



MACHINE LEARNING CON NVIDIA JETSON Y MATLAB

Despliegue de MNIST en Jetson

Diego Banda R.
Universidad Católica San Pablo
diego.banda@ucsp.edu.pe





AGENDA

- El dataset MNIST: Consideraciones.
- Lo que la Jetson necesita.
- Datos relevantes para el modelo.
- Despliegue de la red en Jetson Nano.



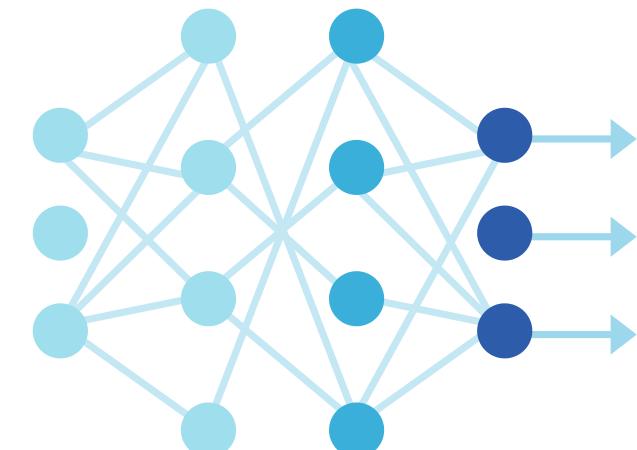
Lista de Archivos Necesarios

Entrenar, crear pipeline y desplegar

trainMNIST.m

Entrenar y validar el modelo

- Entrenar una red neuronal sencilla para MNIST y guardar .mat.
- Muestra la curva de entrenamiento y una matriz de confusión.
- Probar el modelo.
- **Guarda:** net, inputSize, classNames



mnistDetection.m

Codegen para el despliegue

- **Requiere:** Deep Learning Toolbox + GPU Coder + Support Pkg Jetson
- Incluye el pre-procesamiento para adaptar los datos al modelo.
- Realiza la inferencia y el post-procesamiento para mostrar en tiempo real.

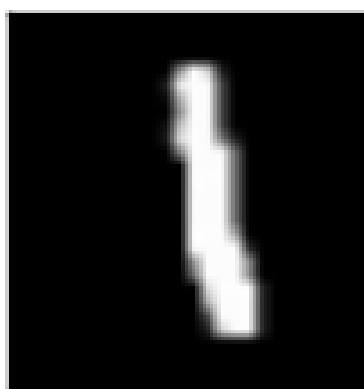


mnistDeployment.m

Prueba del modelo desplegado

- Conecta a la IP de Jetson y define parámetros importantes.
- Genera un ejecutable GPU con MATLAB Coder.
- Muestra la app en la Jetson.

Num: 1



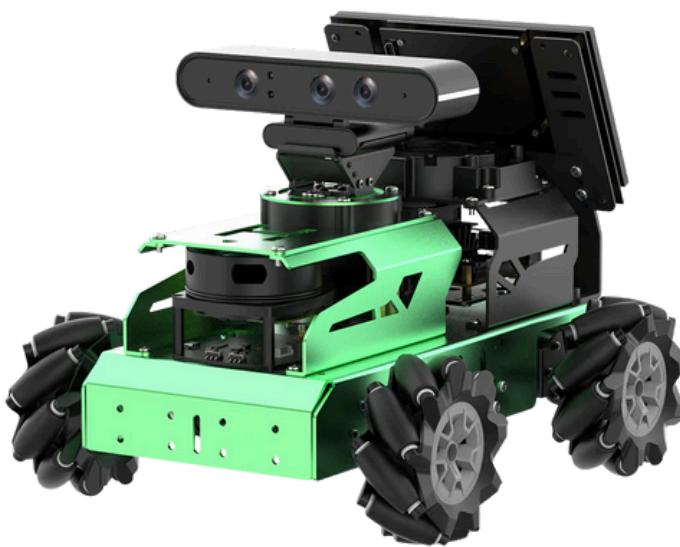
Num: 2





Consideraciones Importantes para el despliegue

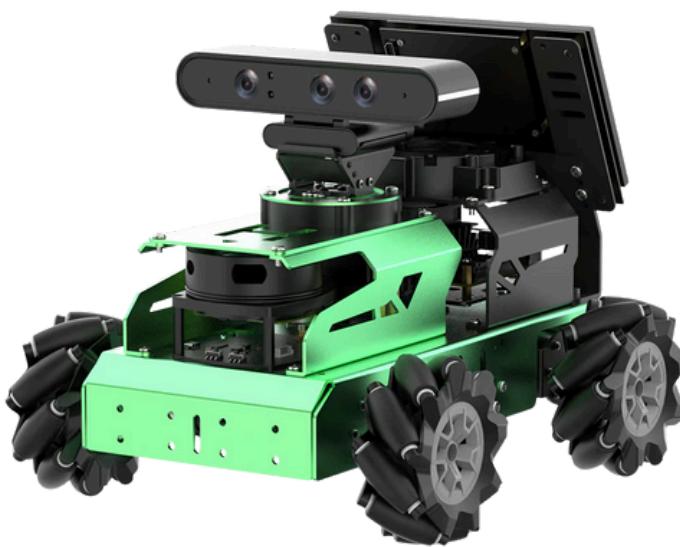
- Librerías necesarias en la Jetson.
- Datos relevantes al entrenar el modelo.
- Despliegue del modelo y visualización.





