UNIVERSIDADE FEDERAL DE ITAJUBÁ

IESTI05 - MACHINE LEARNING SYSTEMS ENGINEERING

Part 1 Final Project: Fixed Function AI Implementation

Eduardo José de Souza Castro

2021009360

ITAJUBÁ

2025

1. INTRODUÇÃO:

Este projeto tem como objetivo principal validar os conhecimentos adquiridos na disciplina de IESTI05 - MACHINE LEARNING SYSTEMS ENGINEERING com relação a treinamento, implementação e teste de modelos de IA para detecção de objetos. O tema escolhido foi a detecção de gestos de um jogo bastante conhecido chamado “JOKENPO” ou “pedra, papel, tesoura” e teve como justificativa o deployment bastante simples e fácil de aplicá-lo dentro do contexto do hardware disponibilizado e da disciplina propriamente dita.

Levando isso em consideração, não foi de interesse optar por temáticas mais aplicáveis na indústria ou mercado de trabalho em virtude das dificuldades de deployments e análise prática.

2. METODOLOGIA:

Para iniciar a aquisição de dados foi, primeiramente, proposto a elaboração de um dataset próprio com imagens com as três classes propostas e imagens sem detecções que serviriam de background. As três classes propostas são as clássicas do jokenpo e já são mencionadas em seu nome popular: rock, paper and scissors. Dessa forma, foi também proposto a busca por dataset prontos e já anotadas na plataforma Roboflow, onde pôde-se encontrar um dataset bem completo que já era a união de outros dataset como pode ser verificado em seu project overview no Roboflow:
https://app.roboflow.com/myworkspace-ihc2s/rock-paper-scissors-sxsw-qcnlp/overview

Com a finalidade de fazer um teste inicial com a temática, foi feito um treinamento com YOLO utilizando o YOLOv11 Nano como modelo pré treinado para fazer o transfer learning e o deploy em uma máquina HP G9 com Intel Core I3 e 8gb de RAM como ser visto na figura 01:

Figura 01: deployment inicial em notebook.

As etapas posteriores se basearam em testes com os modelos com variações de parâmetros de treinamento para que fosse alcançado um ponto ótimo entre acurácia e eficiência, no que se refere à quantidade de frames por segundo. Com esse objetivo o treinamento foi refeito algumas vezes e o deploy feito dentro da Raspberry PI Zero com o modelo otimizado ncnn, como pode ser visto na figura 02.

Figura 02: deployment do modelo na Raspberry PI Zero

A figura 02 mostra o deploy do primeiro modelo treinado com 50 épocas, o qual serviu de base para rever alguns detalhes tanto da implementação do stream da detecção quando do treinamento, que deveriam ser mais fluído (maior FPS e

sincronia dos frames com a bounding box) e de uma melhora nas métricas de avaliação do modelo, respectivamente. Para tanto, foram realizados respectivos treinamentos, análises e revalidações para encontrar um resultado satisfatório.

3. RESULTADOS E ANÁLISE:

O primeiro treinamento realizado se deu por 50 épocas, com o yolo11 nano como modelo base, imagem de input de resolução de 640x640, onde foram obtidos os gráficos de perdas e precisão da figura 03 e a matriz de confusão da figura 04.

