



Demonstration of AI vision State-of-the-Art algorithms on CPU+GPU and FPGA Platforms

Presented by Vignesh - 20021065



A Special Thanks to

Dr David Cho - My supervisor for guiding me throughout the year with the project flow.

Mr Wayne San - A visiting teacher at UNNC who takes Summer class on Advance VLSI for helping me to get the FPGA board and providing suggestions on how to set up the simulation platforms for AI vision.

TABLE OF CONTENTS

Objectives of the Project

Covers the given Objectives of this thesis

01



Achievements of the Project

Covers the things we have achieved through this thesis project

02



Background Info

A brief introduction to AI, ML, CNN Technique, State of the Art CNN algorithm and Hardware Accelerators

03



04

Results and Inference

This section contains the results obtained from our simulations



05

Future Works

The latest advances in this technology and possible future works for improvements



06

Conclusion

Conclusions and may ask questions if you have any.



ABOUT THE PROJECT

The project is about implementing AI Vision Algorithm for Autonomous Vehicles.

The algorithm uses State-of-the-Art CNN models which performs classification and localization when a sample input image is fed into the Network with real time inference capabilities.

Objectives of this project


The primary objectives of this project are:

- ➔ To build a working platform for training Darknet TinyYolo Object Detection Algorithm
- ➔ To Set up the environment on FPGA board and deploy our object detection model

Our Achievements

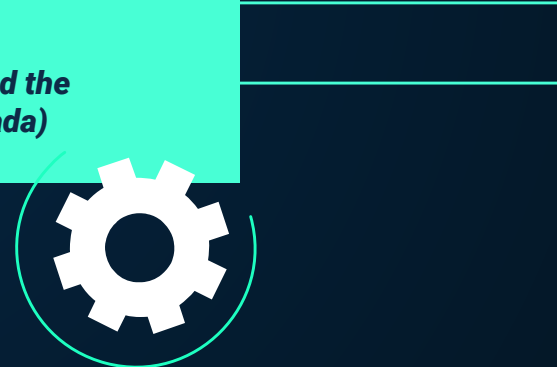
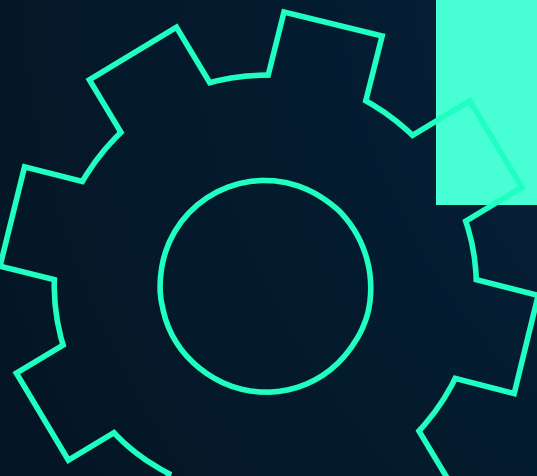

The outcomes achieved through this projects are:

- ➔ **Demonstrated the inference of the latest Darknet YOLOv3 state of the art object detection algorithm**
- ➔ **Demonstrated the training simulations of the famous state of the art algorithms including ResNet, GoogLeNet, NiN, VGG and AlexNet**
- ➔ **Demonstrated the simulation of our AI models on cloud based FPGA platform using Xilinx Vitis AI on nimblex FPGA as a Service (FAAS).**
- ➔ **We have demonstrated deep learning computer vision algorithms using dedicated GPU hardware on google cloud for training the data and obtaining inference. We have also given the inference of the ML models on laptops as the integrated graphics cards on laptops were not suitable for training**



"Assuming that the computer industry can keep producing better hardware, I think business trends that deals with massive data will take us further through big conceptual ideas. I think this breakthrough will happen when we fully understand the brain and replicate its capabilities in an artificial method."

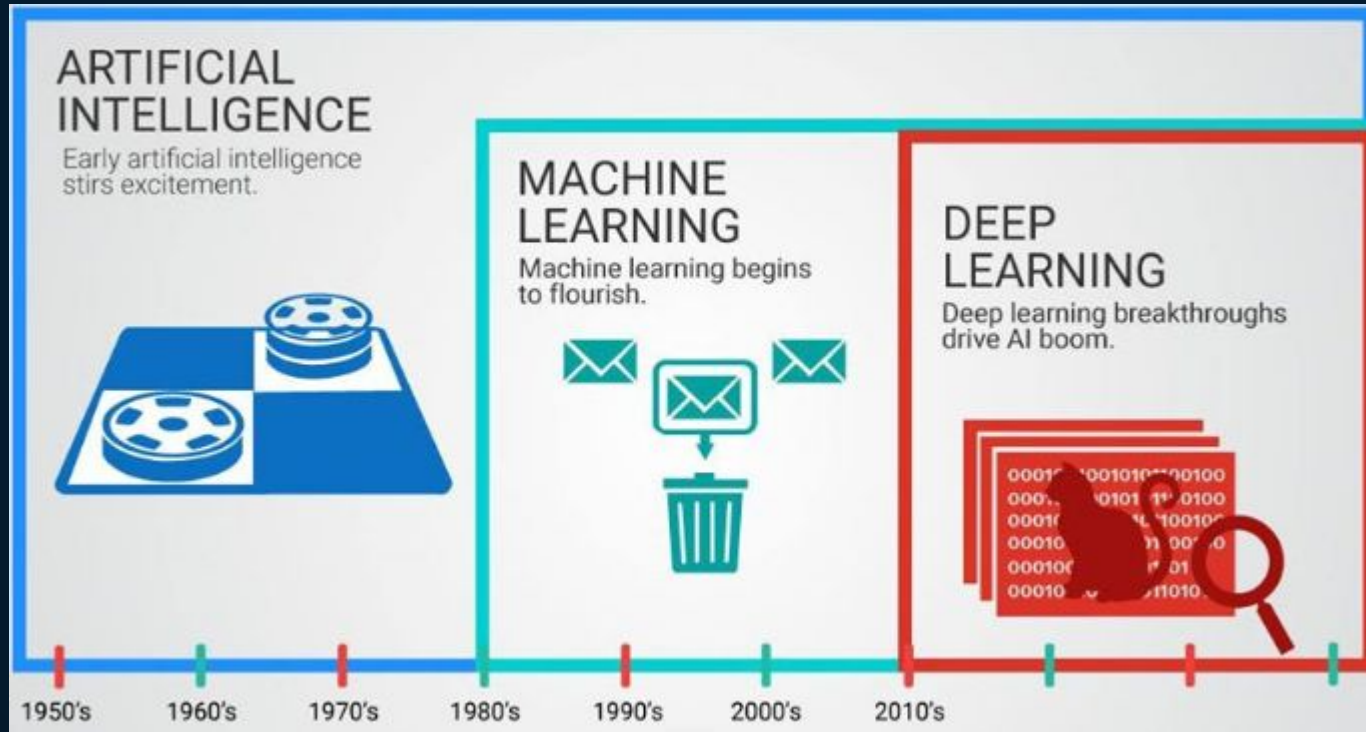
—Geoffrey Hinton, 2016 (*The father of AI and the cofounder of Vector Institute at UoT, Canada*)



Background on AI/ML/DL and how these automate things



AI vs ML vs DL



Artificial Neural Network (ANN)

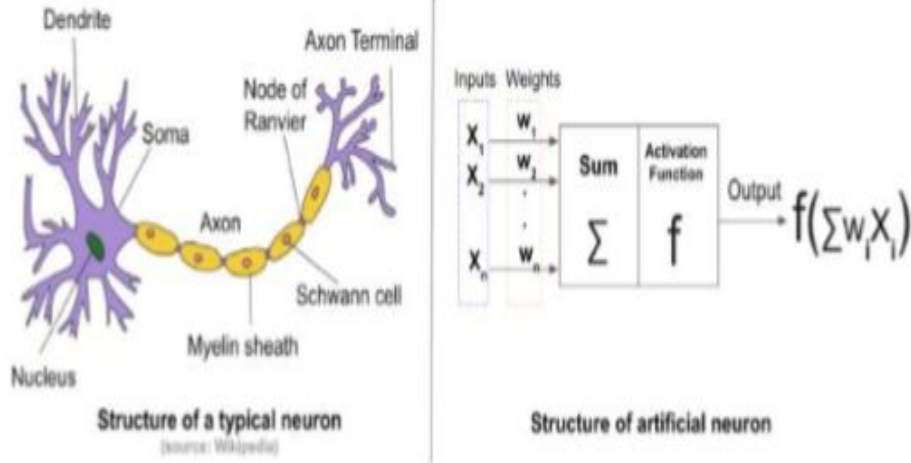


FIG 1: REAL NEURON VS ANN [img src:

https://en.wikipedia.org/wiki/neuron#/media/file:blausen_0657_multipolarneuron.png]