Consideraciones Importantes para el despliegue

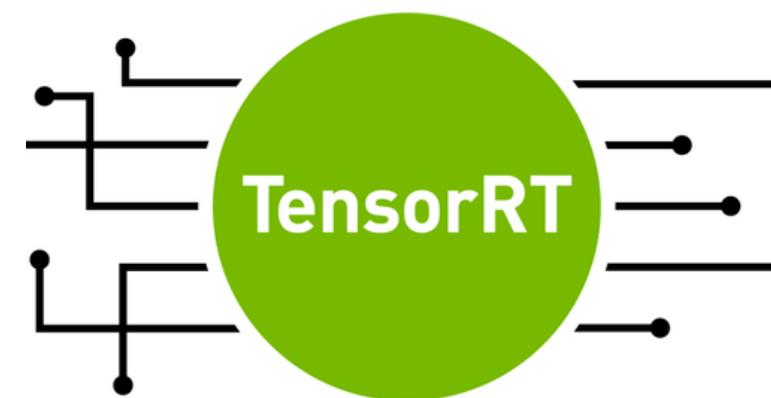
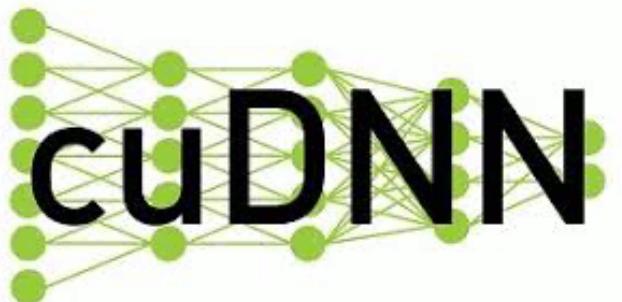
- Librerías necesarias en la Jetson.
- Datos relevantes al entrenar el modelo.
- Despliegue del modelo y visualización.





Lo necesario para la Jetson

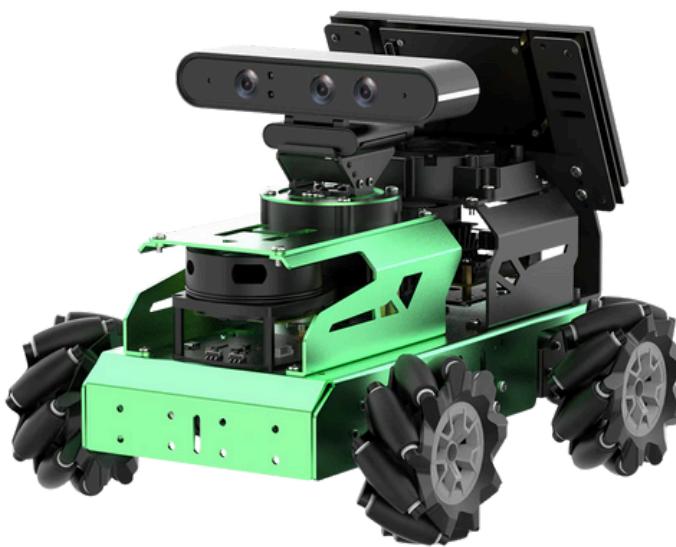
- **CUDA Toolkit:** runtime/SDK para computación acelerada por GPU en Jetson.
- **cuDNN:** primitivas de deep learning (convoluciones, activaciones, transformaciones) optimizadas para GPU.
- **TensorRT:** runtime de inferencia de alto rendimiento para redes de clasificación/detección, reduce latencia y memoria.
- **OpenCV:** librería de visión por computador (con soporte de aceleración en Jetson) para preprocessamiento y utilidades.
- **Jetson Multimedia API (libargus/V4L2):** APIs de bajo nivel para cámara/video; libargus (cámaras CSI, control por-frame) y V4L2 (captura/encode/escala, GStreamer).





