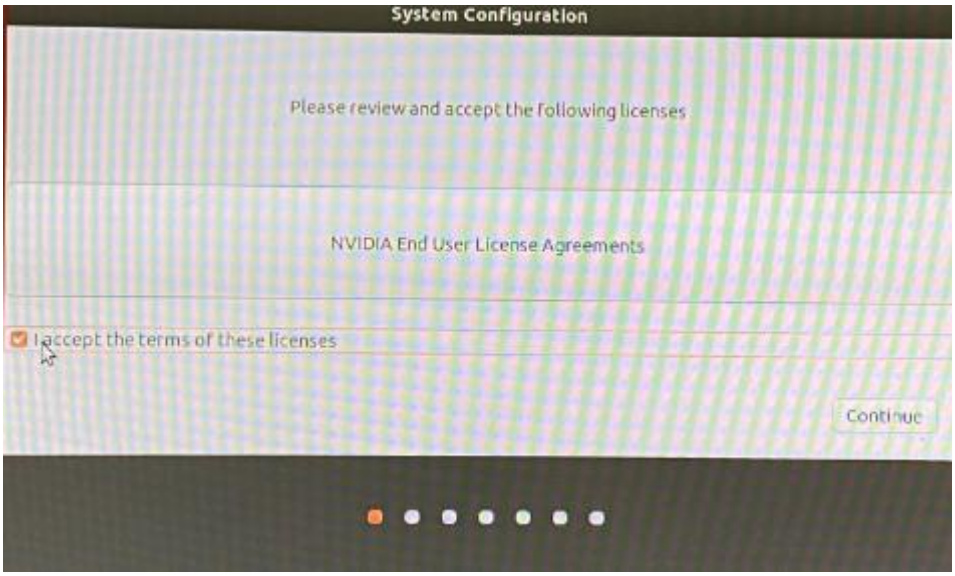

Figure II-23: accepting the license agreement

- Select system language, keyboard layout, and time zone

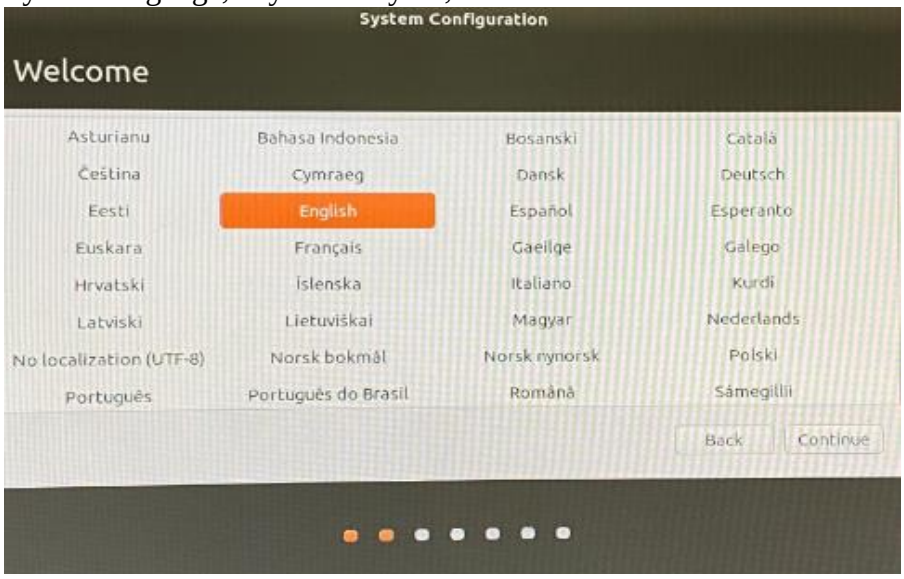

Figure II-24: Select system language

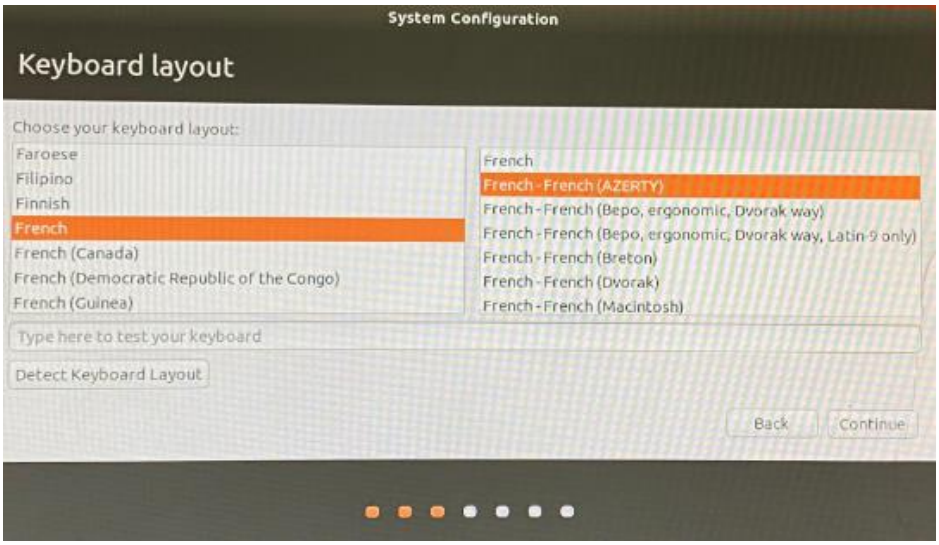

Figure II-25: Select keyboard layout

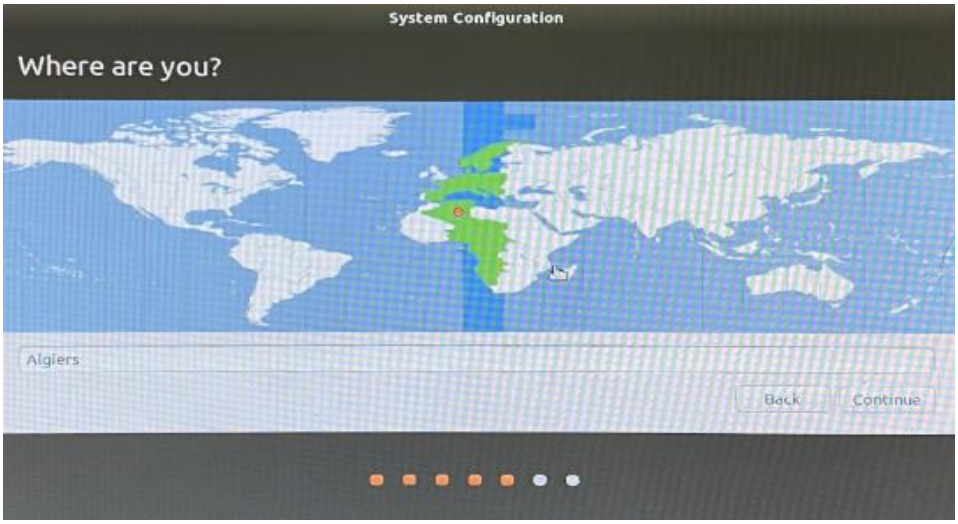

Figure II-26: Select time zone

- Create username, password, and computer name

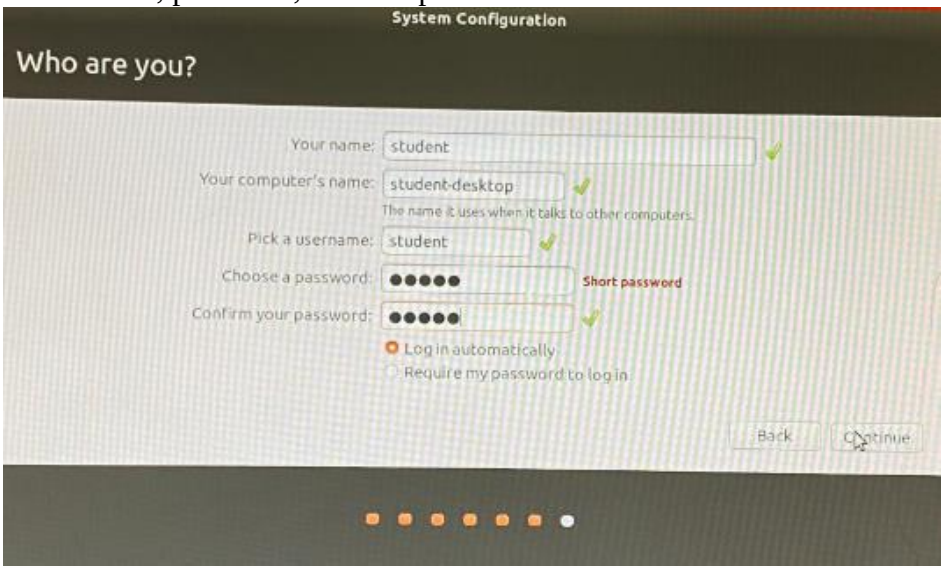

Figure II-27: Create username, password, and computer name

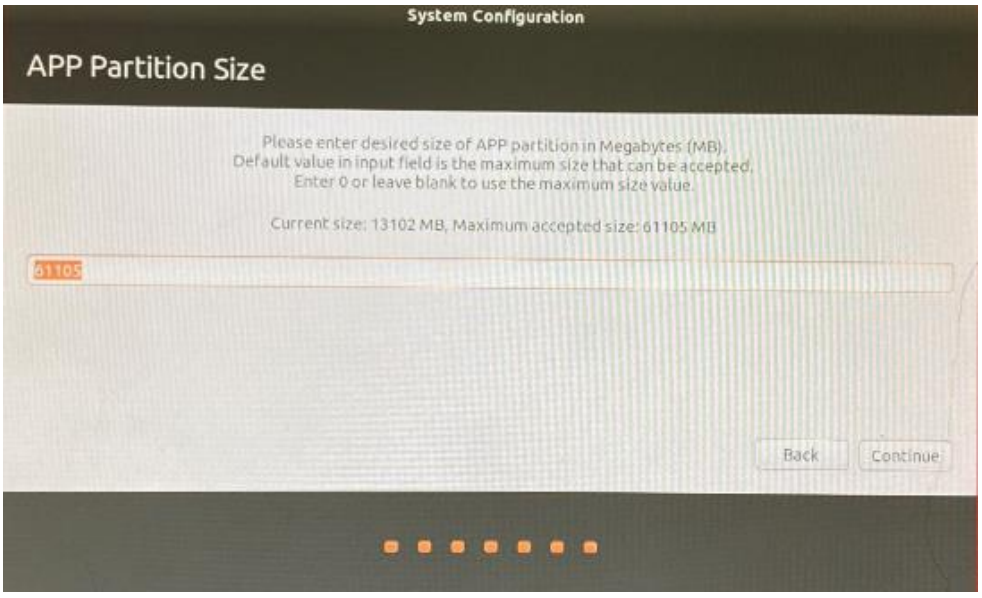

Figure II-28: Select application partition size

- Log in

Figure II-29: preparing to log in

II.7.10.4 After Logging In

- we will see this screen.