- An ANN is a computational model replica of a real neural
- A simple neural network model may have more than thousands of ANN connections within the architecture
- The model can be programmed to activate the desired neuron based on the received input signal

Components of ANN

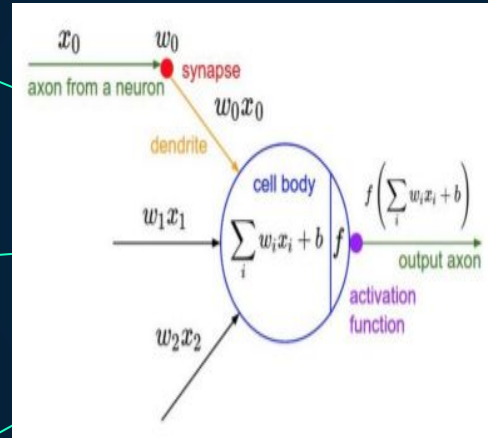
Connections - They carry information signals that are processed by the neurons



Propagation function and network input- the output from a neuron and multiplying with a weight value. They convert the vector value of input signals into a scalar value..



Activation Function- happen when the neuronal input signal crosses a given threshold value and thereby making the neuron sensitive and starts to fire.



Output function - Calculates the values which are transferred to other neurons



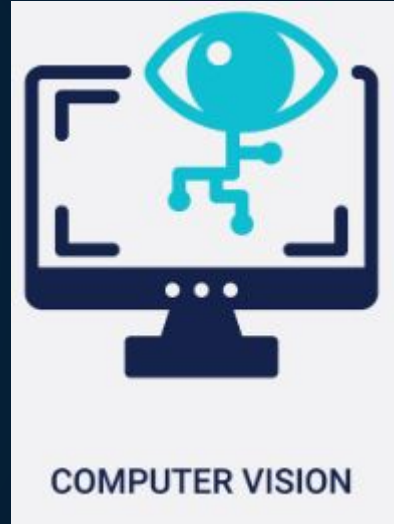
Learning Strategy: Every model of the neural network has specialized learning strategies. It's an algorithm which determines how the neural networks are trained and placed to get the desired output.





Computer vision: Computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos.

It seeks to understand and automate tasks that the human visual system can do.



How Computer Vision Works



Acquiring the image

Images, even large sets, can be acquired in real time through video, photos or 3D technology for analysis.



Analyzing the image

Deep learning models automate much of this process, but the models are often trained by first being fed thousands of labeled or pre-identified images.



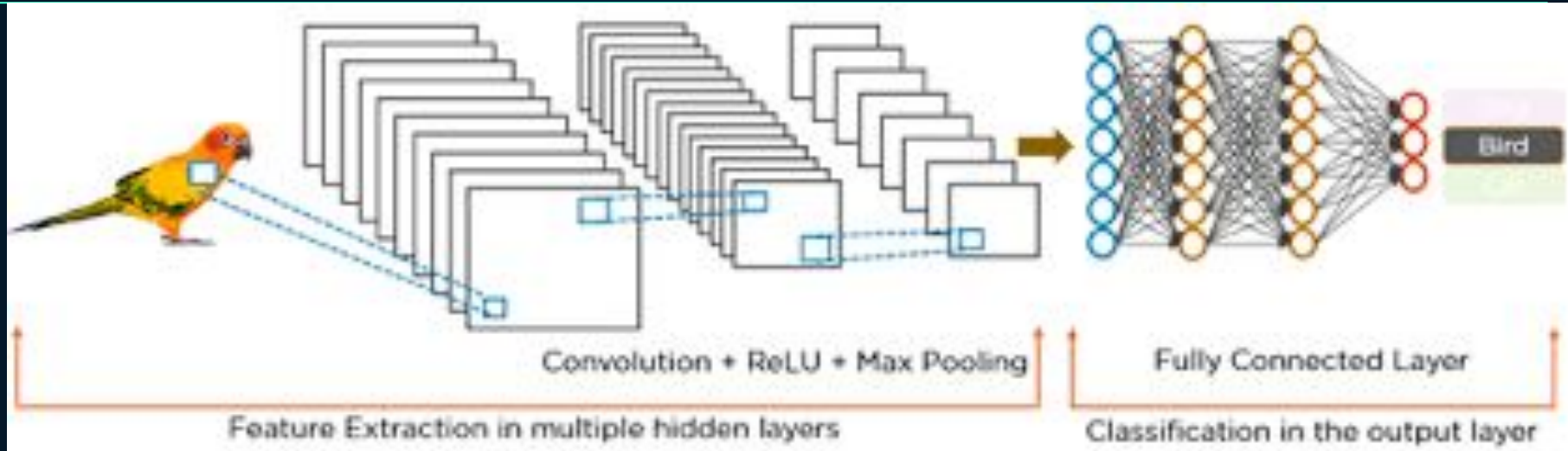
Applying the insights

The final step is the interpretive step, where the deep learning model is deployed to score new image or video feed.

Convolutional Neural Network

- The idea of CNN was introduced long back for hand-written digit recognition by [LeCun et al., 1998].
- CNN, a class of DNN, is a feed-forward ANN that are applied to analyzing computer vision.
- A CNN model has many hidden layers within network where each layer performs some mathematical modelling to the input signal model
- May contains filters, pooling layers, activation layers, fully connected layers and loss layers.

Convolutional Neural Network



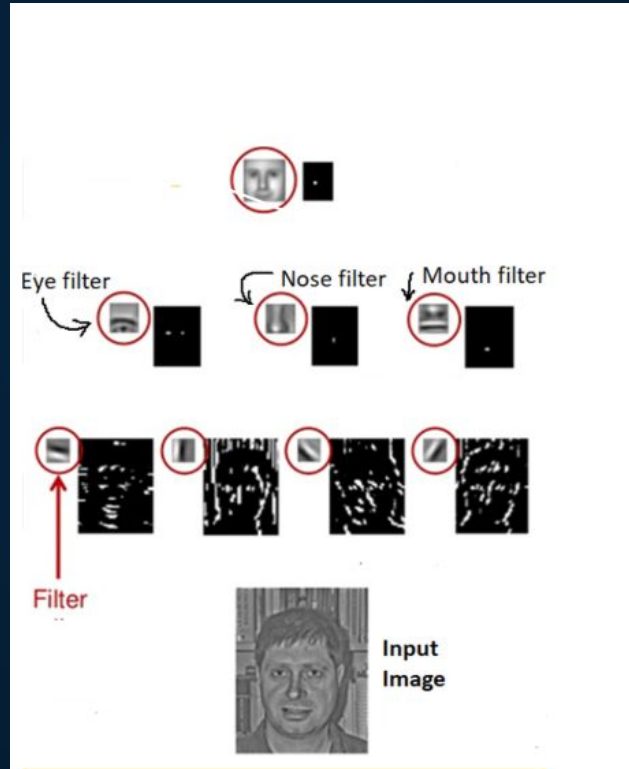
- A class is a output prediction. A set of common features in mapped into a single class.
- Here the bird is a class. The features of this class includes: Colour, position of eye and beak, shape and size of legs, nose and tails
- The dimension of the layer contains feature extraction filters.

CNN Layers

Each layers here performs some pixel computation.

Usually multiplies with the input pixels, hence the word convolutional layers.

These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image



Edge detection

$$\text{Image} * \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \text{Edge Map}$$

Sharpen

$$\text{Image} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \text{Sharpened Image}$$

Examples of kernel Filters for CNNs.