Consideraciones Importantes para el despliegue

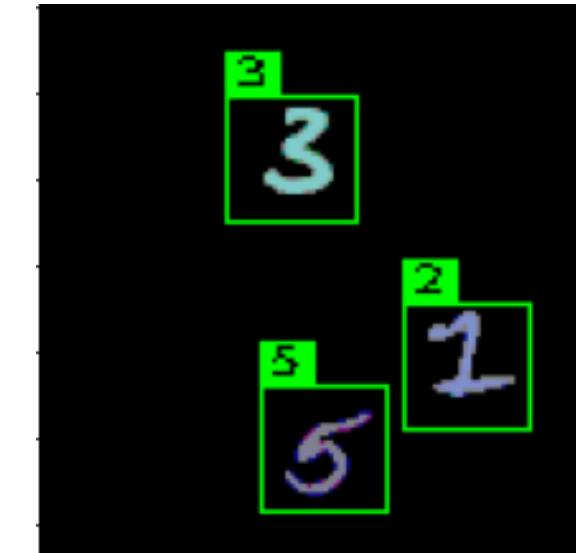
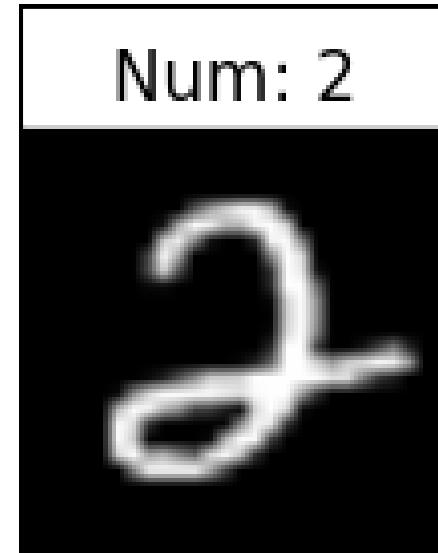
- Librerías necesarias en la Jetson.
- Datos relevantes al entrenar el modelo.
- Despliegue del modelo y visualización.