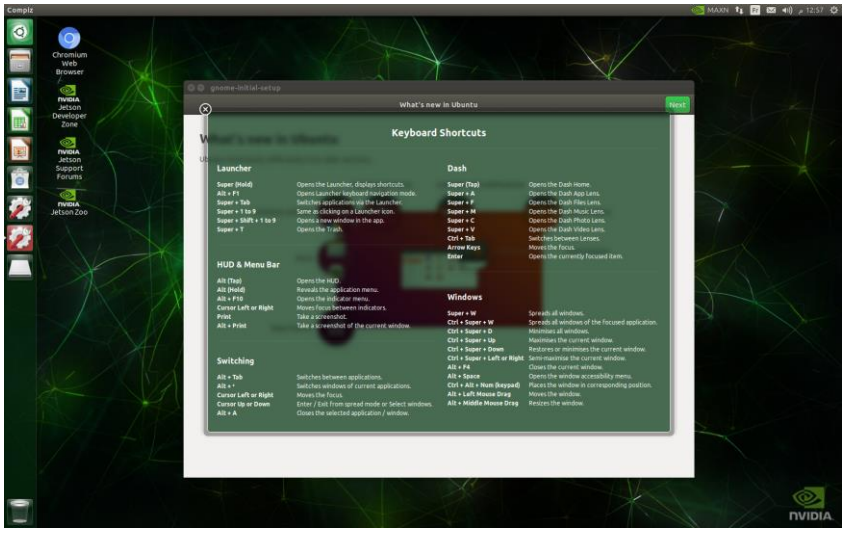

Figure II-30: Nvidia's Jetson nano desktop

Then we make some changes on the behavior settings and the brightness and lock settings

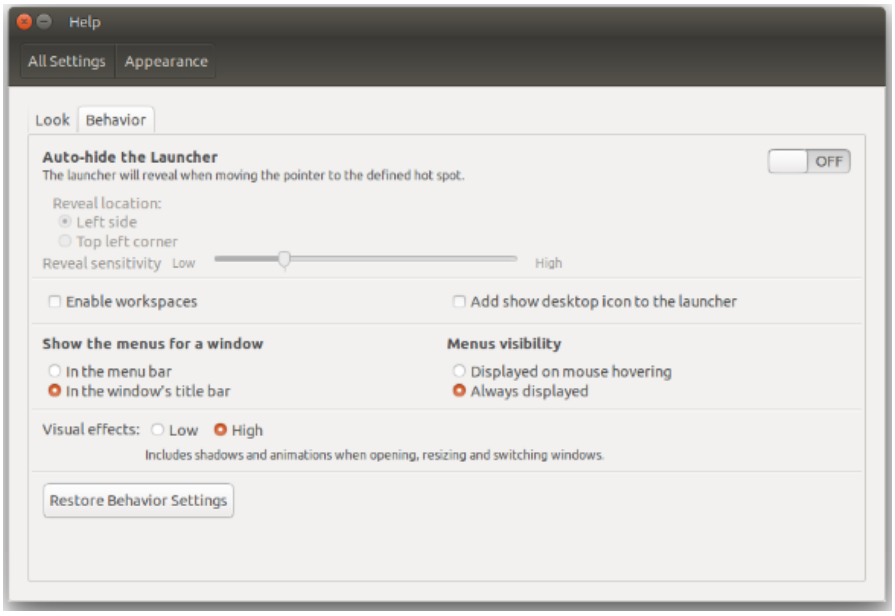

Figure II-31: changing the behavior settings

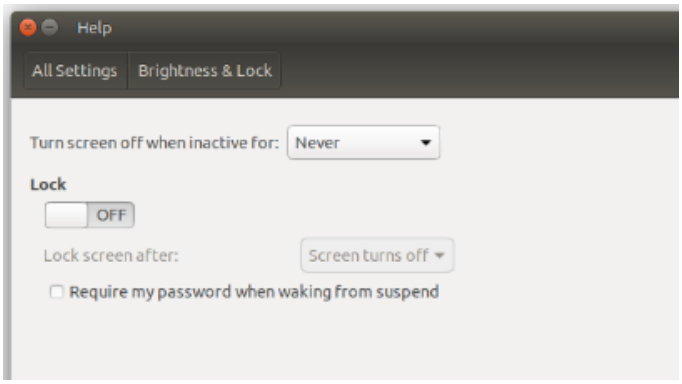

Figure II-32: changing the brightness and lock settings

- To knew what version of L4T are installed we use the command:

cat /etc/nv_tegra_release

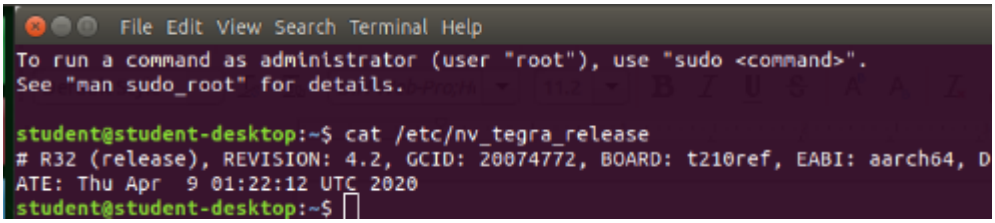

Figure II-33: The version of L4T installed

- To check cmake version we use the command: **cmake –version**
- To check ubuntu version we use the command: **lsb_release -a**
- To check opencv version we use these commands:

  python
  import cv2
  cv2.__version__
  exit()

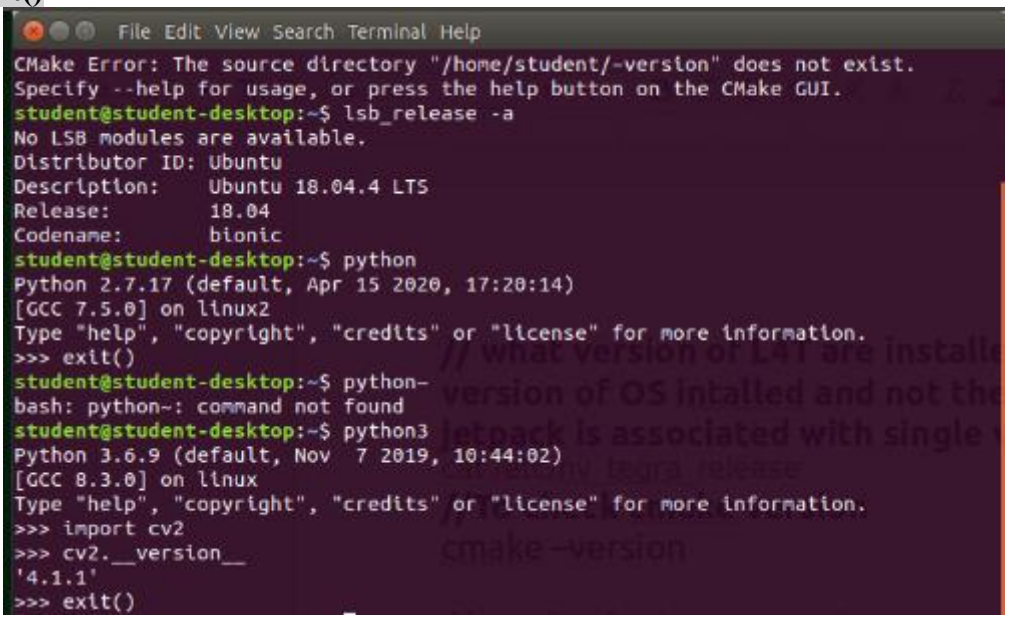

Figure II-34: Ubuntu, cmake, opencv installed versions

- we can execute these commands to get the complete build information of OpenCV

  python
  import cv2
  print(cv2.getBuildInformation())

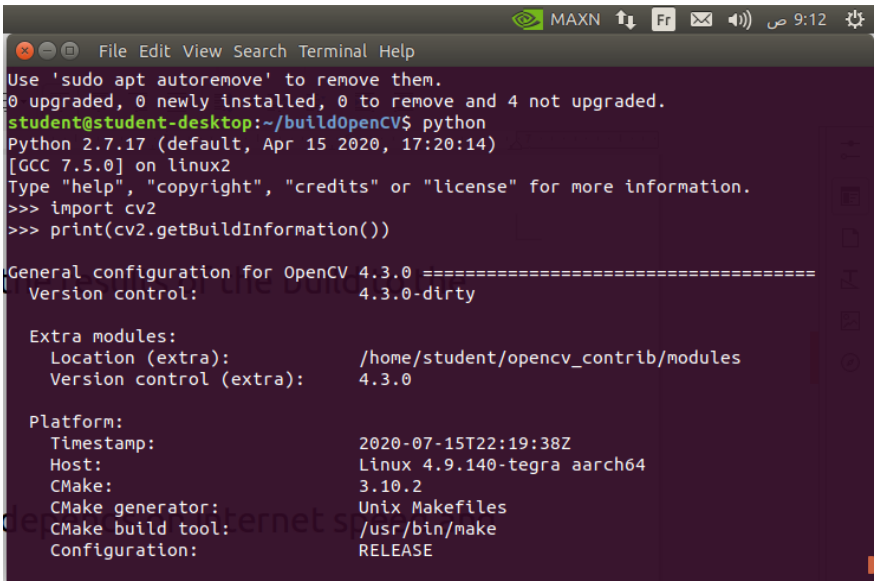

Figure II-35: complete build information of OpenCV

The Opencv that comes with jetpack does not contain GPU files

OPENCV location	/usr/local/include/opencv4/opencv2
OPENCV samples folder	/usr/share/opencv4/samples/
OPENCV header files	/usr/local/include/opencv4 /usr/local/cuda/include

II.7.11 How to install OpenCV on Jetson nano with cuda supports

OpenCV is the common library we use for image processing, deep learning via the DNN module, and basic display tasks.

CUDA is NVIDIA's set of libraries for working with their GPUs. Some non-deep learning tasks can actually run on a CUDA-capable GPU faster than on a CPU. Therefore, we'll install OpenCV with CUDA support, since the NVIDIA Jetson Nano has a small CUDA-capable GPU. [32]

The OpenCV installed with the Jetpack does not have CUDA supported. To use CUDA, we have to download OpenCV and build it from source.

Step 0: delete previous installation of opencv

sudo apt-get purge libopencv*

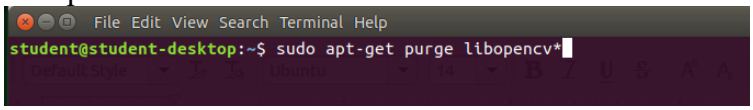

Figure II-36: delete previous installation of opencv

Step 1: increase swap size

By default, the Ubuntu 18.04 distribution of Jetson Nano comes with 2 gb of Swap memory. To increase it we need to open the terminal and type the line:

sudo apt-get install zram-config

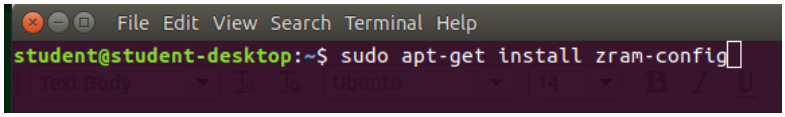

Figure II-37: Increasing swap size

The zram module, on the Jetson nano allocates by default 2gb of Swap memory, so now **we're going to extend the size to 4gb** by changing the configuration file.

