Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação

Trabalho 2 (Final)

Sistemas Embarcados para classificação de imagens e reconhecimento de objetos com TensorFlow Lite

SCC0740 - Sistemas Embarcados
Profº Vanderlei Bonato

Lucas Fernandes da Nobrega - 9805320
João Pedro Doimo Torrezan - 9806933
Leonardo Alves Paiva - 10276911

Sumário

Introdução

Tensor Flow

O TensorFlow é uma plataforma completa de código aberto para machine learning. Foi criado pela equipe Brain do Google para uso interno da empresa, porém com o tempo foi liberado para uso geral. Essa biblioteca reúne vários modelos e algoritmos de machine learning e *deep learning*.

A plataforma pode ser usada para treinamento de redes neurais profundas como por exemplo, reconhecimento de dígitos manuscritos, classificação de objetos em imagens, processamento de linguagem natural, dentre outras funções. O TensorFlow fornece tudo isso para o programador por meio da linguagem Python, que é fácil de aprender e trabalhar. Entretanto, o Python é usado para a transferência e manutenção dos arquivos e modelos, já as operações matemáticas reais são feitas com módulos em C++, para conferir maior desempenho, visto que C++ é uma linguagem de baixo nível, fazendo menos abstrações, otimizando assim o código.

O TensorFlow é executado majoritariamente em ambientes como nuvens, máquinas locais, CPUs e GPUs. Entretanto seria interessante também que pudesse ser executado em dispositivos da borda da rede, como dispositivos móveis, sistemas embarcados, microcontroladores. Isso seria muito melhor do que o dispositivo na borda coletar os dados e enviar a um servidor para fazer o processamento, as vantagens primordiais seriam as seguintes:

- Latência: não há ida e volta para um servidor.
- Privacidade: nenhum dado precisa sair do dispositivo.
- Conectividade: não exige conexão de Internet.
- Consumo de energia: as conexões de rede consomem muita energia.

Com isso em mente foi desenvolvido o TensorFlow Lite, o qual confere aos dispositivos embarcados e dispositivos de IoT a possibilidade da integração com o Machine Learning, mesmo muitas vezes possuindo pouca memória e baixo processamento.

O TensorFlow Lite tem dois componentes principais, o interpretador e o conversor. O primeiro executa os modelos do TensorFlow otimizados em diversos tipos de hardware, como smartphones, linux embarcados e até microcontroladores, como o ESP32. Já o segundo é responsável por fazer a conversão dos modelos TensorFlow para TensorFlow Lite, e o interpretador pode executar esses modelos. Diminuindo o tamanho e a performance dos binários.

Para uso da ferramenta TensorFlow Lite utilizamos um aplicativo Mobile, que roda na plataforma Android, e aplica a classificação de imagens para identificar, continuamente, toda imagem vista a partir da câmera traseira do smartphone. O aplicativo mostra em tempo real todas as opções de classificação do objeto capturado pela câmera. É possível ainda classificar objetos utilizando de 2 diferentes modelos,

cada um deles de 2 formas, selecionar o número de threads usadas na computação e também decidir se o aplicativo rodará em CPU, GPU ou via NNAPI (Neural Network API). Este último é uma API de processamento do Android projetada e desenvolvida para aplicações que envolvam o uso intensivo de computação para aprendizado de máquina, funcionando muito bem com o TensorFlow Lite.

Os modelos viabilizados pelo aplicativo são MobileNetV1 e EfficientNetLite, ambos são desenvolvidos para utilização em dispositivos de borda de rede. Ambos com a opção de utilizar os dados como ponto flutuante ou então com o modelo quantizado. A quantização pós-treinamento é uma técnica de conversão que pode reduzir o tamanho do modelo e, ao mesmo tempo, melhorar a latência da CPU e do acelerador de hardware, com pouca degradação na precisão do modelo.

Na seção seguinte, tabelas relacionando os dois modelos foram feitas para fins de comparação. A comparação foi feita entre o valor de ativação dos neurônios de cada um dos modelos e o tempo de processamento de cada um deles.

