**📌 Onde Implementar a Suavização? No Treinamento ou no Aplicativo Mobile?**

A **suavização** que propomos é um **processo de pós-processamento**, ou seja, ela **não afeta o treinamento** da rede neural. O treinamento continua focado em aprender os padrões dos dados, enquanto a suavização ajuda a reduzir falsos positivos **durante a inferência em tempo real**.

✅ **Resposta:** **A suavização deve ser implementada no aplicativo mobile**, onde o modelo processa imagens capturadas em tempo real.

**🛠 Como Aplicar no Aplicativo Mobile?**

Se o seu modelo já está sendo exportado para TensorFlow Lite (TFLite) e rodando no mobile, a suavização deve ser aplicada **após a saída do modelo**, antes de emitir um alerta sonoro.

**📌 Fluxo do Processamento no Aplicativo Mobile**

1️⃣ O aplicativo **captura** uma imagem da câmera.  
2️⃣ O modelo **classifica** a imagem (caminho livre ou obstáculo).  
3️⃣ O resultado da inferência é **suavizado** usando uma **janela deslizante**.  
4️⃣ Se o filtro indicar **obstáculo consistente**, um **sinal sonoro de alerta** é emitido.

**🔹 Código Kotlin para Implementar a Suavização no Mobile**

Se você está usando **Kotlin** para rodar a inferência do modelo, pode implementar o filtro de suavização assim:

class HistoryFilter(private val windowSize: Int = 3) {

private val history = mutableListOf<Float>()

fun update(newValue: Float): Float {

history.add(newValue)

if (history.size > windowSize) {

history.removeAt(0) // Mantém apenas os últimos N valores

}

return history.average().toFloat() // Retorna a média móvel

}

}

**🔹 Aplicando no Ciclo de Inferência**

Dentro da função que processa a imagem e obtém a saída do modelo TFLite:

val historyFilter = HistoryFilter(windowSize = 3) // Suavização com janela de 3

fun processFrame(bitmap: Bitmap) {

val modelOutput = runInference(bitmap) // Obtém a saída do modelo (ex: probabilidade de obstáculo)

val filteredOutput = historyFilter.update(modelOutput) // Suaviza a saída

if (filteredOutput > THRESHOLD) {

playObstacleAlert() // Emite som de obstáculo

} else {

playClearPathSound() // Som de caminho livre

}

}

📌 **Importante:**

* O THRESHOLD pode ser ajustado com base nos resultados do seu modelo (ex: 0.6).
* A função runInference(bitmap) executa a inferência no TFLite e retorna a saída do modelo (probabilidade de obstáculo).
* playObstacleAlert() e playClearPathSound() controlam os sinais sonoros para o usuário.

**🎯 Por que implementar no Mobile?**

✔ **Reduz falsos positivos em tempo real** sem precisar retreinar a rede.  
✔ **Torna o sistema mais estável**, evitando mudanças bruscas entre "obstáculo" e "caminho livre".  
✔ **Otimiza o processamento**, pois a suavização ocorre após a inferência, sem impactar o desempenho do modelo.

🚀 **Conclusão:** A suavização deve ser aplicada **no aplicativo mobile** para garantir decisões mais confiáveis, sem necessidade de modificar o treinamento do modelo.