UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

Lucas Pedro Bordignon

DESENVOLVENDO UM BENCHMARK PARA DEEP LEARNING SOBRE A PLATAFORMA JETSON TX2

Florianópolis

2019

Lucas Pedro Bordignon

DESENVOLVENDO UM BENCHMARK PARA DEEP LEARNING SOBRE A PLATAFORMA JETSON TX2

Trabalho de Conclusão de Curso submetido ao Programa de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel em Ciência da Computação.
Orientador: Prof. Aldo Von Wangenheim, Dr. rer. nat.
Coorientador: Lucas de Almeida Fernandes, Bs.

Florianópolis

2019

Lucas Pedro Bordignon

DESENVOLVENDO UM BENCHMARK PARA DEEP LEARNING SOBRE A PLATAFORMA JETSON TX2

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciência da Computação", e aprovado em sua forma final pelo Programa de Graduação em Ciência da Computação.

Florianópolis, 18 de novembro 2019.

Prof. José Francisco Danilo de Guadalupe Correa Fletes
Coordenador de Curso

Banca Examinadora:

Prof. Aldo Von Wangenheim, Dr. rer. nat.
Orientador

Lucas de Almeida Fernandes, Bs.
Coorientador

Prof. Mauro Roisenberg, Dr.
Membro da Banca

Prof. Antonio Carlos Sobieranski, Dr.
Membro da Banca

AGRADECIMENTOS

Agradeço imensamente a minha família, por todo o apoio incondicional e sobre todos os conselhos fornecidos durante toda essa jornada percorrida. Em especial a minha mãe, Guiomar, pelo incentivo a busca de conhecimento a todo custo e a meu pai, Volnei, por todos os ensinamentos que compõe minha personalidade e meu caráter como ser humano. Além disso, agradeço a Gabriela por todos os momentos de motivação e positividade que me permitiram chegar ao final dessa jornada.

Agradeço ainda a Universidade Federal de Santa Catarina pela oportunidade do contato com diversas áreas do conhecimento e aos professores do departamento de Informática e Estatística.

Ao professor Aldo, por todo o apoio durante as fases finais do curso, bem como aos amigos Thiago e Lucas pelos conselhos durante o presente trabalho.

Agradeço o apoio da *NVIDIA Corporation* com a doação da plataforma Jetson TX2 utilizada para o projeto de pesquisa em questão.

A todas as boas amizades construídas aqui dentro que certamente perdurarão por anos, em especial aos amigos Matheus, Vinicius, Gabriel e Patrick que me acompanharam durante todo o período na a graduação e ao amigo Victor, que me acompanha e auxilia diariamente na jornada profissional.

Ao curso pré-vestibular Einstein Floripa, todos os alunos, organizadores e professores, por todos os aprendizados que me tornaram uma pessoa íntegra perante a sociedade.

E agradeço a todos os amigos do futebol de toda quinta-feira que, embora estando distante nos últimos momentos da graduação, permitiram finalizar esta etapa da vida com maior leveza.

A vida é muito curta para ser pequena

Benjamin Disraeli

RESUMO

Com os recentes avanços nas áreas de navegação visual e inteligência artificial (*deep learning*), diversas áreas sofrem mudanças quanto ao modo de atacar e solucionar os problemas nelas existentes, como é o caso da área de navegação visual e veículos autônomos. Porém, não só técnicas teóricas tem avançado como também sistemas embarcados com foco em acelerar a prototipação de novas soluções vem acompanhando as novas mudanças. O presente trabalho tem como foco desenvolver um *benchmark* sobre sistemas embarcados especializados em algoritmos de inteligência artificial, realizando um estudo comparativo de modelos alinhados ao estado da arte, sobre a plataforma Jetson TX2, auxiliando a tomada de decisão quanto as ferramentas mais apropriadas para implementação de aplicações reais sobre a área de *deep learning*, como é o caso de carros autônomos.

Palavras-chave: Inteligência artificial, Visão computacional, *Deep learning*, CNN, Carros autônomos

ABSTRACT

With the recent advances in the areas of visual navigation and deep learning, many areas are undergoing changes in the way they tackle and solve problems within them, such as visual navigation and autonomous vehicles. However, not only theoretical techniques have advanced, but also embedded systems with a focus on accelerating prototyping of new solutions are keeping pace with new changes. The present work focuses on developing a benchmark on embedded systems specialized in artificial intelligence algorithms, conducting a comparative study of state-of-the-art models on the Jetson TX2 platform, aiding decision making on the most common tools. suitable for implementing real applications in the field of deep learning, such as autonomous cars.

Keywords: Artificial inteligence, Computer vision, *Deep learning*, CNN, Autonomous cars

LISTA DE FIGURAS

Figura 1 Representação do conceito do Perceptron, apresentado por Rosenblatt 28
Figura 2 Definição de Redes Neurais Artificiais, Fonte: (FAUSETT; FAUSETT, 1994) 29
Figura 3 Ilustração de um neurônio em uma rede neural artificial (FEI-FEI; KARPATHY; JOHNSON,) 30
Figura 4 Mapeamento das diversas funções de ativação na literatura 31
Figura 5 Esquemático de funções de custo (JANOCHA; CZARNECKI, 2017) 32
Figura 6 Relação entre taxa de aprendizado e convergência (CUI, 2018) 34
Figura 7 Representação da arquitetura Neorecognition 35
Figura 8 Funcionamento da etapa de convolução de uma CNN .. 36
Figura 9 Esquemático de uma Rede Neural Convolucional 37
Figura 10 Arquitetura SAF-RCNN 43
Figura 11 Abordagem do modelo YOLOv1 44
Figura 12 Modelo proposto como YOLOv3 45
Figura 13 Arquitetura do modelo *RetinaNet* 47
Figura 14 Modelo de detecção *Single Shot Detector* 48
Figura 15 Convoluções sobre a Arquitetura MobileNet V1 49
Figura 16 Redes residuais inversas sobre a Arquitetura MobileNet V2 50
Figura 17 Convoluções em largura sobre a Arquitetura Inception V2 51
Figura 18 Modelo de quatro fases 53
Figura 19 Arquitetura de sub-redes que compõe o modelo *FlowNet 2.0* 54
Figura 20 Modelo *FlowNetS* 54
Figura 21 Modelo *FlowNetC* 54
Figura 22 Arquitetura *Faster R-CNN* 55
Figura 23 Exemplos de inferência do Modelo *Mask R-CNN* 56
Figura 24 Resultados dos experimentos sobre a Jetson Nano 59

Figura 25 Resultados sobre a plataforma Jetson TX2 em ambiente de produção 60
Figura 26 Resultados comparativos entre TensorRT V1 e V2 61
Figura 27 Especificações Técnicas para as variantes do modelo Jetson TX2 64
Figura 28 NVIDIA Jetson TX2 65
Figura 29 Imagens de exemplo presentes na base de dados Microsft COCO 69
Figura 30 Imagens de exemplo presentes na base de dados KITTI Stereo Vision 70
Figura 31 Resultados de detecção - SSD MobileNet V1 sobre Microsoft COCO 71
Figura 32 Resultados de detecção - SSD MobileNet V2 sobre Microsoft COCO 72
Figura 33 Resultados de detecção - SSD Inception V2 sobre KITTI Stereo Vision 73
Figura 34 Resultados de detecção - Faster R-CNN sobre KITTI Stereo Vision 74
Figura 35 Resultados de detecção - Mask R-CNN sobre KITTI Stereo Vision 75
Figura 36 Resultados do *benchmark* - Frames por Segundo - Microsft COCO 76
Figura 37 Resultados do *benchmark* - Tempo médio de inferência - Microsft COCO 76
Figura 38 Resultados do *benchmark* - Média e desvio padrão de temperatura da GPU - Microsft COCO 77
Figura 39 Resultados do *benchmark* - Desvio padrão de temperatura da GPU - Microsft COCO 77
Figura 40 Resultados do *benchmark* - Média e desvio padrão de memória da GPU - Microsft COCO 78
Figura 41 Resultados do *benchmark* - Desvio padrão de memória da GPU - Microsft COCO 78
Figura 42 Resultados do *benchmark* - Média e desvio padrão de memória da GPU - Microsft COCO 79
Figura 43 Resultados do *benchmark* - Desvio padrão de memória da GPU - Microsft COCO 79
Figura 44 Resultados do *benchmark* - Frames por Segundo - KITTI

Stereo Vision 80
Figura 45 Resultados do *benchmark* - Tempo médio de inferência - KITTI Stereo Vision 80
Figura 46 Resultados do *benchmark* - Média e desvio padrão de temperatura da GPU - KITTI Stereo Vision 81
Figura 47 Resultados do *benchmark* - Desvio padrão de temperatura da GPU - KITTI Stereo Vision 81
Figura 48 Resultados do *benchmark* - Média e desvio padrão de memória da GPU - KITTI Stereo Vision 82
Figura 49 Resultados do *benchmark* - Desvio padrão de memória da GPU - KITTI Stereo Vision 82
Figura 50 Resultados do *benchmark* - Média e desvio padrão de memória da GPU - KITTI Stereo Vision 83
Figura 51 Resultados do *benchmark* - Desvio padrão de memória da GPU - KITTI Stereo Vision 83
Figura 52 Comparativo de resultados apresentados por autores das arquiteturas 84

LISTA DE TABELAS

Tabela 1 Tabela comparativa dos trabalhos apresentados. 58
Tabela 2 Tabela comparativa de componentes existentes 64
Tabela 3 Elementos mensurados durante o *benchmark* 65
Tabela 4 Modelos de *deep learning* utilizados para experimentação 66
Tabela 5 Modos de operação sobre a plataforma Jetson TX2 71
Tabela 6 Sistemas utilizados pelos autores para metrificação 85
Tabela 7 Comparativo dos resultados obtidos 86

LISTA DE ABREVIATURAS E SIGLAS

YOLO	You Only Look Once
COCO	Common Objects in Context
HOG	Histograma de Gradientes Orientados
ANN	Redes Neurais Artificiais
CNN	Redes Neurais Convolucionais
IA	Inteligência Artificial
RNA	Redes Neurais Artificiais

SUMÁRIO

1 INTRODUÇÃO 23
1.1 OBJETIVOS GERAIS 24
1.2 OBJETIVOS ESPECÍFICOS 24
1.3 ESTRUTURA 25
2 FUNDAMENTAÇÃO TEÓRICA 27
2.1 REDES NEURAIS ARTIFICIAIS 27
2.2 NEURÔNIOS E FUNÇÕES DE ATIVAÇÃO 29
2.3 FUNÇÃO DE CUSTO 31
2.4 OTIMIZAÇÃO 32
2.5 DEEP LEARNING 34
2.6 REDES NEURAIS CONVOLUCIONAIS 35
3 REVISÃO DA LITERATURA 39
3.1 QUESTÃO DE PESQUISA 39
3.2 DEFINIÇÃO DA BUSCA 39
3.3 EXECUÇÃO DA BUSCA 40
3.4 ANÁLISE DOS TRABALHOS 41
3.4.1 Ten Years of Pedestrian Detection, What Have We Learned? 41
3.4.2 Scale-Aware Fast R-CNN for Pedestrian Detection 42
3.4.3 You Only Look Once: Unified, Real-Time Object Detection 43
3.4.4 YOLOv3: An Incremental Improvement 44
3.4.5 Focal Loss for Dense Object Detection 46
3.4.6 SSD: Single Shot MultiBox Detector 47
3.4.7 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications 48
3.4.8 MobileNetV2: Inverted Residuals and Linear Bottlenecks 49
3.4.9 Rethinking the Inception Architecture for Computer Vision 50
3.4.10 Dense Scene Flow Based Coarse-to-Fine Rigid Moving Object Detection for Autonomous Vehicle 51
3.4.11 FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks 53
3.4.12 Faster R-CNN 55
3.4.13 Mask R-CNN 56
3.5 COMPARATIVO DOS TRABALHOS ANALISADOS 57

3.6 ACELERADORES EMBARCADOS 58
4 DESENVOLVIMENTO 63
4.1 AMBIENTE DE EXPERIMENTAÇÃO 63
4.2 MÉTRICAS 65
4.3 MODELOS 66
4.4 IMPLEMENTAÇÃO 67
4.5 CONJUNTO DE DADOS 68
4.5.1 Microsoft COCO 68
4.5.2 KITTI Stereo Vision 69
4.6 RESULTADOS 70
5 CONCLUSÕES E TRABALHOS FUTUROS 87
REFERÊNCIAS 89
ANEXO A – Donation Letter 97
APÊNDICE A – Relatório Técnico 101
APÊNDICE B – Artigo 175
APÊNDICE C – Código Fonte 187

1 INTRODUÇÃO

A industria automobilística, direta ou indiretamente, faz parte da rotina de diversos cidadãos do mundo. Atualmente, trilhões de veículos são conduzidos durante o ano ao redor do planeta, passando por diversas situações de alta complexidade e de risco aos condutores e passageiros (Daily et al., 2017).

Para contornar os possíveis riscos presentes na condução de automóveis em rodovias urbanas e rurais, diversos esforços vem sendo dedicados a projetos de pesquisa que utilizem visão computacional e inteligência artificial para auxiliar a reduzir o risco existente, bem como trazer ainda mais comodidade aos passageiros ou condutores dos mesmos.

Aplicações da área de ciência da computação sobre tal cenário vem sendo exploradas desde cerca dos anos 80, no século passado. Entretanto, com ainda mais intensidade, o uso de inteligência artificial e modelos de reconhecimento baseados em redes neurais convolucionais vem sendo explorados desde o início do presente século (BENENSON et al., 2014).

Focando, como exemplo, na área de veículos autônomos, modelos que utilizam navegação visual passiva vem sendo preferidos se comparado com modelos de navegação ativa utilizando sensores como LIDAR. Tal configuração permite conjuntos de hardware embarcado mais baratos e menores fisicamente, além de utilizarem menor quantidade de energia para operação e serem considerados menos invasivos se comparados com técnicas de navegação ativa (BOKOVOY; MURAVYEV; YAKOVLEV, 2019), (STANDARD, 2005), (GROUP SAFETY PUBLICATION. INTERNATIONAL STANDARD, 2001).

Para trabalhar com tais aplicações, diversos aceleradores em *hardware* vem sendo desenvolvidos pela indústria para que as necessidades de sistemas de tempo real e privacidade sejam atendidos. Porém, quando trabalhando com tais aceleradores, os mesmos devem possuir características específicas para que atendam a limitações de espaço, energia, largura de banda e outros (Sze, 2017).

Dados os avanços sobre a área, mais especificamente sobre o campo de estudo de *deep learning* e redes neurais convolucionais, novos desafios surgem. Um dos principais campos o qual acompanha a área em questão relaciona-se a aceleradores de hardware e sistemas embarcados.

Com as recentes demandas de modelos e aplicações quanto a de-

sempenho para processamento de imagens sobre ambientes limitados, diversos fabricantes tem voltado seus olhos para a área e novas arquiteturas e sistemas surgiram para solucionar e atender a esse novo nicho (Li; Xiao; Liang, 2017). É possível encontrar diversos modelos de placas embarcadas com foco em modelos de inteligência artificial (Zhang et al., 2018), (Han; Oruklu, 2017).

Porém, com o crescente número de famílias de aceleradores surgindo, diversos deles possuem focos específicos e otimizações para determinadas tarefas, bem como limitações relacionadas a latência, privacidade, largura de banda, consumo energético e outros. O presente trabalho tem como um de seus focos principais entender como limitações de hardware interferem e comportam-se sobre tais ambientes, auxiliando no processo de tomada de decisão referente as melhores soluções para modelos de *deep learning* embarcados.

1.1 OBJETIVOS GERAIS

O objetivo principal do trabalho consiste em apresentar um estudo comparativo com foco em métricas de desempenho entre modelos de reconhecimento de imagens baseados em redes neurais convolucionais, alinhados ao estado da arte (*SSD*, *Mask R-CNN* e *Faster R-CNN*, bem como suas variações de implementação), através da implementação de um *benchmark* sobre a plataforma embarcada Jetson TX2, da fabricante NVIDIA Corporation.

1.2 OBJETIVOS ESPECÍFICOS

- Explorar os avanços mais recentes na área de *deep learning*, navegação visual e reconhecimento de objetos;
- Realizar um levantamento do estado da arte na área de *deep learning* e aceleradores embarcados;
- Implementar um *benchmark* de desempenho de modelos de reconhecimento baseados em redes neurais convolucionais;
- Realizar a experimentação e coleta de dados sobre o desempenho da plataforma Jetson TX2 em relação a modelos de reconhecimento de objetos, por meio de cenários do mundo real;
- Apresentar um estudo comparativo sobre modelos de reconheci-

mento no estado da arte na área de *deep learning*;

1.3 ESTRUTURA

O Capítulo 2 apresenta a historia relacionada aos avanços na área de inteligência artificial desde os primórdios da mesma, alcançando as técnicas mais sofisticadas utilizadas nos dias de hoje, como é o caso de redes neurais.

Em continuidade, o Capítulo 3 revisa trabalhos considerados o estado da arte na área, trazendo seus detalhes teóricos apresentados pelos autores dos modelos. Além disso, um sub-conjunto dos trabalhos apresentados é utilizado como base para decisão dos modelos a serem utilizados durante a fase de experimentação.

No Capítulo 4 as métricas e detalhes de implementação do *benchmark* sobre a plataforma Jetson TX2 são expostos, bem como argumentos para a decisão de uso do sistema em questão. Além disso, os conjuntos de dados utilizados são apresentados.

Sob o mesmo capítulo, o resultado dos experimentos executados e coletados são expostos, sendo mais profundamente discutidos no Capítulo 5, além de uma discussão sobre possíveis soluções e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Estudos relacionados a área de inteligência artificial existem desde meados do século XX, entretanto, apenas com a capacidade de processamento computacional hoje existentes é possível realizar a implementação de tais modelos de modo eficiente. Consequentemente, se faz possível a execução de tais modelos sobre diversas áreas de aplicação, como exemplo a geração de dados para simulações de modo geral (CHOI et al., 2017), geração de músicas de modo artificial e previsão de movimentos de ativos no mercado financeiro (YAO; NORTH; TAN, 2002).

Para fundamentar o presente trabalho se faz necessária a compreensão de determinados conceitos que compõem a área, sendo apresentados e detalhados a seguir.

2.1 REDES NEURAIS ARTIFICIAIS

Um dos precursores da técnica de redes neurais artificiais (ANNs) utilizada nos dias de hoje foi Rosenblatt, com base em seu trabalho apresentado no ano de 1958, o perceptron (ROSENBLATT, 1958).

A ideia principal do perceptron consiste em uma representação do sistema neurológico biológico que existe em uma grande parte de animais, como seres humanos. Após receberem impulsos de entrada, os neurônios desse sistema em formato de rede, são disparados e ativados caso sejam responsáveis pelo reconhecimento do devido dado de entrada, seja ele visual ou tátil. Assim, o modelo biológico permite que animais possuam tal sistema neurológico bem desenvolvido consigam reconhecer físicos e visuais.

A Figura 1 mostra uma abstração do modelo proposto. Sua arquitetura segue os seguintes passos:

- Recebe uma entrada binaria, como um estímulo externo ou decorrente de neurônios anteriores na rede;
- Cada valor de entrada é multiplicado por um valor, chamado de peso, que indica a intensidade de tal neurônio sendo ativado com relação a entrada passada;
- O modelo limita os resultados para ou 0 ou 1, permitindo que o valor de saída seja então binário. O valor representa, como na biologia, se o neurônio em questão foi ativado pelo estímulo ou não

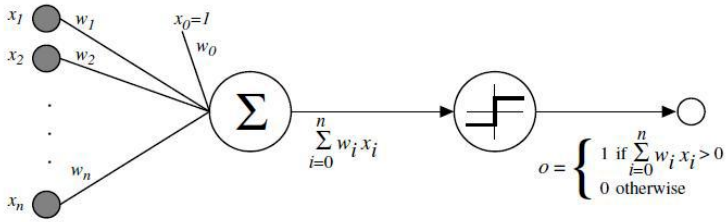


Figura 1 – Representação do conceito do Perceptron, apresentado por Rosenblatt

Para cada neurônio da rede e cada valor de entrada, seja ele a entrada inicial da rede ou a saída de uma camada intermediária, existe um peso representando a intensidade do nodo dado o conjunto de entrada.

Após a descoberta e aparição do modelo *perceptron*, Bernard Widrow e Marcian Hoff desenvolveram em 1960 duas redes neurais artificiais, baseados sobre o mesmo conceito. O modelo foi nomeado de ADALINE e é uma rede com camada única e múltiplos pontos de entrada, retornando um ponto de saída único (WIDROW, 1960).

Uma camada da rede ADALINE pode ser definida, matematicamente, do seguinte modo:

$$output = \sum_{i=0}^{n} w_i x_i + \theta$$

- n é o número de entradas da camada;
- x é o valor atual de entrada, dado o índice i;
- w é o valor do peso do neurônio, dado o índice i;
- θ é um valor constante;

Em 1964, redes treinadas por Valentin Grigorevich Lapa e A. G. Ivakhnenko podem ser consideradas as primeiras com o título de aprendizado profundo em multi-camadas (IVAKHNENKO; LAPA, 1965).

Unidades das redes apresentadas utilizam funções de ativação polinomial, implementando polinômios de Kolmogorov-Gabor, sendo ainda mais genéricas que funções de ativação utilizadas em modelos recentes. Dado um conjunto de dados, as camadas do modelo apresentado são treinadas por análise de regressão.

Cada elemento de uma rede neural artificial é chamado de neurônio artificial, cuja responsabilidade é representar um neurônio biológico real, como mostrado na Figura 2. É possível observar a presença de unidades de entrada de estímulos (*Input Units*), entradas de transmissão (*Hidden Units*) e de saída de estímulos (*Output Units*).

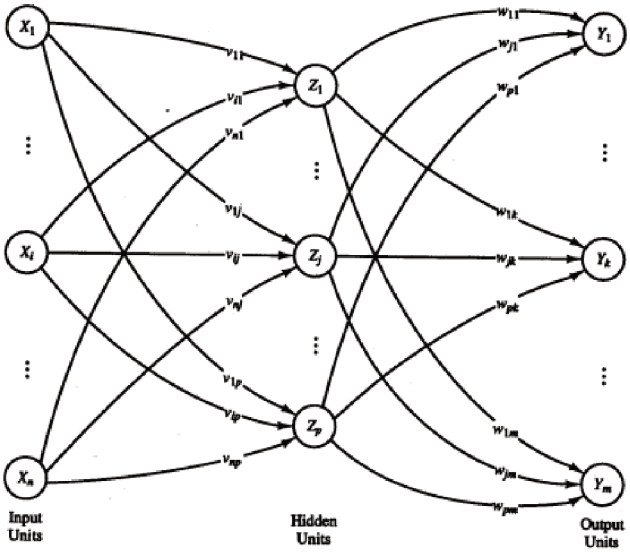


Figura 2 – Definição de Redes Neurais Artificiais, Fonte: (FAUSETT; FAUSETT, 1994)

2.2 NEURÔNIOS E FUNÇÕES DE ATIVAÇÃO

Como apresentado anteriormente, um modelo baseado em redes neurais artificiais é composto por um conjunto de entidades, conhecidas como neurônios. As entidades em questão tem como base a aproximação matemática, se comparados a neurônios biológicos.

Neurônios reais recebem como estímulo inicial um impulso elétrico (dado de entrada) e, com base na "responsabilidade"a eles atribuída, são ativados eletricamente, repassando o estímulo recebido ao próximo na cadeia de neurônios. Para realizar a representação do modelo descrito em uma rede neural artificial, utiliza-se de um conjunto de operações matemáticas, como funções não lineares, chamadas de

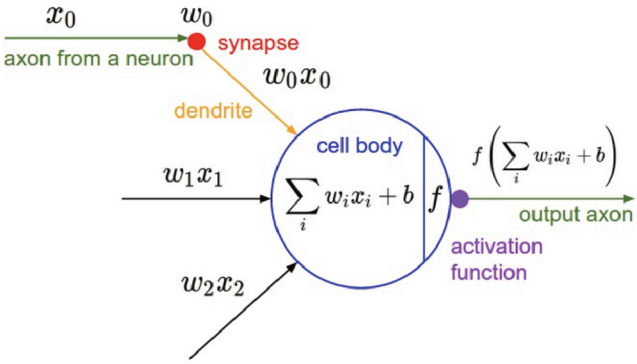


Figura 3 – Ilustração de um neurônio em uma rede neural artificial (FEI-FEI; KARPATHY; JOHNSON,)

funções de ativação.

Matematicamente, é possível separar as variáveis presentes em neurônios artificiais do seguinte modo:

- x_i é uma amostra dos dados de entrada;
- w_i é um peso treinado para aquele índice dos dados de entrada;
- b é o peso de influência para o neurônio;
- f é a função de ativação;

Com as variáveis é possível aplicar uma função de primeiro grau e realizar o somatório de todos os resultados para obter o valor resultante de um neurônio. Vale notar que os pesos para cada valor de entrada podem ser treinados para melhor representação de estímulo (apresentado na seção 2.4). quando inicia-se um modelo sem pesos pré-treinados tende-se a utilizar pesos aleatórios (GLOROT; BENGIO, 2010).

Baseado no resultado do somatório de cada execução, é preciso aplicar uma função não-linear sobre o resultado, responsável por definir os limites superior e/ou inferior para ca da valor de entrada e sinalizar a ativação ou não do neurônio e qual a sua intensidade. As funções não-lineares são chamadas de função de ativação. A Figura 3 representa o esquemático de um neurônio artificial.

Sobre aplicações mais recentes, existem um conjunto de funções de ativação existentes na literatura, sendo apresentados parcialmente abaixo.

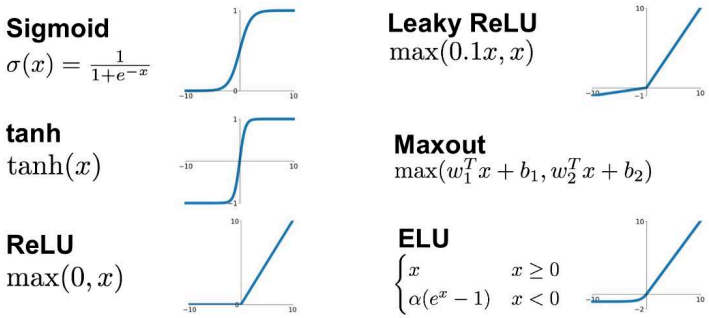


Figura 4 – Mapeamento das diversas funções de ativação na literatura

2.3 FUNÇÃO DE CUSTO

Como apresentado na seção anterior, generalizando, neurônios sobre um modelo baseado em ANN possui funções de agregação que comportam-se da seguinte maneira:

$$neuron\_value = (\sum_{i=0}^{n} w_i x_i) + b$$

Aonde w e b são inicialmente configurados como valores aleatórios. Para que possamos treinar os modelos e fazer com que seus pesos convirjam a valores que respondam de maneira satisfatória a problemas de classificação e regressão eles necessitam adaptar-se dinamicamente.