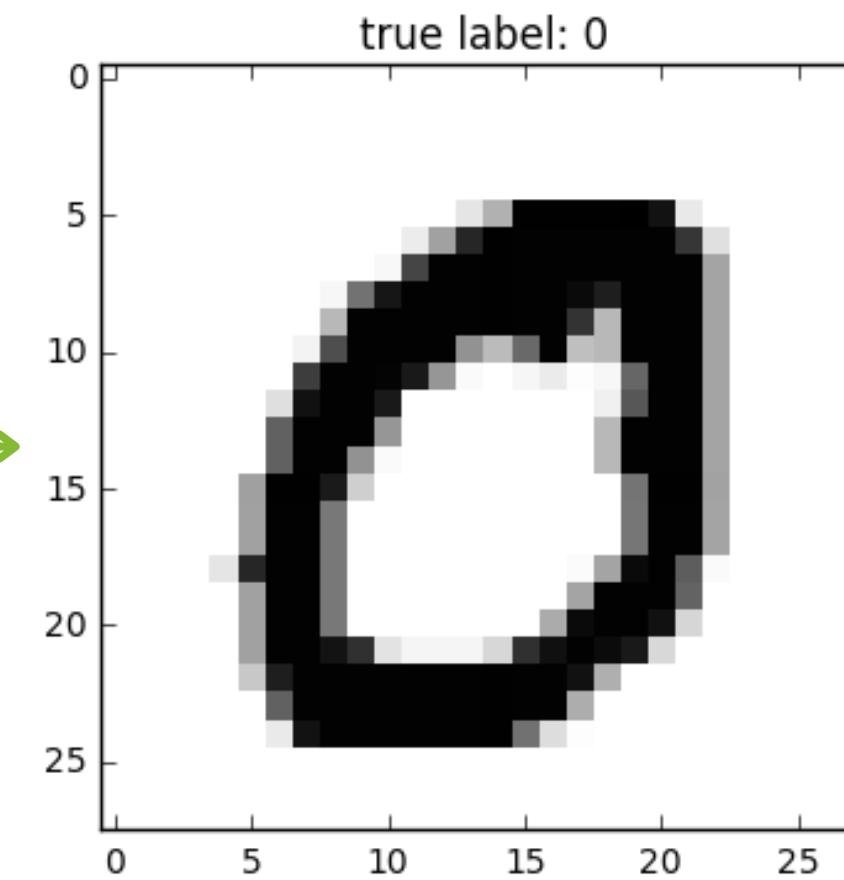
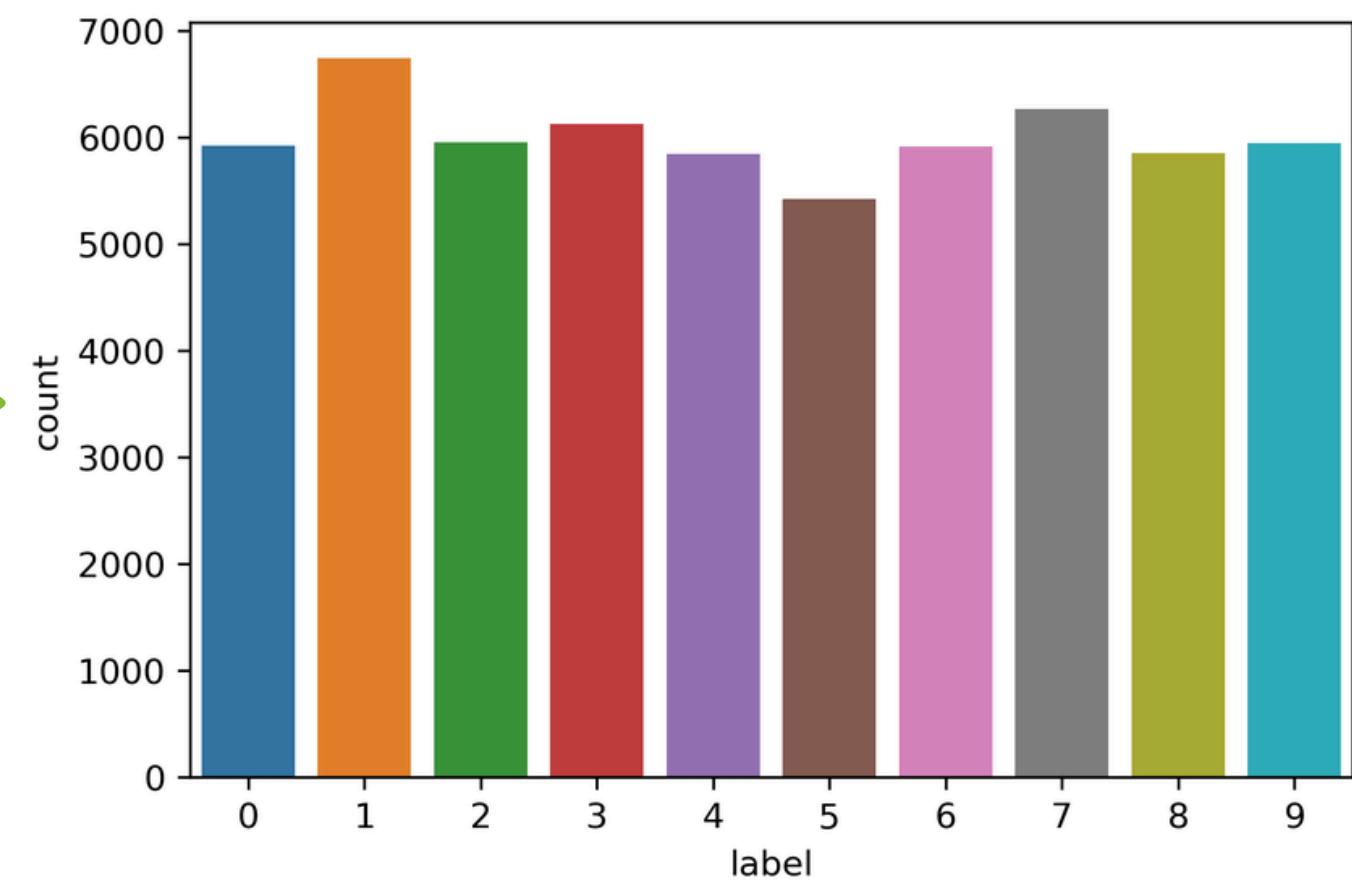


¿Qué se debe considerar de MNIST?

- Dataset de referencia para **clasificación** de dígitos.
- 60 000 imágenes de entrenamiento.
- 10 000 imágenes de prueba.
- **28×28 píxeles** en **escala de grises**.
- Etiquetas y clases: **10 clases (0-9)**.



vs





Código trainingMNIST.m

```
inputSize = [28 28 1];
classNames = categories(imdsTrain.Labels);
```

