



Despliegue de AlexNet en una NVIDIA Jetson Nano con MATLAB R2024a, GPU Coder y ROS Melodic


Universidad Católica San Pablo

1 Resumen

El objetivo de esta guía es proporcionar una ruta clara y detallada para generar, compilar y desplegar un modelo AlexNet preentrenado en una Jetson Nano, partiendo de MATLAB R2024a en un PC de alto rendimiento hasta la inferencia en tiempo real sobre ROS Melodic en la Jetson. Se aborda la creación de la biblioteca dinámica (`.so`) mediante GPU Coder, la configuración de un nodo C++ en ROS para procesar flujos de cámara USB, y la visualización de resultados en MATLAB, todo ello optimizado para aprovechar CUDA, cuDNN y TensorRT disponibles en JetPack 4.x . El resultado final, después de seguir los pasos de instalación, generación de código y despliegue, corresponde a un sistema capaz de clasificar imágenes a 227×227 px con baja latencia y uso eficiente de recursos .

2 Requisitos de Hardware

2.1 PC Anfitrión (Entrenamiento y generación de código)

- **CPU:** Procesador Intel Core i7 o i9 (o AMD Ryzen 7/9), con múltiples núcleos para acelerar compilaciones con MATLAB Coder y GPU Coder. Esta configuración reduce significativamente los tiempos de generación de código C++ optimizado para CUDA .
- **Memoria RAM:** Mínimo 16 GB; se recomienda 32 GB o más para manejar datasets voluminosos y compilaciones simultáneas sin cuellos de botella .
- **GPU:** Tarjeta NVIDIA compatible con CUDA (por ejemplo, GeForce RTX 2060/3060 o superior, Compute Capability ≥ 7.5) para acelerar pruebas locales y generación de kernels CUDA con GPU Coder .
- **Almacenamiento:** SSD NVMe de al menos 500 GB, que ofrece velocidades de lectura/escritura superiores a 3 000 MB/s, garantizando rapidez en la carga de MATLAB, toolboxes y datos de entrenamiento.
- **Conectividad:** Interfaz Ethernet Gigabit para transferencia eficiente de archivos  al dispositivo y comunicación ROS de baja latencia.

2.2 NVIDIA Jetson Nano Developer Kit (Inferencia)

- **Fuente de alimentación:** Adaptador 5 V = 3 A conectado vía Micro-USB o conector barrel, capaz de mantener ≥ 4.75 V.
- **Refrigeración:** Disipador pasivo o ventilador para reducir la temperatura durante periodos prolongados o tareas pesadas .
- **Tarjeta microSD:** Mínimo 64 GB UHS-I (clase 10); se recomiendan tarjetas 128 GB para modelos más pesados, funciones adicionales, logs y aplicaciones.
- **Periféricos:** Monitor HDMI (o adaptador), teclado y ratón USB para interacción local, y preferiblemente conexión Ethernet para estabilidad en ROS; Wi-Fi USB opcional (baja latencia y drivers requeridos).

3 Requisitos de Software

3.1 PC Host

- **MATLAB R2024a** con los siguientes complementos:

- *Deep Learning Toolbox* y **Deep Learning Toolbox Model for AlexNet Network**, que proporciona el modelo preentrenado sobre *ImageNet*.
- **GPU Coder** y **MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms**, necesarios para generar y empaquetar la biblioteca dinámica optimizada para ARM 64-bit y CUDA .
- **Image Processing Toolbox** y **Computer Vision Toolbox** para preprocesado de imágenes y algoritmos auxiliares.
- **ROS Toolbox** para establecer comunicación nativa entre MATLAB y un master ROS en la Nano.
- **Compiladores y herramientas:** GCC/G++ (≥ 7.5), CMake (≥ 3.10) y Make para compilar los nodos C++ generados por GPU Coder.

3.2 NVIDIA Jetson Nano

- **NVIDIA JetPack 4.6** (L4T R32.6.1), que incluye:
 - **CUDA 10.2** para aceleración de kernels CUDA en la GPU integrada .
 - **cuDNN 8.2.1** para primitivas de redes neuronales (convolución, pooling, normalización) altamente optimizadas .
 - **TensorRT 8.2.1** para optimización adicional de la inferencia y generación de motores ejecutables .
 - **OpenCV 4.x con soporte CUDA** (OpenCV 4.1.1 por defecto, actualizable a 4.8.0) para procesamiento de imagen en C++ si se requiere .
 - **GStreamer 1.14.5** y **v4l-utils (V4L2 1.14.2)** para manejo de cámara USB en flujo MJPEG/YUV420 que minimiza la carga de la CPU .
- **ROS Melodic Morenia** sobre Ubuntu 18.04 LTS, incluyendo `ros-*-usb-cam` e `image_transport` , para integración del driver de cámara y transporte de imágenes .