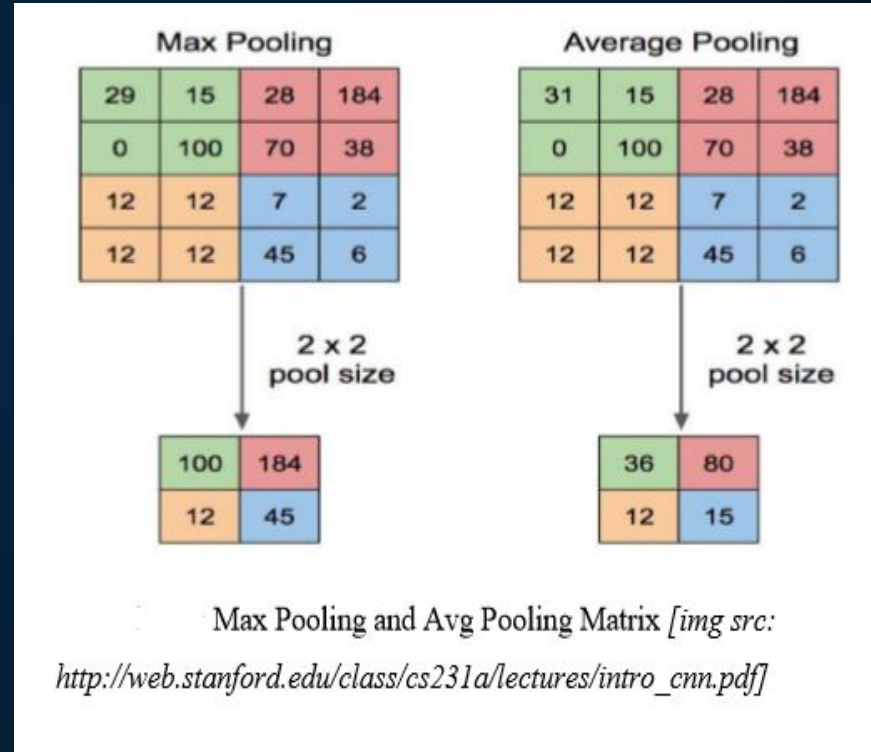
CNN Layers

Pooling Layers: To reduce the spatial size of the representation to reduce the parameter counts which reduces the computational complexity.

Basically we select a pooling size to reduce the amount of the parameters

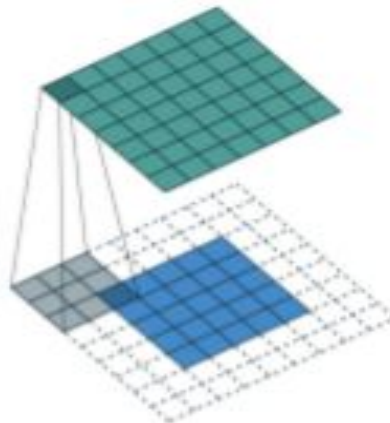
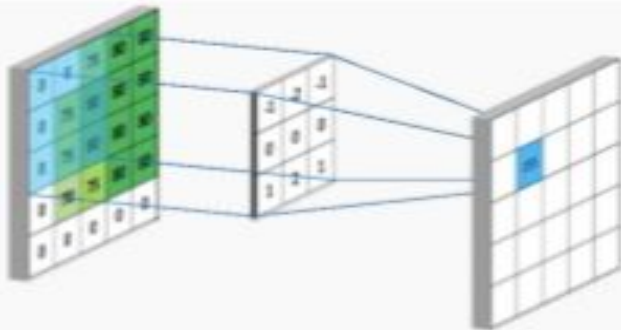
Accepts a volume of size $W1 \times H1 \times D1$ and usually requires 2 hyperparameters namely: Their Spatial extent F , and the Stride S . It then produces a volume of size $W2 \times H2 \times D2$ where

$$W2 = (W1 - F)/S + 1 \parallel H2 = (H1 - F)/S + 1 \parallel D2 = D1$$

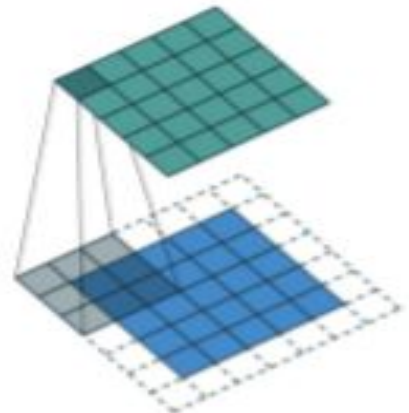


CNN Layers

Example of how convolutions are applied to images using kernel filters.



Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



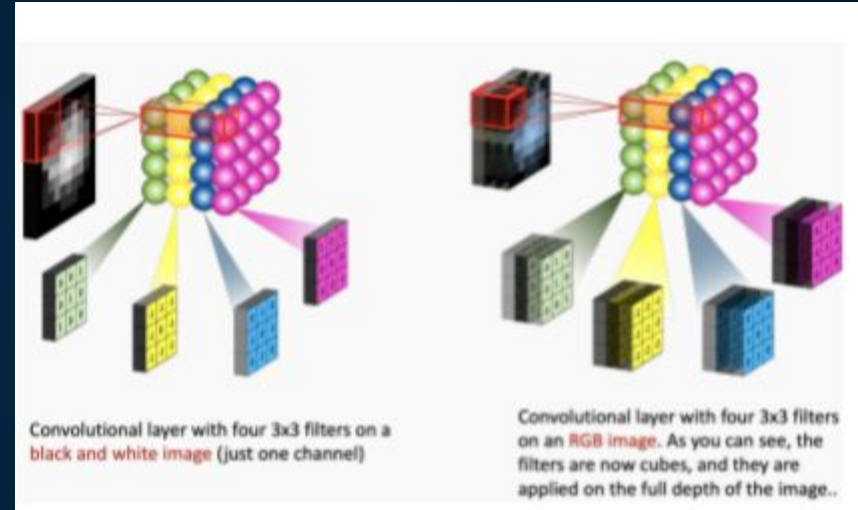
Same padding. Ensures that the output has the same size as the input.

How do we connect our filters together?

Each filter is convolved with the entirety of the 3D input cube but generates a 2D feature map. Because we have multiple filters, we end up with a 3D output: one 2D feature map per filter

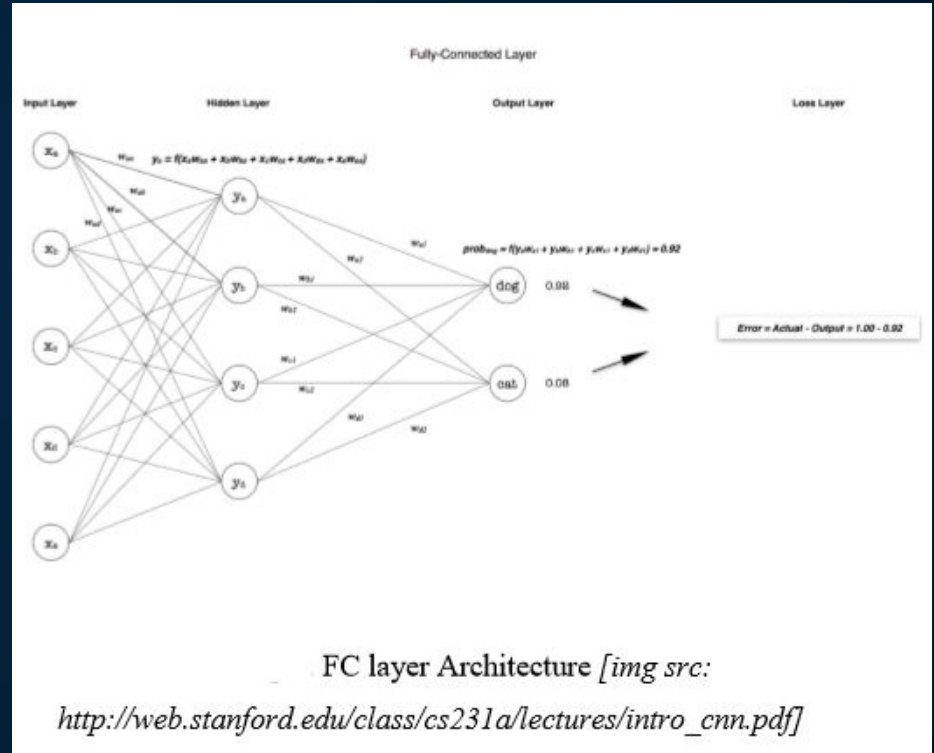
The feature map dimension can change drastically from one convolutional layer to the next: we can enter a layer with a $32 \times 32 \times 16$ input and exit with a $32 \times 32 \times 128$ output if that layer has 128 filters.

