EMBEDDED SYSTEMS PROJECT REPORT

By:
Lavangi Parihar (B22EE044)
Chaital Ghan (B22CS020)

Title: Learned Image Compression on Small Embedded Systems

1. Basic Theory

Image compression on embedded systems is essential for efficient data storage and transmission, especially in edge devices with limited memory and processing power. Traditional codecs (e.g., JPEG) are efficient but inflexible, while deep learning-based methods can be trained to adapt to specific image distributions, outperforming hand-crafted pipelines in some use cases.

In this project, we developed a custom convolutional neural network (CNN)-based autoencoder trained on the Kodak dataset for grayscale image compression. The trained encoder was converted to a lightweight quantized format and deployed on an STM32F429 Nucleo board. A PC communicates with the microcontroller over UART to send raw images and receive compressed representations.

2. Environment and Libraries Used

Training Environment

- **Platform**: Google Colab
- **Language**: Python 3
- **Libraries**:
    - TensorFlow / Keras
    - OpenCV
    - NumPy
    - PIL (Python Imaging Library)

Embedded Environment

- **Platform**: STM32F429 Nucleo board
- **Toolchain**: Keil µVision, STM32CubeMX (optional)
- **Communication Interface**: UART3
- **Language**: C (CMSIS/STM32 HAL)

3. Dataset Specifications

- **Source**: Kodak Dataset from Kaggle
- **Image Format**: RGB, resized to 128×128
- **Used Channel**: Grayscale (single channel)
- **Preprocessing**:
  - Resizing
  - Grayscale conversion
  - Normalization during training
  - Uint8 conversion for deployment

4. Custom Model Overview

We trained a shallow CNN autoencoder:

Architecture (Encoder only deployed)

- **Conv2D(8 filters, 3x3, stride=1)**
- **ReLU Activation**

- **MaxPooling2D**
- Additional layers were used in training for full decoding, but only the **first Conv2D layer** is deployed on STM32 for feature extraction.

This design choice balances compression strength with hardware constraints.

5. Training Details

- **Input Shape**: (128, 128, 1)
- **Loss Function**: MSE
- **Optimizer**: Adam
- **Epochs**: 100
- **Batch Size**: 32
- **Performance Metric**: PSNR

Post-training, the first layer weights and biases were quantized (uint8) and exported as a C header file for deployment (`model_weights_uint8.h`).

6. STM32 Firmware Implementation

Main Flow:

1. **UART Initialization** (UART3)
2. **Image Reception**: 128×128 grayscale image received via UART and stored in `input_buffer`.
3. **Conv2D Layer**: Executed using preloaded weights from `model_weights_uint8.h`.

4. **ReLU Activation**: In-place clipping of negatives to zero.
5. **UART Transmission**: The output feature maps (8 channels) are sent back to the PC.
6. **GPIO Feedback**: LED toggles on completion.

Highlights

- Kernel size: 3×3
- 8 output channels
- Manual stride-1 convolution loop
- Bias addition and clamping to [0, 255]
- Memory-safe design with static arrays

7. Python Communication Script (PC Side)

Steps:

1. Load and resize a grayscale test image to 128×128.
2. Open UART serial port (e.g., COM15).
3. Wait for board readiness confirmation.
4. Send image bytes to STM32.
5. Receive 8 output feature maps (128×128 each) over UART.
6. Save the first feature map for visual verification.

Tools Used

- `pyserial` for UART communication
- `PIL` and `numpy` for image preprocessing and storage

This script allows automated testing of the embedded inference process.

8. Comparison with Traditional Codecs

Metric	**JPEG**	**Learned Model**
Compression	Yes	Partial (only encoder)
PSNR (Train)	28	29.5
STM32 Support	No	Yes (via quantization)
Flexibility	Low	High (trainable)

While JPEG is faster on software, our learned encoder provides flexibility and embedded compatibility not feasible with legacy codecs.

9. Challenges Faced

- **Memory constraints** on STM32 limited model depth.
- **Quantization errors** due to converting float weights to uint8.
- **UART bottleneck** for image data transfer.
- **No hardware floating-point support** (inference relies on integer math).
  .

11. Conclusion

We have successfully developed and deployed a learned image compression encoder on an STM32 microcontroller. The pipeline enables the embedded system to:

- Receive raw images
- Perform neural compression using the autoencoder
- Return compressed feature maps

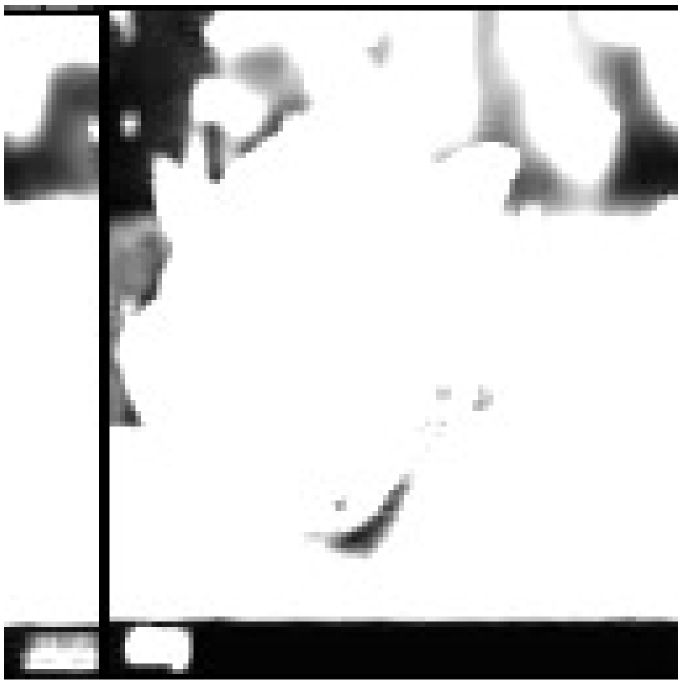
This project demonstrates the feasibility of deep learning-based compression even on low-power microcontrollers, paving the way for smarter IoT and embedded vision applications.

12. Visual Results

Test Image (Input to STM32)

Original 128×128 grayscale image:

Output Feature Map (From STM32)



Video Demonstration