4 Estructura de Carpetas del Proyecto

La correcta organización de los ficheros facilita la mantenibilidad y replicación del sistema. A continuación se muestra el layout recomendado e implementado en las pruebas:

4.1 Estructura de carpetas en Jetson Nano

- El directorio `alexnetCodeGen/codegen/dll/myAlexNetGPU/` contiene la biblioteca dinámica `.so` generada por GPU Coder, lista para enlazarse en la Nano.

```
/home/ucspjason/
├── alexnetCodeGen/
│   ├── codegen/
│   │   ├── dll/
│   │   │   ├── myAlexNetGPU/
│   │   │   └── myAlexNetGPU.so
```

4.2 Estructura de carpetas de ROS Workspace

- El **workspace** de ROS (`~/catkin_ws/src/msra_deployed_nn`) sigue la convención de Catkin, con un paquete que agrupa código, manifiestos y launch files según las buenas prácticas de ROS.

```
/home/ucspjason/catkin_ws/
├── src/
│   ├── msra_deployed_nn/
│   │   ├── CMakeLists.txt
│   │   ├── package.xml
│   │   ├── launch/
│   │   │   ├── alexnet_usb.launch
│   │   └── src/
│   │       ├── alexnet.cpp
│   │       └── alexnet.hpp
```

4.3 Estructura de carpetas PC Host

- En el host, `HostComputerFiles /` agrupa scripts de MATLAB para exportar el modelo, probar la inferencia localmente y visualizar resultados vía ROS.

```
HostComputerFiles/
├── alexnet.mat
├── classNames.mat
├── myAlexNet.m
├── myAlexNetGPU.m
└── myAlexNetGPU.prj
```

```
|— myAlexNetGPU_generation_script.m
|— saveAlexNetToFile.m
|— testAlexNet.m
|— visualizeROSDData.m
```

5 Preparación del Modelo en MATLAB

Esta fase convierte el modelo pre-entrenado de AlexNet (importado de **Deep Learning Toolbox Model for AlexNet Network**) en artefactos exportables y ejecutables en la Nano.

5.1 Guardar la red en un archivo **.mat**

En `saveAlexNetToFile.m` se define la función `saveAlexNetToFile()` que utiliza el comando

```
net = alexnet;
save('alexnet.mat','net');
```

para almacenar el objeto `SeriesNetwork` en `alexnet.mat`, que sólo contiene la red entrenada ([MathWorks](#)).

5.2 Función de inferencia compatible con GPU Coder

`myAlexNetGPU.m` define la entrada y la llamada a la red en formato MAT-file:

```
function classIdx = myAlexNetGPU(im)
persistent net
if isempty(net)
    net = coder.loadDeepLearningNetwork('alexnet.mat');
end
output = predict(net, im);
[~,classIdx] = max(output);
end
```

La instrucción `coder.loadDeepLearningNetwork` carga el MAT-file de AlexNet directamente, asegurando la compatibilidad con GPU Coder ([MathWorks](#)).

5.3 Generar la biblioteca CUDA para Jetson Nano

El script `myAlexNetGPU_generation_script.m` orquesta la configuración de GPU Coder y MATLAB Coder para producir la biblioteca compartida `myAlexNetGPU.so`, compatible con la arquitectura ARM 64-bit y optimizada para cuDNN y TensorRT. A continuación se muestra el fragmento clave:

```
%% Crear objeto de configuración 'coder.EmbeddedCodeConfig'.
cfg = coder.gpuConfig('lib','ecoder',true);
cfg.GenerateReport          = true;
cfg.ReportPotentialDifferences = false;
cfg.InitFltsAndDblsToZero   = false;
cfg.BuildConfiguration      = 'Faster Builds';
cfg.Hardware                 = coder.hardware('NVIDIA Jetson');
cfg.HardwareImplementation.TargetHWDeviceType = ...
    'ARM Compatible→ARM 64-bit (LP64)';

%% Define los tipos de argumentos de entrada 'myAlexNetGPU'.
ARGS = cell(1,1);
ARGS{1} = coder.typeof(uint8(0),[227 227 3]); % Imagen 227×227×3 en ui
nt8

%% Invocar MATLAB Coder.
codegen -config cfg -o myAlexNetGPU myAlexNetGPU -args ARGS{1} -nar
gout 1
```

- `coder.gpuConfig('lib','ecoder',true)` : crea un objeto de configuración para generar una **biblioteca compartida** en plataforma NVIDIA y habilita el flujo de **Embedded Coder** para optimizaciones de código.
- `GenerateReport = true` : produce un informe detallado en HTML que documenta la generación de código, las métricas de cobertura y posibles advertencias (Importante para debugging).
- `ReportPotentialDifferences = false` : desactiva la generación de reportes de discrepancias numéricas menores entre MATLAB y el código generado, reduciendo ruido en la revisión de resultados.
- `InitFltsAndDblsToZero = false` : deja sin inicializar variables de punto flotante para preservar el comportamiento por defecto de la red, a fin de evitar diferencias sutiles en estilos de inicialización.