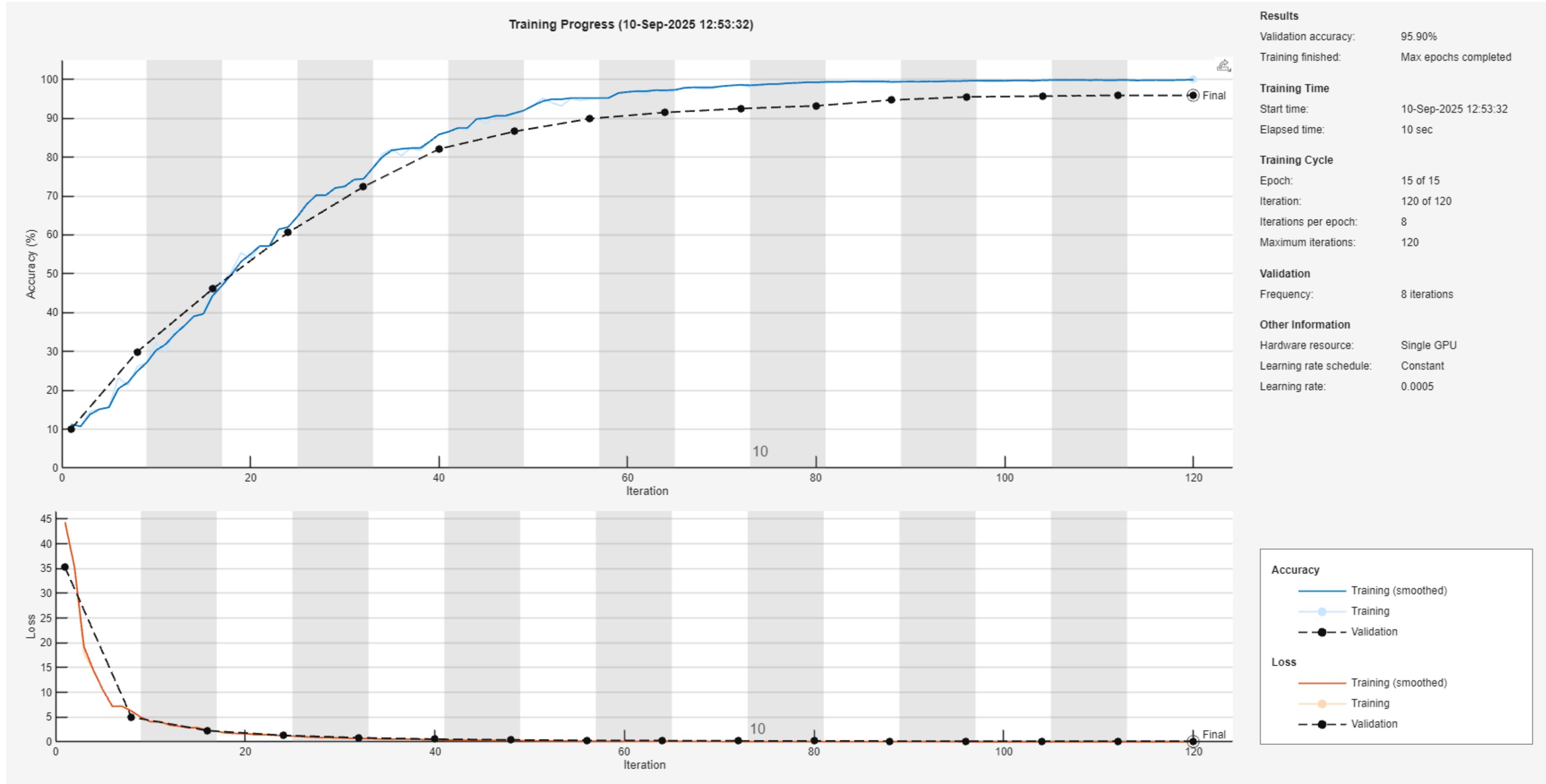
```
% -----
% MLP puro: (FC-ReLU) x 2 -> FC -> softmax
% ¡OJO! Sin normalización: 'Normalization', 'none'
% -----
% SIN escala a [0,1] (rescale-zero-one)
layers = [
    imageInputLayer(inputSize, 'Normalization', 'none')
    fullyConnectedLayer(1024)
    reluLayer
```

```
% Guardar
save(outputMat,'net','inputSize','classNames','-v7.3');
fprintf('Saved %s\n', outputMat);
```

- Fija la forma de entrada, alineando pre-procesado y arquitectura a 28×28 en grises.
- Extrae clases reales del dataset para dimensionar la última capa y mapear salidas a 0–9
- ¡Sin escalado! entrenas con valores 0–255 y debes inferir con el mismo rango (train→serve).
- MLP puro (FC-ReLU...→softmax): baseline simple y rápido para MNIST; facilita portarlo con GPU Coder.
- Guarda modelo + metadatos mínimos para reproducir entrada/clases y cargar en despliegue.