Convolving the image with a filter produces a feature map that highlights the presence of a given feature in the image.

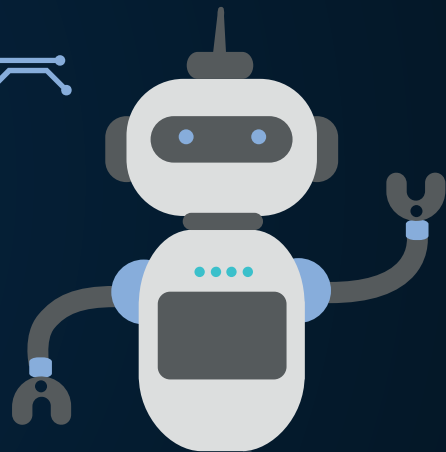


How do we connect our filters together?

- Here we enter into the ANN architecture
- The objective of this layer is to take the result of conv and pooling layer and map it into a label.
- FC layer has 3 parts: input layer with the previous layer's flattened array of values, hidden layer and output layer which is a single value result from the neural network.
- It is then passed to a activation function to give confidence
- The loss layer gives how good our current value is, and keeps improving at each iterations



Training a Neural Network



Training a Neural Network

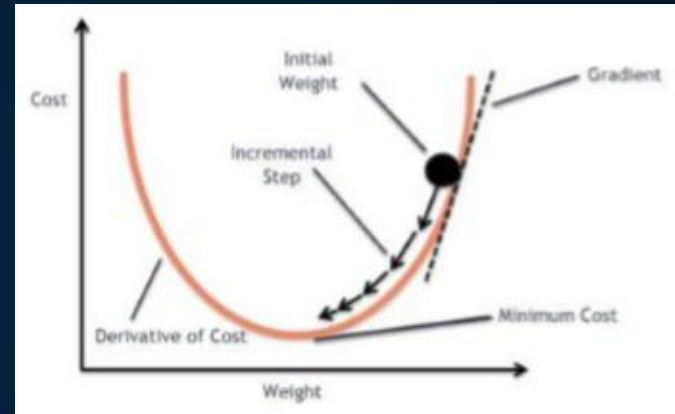
We start off training the neural network by giving some random weights value to the network.

Perform the loss function at the end and let the algorithm know how much error is present and tune it to minimize the error.

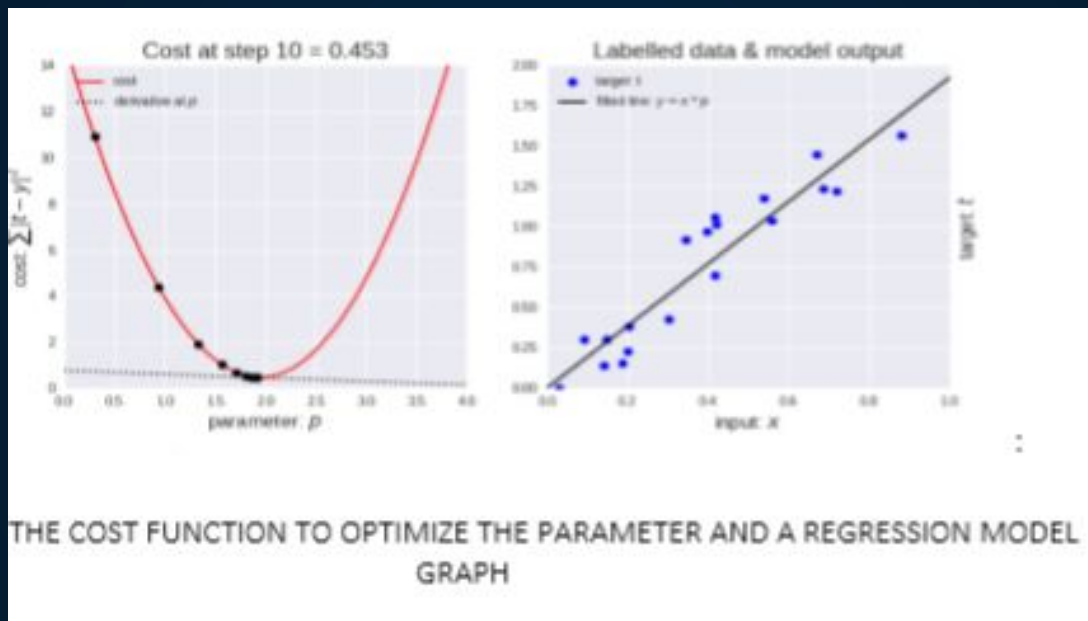
In this process, we are trying to minimize the loss as it is an efficient way of optimization. Many optimization methods can be used.

Popular ones include gradient-based optimization

We define a constant Learning rate value at this stage.
 $Lr = 0.01$ units is preferred



Training a Neural Network

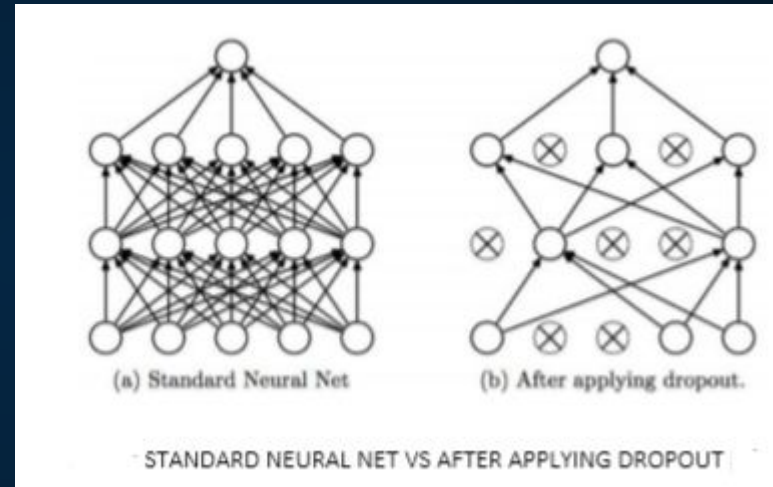


Training a Neural Network

Lets add a dropout function here and get rid of neurons whose parameters are not necessary for training a particular class.

Dataset Samples: Dataset contains data that is fed into the network for training purpose. The selection of dataset consists of 3 steps. A sample may also be called an instance, an observation, an input vector, or a feature vector

--Gather--Clean--Annotate--Feed--

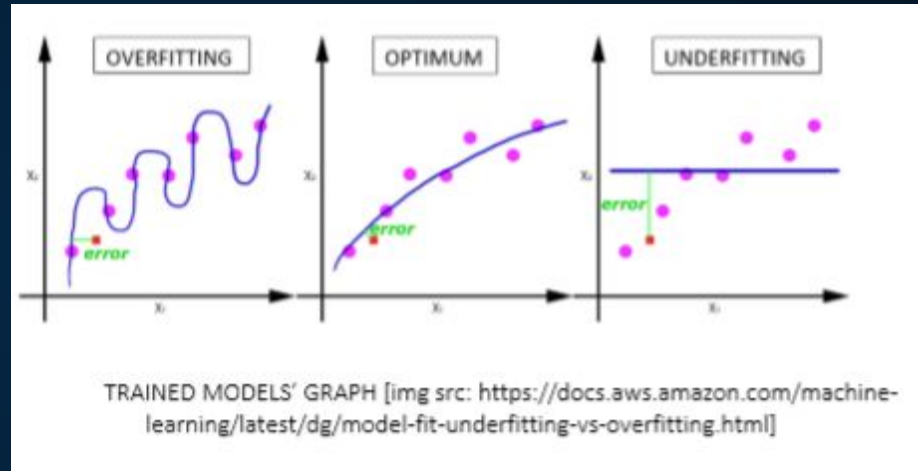


Training a Neural Network

Batch Size: It is a hyperparameter that defines the number of samples that must be processed before updating, or tuning, the network parameters.