- `BuildConfiguration = 'Faster Builds'` : prioriza tiempos de compilación reducidos (menor optimización de código) para iteraciones de desarrollo más ágiles.
- `cfg.Hardware = coder.hardware('NVIDIA Jetson')` : especifica la plataforma de destino, de manera que las rutas de herramienta y los flags de compilador se ajusten a JetPack y a la arquitectura ARM propia de Jetson Nano.
- `TargetHWDeviceType = 'ARM Compatible→ARM 64-bit (LP64)'` : define la *Application Binary Interface* (ABI) y la serie de instrucciones para compilar el código en el procesador ARM 64-bit.
- **Definición de argumentos (`ARGS`)**: declara que la función de entrada `myAlexNetGPU` recibe una matriz de tipo `uint8` de dimensiones $227 \times 227 \times 3$, lo cual permite a GPU Coder generar kernels correctamente y optimizados.
- **Invocación de `codegen`** : ejecuta la generación de código con la configuración especificada, produciendo la biblioteca `myAlexNetGPU.so` (o su equivalente en la carpeta de salida) y exportando un único valor de salida (`class_idx`).

Esta configuración asegura que la biblioteca compartida resultante esté preparada para su despliegue en NVIDIA Jetson Nano, aprovechando cuDNN para acelerar convoluciones y TensorRT cuando esté habilitado, al tiempo que mantiene un flujo de desarrollo eficiente gracias a las opciones de "Faster Builds" y a los informes de generación de código.

5.4 Configuración y verificación de la conexión MATLAB–Jetson

La primera etapa de integración consiste en crear un objeto `jetson` en MATLAB que represente la conexión SSH al dispositivo NVIDIA Jetson Nano y permita interactuar con sus periféricos, como la cámara USB o los pines GPIO.

```
% Setup the Jetson Object in MATLAB
ipaddress = "192.168.0.108";
username = "ucspjason";
password = "JasonNano10";
```

Para establecer esta conexión, se emplea la función

```
hwobj = jetson(ipaddress, username, password);
```

donde `ipaddress` , `username` y `password` especifican las credenciales del host Jetson Nano. Una vez conectado, se genera un objeto de configuración de GPU Coder específico para Jetson mediante

```
% Create a gpuEnvConfig object
gpuEnvObj = coder.gpuEnvConfig('jetson');
```

que prepara el entorno de generación y ejecución de código optimizado para la arquitectura CUDA de la Nano ([MathWorks](#)).

A continuación, se activan las etapas de verificación y generación de código en CPU y GPU asignando propiedades al objeto:

- `BasicCodegen = 1` y `BasicCodeexec = 1` habilitan la generación y ejecución de un test básico de CPU.

```
gpuEnvObj.BasicCodegen = 1;
gpuEnvObj.BasicCodeexec = 1;
```

- `DeepLibTarget = 'cudnn'` , `DeepCodeexec = 1` y `DeepCodegen = 1` configuran la verificación profunda de la librería cuDNN y la generación de kernels CUDA avanzados.

```
gpuEnvObj.DeepLibTarget = 'cudnn';
gpuEnvObj.DeepCodeexec = 1;
gpuEnvObj.DeepCodegen = 1;
```

- El campo `HardwareObject` vincula explícitamente el objeto `jetson` al `gpuEnvObj` , de modo que las comprobaciones de entorno se realicen directamente sobre la plataforma remota.

```
gpuEnvObj.HardwareObject = hwobj;
```

- Para verificar que MATLAB encuentra todas las dependencias (CUDA, cuDNN, TensorRT) y que la configuración de variables de entorno (`PATH` , `LD_LIBRARY_PATH`) es correcta, se ejecuta:

```
results = coder.checkGpuInstall(gpuEnvObj);
disp(results);
```


La función `coder.checkGpuInstall` devuelve un resumen de las pruebas realizadas en la Jetson Nano, informando sobre el estado de cada instalación requerida para la generación de código GPU. Esta comprobación previa evita fallos de compilación o enlace en etapas posteriores de generación de la biblioteca dinámica.

```
Checking for CUDA availability on the Target...
Checking for 'nvcc' in the target system path...
Checking for cuDNN library availability on the Target...
Checking for TensorRT library availability on the Target...
Checking for prerequisite libraries is complete.
Gathering hardware details...
Checking for third-party library availability on the Target...
Gathering hardware details is complete.
```

```
Board name      : NVIDIA Jetson Nano Developer Kit
CUDA Version    : 10.2
cuDNN Version   : 8.2
TensorRT Version : 8.2
GStreamer Version : 1.14.5
V4L2 Version    : 1.14.2-1
SDL Version     : 1.2
OpenCV Version  : 4.8.0
Available Webcams : USB-General-Webcam
Available GPUs   : NVIDIA Tegra X1
Available Digital Pins : 7 11 12 13 15 16 18 19 21 22 23 24 26 29 31 32 33 35
36 37
```

Finalmente, mostrar `results` en pantalla proporciona un diagnóstico claro del entorno, simplificando la integración entre MATLAB y la Jetson Nano, garantizando la compatibilidad de bibliotecas antes de proceder con la generación y despliegue de la librería `.so`.