Materiais e Métodos

Para fazer as comparações dos métodos e parâmetros utilizados no aplicativo, faremos o reconhecimento de diversos objetos. Eles são:

1. Caneta esferográfica;
2. Raquete de Tênis;
3. Tênis;
4. Bola de tênis;
5. Controle remoto;
6. Garrafa PET;
7. Mouse;
8. Fósforo;
9. Régua; e
10. Banana.

Além destes objetos para classificação, foi utilizado um smartphone Xiaomi Redmi Note 8 (Android 10) para executar o aplicativo e o ambiente de desenvolvimento Android Studio para compilação. Segue um vídeo com a execução da aplicação: https://drive.google.com/file/d/1lrH96ktsMRSsbglGY25r-Bs1k9mXvY8r/view?usp=sharing.

Modelos

Como dito anteriormente, os modelos pré-treinados aplicados para a classificação de imagens são *MobileNetV1* e *EfficientNetLite*, ambos para ponto flutuante ou quantizados.

MobileNets

As *MobileNets* são uma classe de convolução de redes neurais artificiais criada pelo Google. Elas são projetadas para maximizar a precisão eficientemente, enquanto consideram as restrições de recursos para um aplicação embarcada.

Seus modelos são pequenos, possuem baixa latência e baixo consumo de energia, parametrizados para atender com recursos restritos a uma variedade de situações de uso. Eles podem ser usados de forma semelhante a outros modelos populares de grande escala, como o Inception.

EfficientNets

As *EfficientNets*, propostas inicialmente em 2019, são um novo tipo de modelo que usa um coeficiente composto simples, mas altamente eficiente, para aumentar a escala de CNNs de maneira estruturada. Ao contrário das abordagens convencionais que dimensionam arbitrariamente as dimensões da rede, como largura, profundidade e resolução, este método dimensiona uniformemente cada dimensão com um conjunto fixo de coeficientes de escala, levando em consideração os recursos disponíveis.

Com este novo método de escalonamento e o avanço do AutoML (aprendizado de máquina automatizado), foi possível superar com eficiência de até 10 vezes (menor e mais rápido) a precisão dos modelos do estado da arte na época em que foi lançado.

Análise

De início, fizemos o reconhecimento dos objetos utilizando as seguintes especificações, que chamaremos de Modo 01:

- Threads: 1
- Modelo: Quantized_EfficientNet

Seguem as análises feitas pelo aplicativo:

Figura 01: Reconhecimento do Objeto 01 no modo 01

Figura 02: Reconhecimento do Objeto 02 no modo 01

Figura 03: Reconhecimento do Objeto 03 no modo 01

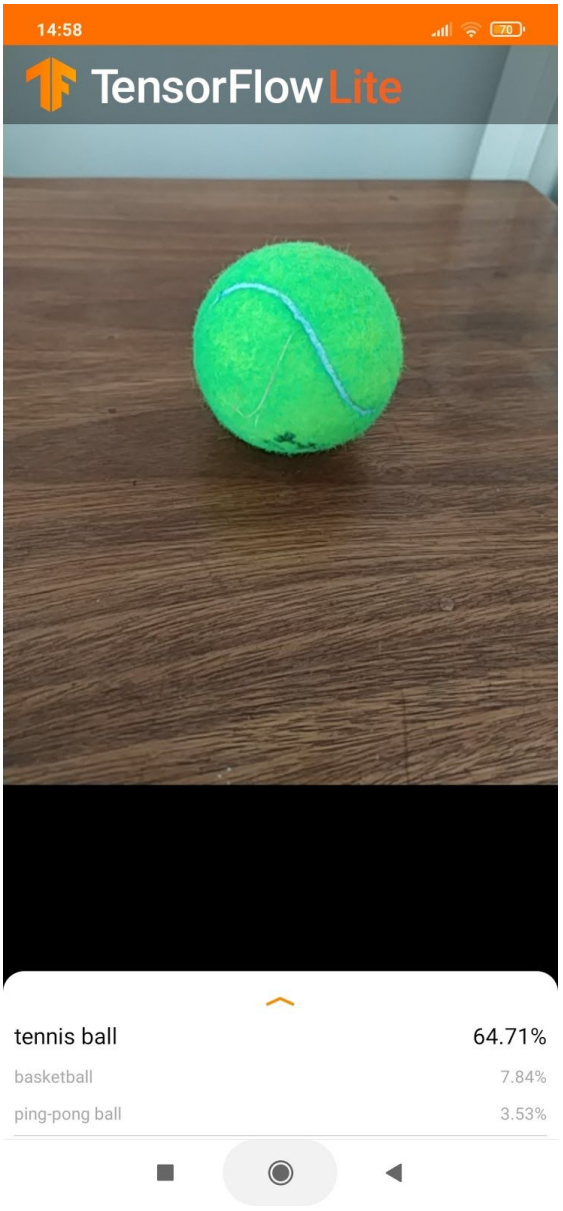

Figura 04: Reconhecimento do Objeto 04 no modo 01

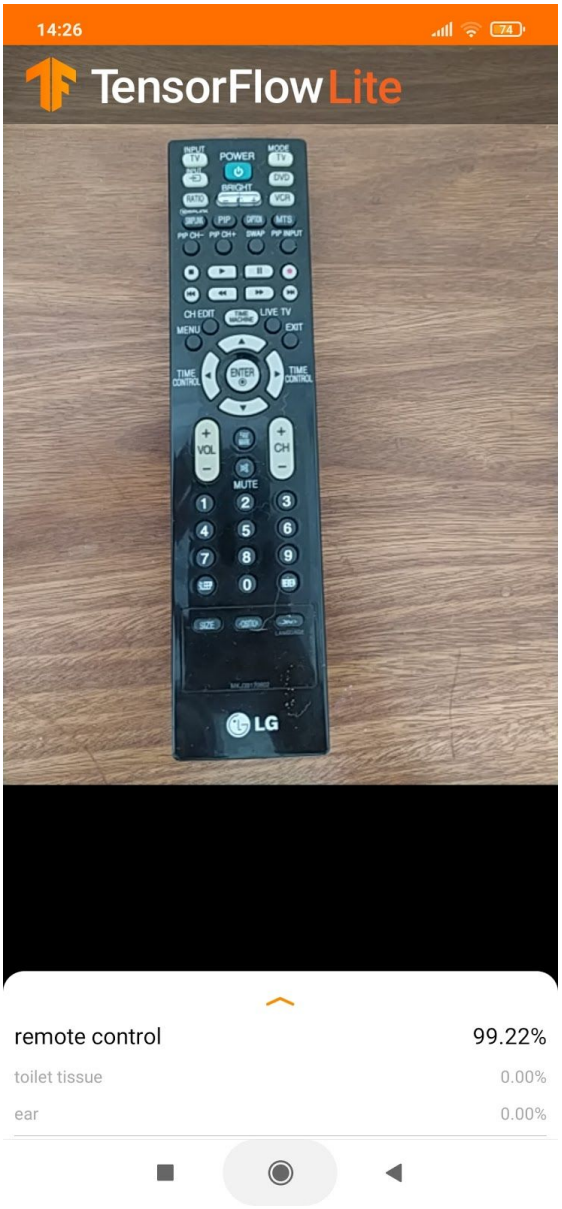

Figura 05: Reconhecimento do Objeto 05 no modo 01

Figura 06: Reconhecimento do Objeto 06 no modo 01

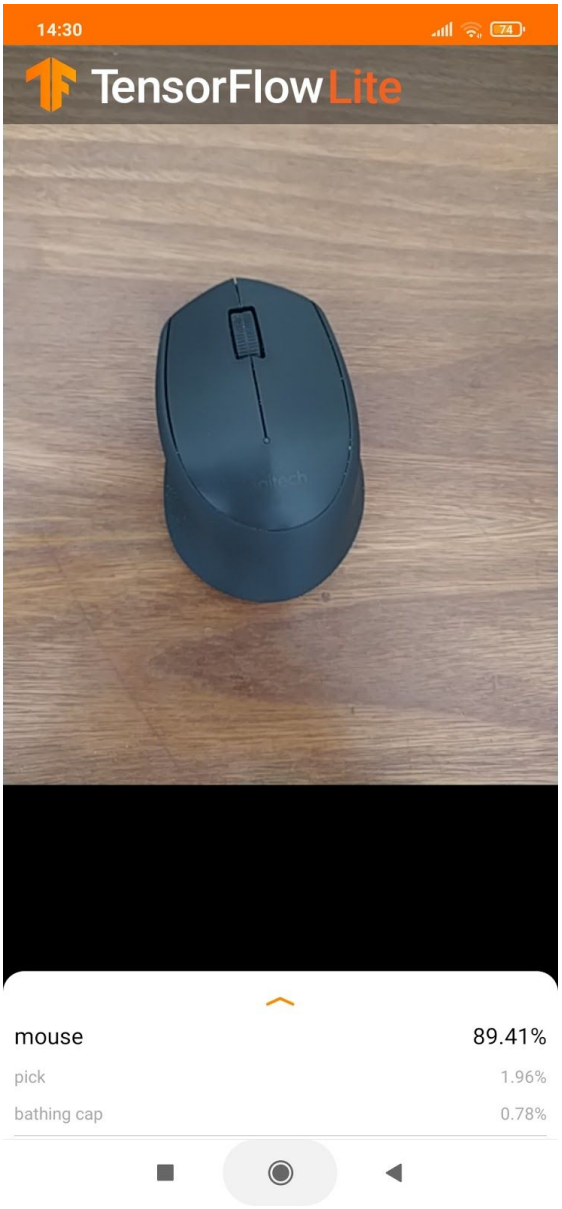

Figura 07: Reconhecimento do Objeto 07 no modo 01

Figura 08: Reconhecimento do Objeto 08 no modo 01

Figura 09: Reconhecimento do Objeto 09 no modo 01