By Just typing on the terminal: **sudo gedit /usr/bin/init-zram-swapping**

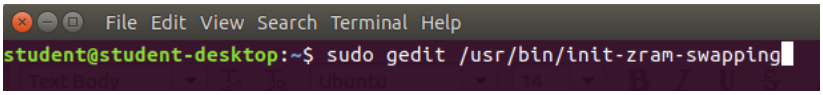

Figure II-38: Modification on the configuration file

- We replace the line: mem=$(((totalmem / 2 / ${NRDEVICES}) * 1024))

  with this line: mem=$(((totalmem / ${NRDEVICES}) * 1024))
- And then reboot.

Step 2: download and execute the Sh file that contains all instructions to install OpenCV

1- We execute this command in the terminal to clone the git repository that contains instructions to install opencv on nano :
git clone https://github.com/JetsonHacksNano/buildOpenCV

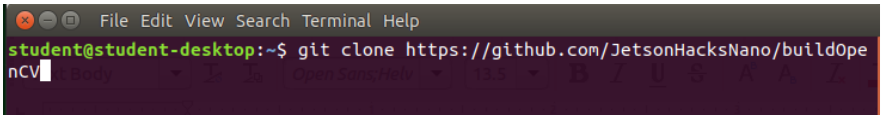

Figure II-39: clone the git repository

2- enter to buildOpenCV folder from the terminal: **cd buildOpenCV**

3- enter to the buildOpenCV folder from the explorer and open the Sh file or use the command **sudo gedit buildOpenCV.sh**

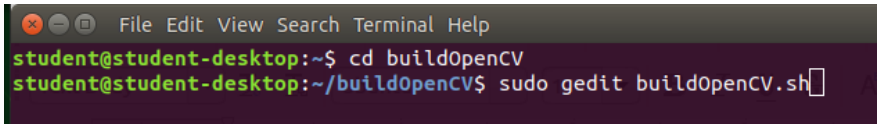

Figure II-40: explorer and open the Sh file

then we change **NUM_JOBS=$(nproc)** to **NUM_JOBS=1**

this avoids the nano to hang when building the opencv from the SD card

the sh file contains this instructions, we can install the latest version by changing the

fifth line in this file to the desired version **OPENCV_VERSION=4.x.x**

Also change build properties by modifying the Cmake build section on the Sh file

```
time cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=${CMAKE_INSTALL_PREFIX} \
    -D WITH_CUDA=ON \
    -D CUDA_ARCH_BIN=${ARCH_BIN} \
    -D CUDA_ARCH_PTX="" \
    -D ENABLE_FAST_MATH=ON \
    -D CUDA_FAST_MATH=ON \
    -D WITH_CUBLAS=ON \
    -D WITH_LIBV4L=ON \
    -D WITH_V4L=ON \
    -D WITH_GSTREAMER=ON \
    -D WITH_GSTREAMER_0_10=OFF \
    -D WITH_QT=ON \
    -D WITH_OPENGL=ON \
    -D BUILD_opencv_python2=ON \
    -D BUILD_opencv_python3=ON \
    -D BUILD_PERF_TESTS=OFF \
    -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules \
    -D INSTALL_TESTS=ON \
    -D INSTALL_C_EXAMPLES=ON \
    -D INSTALL_PYTHON_EXAMPLES=ON \
```

The instructions above allow us to build OpenCV with Cuda support and with Python and C examples taking into account different libraries (CUBLAS, LIBV4L, FAST_MATH, OPENGL…)

the source of the sh file can be found here:

(**https://www.jetsonhacks.com/2019/11/22/opencv-4-cuda-on-jetson-nano/**)

4- In the terminal we execute the Sh file and send the results of the build to the file **openCV_build.log** using this command

./buildOpenCV.sh |& tee openCV_build.log

the build will take between **2 hours to 4H s** depends on internet speed and build time

sudo ldconfig –v : Ldconfig creates, updates, and removes the necessary links and cache to the most recent shared libraries found in the directories specified on the command line, in the file (/etc/ld.so.conf), and in the trusted directories (/usr/lib and /lib). Ldconfig checks the header and file names of the libraries it encounters when determining which versions should have their links updated. Ldconfig ignores symbolic links when scanning for libraries.

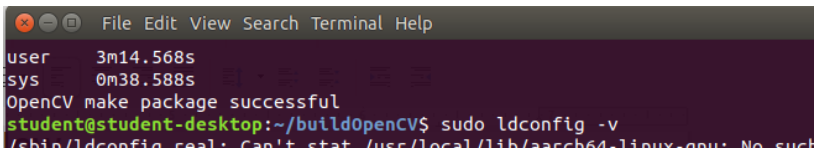

Figure II-41: updates the cache for the linker

5- We have to install Glu to run a CUDA GPU examples:

sudo apt-get install libglew-dev: (OpenGL Extension Wrangler - development environment), The OpenGL Extension Wrangler, GLEW for short, is a library that handles initialization of OpenGL extensions in a portable and simple way. Once the program initializes the library and checks the availability of extensions, it can safely call the entry points defined by the extension. Currently GLEW supports almost all the extensions found in the OpenGL extension registry (http://www.opengl.org/registry). This package contains the development documentation as well as the required header files.

```
student@student-desktop:~$ sudo apt-get install libglew-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
```

Figure II-42: Install libglew-dev deb package

- **To check opecnv build file, we type:**
  python
  import cv2
  print(cv2.getBuildInformation())

```
MAXN  Fr  9:12
File  Edit  View  Search  Terminal  Help
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 4 not upgraded.
student@student-desktop:~/buildOpenCV$ python
Python 2.7.17 (default, Apr 15 2020, 17:20:14)
[GCC 7.5.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> print(cv2.getBuildInformation())

General configuration for OpenCV 4.3.0 =====================================
  Version control:               4.3.0-dirty

  Extra modules:
    Location (extra):            /home/student/opencv_contrib/modules
    Version control (extra):     4.3.0

  Platform:
    Timestamp:                   2020-07-15T22:19:38Z
    Host:                        Linux 4.9.140-tegra aarch64
    CMake:                       3.10.2
    CMake generator:             Unix Makefiles
    CMake build tool:            /usr/bin/make
    Configuration:               RELEASE
```

Figure II-43: check opencv built file

II.7.12 Installing a Good Python IDE Environment (Visual Studio Code)

Visual Studio Code is a powerful open-source code editor developed by Microsoft. It has built-in debugging support, embedded Git control, syntax highlighting, code completion, integrated terminal, code refactoring, and snippets. [33]

Visual Studio Code is cross-platform, available on Windows, Linux, and macOS.

Code-OSS: The Code - OSS repository is where Microsoft develops the open source editor upon which it build the Visual Studio Code product. it contributes source code and

manages issues in this repository. The source code in this repository is available to everyone under a standard MIT license. [33]

1- Install Curl (it will help us to install visual studio code code-oss): curl command is a tool to download or transfer files/data from or to a server using FTP, HTTP, HTTPS, SCP, SFTP, SMB and other supported protocols on Linux or Unix-like system.

sudo apt-get install curl

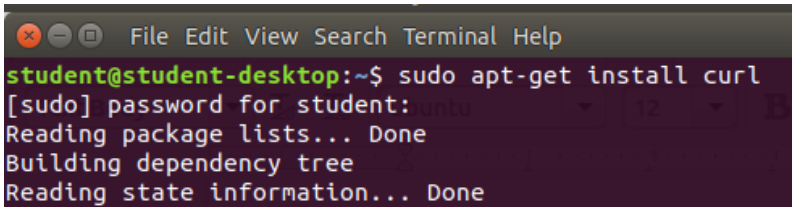

Figure II-44: Install Curl

2- Download code-oss

curl -L https://github.com/toolboc/vscode/releases/download/1.32.3/code-oss_1.32.3-arm64.deb -o code-oss_1.32.3-arm64.deb

