Building Lightweight Architectures:

Welcome

This book is your guide to the exciting world of **Tiny Machine Learning (TinyML)**. We will explore how to build intelligent systems that can run directly on small, low-power devices, opening up a world of possibilities for real-time decision-making at the edge.

1. Introduction to TinyML - Intelligence at the Edge

Imagine a self-driving car speeding down a highway. Suddenly, a child runs into the road. The car's sensors detect the obstacle. Now, what happens next determines whether an accident occurs.

Scenario 1: Cloud-Based Processing (The Traditional Way)

The car sends the sensor data (camera images, radar readings, etc.) to a powerful cloud server located miles away via the internet. The cloud server analyzes the data, identifies the child, and sends instructions back to the car to brake.

This process takes time – perhaps half a second, maybe even two. In that critical moment, even a fraction of a second can mean the difference between a safe stop and a collision. This delay is known as **latency**.

Scenario 2: TinyML - Intelligence on the Device

Instead of sending data to the cloud, the car itself has a small, specialized computer running a TinyML model. This model has been pre-trained to recognize obstacles and make decisions instantly. When the child runs into the road, the TinyML model recognizes the danger immediately and commands the car to brake. This decision is made in real time, eliminating latency and significantly improving safety.

This simple example illustrates the core idea behind TinyML: bringing intelligence to the edge.

Why is TinyML Important?

- **Low Latency:** As the car example shows, TinyML enables real-time decision-making without relying on a network connection or cloud processing.
- **Privacy:** Data is processed locally on the device, enhancing privacy as sensitive information does not need to be transmitted over the internet.
- **Power Efficiency:** TinyML models are designed to run on resource-constrained devices with limited power, making them ideal for battery-powered applications.
- **Connectivity Issues:** TinyML allows devices to continue operation even without a network connection.
- **Cutting-Edge Technology:** TinyML is a relatively new and rapidly evolving field, putting you at the forefront of innovation.

2. Understanding the Building Blocks of TinyML

Let us explore the main stages involved in building a TinyML application. We will use a **fall detection system for the elderly** as our running example. Imagine an elderly person living alone. A fall could have severe consequences, and timely assistance is crucial. A TinyML-powered device could detect falls and alert emergency services.

The four major stages are:

1. Data Collection

- **The Idea:** Just like learning to recognize your uncle over time, TinyML models need data to learn. This involves gathering sensor data that represents different scenarios.
- **Our Example:** For our fall detection system, we need data representing both *safe* and *fall* conditions. The safe data could include normal walking, sitting, or lying down movements. The fall data would simulate the motion of a person falling.
- **Sensors:** Smartphones can be used as the training device and contain a variety of sensors. Accelerometers capture motion along three axes (x, y, z). Gyroscopes measure rotational velocity. Magnetometers measure magnetic fields. Combining these provides a comprehensive understanding of device movement.

2. Feature Extraction

- **The Idea:** The raw sensor data needs to be converted into meaningful features that the TinyML model can understand. Think of it as highlighting the key char-

acteristics that distinguish a *safe* movement from a *fall*.

- **Spectral Analysis:** Spectral analysis helps identify the frequency components within the sensor data. The sensor data needs to undergo this kind of transformation to bring out meaningful insights.