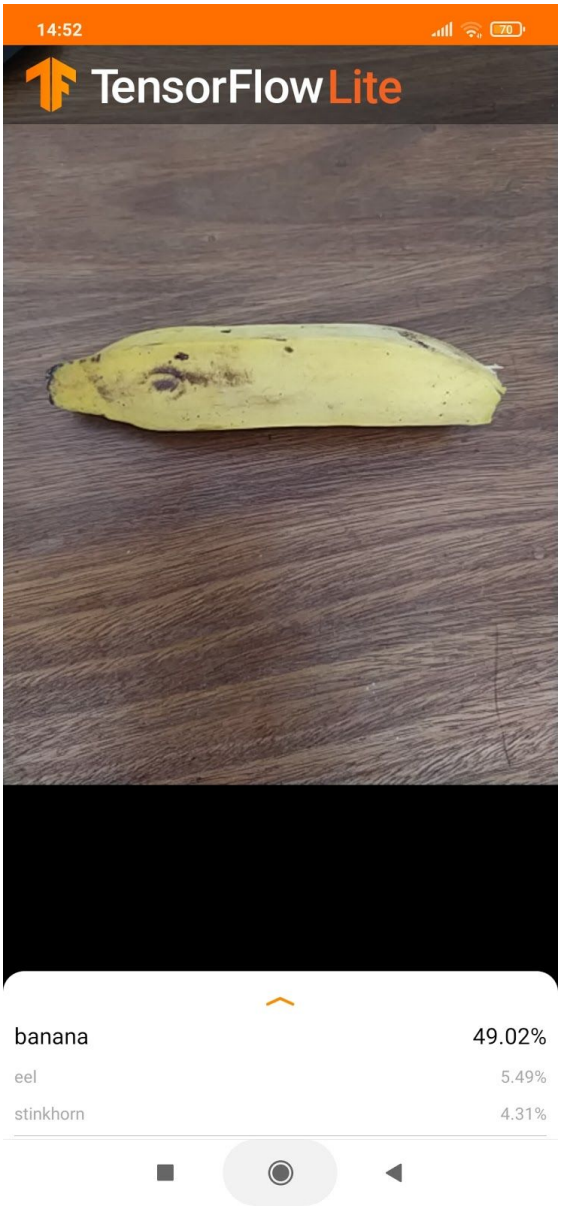

Figura 10: Reconhecimento do Objeto 10 no modo 01

Foi realizada a análise do sistema com outros modos, variando o número de threads, o modelo e o dispositivo utilizado. Por exemplo, o modo 02 tem como parâmetros:

- Threads: 8
- Modelo: Quantized_EfficientNet
- Dispositivo: CPU

Para que o relatório não fique muito extenso, todas as fotos e análises feitas pelo sistema podem ser constatadas na seguinte pasta do Google Drive. Com isso, aqui mostraremos apenas os resultados obtidos em forma de tabelas.