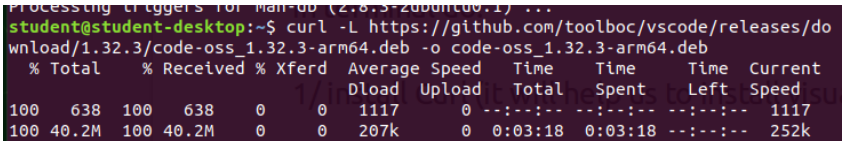

Figure II-45: Download code-oss

3- Install code-oss: **sudo dpkg -i code-oss_1.32.3-arm64.deb**

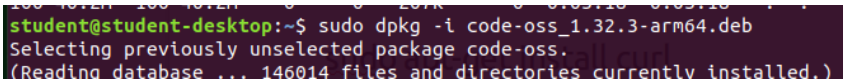

Figure II-46:Install code-oss

We see that we successfully install Code-OSS

Figure II-47: Code-OSS application

4- We lunch Code-OSS and install python extensions

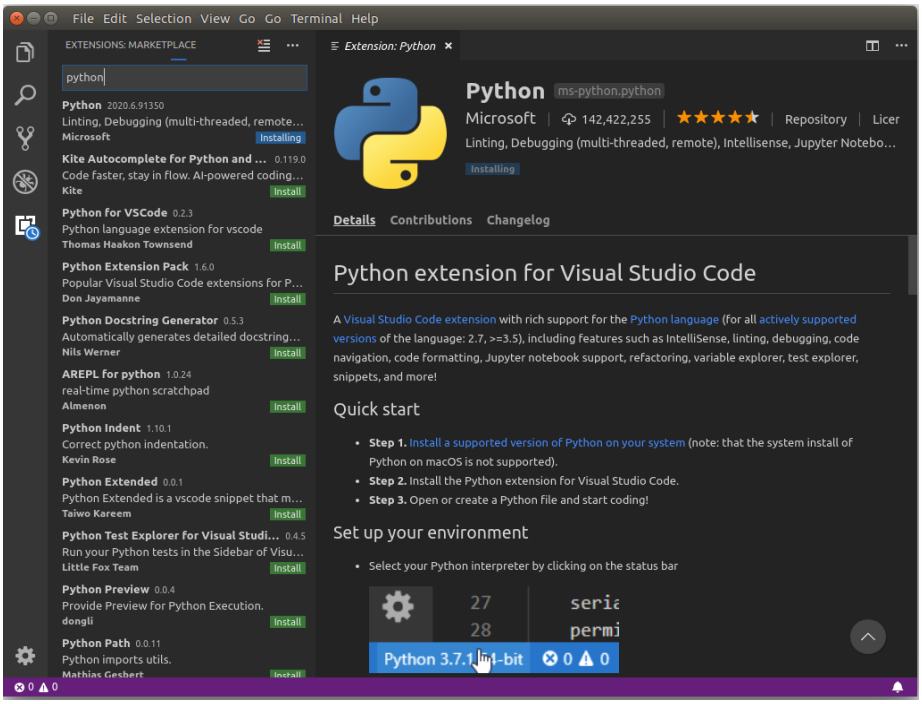

Figure II-48: install python extensions

- After that we select python3 as the default interpreter

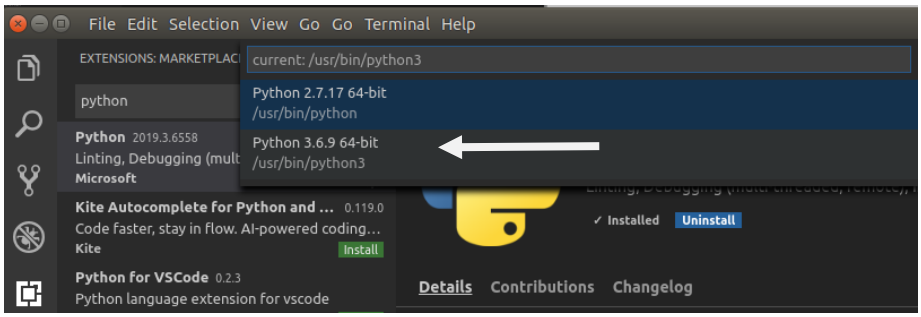

Figure II-49: select the default interpreter

5- To add auto-completion (intellisense) option we do the following:

We type Preferences: **Open Settings(JSON)** then we add these settings inside the { } :

"python.linting.pylintArgs": ["--generate-members", "--extension-pkg-whitelist=cv2"],

"python.autoComplete.extraPaths": ["/usr/lib/python3.6/dist-packages/cv2/python-3.6"]

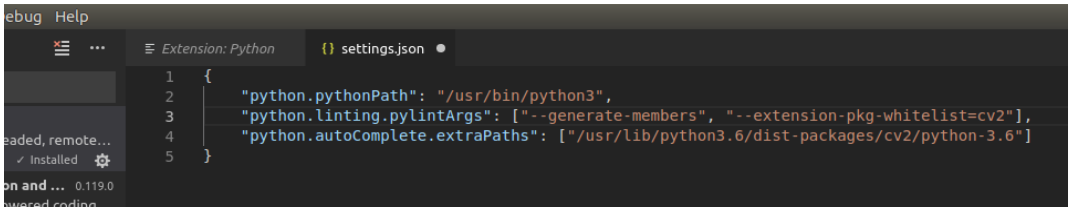

Figure II-50: add auto-completion

II.7.13 Install libraries

- **apt-get update:** Update is used to resynchronize the package index files from their sources on Ubuntu Linux via the Internet.

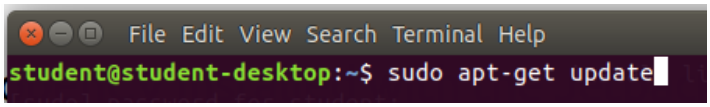

Figure II-51: Update the package index files

- **apt-get upgrade:** Upgrade is used to install the newest versions of all packages currently installed on the Ubuntu system.

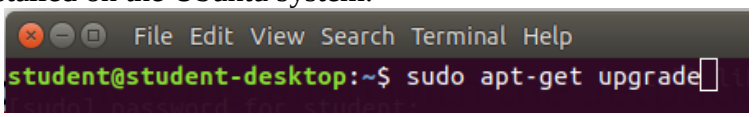

Figure II-52: Upgrade all packages installed

- **sudo apt-get install:** is used to install new packages.
- **libhdf5-serial-dev hdf5-tools:** Hierarchical Data Format 5 (HDF5) - development files - serial version library + hdf5 tools
- **zlib1g-dev:** compression library - runtime
- **libjpeg8-dev:** Independent JPEG Group's JPEG runtime library (dependency package) library
- **liblapack-dev:** Library of linear algebra routines library
- **libblas-dev:** Basic Linear Algebra Subroutines library
- **gfortran:** GNU Fortran 95 compiler

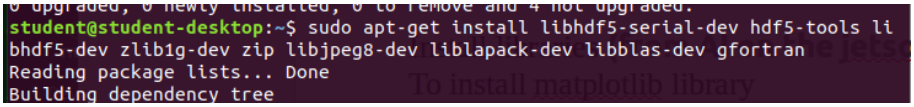

Figure II-53: install new packages

- **sudo apt-get install libcanberra-gtk-module:** translates GTK+ widgets signals to event sounds

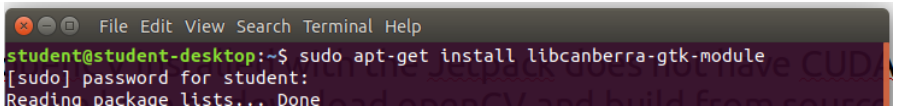

Figure II-54: Install libcanberra-gtk-module

- **Matplotlib library:** is a comprehensive library for creating static, animated, and interactive visualizations in Python. To install it we use the command:
  sudo apt-get install python3-matplotlib

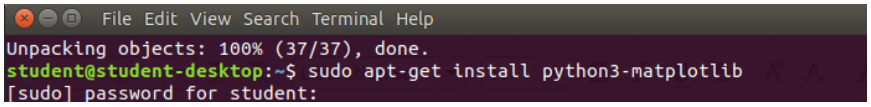

Figure II-55: Install Matplotlib library

- **sudo apt-get install python3-pip:** install pip, which is the package installer for Python.

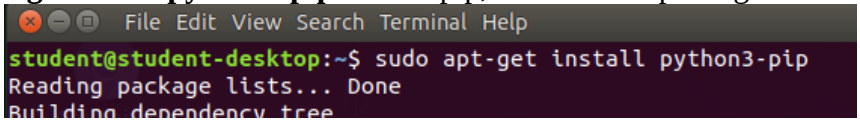

Figure II-56: Installing the package istaller for Python (pip)

- **sudo pip3 install -U pip testresources setuptools:** testresources: extensions to python unit test to allow declarative use of resources by test cases. Setuptools: is a package development process library designed to facilitate packaging Python projects by enhancing the Python standard library distutils.

II.7.14 Install the Python package dependencies

Numpy==1.16.1: numpy is the fundamental package for array computing with Python.

Future==0.17.1: future is the missing compatibility layer between Python 2 and Python 3. It allows us to use a single, clean Python 3.x-compatible codebase to support both Python 2 and Python 3 with minimal overhead.

Mock==3.0.5: mock is a library for testing in Python. It allows us to replace parts of our system under test with mock objects and make assertions about how they have been used. This package contains a rolling backport of the standard library mock code compatible with Python 3.6 and up.

H5py==2.9.0: The h5py package provides both a high- and low-level interface to the HDF5 library from Python. The low-level interface is intended to be a complete wrapping of the HDF5 API, while the high-level component supports access to HDF5 files, datasets and groups using established Python and numpy concepts. A strong emphasis on automatic conversion between Python (Numpy) datatypes and data structures and their HDF5 equivalents vastly simplifies the process of reading and writing data from Python.

Keras_preprocessing==1.0.5 : Keras Preprocessing is the data preprocessing and data augmentation module of the Keras deep learning library. It provides utilities for working with image data, text data, and sequence data.

Keras_applications==1.0.8 : Keras Applications is the applications module of the Keras deep learning library. It provides model definitions and pre-trained weights for a number of popular architectures, such as VGG16, resnet50, Xception, mobilenet, and more.

Gast==0.2.2 : A generic AST to represent Python2 and Python3's Abstract Syntax Tree(AST). GAST provides a compatibility layer between the AST of various Python versions, as produced by ast. parse from the standard ast module.

The ast module helps Python applications to process trees of the Python abstract syntax grammar. The abstract syntax itself might change with each Python release; this module helps to find out programmatically what the current grammar looks like.

Futures: This is a backport of the concurrent. Futures standard library module to Python2.

The concurrent.futures (Launching parallel tasks) module provides a high-level interface for asynchronously executing callables.

Protobuf: Protocol Buffers are Google's data interchange format

Pybind11: is a lightweight header-only library that exposes C++ types in Python and vice versa, mainly to create Python bindings of existing C++ code. Its goals and syntax are similar to the excellent Boost.

To install these dependencies, we use the command:

sudo pip3 install -U numpy==1.16.1 future==0.17.1 mock==3.0.5 h5py==2.9.0 keras_preprocessing==1.0.5 keras_applications==1.0.8 gast==0.2.2 futures protobuf pybind11

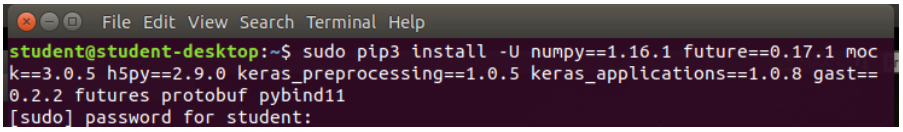

Figure II-57: Install the Python package dependencies

Install Imutils: A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, displaying Matplotlib images, sorting contours, detecting edges, and much easier with OpenCV and both Python 2.7 and Python 3. [34]

II.8 Conclusion

In this chapter, we present the materials and methods used in our study, starting by defining the embedded applications and internet of things applications, then we define GPUs and give a simplified comparison between CPUs and GPUs and we detail some embedded acronyms, then we show the different characteristics of our development board the Jetson Nano, and we also detailed step by step how to configure and install the jetpack SDK, and OpenCV with CUDA supports on this board.

Chapter III:

Implementation Results and discussion

III.1 Introduction

The ability to identify the objects present in an image or scene is one of the most basic requirements when it comes to interacting with one's environment. While it seems quite easy to humans and most animals, trying to teach computers to see and understand what they see is very difficult. The goal of object detection is to detect all instances of objects from a known class, such as people, cars or faces in an image.

In this chapter, we'll give a general overview about Single Shot Detectors and MobileNets in detail in their architectures. And then we test these DNNs trained using Caffe model. Finally, we present how to implement and deploy these models on our development board the Jetson Nano.

Our work includes training

- The SSD model on PascalVOC datasets
- The YOLO model on COCO dataset

Which are the largest and most diverse video datasets so far. They contain more pedestrian instances than previous specialized datasets, which makes it more viable for performing pedestrian detection.

III.2 Single Shot Detectors for object detection

When it comes to deep learning-based object detection there are three primary object detection methods that we'll likely encounter:

- Faster R-CNNs (Girshick et al., 2015)
- You Only Look Once (YOLO) (Redmon and Farhadi, 2015)
- Single Shot Detectors (SSDs) (Liu et al., 2015)

III.2.1 Faster R-CNNs

Faster R-CNNs are likely the most "heard of" method for object detection using deep learning; however, the technique can be difficult to understand, hard to implement, and challenging to train.