O passo descrito é conhecido como o processo de treinar um modelo, tornando pesos mais precisos e aderentes sobre os dados de entrada fornecidos para o contexto em questão. Durante a fase de treino, cada entidade é considerada como uma função matemática, permitindo que métodos de otimização matemática sejam utilizados para encontrar as melhores aproximações possíveis para os parâmetros do modelo.

Para utilizar um algoritmo de otimização, primeiramente, se faz necessário definir uma função de custo responsável por mensurar quão bem o modelo resulta nos dados esperados. A Figura 5 apresenta algumas das funções mais utilizadas para treinamento de modelos recentes.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y} - \sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y} - \sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$

Figura 5 – Esquemático de funções de custo (JANOCHA; CZARNECKI, 2017)

2.4 OTIMIZAÇÃO

Dado que modelos possuem funções de custo que expressam o quão longe o modelo em questão está da "predição perfeita", já é possível executar algoritmos de otimização sobre tal função, reduzindo ao máximo essa distância.

Existem alguns algoritmos de propósito geral herdados do campo da matemática utilizados para treinamento de redes neurais. De modo geral, o problema ao qual se confronta quando treina-se tais modelos é classificado como um problema de minimização, através de alterações nos parâmetros.

Um dos métodos utilizados para aproximação de mínimos locais para uma função de custo consiste no cálculo de derivadas. Considerando-se funções uni-dimensionais, sua derivada em um determinado ponto expressa o vetor de mudança da função no determinado ponto.

Sabendo isso, é possível calcular a direção de movimento com base no resultado coletado, bem como a intensidade do movimento sobre tal função. A expressão a seguir representa a derivada de uma função uni-dimensional, com respeito a sua entrada:

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

No caso de redes neurais, aonde temos múltiplos pontos de en-

trada, as derivadas são chamadas de derivadas parciais e o vetor de direções resultante do cálculo é chamado de gradiente. Utilizando os gradientes, é possível encontrar a melhor direção na qual o modelo tende a convergir para o resultado ótimo.

Realizadas as operações matemáticas, faz-se necessário atualizar os parâmetros da rede treinada, para que futuras inferências do modelo sejam mais próximas do ideal. Um método comumente utilizado de atualização de pesos é chamado de gradiente descendente.

O conceito por trás do gradiente descendente baseia-se em atualizar os parâmetros a cada iteração sobre dados de entrada. Durante a fase de treinamento define-se um parâmetro, comumente chamado de taxa de aprendizado (*learning rate*), simbolizando a intensidade de atualização a cada passo para cada peso. Entretanto, é preciso ter cuidado na escolha do valor da taxa, dado que é possível que o algoritmo utilize um passo demasiado grande ou pequeno, apresentando problemas de convergência, como mostra a Figura 6.

Um passo do algoritmo de gradiente descendente pode ser descrito da seguinte maneira:

$$gradiente = derivada(l(x), i, w)$$

$$peso_k = peso_k - (learning\_rate \times gradiente)$$

Aonde $l(x)$ é a função de custo definida, i é o vetor de entrada, w é o vetor de pesos e $derivada(x, y, z)$ é uma função matemática responsável pelo cálculo da derivada da função custo no ponto determinado.

Vale notar a subtração do valor do gradiente, considerando a taxa de aprendizado. Tal sinal é utilizado para atualizar os pesos na direção correta, já que sinais negativos e positivos devem ser invertidos. Diversas outras implementações existem na literatura utilizando melhorias sobre o conceito apresentado (NESTEROV, 1983), (SUTSKEVER et al., 2013).

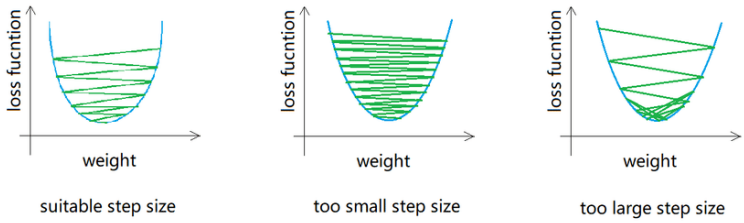


Figura 6 – Relação entre taxa de aprendizado e convergência (CUI, 2018)

2.5 DEEP LEARNING

Conceitualmente, o campo é definido como uma especialização do aprendizado de máquina (*machine learning*) baseado em algoritmos que utilizam-se de grafos profundos multi-camadas, aplicando transformações lineares e não-lineares a dados de entrada (Du et al., 2016).

Após o surgimento de novos modelos estatísticos e técnicas de otimização, como apresentados anteriormente, redes neurais profundas começaram a tornar-se mais populares e resolver uma nova gama de problemas relacionados a reconhecimento de padrões e aprendizado de máquina (SCHMIDHUBER, 2015).

Em 1950, Hubel e Wiesel publicaram um trabalho no qual realizam experimentos sobre o sistema neurológico de mamíferos. A grande parte do trabalho apresentado consiste em induzir estímulos elétricos ao cérebro de tais mamíferos e realizar a coleta da atividade cerebral durante o experimento.

Os estímulos enviados ao sistema neurológico consistem de informações visuais sendo apresentadas aos animais. Anos depois os mesmos autores receberam o prêmio Nobel da ciência sobre a contribuição a área (HUBEL; WIESEL, 1959).

Os autores classificaram as observações, brevemente, em duas categorias:

- **Células Simples**, responsáveis pela detecção de fronteiras nas imagens;
- **Células Complexas**, responsáveis pela detecção de luz e movimento espacial;

Após a descoberta, in 1979, um dos modelos pioneiros a abs-

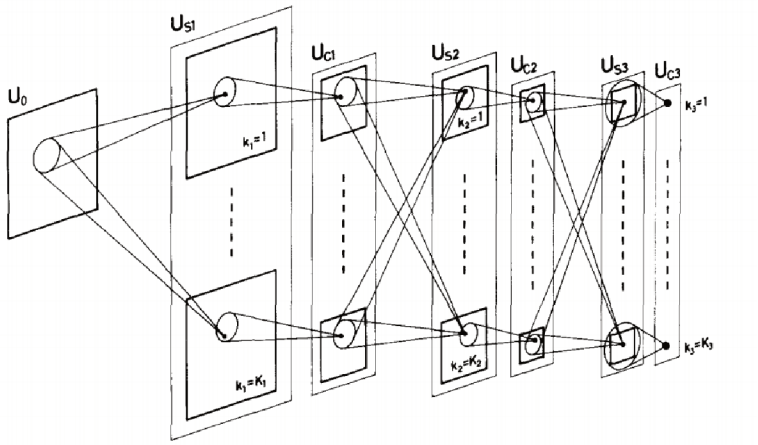


Figura 7 – Representação da arquitetura Neorecognition

trair matematicamente a ideia apresentada por Hubel e Wiesel surge, sendo conhecido como Neocognitron (FUKUSHIMA, 1980). Utilizando os conceitos de células simples e complexas, a rede neural propõe mecanismos de reconhecimento de padrões visuais. A arquitetura segue o conceito de aprendizado sem um tutor, adquirindo a habilidade de reconhecimento baseado em similaridade de formas de objetos.

A primeira camada de cada módulo consiste no que o autor nomeia de "Células-S", representando o conceito de detecção de bordas em células simples (ou células hiper-complexas de baixa ordem), e a segunda camada consiste de "Células-C", abstraindo o conceito de células complexas (ou células hiper-complexas de alta ordem).

A Figura 7 apresenta um resumo do modelo descrito. O Neocognitron pode ser considerado uma das primeiras versões de redes neurais convolucionais já existentes.

2.6 REDES NEURAIS CONVOLUCIONAIS

Redes Neurais Convolucionais são similares a Redes Neurais Artificiais. Elas são constituídas de neurônios com pesos e constantes. Cada neurônio recebe dados de entrada, na forma de imagens e realiza uma multiplicação matricial, seguido opcionalmente por uma função não-linear de ativação. CNNs ainda trabalham sobre funções de custo

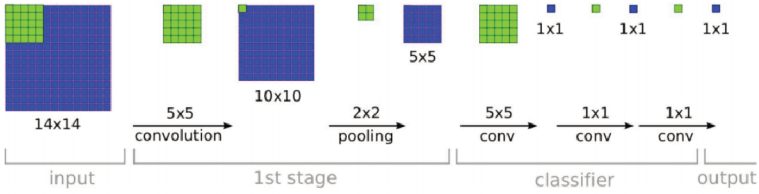


Figura 8 – Funcionamento da etapa de convolução de uma CNN

em sua última camada (FEI-FEI; KARPATHY; JOHNSON,).

Uma das maiores mudanças com relação a redes tradicionais é a utilização de dados de entrada serem multidimensionais, permitindo que propriedades sejam aplicadas diretamente na arquitetura.

ANNs regulares são compostas por camadas totalmente conectadas do começo ao fim, com duas camadas independentes, sendo elas a camada de entrada e de saída. Tal arquitetura desempenha muito bem para problemas de regressão linear e dados bem estruturados sem resíduo contextual (TOSUN; AYDIN; BILGILI, 2016).

A base para modelos de CNNs parte de operações de convolução sobre os dados de entrada. Nessa operação, cria-se o que é chamado de filtro, uma matriz bidimensional na qual os parâmetros a serem treinados estão localizados.

É utilizada a técnica de janelas deslizantes dos filtros, de dimensões inferiores aos dados de entrada, realizando o produto matricial entre a o conjunto de dados e o filtro em questão. Dadas as dimensões, os filtros realizam a multiplicação em partes da imagem de entrada, resultando em mapas intermediários, como mostra a Figura 8.

Com o resultado da multiplicação de cada filtro, aplica-se um somatório e uma função de ativação, como em redes neurais artificiais, resultando na intensidade de ativação do filtro para a parte específica da imagem de entrada. Assim, é possível perceber que os filtros tendem a ser responsáveis pela ativação de partes de objetos ou cenas, aprendidos durante a fase de treinamento (SERMANET et al., 2013).

Dados os filtros presentes no modelo, é possível reduzir a quantidade de parâmetros treináveis se comparado a modelos aonde existe a rede encontra-se totalmente conectada do inicio ao fim, para o tipo de dado em questão.

Entretanto, mesmo se tais redes possuem características específicas para oferecer, como regra geral quando trabalhado com detecção

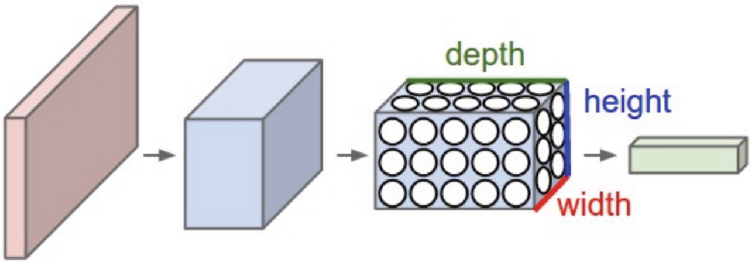


Figura 9 – Esquemático de uma Rede Neural Convolucional

de objetos, elas ainda possuem camadas totalmente conectadas ao final da rede, responsáveis pela etapa de inferência. Isso permite a geração de predições e probabilidades sobre as classes definidas. A Figura 9 apresenta um exemplo de arquitetura CNN.

3 REVISÃO DA LITERATURA

Este capítulo apresenta o levantamento do estado da arte obtido por meio das revisões sistemáticas da literatura sobre *Deep Learning* e Visão Computacional.

3.1 QUESTÃO DE PESQUISA

Questão: Qual é o estado da arte em detecção de objetos, utilizando *deep learning* e visão computacional, com foco em detecção de pedestres?

População: Pesquisas sobre detecção de objetos, utilizando *deep learning*, disponíveis em bibliotecas eletrônicas.

Intervenção: Análise de pesquisas finalizadas ou em andamento na área de detecção de objetos.

Resultado: Comparação de pesquisas publicadas quanto a maturidade, uso de frameworks e técnicas de *deep learning* aplicadas.

Contexto: Bibliotecas digitais com acesso disponibilizado dentro da rede da Universidade Federal de Santa Catarina.

3.2 DEFINIÇÃO DA BUSCA

As buscas foram feitas sobre bibliotecas digitais renomadas e reconhecidas, como *Springer*, *ACM Digital Library*, *IEEE Xplore* e *Science Direct*.

Durante o levantamento, foram excluídas publicações que não implementam, experimentam ou propõe arquiteturas de *deep learning* com enfoque em detecção de objetos, descartando assim as que apenas tocam ou citam o tópico ou que utilizam como meio para introduzir assuntos específicos, como sistemas embarcados ou otimizações em protocolos de rede.

Os termos de busca utilizados foram *deep learning*, *computer vision*, *pedestrian detection* e *object detection*. Além disso, a busca foi limitada aos últimos dez anos de publicações, entre 2009 e 2019.

3.3 EXECUÇÃO DA BUSCA

Inicialmente a busca retornou um total de 567 resultados, sendo posteriormente analisados e estudados, excluindo assim artigos e publicações que não se enquadravam ao escopo do trabalho proposto. Os resultados de cada plataforma foram os seguintes:

- Springer: 132 obras;
- ACM Digitial Library: 85 obras;
- IEEE Xplore: 219 obras;
- Science Direct: 131 obras;

Sendo utilizadas as seguintes *strings* de busca e suas respectivas bibliotecas digitais:

- **Springer**

  TITLE("pedestrian detection"OR "pedestrian") AND ABSTRACT("deep learning"OR "object detection"OR "computer vision")

- **ACM Digital Library**

  acmdlTitle:('computer vision' 'pedestrian detection' 'pedestrian') AND recordAbstract:('deep learning' 'object detection' 'pedestrian detection' 'computer vision') AND keywords.author.keyword:('deep learning' 'object detection' 'pedestrian detection' 'computer vision')

- **IEEE Xplore**

  ("Document Title":"computer vision"OR "Document Title":"pedestrian detection"OR "Document Title":"pedestrian"OR "Document Title":"autonomous vehicle") AND ("Author Keywords": "deep learning"OR "Author Keywords": "object detection"OR "Abstract": "deep learning"OR "Abstract":"object detection")

- **Science Direct**

  TITLE("pedestrian"OR "pedestrian detection"OR "computer vision") AND TITLE-ABSTR-KEY("deep learning"OR "object detection"OR "pedestrian detection")

3.4 ANÁLISE DOS TRABALHOS

Durante o período de leitura e análise dos resultados da busca realizada, diversas publicações foram deixadas de lado, dado que as mesmas não se enquadravam com o escopo do presente trabalho. Os principais critérios de busca foram os seguintes:

I Relação com o escopo de detecção de pedestres;

II Relação com o escopo de detecção de objetos;

III Relação com a área de Inteligência Artificial;

IV Relação com a subárea de *Deep Learning*;

V Apresentam resultados compatíveis com modelos bem estabelecidos na comunidade científica;

VI Apresentam comparativos ou estudos atuais, a partir do ano de 2009;

Abaixo são apresentados artigos e publicações consideradas mais relevantes, baseado na relação dos mesmos com o tópico em questão e os critérios explicitados.

3.4.1 Ten Years of Pedestrian Detection, What Have We Learned?

- **Ano:** 2014
- **Evento ou obra:** Computer Vision - ECCV 2014 Workshops
- **Biblioteca digital:** *Springer*
- **Referência:** (BENENSON et al., 2014)

Esse artigo nos traz uma comparação de dez anos, entre o período de 2004 e 2014, na área de detecção de pedestres e objetos.

Os autores apresentam diversas técnicas e famílias de algoritmos para tal tarefa, bem como trazem provocações e experimentações em tópicos, observando quais dos assuntos causam maior impacto e melhoram os erros dos modelos propostos. Exemplos de tópicos abordados

são a adição de dados (de múltiplas bases de caso), detecção de contexto, modelos *multi-scale*, melhoria na qualidade de propriedades de entrada, entre outros.

Os mesmos apontam e destacam que, embora modelos treinados sobre um conjunto de dados não possuam um desempenho necessariamente bom quando testados em outros conjuntos, é perceptível que os mesmos mantém uma certa estabilidade sobre tais circunstâncias. Além disso, a adição de propriedades extras (como fluxo óptico e informações sobre o contexto das imagens utilizadas) aos modelos pode ser capaz de adicionar até cerca de 12% de ganho em questão de precisão.

O artigo finaliza propondo um novo modelo, utilizando conceitos de fluxo óptico e HOG, cuja performance supera a grande maioria de modelos na base de casos chamada de *Caltech-USA (DOLLÁR et al., 2012)*, como mostram os autores.

3.4.2 Scale-Aware Fast R-CNN for Pedestrian Detection

- **Ano:** 2018
- **Evento ou obra:** IEEE Transactions on Multimedia
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (LI et al., 2018)

Os autores do artigo em questão trazem a tona o problema de escala das imagens, quando trabalhando com reconhecimento de pedestres. Os mesmos citam que tal problema é apresentado por diversos motivos, como a diferença de distância entre os objetos em cena e ângulo de visão.

Para abordar tal problema, propõem um modelo de reconhecimento baseado no modelo *Fast R-CNN* (GIRSHICK, 2015), adicionando duas sub-redes. A primeira sendo responsável por classificar objetos que possuem um tamanho considerado pequeno, já a outra por objetos maiores, como mostra a Figura 10. Cada sub-rede possui um determinado peso, baseado no tamanho do objeto detectado. Assim, caso um pedestre esteja presente na imagem, mas em uma dimensão pequena, a primeira sub-rede terá uma influência maior no resultado do modelo.

Ao final do modelo, ele é capaz de informar qual a probabilidade de cada objeto detectado ser um pedestre, no escopo em questão, bem como *bounding boxes* que sinalizam a posição dos mesmos na cena.

Os experimentos realizados, principalmente os quais utilizam o conjunto de dados *KITTI*, dado que o mesmo será utilizado no presente trabalho, mostram um bom desempenho, se comparado com outras arquiteturas existentes.

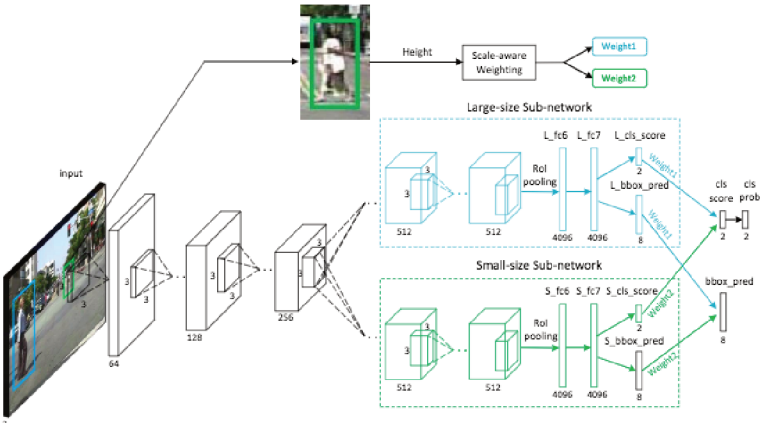


Figura 10 – Arquitetura SAF-RCNN

3.4.3 You Only Look Once: Unified, Real-Time Object Detection

- **Ano:** 2016
- **Evento ou obra:** IEEE Conference on Computer Vision and Pattern
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (REDMON et al., 2015), (Redmon et al., 2016)

O mesmo é citado por (GIRSHICK, 2015) como uma alternativa a detecção genérica de objetos em cenas, dado um conjunto de classes. Os autores dão maior enfoque no desempenho do modelo, sendo possível executá-lo em até 150 imagens por segundo (*fps*).

A abordagem apresentada consiste em fazer com que o modelo visualize a imagem inicial apenas uma vez, sem a necessidade de observar diversas partes da imagem mais de uma vez, como é o caso quando usamos técnicas baseadas em *sliding windows* ou *region proposals.*

A arquitetura consiste em dividir as imagem inicial em uma grade, na qual cada bloco ou elemento da mesma possui um tamanho pré-definido S, como mostra a Figura 11. Cada bloco dessa imagem inicial, dado o modelo proposto, pode prever até duas classes distintas, dentre as 20 existentes.

Em cada uma das células é gerado um conjunto de possíveis caixas de detecção de diversos tamanhos e proporções que são responsáveis por encontrar objetos contidos dentro de suas extensões. Ao final do processo, cada célula contém a predição de uma das classes propostas, sendo unidas com as predições das partes restantes da grade para assim obter os objetos presentes na imagem inicial.

Tal modelo possui a vantagem de levar em consideração o contexto no qual os objetos estão inseridos na cena, dado que a imagem como um todo é observada durante os processos de treinamento. Em contra-partida, a arquitetura proposta não possui o mesmo desempenho quando os objetos em cena são consideravelmente pequenos, dado o tamanho S de cada bloco.

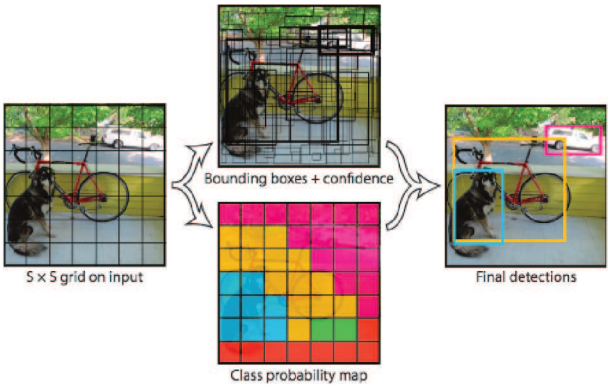


Figura 11 – Abordagem do modelo YOLOv1

3.4.4 YOLOv3: An Incremental Improvement

- **Ano:** 2018
- **Evento ou obra:** Computing Research Repository
- **Biblioteca digital:** *arXiv*

- **Referência:** (REDMON; FARHADI, 2018)

Os autores apresentam a última versão do modelo anteriormente proposto em (REDMON et al., 2015), trazendo diversas melhorias incrementais ao mesmo. O modelo manteve seu comportamento de extração de objetos através do modelo de *grids*.

A principal modificação está na arquitetura, onde agora a mesma apresenta 53 camadas de convolução, ao contrário das 24 camadas propostas pela primeira versão. Assim, um conjunto de operações de convolução utilizando filtros de dimensão `1x1`, seguidos de operações com filtros de dimensão `3x3` são aplicados. A Figura 12, extraída do próprio artigo relacionado, mostra as mudanças acima citadas.

É relatado que tal abordagem auxilia no reconhecimento de objetos de tamanho reduzido, dado que as operações são feitas com filtros menores. Entretanto, o desempenho de reconhecimento de objetos de tamanho médio e grande é prejudicado, bem como a performance computacional do modelo. Os autores citam como um ponto a ser explorado no futuro.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 12 – Modelo proposto como YOLOv3

3.4.5 Focal Loss for Dense Object Detection

- **Ano:** 2017
- **Evento ou obra:** IEEE International Conference on Computer Vision (ICCV)
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (LIN et al., 2017)

Os autores do artigo demonstram grande interesse em solucionar o problema de desequilíbrio(*imbalance*) de classes apresentado em diversos conjuntos de imagens. Tal conceito é apresentado como sendo a grande diversidade de objetos presentes nas mesmas em primeiro e segundo plano.

É apresentado uma nova função de custo para os modelos de aprendizado de máquina, chamado de *focal loss*. Tal função é definida de maneira a reduzir o resultado de custo de objetos classificados como objetos de interesse, dado que, de acordo com os autores, objetos falso-positivos podem ser responsáveis por consumo de recurso computacional que não retorna um aprendizado útil.

Assim sendo, com um custo maior para objetos considerados falso-positivos, os mesmos são descartados mais facilmente e não influenciam na performance do modelo. Além disso, é proposto um novo modelo de classificação, chamado de *RetinaNet*, mostrado na Figura 13.

Consiste de um nível inicial utilizando o conceito de rede piramidal de características, na qual extrai mapas de características de menor resolução, porém, com maior qualidade semântica. Tais mapas são expandidos com o intuito de detectar posições mais fiéis a imagem original.

Dada essa camada, duas sub-redes são responsáveis por gerar as posições e classificar as mesmas. É utilizado a função de custo proposta. Os resultados mostram que a arquitetura é responsável por uma precisão média(AP) superior a diversos outros modelos como YOLOv2 e Faster-RCNN no conjunto de dados COCO(LIN et al., 2014).

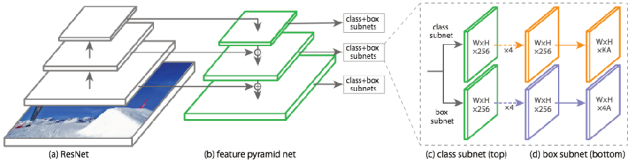


Figura 13 – Arquitetura do modelo *RetinaNet*

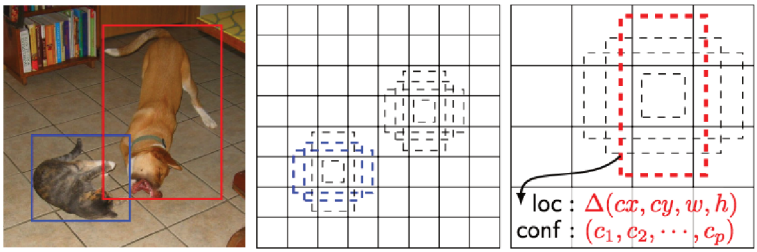
3.4.6 SSD: Single Shot MultiBox Detector

- **Ano:** 2016
- **Evento ou obra:** European Conference on Computer Vision (ECCV)
- **Biblioteca digital:** *Springer*
- **Referência:** (LIU et al., 2016)

Modelo utilizado como comparativo, principalmente dado seu desempenho computacional, por diversos artigos ((REDMON; FARHADI, 2018), (LIN et al., 2017)), utiliza-se da mesma ideia presente em YOLO, na qual é realizada uma divisão da imagem inicial em uma forma de grade, sendo a imagem inicial observada apenas uma vez.

São utilizadas 4 caixas de detecção para cada célula da grade, sendo as mesmas de proporções distintas, como mostra a Figura 14. É perceptível que a arquitetura utiliza caixas com proporções menores, referentes ao tamanho de cada célula, sendo responsáveis por detectar objetos menores.