It is chosen as a power of 2 to better fit with the hardware runtime as the memory distribution in hardware is a power of 2.

Epochs: Epochs are a hyperparameter term used to denote how many times the same dataset is fed into the network for tuning.



Summery of a CNN Model

- Starts with an input image • applies many different filters to it to create a feature map
- Applies a ReLU function to increase non-linearity
- Applies a pooling layer to each feature map
- Flattens the pooled images into one long vector.
- Inputs the vector into a fully connected artificial neural network.
- Processes the features through the network. The final fully connected layer provides the “voting” of the classes that we’re after.
- Trains through forward propagation and backpropagation for many, many epochs. This repeats until we have a well-defined neural network with trained weights and feature detectors.

State-of -the-Art CNN Model



Pre Defined Learning Strategy

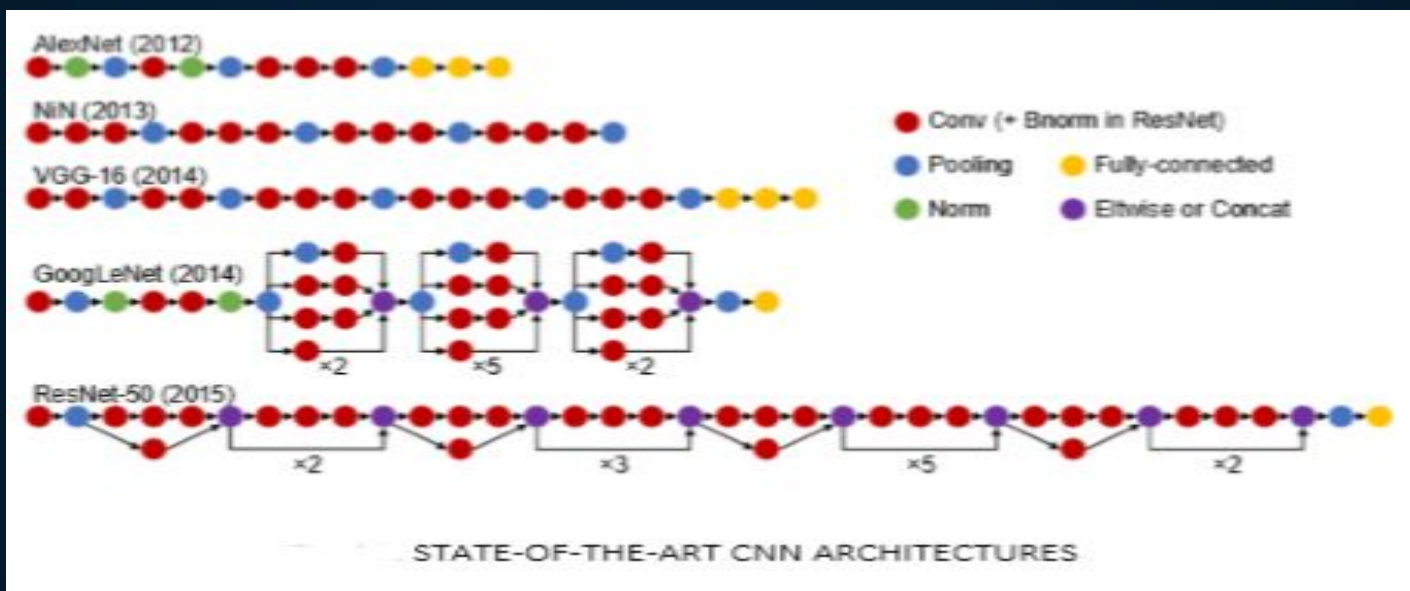


Easy to Train



Accurate Prediction Results

Popular State-of-the-Art Models



Popular State-of-the-Art Models

YOLOv3 OBJECT DETECTION ALGORITHM

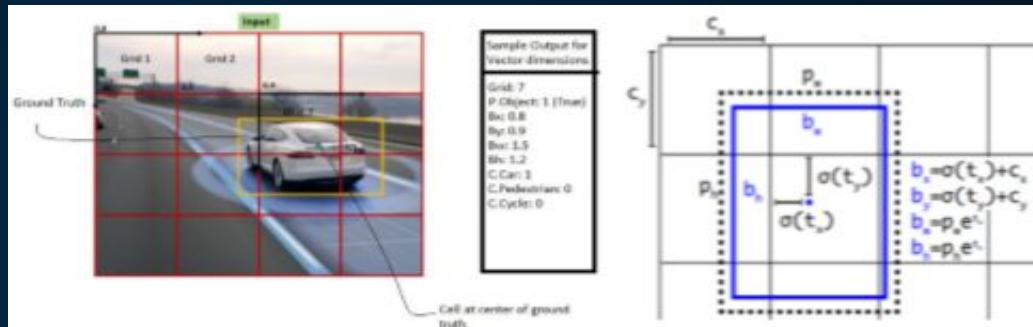
The state of the art object detection models perform some localization techniques along with classification. This enables the model to find and locate multiple images within a image.

We will be demonstrating the YOLO object detection which works under bounding box and object detection.

First, the image is cut into various equally spaced grids ($S \times S$ size), and for each grid, it detects bounding box and the corresponding classes by an output vector of dimension $S \times S \times (C+5)$, C is the number of classes. Then a single convolutional layer is used to perform the classification for that grid. This layer has a typical loss function as an error between the label vector and output activation. All the classes are generally fed forward into the same conv layer to define the parameters

Yolo Object Detection Technique

y	px	Gives TRUE (1) if the object present in an $S \times S$ grid, otherwise 0
	bx	These are responsible for creating the boundary box within the grid. x, y defines the midpoint of the object
	by	
	bh	
	bw	
	$c1$	These are responsible for detecting the classes. $c1, c2, c3$ are the three classes. n classes can be represented as $c1, c2, \dots, cn$ in the table. When a certain class is detected, its vector cn will be 1.
	$c2$	
	$c3$	



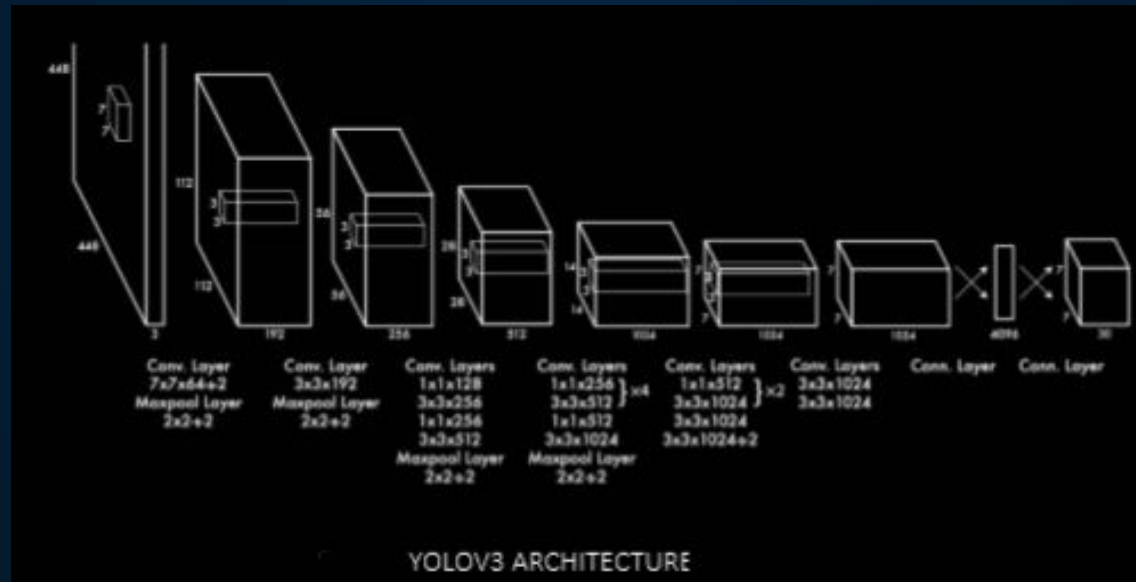
MAPPING THE BOUNDARY BOX ON SAMPLE IMAGE[33]

$$scores = p_c * \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{78} \\ c_{79} \\ c_{80} \end{pmatrix} = \begin{pmatrix} p_c c_1 \\ p_c c_2 \\ p_c c_3 \\ \vdots \\ p_c c_{78} \\ p_c c_{79} \\ p_c c_{80} \end{pmatrix} = \begin{pmatrix} 0.12 \\ 0.13 \\ 0.44 \\ \vdots \\ 0.07 \\ 0.01 \\ 0.09 \end{pmatrix}$$

Find the max \Rightarrow **score:** 0.44
box: (b_x, b_y, b_h, b_w)
class: $c = 3$ ("car")

Yolo has 106 layers in total. Yolo uses some 1x1 conv layers to reduce the depth of the reduction layers. The last 7x7x1024 deep conv layer is then flattened and fed into those 2 FC layers and the last conv layer of output 7x7x30.