6 Nodo ROS en C++

El nodo `alexnet` actúa como el puente entre la cámara USB y el modelo GPU-Coder, suscribiéndose al stream de imágenes crudas, recortando y

reformateando cada fotograma para la inferencia, y publicando tanto la imagen procesada como el índice de la clase con mayor confianza.

6.1 Definición de la interfaz (alexnet.hpp)

```
#ifndef ALEXNET_HPP_
#define ALEXNET_HPP_

#include <ros/ros.h>
#include <sensor_msgs/Image.h>
#include <std_msgs/UInt32.h>
#include <vector>
#include <cstdlib>
#include "myAlexNetGPU.h"

class GpuNetPub
{
private:
    ros::Publisher data_pub_;
    ros::Publisher img_pub_;

public:
    GpuNetPub(const std::string& data_topic_name,
              const std::string& img_topic_name,
              int msg_limit,
              ros::NodeHandle* nh)
    {
        // Publica índices de clase como UInt32
        data_pub_ = nh->advertise<std_msgs::UInt32>(data_topic_name, msg_limit);
        // Publica la subimagen 227×227 en formato rgb8
        img_pub_ = nh->advertise<sensor_msgs::Image>(img_topic_name, msg_limit);
    }

    void msgCallback(const sensor_msgs::Image::ConstPtr& inmsg)
    {
        constexpr int32_t SUB_SIZE = 227;
```

```

constexpr int32_t CHANNELS = 3;

// Verifica tamaño mínimo
if (inmsg->width < SUB_SIZE || inmsg->height < SUB_SIZE)
{
    ROS_WARN_THROTTLE(5, "Imagen recibida demasiado pequeña (%u
x%u)",
                      inmsg->width, inmsg->height);
    return;
}

// Buffer de entrada para GPU Coder y vector para la subimagen
uint8_t input[SUB_SIZE * SUB_SIZE * CHANNELS];
std::vector<uint8_t> img_msg_data(SUB_SIZE * SUB_SIZE * CHANNEL
S);

// Cálculo de offsets para recorte central
int32_t h_off = (inmsg->height - SUB_SIZE) / 2;
int32_t w_off = (inmsg->width - SUB_SIZE) / 2;
int32_t out_idx = 0;

// Recorte central y reordenado RGB row-major → column-major
for (int32_t i = 0; i < SUB_SIZE; ++i)
{
    for (int32_t j = 0; j < SUB_SIZE; ++j)
    {
        int32_t base = (h_off + i) * inmsg->step
                      + (w_off + j) * CHANNELS;
        for (int32_t k = 0; k < CHANNELS; ++k)
        {
            int32_t idx = (SUB_SIZE * SUB_SIZE * k)
                      + (SUB_SIZE * j) + i;
            uint8_t v = inmsg->data[base + k];
            input[idx] = v;           // buffer GPU Coder
            img_msg_data[out_idx++] = v; // buffer ROS
        }
    }
}

```

```

// Invoca la red generada por GPU Coder
float class_idx = myAlexNetGPU(input);
ROS_INFO_STREAM_THROTTLE(1, "Índice de clasificación: " << class_i
dx);

// Publica índice de clase
std_msgs::UInt32 cls_msg;
cls_msg.data = static_cast<uint32_t>(class_idx);
data_pub_.publish(cls_msg);

// Construye y publica mensaje de imagen recortada
sensor_msgs::Image img_msg;
img_msg.height = SUB_SIZE;
img_msg.width = SUB_SIZE;
img_msg.encoding = "rgb8";
img_msg.step = CHANNELS * SUB_SIZE;
img_msg.data = std::move(img_msg_data);
img_pub_.publish(img_msg);
}
};

#endif // ALEXNET_HPP_

```

- **Verificación de tamaño:** evita procesar imágenes menores a 227×227 px, informando vía `ROS_WARN_THROTTLE` en lugar de abortar el nodo .
- **Offsets:** calcular `h_off` y `w_off` asegura un recorte simétrico, evitando distorsión geométrica .
- **Reordenado:** `input[idx]` convierte la imagen de layout row-major (C/C++) a column-major (MATLAB GPU Coder) .
- **Publicación:** usa dos topics, uno para la etiqueta (`std_msgs::UInt32`) y otro para la subimagen (`sensor_msgs::Image`), facilitando la suscripción independiente en MATLAB o herramientas ROS.