Os autores realizam experimentos em diversos conjuntos de dados como Pascal VOC 2012 (M. et al.,) e MS COCO (LIN et al., 2014) que demonstram uma superioridade, em questão de capacidade computacional exigida, em relação a arquitetura YOLO. Além disso, são apresentados resultados que mostram uma precisão de classificação superior tanto ao modelo YOLO quanto ao modelo *Fast R-CNN*, anteriormente citados, principalmente se tratando de classes que envolvem o escopo do presente trabalho, como pessoas.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figura 14 – Modelo de detecção *Single Shot Detector*

3.4.7 MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications

- **Ano:** 2017
- **Evento ou obra:** Computing Research Repository (CoRR)
- **Biblioteca digital:** *arXiv*
- **Referência:** (HOWARD et al., 2017)

Os autores trazem a discussão a necessidade de utilização de modelos arquitetados para plataformas embarcadas e com baixa capacidade de recursos, dando foco em um modelo que possua baixa latência e que possa ser facilmente portado.

O trabalho apresenta o modelo MobileNet, tendo como focos principais os objetivos acima citados. As otimizações apresentadas sobre o modelo proposto devem-se a utilização de convoluções separáveis e de profundidade, sendo uma técnica responsável por separar convoluções iniciais em novas sobre imagens de dimensões menores, porém, em sequência, produzindo uma rede mais profunda.

Sua principal vantagem consiste no fato de redução de pesos para realizar a convolução em profundidade, dado que o resultado dessa camada pode ser controlada através do número de *kernels* utilizados na convolução. A Figure 15 apresenta a técnica utilizada pela arquitetura proposta.

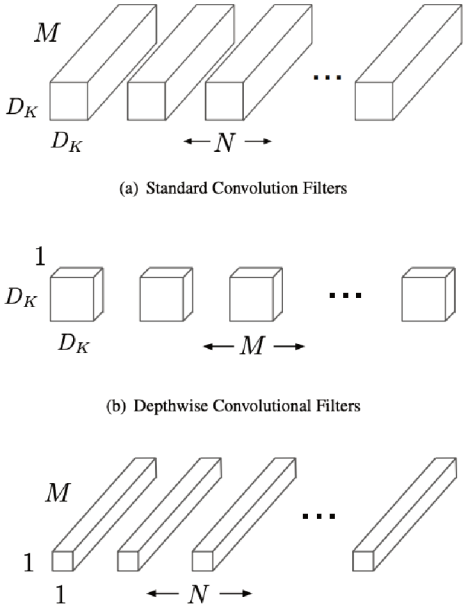


Figura 15 – Convoluções sobre a Arquitetura MobileNet V1

3.4.8 MobileNetV2: Inverted Residuals and Linear Bottlenecks

- **Ano:** 2018
- **Evento ou obra:** IEEE/CVF Conference on Computer Vision and Pattern Recognition
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (Sandler et al., 2018)

Consiste no modelo sucessor do trabalho apresentado anteriormente. Os autores propõe duas principais melhorias sobre a arquitetura já existente.

A primeira é nomeada de redes residuais inversas, responsáveis por conectar as camadas iniciais do modelo diretamente com as camadas finais. A abordagem permite que o modelo utilize o padrão que inicia com convoluções de grande número canais, reduza as dimensões nas camadas intermediárias e volte a ampliar ao final, reduzindo assim a quantidade de pesos necessários para serem armazenados pelo modelo. A Figura 16 apresenta o conceito descrito.

Outra melhoria proposta consiste no uso da função de ativação ReLU6, sendo uma alteração do modelo original, limitando o valor máximo da função a um número de 3 bits, o valor decimal 6. Isso garante precisão de ponto flutuante.

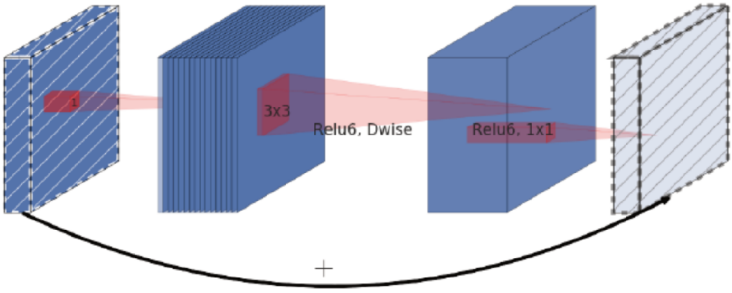


Figura 16 – Redes residuais inversas sobre a Arquitetura MobileNet V2

3.4.9 Rethinking the Inception Architecture for Computer Vision

- **Ano:** 2016
- **Evento ou obra:** Computer Vision and Pattern Recognition
- **Biblioteca digital:** *arXiv*
- **Referência:** (SZEGEDY et al., 2016)

Com foco em resolver problemas de variância de altura e largura aparente de objetos em imagens, o modelo que serve como base para o trabalho apresentado utiliza o conceito de múltiplas convoluções por camada, utilizando dimensões distintas para os filtros em cada uma

delas. Ao final, o modelo realiza a concatenação dos resultados de cada filtro (Szegedy et al., 2015).

A versão 2, apresentada no trabalho, baseia-se no mesmo conceito como mostrado na Figura 17, porém, reduz convoluções de dimensão mais alta com foco em melhoria de desempenho. Os autores citam uma melhoria de 2.78 vezes utilizando duas convoluções de tamanho 3x3 sobre uma de tamanho 5x5 (utilizada no modelo inicial).

Além disso, os pesquisadores fizeram alterações para filtros que utilizem apenas convoluções do tipo 1xN ou Nx1, reduzindo a quantidade de pesos necessária a serem treinados.

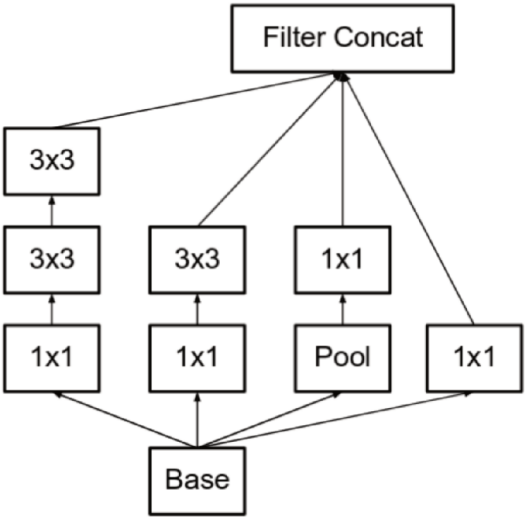


Figura 17 – Convoluções em largura sobre a Arquitetura Inception V2

3.4.10 Dense Scene Flow Based Coarse-to-Fine Rigid Moving Object Detection for Autonomous Vehicle

- **Ano:** 2017
- **Evento ou obra:** IEEE Access
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (XIAO et al., 2017)

O trabalho em questão tem como foco buscar por objetos dinâmicos, ou seja, através de um conjunto de imagens e o significado de contexto presente entre cada sequência das mesmas, ao contrário de abordagens tradicionais, nais quais executam modelos sobre imagens estáticas.

Seu modelo é composto por quatro fases, como mostra a Figura 18, nas quais são responsáveis pelas seguintes tarefas:

1. Nessa etapa, é calculado o mapa de disparidade e de fluxo óptico denso para realizar a estimativa de movimento. Sendo assim aplicados sobre um processo de extração de *superpixels*, responsável por agrupar pontos com características comuns de cores entre si;

2. No segundo passo do modelo, são segmentados objetos que serão classificados mais adiante, independente de seu movimento;

3. Logo após, esse passo é responsável por segmentar e entender apenas os objetos que se movimentam na cena original, detectando assim sua direção;

4. Como etapa final, são utilizados algoritmos de refinamento para geração de resultados mais precisos de classificação e estimativa de movimento;

A técnica adotada pelos autores não utiliza o conceito de *deep learning*, não fazendo assim o uso de técnicas como redes neurais convolucionais, mas sim de técnicas de visão computacional clássica. Como consequência, é necessária a abordagem em diversos passos.

Os experimentos mostram que melhorias em questão de performance de classificação sobre a ferramenta (GEIGER; ZIEGLER; STILLER, 2011), enfrentando problemas de detecção em objetos cuja luminosidade é reduzida (sombras e cenas noturnas). Nenhum comparativo com modelos de *deep learning* é apresentado.

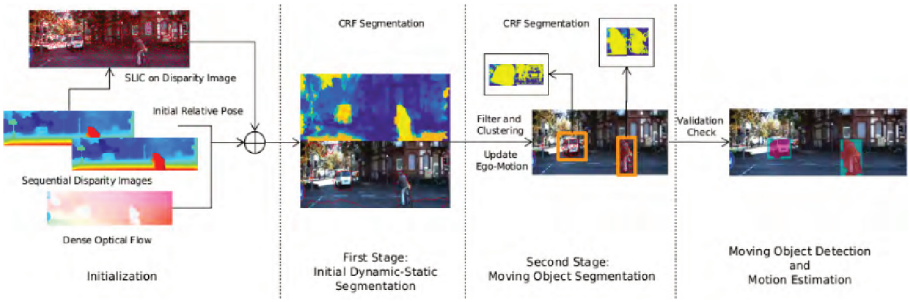


Figura 18 – Modelo de quatro fases

3.4.11 FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks

- **Ano:** 2017
- **Evento ou obra:** IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (ILG et al., 2017)

Citado por (XIAO et al., 2017), o texto em questão define o modelo nomeado *FlowNet 2.0*. Sendo uma evolução da arquitetura *FlowNet*(FISCHER et al., 2015), os autores focam na detecção de deslocamentos pequenos, cujo modelo inicial não apresentava bom desempenho.

A ideia básica da arquitetura consiste em conectar diversas subredes para realizar a estimativa de fluxo óptico de um par de imagens sequenciais em uma cena dinâmica, como mostra a Figura 19. É utilizada uma combinação de modelos *FlowNetS* e *FlowNetC* para a construção final, proposta pelos próprios autores.

FlowNetS, apresentado na Figura 20, é composto por uma rede neural convolucional, cuja entrada são duas imagens de uma sequência concatenadas e a arquitetura é responsável por estimar o mapa de fluxo óptico para tal. *FlowNetC* aproveita-se da mesma ideia, porém, inicialmente cada imagem passa por uma rede distinta, sendo assim correlacionadas em passos mais adiante, sendo assim possível realizar a estimativa, como mostra a Figura 21.

Os mesmos ainda apresentam uma alternativa para detecção de pequenos deslocamentos que, em muitos casos, acabam sendo ofusca-

dos por deslocamentos maiores ou movimentos bruscos nas imagens. O modelo alternativo é alcançado através de um conjunto de dados distinto do original e aplicando técnicas para penalização de pesos para grandes deslocamentos.

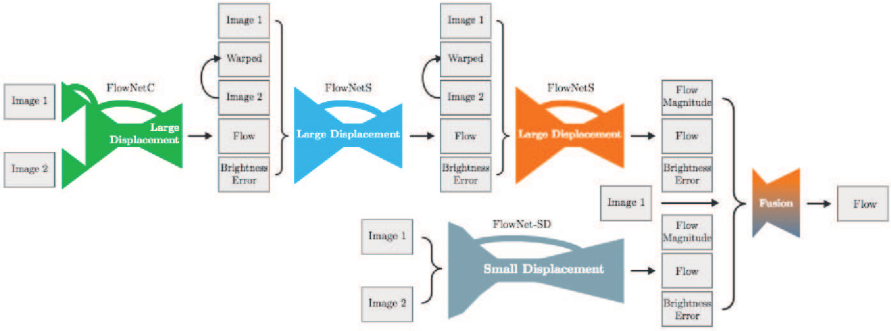


Figura 19 – Arquitetura de sub-redes que compõe o modelo *FlowNet 2.0*

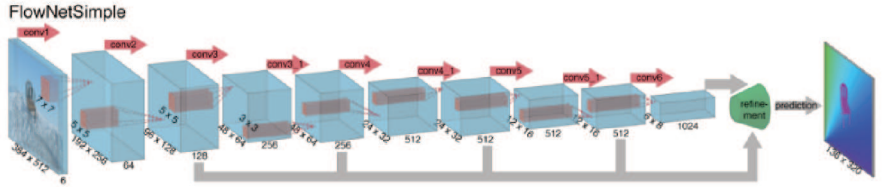


Figura 20 – Modelo *FlowNetS*

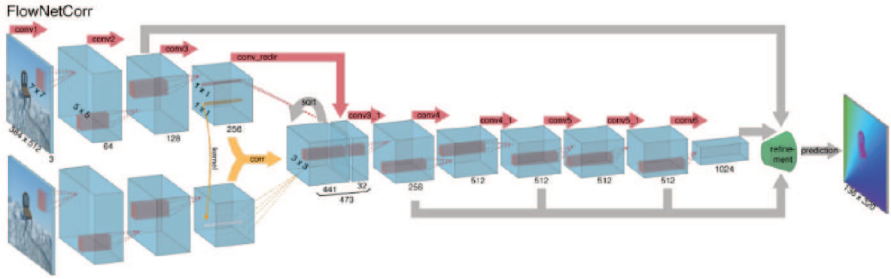


Figura 21 – Modelo *FlowNetC*

3.4.12 Faster R-CNN

- **Ano:** 2017
- **Evento ou obra:** IEEE Transactions on Pattern Analysis and Machine Intelligence
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (REN et al., 2017)

O modelo apresentado pelos autores consiste em uma melhoria na arquitetura apresentada pelo modelo *Fast R-CNN*. Sua principal modificação consiste na ausência de busca seletiva no processo de geração de regiões propostas, aumentando assim o desempenho e reduzindo o uso de recursos do dispositivo de inferência.

Para obter o mesmo resultado, os autores propõe uma rede separada da original utilizada somente para a predição das regiões propostas. Após essa predição, o modelo passa por uma camada de região de interesse para redução de dimensões, sendo assim utilizada para auxílio na classificação dos objetos. A Figura 22 apresenta as modificações na arquitetura em questão.

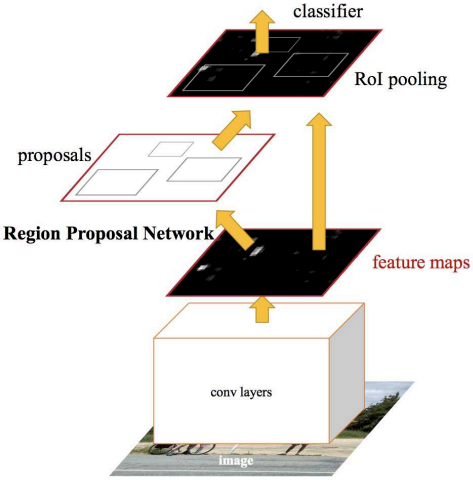


Figura 22 – Arquitetura *Faster R-CNN*

3.4.13 Mask R-CNN

- **Ano:** 2017
- **Evento ou obra:** IEEE International Conference on Computer Vision (ICCV)
- **Biblioteca digital:** *IEEE Xplore*
- **Referência:** (He et al., 2017)

O artigo em questão traz uma proposta distinta das outras, realizando detecção e segmentação dos conjuntos de entrada. Como sucessor do modelo *Faster R-CNN* (REN et al., 2017), um de seus princípios consiste na busca por desempenho ao realizar a inferência de dados de entrada.

A principal mudança com relação a seu predecessor é a adição, além do ramo de detecção de objetos, de um ramo de extração de máscaras de segmentação de objetos. Ambos os ramos executam em paralelo. A Figura 23 mostra alguns exemplos de resultados da execução do modelo.

Figura 23 – Exemplos de inferência do Modelo *Mask R-CNN*

O modelo em questão é dividido em estágios. O primeiro estágio, assim como seu predecessor, consiste de uma rede para geração de possí-

veis objetos a serem detectados (*Region Proposals Network*). Em um segundo estágio, a rede se divide em um ramo de detecção/categorização e outro de segmentação.

Nesse estágio, vale notar que não há a dependência dos resultados de segmentação para obter os resultados de detecção, dado que os mesmos são calculados isoladamente em paralelo, melhorando assim o desempenho geral da rede.

Para que seja possível uma boa precisão de detecção de objetos, o modelo realiza a compensação quanto as divisões através de seu treinamento. Durante essa fase, cada região de interesse gerada leva uma erro, calculado pela seguinte fórmula

$$L = L_{cls} + L_{box} + L_{mask}$$

Na qual, L_{cls} corresponde a função custo da etapa de classificação, L_{box} a função de custo para as caixas de seleção geradas (*bounding boxes*) e, diferente de seu modelo predecessor, L_{mask} corresponde a média binária do erro de entropia.

Os resultados apresentados mostram que o modelo, com todos os seus ramos, consegue inferir a uma velocidade de 5 imagens por segundo, sendo 195ms por imagem sobre a GPU Tesla M40, da fabricante NVIDIA Corporation.

3.5 COMPARATIVO DOS TRABALHOS ANALISADOS

Através do estudo dos trabalhos escolhidos é possível perceber a velocidade na qual o campo de estudo avança, dado que, dentro dos estudos analisados, diversos deles são continuações e maiores explorações sobre trabalhos prévios, embora os mesmos já possuam resultados expressivos e sejam capazes de se adaptar a um grande número de cenários distintos.

Dado o conjunto de obras acima resumidas, é possível comparar tais modelos teóricos, baseando-se nos resultados apresentados pelos autores, como mostra a Tabela 1.

É possível traduzir as legendas da mesma com os seguintes identificadores:

I Proposta de arquitetura de detecção;

II Apresenta estudo comparativo;

III Modelo possui ferramentas ou métricas para rastreamento de objetos;

IV Modelo possui ferramentas ou métricas para detecção de pose;

V Modelo possui ferramentas ou métricas para geração de fluxo óptico;

VI O trabalho utiliza técnicas de *deep learing*;

Referência	**I**	**II**	**III**	**IV**	**V**	**VI**
(REDMON et al., 2015)	✓		✓	✓		✓
(REDMON; FARHADI, 2018)	✓		✓	✓		✓
(GIRSHICK, 2015)	✓		✓	✓		✓
(ILG et al., 2017)	✓		✓	✓	✓	✓
(XIAO et al., 2017)	✓			✓	✓	
(LIN et al., 2017)	✓		✓	✓		✓
(LI et al., 2018)	✓		✓	✓		✓
(HOWARD et al., 2017)	✓	✓	✓			✓
(Sandler et al., 2018)	✓	✓	✓			✓
(BENENSON et al., 2014)		✓				✓
(LIU et al., 2016)	✓		✓	✓		✓
(REN et al., 2017)	✓	✓	✓			✓
(He et al., 2017)	✓	✓	✓	✓		✓
(SZEGEDY et al., 2016)	✓	✓	✓			✓

Tabela 1 – Tabela comparativa dos trabalhos apresentados.

3.6 ACELERADORES EMBARCADOS

Além da pesquisa acima, com foco principal em utilização e modelos de reconhecimento, foi realizada uma nova busca com foco no *benchmark* dos modelos, com foco na família de sistemas embarcados Jetson, apresentada em mais detalhes na seção 4.1.

(FRANKLIN, 2019) expõe uma análise de desempenho do modelo Jetson Nano. O autor executa um conjunto de modelos de *deep learning* pré-treinados existentes na literatura sobre um conjunto próprio para extrair dados relacionados a performance do dispositivo. A biblioteca

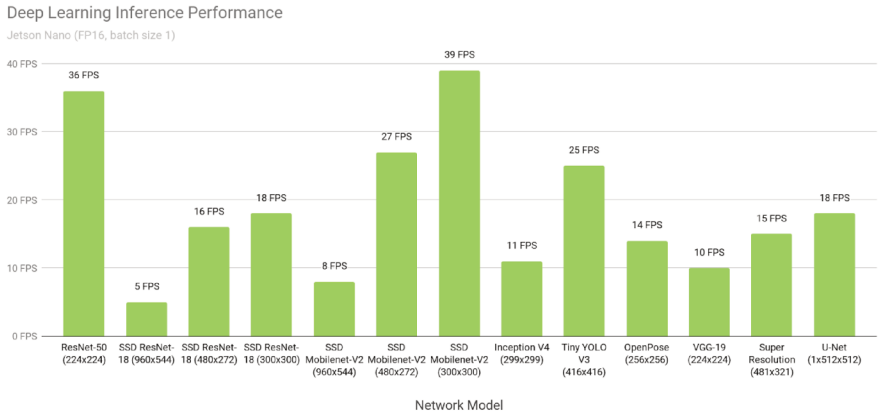


Figura 24 – Resultados dos experimentos sobre a Jetson Nano

TensorRT é utilizada para otimizar os passos de inferência de cada modelo. Os resultados podem ser vistos na Figura 24.

O principal diferencial em comparação ao presente trabalho é em relação a normalização da resolução dos dados de entrada para a inferência, o que traz uma visão mais clara quanto a comparativo de desempenho de modelos de reconhecimento.

(FRANKLIN, 2017) apresenta um *benchmark* sobre a plataforma otimizada para o ambiente de produção do mesmo modelo utilizado no trabalho vigente. Uma das principais diferenças se comparado ao relatório é o tipo de redes testadas sobre a plataforma, na qual autores utilizam redes conhecidas como base, geralmente utilizadas em conjunto com redes de detecção. Esse conjunto de redes é o foco do trabalho.

		NVIDIA Jetson TX1	NVIDIA Jetson TX2		
		Max Clock (998 MHz)	Max-Q (854 MHz)	max-P (1122 MHz)	Max Clock (1302 MHz)
GoogLeNet batch=2	Perf	141 FPS	138 FPS	176 FPS	201 FPS
	Power (AP+DRAM)	9.14 W	4.8 W	7.1 W	10.1 W
	Efficiency	15.42	28.6	24.8	19.9
GoogLeNet batch=128	Perf	204 FPS	196 FPS	253 FPS	290 FPS
	Power (AP+DRAM)	11.7 W	5.9 W	8.9 W	12.8 W
	Efficiency	17.44	33.2	28.5	22.7
AlexNet batch=2	Perf	164 FPS	178 FPS	222 FPS	250 FPS
	Power (AP+DRAM)	8.5 W	5.6 W	7.8 W	10.7 W
	Efficiency	19.3	32	28.3	23.3
AlexNet batch=128	Perf	505 FPS	463 FPS	601 FPS	692 FPS
	Power (AP+DRAM)	11.3 W	5.6 W	8.6 W	12.4 W
	Efficiency	44.7	82.7	69.9	55.8

Figura 25 – Resultados sobre a plataforma Jetson TX2 em ambiente de produção

Por último, (FRANKLIN, 2016) traz um *benchmark* comparativo sobre o desempenho da biblioteca TensorRT, além de expor suas melhorias referentes a geração anterior do mesmo pacote de ferramentas. Os resultados do estudo, apresentado na Figura 26, são úteis considerando a versão que mais encaixa-se com as necessidades do desenvolvedor, sendo que a biblioteca não é mandatória para uso sobre a plataforma.

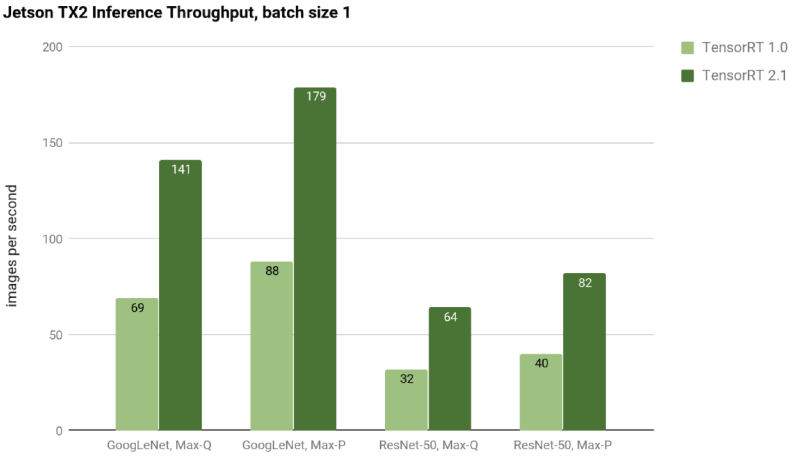


Figura 26 – Resultados comparativos entre TensorRT V1 e V2

4 DESENVOLVIMENTO

4.1 AMBIENTE DE EXPERIMENTAÇÃO

Com os recentes avanços nas áreas apresentadas, a indústria de hardware computacional encontrou um nicho de produção sobre plataformas embarcadas com aceleradores otimizados para modelos de IA. A Tabela 2 apresenta exemplos de aceleradores produzidos nos últimos anos pela indústria.

Baseando-se em tais modelos, foi determinado a plataforma Jetson TX2, produzida pela empresa NVIDIA Corporation, como base para o presente trabalho. A Figura 28 apresenta o kit de desenvolvimento sobre a plataforma, utilizado para os experimentos.

A plataforma utilizada executa o sistema operacional Ubuntu 16.04 LTS. Com uma central de processamento (CPU) utilizando a arquitetura ARM64, os seguintes pacotes e dependências foram utilizados:

- JetPack 3.3 for Jetson TX2

  Conjunto de software fornecido pelo fabricante como kit de desenvolvimento, contendo *drivers* e auxiliares de compilação.

- CUDA 9.0 (NICKOLLS et al., 2008)

  Conjunto de ferramentas fornecido pela fabricante. Disponibiliza uma interface comum e otimização de operações para o chip gráfico presente na plataforma.

- OpenCV 3.4.6 (BRADSKI, 2000)

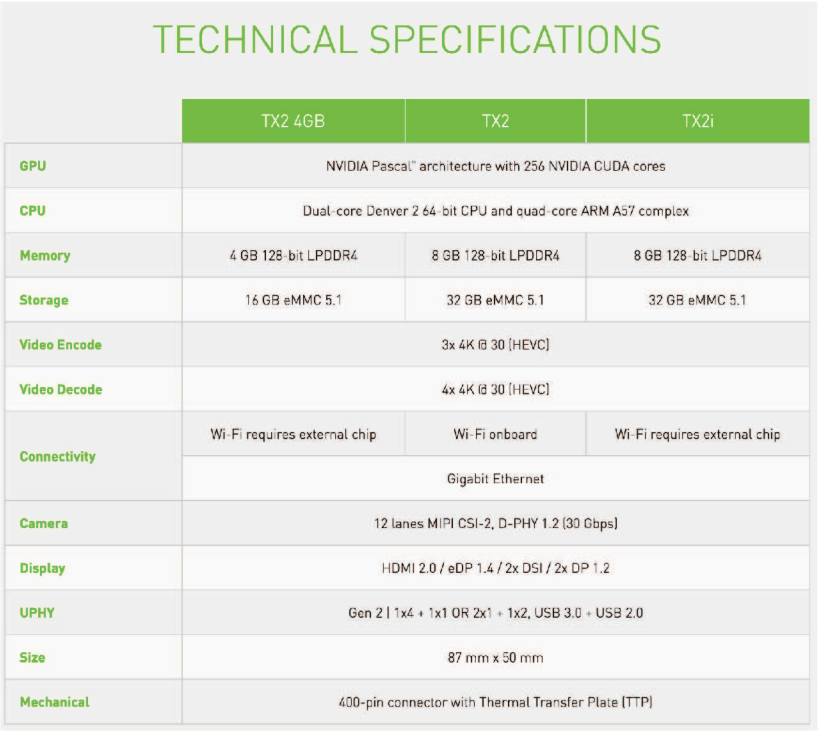
  Biblioteca de código aberto para processamento de imagens. Utilizada como alternativa para execução e treinamento dos modelos e pré e pós processamento das imagens de validação.