It uses 3 loss function. One is for classification, another is for localization to define the error between the predicted boundary box and ground truth, and the confident loss for object detection.

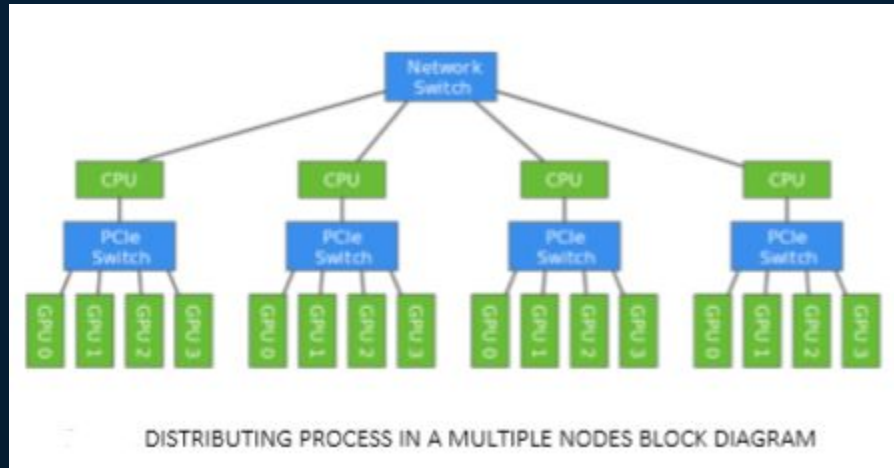


Hardware as AI Accelerators

- CNN network needs large datasets and requires huge runtime, maybe weeks or even months if the network is trained for more classes
- To address this, the use of dedicated hardware came to existence. AI Accelerators are specialized hardware designed to run algorithms that require parallel operations to reduce the time complexity problems.
- Computers traditionally use a special-purpose processor to rely on along with a general CPU. This includes GPU, video cards, TPU and FPGAs
- DL Frameworks: Used to easily deploy algorithm in distributed computer memory. Tensorflow, Tensorflow Lite, Tensorflow JS, Darknet, CUDA and MxNet.



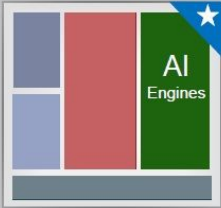

GPU as Accelerators

GPUs are always preferred for HPC implementation which enable users to deploy their training algorithm to multiple cores, more than 100 cores and are also capable of utilizing the entire HPC portal. A typical system with these configurations generally requires various components like RAM, motherboard, and dedicated power supply units to power up.



FPGA as Accelerators

- **Field Programmable Gate Arrays are reconfigurable hardware units with mighty memory distribution and connection capabilities which can be individually customized by the developer**
- **As the ML models keep changing adding new features and increasing efficiency of the inference every year, engineers, especially in the self-driving cars industry, opt for FPGA to constantly develop their algorithms without having to change the architecture of the model**
- **The CNN can be developed by HLS or HDS from scratch or we can make use of easy to deploy AI SDKs for FPGA like Xilinx Vitis-AI Suite, Intel FPGA SDK or Mentor Graphics EDA**
- **The Xilinx xDNN processing engine that are used in Xilinx Alveo Data Center accelerator cards is a high-performance energy-efficient deep neural network accelerator and outperforms many common CPU and GPU platforms today in raw performance and power efficiency for real-time inference workloads**

	<div> <div>CPU (Sequential)</div> <div>GPU (Parallel)</div> </div>		UltraScale+™	Custom ASIC
	 			
SW Programmable	✓	✓	✓	✓
HW Adaptable	—	—	✓	—
Workload Flexibility	✓	✓	✓	—
Throughput vs. Latency	—	—	✓	✓
Device / Power Efficiency	—	—	✓	✓

Vitis: Unified Software Platform

Domain-specific
development
environment

Vitis accelerated
libraries

Vitis core
development kit

OpenCV
Library

BLAS
Library

Finance
Library

TensorFlow

Vitis AI

Coming soon...

FFmpeg

Vitis Video

Partners
Genomics,
Data Analytics,
And more

Compilers

Analyzers

Debuggers

Xilinx runtime library

Vitis target platform



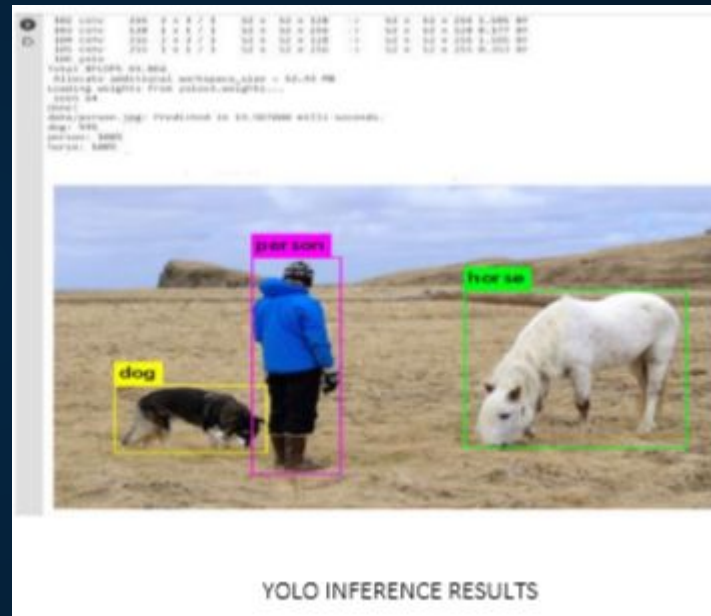
Results and Inference



Results and Inference - On PC

We have performed an object detection algorithm on the personal laptop with intel i5 CPU+2G Nvidia MX150 integrated GPU+8G RAM. We were able to get real-time video inference at less than 20 FPS. The model claims to provide a better result with higher system configurations.

75 convolution layers and the remaining 31 includes pooling, FCN and other layers.



Results and Inference

More Inferences:



YOLO INFERENCE RESULT ON OUR CUSTOM SAMPLE DATA

Results and Inference - On Cloud FPGA

We have performed an inference simulation for a set of sample inputs and the output is predicted with prediction probabilities. A set of 10 sample images has been fed and each image may have multiple objects in it. The YOLO algorithm can predict all the objects in a given image with prediction probability

```
File Edit View Terminal Help
Image : 002.png
top[0] prob = 0.286247 name = traffic light, traffic signal, stoplight
top[1] prob = 0.222938 name = street sign
top[2] prob = 0.082811 name = cab, hack, taxi, taxicab
top[3] prob = 0.063879 name = obelisk
top[4] prob = 0.049742 name = triumphal arch

Image : 015.jpg
top[0] prob = 0.746657 name = alexander
top[1] prob = 0.129758 name = hay
top[2] prob = 0.037174 name = Old English sheepdog, bobtail
top[3] prob = 0.022547 name = ram, tup
top[4] prob = 0.008295 name = ox

Image : 010.png
top[0] prob = 0.329588 name = streetcar, tram, tramcar, trolley, trolley car
top[1] prob = 0.121248 name = cab, hack, taxi, taxicab
top[2] prob = 0.094429 name = trolleybus, trolley coach, trackless trolley
top[3] prob = 0.057274 name = traffic light, traffic signal, stoplight
top[4] prob = 0.057274 name = triumphal arch
```

YOLO OBJECT DETECTION INFERENCE RESULT ON ALVEO_U50

Results and Inference

All the simulations here are performed using ipython notebooks on google collab platform. We made use of Fashion MNIST dataset with 10 classes.