Reconhecimento

A seguir são mostradas tabelas que mostram o desempenho na tarefa de classificar os objetos.

	Modo							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Threads	1	8	1	8	1	8	1	8
Modelo	Quantized Efficient Net	Quantized Efficient Net	Float Efficient Net	Float Efficient Net	Quantized Mobile Net	Quantized Mobile Net	Float Mobile Net	Float Mobile Net

Tabela 01: Definição dos modos de análise.

	Modo _ (%)							
Objeto	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
01	47	65	60	64	28	29	57	40
02	88	94	96	93	100	94	96	95
03	58	71	86	82	65	57	25	36
04	64	95	94	95	99	91	92	99
05	99	99	97	94	97	99	97	92
06	22	85	95	97	97	98	96	98
07	89	94	92	86	87	87	81	87
08	22	64	63	72	75	90	95	89
09	76	89	90	97	98	100	88	99
10	49	64	68	73	85	76	91	99

Tabela 02: Maior valor de ativação dos neurônios de cada um dos modos testados

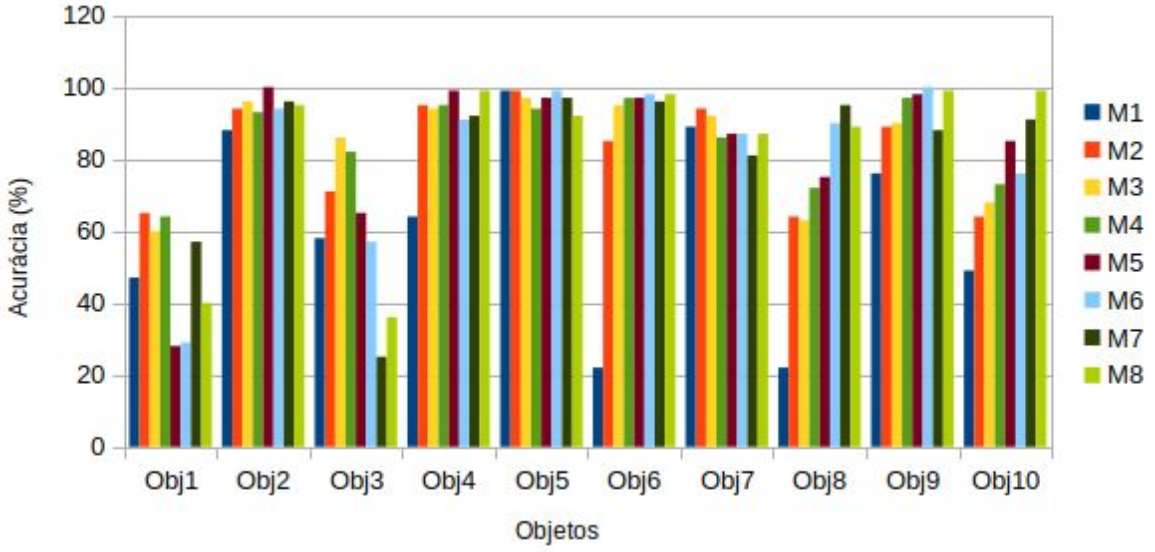

Gráfico 01: Maior valor de ativação dos neurônios de cada um dos modos testados.