- Tensorflow 1.14.0 (ABADI et al., 2015)

  Biblioteca de código aberto para treinamento e execução de fases de inferência das redes avaliadas, bem como serialização dos modelos para uso futuro.

Fabricante	Modelo
NVIDIA	Jetson Nano
NVIDIA	Jetson AGX Xavier
NVIDIA	Jetson TX2
Intel®	Nervana™ NNP-T
Intel®	Nervana™ NNP-I
Qualcomm	Snapdragon 855
IBM	TrueNorth (Sawada et al., 2016)

Tabela 2 – Tabela comparativa de componentes existentes



TECHNICAL SPECIFICATIONS

	TX2 4GB	TX2	TX2i
GPU	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores		
CPU	Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex		
Memory	4 GB 128-bit LPDDR4	8 GB 128-bit LPDDR4	8 GB 128-bit LPDDR4
Storage	16 GB eMMC 5.1	32 GB eMMC 5.1	32 GB eMMC 5.1
Video Encode	3x 4K @ 30 (HEVC)		
Video Decode	4x 4K @ 30 (HEVC)		
Connectivity	Wi-Fi requires external chip	Wi-Fi onboard	Wi-Fi requires external chip
	Gigabit Ethernet		
Camera	12 lanes MIPI CSI-2, D-PHY 1.2 (30 Gbps)		
Display	HDMI 2.0 / eDP 1.4 / 2x DSI / 2x DP 1.2		
UPHY	Gen 2	1x4 + 1x1 OR 2x1 + 1x2, USB 3.0 + USB 2.0		
Size	87 mm x 50 mm		
Mechanical	400-pin connector with Thermal Transfer Plate (TTP)		

Figura 27 – Especificações Técnicas para as variantes do modelo Jetson TX2

Figura 28 – NVIDIA Jetson TX2

4.2 MÉTRICAS

Analisando a literatura relacionada a *benchmarks* sobre desempenho de modelos de IA em sistemas embarcados de propósito geral, o presente trabalho possui foco no fomento do tópico e realização de experimentos sobre sistemas reais.

Foi desenvolvido um *benchmark* de propósito geral sobre determinados componentes de um sistema embarcado de inferência. A Tabela 3 apresenta as condições e os componentes metrificados.

A decisão sobre tais componentes baseia-se na alta busca por tais elementos em *benchmarks* existentes na literatura. Além disso, sabe-se que o desempenho de GPUs é um dos fatores de maior influência sobre um modelo baseado em redes neurais (Li et al., 2017).

Para trabalhar com o tópico, temperatura média de GPU e tempo de inferência, por exemplo, foram coletadas no processo, com o intuito fornecer uma visão mais clara das capacidades do sistema.

Métrica	**Mensurado por**
Frames por Segundo (FPS)	Agrupado por interação
Tempo de Inferência	Tempo médio de todas as interações
Temperatura da GPU	Tempo médio e desvio padrão
Uso de memória	Tempo médio e desvio padrão

Tabela 3 – Elementos mensurados durante o *benchmark*

Para ser possível coletar dados que forneçam informação suficiente para a tomada de decisão da proposta, decidiu-se por executar um conjunto de modelos baseados em redes neurais convolucionais já existentes na literatura. A Tabela 4 apresenta as arquiteturas utilizadas no projeto. Vale notar que os modelos utilizados foram coletados dos autores originais, treinados sobre os conjuntos de dados apresentados na tabela.

A decisão de utilizar CNNs foi tomada com base nos componentes visuais presentes no micro-chip, bem como a facilidade de trabalhar com tais dados e modelos em arquiteturas baseadas em GPU. Além disso, o contexto mais provável de execução do sistema em questão é sobre carros autônomos, sendo o foco da fabricação do sistema.

4.3 MODELOS

Além disso, a decisão de utilizar modelos pré-treinados para a coleta de dados foi tomada, dado que métricas de performance de precisão das etapas de detecção, como intersecção sobre união (IoU), não são foco do presente trabalho e os autores dos modelos utilizados, em maioria, apresentam estudos comparativos com tais métricas.

Arquitetura	**Treinado em**	**Referência**
Faster R-CNN	Microsoft COCO	(REN et al., 2017)
Mask R-CNN	Microsoft COCO	(He et al., 2017)
SSD Inception V2	Microsoft COCO	(SZEGEDY et al., 2016), (LIU et al., 2016)
SSD Mobilenet V1	Microsoft COCO	(HOWARD et al., 2017), (LIU et al., 2016)
SSD Mobilenet V2	Microsoft COCO	(Sandler et al., 2018), (LIU et al., 2016)

Tabela 4 – Modelos de *deep learning* utilizados para experimentação

Os modelos acima descritos e utilizados para experimentação e coleta de dados foram explorados em detalhes de implementação através da Seção 3.

4.4 IMPLEMENTAÇÃO

Em relação a implementação, algumas premissas foram utilizadas como base para a coleta de informações sobre o sistema embarcado, descritas a seguir.

- Utilização de TensorFlow[1] com cuDNN, da NVIDIA;
- Ser, ao máximo, de propósito geral, permitindo modificações futuras;

A decisão de utilizar a biblioteca TensorFlow baseia-se no suporte ao uso de tecnologias proprietárias de aceleração dos modelos de inferência, diferente de algumas de suas alternativas diretas.

A implementação do *benchmark* utilizado no projeto foi realizada através da linguagem de programação Python, sendo executada por qualquer interpretador com suporte a versão 3 ou superior. Todo o código fonte pode ser encontrado no sistema de controle de repositórios GitHub [2] sobre a licença GPL 3.0.

Durante o processo de inferência, cada imagem de entrada é redimensionada para uma dimensão em comum, com foco em normalizar os resultados do comparativo. Um dos principais argumentos sobre a premissa tem como base cenários de teste reais, nos quais tende-se a utilizar um mesmo equipamento para coleta de imagens de entrada em uma dimensão padronizada.

Para o código referenciado, é inicializado um coletor de métricas. Após a execução de cada passo de inferência, os coletores são parados e registram as informações referentes a imagem de entrada. Os coletores utilizados são responsáveis pela coleta, de modo geral, de 4 áreas do sistema:

- Dados temporais: Como tempo decorrido, tempo de inferência e outras;
- Dados de CPU/GPU: Como temperatura média de ambos os componentes;
- Dados de memória: Como uso do processo, uso de *swap* e outras;
- Dados de FPS: Como número de imagens inseridas e contagem de predições;

[1] (ABADI et al., 2015)
[2] https://github.com/lucaspbordignon/onYourMarks

4.5 CONJUNTO DE DADOS

Para a execução dos experimentos do trabalho em questão foram tomados como base dois conjuntos de dados existentes na academia. A decisão sobre os mesmos consiste em conjuntos de dados de propósito geral e com dimensões e cenários factíveis de um ambiente de carro autônomo, ambiente provável para utilização do sistema embarcado utilizado para experimentação.

4.5.1 Microsoft COCO

O conjunto de dados Microsoft COCO tem como objetivo ser uma base de dados com imagens reais com foco em detecção de objetos, segmentação e legendas. Suas imagens possuem diversos contextos sobre 80 categorias de objetos distintos. A Figura 29 mostra alguns exemplos de dados extraídos da base.

Figura 29 – Imagens de exemplo presentes na base de dados Microsft COCO

Os modelos testados no presente trabalho foram treinados sobre o conjunto de dados. A base fornece 330 mil imagens, sendo cerca de 41 mil delas separadas pelos próprios autores como imagens de teste e validação, sendo essas testadas sobre o *benchmark* em questão. As imagens presentes no conjunto de dados possuem dimensões variadas, para permitir aprendizado sobre diversos modelos de dispositivos de captura de imagem.

4.5.2 KITTI Stereo Vision

A base de dados KITTI Stereo Vision, também chamado de conjunto de imagens de *benchmark* consiste de 200 imagens de treinamento e 200 cenários de teste sobre o formato PNG, reduzindo perdas e ruídos sobre os recursos. Ambos cenários de teste e treinamento foram utilizados durante os experimentos.

Os dados apresentados pelos autores possuem uma dimensão pa-

dronizada de 375 pixeis de altura por 1242 pixeis de largura e 3 canais de cores (RGB). Além disso, as cenas apresentadas pelos dados foram coletados sobre cenários rodoviários, tanto urbano quanto rural, e através de câmeras estéreo.

Figura 30 – Imagens de exemplo presentes na base de dados KITTI Stereo Vision

4.6 RESULTADOS

Após o desenvolvimento do algoritmo, o mesmo foi executado sobre dois conjuntos de dados popularmente utilizados no contexto de carros autônomos, chamados de Microsoft COCO (LIN et al., 2014) e KITTI Stereo Vision (MENZE; GEIGER, 2015).

A plataforma utilizada para teste, NVIDIA Jetson TX2, possui um conjunto de modos de operação, sendo conjuntos de configurações com foco em otimização para utilização do sistema. Durante os experimentos o modo "Max-N"foi utilizado e os testes sobre cada modelos foram seguidos de um período de repouso, de em torno de 20 minutos, entre iterações com foco em reduzir a interferência de temperatura sobre os componentes entre execuções. Os modos são apresentados na Tabela 5.

As imagens abaixo apresentam resultados de detecção para cada um dos conjuntos de dados utilizados nos experimentos.

Modo de operação	Nome	Núcleos Denver Habilitados	Núcleos ARM A57 Habilitados	Frequência de GPU
0	Max-N	2	4	1.30GHz
1	Max-Q	0	4	0.85GHz
2	Max-P Core-All	2	4	1.12GHz
3	Max-P ARM	0	4	1.12GHz
4	Max-P Denver	2	1	1.12GHz

Tabela 5 – Modos de operação sobre a plataforma Jetson TX2

Figura 31 – Resultados de detecção - SSD MobileNet V1 sobre Microsoft COCO

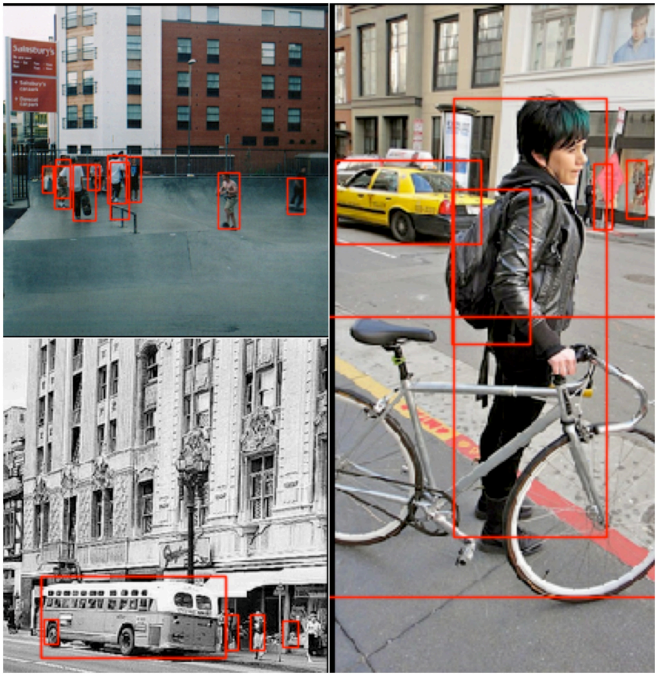


Figura 32 – Resultados de detecção - SSD MobileNet V2 sobre Microsoft COCO

Figura 33 – Resultados de detecção - SSD Inception V2 sobre KITTI Stereo Vision

Figura 34 – Resultados de detecção - Faster R-CNN sobre KITTI Stereo Vision

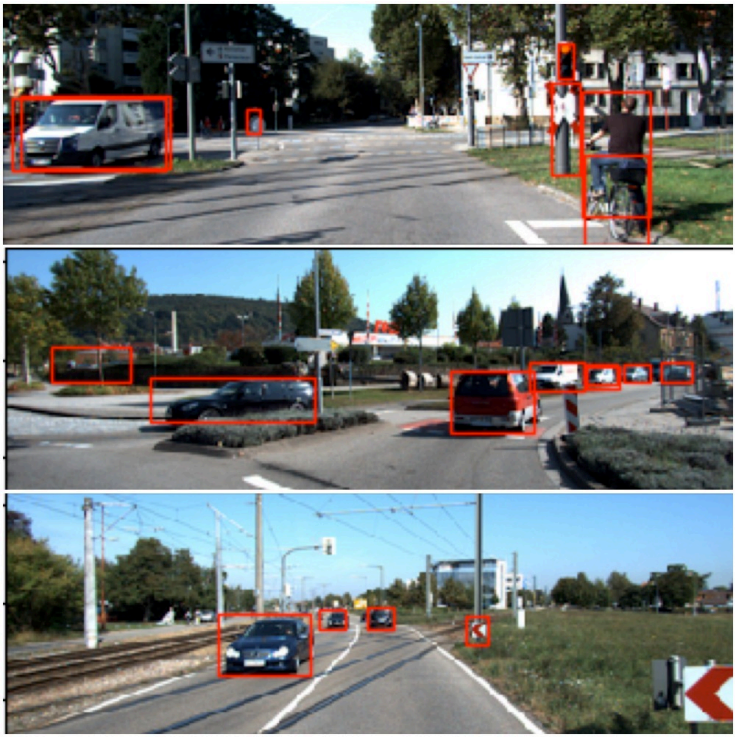


Figura 35 – Resultados de detecção - Mask R-CNN sobre KITTI Stereo Vision

Os gráficos a seguir exploram os resultados do *benchmark* sobre os conjuntos de dados previamente descritos. Cada entidade realiza expõe dados sobre um recurso de sistema distinto.

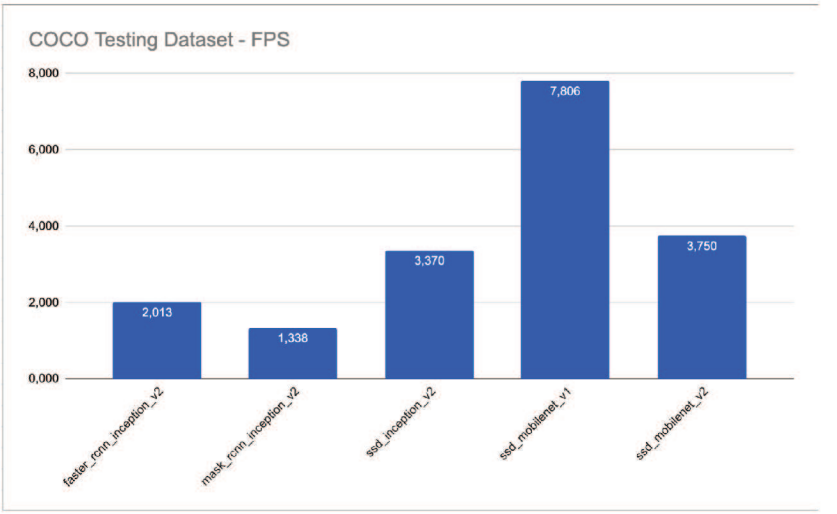


Figura 36 – Resultados do *benchmark* - Frames por Segundo - Microsft COCO

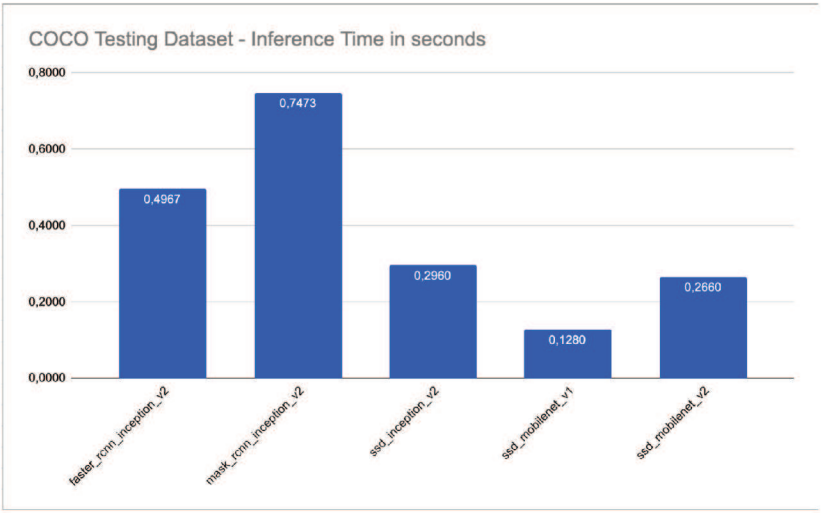


Figura 37 – Resultados do *benchmark* - Tempo médio de inferência - Microsft COCO

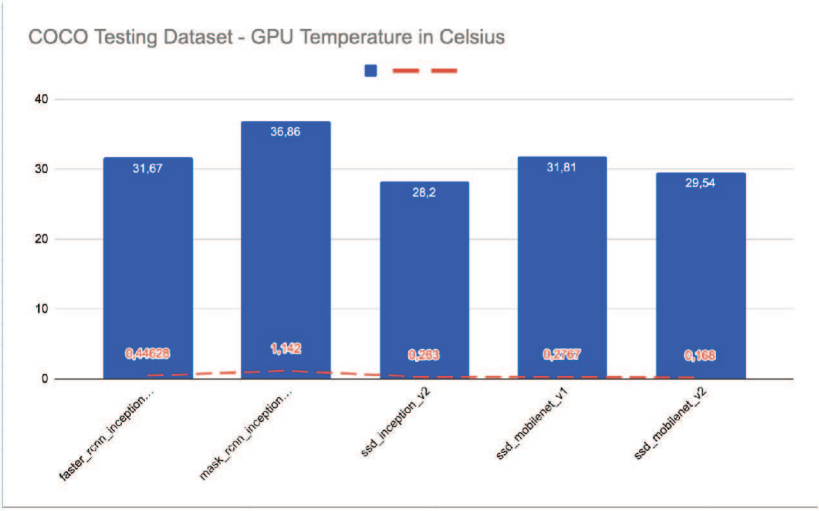


Figura 38 – Resultados do *benchmark* - Média e desvio padrão de temperatura da GPU - Microsft COCO

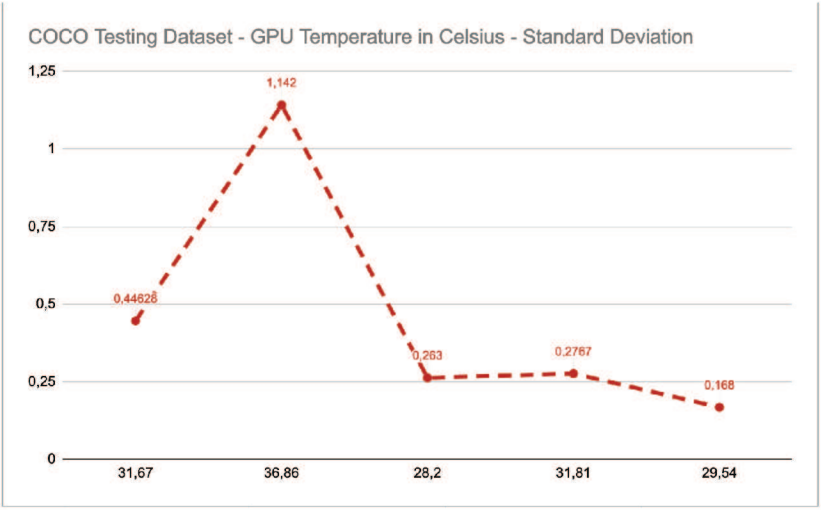


Figura 39 – Resultados do *benchmark* - Desvio padrão de temperatura da GPU - Microsft COCO

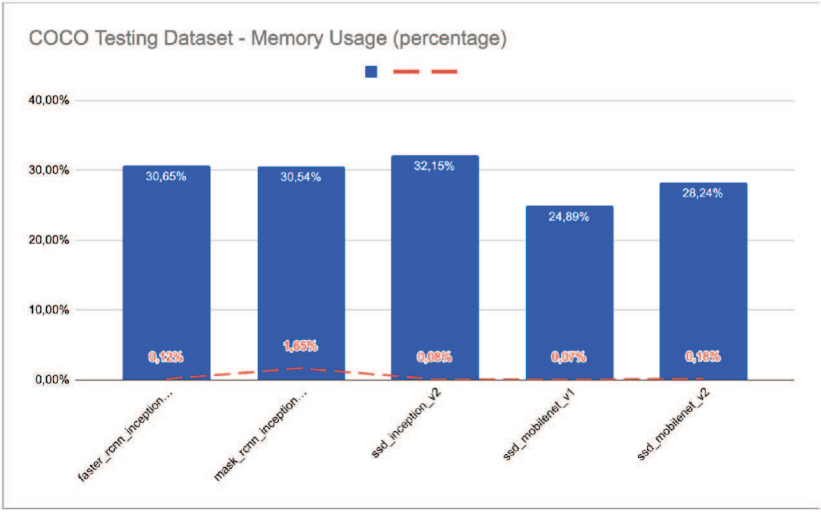


Figura 40 – Resultados do *benchmark* - Média e desvio padrão de memória da GPU - Microsft COCO

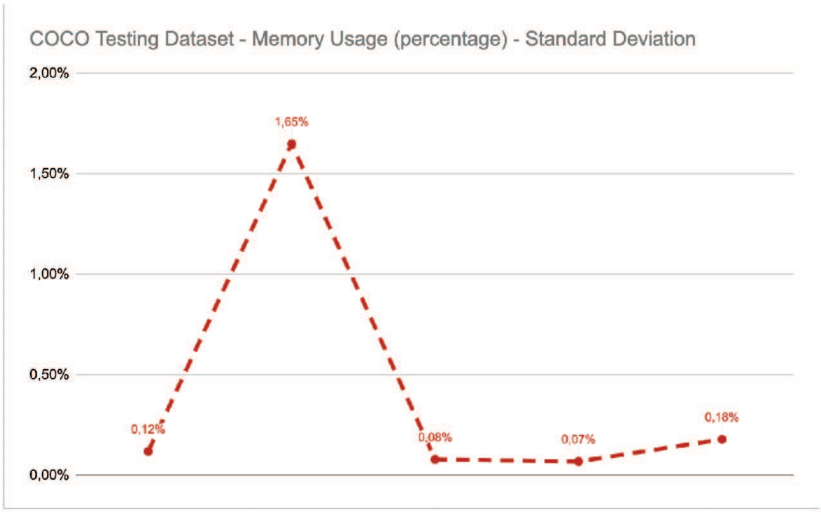


Figura 41 – Resultados do *benchmark* - Desvio padrão de memória da GPU - Microsft COCO

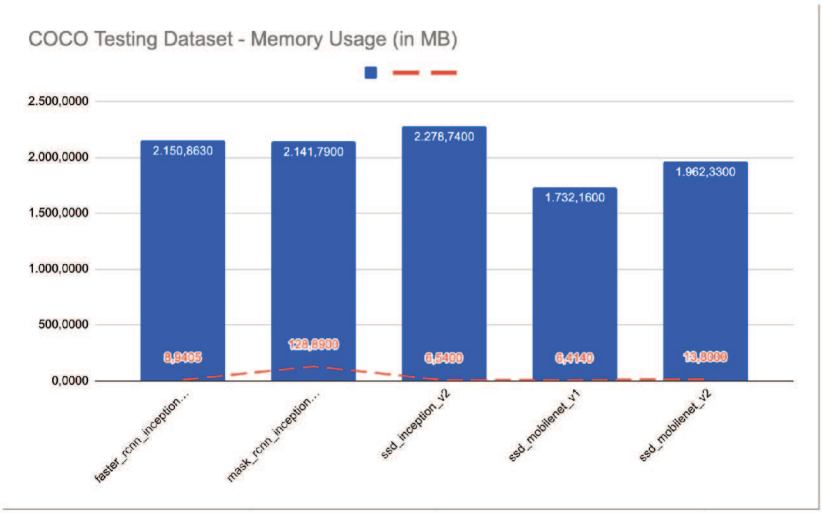


Figura 42 – Resultados do *benchmark* - Média e desvio padrão de memória da GPU - Microsft COCO

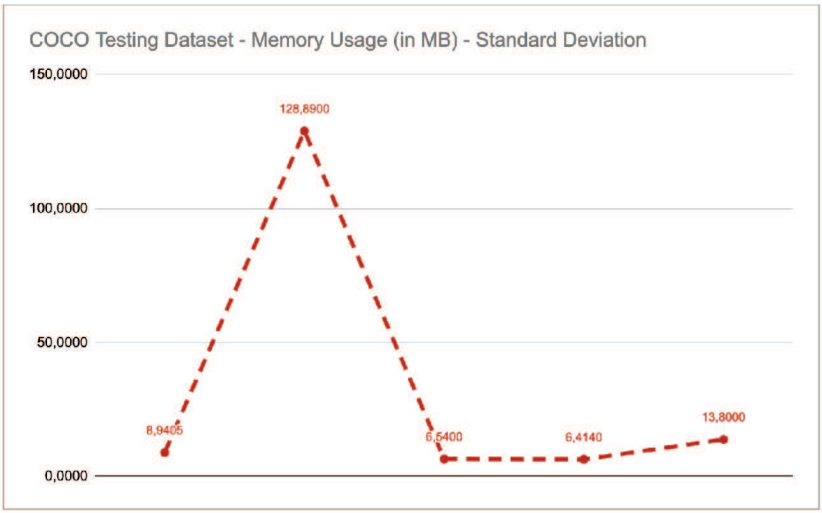


Figura 43 – Resultados do *benchmark* - Desvio padrão de memória da GPU - Microsft COCO