6.2 Implementación del nodo (alexnet.cpp)

```

#include <ros/ros.h>
#include <sensor_msgs/Image.h>
#include "alexnet.hpp"
#include <string>

int main(int argc, char** argv)
{
    ros::init(argc, argv, "alexnet");
    ros::NodeHandle nh;          // Nodo global
    ros::NodeHandle pnh("~");    // Nodo privado para parámetros

    // Constructor registra tópicos "network_out" y "network_view"
    GpuNetPub netpub("network_out", "network_view", 1, &nh);

    // Lee parámetro privado 'input_topic', por defecto la cámara USB
    std::string input_topic;
    pnh.param<std::string>("input_topic",
                          input_topic,
                          "/usb_cam/image_raw");
    ROS_INFO_STREAM("AlexNet suscrito a: " << input_topic);

    // Suscripción con callback miembro
    ros::Subscriber img_sub =
        nh.subscribe(input_topic, 1, &GpuNetPub::msgCallback, &netpub);

    ros::spin(); // Espera callbacks
    return 0;
}

```

- **Namespaces:** `pnh("~")` crea un espacio privado de parámetros, permitiendo modificar `input_topic` desde el launch sin recompilar .
- **Estructura limpia:** separa inicialización, lectura de parámetros, suscripción y bucle de eventos, facilitando la extensión futura.

6.3 Declaración de dependencias (CMakeLists.txt)

```

cmake_minimum_required(VERSION 3.10)
project(msra_deployed_nn LANGUAGES CXX)

find_package(catkin REQUIRED COMPONENTS
  roscpp
  std_msgs
  sensor_msgs
  image_transport
)
catkin_package(CATKIN_DEPENDS roscpp std_msgs sensor_msgs image_t
ransport)

# Importa la biblioteca GPU Coder
set(CODEGEN_DIR /home/ucspjason/alexnetCodeGen)
add_library(myAlexNetGPU SHARED IMPORTED)
set_target_properties(myAlexNetGPU PROPERTIES
  IMPORTED_LOCATION
    ${CODEGEN_DIR}/codegen/dll/myAlexNetGPU/myAlexNetGPU.so
)

# Soporte CUDA
find_package(CUDA REQUIRED)

include_directories(
  ${catkin_INCLUDE_DIRS}
  ${CODEGEN_DIR}/codegen/dll/myAlexNetGPU # Contiene myAlexNetGP
U.h
  ${CUDA_INCLUDE_DIRS}
)

add_executable(alexnet src/alexnet.cpp)
target_link_libraries(alexnet
  myAlexNetGPU      # Biblioteca GPU Coder
  ${CUDA_LIBRARIES} # CUDA runtime
  /usr/lib/aarch64-linux-gnu/libcudnn.so # cuDNN
  ${catkin_LIBRARIES} # ROS

```

```
)
add_dependencies(alexnet sensor_msgs_generate_messages_cpp)
```

- **Biblioteca importada:** `myAlexNetGPU.so` se enlaza directamente, evitando recompilar el código generado .
- **CUDA y cuDNN:** `find_package(CUDA)` rellena rutas de headers y libs necesarias para la compilación cruzada en ARM64 .
- **image_transport:** declarado en `find_package` y `catkin_package` para habilitar publishers/subscribers eficientes .

6.4 Configuración de usb_cam y formatos YUYV (alexnet_usb.launch)

```
<launch>
<!-- AlexNet node -->
<node pkg="msra_deployed_nn" type="alexnet" name="alexnet"
      output="screen">
  <param name="input_topic" value="/usb_cam/image_raw"/>
</node>

<!-- usb_cam with YUYV 640×360 @30 FPS -->
<node pkg="usb_cam" type="usb_cam_node" name="usb_cam" output
="screen">
  <param name="video_device"      value="/dev/video0"/>
  <param name="image_width"       value="640"/>
  <param name="image_height"      value="360"/>
  <param name="framerate"         value="30"/>
  <param name="pixel_format"      value="yuyv"/>
  <param name="auto_white_balance" value="false"/>
  <param name="auto_focus"        value="false"/>
</node>
</launch>
```

- **YUYV 2 B/px** elimina decodificación JPEG, minimizando latencia en la Nano .
- **Parámetros:** `auto_white_balance` y `auto_focus` desactivados para evitar mix de controles no soportados por algunos módulos V4L2 .

- **Separación de nodos:** `alexnet` procesa el topic publicado por `usb_cam_node`, manteniendo claridad en el flujo de datos.