Furthermore, even with the "faster" implementation R-CNNs (where the "R" stands for "Region Proposal") the algorithm can be quite slow, on the order of 7 FPS. [35]

III.2.2 YOLO: Real-Time Object Detection

You Only Look Once: Unified, Real-Time Object Detection proposed by Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi (2015). Many improvements proposed to the first model, which was combined in the newer YOLOv2 and YOLOv3 versions. [35]

III.2.2.1 What is YOLO?

YOLO is a network for object detection. The object detection task consists in determining the location on the image where certain objects are present, as well as classifying those objects. YOLO model has several advantages over classifier-based systems. It looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation unlike systems like R-CNN, which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN. We reframe the object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. [35]

III.2.2.2 The Predictions Vector

The first step to understanding YOLO is how it encodes its output. The input image is divided into an S x S grid of cells. For each object that is present on the image, one grid cell is said to be "responsible" for predicting it. That is the cell where the center of the object falls into. [35]

Each grid cell predicts B bounding boxes as well as C class probabilities. The bounding box prediction has 5 components: (x, y, w, h, confidence). The (x, y) coordinates represent the center of the box; relative to the grid cell location (remember that, if the center of the box does not fall inside the grid cell, then this cell is not responsible for it). These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size. This is an example: [35]

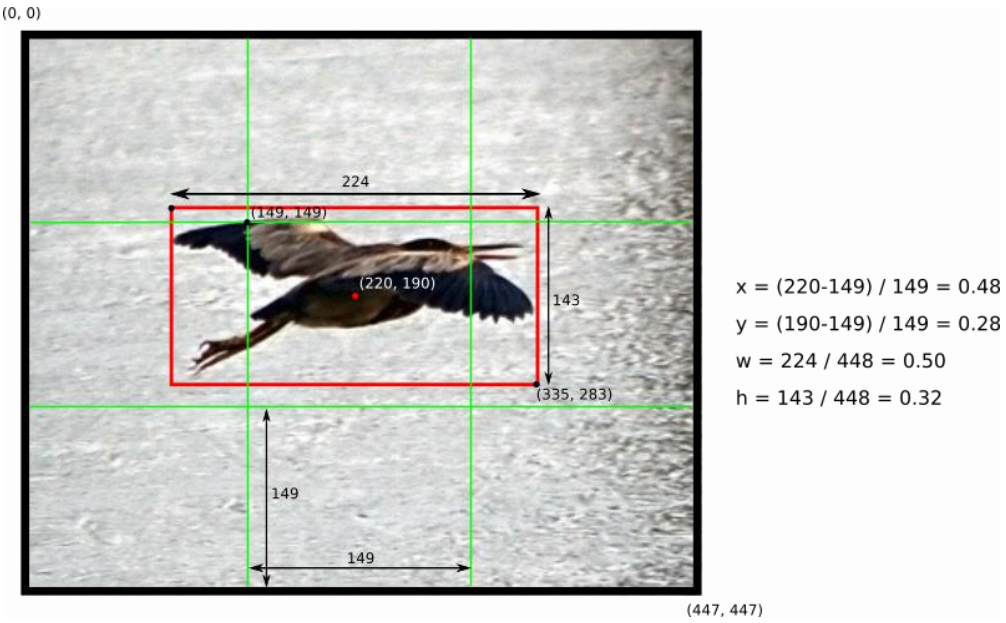

Figure III-1: e.g of calculate box coordinates in a 448x448 image with S=3

There is still one more component in the bounding box prediction, which is the confidence score.

Formally we define confidence as the multiplication of object probability with the intersection over union IOU (described below) between the predicted box and the ground truth.

$$\Pr(Object) * IOU(pred, truth)$$

If no object exists in that cell, the confidence score should be zero. Otherwise, we want the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth.

The confidence reflects the presence or absence of an object of any class.

Each grid cell makes B of those predictions, so there are in total $S \times S \times B * 5$ outputs related to bounding box predictions.

It is also necessary to predict the class probabilities, Pr(Class(i) | Object). This probability is conditioned on the grid cell containing one object. The network only predicts one set of class probabilities per cell, regardless of the number of boxes B. That makes S x S x C class probabilities in total

Adding the class predictions to the output vector, we get a: S x S x (B*5+C) tensor as output.

III.2.2.3 The YOLO Network

The network structure looks like a normal CNN, with convolutional and max pooling layers, followed by 2 fully connected layers in the end:

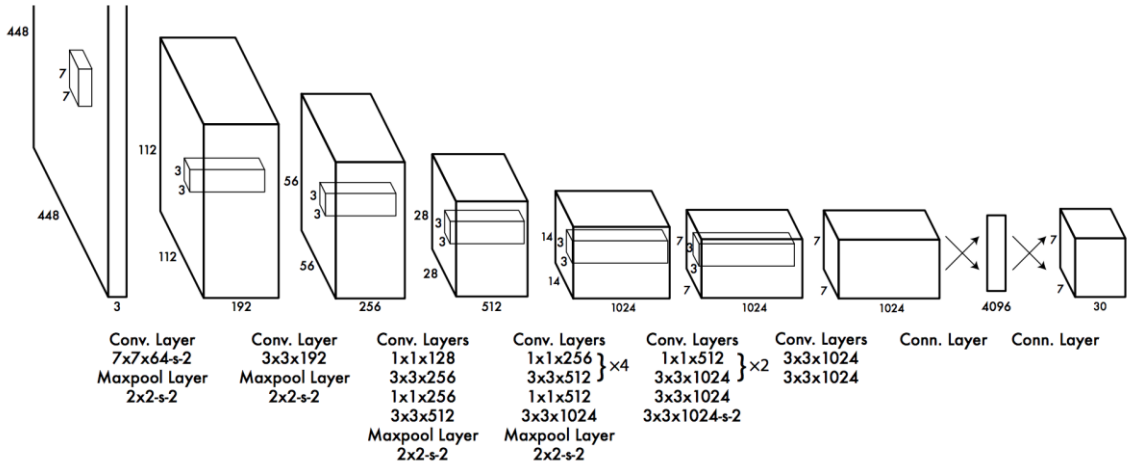

Figure III-2: The model architecture of YOLO

YOLO has 24 convolutional layers followed by 2 fully connected layers (FC). Some convolution layers use 1 × 1 reduction layers alternatively to reduce the depth of the features maps. For the last convolution layer, it outputs a tensor with shape (7, 7, 1024). The tensor is then flattened. Using 2 fully connected layers as a form of linear regression, it outputs 7×7×30 parameters and then reshapes to (7, 7, 30), i.e. 2 boundary box predictions per location. [37]

A faster but less accurate version of YOLO, called Fast YOLO, uses only 9 convolutional layers with shallower feature maps.

III.2.2.4 Loss function

YOLO predicts multiple bounding boxes per grid cell. To compute the loss for the true positive, we only want one of them to be responsible for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth. This strategy leads to specialization among the bounding box predictions. Each prediction gets better at predicting certain sizes and aspect ratios. [37]

YOLO uses sum-squared error between the predictions and the ground truth to calculate loss.

III.2.2.5 Intersect over Union (IoU)

It's a method used to evaluate how well an object detection output is related to some ground truth, the IoU is normally used during training and testing by comparing how the bounding box given during prediction overlap with the ground truth (training/test data) bounding box. [38]

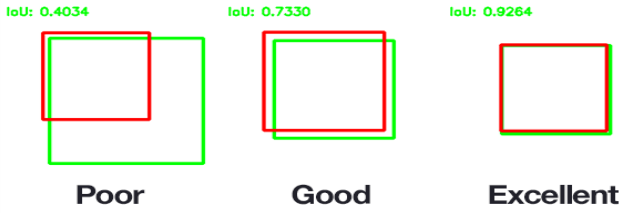

Figure III-3: e.g of computing IoU for various bounding boxes.

III.2.2.6 Benefits of YOLO

- Fast. Good for real-time processing.
- Predictions (object locations and classes) are made from one single network. Can be trained end-to-end to improve accuracy.
- YOLO is more generalized. It outperforms other methods when generalizing from natural images to other domains like artwork.

III.2.3 Single Shot Detector (SSD):

The Single Shot Detector architecture (SSD) was published in 2016 by researchers from Google (Liu et al). It presents an object detection model using a single deep neural network combining regional proposals and feature extraction. It is one of the first attempts at using convolutional neural network's pyramidal feature hierarchy for efficient detection of objects of various sizes.

III.2.3.1 Image Pyramid

SSD uses the VGG-16 model pre-trained on ImageNet as its base model for extracting useful image features. On top of VGG16, SSD adds several conv feature layers of decreasing sizes. They can be seen as a pyramid representation of images at different scales. Intuitively large fine-grained feature maps at earlier levels are good at capturing small objects and small

coarse-grained feature maps can detect large objects well. In SSD, the detection happens in every pyramidal layer, targeting at objects of various sizes. [39]

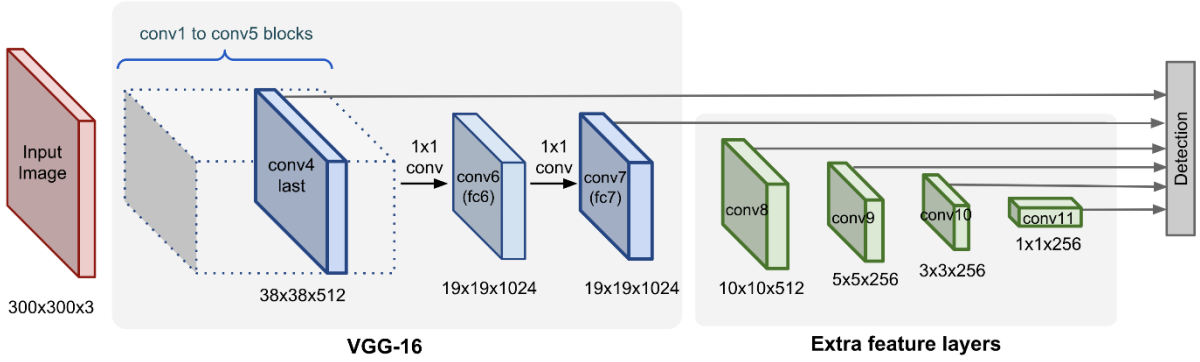

Figure III-4: The model architecture of SSD.

III.2.3.2 Workflow

SSD predicts offset of predefined anchor boxes (this is called "default boxes" in the paper) for every location of the feature map. Each box has a fixed size and position relative to its corresponding cell. All the anchor boxes tile the whole feature map in a convolutional manner.

Feature maps at different levels have different receptive field sizes. The anchor boxes on different levels are rescaled so that one feature map is only responsible for objects at one particular scale.