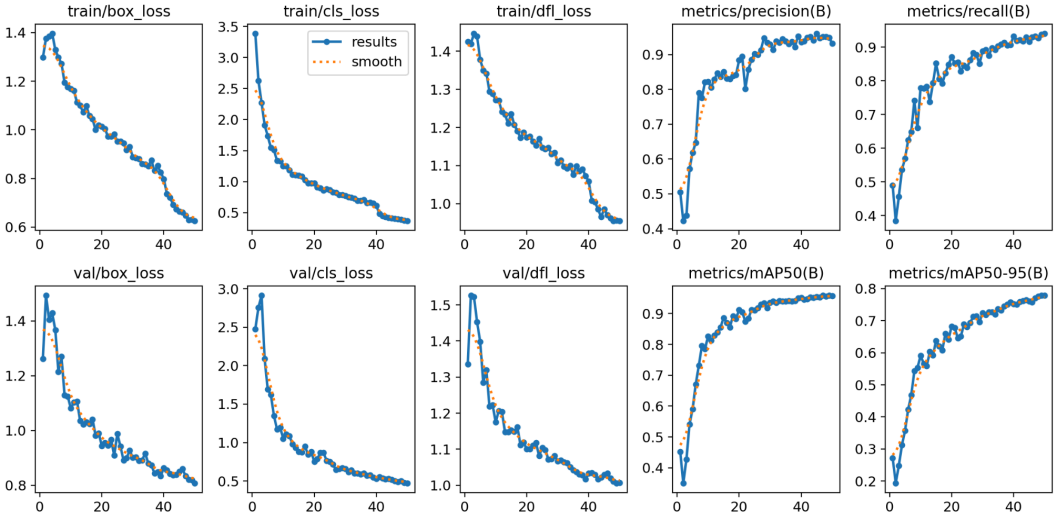

Figura 03: gráficos de precisão e perdas do primeiro treinamento.

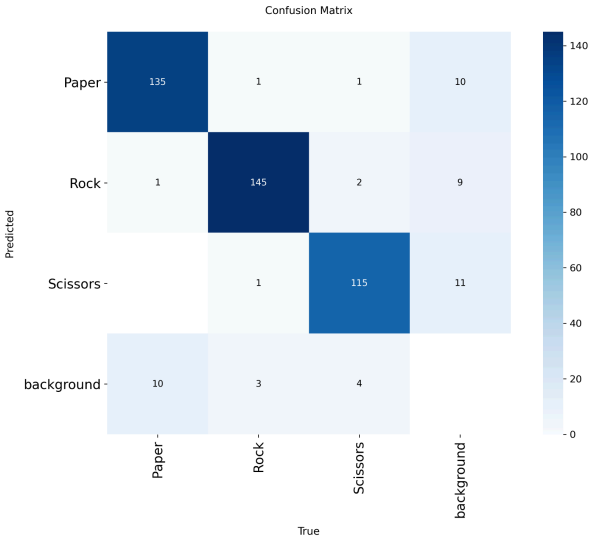

Figura 04: matriz de confusão do primeiro treinamento.

Os gráficos e a matriz de confusão permitiram visualizar que os resultados se apresentaram satisfatoriamente bons, muito em virtude da elaboração bem feita do dataset, mas com grande potencial de melhora, o que pode ser claramente percebido pela crescente nas curvas de precisão e recall que demonstra, ainda, um ascendência contínua, bem como as curvas referentes às perdas reproduzem uma decadência que tende a ser manter. Além disso, com o script do deploy na raspberry PI zero utilizando as bibliotecas ultralytics, flask, PIL e outras foi possível fazer o deployment diretamente no hardware final, tendo um desempenho razoável na detecção, mas medíocre na eficiência em aplicações real time, mesmo com a conversão para o formato otimizado ncnn.

O próximo modelo treinado teve como parâmetros 150 épocas e uma diminuição na resolução de entrada para 320x320 com a finalidade da posterior avaliação de seus efeitos em stream sem que haja perdas significativas na acurácia. Com relação aos gráficos dos resultados de treinamento pode se observar na figura 01 o prosseguimento do que se chamou anteriormente de tendência à diminuição das métricas de perda, bem como ao aumento das métricas de precisão e recall. O que confirmou a necessidade do treinamento por mais épocas.