Performance

Podemos também comparar os modos e modelos a partir de suas performances. Para isso, fizemos as análises dos Objeto 7 (Mouse) e 4 (Bola de tênis), e comparamos os diferentes tempos de inferência para cada modo e cada dispositivo utilizado. Com isso, chegamos nos seguintes dados (verifique as fotos aqui):

	Modo													
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Thread	1	8	1	8	1	8	1	8	N/A	N/A	N/A	N/A	N/A	N/A
Model	Quant Effic. Net	Quant Effic. Net	Float Effic. Net	Float Effic. Net	Quant Mobil Net	Quant Mobil Net	Float Mobil Net	Float Mobil Net	Float Effic. Net	Float Mobil Net	Quant Effic. Net	Quant Effic. Net	Float Effic. Net	Float Effic. Net
Disp.	CPU	CPU	CPU	CPU	CPU	CPU	CPU	CPU	GPU	GPU	NN API	NN API	NN API	NN API

Tabela 03: Definição dos modos de análise de performance.

Objeto	**M1**	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**	**M10**	**M11**	**M12**	**M13**	**M14**
04	84	71	118	81	95	60	157	95	55	66	729	95	23	98
07	100	85	135	92	102	70	157	85	62	56	727	111	21	92

Tabela 04: Performance dos modos testados (ms).

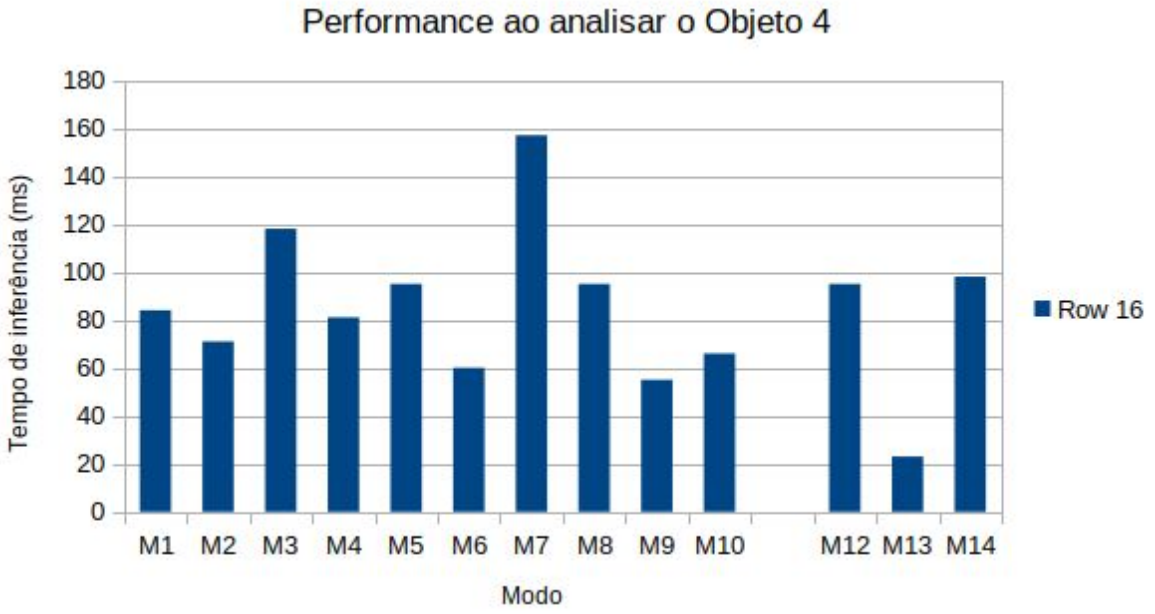

Gráfico 02: Tempo de inferência dos modos testados ao analisar o Objeto 04 (Bola de tênis). O Modo 11 foi suprimido para uma melhor comparação dos demais modos.