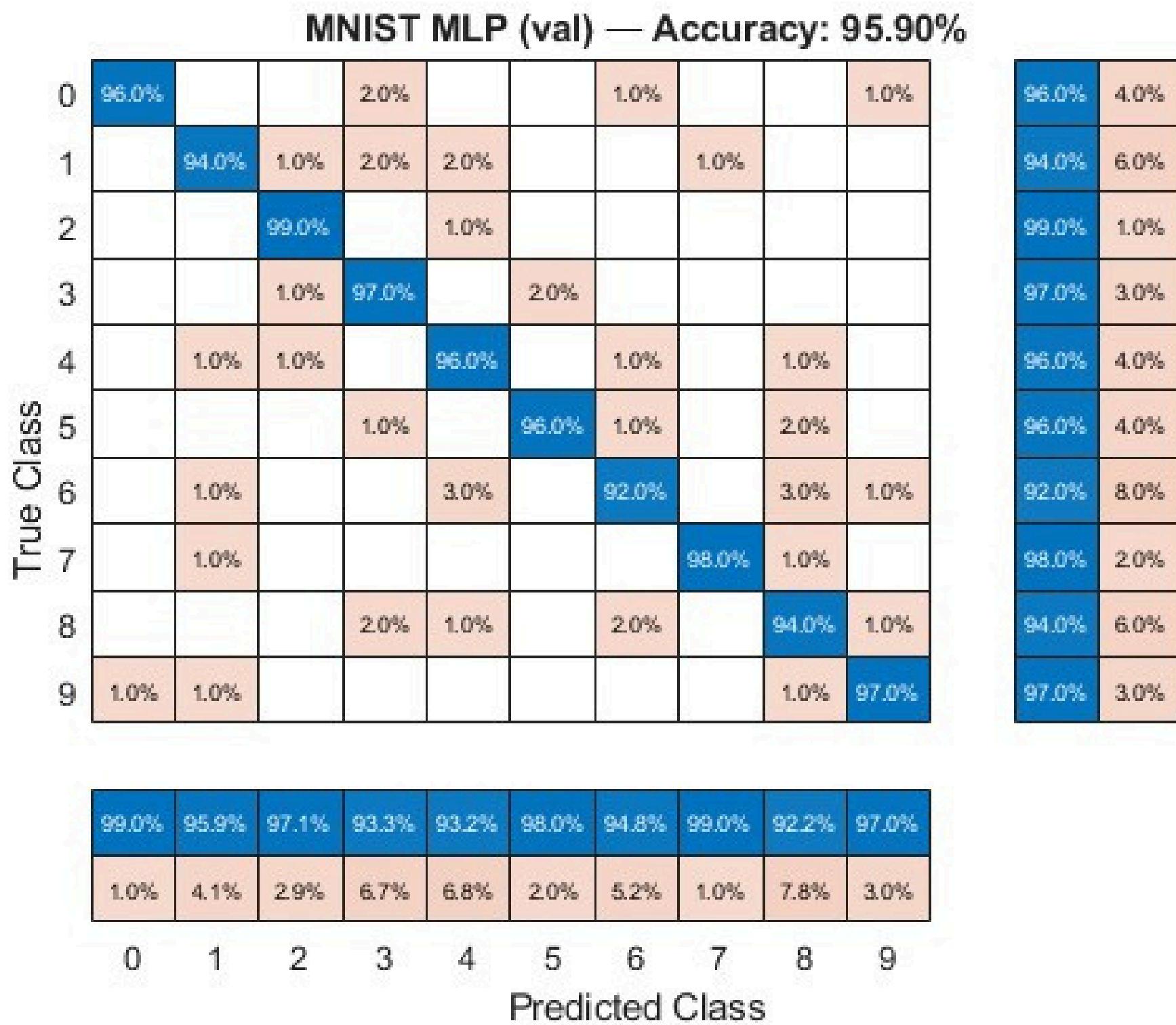


Curva de Entrenamiento





Matriz de Confusión y Prueba



Predicted: 8





Código mnistDetection.m

```
persistent net;
if isempty(net)
    % El string debe ser constante en compilación
    % net = coder.loadDeepLearningNetwork('Models/mnistNet.mat');
    net = coder.loadDeepLearningNetwork('Models/mnistMLP.mat');
end
```

Modelo persistente y carga (se inicializa una sola vez)

- **persistent net** mantiene la red viva entre iteraciones, evitando recargar del disco en cada frame.
- **La ruta al .mat debe ser constante** en compilación para GPU Coder.

```
hw    = jetson;
cam  = camera(hw, cameraName, resolution);
disp = imageDisplay(hw);
```

I/O en la Jetson (cámara y display)

- **jetson, camera e imageDisplay** crean endpoints de hardware para captura y visualización.
- En el script de despliegue, pasa **cameraName** y **resolution** como **coder.Constant**.



Código mnistDetection.m

```
frameRGB = snapshot(cam);      % uint8 HxWx3
% --- Preprocesado a 28x28x1 (MNIST) ---
gray   = rgb2gray(frameRGB);          % HxW uint8
small  = imresize(gray, [28 28], 'bilinear');    % 28x28
% in   = im2single(small);           % single [0,1]
in     = reshape(small, [28 28 1]);    % 28x28x1
```

```
scores = predict(net, in);          % 1x10
 [~, idx] = max(scores);           % 1..10
 digitChar = char('0' + (idx-1));  % '0'..'9'
```

```
labelStr = ['Digit: ' digitChar];
frameOut = insertText(frameRGB, [20 20], labelStr, ...
    'FontSize', coder.const(32), ...
    'BoxOpacity', coder.const(0.6));    % devuelve RGB
% --- Mostrar en la pantalla de la Jetson (SDL + X11) ---
image(disp, permute(frameOut, [2 1 3]));
```

Captura + preprocesamiento a 28×28×1 (consistencia con entrenamiento)

- MNIST espera **28×28** en grises, **1 canal**.
- Mantén el **mismo rango/tipo** que usaste al entrenar (sin normalizar → uint8; normalizado → im2single).