The Table below shows a performance comparison between different SSD and YOLO architectures, trained using COCO dataset, where mAP is *mean Average Precision* and FLOP is *floating point operations* (in Billion) and FPS is *frame per second* [37]

Table III-1: Performance on the COCO Dataset [37]

Model	Train	Test	mAP	FLOP	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244

The Table below shows a performance comparison between different SSD and YOLO architectures, trained using PASCAL VOC 2007.

Table III-2: results of PASCAL VOC 2007 [40]

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 X 600
Fast YOLO	52.7	155	1	98	448 X 448
YOLO (VGG16)	66.4	21	1	98	448 X 448
SSD300	74.3	46	1	8732	300 X 300
SSD512	76.8	19	1	24564	512 X 512
SSD300	74.6	59	8	8732	300 X 300
SSD512	76.8	22	8	24564	512 X 512

Different feature maps in the convolutional network correspond with different receptive fields and are used to naturally handle objects at different scales. As all the computation is encapsulated in a single network and fairly high computational speeds are achieved (e.g., for 300 × 300 input 59 FPS).Single Shot Detector achieves a good balance between speed and accuracy. SSD runs a convolutional network on input image only once and calculates a feature map. SSD also uses anchor boxes at various aspect ratios similar to Faster-RCNN and learns the off-set rather than learning the box. In order to handle the scale, SSD predicts bounding boxes after multiple convolutional layers. Since each convolutional layer operates at a different scale, it is able to detect objects of various scales. [40]

III.3 MobileNets: Efficient (deep) neural networks

When building object detection networks, we normally use an existing network architecture, such as VGG or ResNet, and then use it inside the object detection pipeline. The problem is that these network architectures can be very large in the order of 200-500MB.

Network architectures such as these are unsuitable for resource constrained devices due to their sheer size and resulting number of computations.

Instead, we can use MobileNets (Howard et al., 2017), another paper by Google researchers. We call these networks "MobileNets" because they are designed for resource constrained devices such as smartphones. MobileNets differ from traditional CNNs through the usage of depth wise separable convolution (Figure III-5). [41]

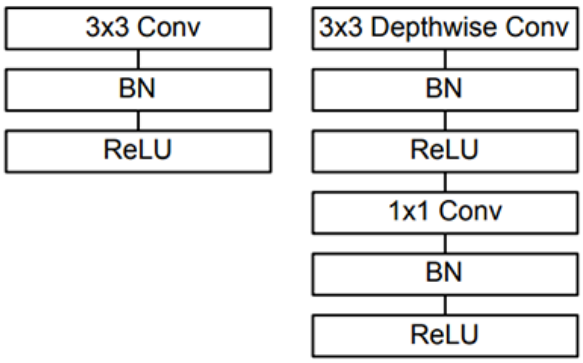

Figure III-5: MobileNets stages

(Left) Standard convolutional layer with batch normalization and ReLU.
(Right) Depthwise separable convolution with depthwise and pointwise layers followed by batch normalization and ReLU.

The general idea behind depth wise separable convolution is to split convolution into two stages:

- A 3×3 depth wise convolution.
- Followed by a 1×1 pointwise convolution.

This allows us to actually reduce the number of parameters in our network. The problem is that we sacrifice accuracy.

MobileNets are normally not as accurate as their larger big brothers. But they are much more resource efficient.

III.4 Combining MobileNets and Single Shot Detectors

If we combine both the MobileNet architecture and the Single Shot Detector (SSD) framework, we arrive at a fast, efficient deep learning-based method to object detection.

The model we'll be using in this chapter is a Caffe version of the original TensorFlow implementation by Howard et al. and was trained by chuanqi305.

The MobileNet SSD was first trained on the COCO dataset (Common Objects in Context) and was then fine-tuned on PASCAL VOC reaching 72.7% mAP (mean average precision).

We can therefore detect 20 objects in images (+1 for the background class), including airplanes, bicycles, birds, boats, bottles, buses, cars, cats, chairs, cows, dining tables, dogs, horses, motorbikes, people, potted plants, sheep, sofas, trains, and tv monitors.

III.5 Datasets

III.5.1 ImageNet dataset

The ImageNet project is a large visual database designed for use in visual object detection software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories with a typical category, consisting of several hundred images. The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The challenge uses a "trimmed" list of one thousand non-overlapping classes. [42]

III.5.2 PascalVOC dataset

Pascal VOC is a collection of datasets for object detection. The most commonly combination for benchmarking is using 2007 trainval and 2012 trainval for training and 2007 test for validation.

The PASCAL VOC datasets contained 20 object categories spread over 11,000 images. Over 27,000 object instance bounding boxes were labeled, of which almost 7,000 had detailed segmentations.

Recently, a detection challenge has been created from 200 object categories using a subset of 400,000 images from ImageNet. An impressive 350,000 objects have been labeled using bounding boxes. [43]

III.5.3 COCO dataset

Common Objects in Context (COCO) is a large-scale database that aims to enable future research for object detection, instance segmentation, image captioning, and person keypoints localization. This dataset contains 91 common object categories with 82 of them having more than 5,000 labeled instances. In total the dataset has 2,500,000 labeled instances in 328,000 images. In contrast to the popular ImageNet dataset, COCO has fewer categories but more instances per category. This can aid in learning detailed object models capable of precise 2D localization. The dataset is also significantly larger in number of instances per category than the PASCAL VOC datasets. [44]

COCO dataset features:

- Object segmentation
- Recognition in context
- 330K images (>200K labeled)
- 1.5 million object instances

III.6 Caffe model

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR)/The Berkeley Vision and Learning Center (BVLC) and community contributors.

The Caffe strategy for convolution is to reduce the problem to matrix-matrix multiplication. This linear algebra computation is highly-tuned in BLAS libraries and efficiently computed on GPU devices.

Caffe offers the model definitions, the optimization settings and pre-trained weights

To create a Caffe model we need to define the model architecture in a protocol buffer definition file (prototxt).

Caffe layers and their parameters are defined in the protocol buffer definitions for the project in caffe.proto.

III.7 Implementation

In this section we will use the MobileNet SSD + deep neural network (dnn) module in OpenCV to build our object detector.

III.7.1 Flowchart of our algorithm

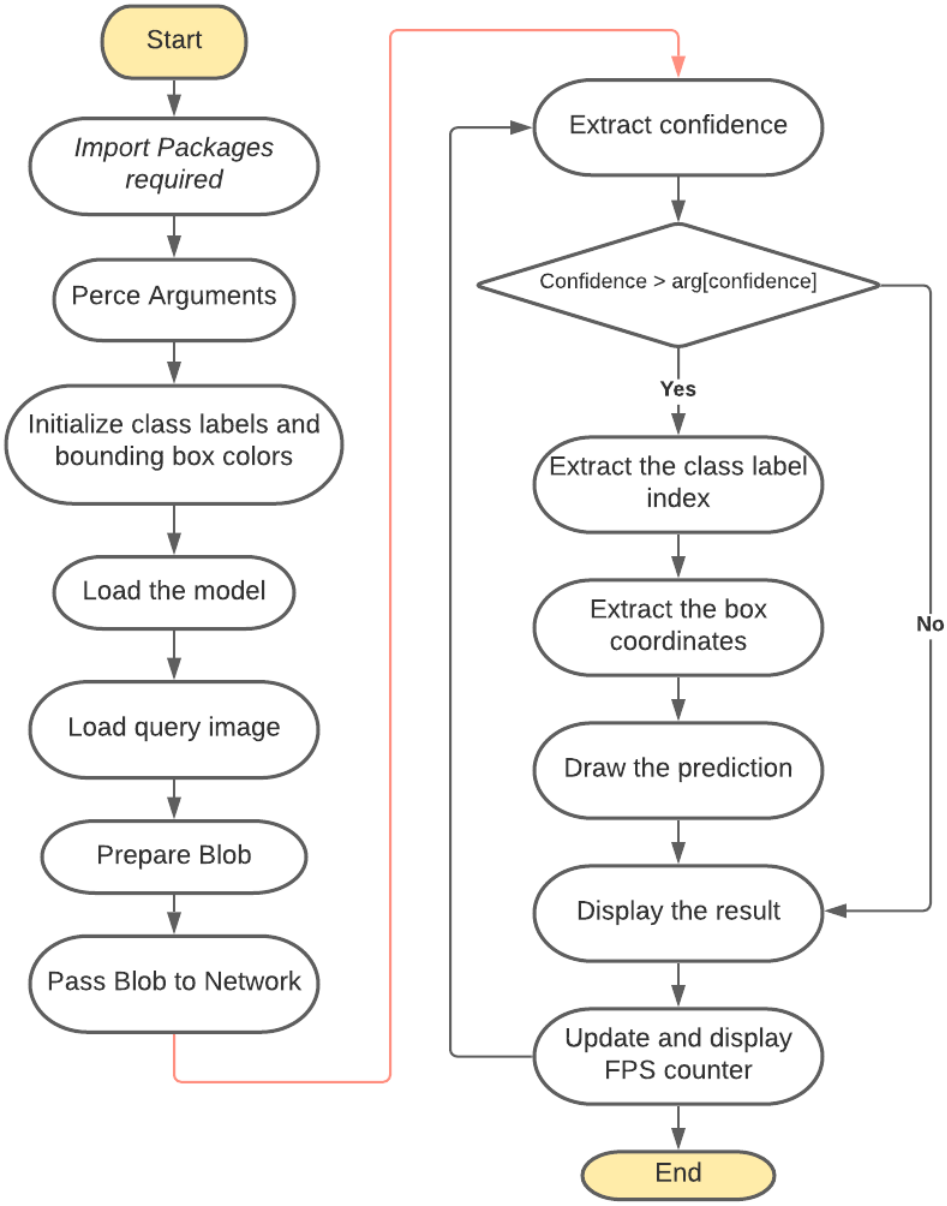

Figure III-6: Flowchart of our object detection algorithm

III.7.2 Explaining the algorithm steps

- ✓ We create a new file, name it deep_learning_object_detection.py
  - We import packages required for this script, numpy library, argparse, opencv cv2, the dnn module is included in cv2.