The models are capable of detecting more classes, but since it takes more time, we limited to 10.

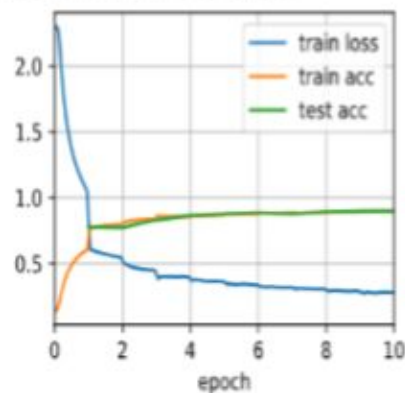
All the models were trained using 64 batches, 10 epochs and with learning rate at 0.01.

Results and Inference - On Google Colab

```
X = np.random.uniform(size=(1, 1, 224, 224))
net.initialize()
for layer in net:
    X = layer(X)
    print(layer.name, 'output shape:\t', X.shape)
```

```
conv0 output shape: (1, 96, 54, 54)
pool0 output shape: (1, 96, 26, 26)
conv1 output shape: (1, 256, 26, 26)
pool1 output shape: (1, 256, 12, 12)
conv2 output shape: (1, 384, 12, 12)
conv3 output shape: (1, 384, 12, 12)
conv4 output shape: (1, 256, 12, 12)
pool2 output shape: (1, 256, 5, 5)
dense0 output shape: (1, 4096)
dropout0 output shape: (1, 4096)
dense1 output shape: (1, 4096)
dropout1 output shape: (1, 4096)
dense2 output shape: (1, 10)
```

loss 0.280, train acc 0.899, test acc 0.899
1975.0 examples/sec on gpu(0)

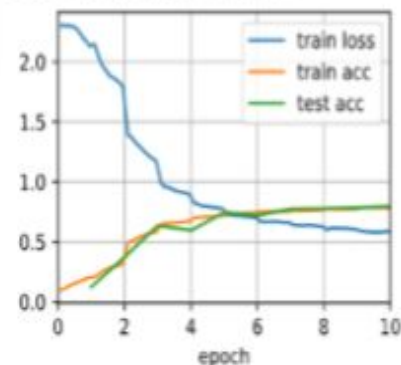


ALEXNET TRAINING RESULT GRAPH

```
X = np.random.uniform(size=(1, 1, 224, 224))
net.initialize()
for layer in net:
    X = layer(X)
    print(layer.name, 'output shape:\t', X.shape)
```

```
sequential1 output shape: (1, 96, 54, 54)
pool0 output shape: (1, 96, 26, 26)
sequential2 output shape: (1, 256, 26, 26)
pool1 output shape: (1, 256, 12, 12)
sequential3 output shape: (1, 384, 12, 12)
pool2 output shape: (1, 384, 5, 5)
dropout0 output shape: (1, 384, 5, 5)
sequential4 output shape: (1, 10, 5, 5)
pool3 output shape: (1, 10, 1, 1)
flatten0 output shape: (1, 10)
```

loss 0.584, train acc 0.781, test acc 0.781
3032.4 examples/sec on gpu(0)



NIN TRAINING RESULT GRAPH

Results and Inference - On Google Colab

```
net.initialize()
X = np.random.uniform(size=(1, 1, 224, 224))
for blk in net:
    X = blk(X)
    print(blk.name, 'output shape:\t', X.shape)
```

```
sequential1 output shape: (1, 64, 112, 112)
sequential2 output shape: (1, 128, 56, 56)
sequential3 output shape: (1, 256, 28, 28)
sequential4 output shape: (1, 512, 14, 14)
sequential5 output shape: (1, 512, 7, 7)
dense0 output shape: (1, 4096)
dropout0 output shape: (1, 4096)
dense1 output shape: (1, 4096)
dropout1 output shape: (1, 4096)
dense2 output shape: (1, 10)
```

loss 0.172, train acc 0.937, test acc 0.928
1826.9 examples/sec on gpu(0)

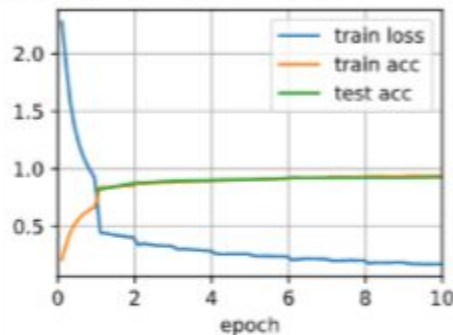


FIG 34: VGG-11 TRAINING RESULT GRAPH

```
X = np.random.uniform(size=(1, 1, 96, 96))
net.initialize()
for layer in net:
    X = layer(X)
    print(layer.name, 'output shape:\t', X.shape)
```

```
sequential0 output shape: (1, 64, 24, 24)
sequential1 output shape: (1, 192, 12, 12)
sequential2 output shape: (1, 480, 6, 6)
sequential3 output shape: (1, 832, 3, 3)
sequential4 output shape: (1, 1024, 1, 1)
dense0 output shape: (1, 10)
```

loss 0.330, train acc 0.876, test acc 0.876
733.9 examples/sec on gpu(0)