Inferencia y decodificación del dígito

- predict devuelve un vector 1×10 (probabilidades por clase).
- Convertimos el índice máximo al carácter '0'..'9'.

Overlay del resultado y visualización

- En GPU Coder, opciones como **FontSize/BoxOpacity** deben ser coder.const(...).
- **permute** ajusta el orden de ejes para el **backend** de display.



Código mnistDeployment.m

%% Conexión

```
ipaddress = "192.168.0.149";
username   = "ucspjason";
password   = "JasonNano10";
hwobj = jetson(ipaddress, username, password);
disp(hwobj);
```

Conexión al dispositivo (SSH + utilidades remotas)

- Establece la sesión con la Jetson y devuelve un objeto jetson para ejecutar comandos, copiar archivos y lanzar apps.
- Verifica que la IP/credenciales sean correctas; disp(hwobj) muestra versión de JetPack, CUDA, cuDNN, etc.

Chequeo del entorno GPU/Deep Learning

- Útil para diagnosticar que CUDA/cuDNN/TensorRT y drivers están bien instalados/visibles.
- Selecciona la librería de DL predominante que usarás al compilar con GPU Coder (puedes probar ambas).

%% (Opcional) Chequeo GPU/Deep Learning

```
gpuEnv = coder.gpuEnvConfig('jetson');
gpuEnv.DeepLibTarget = 'cudnn'; % o 'tensorrt';
gpuEnv.HardwareObject = hwobj;
results = coder.checkGpuInstall(gpuEnv);
disp(results);
```



Código mnistDeployment.m

```
%% Cámara (usa resoluciones soportadas por /dev/video0)
camlist = getCameraList(hwobj);
camName = char(camlist{1,"Camera Name"});
camResolution = [640 360];
```

Configuración de compilación para ejecutable remoto

- coder.gpuConfig('exe') crea una app nativa que corre en la Jetson (no MEX en el host).
- cfg.DeepLearningConfig define el backend de inferencia: cuDNN (general, estable) o TensorRT (más optimizado, requiere compatibilidad con capas/formatos).

Descubrir cámara y fijar constantes de compilación

- getCameraList(hwobj) devuelve las cámaras visibles en la Jetson; elige por nombre y una resolución soportada.
- Importante: cameraName y resolution se pasan como coder.Constant para generar código estático y eficiente.

```
%% Config de código (EXE para Jetson)
cfg = coder.gpuConfig('exe');
cfg.GenerateReport = true;
cfg.Hardware = coder.hardware('NVIDIA Jetson');
cfg.Hardware.BuildDir = '~/remoteBuildDir';
cfg.GenerateExampleMain = 'GenerateCodeAndCompile';

% Acelera inferencia con TensorRT (o cuDNN):
cfg.DeepLearningConfig = coder.DeepLearningConfig('cudnn'); % Jetson
% alternativa: cfg.DeepLearningConfig = coder.DeepLearningConfig('tensorrt');

% Entradas constantes del entry-point
inputArgs = {coder.Constant(camName), coder.Constant(camResolution)};

% Asegúrate de que 'mnistNet.mat' esté en el path actual (se usa en codegen)
codegen('-config', cfg, 'mnistDetection', '-args', inputArgs, '-report');
```



Código mnistDeployment.m

```
% Entradas constantes del entry-point
inputArgs = {coder.Constant(camName), ...
    coder.Constant(camResolution)};

% 'mnistNet.mat' debe estar en el path (se usa en codegen)
codegen('-config', cfg, 'mnistDetection', ...
    '-args', inputArgs, '-report');
```

Autorizar X11 y lanzar la app en la Jetson

- Para mostrar ventana en la Jetson, configura DISPLAY en el dispositivo y permite al usuario X11 con xhost.
- Ejecuta y detén la app remotamente con runApplication/killApplication.
- Nota: prueba DISPLAY=:0 y :1 según tu sesión gráfica.