7 Despliegue y Ejecución

7.1 Verificación de formatos soportados

Antes de lanzar el sistema, conviene comprobar qué modos acepta la cámara USB:

```
v4l2-ctl -d /dev/video0 --list-formats-ext
```

Esto muestra todas las resoluciones y formatos (MJPEG, YUYV, etc.), ayudando a ajustar `image_width`, `image_height` y `pixel_format` en el launch. Ejemplo de la ejecución de lo antes mencionado, es:

```
ucspjason@ucspjason-desktop:~$ v4l2-ctl -d /dev/video0 --list-formats-ext
ioctl: VIDIOC_ENUM_FMT
  Index    : 0
  Type     : Video Capture
  Pixel Format: 'MJPG' (compressed)
  Name      : Motion-JPEG
    Size: Discrete 1920×1080
      Interval: Discrete 0.033s (30.000 fps)
    Size: Discrete 1280×720
      Interval: Discrete 0.033s (30.000 fps)
    Size: Discrete 640×360
      Interval: Discrete 0.033s (30.000 fps)
    Size: Discrete 1920×1080
      Interval: Discrete 0.033s (30.000 fps)
    Size: Discrete 1920×1080
      Interval: Discrete 0.033s (30.000 fps)

  Index    : 1
  Type     : Video Capture
  Pixel Format: 'YUYV'
  Name      : YUYV 4:2:2
```



```
Size: Discrete 640×360
Interval: Discrete 0.033s (30.000 fps)
Size: Discrete 640×360
Interval: Discrete 0.033s (30.000 fps)
Size: Discrete 640×360
Interval: Discrete 0.033s (30.000 fps)
Size: Discrete 640×360
Interval: Discrete 0.033s (30.000 fps)
Size: Discrete 640×360
Interval: Discrete 0.033s (30.000 fps)
```

Del resultado se observan las diferentes configuraciones, donde se debe considerar que a mayor resolución, mayor carga computacional.

7.2 Compilación del workspace ROS

```
cd ~/catkin_ws
catkin_make -DCMAKE_BUILD_TYPE=Release
source devel/setup.bash
```

- `DCMAKE_BUILD_TYPE=Release` habilita optimizaciones de compilación en C++ (inline, O3) imprescindibles en Jetson Nano .
- El script de CMake incluye la importación de `myAlexNetGPU.so` , CUDA y cuDNN, garantizando que el ejecutable `alexnet` enlace correctamente con la librería generada por GPU Coder .

7.3 Inicio de nodos via roslaunch

```
roslaunch msra_deployed_nn alexnet_usb.launch
```

- El nodo `usb_cam_node` abre `/dev/video0` con YUYV a 640×360 px @ 30 FPS.
- El nodo `alexnet` suscribe a `/usb_cam/image_raw` , procesa cada frame con `myAlexNetGPU` y publica:
 - `/network_view` (sensor_msgs/Image RGB8 227×227 px).
 - `/network_out` (std_msgs/UInt32 con índice de clase).

8 Visualización en MATLAB

```
% visualizeROSData.m
rosshutdown;
rosinit('192.168.0.107');    % IP de la Jetson Nano
imgSub = rossubscriber('/network_view');
classSub = rossubscriber('/network_out');

while true
    if ~isempty(imgSub.LatestMessage)
        frame = readImage(imgSub.LatestMessage); % convierte a uint8
    else
        frame = zeros(227,227,3,'uint8');
    end

    if ~isempty(classSub.LatestMessage)
        idx = classSub.LatestMessage.Data;
        label = classNames{idx};
    else
        label = 'NaN';
    end

    imshow(frame);
    title(label);
    drawnow;
end
```

- `readImage` traduce el mensaje ROS a una matriz uint8 en MATLAB, simplificando la conversión de `sensor_msgs/Image` a imagen RGB8 .
- El loop robusto gestiona mensajes nulos para evitar bloqueos si el nodo `alexnet` tarda en publicar .
- Mostrar `label` sobre la imagen facilita la validación visual de la inferencia en tiempo real.

9 Optimización de Rendimiento y Buenas Prácticas

Acción	Descripción y Relevancia
--------	--------------------------

<code>nvmodel -m 0 && jetson_clocks</code>	Bloquea CPU y GPU en frecuencias máximas antes de medición de FPS, evitando fluctuaciones por thermal throttling .
Formatos MJPEG + YUV420 vs YUYV	MJPEG+YUV420 reduce ancho de banda (1.5 B/px) y decodificación (GPU_Coder espera RGB8), pero YUYV (2 B/px) elimina JPEG completo .
Throttle de logs ROS	Usar <code>ROS_INFO_STREAM_THROTTLE(1,...)</code> previene saturar la consola y evita I/O blocking durante alta frecuencia de callbacks .
Ventilación activa	Instalar ventilador PWM para mantener < 60 °C bajo carga pesada; previene reducción de frecuencia por temperatura. .
<code>v4l2-ctl --list-formats-ext</code>	Verifica compatibilidad real de la webcam y ajusta parámetros en el launch para evitar out-of-buffer mismatches. .
<code>image_transport/compressed</code> (opcional)	Comprime tópicos ROS con JPEG en caso de streaming multi-nodo, reduciendo ancho de banda de red y almacenamiento. .