  Then, we parse our command line arguments

    - image: The path to the input image.
    - prototxt: The path to the Caffe prototxt file.
    - model: The path to the pre-trained model.
    - confidence: The minimum probability threshold to filter weak detections. The default is 20%.
    - Next, we initialize class labels and bounding box colors.
- ✓ Now load our model:
- ✓ Next we initialize the video stream, allow the camera sensor to warm up, and initialize the FPS counter to do that we:
    - USB Camera: Currently active, to use our USB webcam, we simply need to provide src = 0 or another device ordinal if we have more than one USB camera connected to our Nano
    - PiCamera: Currently commented out, to work with the driver on the Nano to access a PiCamera plugged into the MIPI port.
- ✓ Next, we loop over the frames from the video stream, grab frame and resize it to have a maximum width of 400 pixels, and prepare our blob, which we will feed-forward through the network. We grab our frame, extract the height and width, and calculate a 300 by 300 pixel blob from our image.
- ✓ We will pass this blob through the neural network:
  - We set the input to the network and compute the forward pass for the input, storing the result as detections.
- ✓ We loop through our detections and determine what and where the objects are in the frame:
  - We start by looping over our detections, keeping in mind that multiple objects can be detected in a single frame. We also apply a check to the confidence (i.e., probability) associated with each detection. If the confidence is high enough (i.e. above the threshold), then we'll display the prediction in the terminal as well as draw the prediction on the frame with text and a colored bounding box. For break it down line-by-line:
    - Looping through our detections, first we extract the confidence value.
    - If the confidence is above our minimum threshold, we extract the class label index and compute the bounding box around the detected object.
    - Then, we extract the (x, y)-coordinates of the box, which we will use shortly for drawing a rectangle and displaying text.
    - Next, we build a text label containing the CLASS name and the confidence.
    - Using the label, we print it to the terminal, followed by drawing a colored rectangle around the object using our previously extracted (x, y)-coordinates.
    - In general, we want the label to be displayed above the rectangle, but if there isn't room, we'll display it just below the top of the rectangle.
  - Finally, we overlay the colored text onto the frame using the y-value that we just calculated.
- ✓ The only remaining step is to display the result:

– We display the resulting output frame until a key is pressed.

III.8 Results

To execute the code, we enter the following command:

```
student@student-desktop:~/Desktop/PyPro/MyexamplesPython/deeplearningcaffe4$ pyt
hon3 dnnopencvMobileSSDvideo.py -p MobileNetSSD_deploy.prototxt.txt -m MobileNet
SSD_deploy.caffemodel
```

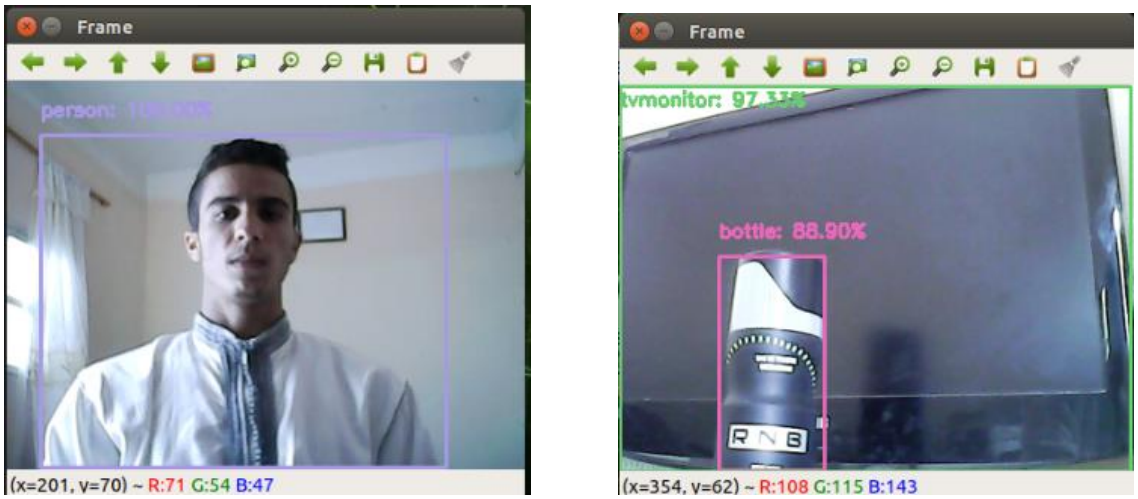

Figure III-7: SSD model object detection results

Our first result shows person, tv-monitor and a bottle recognized and detected with near-100% confidence.

Next, we will try to detect some pedestrians using SSD:

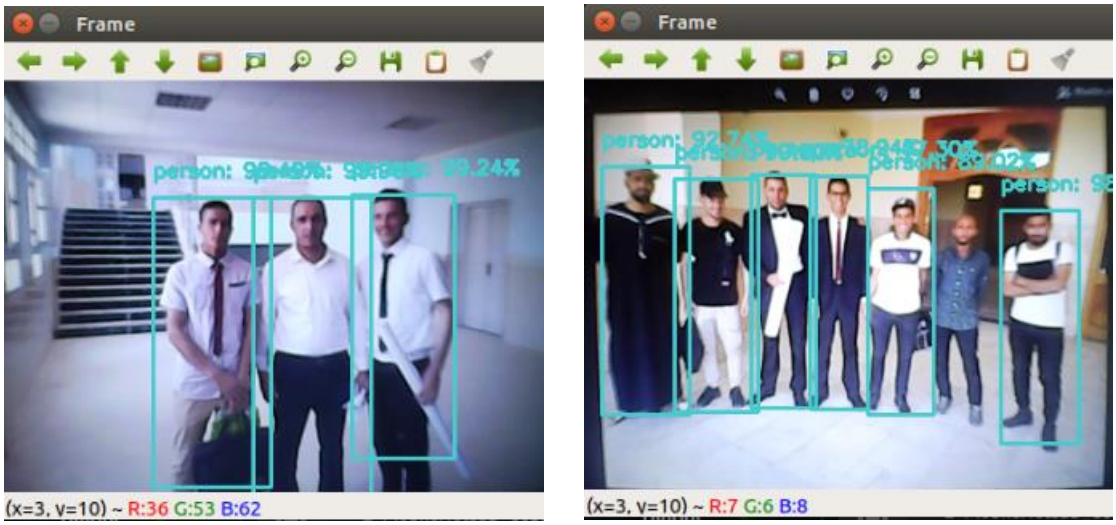

Figure III-8: SSD model pedestrians detection results

```
student@student-desktop:~/Desktop/PyPro/MyexamplesPython/deeplearningcaffe4$ pyt
hon3 dnnopencvMobileSSDvideo.py -p MobileNetSSD_deploy.prototxt.txt -m MobileNet
SSD_deploy.caffemodel
[INFO] loading model...
[INFO] starting video stream...
[ WARN:0] global /home/student/opencv/modules/videoio/src/cap_gstreamer.cpp (935
) open OpenCV | GStreamer warning: Cannot query video position: status=0, value=
-1, duration=-1
[INFO] elapsed time: 910.97
[INFO] approx. FPS: 3.92
```

Figure III-9: Informations about the SSD executed code

The FPS counter for the MobileNet SSD count 3.92fps

Now we'll try to detect objects and pedestrians using the YOLOv3. The YOLOv3 model was trained on the COCO dataset (Common Objects in Context). The next figures show the command line to execute the code and some results:

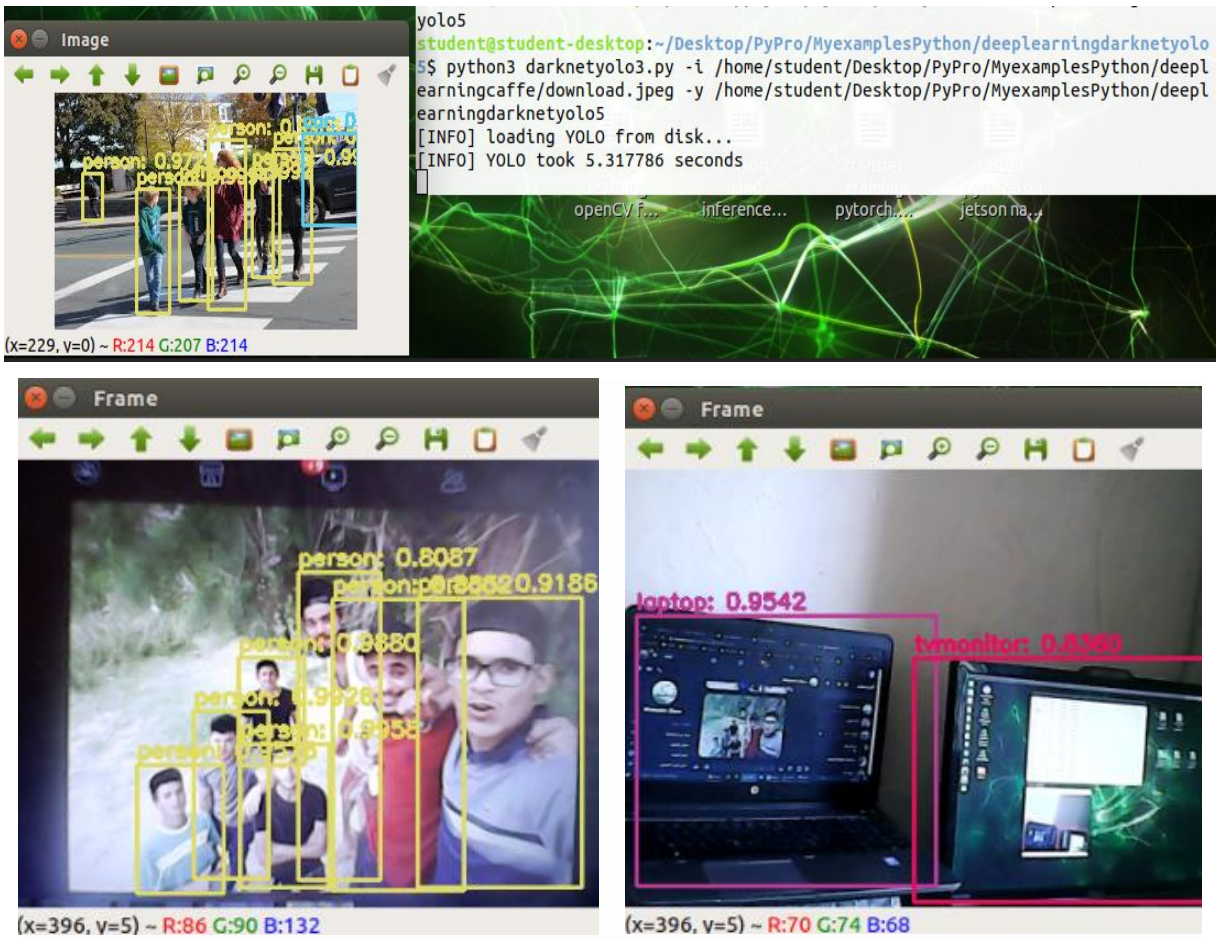

Figure III-10: YOLO model object detection results

Our results show obscured objects are recognized and detected with near-100% of confidence.

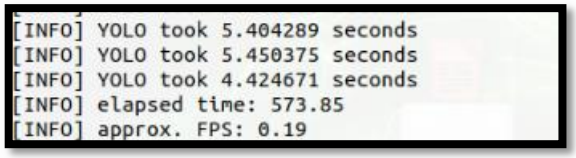

Figure III-11: Informations about the YOLO executed code

The FPS counter for the MobileNet SSD count 0.19fps. Which is much-lower and slower than the SSD model results.

III.9 Conclusion

In this final chapter we learned how to perform object detection and pedestrian detection using deep learning and OpenCV library. Specifically, we used both MobileNets + Single Shot Detectors (SSD and YOLOv3 models) along with OpenCV dnn module to detect objects and pedestrian in images and videos.