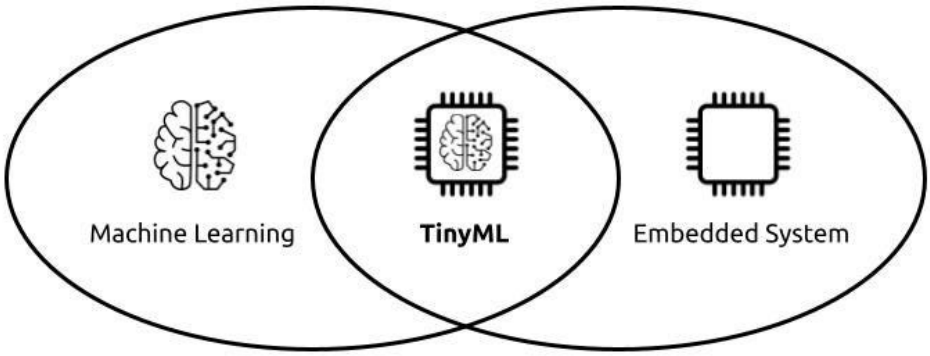

Figure 1: Edge impulse cycle

3. Model Training

- **The Idea:** The TinyML model learns from the extracted features to distinguish between different scenarios. This is where the training happens.
- **Neural Networks:** Inspired by the structure of the human brain, neural networks consist of interconnected neurons that process information. The training process adjusts the connections between these neurons to improve model accuracy.
- **Training Cycles (Epochs):** The model needs to be trained for a certain number of epochs or cycles. Too few epochs may result in underfitting and poor performance. Too many epochs may lead to overfitting, where the model learns the training data too well but performs poorly on new, unseen data.

4. Deployment

- **The Idea:** The trained TinyML model is deployed to a small, low-power device where it can operate independently and make real-time decisions.
- **Microcontrollers:** Microcontrollers are small, embedded computers that are commonly used in TinyML applications. Some microcontrollers, like the Arduino Nano 33 BLE Sense, are specifically designed to support AI workloads.

3. Practical Considerations for Building TinyML Architectures

While building a TinyML application, there are several practical considerations:

- **Window Size:** Sensor data is analyzed in windows of time. The window size defines the length of each window. For example, a 2-second window means the model analyzes 2 seconds of data at a time. A smaller window can lead to faster reaction times but may also be more susceptible to noise.
- **Data Splitting:** It is important to split your data into training and testing sets. A common split is 80% for training and 20% for testing. This helps evaluate how well your model generalizes to unseen data.
- **On-Device Performance:** Key metrics include processing time (latency), RAM usage, and flash memory usage. You must ensure your model can run efficiently within the device constraints. On-device performance helps identify real-time system efficiency in finding intelligence on time.

4. Building and Improving Your TinyML Models

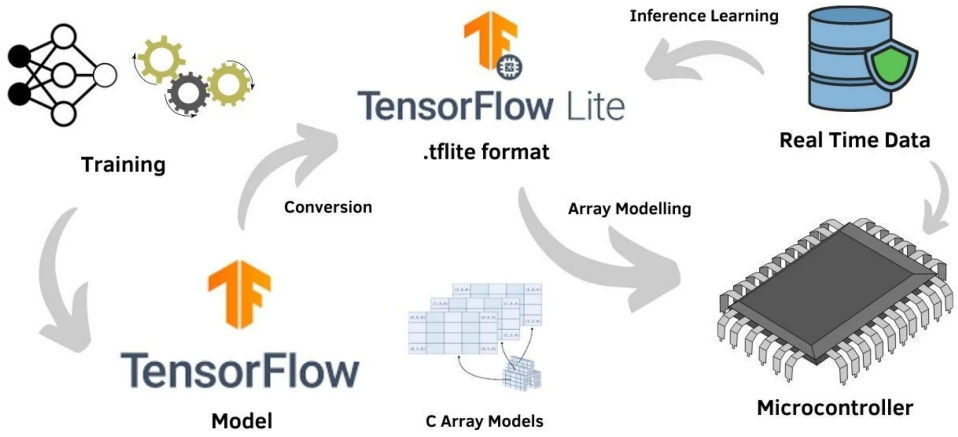

Figure 2: Edge impulse cycle

- **Accuracy:** The accuracy of your model is crucial. Aim for high accuracy, but be aware of the risk of overfitting.
- **Interpreting Results:** The model outputs a probability score for each class (e.g., *safe* and *fall*). The higher the score, the more confident the model is in its prediction.
- **Improving Accuracy:** If model accuracy is low, consider:

  - **Data Quality:** Ensure that your data is clean and representative of real-world conditions.
  - **Feature Engineering:** Experiment with different features to improve the model's ability to distinguish between classes.
  - **Model Architecture:** Try different neural network types or adjust the number of layers and neurons.
  - **Addressing Confusion:** Identify where the model consistently makes incorrect predictions to find improvement areas.
- **Inference:** Inference involves deploying and testing the model in real-world scenarios.

Conclusion

This book provides a foundational understanding of TinyML and its practical applications. By understanding the key concepts and following the guidelines outlined, you can start building your own intelligent systems that operate directly on small, low-power devices, bringing the power of machine learning to the edge.

Now, get out there and explore the endless possibilities of TinyML!