Estas prácticas aseguran que la Jetson Nano entregue inferencia con baja latencia (~30 FPS) y un uso de CPU constante (< 25 %), manteniendo la estabilidad del sistema en entornos de producción.

10 Solución de problemas comunes

A continuación se describen los fallos más frecuentes durante la puesta en marcha y su corrección.

1. Error `CUDNN_STATUS_NOT_SUPPORTED` al invocar la librería cuDNN

- **Síntoma:** la llamada a `myAlexNetGPU` devuelve un error de cuDNN no soportado.
- **Causa:** incompatibilidad entre la versión de cuDNN instalada en JetPack y la esperada por GPU Coder.
- **Solución:** verificar la versión de cuDNN con `dpkg -l | grep cudnn` y asegurarse de usar cuDNN 8.2.1, que es la suministrada en JetPack 4.6. De lo contrario, reinstalar el paquete adecuado desde repositorio NVIDIA ([NVIDIA Developer Forums](https://forums.nvidia.com/)).

2. Imposible cargar `libnvinfer.so.7` (TensorRT)

- **Síntoma:** al arrancar el nodo, aparece `Could not load dynamic library 'libnvinfer.so.7'`.
- **Causa:** el sistema no encuentra la versión de TensorRT instalada o hay un desajuste de versiones.
- **Solución:** instalar TensorRT 8.2.1 mediante `sudo apt install nvidia-tensorrt` o copiar manualmente `libnvinfer.so.8` y `libnvinfer_plugin.so.8` renombrándolos a la versión requerida ([NVIDIA Developer Forums](#), [Stack Overflow](#)).

3. Permiso denegado al abrir `/dev/video0`

- **Síntoma:** `Failed to open video device /dev/video0: Permission denied`.
- **Causa:** el usuario `ubuntu` (o `ubuntu`) no pertenece al grupo `video`.
- **Solución:** agregar el usuario al grupo con `sudo usermod -aG video $USER`, o bien cambiar permisos con `sudo chmod 660 /dev/video0` y reiniciar el servicio udev ([forums.zoneminder.com](#), [GitHub](#)).

4. Fugas de memoria o consumo excesivo de RAM

- **Síntoma:** uso de memoria crece hasta saturar la Nano, causando cuelgues o procesos "killed".
- **Causa:** buffers de vídeo no liberados o librerías no compatibles (p. ej., repetidas cargas del objeto `net`).
- **Solución:** asegurar que `myAlexNetGPU` se cargue una sola vez (uso de `persistent net`), y monitorizar con `htop` para identificar procesos que aumentan uso de memoria. Actualizar JetPack a la versión recomendada si persisten fugas ([NVIDIA Developer Forums](#), [NVIDIA Developer Forums](#)).

5. Out of Memory durante entrenamiento o inferencia

- **Síntoma:** mensajes "Killed" o "Out Of Memory" en los logs al procesar imágenes grandes o lotes.
- **Causa:** versión 2 GB de RAM en Nano 2 GB; uso de lotes > 1 .
- **Solución:** reducir la resolución de entrada (usar 640×360 en launch), procesar de uno en uno (batch 1) y contemplar pasar a Jetson Orin Nano si se requiere más memoria ([Reddit](#)).

6. Error de enlace con librerías TensorRT en contenedores Docker

- **Síntoma:** al compilar dentro de un contenedor, `failed to link to libnvinfer.so`.
- **Causa:** toolchain incorrecto o rutas de librerías no exportadas.
- **Solución:** usar la cadena de herramientas de L4T (arch=aarch64-linux-gnu) y definir `CMAKE_TOOLCHAIN_FILE=/usr/share/cmake-3.10/Modules/Platform/Linux-aarch64.cmake` ([NVIDIA Developer Forums](#)).

7. Arquitectura de librería incorrecta (`wrong arch of a lib`)

- **Síntoma:** errores de "undefined reference" o incompatibilidad al enlazar.
- **Causa:** compilación cruzada con toolchain x86 en lugar de ARM64.
- **Solución:** usar `aarch64-linux-gnu-gcc / g++` o configurar `CMake` con el toolchain de JetPack para ARM64 ([GitHub](#)).

8. Problemas con versiones antiguas de TensorRT

- **Síntoma:** quejas sobre falta de `libnvinfer.so.6`.
- **Causa:** proyectos previos solicitando versiones antiguas.
- **Solución:** enlazar dinámicamente desde TensorRT 8.2, renombrando o instalando el paquete de compatibilidad `nvidia-tensorrt-compat` ([Stack Overflow](#)).