Conclusion

General conclusion

This thesis has presented the testing of object and pedestrians detection through the use of the Jetson Nano NVIDIA embedded device. According to the tests performed, Jetson Nano has a good performance, which is not the same as the most expensive models with better features, but its development kit allows the implementation of complex models to create a variety of applications. Additionally, NVIDIA's hardware platform is constantly being updated with new deep neural network libraries in an efficient and practical way, with the objective of improving the performance of computer vision tasks.

The models such as ssd-mobilenet and YOLO v3 perform well when detecting pedestrians. In addition, these models have a lower processing time, which is a useful advantage when designing applications in embedded systems.

The different modules that can be used through the JetPack permit easily to implement and develop different types of applications. In this case, the Caffe module with its different pre-trained models made it possible to optimize neural network models trained in all major frameworks.

This thesis allowed us to enrich our knowledge in image and video processing techniques through the study of the different object detection techniques. It also allowed us to study well the ML and DL, embedded systems and computer vision algorithms in Python.

We have also reinforced this information with implementation on the Jetson Nano development board, in which we validated our results with many real tests.

Abstract

Pedestrian detection is one of the important fields of object detection and computer vision. It is being applied in a wide range of applications such as video surveillance, automated driving, etc.

In this work we show how to create a real-time multiple object detection such as pedestrians and recognition application in Python on the Jetson Nano developer kit using a camera and deep learning models and libraries that Nvidia provides.

Keywords: Jetson Nano Nvidia; object detection; neural networks; deep learning; pedestrians detection; computer vision.

ملخص

من أهم مجالات رؤية الكمبيوتر هو مجال اكتشاف الكائنات والمشاة. يتم تطبيق اكتشاف المشاة في العديد من التطبيقات مثل المراقبة بالفيديو والقيادة الآلية وما إلى ذلك.

في هذا العمل، نعرض كيفية إنشاء اكتشاف كائنات متعددة في الوقت الفعلي مثل اكتشاف المشاة وتطبيق التعرف في Python على مجموعة مطوري Jetson Nano باستخدام كاميرا ونماذج التعلم العميق والمكتبات التي توفرها Nvidia.

كلمات مفتاحية: Jetson Nano Nvidia؛ كشف الكائن؛ الشبكات العصبية؛ التعلم العميق؛ كشف المشاة؛ رؤية الكمبيوتر؛ التعرف على الأشياء.

Bibliography

[1] A. Peart, "Homage to John McCarthy, the Father of Artificial Intelligence (AI)," 2 June 2017. [Online]. Available: https://www.artificial-solutions.com/blog/homage-to-john-mccarthy-the-father-of-artificial-intelligence. [Accessed 12 april 2020].

[2] "ARTIFICIAL INTELLIGENCE LATEST TECHNOLOGY," 06 03 2019. [Online]. Available: https://www.nairalovers.com/4-technology-artificial-intelligence-latest-technology/. [Accessed april 2020].

[3] MathWorks, "Introducing Machine Learning," p. 2, 2016.

[4] "Deep Learning and Traditional Machine Learning: Choosing the Right Approach," MathWorks, [Online]. Available: https://explore.mathworks.com/machine-learning-vs-deep-learning/chapter-1-129M-833I7.html. [Accessed 12 April 2020].

[5] MathWorks, "Applying Supervised Learning," 2016.

[6] "Unsupervised Learning," MathWorks, [Online]. Available: https://www.mathworks.com/discovery/unsupervised-learning.html. [Accessed 20 Apr 2020].

[7] A. Zhigalov, "Clustering and Classification," *Time series analysis in neuroscience* .

[8] MathWorks, "Introducing Deep Learning with MATLAB," *MathWorks.*

[9] F. Malik, "Neural Networks Bias And Weights," 18 May 2019. [Online]. Available: https://medium.com/fintechexplained/neural-networks-bias-and-weights-10b53e6285da. [Accessed 25 Apr 2020].

[10] Guru99, "Deep Learning Tutorial for Beginners: Neural Network Classification," [Online]. Available: https://www.guru99.com/deep-learning-tutorial.html. [Accessed May 2020].

[11] G. H. S. E. W. Yuji Roh, A Survey on Data Collection for, arXiv, 2019.

[12] A. Nguyen, "8 Best Data Labeling Tools for Machine Learning Projects," LIONBRIDGE, 15 November 2019. [Online]. Available: https://lionbridge.ai/articles/best-data-labeling-tools-machine-learning-projects/. [Accessed 18 april 2020].

[13] M. Imran, "11 Best Labelling Images And Annotation Tools in 2020," folio3, 11 May 2020. [Online]. Available: https://www.folio3.ai/blog/labelling-images-annotation-tool/. [Accessed May 2020].

[14] Humans in the Loop, "The best image annotation platforms for computer vision (+ an honest review of each)," Humans in the Loop, 2020. [Online]. Available: https://humansintheloop.org/the-best-image-annotation-platforms/. [Accessed May 2020].

[15] MahWorks, "Deep Learning," MahWorks, [Online]. Available: https://www.mathworks.com/solutions/deep-learning.html. [Accessed May 2020].

[16] MathWorks, "Convolutional Neural Network," MathWorks, [Online]. Available: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html. [Accessed May 2020].

[17] Rahul_Roy, "Best Python libraries for Machine Learning," GeeksforGeeks, 23 08 2019. [Online]. Available: https://www.geeksforgeeks.org/best-python-libraries-for-machine-learning/. [Accessed 17 08 2020].

[18] Y. Jia, "Caffe," BAIR, [Online]. Available: https://caffe.berkeleyvision.org/. [Accessed 12 Aug 2020].

[19] J. Brownlee, "A Gentle Introduction to Object Recognition With Deep Learning," 22 May 2019. [Online]. Available: https://machinelearningmastery.com/object-recognition-with-deep-learning/. [Accessed 19 Aug 2020].

[20] Wikipedia, "List of datasets for machine-learning research," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research. [Accessed 22 Aug 2020].

[21] susedefines, "Embedded Application," susedefines, [Online]. Available: https://susedefines.suse.com/definition/embedded-application/. [Accessed Aug 2020].

[22] Upasana, "Real World IoT Applications in Different Domains," [Online]. Available: https://www.edureka.co/blog/iot-applications/. [Accessed 17 Aug 2020].

[23] J.E.N.A.O. Luis Barba-Guaman, Deep Learning Framework for Vehicle and Pedestrian, electronics, 2020.

[24] A. Ayibiowu, "HARDWARE ACRONYMS: SIP, SOC, SOM, COM, SBC – WHAT ARE THEY?," 10 April 2018. [Online]. Available: https://www.electronics-lab.com/hardware-acronyms-sip-soc-som-com-sbc/. [Accessed Aug 2020].

[25] Nvidia, "Jetson FAQ," Nvidia, [Online]. Available: https://developer.nvidia.com/embedded/faq. [Accessed Aug 2020].

[26] elinux, "Jetson Zoo," elinux, 12 August 2020. [Online]. Available: https://elinux.org/Jetson_Zoo. [Accessed 22 Aug 2020].

[27] "Tensorflow," Software package details, [Online]. Available: https://openpath.software/software_package/tensorflow.html. [Accessed Aug 2020].

[28] Nvidia, "Jetson SDKs," Nvidia, [Online]. Available: https://developer.nvidia.com/embedded/jetson-sdks. [Accessed Aug 2020].

[29] Nvidia, "Jetson Nano Developer Kit," Nvidia, 2019. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed Apr 2020].

[30] Raspberry Pi, "Camera Module V2," Raspberry Pi, [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/. [Accessed june 2020].

[31] Nvidia, "User Guide," *JETSON NANO Developper Kit*, 2020.

[32] A. Rosebrock, "How to configure your NVIDIA Jetson Nano for Computer Vision and Deep Learning," 25 March 2020. [Online]. Available: https://www.pyimagesearch.com/2020/03/25/how-to-configure-your-nvidia-jetson-nano-for-computer-vision-and-deep-learning/. [Accessed 12 April 2020].

[33] "How to Install Visual Studio Code on Ubuntu 20.04," Linuxize, 1 May 2020. [Online]. Available: https://linuxize.com/post/how-to-install-visual-studio-code-on-ubuntu-20-04/. [Accessed Aug 2020].

[34] @jrosebr1, "imutils," 18 Aug 2019. [Online]. Available: https://github.com/jrosebr1/imutils. [Accessed 14 Aug 2020].

[35] A. Rosebrock, "Object detection with deep learning and OpenCV," pyimagesearch, 11 September 2017. [Online]. Available: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/. [Accessed 12 Aug 2020].

[36] M. Menegaz, "Understanding YOLO," 16 March 2018. [Online]. Available: https://hackernoon.com/understanding-yolo-f5a74bbc7967. [Accessed 21 aug 2020].

[37] J. Hui, "Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3," medium, 18 Mar 2018. [Online]. Available: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088. [Accessed 14 Aug 2020].

[38] A. Rosebrock, "Intersection over Union (IoU) for object detection," pyimagesearch, 7 November 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/. [Accessed 14 Aug 2020].

[39] S.-t. X. Zhong-Qiu Zhao, Object Detection with Deep Learning: A Review, arXiv 1807.05511v2, 2019.

[40] J. Hui, "Object detection: speed and accuracy comparison (Faster R-CNN, R-FCN, SSD, FPN, RetinaNet and YOLOv3)," medium, 28 Mar 2018. [Online]. Available: https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359. [Accessed 12 Aug 2020].

[41] M. Sanatkar, "Analysis and Applications of Multi-Scale CNN Feature Maps," Towards Data Science, 15 April 2020. [Online]. Available: https://towardsdatascience.com/analysis-and-applications-of-multi-scale-cnn-feature-maps-a6804bbac8. [Accessed 14 Augg 020].

[42] Divyanshupy, "OBJECT DETECTION USING MOBILENET," github, 19 March 2019. [Online]. Available: https://github.com/Divyanshupy/MobileNet-Object-Detection. [Accessed 12 Aug 2020].

[43] wikipedia, "ImageNet," wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/ImageNet. [Accessed 12 Aug 2020].

[44] Gluon, "Prepare PASCAL VOC datasets," Gluon, [Online]. Available: https://gluon-cv.mxnet.io/build/examples_datasets/pascal_voc.html. [Accessed 15 Aug 2020].

[45] H. Caesar, J. Uijlings and V. Ferrari, "COCO-Stuff: Thing and Stuff Classes in Context".

[46] O. Nicholas, "Machine Learning Works," 20 Mei 2019. [Online]. Available: https://quantumcomputingtech.blogspot.com/2019/05/machine-learning-works.html. [Accessed dec 2019].

[47] A. Rosebrock, "imutils 0.5.3," 18 Aug 2019. [Online]. Available: https://pypi.org/project/imutils/. [Accessed 15 sep 2020].

[48] MathWorks, "Machine Learning with MATLAB," *The MathWorks, Inc*, 2018.