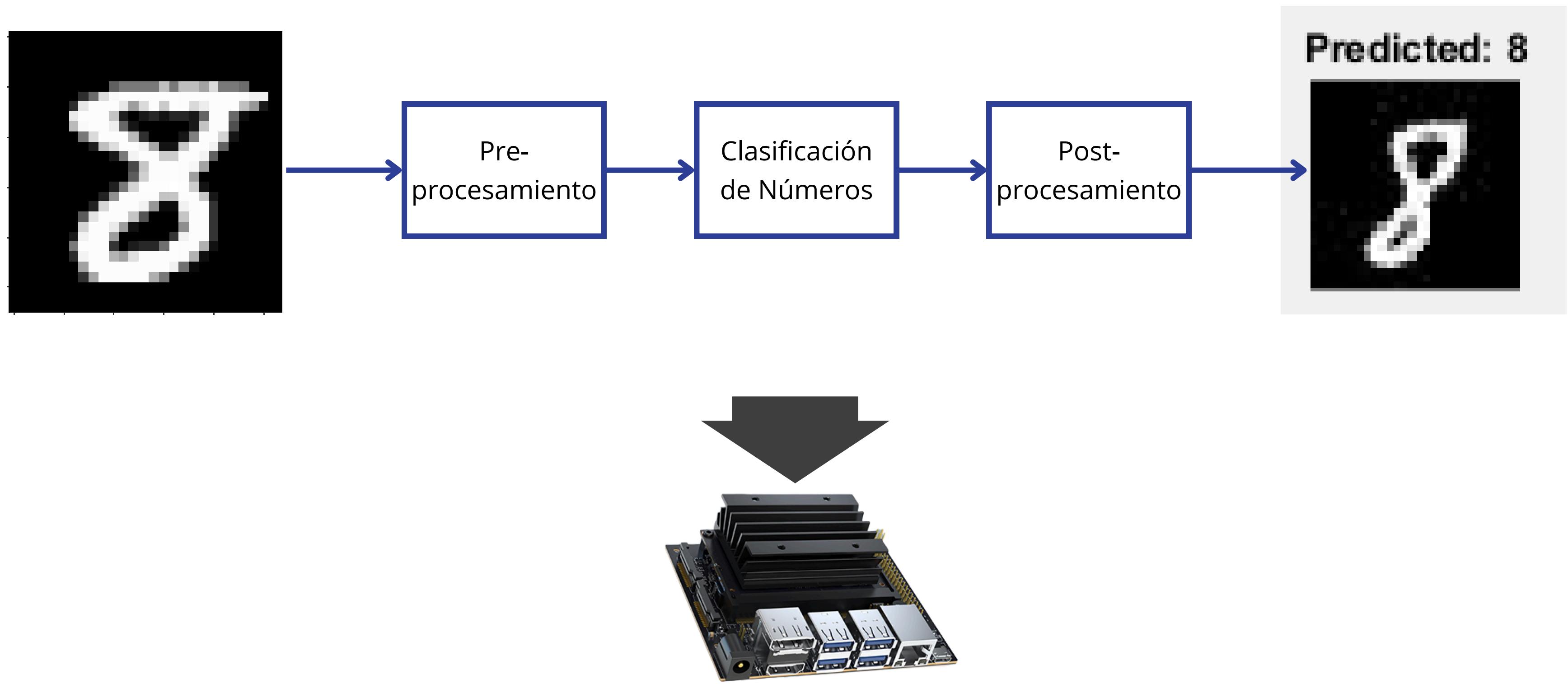
Argumentos de entrada constantes y generación del binario

- El entry-point **mnistDetection(cameraName, resolution)** exige constantes de compilación para permitir inlining/optimizaciones.
- **OJO:** El .mat de la red debe ser accesible en el directorio actual antes de codegen.

```
% Prueba ambos displays y da permiso al usuario ante el servidor X.
for dispVal = ["0.0","1.0"]
    try
        setDisplayEnvironment(hwobj, char(dispVal));
    end
    try
        dshort = extractBefore(dispVal,'.');
        system(hwobj, ...
            sprintf('bash -lc "DISPLAY=%s xhost +SI:localuser:%s"', ...
                dshort, username));
    end
end
```



¡Vamos a Desplegar!



¿PREGUNTAS?

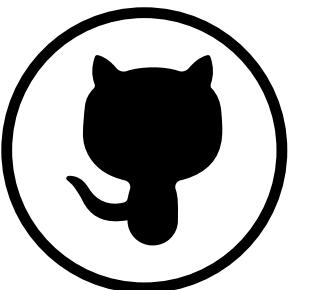


Diego Banda R.
Universidad Católica San Pablo
diego.banda@ucsp.edu.pe





Página de recursos



DiegoABR07/Deep-Learning-Deployment...

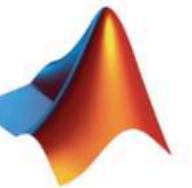


Implementation of object detection and classification models on NVIDIA Jetson Nano Developer Kit, using MATLAB R2024a, GPU Coder, cuDNN, TensorRT...

1 Contributor 0 Issues 0 Stars 0 Forks



matlab-deep-learning/MATLAB-Deep-Learning...



Discover pretrained models for deep learning in MATLAB

4 Contributors 12 Issues 533 Stars 117 Forks