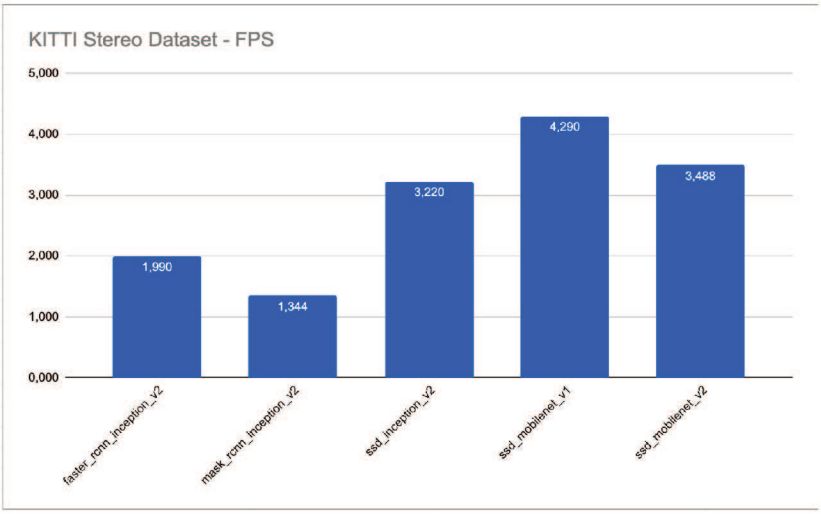


Figura 44 – Resultados do *benchmark* - Frames por Segundo - KITTI Stereo Vision

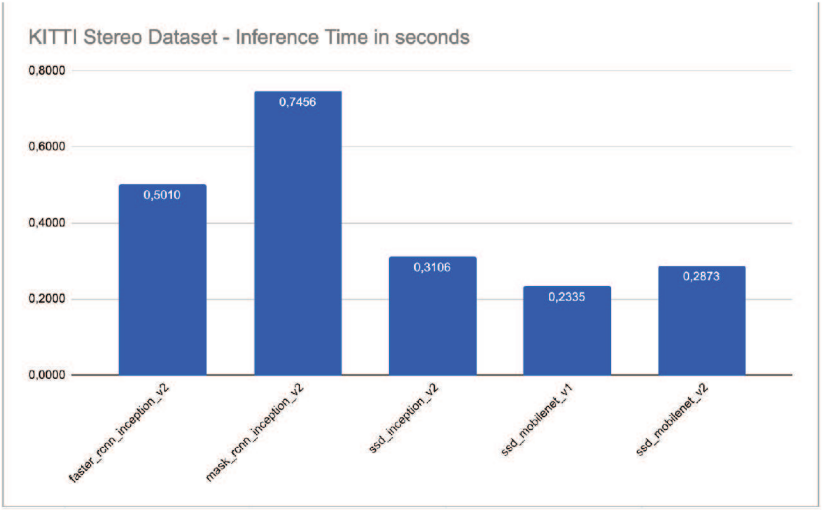


Figura 45 – Resultados do *benchmark* - Tempo médio de inferência - KITTI Stereo Vision

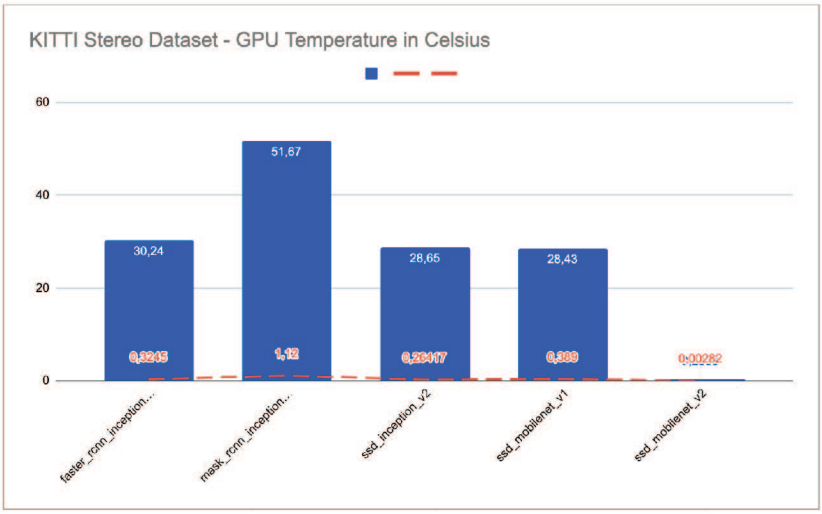


Figura 46 – Resultados do *benchmark* - Média e desvio padrão de temperatura da GPU - KITTI Stereo Vision

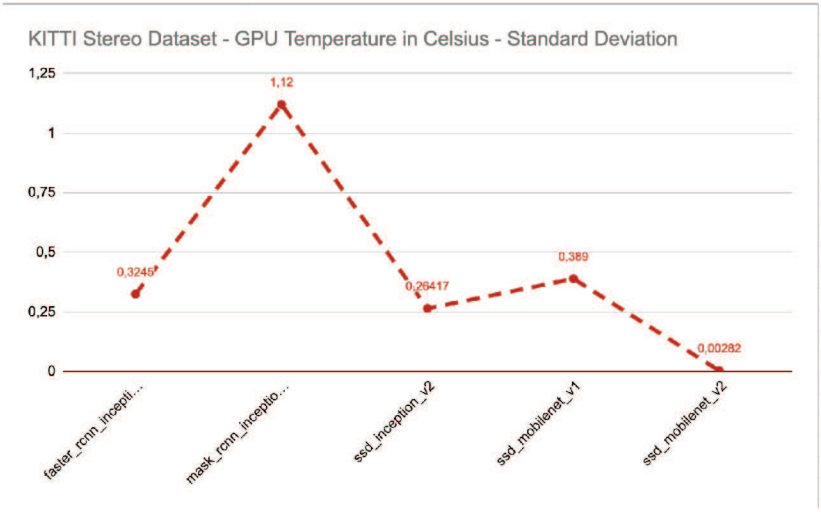


Figura 47 – Resultados do *benchmark* - Desvio padrão de temperatura da GPU - KITTI Stereo Vision

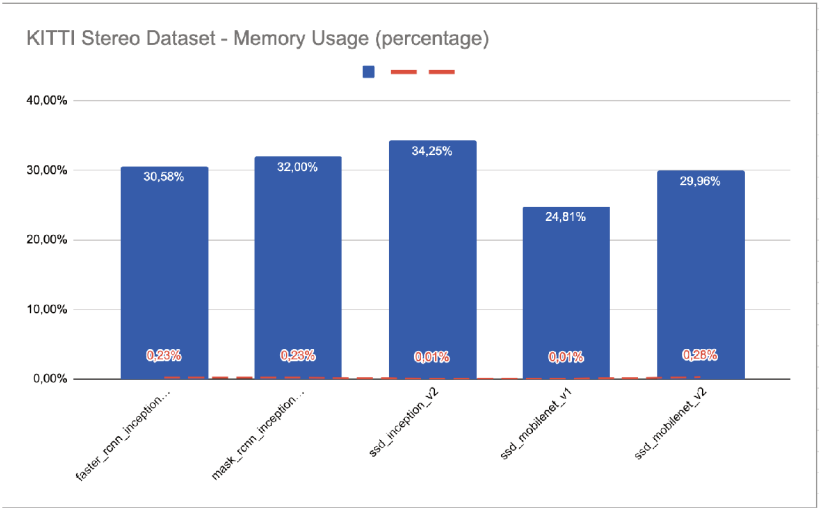


Figura 48 – Resultados do *benchmark* - Média e desvio padrão de memória da GPU - KITTI Stereo Vision

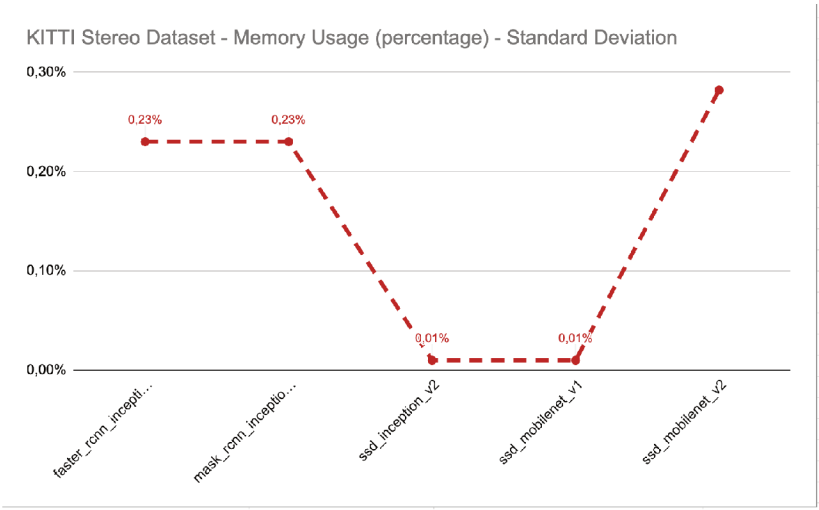


Figura 49 – Resultados do *benchmark* - Desvio padrão de memória da GPU - KITTI Stereo Vision

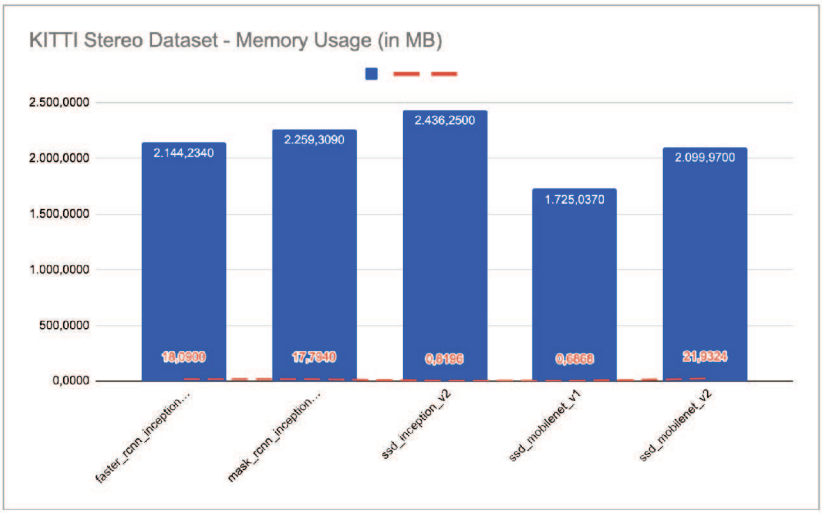


Figura 50 – Resultados do *benchmark* - Média e desvio padrão de memória da GPU - KITTI Stereo Vision

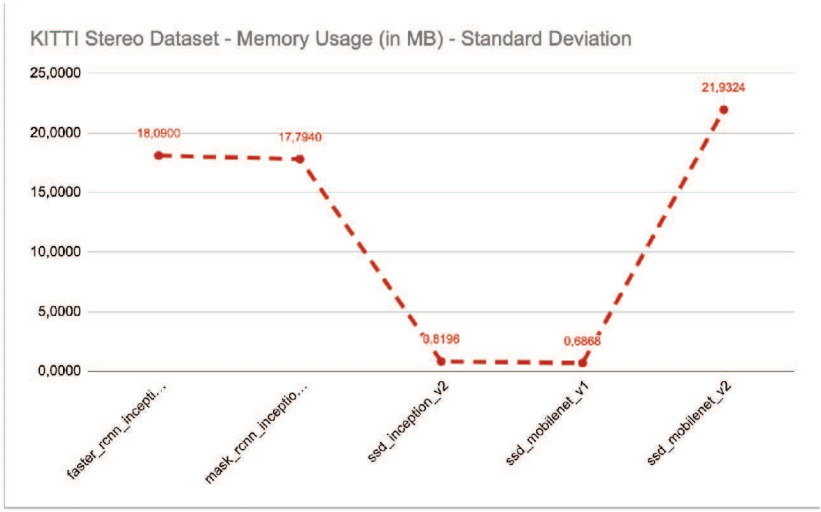


Figura 51 – Resultados do *benchmark* - Desvio padrão de memória da GPU - KITTI Stereo Vision

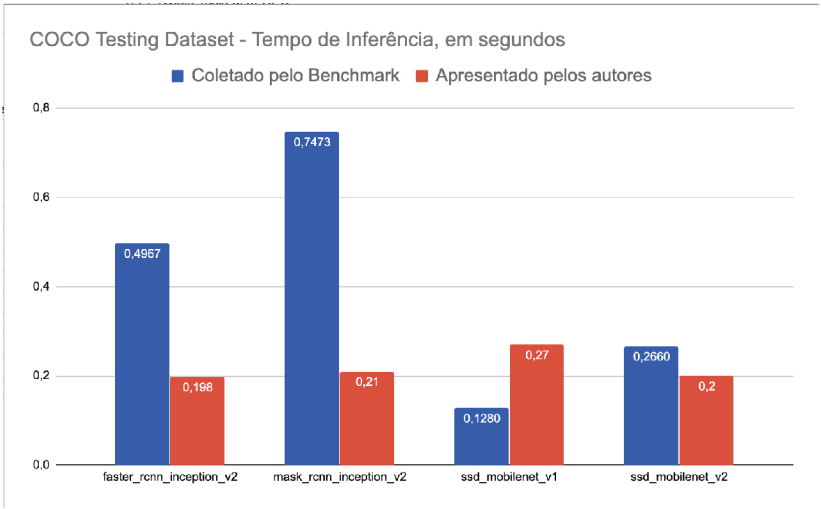


Figura 52 – Comparativo de resultados apresentados por autores das arquiteturas

Os gráficos apresentados trazem informações relevantes sobre os modelos testados. É possível perceber que a implementação SSD v1 sobre a rede base *MobileNet* é o modelo com melhor desempenho sobre a Jetson TX2.

É interessante notar que, como apresentado na literatura, o uso de memória, com exceção de modelos baseados em *MobileNet* os quais possuem foco em redução de uso de recursos, a necessidade de uso de memória para execução dos modelos não difere muito entre eles.

Como referência, A Figura 52 apresenta um comparativo entre os tempos de inferência coletados sobre o conjunto Microsoft COCO utilizando o *benchmark* e resultados coletados pelos autores das respectivas arquiteturas.

Os autores do modelo *Faster R-CNN* utilizam a plataforma NVIDIA Tesla K40 para inferência. Os autores do trabalho *Mask R-CNN* realizam os experimentos sobre a GPU NVIDIA Tesla M40. Por último, ambos os modelos sobre a rede MobileNets são testados sobre o dispositivo Google Pixel 1. A Tabela 6 apresenta a relação entre consumo energético utilizado pela plataforma Jetson TX2 se comparado aos sistemas testados pelos autores das arquiteturas.

Por fim, o pior tempo médio de inferência e quantidade de frames por segundo deve-se ao modelo *Mask R-CNN*. Dada a necessidade de

GPU	Consumo Energético
NVIDIA Jetson TX2 (FRANKLIN, 2017)	7.5/15W
NVIDIA Tesla K40 (CNET, a)	235W
NVIDIA Tesla M40 (CNET, b)	250W
Google Pixel 1	Fonte de carregamento de 18W

Tabela 6 – Sistemas utilizados pelos autores para metrificação

execução de ambos os ramos apresentados pelos autores do modelo (detecção e segmentação), tal modelo possui seu desempenho afetado. A Tabela 7 mostra um comparativo das métricas entre os modelos para ambos os conjuntos de dados. É possível traduzir as legendas do seguinte modo:

I Quadros por Segundo (FPS);

II Tempo de inferência, en segundos;

III Temperatura da GPU, em graus célsius;

IV Uso de memória total;

Modelo	I	II	III	IV
Faster R-CNN - COCO	2,013	0,4967	31,67	30,65%
Mask R-CNN - COCO	1,338	0,7473	36,86	30,54%
SSD Inception V2 - COCO	3,370	0,2960	28,20	32,15%
SSD Mobilenet V1 - COCO	7,806	0,1280	31,81	24,89%
SSD Mobilenet V2 - COCO	3,750	0,2660	29,54	28,24%
Faster R-CNN - KITTI	1,990	0,5010	30,24	30,58%
Mask R-CNN - KITTI	1,344	0,7456	51,67	32,00%
SSD Inception V2 - KITTI	3,220	0,3106	28,65	34,25%
SSD Mobilenet V1 - KITTI	4,290	0,2335	28,43	24,81%
SSD Mobilenet V2 - KITTI	3,488	0,2873	28,43	29,96%

Tabela 7 – Comparativo dos resultados obtidos

5 CONCLUSÕES E TRABALHOS FUTUROS

Em conclusão, o presente trabalho revisa a literatura e realiza o levantamento do estado da arte sobre o campo de *deep learning*, dando foco em algoritmos de redes neurais convolucionais e detecção de objetos.

Além disso, foi realizada a implementação de um *benchmark* comparativo utilizando o *framework* TensorFlow e as otimização da biblioteca cuDNN, com foco em desempenho de modelos alinhados ao estado da arte, sobre a plataforma NVIDIA Jetson TX2, sendo extensível para outros sistemas embarcados existentes na literatura.

Uma análise comparativa de cinco modelos de CNN foi realizada, sobre dois conjuntos de dados amplamente utilizados na comunidade acadêmica. O estudo permite extrair métricas e auxiliar no processo de tomada de decisão sobre quais componentes são otimizados para a plataforma testada, dados os componentes de *hardware* nela presentes.

Através dos resultados apresentados, é possível realizar um comparativo relacionado ao consumo energético da plataforma apresentada e sistemas especializados não embarcados, como mostrado na seção 4.6. Embora a performance durante a fase de inferência possa ser inferior a aceleradores tradicionais, é notável a proximidade da capacidade de processamento de sistemas embarcados, dadas suas limitações.

Como trabalhos futuros, dado o levantamento realizado durante a revisão da literatura, a aplicação da biblioteca TensorRT sobre a coleta de dados, com otimizações perante aos sistemas da fabricante NVIDIA, é esperada.

Além disso, a aplicação do *benchmark* sobre modelos distintos, conjuntos de dados alternativos e com características distintas as presentes nos dados explorados (como dados de vídeo e cenários diversos) e plataformas alternativas são os focos principais para o futuro.

REFERÊNCIAS

ABADI, M. et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2015. <https://www.tensorflow.org/>.

BENENSON, R. et al. Ten years of pedestrian detection, what have we learned? Springer International Publishing, Cham, p. 613–627, 2014.

BOKOVOY, A.; MURAVYEV, K.; YAKOVLEV, K. Real-time vision-based depth reconstruction with nvidia jetson. 07 2019.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

CHOI, E. et al. Generating multi-label discrete patient records using generative adversarial networks. p. 286–305, 2017.

CNET. *NVIDIA Tesla K40 GPU computing processor Specs*. https://www.cnet.com/products/nvidia-tesla-k40-gpu-computing-processor-tesla-k40-12-gb/. Acesso em: 26 nov. 2019.

CNET. *NVIDIA Tesla M40 GPU computing processor Specs*. https://www.cnet.com/products/nvidia-tesla-m40-gpu-computing-processor-tesla-m40-24-gb/. Acesso em: 26 nov. 2019.

CUI, N. Applying gradient descent in convolutional neural networks. *Journal of Physics: Conference Series*, v. 1004, p. 012027, 04 2018.

Daily, M. et al. Self-driving cars. *Computer*, v. 50, n. 12, p. 18–23, December 2017.

DOLLÁR, P. et al. Pedestrian detection: An evaluation of the state of the art. *PAMI*, v. 34, 2012.

Du, X. et al. Overview of deep learning. In: *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*. [S.l.: s.n.], 2016. p. 159–164. ISSN null.

FAUSETT, L.; FAUSETT, L. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. [S.l.]: Prentice-Hall, 1994. (Prentice-Hall international editions). ISBN 9780133341867.

FEI-FEI, L.; KARPATHY, A.; JOHNSON, J. *CS231n Convolutional Neural Networks for Visual Recognition*. http://cs231n.github.io/. Acesso em: 20 out. 2019.

FISCHER, P. et al. Flownet: Learning optical flow with convolutional networks. *2015 IEEE International Conference on Computer Vision (ICCV)*, p. 2758–2766, 2015.

FRANKLIN, D. *JetPack 2.3 with TensorRT Doubles Jetson TX1 Deep Learning Inference*. 2016. https://devblogs.nvidia.com/jetpack-doubles-jetson-tx1-deep-learning-inference/. Acesso em: 20 out. 2019.

FRANKLIN, D. *JetPack doubles Jetson inference performance*. 2017. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/. Acesso em: 20 out. 2019.

FRANKLIN, D. *Jetson nano AI computing*. 2019. https://devblogs.nvidia.com/jetson-nano-ai-computing/. Acesso em: 20 out. 2019.

FUKUSHIMA, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, v. 36, n. 4, p. 193–202, Apr 1980. ISSN 1432-0770. <https://doi.org/10.1007/BF00344251>.

GEIGER, A.; ZIEGLER, J.; STILLER, C. Stereoscan: Dense 3d reconstruction in real-time. In: *Intelligent Vehicles Symposium (IV)*. [S.l.: s.n.], 2011.

GIRSHICK, R. Fast r-cnn. In: *International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2015.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. <http://proceedings.mlr.press/v9/glorot10a.html>.

GROUP SAFETY PUBLICATION. INTERNATIONAL STANDARD. *SAFETY OF LASER PRODUCTS. Part 1: Equipment classification, requirements and user's guide*. 2001. <http://www.microasu.com/download/jqlaser/certificate/IEC_60825_1.pdf>. Acessado em 18 fev. 2015.

Han, Y.; Oruklu, E. Traffic sign recognition based on the nvidia jetson tx1 embedded system using convolutional neural networks. In: *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*. [S.l.: s.n.], 2017. p. 184–187.

He, K. et al. Mask r-cnn. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. [S.l.: s.n.], 2017. p. 2980–2988.

HOWARD, A. G. et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. <http://arxiv.org/abs/1704.04861>.

HUBEL, D. H.; WIESEL, T. N. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, v. 148, p. 574–591, 1959.

ILG, E. et al. Flownet 2.0: Evolution of optical flow estimation with deep networks. p. 1647–1655, July 2017. ISSN 1063-6919.

IVAKHNENKO, A. G.; LAPA, V. G. *Cybernetic Predicting Devices*. [S.l.: s.n.], 1965.

JANOCHA, K.; CZARNECKI, W. On loss functions for deep neural networks in classification. *Schedae Informaticae*, v. 25, 02 2017.

LI, J. et al. Scale-aware fast r-cnn for pedestrian detection. *IEEE Transactions on Multimedia*, v. 20, p. 985–996, April 2018. ISSN 1520-9210.

Li, J. et al. An experimental study on deep learning based on different hardware configurations. In: *2017 International Conference on Networking, Architecture, and Storage (NAS)*. [S.l.: s.n.], 2017. p. 1–6.

Li, Q.; Xiao, Q.; Liang, Y. Enabling high performance deep learning networks on embedded systems. In: *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society*. [S.l.: s.n.], 2017. p. 8405–8410.

LIN, T.-Y. et al. Focal loss for dense object detection. *2017 IEEE International Conference on Computer Vision (ICCV)*, p. 2999–3007, 2017.

LIN, T.-Y. et al. Microsoft coco: Common objects in context. Springer International Publishing, p. 740–755, 2014.

LIU, W. et al. Ssd: Single shot multibox detector. Springer International Publishing, p. 21–37, 2016.

M., E. et al. *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. Http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

MENZE, M.; GEIGER, A. Object scene flow for autonomous vehicles. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015.

NESTEROV, Y. E. A method for solving the convex programming problem with convergence rate $o(1/k^2)$.*Dokl. Akad. Nauk SSSR, v.* $269, p.543 - -547, 1983$. <*https* : *//ci.nii.ac.jp/naid*/10029946121/*en/*>.

NICKOLLS, J. et al. Scalable parallel programming with cuda. *Queue*, ACM, New York, NY, USA, v. 6, n. 2, p. 40–53, mar. 2008. ISSN 1542-7730. <http://doi.acm.org/10.1145/1365490.1365500>.

Redmon, J. et al. You only look once: Unified, real-time object detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2016. p. 779–788.

REDMON, J. et al. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015. <http://arxiv.org/abs/1506.02640>.

REDMON, J.; FARHADI, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. <http://arxiv.org/abs/1804.02767>.

REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, IEEE Computer Society, Washington, DC, USA, v. 39, n. 6, p. 1137–1149, jun. 2017. ISSN 0162-8828. <https://doi.org/10.1109/TPAMI.2016.2577031>.

ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

Sandler, M. et al. Mobilenetv2: Inverted residuals and linear bottlenecks. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. [S.l.: s.n.], 2018. p. 4510–4520.

Sawada, J. et al. Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications. In: *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. [S.l.: s.n.], 2016. p. 130–141.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, v. 61, p. 85 – 117, 2015. ISSN 0893-6080. <http://www.sciencedirect.com/science/article/pii/S0893608014002135>.

SERMANET, P. et al. Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations (ICLR) (Banff)*, 12 2013.

STANDARD, A. N. *American National Standard for Safe use of Lasers Outdoors*. Orlando, FL, 2005. <https://www.lia.org/PDF/Z136_6_s.pdf>. Acessado em 18 fev. 2015.

SUTSKEVER, I. et al. On the importance of initialization and momentum in deep learning. *30th International Conference on Machine Learning, ICML 2013*, p. 1139–1147, 01 2013.

Sze, V. Designing hardware for machine learning: The important role played by circuit designers. *IEEE Solid-State Circuits Magazine*, v. 9, n. 4, p. 46–54, Fall 2017.

SZEGEDY, C. et al. Rethinking the inception architecture for computer vision. In: . [S.l.: s.n.], 2016.

Szegedy, C. et al. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. [S.l.: s.n.], 2015. p. 1–9.

TOSUN, E.; AYDIN, K.; BILGILI, M. Comparison of linear regression and artificial neural network model of a diesel engine fueled with biodiesel-alcohol mixtures. *Alexandria Engineering Journal*, v. 55, n. 4, p. 3081 – 3089, 2016. ISSN 1110-0168. <http://www.sciencedirect.com/science/article/pii/S1110016816302228>.

WIDROW, B. *Adaptive "adaline"neuron using chemical "memistors."*. [s.n.], 1960. <https://books.google.com.jm/books?id=Yc4EAAAAIAAJ>.

XIAO, Z. et al. Dense scene flow based coarse-to-fine rigid moving object detection for autonomous vehicle. *IEEE Access*, v. 5, p. 23492–23501, 2017. ISSN 2169-3536.

YAO, J.; NORTH, P.; TAN, C. L. Guidelines for financial forecasting with neural networks. 08 2002.

Zhang, Y. et al. The implementation of cnn-based object detector on arm embedded platforms. In: *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*. [S.l.: s.n.], 2018. p. 379–382.