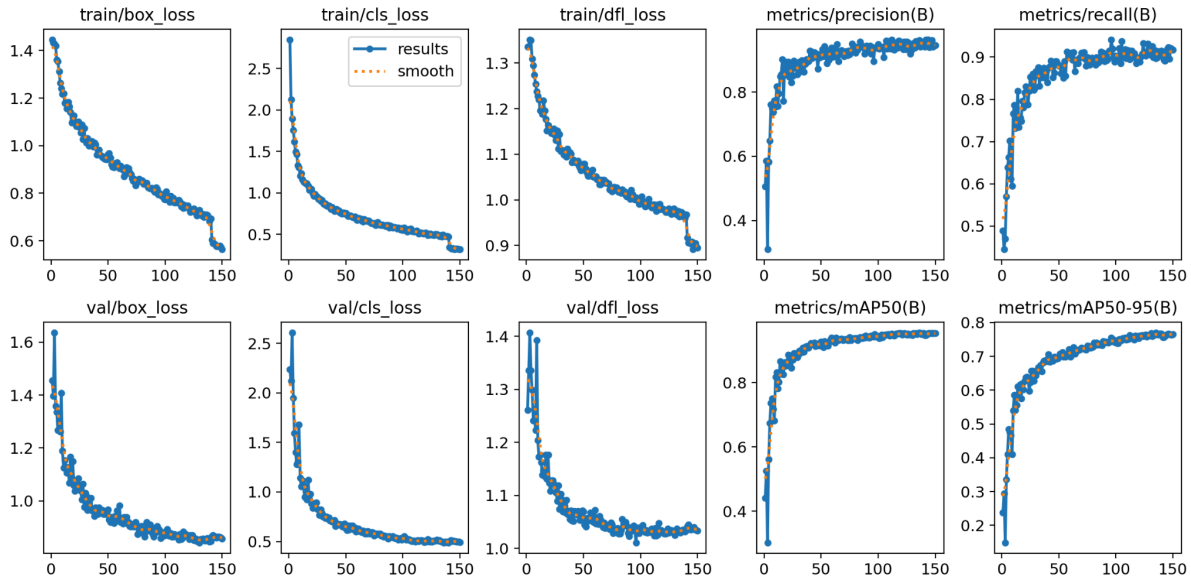

Figura 05: gráficos de precisão e perdas do treinamento 2.

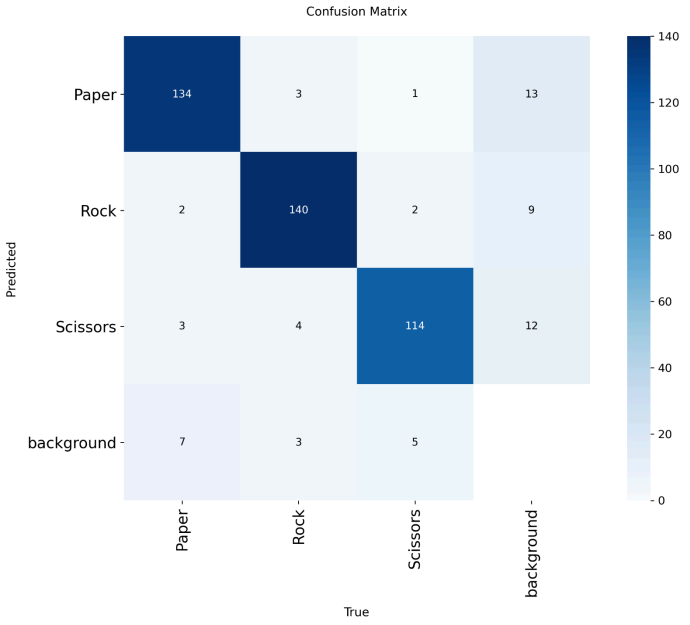

Figura 06: matriz de confusão do treinamento 2.

Dessa forma, com esse novo treinamento, foi possível alcançar maior eficiência, maior acurácia e uma fluidez maior no stream no servidor flask. Colocando essas informações em números, observou-se que, com a picamera funcionando sem inferência em tempo real, o fps é na casa dos 7, 8. Já com a inferência desse último modelo treinado, a taxa de quadros caiu para 4, 5.

Figuras 07 e 08: taxa de quadros sem e com inferência em tempo real.

4. CONCLUSÃO

Tendo em vista o exposto, os testes, validações e treinamentos do modelo foram necessários para que fosse atingido o objetivo. De modo que existisse um trade-off bem estruturado entre acurácia, FPS e sincronismo. O projeto se pautou em um dataset já anotado bem promissor que se mostrou bem elaborado e os treinamentos confirmaram as primeiras impressões. Como foi possível verificar no presente relatório, o cerne do trabalho consistiu em integrar o modelo treinado com um hardware de borda capaz de gerar um servidor web para streamar em tempo real a captura das imagens, bem como a detecção.

Em suma, o código final, que implementou a união de todos os quesitos se apresentou de forma razoável, capturando a imagem, realizando a inferência, desenhando as bounding boxes no frame, calculando o FPS e gerando um servidor Flask interativo capaz de exibir a imagem da câmera da raspberry em tempo real.