La guía ha recorrido desde la preparación de hardware y software en el PC anfitrión y en la Jetson Nano, hasta la generación de la biblioteca CUDA con GPU Coder, la construcción del nodo ROS en C++ y su integración con la cámara USB y MATLAB. Se han detallado los flujos de datos, la parametrización de tópicos, la estructura de carpetas y fragmentos críticos de código, así como estrategias de optimización y resolución de errores habituales.

Próximos pasos sugeridos:

- Extender el pipeline para soportar distintos modelos (ResNet, MobileNet) mediante la misma arquitectura de GPU Coder.
- Integrar TensorRT directamente en C++ para optimizar aún más la inferencia en la Jetson Nano u Orin Nano.
- Incorporar transporte comprimido con `image_transport/compressed` para streaming remoto de alta eficiencia.

- Automatizar pruebas de rendimiento y latencia usando `rosbag` y scripts de benchmark para validar throughput en distintos escenarios.

12 Bibliografía

1. **NVIDIA Jetson Nano Developer Kit User Guide – Module and Thermal Design**: sección del manual oficial que explica cómo el disipador pasivo soporta hasta 10 W a 25 °C, y cómo añadir un ventilador PWM si se prevén cargas prolongadas para evitar thermal throttling. ([NVIDIA Developer](#))
2. **NVIDIA JetPack SDK 4.6 Release Notes – JetPack 4.6 Features**: página oficial que describe las versiones de CUDA (10.2), cuDNN (8.x), TensorRT (8.x) y VPI incluidas, así como mejoras de seguridad y soporte de OTAs para L4T 32.6.1. ([NVIDIA Developer](#))
3. **GStreamer 1.14 Release Notes – 1.14.5 Bug-Fix Release**: documento de [Freedesktop.org](#) que enumera correcciones de errores de la serie 1.14 y confirma que la versión 1.14.5 (29 de mayo 2019) es la última antes de pasar a 1.16. ([gststreamer.freedesktop.org](#))
4. **v4l-utils 1.14.2 Documentation** – [GitLab / LinTV](#): repositorio que aloja la documentación de V4L2 y libdvbv5, esencial para entender parámetros como `pixel_format` y `framerate` al usar la webcam en Jetson. ([GitLab](#))
5. **ROS Wiki – Ubuntu install of ROS Melodic**: instrucciones oficiales para configurar repositorios, claves, dependencias y entorno en Ubuntu 18.04, requisito previo para instalar `ros-melodic-usb-cam` e `image_transport`. ([wiki.ros.org](#))
6. **MathWorks File Exchange – Deep Learning Toolbox Model for AlexNet Network**: paquete oficial que provee un objeto `SeriesNetwork` entrenado sobre ImageNet, con 23 capas y capaz de reconocer 1 000 categorías, listo para exportar a `.mat`. ([MathWorks](#))
7. **MathWorks Documentation – MATLAB Coder Support Package for NVIDIA Jetson and NVIDIA DRIVE Platforms**: descripción del paquete que automatiza la generación remota y despliegue de código MATLAB/Simulink en plataformas NVIDIA, incluyendo configuración SSH y manejo de binarios `.so`. ([MathWorks](#))
8. **Deep Learning with MATLAB, NVIDIA Jetson, and ROS**: vídeo de MathWorks en el que Jon Zeosky y Sebastian Castro muestran cómo partir de una red neuronal preentrenada en MATLAB para generar, con GPU

Coder, una biblioteca CUDA independiente, desplegarla en una plataforma NVIDIA Jetson y, finalmente, empaquetarla como nodo ROS en C++ para integrarla con otros nodos de la red. ([MathWorks](#))

9. **Deep Learning with NVIDIA Jetson and ROS:** paquete de ejemplo en MATLAB Central File Exchange que contiene los archivos para desplegar en hardware NVIDIA Jetson una red neuronal preentrenada desde MATLAB, y utilizar la biblioteca generada dentro de un nodo ROS escrito en C++. Incluye README con instrucciones de configuración y requisitos de Toolbox. ([MathWorks](#))
10. **MathWorks Miniseries | April 29, 2024 | Object Detection with ROS 2 and Jetson Nano:** episodio en directo de la serie *Miniseries* donde se enseña a usar el Image Labeler App de MATLAB para etiquetar datos, entrenar una red de detección de objetos y, mediante la generación de código de MATLAB, desplegarla como nodo ROS 2 en un Jetson Nano (u Orin Nano), cubriendo el flujo completo de etiqueta→entrenamiento→generación→despliegue. ([YouTube](#))
11. **Deploy YOLOv2 to an NVIDIA Jetson:** vídeo en el que Connell D'Souza y Neha Goel explican cómo usar GPU Coder para generar código CUDA de la red YOLOv2 diseñada en MATLAB, configurar el objetivo Jetson y desplegar los binarios resultantes en la GPU embebida, permitiendo realizar inferencia en tiempo real directamente en el dispositivo. ([YouTube](#))