ANEXO A – Donation Letter

March 6, 2018

Aldo von Wangenheim
Federal University of Santa Catarina - UFSC
Rua Ana Maria Nunes, 275
Res. Araquãs, Casa 20
Córrego Grande
Florianópolis, SC 88.037-020 Brazil
TaxID 660.566.679-87

Re: Equipment Donations for Research Purposes

Dear Aldo,

NVIDIA's GPU Grant Program recently donated (1) Jetson TX2 Developer Kit for your research efforts. This donation is an unrestricted gift to support your research efforts and comes at no charge to you. NVIDIA has paid all shipping.

Best Regards,

Chandra Cheij
Academic Programs Manager

APÊNDICE A – Relatório Técnico

Benchmarking deep learning models on Jetson TX2

Lucas Pedro Bordignon
Aldo Von Wangenheim

October 31, 2019

Contents

1 Acknowledgements **3**

2 Introduction **4**

3 Objectives **5**
3.1 General Objectives . . . 5
3.2 Specific Objectives . . . 5
3.2.1 Methodology . . . 6

4 Fundamentals **7**
4.1 Artificial Intelligence . . . 7
4.2 Artificial Neural Networks . . . 7
4.3 Activation Functions . . . 9
4.4 Loss Functions . . . 10
4.5 Optimization . . . 11
4.6 Deep Learning . . . 12
4.7 Convolutional Neural Networks . . . 13

5 State of the art **16**
5.1 A Hybrid Gomoku Deep Learning Artificial Intelligence - (57) . . . 17
5.2 Application of Computer Vision and Deep Learning in Breast Cancer Assisted Diagnosis - (58) . . . 17
5.3 GeoAI 2017 workshop report: the 1st ACM SIGSPATIAL International Workshop on GeoAI: @AI and Deep Learning for Geographic Knowledge Discovery - (37) . . . 17
5.4 Visual landmark sequence-based indoor localization - (33) . . . 18
5.5 Deep Learning in the Field of Art - (59) . . . 18
5.6 Review of State-of-the-Art in Deep Learning Artificial Intelligence - (47) . . . 19
5.7 Driver information system: a combination of augmented reality, deep learning and vehicular Ad-hoc networks - (9) . . . 19
5.8 Road vehicle detection and classification based on Deep Neural Network - (61) . . . 19
5.9 Pedestrian Detection Algorithm Based on the Improved SSD - (34) . . . 19

6 Embedded Platforms **20**
6.1 Jetson TX2 . . . 20
6.2 Getting Started . . . 25
6.2.1 Automatic Setup . . . 28
6.2.2 Manual Setup . . . 28
6.3 Troubleshooting . . . 30
6.3.1 tensorflow.python.framework.errors_impl.InternalError: Failed to create session . . . 30

7 Benchmark **31**
7.1 Existing approaches . . . 32
7.2 Models . . . 34
7.2.1 Faster R-CNN . . . 34
7.2.2 Mask R-CNN . . . 35
7.2.3 Single Shot Multibox Detector (SSD) . . . 36
7.2.4 Inception V2 . . . 37
7.2.5 MobileNets . . . 38
7.3 Datasets . . . 40
7.3.1 Microsoft COCO . . . 40
7.3.2 KITTI Stereo Vision . . . 40
7.4 Implementation . . . 41
7.5 Experiment Results . . . 42
7.6 Microsoft COCO . . . 43
7.7 KITTI Stereo Vision . . . 51
7.8 Sample Results . . . 59
7.9 Results . . . 64

8 Conclusion and Future Work **65**

9 Attachments **70**
9.1 Donation Letter . . . 71

Chapter 1

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Jetson TX2 platform used for this research. The resource donated to the INCoD (Instituto Nacional para Convergência Digital) research group was of huge importance to the research project presented through this report.

Chapter 2

Introduction

Artificial intelligence has evolved from the last years towards to large diversity of areas, such as image recognition, audio recognition and data recommendation. To allow these areas to grow, artificial intelligence techniques must have great accuracy and power.

Systems that involve human and computer interaction has become even more common and inside our daily lives. These interactions occur whether virtually, with message exchanging for example, or through physical machines that help human beings in common and most of the time repetitive tasks. One of the examples of it is the field of self-driving cars (16).

Focusing only on the self-driving cars scopes, models that use passive navigation systems in exchange of active sensors, as LIDAR, are preferred. They allow cheaper and smaller production-ready setups for inference, lower energy to execute and are considered less-intrusive approaches if compared to active sensors (12), (49), (7).

With the advances over the field, more specifically over deep learning models as convolutional neural networks, new challenges have emerged as a counterpart. One of the main areas that allow the field to evolve is related to embedded systems and hardware accelerators. Many manufacturers have started working on high-performance computers embedded in single-board platforms (32).

There are a few reasons and motivations to create such platforms. In a scenario where the internet of things is evolving and getting closer to final users, the latency of responses on environments as these must be reduced to the minimum, in order the work with intelligence on the devices of the future. Moreover, privacy is still a concern, as in traditional models executed at cloud providers instead of embedded platforms the information must navigate through internet channels to get the expected results. Working with devices as closer to the edge of the networks allow researchers to mitigate such kind of problem (28).

Nowadays, researchers can find platforms to execute their models, both for training and testing, with large manufacturers and semiconductor design companies as ARM (60) and NVIDIA Corporation (20). However, the capabilities of embedded boards are distant for server-ready accelerators, as a result of optimizations in areas as latency, energy consumption and bandwidth, making the decision to use such boards to involve a large number of factors (22).

To help on the decision process, the current work introduces one of the existing embedded boards family, the Jetson TX2, and has a focus on collecting data and information over the performance of the system under deep learning models workload.

Chapter 3

Objectives

Recent advances towards the area of artificial intelligence have been made by a number of researchers around the world. More specifically, projects that explore new changes and propose new models for modern real world problems have expanded. One of the main artificial intelligence techniques evolved rapidly during the past years are called convolutional neural networks, responsible and optimized for object segmentation and detection over image data (10).

In order to work with the new solutions proposed by the academia, a number of hardware accelerators have been developed with the focus in improving solutions performance and fulfill real-time systems and data privacy necessities. However, embedded systems as them have specific space, energy and many other limitations (50).

To follow the latest advances in the field of artificial intelligence embedded accelerators, the present work has focused on collecting data about one of the production-ready embedded platforms existing on the world, through bench-marking the components present on-board, helping researchers on the decision making process to decide whether an embedded platform has the right capabilities for their needs.

3.1 General Objectives

The objective of the current work is to present the NVIDIA's Jetson TX2 platform, exposing instructions for usage and prototyping. As an application of the knowledge presented, the work implements a convolutional neural networks benchmark and collect information related to the performance of each component of the platform.

3.2 Specific Objectives

- Explore the recent state of the art of the areas of convolutional neural networks and computer vision applications;
- Expose the Jetson TX2 platform, from NVIDIA Corporation, presenting the components and capabilities of the system;
- Guide developers on how to implement and prototype systems under the Jetson TX2 platform, presenting possible issues that they may find on the path;
- Implement a convolutional neural networks benchmark over the Jetson TX2 platform;

3.2.1 Methodology

With the given objectives, Chapter 4 presents a study of the history of artificial intelligence and how research got to the point where it is nowadays, passing through artificial neural networks, how inference works and how networks are trained and specializing to convolutional neural networks, the focus of the present work.

Chapter 5 presents the state of the art for real-world applications present in the literature, exposing the usage of deep learning and convolutional neural networks in fields such as arts, logistics and road detection for autonomous vehicles.

Chapter 6 brings a summary of existing approaches for single-board computers and hardware accelerators with a focus in artificial intelligence, presenting one in details, the Jetson TX2 board. All the specifications and a guide for developers are presented in this chapter.

Chapter 7 works with the implementation of a benchmark on the Jetson TX2 embedded platform, to collect metrics on which components of the board are most used on real-world scenarios. The first block of the chapter discusses existing benchmarks on the literature for the board.

After the initial discussion, a set of 5 deep learning models were chosen, following the state of the art for object recognition, to test the board under the most likely real-world scenarios as possible. Metrics as inference time, frames per second and GPU temperature were collected during the experiments, written in more details at the chapter.

Chapter 4

Fundamentals

To start with, it is first necessary to define what exactly artificial intelligence is and which areas can have their own problems solved through these techniques.

4.1 Artificial Intelligence

The concept of artificial intelligence is in vogue in the first decades of the 21st century. With the advances in different directions of the area of computer science, computers are now able to execute tasks that were not possible in the past and in performances that are acceptable for modern needs. However, the initial definitions of the field have been crafted by researchers from the middle of the past century, from the decades of 1950 and 1960 on.

Alan Turing, considered by most the father of artificial intelligence and even computer science as a whole, wrote one of the first articles for the field, called by him, "Computing Intelligence", in 1950 (55). The work exposes a vision of what is computing intelligence and discuss which parts of the field must be researched to build trustful intelligence on such machines.

The given work even brings the concept of the Turing's Test for machines. In summary, the test consists of two human beings and a machine. The machine and one of the individuals only type with a teleprinter for the other one, called the interrogator. If the machine can make the interrogator think that it is a human, it can be called as "intelligent".

After Alan Turing's work, the fields slowly started to grow and a few branches started to appear, as natural language (11), human logic (39) and, most important for this report, the human neural network representation, mainly with the appearance of the perceptron (43).

4.2 Artificial Neural Networks

The main idea of the perceptron introduced by Rosenblatt, in 1958, consists of a representation of the biological neurological system that exists on a large part of animals, such as humans.

The neurological system of such animals behaviors based on given stimuli on an input level. Consequently, after receiving the stimuli, neurons are fired based on which task they are designed for, allowing the information to be passed through a deep network of them until the final receptor.

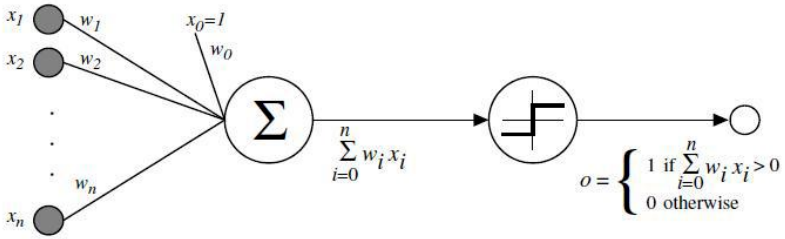


Figure 4.1: An abstraction of a perceptron network

Figure 4.1 shows an abstraction of a perceptron, representing the concept presented by Rosenblatt. The structure exposed has the following steps:

- Receives binary inputs from external stimuli or previous neurons;
- Multiplies each input by a continuous valued weight, representing the intensity of that neuron being triggered on the given input;
- Threshold the output for either a 0 or 1, allowing the output to be a binary value representing, as biologically, whether a neuron triggers or not for that stimuli

For each neuron of the layer, we have a specific weight, which is a number that is multiplied by a given input entry and defines the intensity of that node on the given input. It is possible to have more than a single weight on an artificial neural network as the concept allows for multi-layer perceptrons.

After the discovery and first appearance of the perceptron, Bernard Widrow and Marcian Hoff developed in 1960 two artificial neural networks based on the concept, being the first ones applied to a real-world problem. It is called ADALINE, which stands for *adaptive linear neuron*, and is a single layer network with multiple inputs, yielding a single output of it (56).

An ADALINE layer can be mathematically defined as following:

$$output = \sum_{i=0}^{n} w_i x_i + \theta$$

- n is the number of inputs of the layer;
- x is the current value of the input on index i;
- w is the weight value of the network on index i;
- θ is some constant value;

It was implemented by Widrow and the core concept on the given work is around the memistor, which is a species of transistor containing memory and allowing them to be trained and keep a memory for weeks, based on the author's report.

In 1965, Artificial Neural Networks trained by the Group Method of Data Handling (GMDH), in which Valentin Grigorevich Lapa and A. G. Ivakhnenko worked on can be considered the first deep learning system of a feed-forward multilayer perceptron on academia (24).

Units from GMDH networks may have polynomial activation functions, implementing Kolmogorov-Gabor polynomials, even more general than common activation functions for today's standards. Given a training dataset, layers are grown and trained by regression analysis.

4.3 Activation Functions

When working with neural networks, the individual computation entity in which we can retrieve and extract information from the data collected is called the neuron. A neuron is responsible by reacting or not given input stimuli.

To abstract the idea of a neuron mathematically, we separate the variables as follows:

- x_i is one sample from the input array;
- w_i is the weight trained for that index from the input array;
- b is the bias value for the given neuron;
- f is the activation function;

With the given variables, we can apply a first-degree function and we sum up every result of it to generate the value of the individual neuron. Worth notice that the individual *weights* for each input index are values that can be trained to better represent the desired output of the network (explored at "Optimization" section). When starting a fresh new model, usually, weights and bias values are set to random values (19).

Based on the result of the sum of each execution, it is needed to apply a non-linear function, responsible to generate a threshold for input values and decide whether the neuron is "active" for the input given and at which intensity. These non-linear functions applied are called activation functions. Figure 4.2 represents the schema of an individual neuron.

Currently, there are a couple of activation functions existing on the literature. A few of them are presented bellow. Each individual function produces a more precisely representation of whether the neuron is active or not.

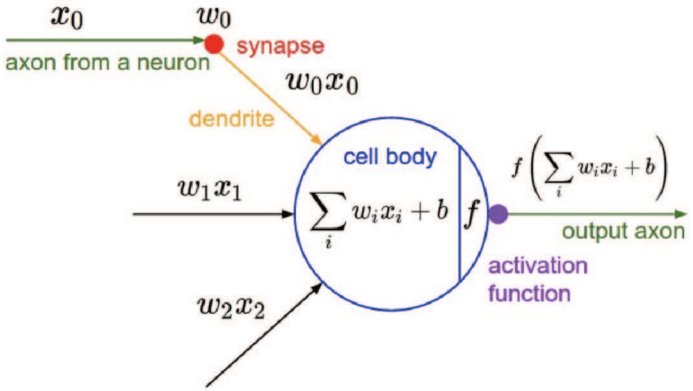


Figure 4.2: An illustration of an individual neuron on an artificial neural network

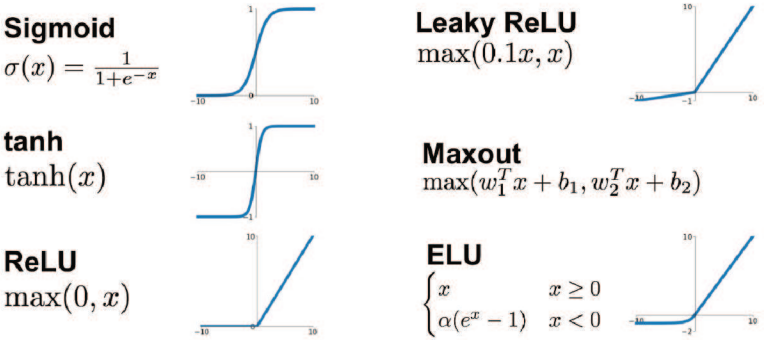


Figure 4.3: A map of a variety of activation functions from literature

4.4 Loss Functions

As seen at the activation functions section, for each individual neuron we have aggregation functions that behave as:

$$neuron\_value = (\sum_{i=0}^{n} w_i x_i) + b$$

Where w and b are set as random values when we first initialize a network. In order to our neurons to converge to a result that gives a satisfactory classification or detection precision they cannot be static but adapt themselves with input data.

This step is called as training the model, making the weights and biases as precise as possible to fit the data that is going to be passed to the network built. During the phase of training a network, as each individual entity is considered as a mathematical function, optimization techniques must be used to better find the optimal parameters.

To define an optimization algorithm, first, it is needed to define a loss function to measure how far the model is from answering the right choice. Figure 4.4 displays the description of a couple of existing functions on literature. Loss functions take, usually, as inputs the predicted results, the right result (during training phase) and a "safe" threshold.

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y} - \mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y} - \mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y} - \sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y} - \sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)} - \mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2} - \hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
$\log^2$	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$

Figure 4.4: A schema of different loss functions (26)

4.5 Optimization

Given that we already have loss functions to express "how far from the perfect prediction the model is", we are now able to execute optimization algorithms to find the best weights that will result in a smaller loss as possible.

There are a few general-purpose optimization algorithms inherited from the field of mathematics that are used to train neural networks. In this case, we are going to work with a minimization problem, where we try to reduce the value of the output from loss function.

One method to try to find the global minimum of a loss function for a given model is using derivatives. Considering one-dimensional functions, a derivative of it at a given point informs us the vector of the rate of change for that function on that point. Knowing that we can have the direction of the slope and the magnitude of it just calculating the derivative.

The following expression represents a derivative for a one-dimensional function with respect to its input:

$$\frac{\partial f(x)}{\partial x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

In the case of multiple inputs, as we have from the inputs from neural networks, the derivatives are called *partial derivatives* and the vector of directions that is the result of the calculus is called a gradient.

Knowing that we are able to use individual gradients to understand which is the best direction to reach an optimal result, we now must update the parameters of our function to be able to get the expected result. The most common method for updating the weights is called gradient descent.

The concept of gradient descent is based on updating the parameters at each iteration of image inputs. During the training phase, is defined a parameter, called *step_size* (commonly called learning rate), which is the intensity of the update for the given weights, as shows Figure 4.5. A step of the default gradient descent algorithm can be described as follows:

$$gradient = derivative(l(x), i, w)$$

$$weights = weights - (step\_size \times gradient)$$

Where $l(x)$ is the specified loss function for the model, i is the input vector, w is the weights vector and $derivative(x, y, z)$ is a mathematical function that finds the derivative of the loss function at the given input point.

Worth notice the subtraction of the gradient, given the step size. It is used in order to update the weights in the right direction, as gradients positive and negative signs must be inverted. Many other implementations exist in the literature, with different optimizations over the concept presented above.

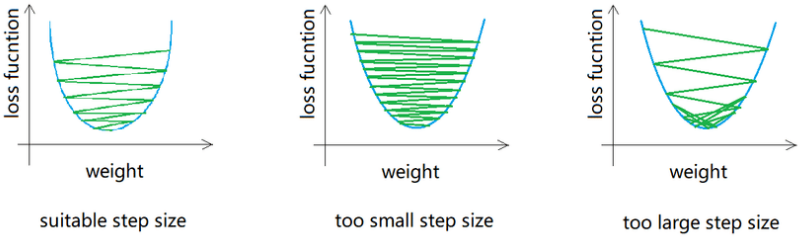


Figure 4.5: A comparative of step sizes on gradient descent (15)

4.6 Deep Learning

After a few years has evolved and some new statistical and optimization techniques evolved in computer science, deep artificial neural networks started to become more popular and solve a new range of problems in pattern

recognition and machine learning (46).

In 1950, Hubel and Wiesel published a work in which experiments over mammals neurological system have been conducted. Most part of the experiments are executed inducing electrical stimuli and collecting brain activity over the duration of it. While the stimuli have been triggered over animals neurological system, visual information was shown to them in order to recognize a pattern. After some time, the same authors received a Nobel prize over the contributions made by them on the given area (**?**).

The authors found and classified the observations roughly in two categories:

- **Simple Cells**, responsible by the detection of image frontiers;
- **Complex Cells**, responsible by the detection of light and spacial movements;

After the discovery, in 1979, one of the first models to abstract the idea shown by Hubel and Wiesel work and translate it in terms of what a machine can understand is called Neorecognition, described by Kunihiko Fukushima in 1980 (18).

The model described can be considered a deep model based on the architecture used by the author. Using the concepts of simple and complex cells, the neural network proposes a visual pattern recognition mechanism. It works following the idea of "learning without a teacher", acquiring the ability to recognize patterns based on geometrical similarity of objects shapes.

Considered a deep neural network, it consists of an input layer (photo-receptor array) followed by connections of a number of modular structures, each one being composed of two layers of cells connected in a cascade. The first layer of each module consists of what the author calls "S-cells", representing the idea of border detection of simple cells (or lower-order hypercomplex cells), and the second layer consists of "C-cells", which abstracts the concept of complex cells (or higher-order hypercomplex cells).

Figure 4.6 presents an overview of the model described above. The Neorecognition can be considered one of the initial versions of convolutional neural networks ever seen and described in the literature.

4.7 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are similar to Artificial Neural Networks (ANNs). They are made up of neurons with weights and biases. Each neuron receives image inputs performs a product and optionally follows it with a non-linearity. The network still represents a simple score function. Given raw image pixels on one end of the network to class scores at the other. CNNs still work with loss functions (e.g. SVM/Softmax) on the last layer (2).

One of the major changes on-premises if compared to traditional networks is the enforcement of inputs to be images, allowing us to imply properties into the architecture.

Regular ANNs are composed by fully-connected layers from end to end, with two independent layers, which are the input layer and output predictions. Such architecture performs really well for linear regression problems and well-structured data without contextual remaining.

When working with images, considering RGB based images, the network must process inputs with the size of $image\_width \times image\_height \times 3$ (channels from RGB images). If using a traditional approach, the number of weights needed to complete all the connections are going to be large and the network will not be optimized to work with contextual information.

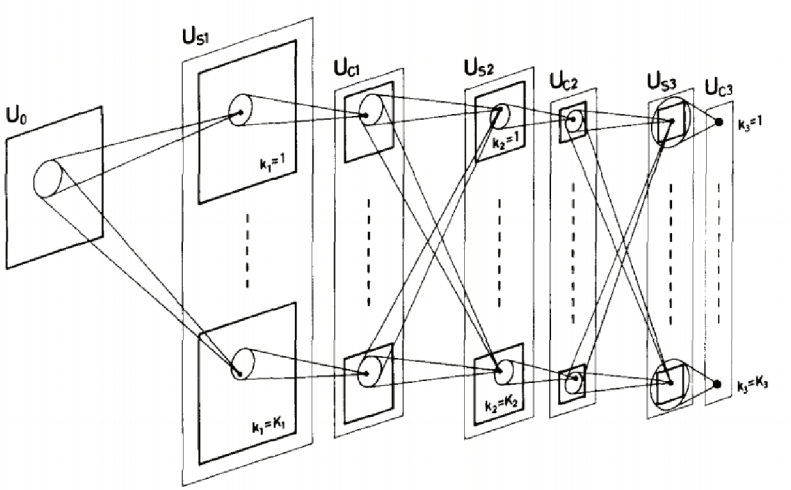


Figure 4.6: An illustration of the Neorecognition network structure

CNNs take advantage of the fact that inputs consist of images and they can adapt the architecture in a more sensible way. Unlike a regular one, layers of a convolutional network have neurons arranged in 3 dimensions, one for width, one for height and another for color model dimensions.

However, even if the networks have specific characteristics to offer, as a general rule when working with object detection, they still have a fully-connected layer at the end of the inference step. This allows us to generate predictions and likelihoods for the classes defined. Figure 4.7 presents the base architecture for CNNs.

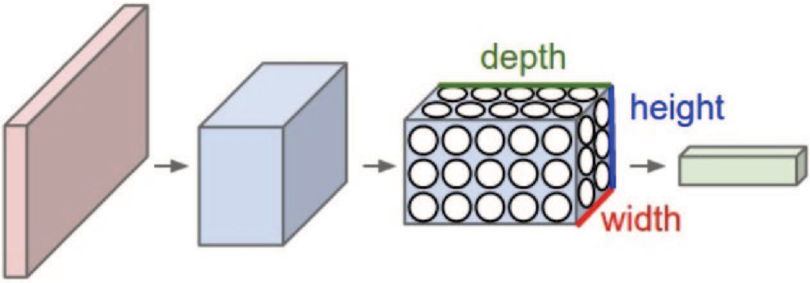


Figure 4.7: Schema of a generic Convolutional Neural Network

Chapter 5

State of the art

To bring use cases, search over academic digital libraries were made. Some of the selected libraries are *Springer*, *ACM Digital Library* and *IEEE Xplore*.

During the survey, every type of application with focus on deep learning were considered and brought as the result. For each individual digital library is exposed the specific search query used to find the resources.

The phrases with more focus used on the research were:

- Deep learning
- Computer Vision
- Artificial Intelligence
- Application

Moreover, the search considered articles and reports from the period between 2009 and 2019. After executing the search on all the selected digital libraries, X results have returned, being analyzed and studied afterwards. This step had the focus on filtering results that does not fit the premises defined at first place.

The results grouped by platform are the following:

- **Springer**: 39,163 results;
- **ACM Digitial Library**: 1,703 results;
- **IEEE Xplore**: 33,302 results;

The following query strings were used on each of the digital libraries cited:

- **Springer**
  ("deep learning" OR "computer vision" OR "artificial intelligence")
- **ACM Digital Library**
  (
  acmdlTitle:(+deep +learning) OR
  acmdlTitle:(+computer +vision) OR
  acmdlTitle:(+artificial +intelligence)
) AND (

recordAbstract:(+deep +learning) OR
recordAbstract:(+artificial +intelligence) OR
recordAbstract:(+computer +vision)
)

- **IEEE Xplore**
(((”Abstract”:deep learning) OR
”Abstract”:artificial intelligence) OR
”Abstract”:computer vision)

After reading and filtering based on relevancy, the final applications that bring most content to the discussion of this report are the following.

5.1 A Hybrid Gomoku Deep Learning Artificial Intelligence - (57)

This work brings an alternative to the usage of search trees for game solving, with the example on the board game called Gomoku. The authors followed an approach of using deep convolutional neural networks for supervised learning solving the game. Authors expose an accuracy of 69% on training data and 38% accuracy on testing data.

5.2 Application of Computer Vision and Deep Learning in Breast Cancer Assisted Diagnosis - (58)

Recent advances in medicine were only made possible through a series of external structures and techniques that made doctors and health care professionals achieve better results. This work has a focus in one of these structures to process breast cancer diagnosis.

Traditionally, the process of detecting breast cancer passes through a doctor analyzing and judging B-mode ultrasound images, visually. This method can generate different results based on different doctors experiences. The article utilizes deep learning concepts to, through computer vision, help the decision-taking process of doctors on the diagnosis, based on historical data.

5.3 GeoAI 2017 workshop report: the 1st ACM SIGSPATIAL International Workshop on GeoAI: @AI and Deep Learning for Geographic Knowledge Discovery - (37)

With the advances on deep learning and multi-layer information processing of networks a lot of fields can gather benefits from this and apply in very specific ideas. The field of geographical information and processing has a vast diversity of geospatial data in high-resolution already collected through years, such data processed with the right techniques can retrieve results not possible in the past years.

The work cites study cases in which computer vision algorithms on deep learning were able to segment and extract building information on remote zones of African continent using satellite images. The information exported from such models may allow, for example, humanitarian organizations to map buildings and community structures over remote places of the globe.