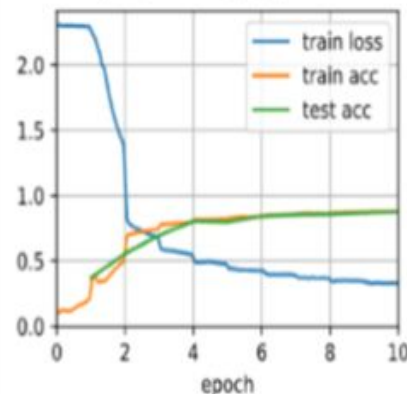
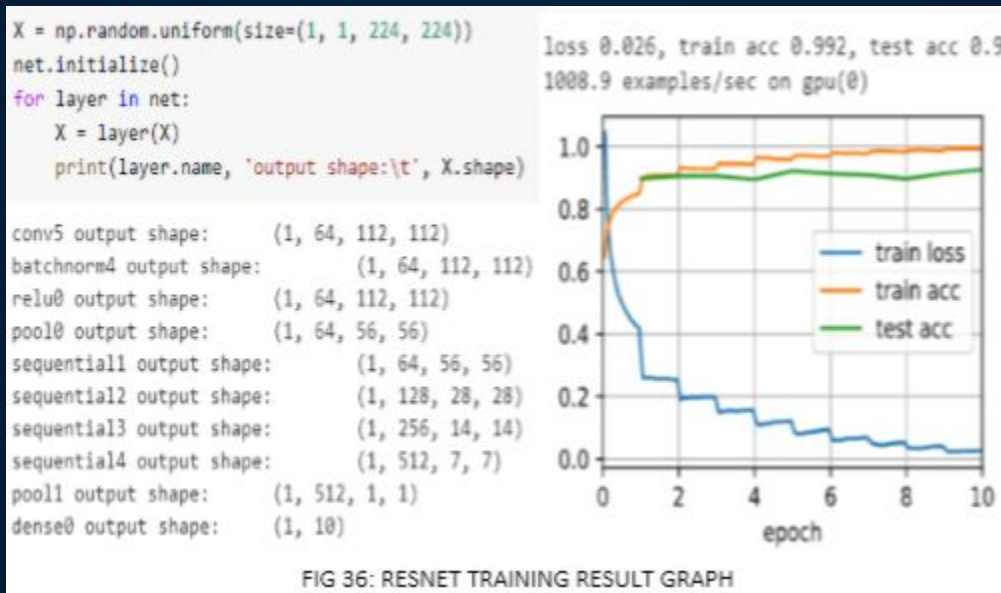
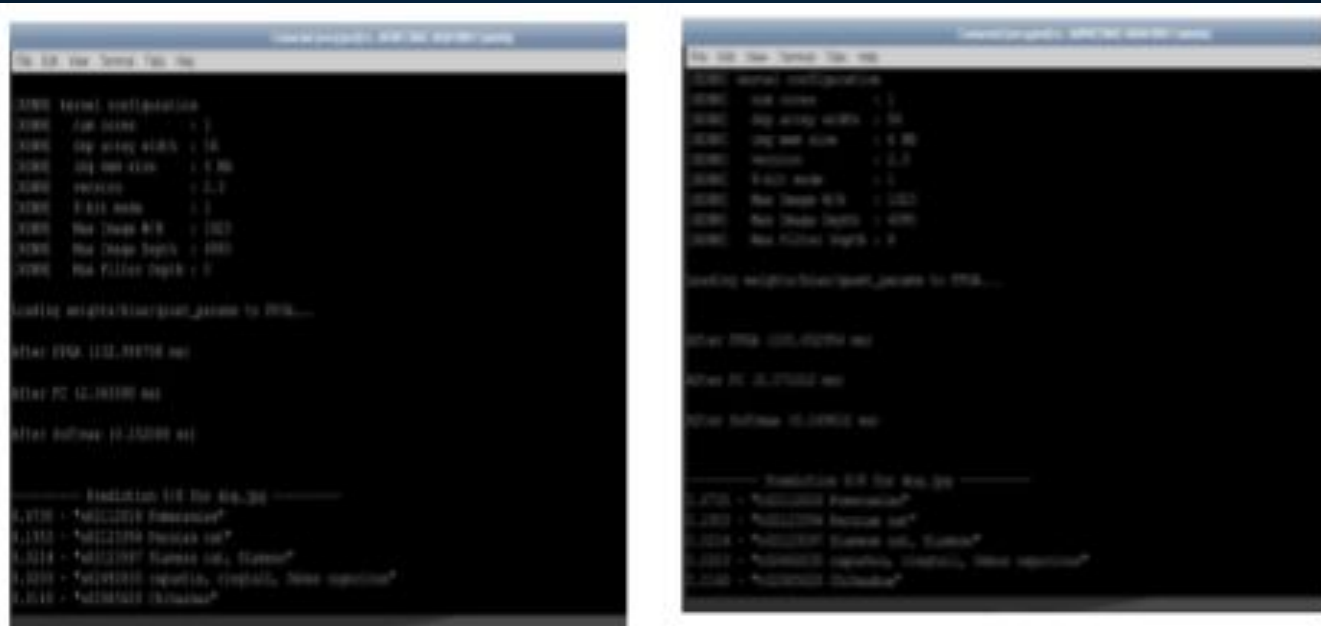


FIG 35: GOOGLNET TRAINING RESULT GRAPH

Results and Inference - On Google Colab



Results and Inference - On Cloud FPGA



SIMULATION RESULTS FOR GOOGLNET CLASSIFICATION USING INT8(LEFT) AND INT16(RIGHT) KERNAL PRECISION

Results and Inference - On Cloud FPGA

```
Terminal [project: C:\JUNK\RES-INT8\RES-INT8] (cmd)
Terminal window [C:\JUNK\RES-INT8\RES-INT8] at Microsoft Windows [PowerShell]

File Edit View Terminal Tools Help
Terminal window [C:\JUNK\RES-INT8\RES-INT8] at Microsoft Windows [PowerShell]

0.0007 - "00000000 Persian cat"
0.0012 - "00000000 Chihuahua"
0.0040 - "00000000 papillon, papillon, Chien lapin"
0.0040 - "00000000 huskies terrier"

Other PE (0.000000 ms)

Other Software (0.000000 ms)

----- Prediction V/F for dog.jpg -----
0.7961 - "00000000 Persian cat"
0.1602 - "00000000 Pomeranian"
0.0049 - "00000000 Pomeranian, Pomeranian, Pomeranian"
0.0027 - "00000000 Japanese spitz"
0.0010 - "00000000 teddy, teddy bear"

Finished!
```

```
Terminal [project: C:\JUNK\RES-INT16\RES-INT16] (cmd)
Terminal window [C:\JUNK\RES-INT16\RES-INT16] at Microsoft Windows [PowerShell]

File Edit View Terminal Tools Help
Terminal window [C:\JUNK\RES-INT16\RES-INT16] at Microsoft Windows [PowerShell]

0.0007 - "00000000 Persian cat"
0.0012 - "00000000 Chihuahua"
0.0040 - "00000000 papillon, papillon, Chien lapin"
0.0040 - "00000000 huskies terrier"

Other PE (0.000000 ms)

Other Software (0.000000 ms)

----- Prediction V/F for dog.jpg -----
0.7961 - "00000000 Persian cat"
0.1602 - "00000000 Pomeranian"
0.0049 - "00000000 Pomeranian, Pomeranian, Pomeranian"
0.0027 - "00000000 Japanese spitz"
0.0010 - "00000000 teddy, teddy bear"

Finished!
```

SIMULATION RESULTS FOR RES NET CLASSIFICATION USING INT8(LEFT) AND INT16(RIGHT) KERNEL PRECISION

Future Works



Future Works

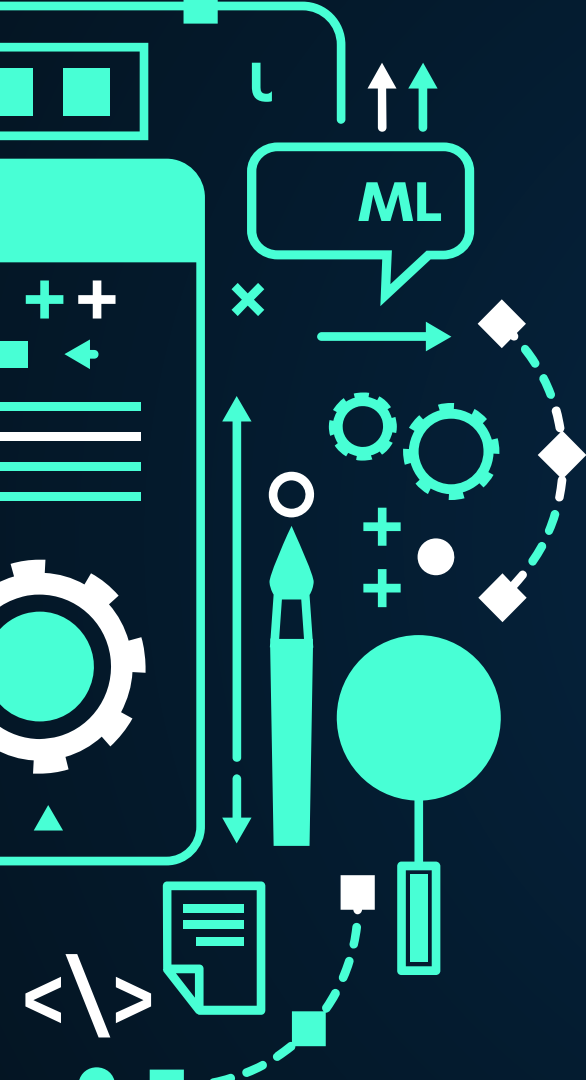
Our model being ready to be deployed in FPGA boards, the FPGA can be reconfigured to actuate actual wheels or something based on real time inference results.

With the increasing 5G technology capabilities and storage capabilities, it is possible for Bots to learn continuously with new type of data in cloud and provide real time inference.

A more efficient and robust object detection algorithm called YOLOv4 was recently released in early march which claims that it can be deployed on mobile phones using PC hardware for inference via wireless connections

This has lots of scopes in terms of education, GPS guidance and even on cooking. With the growing 5G Technology, the application for real-time inference can find its true potential enabling massive wireless data transfer with the grid and can provide a robust incremental learning environment





THANKS!

Does anyone have any question?

Useful Resources:

Thesis:

https://drive.google.com/file/d/1HKon_k9HBSsm4QW6hHjnUbccr0V3R7oF/view?usp=sharing

Github Repo: <https://github.com/vicky-ml/EEEE3078-Project>