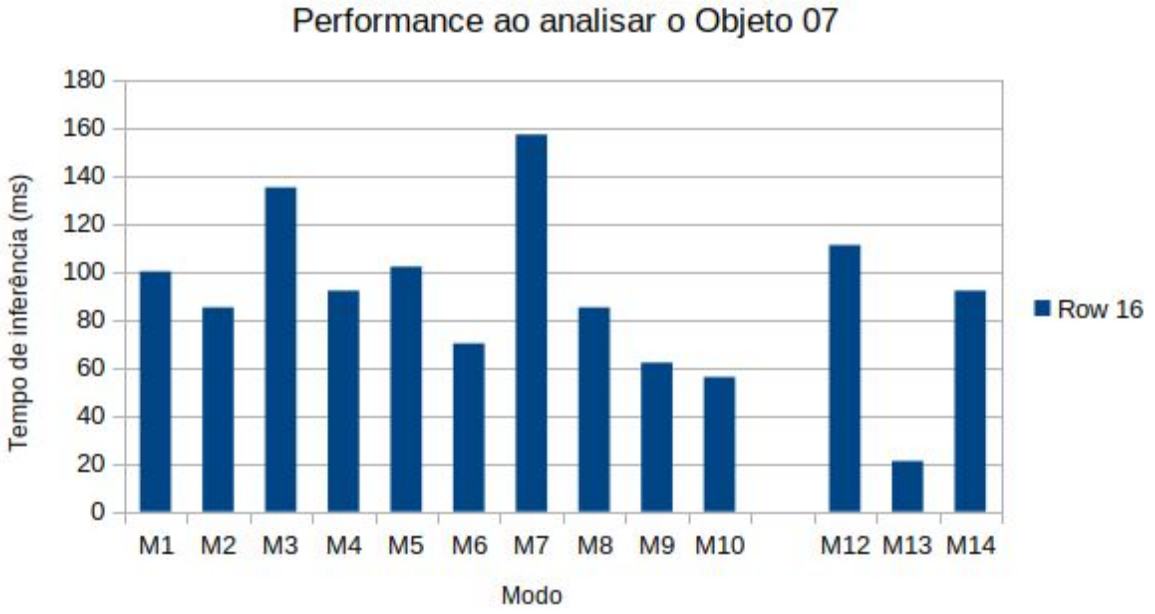

Gráfico 03: Tempo de inferência dos modos testados ao analisar o Objeto 07 (Mouse). O Modo 11 foi suprimido para uma melhor comparação dos demais modos.

Dificuldades Encontradas

Durante o desenvolvimento do trabalho não foram encontradas grandes dificuldades. Como foi decidido utilizar o TensorFlow Lite, foi preciso realizar a instalação do aplicativo TFL Classify em um dispositivo Android. Para isso, foi necessário a instalação e correta configuração do Android Studio e do SDK correto, o que demorou algumas horas.

Outra dificuldade foi a extensa fotografia de diversos objetos para a análise do valor de ativação dos neurônios de cada um dos do software, o que demorou mais algumas horas. Durante essa etapa, foi constatado que o software é muito sensível a qualquer alteração na imagem, principalmente no que se refere a iluminação sobre os objetos. Podemos constatar isso na análise do valor de ativação dos neurônios de cada um dos modelos sobre os Objetos 1 e 3, que acabou variando muito, fazendo com que tais dados não pudessem ser muito confiáveis, já que o *setup* feito para análise foi alterado.

Conclusão

A partir das análises feitas, podemos ver que o melhor modelo para o reconhecimento de objetos é o Modelo 13 (NNAPI com o modelo Float EfficientNet), já que apresenta um menor tempo de inferência (cerca de 20 ms). Vemos também que o Software e modelos utilizados foram capazes de distinguir e identificar os objetos selecionados, como esperado.

Pudemos verificar também que, quando foi utilizada a CPU, o aumento no número de Threads fez com que o tempo de latência diminuísse e o valor de ativação dos neurônios de cada um das redes aumentasse. Além disso, a utilização da GPU se mostrou mais eficiente do que a CPU, mesmo utilizando múltiplas Threads.

Referências

EfficientNet: Improving Accuracy and Efficiency through AutoML and Model Scaling. **Google AI Blog**, 2020. Disponível em: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. Acesso em: 18/12/2020.

Guia do TensorFlow. **TensorFlow**, 2020. Disponível em: https://www.tensorflow.org/overview. Acesso em: 15/12/2020.

Guia do TensorFlow Lite. **TensorFlow**, 2020. Disponível em: https://www.tensorflow.org/lite/guide. Acesso em: 15/12/2020.

MobileNets: Open-Source Models for Efficient On-Device Vision. **Google AI Blog**, 2020. Disponível em: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html. Acesso em: 18/12/2020.

Imagens e vídeos: https://drive.google.com/drive/folders/1EzD4QeywGQLW8QCRUMufG-QvGL8DoniX?usp=sharing