5.4 Visual landmark sequence-based indoor localization - (33)

The importance of indoor localization, mainly for the logistics industry, is crucial, allowing companies to deliver their or third-party products as fast as possible to final consumers (3).

This paper presents a method that uses common objects as landmarks for mobile-based indoor localization and navigation. Authors use data collected from mobile videos captured by users holding smartphones while walking through corridors of an office building. Figure 5.1 brings examples of indoor objects detected by the proposal.

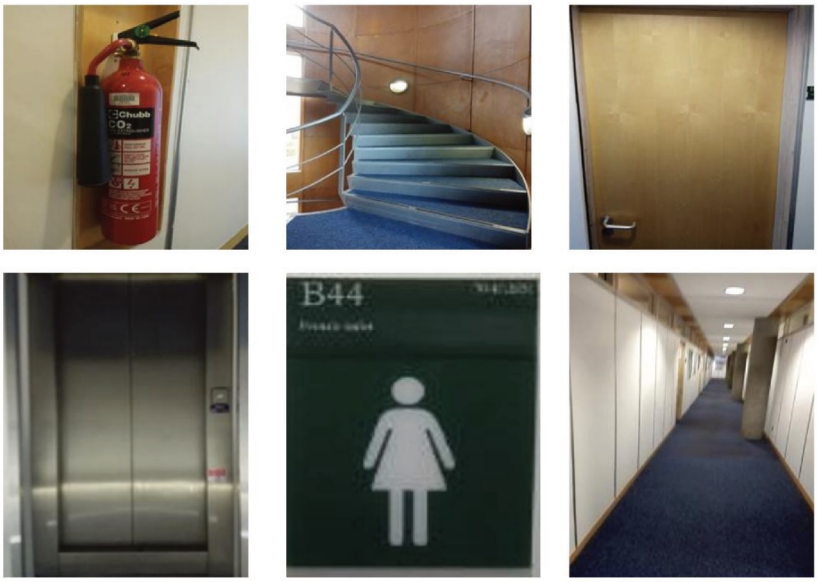


Figure 5.1: Indoor objects and challenging locations

5.5 Deep Learning in the Field of Art - (59)

The area of machine learning and deep learning are well known as areas in which computers may overtake human ability, based on the capabilities of replication such cognitive behavior through computer algorithms.

However, the authors show that not only exact science can use the benefits of such techniques but abstract sciences too. The paper reviews the application of deep learning in painting, music and literature.

5.6 Review of State-of-the-Art in Deep Learning Artificial Intelligence - (47)

The given article brings a review of the literature related to the current state-of-the-art on deep learning based artificial intelligence. Beyond that, several estimates to compare the current ability of deep learning models to replicate human-level capabilities is proposed on the work.

5.7 Driver information system: a combination of augmented reality, deep learning and vehicular Ad-hoc networks - (9)

Paper that presents current state-of-the-art on the vehicular navigation on driverless cars. Based on the authors, In vehicle-based safety systems, it is more desirable to prevent an accident than to reduce the severity of injuries. Critical traffic problems such as pedestrian accidents and traffic congestion require the development of new transportation systems.

Research in recognition and human decision-making process is needed for the relevant and correct display of this information, for maximal road traffic safety. It is presented the concept of Augmented Reality Head-Up Display (AR-HUD), which can facilitate a new form of dialogue between the vehicle and the driver.

Afterwards, researchers propose a deep-learning-based object detection approach for identifying and recognizing road obstacles types. A single convolutional neural network predicts a region of interest and class probabilities directly from full images in one evaluation.

5.8 Road vehicle detection and classification based on Deep Neural Network - (61)

This paper analyzes and brings to discussion the superiority of the deep learning techniques over the aspect of feature extraction. Consequently, authors propose that using deep learning can extract high-level features from low-lever features though its given layer structure.

After the initial decisions, an algorithm is proposed and applied in the scenario of road vehicle detection. Experiment results show that increasing the amount of the data, the mean error and misclassification rate gradually decreases. Finally, the authors propose suggestions of improvements for future works.

5.9 Pedestrian Detection Algorithm Based on the Improved SSD - (34)

The work describes the popularity of the research area of pedestrian detection over the field of computer vision, widely used in the fields of automatic driving safety and security and pedestrian analysis.

With the development of deep learning, pedestrian detection methods based on it greatly improves the accuracy of pedestrian detection models. The paper proposes an improved version of the deep learning architecture called SSD (35) over pedestrian detection, based on the context information.

Chapter 6

Embedded Platforms

With the advance of the academia and industry applications towards the area during the past years, the existing computer hardware produced by hardware manufactures needed to evolve to follow these needs. The techniques used in artificial intelligence, as seen on the sections above, have been crafted in the past century, however, only after the 2000s we started to apply them in real use cases with efficiency.

It started with the creation and production of the initial multi-core processors, allowing basic artificial neural networks calculations to be executed in concurrent paradigms. As models started to grow and the field grew towards deep neural networks, the need to execute complex mathematical operations and storage of large count of activation weights for inference and training models rose.

Even with the attempt to create specific hardware for artificial neural networks optimization at the end of the 20th century (25), (41) the real breaking point for deep models was the popularization of Graphics Processing Units (GPUs). GPUs are classified as a type of accelerator with main focus on graphics processing that started to grow with the advances in the entertainment industry, including audio and video processing and games industries.

GPUs excel at matrix operations and vector multiplications, used for fast-forward pass and backpropagation optimization on DNNs. Beyond that, having a single accelerator to process the entire pipeline for training and inference using the architectures allows faster memory read and write operations, given the higher memory bandwidth.

Recently, a movement of cloud providers to allow developers to access accelerators for the execution of deep learning models occurred, mitigating the need to obtain physical hardware to manipulate them. However, a large number of applications emerged with the necessity of low latency inference or decision making process (51).

Consequently, the hardware industry started to focus on embedded platforms that suppress the problem and delivers low latency high precision results. Embedded hardware for artificial intelligence requires specific tasks to allow the accelerators to be optimized to the given problem. The Table 6.1 shows examples of problem-specific hardware existing in the industry these days and the focus, as specified by each manufacturer.

6.1 Jetson TX2

One of the most popular embedded hardware in usage for artificial intelligence on current days is the Jetson TX2, from NVIDIA Corp. It is an AI general purpose hardware, with focus in power management and high performance inference for real time situations, as in autonomous machines for example. The Figure 6.1 presents

Manufacturer	Model
NVIDIA	Jetson Nano
NVIDIA	Jetson AGX Xavier
NVIDIA	Jetson TX2
Intel®	Nervana™ NNP-T
Intel®	Nervana™ NNP-I
Qualcomm	Snapdragon 855
IBM	TrueNorth (45)

Table 6.1: Comparative table of existing embedded hardware for AI.

the technical specifications for the platforms.

TECHNICAL SPECIFICATIONS

	TX2 4GB	TX2	TX2i
GPU	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores		
CPU	Dual-core Denver 2 64-bit CPU and quad-core ARM A57 complex		
Memory	4 GB 128-bit LPDDR4	8 GB 128-bit LPDDR4	8 GB 128-bit LPDDR4
Storage	16 GB eMMC 5.1	32 GB eMMC 5.1	32 GB eMMC 5.1
Video Encode	3x 4K @ 30 (HEVC)		
Video Decode	4x 4K @ 30 (HEVC)		
Connectivity	Wi-Fi requires external chip	Wi-Fi onboard	Wi-Fi requires external chip
	Gigabit Ethernet		
Camera	12 lanes MIPI CSI-2, D-PHY 1.2 (30 Gbps)		
Display	HDMI 2.0 / eDP 1.4 / 2x DSI / 2x DP 1.2		
UPHY	Gen 2	1x4 + 1x1 OR 2x1 + 1x2, USB 3.0 + USB 2.0	
Size	87 mm x 50 mm		
Mechanical	400-pin connector with Thermal Transfer Plate (TTP)		

Figure 6.1: Technical specifications for all three Jetson TX2 models

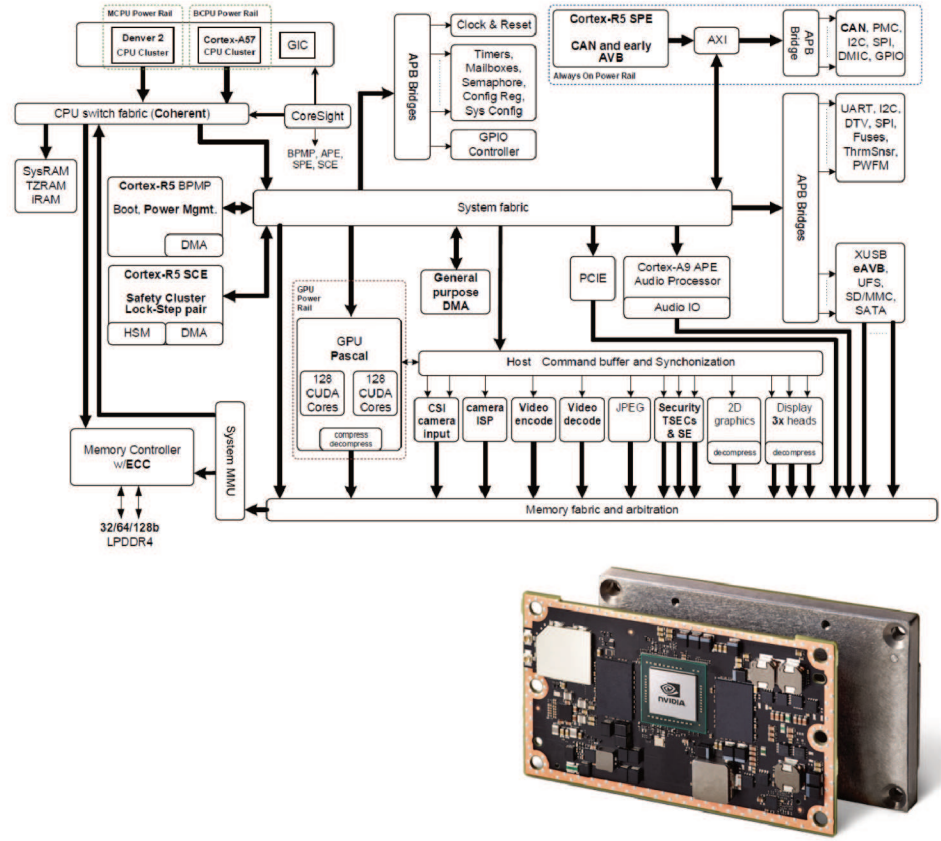


Figure 6.2: Hardware specification for the default Jetson TX2 model

Figure 6.3: Jetson TX2 Development kit and accessories

With the platform described above, researchers around the world have been able to execute models for surveillance and face recognition (27), (17), remote medical analysis (48), (40) and autonomous vehicles (29), (30).

Figure 6.2 shows a schema of the hardware present on the embedded board. Worth notice that the platform has a robust NVIDIA Pascal™ GPU, with 256 CUDA cores and combines two different CPU processors, a Denver 2 dual-core processor and an ARM Cortex-A57 quad-core processor. The image presented bellow the schema is the default format for the Jetson TX2.

Figure 6.3 shows the development kit for the model, which is the model used in the following test results presented on this report. Accordingly to the manufacturer, the main difference between the development kit and the production-ready model are the components present on the model, which may vary without previous notes from board to board.

6.2 Getting Started

To get started using the Jetson TX2 platform, it is needed to have a host computer running Linux and with an internet connection. NVIDIA has developed a set of tools and grouped them in a package called JetPack (1).

JetPack can be downloaded at NVIDIA's website. The current latest release is version 4.4.2 and includes, among other packages, the following ones:

- CUDA
- cuDNN
- TensorRT
- OpenCV

After downloading the JetPack at the host computer, simply execute the installer on it. It can be made by installing NVIDIA's software development kit (SDK) manager. After executing it, will be required to log in and authenticate a developer account and the installer will redirect to the hardware selection page, as shown in Figure 6.4.

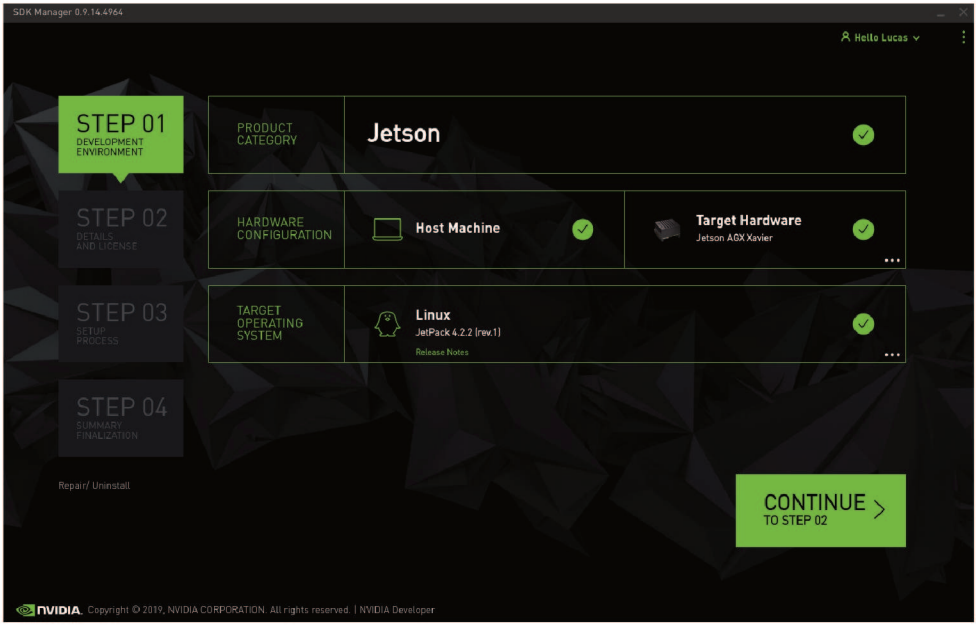


Figure 6.4: Initial page for the SDK manager

To install the pack for TX2, click at the ellipsis button at the "Target Hardware" section and select the right model, as presented at Figure 6.5.

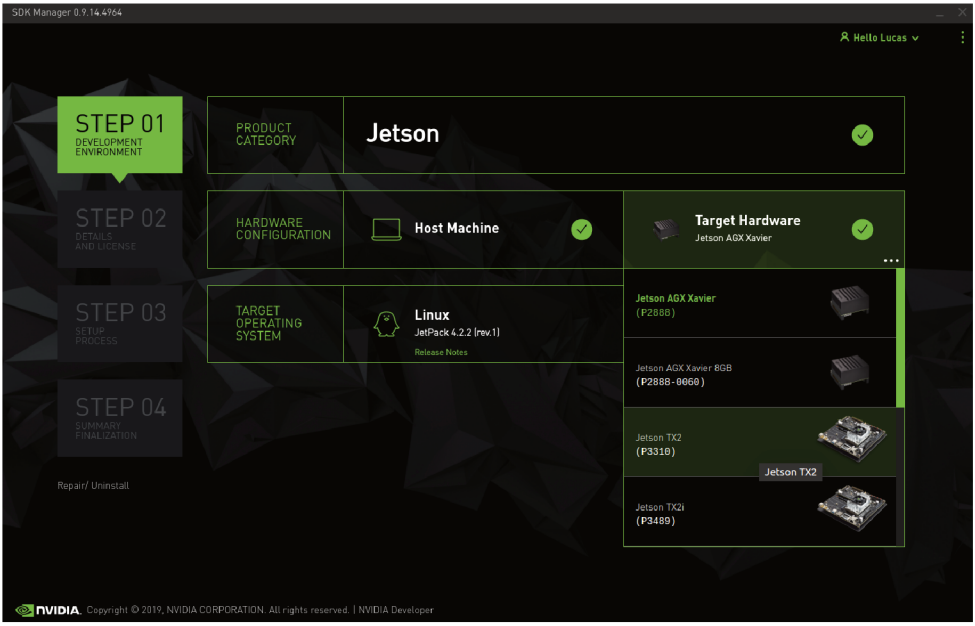


Figure 6.5: Hardware selector at the SDK manager

After the selection of the specific board model, the screen presented at Figure 6.6 show all the component packages to be installed during the next phase. The packages under the "Host Components" are going to be installed at the host computer, executing the manager, and "Target Components" are packages to be installed at the embedded platform.

Additional artificial intelligence libraries can be seen at the last section in Figure 6.7. To install tensorflow, base for the benchmark at this project, select that option.

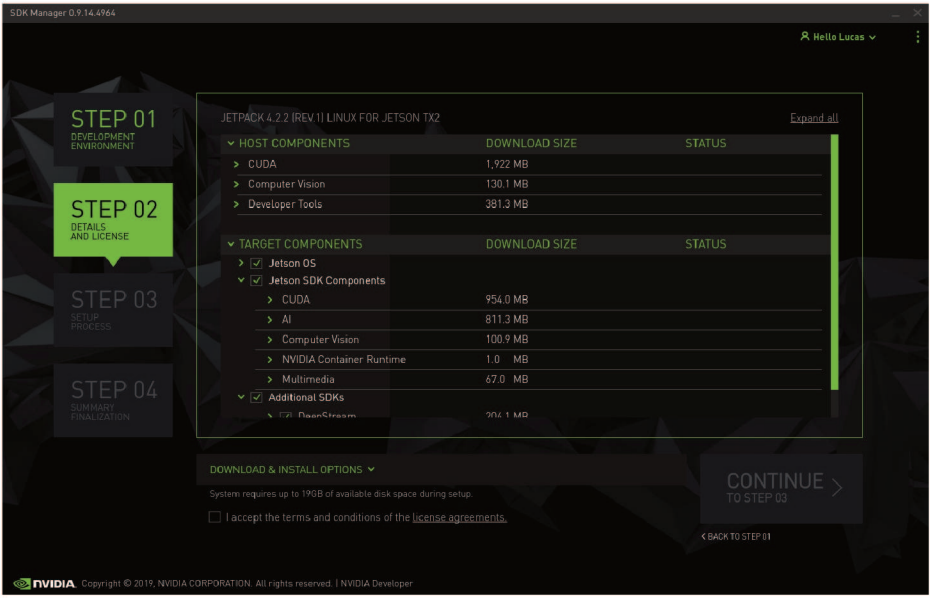


Figure 6.6: Packages to be installed at both host and final hardware

Figure 6.7: Additional packages support on JetPack

When selecting the next step, each one of the packages are going to be downloaded and host ones installed already. After installing packages at the host machine, the SDK manager will ask the developer to setup the target device.

For that, it is needed that the developer execute the following steps:

a. Connect the device at the host machine through the Micro B-USB port of the Jetson

b. Turn the device into Recovery Mode to allow installation of final components

6.2.1 Automatic Setup

To put the device in recovery mode, there are two different ways. The first one consists in turn on the board, login with one user and connect to the same network as the host machine. After that, the user must inform the IPv4 address of the board, username and password, as shown at Figure 6.8.

6.2.2 Manual Setup

One alternative, in case the developer is not able to connect the board into the internet, is to chose the "Manual Setup" at the wizard and follow the steps:

a. Power down the device and remove the AC adapter, if connected;

b. Connect the Micro B-USB port at both the Jetson board and host machines;

c. Connect the power adapter to Jetson;

d. Press and release the **power** button to start the board;

e. Press and hold the **force recovery** button and, while holding it, press and release **restart** button, releasing the initial button 2 seconds after;

f. It must be booted in recovery mode already. It can be checked executing the command *lsusb*.

After choosing one of two methods, the packages are going to be installed into the target machine and you will be able to use it normally.

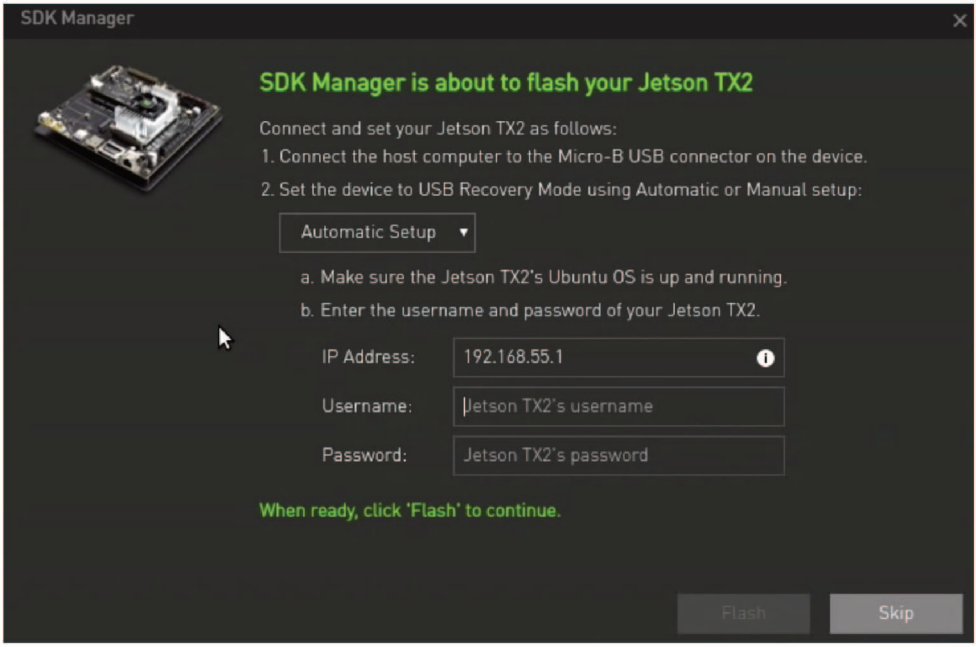


Figure 6.8: Setup Jetson to recovery mode

6.3 Troubleshooting

During the process of setting up the development kit to usage for the benchmark proposed, a few issues were faced and documented for future reference. Some of them are presented bellow, as a source of knowledge, in case any developer suffers from close issues.

6.3.1 tensorflow.python.framework.errors_impl.InternalError: Failed to create session

This problem existed right after the testing phase of the initial models. It is an issue related to the lack of HDF5 file format libraries. This problem can be solved executing the following command on the device:

```
sudo apt-get install libhdf5-dev python-h5py
```

Chapter 7

Benchmark

Based on the previous statements and analyzing the literature on benchmarks over AI models performance with general-purpose embedded hardware, more specifically related to the Jetson platform, from NVIDIA Corp. the present work intends to bring to literature a benchmark over different areas of such type of systems.

It proposes a general-purpose benchmark over certain components of an inference system. Table 7.1 presents the conditions and components measured.

The decision over components and data to be collected is based on most common metrics looked at when working over such environment. It is known that GPU performance is one of the most influencing factors over the training and inference process of a neural network based model (31). To work with such topic, metrics as GPU temperature and Inference time were collected, to give researchers a clearer vision of the capabilities of the system.

Component and Metric	**Measured through**
Frames per Second (FPS)	Grouped based on the iterations
Inference time	Average time through all iterations
GPU Temperature	Average and standard deviation through all iterations
Memory usage.	Average and standard deviation through all iterations

Table 7.1: Elements measured Comparative table of existing embedded hardware for AI.

In order to test the capabilities of the board for the given aspects proposed above, it was decided to execute a set of models based on convolutional neural networks (CNNs) already existent on literature. Table 7.2 presents the architectures used for this benchmark.

The decision of using CNNs was taken based on the embedded camera present on the microchip and the facility to work with such models on a GPU based architecture. Beyond that, the context on which the board is likely to be executed interferes on the decision, considering that the controller is optimized for self-driving cars scenarios.

Architecture	Dataset Trained	Reference
Faster R-CNN	Microsoft COCO	(42)
Mask R-CNN	Microsoft COCO	(21)
SSD Inception V2	Microsoft COCO	(52), (35)
SSD Mobilenet V1	Microsoft COCO	(23), (35)
SSD Mobilenet V2	Microsoft COCO	(44), (35)

Table 7.2: Deep learning models used for the benchmark experiments.

Furthermore, the decision of use already pre-trained models to the collect data for the benchmark was taken, given that metrics such as intersection over union (IoU) and prediction reliability were not the focus of the benchmark, as authors from the original models already have presented sets of results over them. Consequently, being as efficient as possible, we worked over weights already trained over the Microsoft COCO dataset (54).

7.1 Existing approaches

Researching through existing articles and benchmarks over the Jetson family, even being a recent family from NVIDIA to be released, we can find interesting resources to base projects on.

Dustin Franklin's article (6) exposes a benchmark over the Jetson Nano model. The author executes a set of pre-trained deep learning models existing on literature to extract data over the performance of the device. The framework TensorRT is used to optimize the inference steps for each model and the results can be seen in Figure 7.1.

One of the differentials of the benchmark proposed by the current work is the normalization over the resolution of images passed for inference in each batch, which brings a clearer argument comparing different models over the same environment.

(5) presents a benchmark over the production ready platform of the same model used at the current work. One of the main differences of the proposed report is the type of networks tested under the development kit. The authors of the article used base network architectures, usually used along with detection networks. This joint architecture is the main focus of the models tested by the present work.

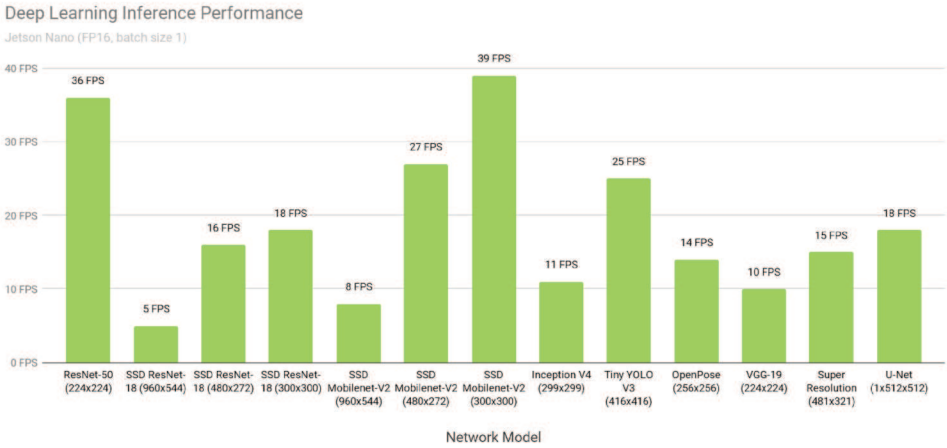


Figure 7.1: Results from Jetson Nano benchmark

		NVIDIA Jetson TX1	NVIDIA Jetson TX2		
		Max Clock (998 MHz)	Max-Q (854 MHz)	max-P (1122 MHz)	Max Clock (1302 MHz)
GoogLeNet batch=2	Perf	141 FPS	138 FPS	176 FPS	201 FPS
	Power (AP+DRAM)	9.14 W	4.8 W	7.1 W	10.1 W
	Efficiency	15.42	28.6	24.8	19.9
GoogLeNet batch=128	Perf	204 FPS	196 FPS	253 FPS	290 FPS
	Power (AP+DRAM)	11.7 W	5.9 W	8.9 W	12.8 W
	Efficiency	17.44	33.2	28.5	22.7
AlexNet batch=2	Perf	164 FPS	178 FPS	222 FPS	250 FPS
	Power (AP+DRAM)	8.5 W	5.6 W	7.8 W	10.7 W
	Efficiency	19.3	32	28.3	23.3
AlexNet batch=128	Perf	505 FPS	463 FPS	601 FPS	692 FPS
	Power (AP+DRAM)	11.3 W	5.6 W	8.6 W	12.4 W
	Efficiency	44.7	82.7	69.9	55.8

Figure 7.2: Results from Jetson TX2 benchmark

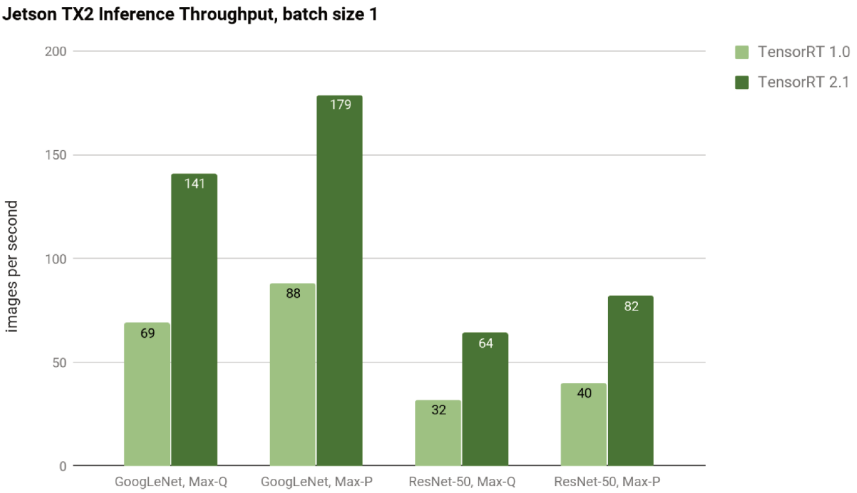


Figure 7.3: Results from TensorRT V1 and V2 comparative

Finally, (4) presents a benchmark over the performance of the TensorRT framework and a comparison from the latest release, considering the present date, and a previous generation. The results of the comparative, at Figure 7.3, are useful when considering the version that better suits the needs of the developer's application, as the usage of the framework is not required over the platform.

7.2 Models

For a better understanding of the experiments, it is first needed to summarize the behavior and expose nuances of each individual model to be tested.

7.2.1 Faster R-CNN

The model presented by the authors consists of an architecture improvement over the Fast R-CNN model. Its main modification is the absence of selective search in the proposed region generation process, thus improving the performance and reducing the resource usage of the inference device (42).

To obtain the same result, the authors propose a separate network from the original used only for prediction of the region proposals. After the prediction, the model goes through a region of interest layer to reduce dimensions, thus being used to aid in the classification of objects. Figure 7.4 presents the modifications in the architecture in question.

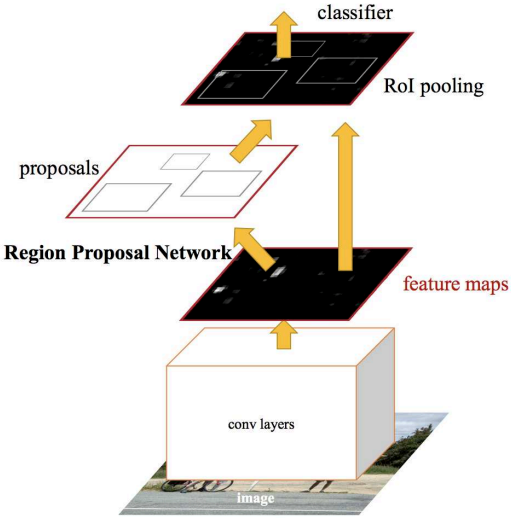


Figure 7.4: Faster R-CNN Architecture

7.2.2 Mask R-CNN

The article presents a different proposal from the others, performing detection and segmentation of the input sets. As a successor to the Faster R-CNN model, one of its principles is the pursuit of performance when inferring input data (21).

The main change from its predecessor is the new object segmentation mask extraction branch, in addition to the object detection branch. Both branches run in parallel. Figure 7.5 shows some examples of model execution results.

Figure 7.5: *Mask R-CNN*

The model in question is divided into stages. The first stage, like its predecessor, consists of a network for generating possible objects to detect Region Proposals Network. In a second stage, the network is divided into a detection branch and a segmentation branch.

The results presented show that the model, with all its branches, can infer at a speed of 5 frames per second, with 195ms per image on a Nvidia Tesla M40 GPU.

7.2.3 Single Shot Multibox Detector (SSD)

The model uses a concept in which a grid division of the initial image is performed, with the initial image being observed only once. Four detection boxes are used for each grid cell, and they are of different proportions, as shown in Figure 7.6. It is noticeable that the architecture uses boxes with smaller proportions, referring to the size of each cell, being responsible for detecting smaller objects (35).

The authors perform experiments on various data sets such as Pascal VOC 2012 (36) and Microsoft COCO, presenting expressive results in terms of required computing power. In addition, results are presented that show a classification accuracy superior to Fast R-CNN model, previously mentioned.

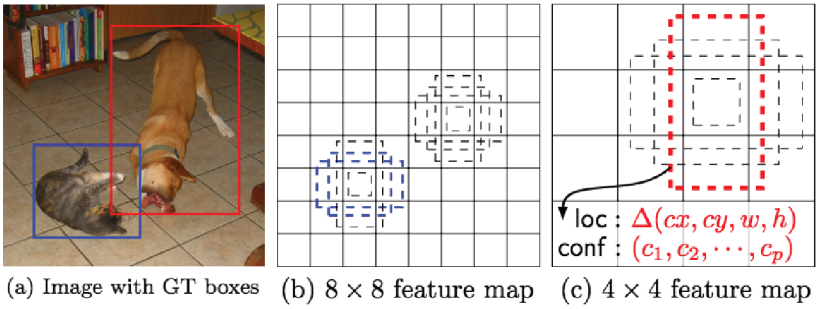


Figure 7.6: SSD Architecture

7.2.4 Inception V2

Focused on solving apparent height and width variance problems of objects in images, the model that serves as the basis for the work presented uses the concept of multiple convolutions per layer, using filters with different dimensions for each convolution. At the end, the model performs concatenation of the results of each filter (53).

Version 2, presented in the paper, is based on the same concept as shown in Figure 7.7, however, reduces higher dimension convolutions with focus on performance improvement. The authors cite a 2.78 times improvement using two convolutions of size 3x3 over one of size 5x5 (used in the initial model) (52).

In addition, the researchers made changes to filters that use only 1xN or Nx1 type convolutions, reducing the amount of weight required to be trained.

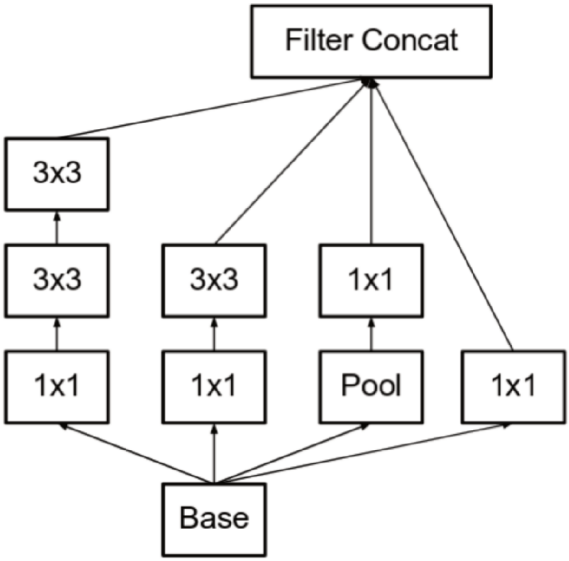


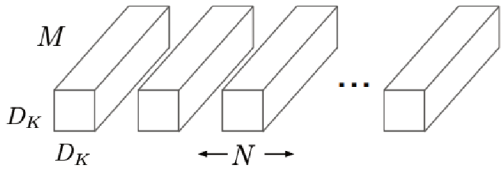
Figure 7.7: Width convolutions with the Inception V2 architecture

7.2.5 MobileNets

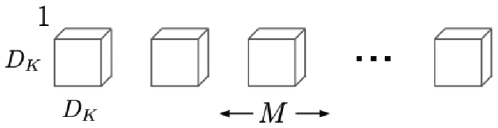
The authors bring to the discussion the need to use models architected for low capacity and low latency on embedded platforms. The initial work presents the MobileNet model, having as main focuses the above objectives (23). The optimizations presented about the proposed model are due to the use of separable and depth convolutions, being a technique responsible for separating convolutions new images on smaller images, but in sequence, producing a deeper network.

Its main advantage is that it reduces weights number since the result of this layer can be controlled by the number of kernels used in the convolution. Figure 7.8 presents the technique used by the proposed architecture. Advancing the concept, a few new techniques are extended by the model at it's second version (44). The first is called reverse residual networks, responsible for connecting the initial layers of the model directly with the final layers. The approach allows the model to use a pattern that starts with large channels convolutions, reduce the dimensions in the middle layers, and return to expand at the end, thereby reducing the amount of weights needed to be stored by the architecture. Figure 7.9 presents the concept described.

Another proposed improvement is the use of the activation function ReLU6. modification of the original function by limiting the maximum value of the output to a 3-bit number, decimal value 6. This ensures floating point accuracy and improves the performance, eliminating unnecessary precision information.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters

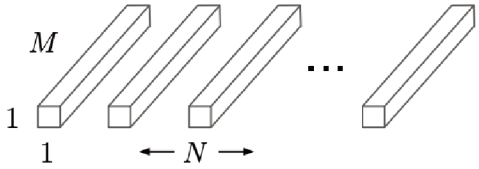


Figure 7.8: Depth Convolutions over the MobileNet V1 architecture

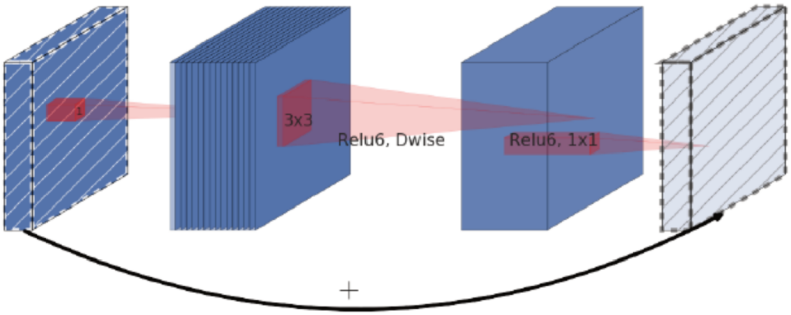


Figure 7.9: Residual Networks over the MobileNet V2 architecture

7.3 Datasets

For the execution of the experiments of the work in question, two data sets were chosen as the real world scenarios to be tested. The decision about them is based on them being general purpose data sets with feasible dimensions and represent likely environments where the Jetson TX2 platform would be used at, such as autonomous cars.

7.3.1 Microsoft COCO

The Microsoft COCO dataset aims to be a real-image database focused on object detection, segmentation and subtitles. The images bank has diverse contexts over 80 distinct object categories (54). Figure 7.10 shows samples extracted from the base.

The models tested in the present work were trained on the data set. The database provides 330,000 images total, 41,000 of which are separated by the authors themselves as testing images, which are tested against the benchmark in question. The images in the dataset have varying dimensions, to enable learning about various models of image capture devices.

7.3.2 KITTI Stereo Vision

The KITTI Stereo Vision database, also entitled as a benchmark image set, consists of 200 training images and 200 test scenarios. The images are formatted under the PNG format, reducing resource loss and noise. Both test and training scenarios were used during the experiments of this work (38).

The data presented by the authors have a standardized size of 375 pixels of height, 1242 pixels of width and 3 color channels (RGB). In addition, the scenes presented by the set were collected about road scenarios, both urban and rural, and through stereo cameras.

Figure 7.10: Sample images from Microsoft COCO dataset

Figure 7.11: Sample images from KITTI Stereo Vision dataset

7.4 Implementation

Related about the implementation, a few premises were taken to better collect information over the embedded system. They are described as:

- Usage of Tensorflow with cuDNN, from NVIDIA;
- Be as general purpose as possible, allowing future testing easily;

During the implementation of the benchmark, the first premise changed. Initially, the main goal was to work with the already consolidated framework OpenCV (13), based on computer vision functionalities and functions. However, during the initial phase of the experiments, it was detected that OpenCV framework does not support NVIDIA's GPU-accelerated library cuDNN (14).

Based on the most recent implementation supported for the Jetson at the time of the OpenCV framework (3.4.6), only the OpenCL framework is supported. However, there is no official support for OpenCL as an alternative to the cuDNN library on the Jetson family.

To get around the problem, the Tensorflow (8) framework was the closest alternative and was taken as the choice for the benchmark as it supports cuDNN library.

The benchmark is all written using python, being executed by the interpreter for version 3 or higher. All the source code for the software can be found over the address
"https://github.com/lucaspbordignon/onYourMarks" and under the license
GPL 3.0.

During the process of inference of each image passed to an individual network, input images are resized in a common dimension, to avoid any influence over the final results based on the image size of the inputs and possible delays that using custom images may appear. The decision is based over the argument that for real applications, it is very likely that the same equipment will be used to collect data from the source.

For the code above cited, we initialize a metrics collector. After the execution of each individual inference step the collectors are stopped, they register the checkpoint for the given model and finish the step for a single image.

The collectors used in the benchmark are responsible to collect 4 general metrics of the system:

- Time data: As elapsed time, inference time and others;
- CPU/GPU data: As temperature for both CPU and GPU;
- Memory data: As process usage, swap usage and others;
- FPS data: As the number of images passed and predictions count;

All the implementation details can be found at the repository on GitHub, as with instructions for the execution and testing the benchmark over Jetson TX2.

7.5 Experiment Results

NVIDIA's Jetson TX2 platform has a set of custom modes of operation, fitting exactly to the problem needs. The modes present on the board are a set of settings that can be made to the system, changing the frequency of both GPU and CPU frequency and cores.

To manipulate such modes, one can make it manually for setting specific values for each parameter or use the *nvpmodel* tool, developed by the manufacturer to change it. It comes with 5 different modes of operation, presented on Table 7.3.

Operation Mode	Mode Name	Denver Cores Enabled	ARM A57 Cores Enabled	GPU Frequency
0	Max-N	2	4	1.30GHz
1	Max-Q	0	4	0.85GHz
2	Max-P Core-All	2	4	1.12GHz
3	Max-P ARM	0	4	1.12GHz
4	Max-P Denver	2	1	1.12GHz

Table 7.3: Jetson TX2 modes of operation

For all the experiments executed over the board, the operation mode "Max-N" was used, which is the one with higher frequency over the Pascal GPU. Worth notice for temperature tests that all experiments were executed after fresh reboot of the system with a standby period of 15 to 30 minutes between interactions.

7.6 Microsoft COCO

The following charts explore the results of the benchmark execution over the Microsoft COCO dataset. It consists in a general purpose dataset, containing 80 different object types and 40670 images for testing the models. The execution of the benchmark was over all the testing images for each individual model.

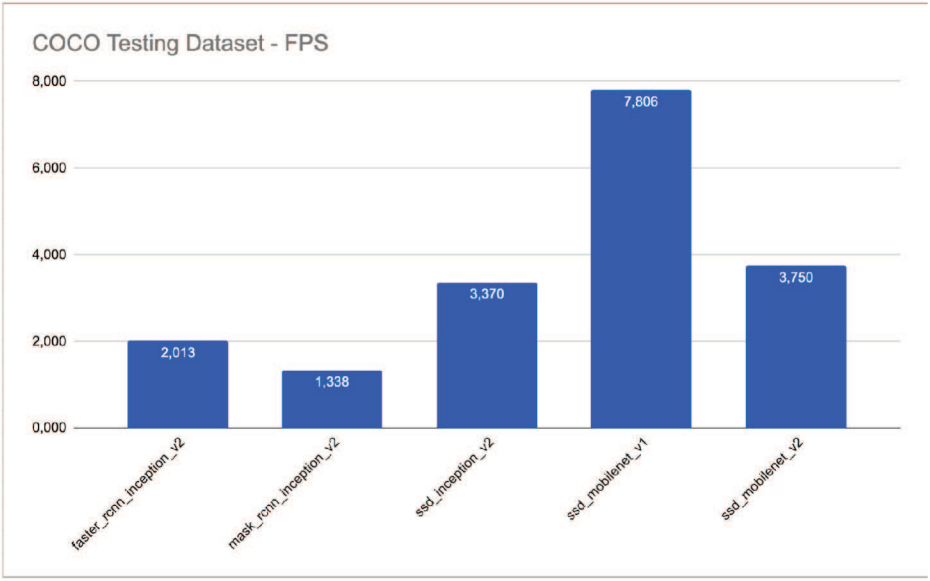


Figure 7.12: Benchmark resulting Frames per Second over COCO dataset

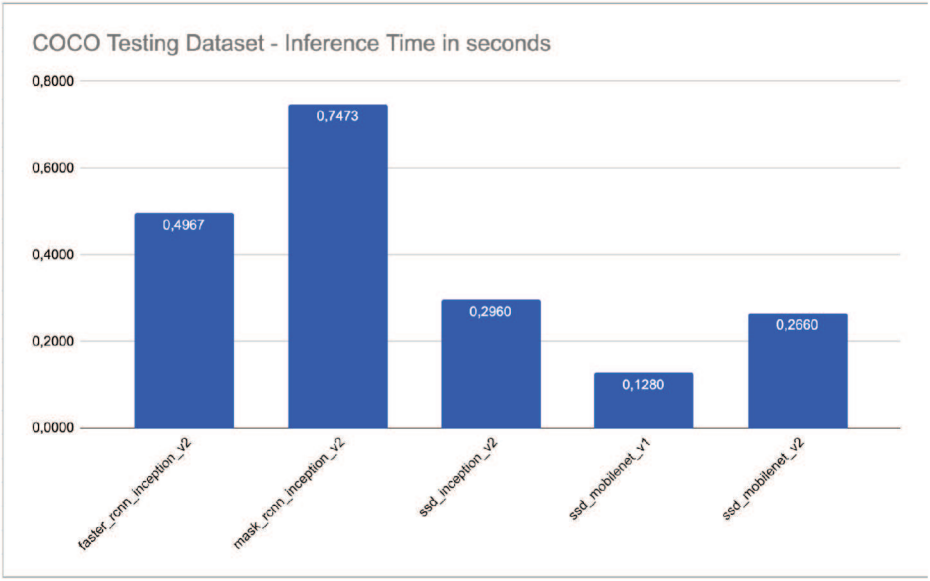


Figure 7.13: Benchmark resulting Average Inference Time over COCO dataset

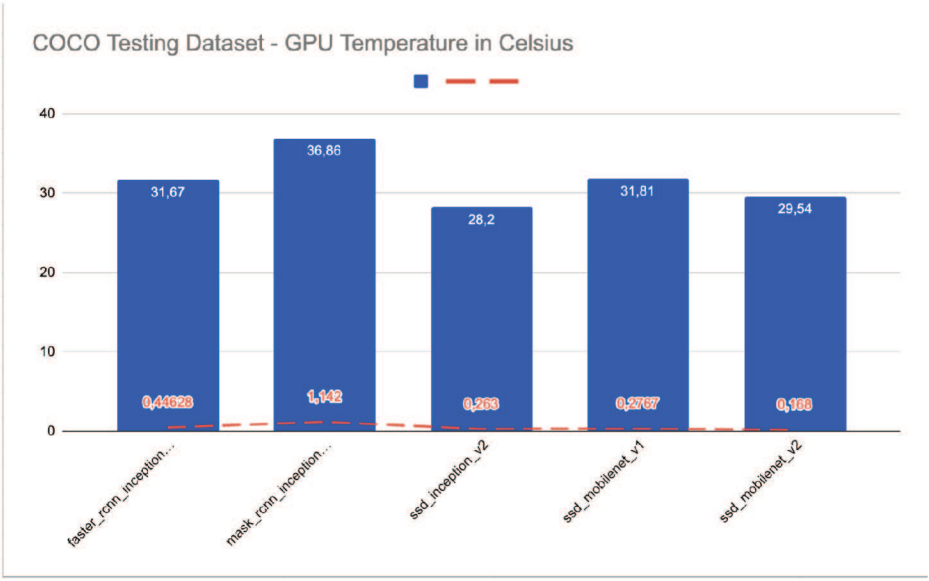


Figure 7.14: Benchmark resulting Average and Standard Deviation of GPU Temperature over COCO dataset

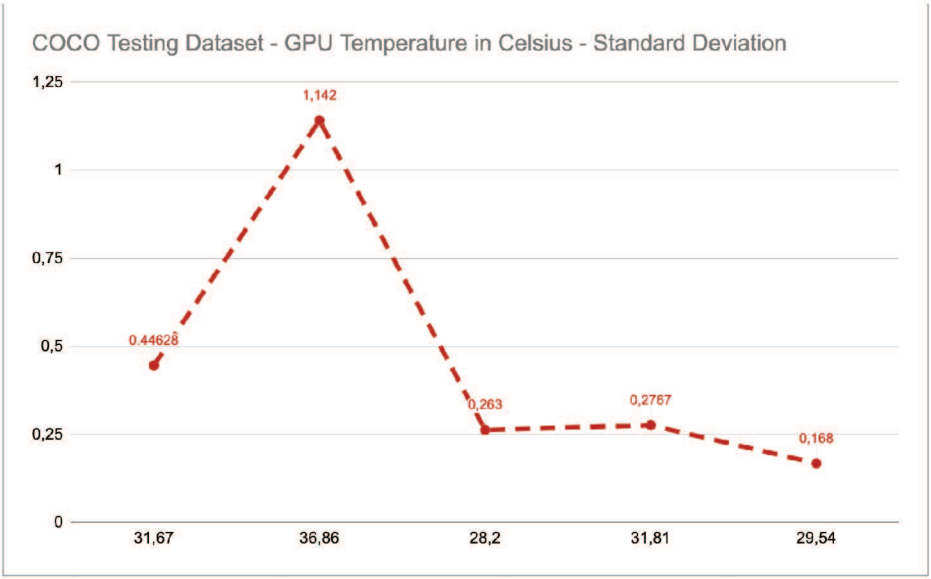


Figure 7.15: Benchmark resulting Standard Deviation of GPU Temperature over COCO dataset

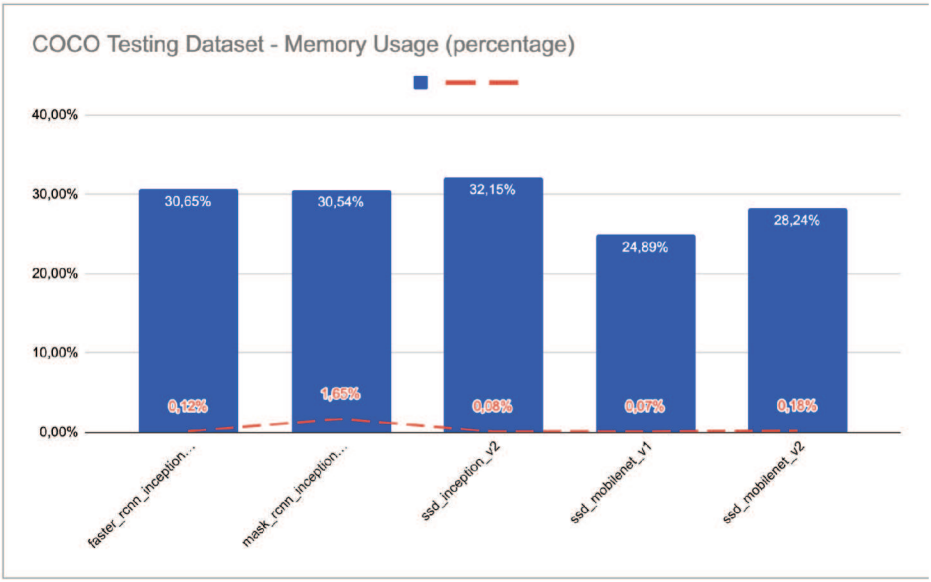


Figure 7.16: Benchmark resulting Average and Standard Deviation of GPU Memory use over COCO dataset

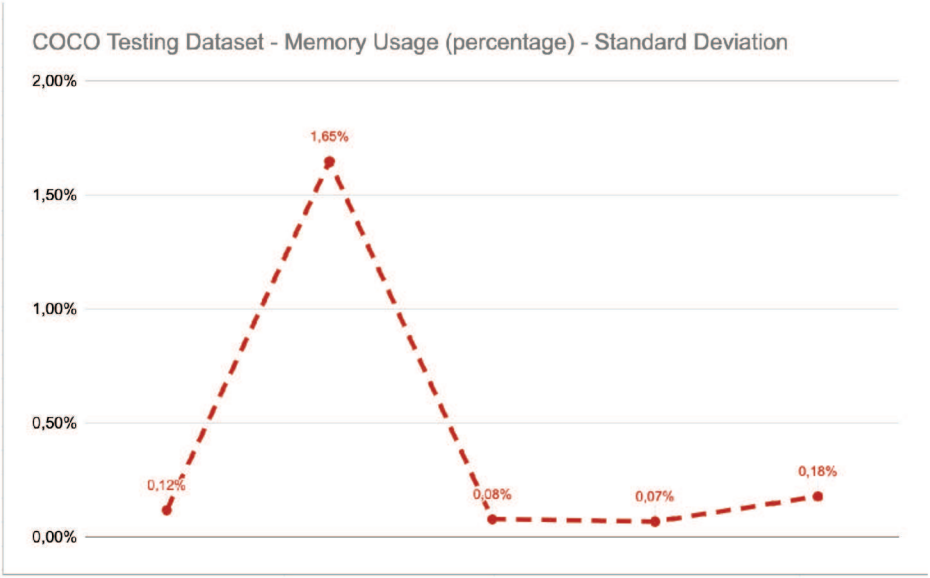


Figure 7.17: Benchmark resulting Standard Deviation of GPU Memory use over COCO dataset

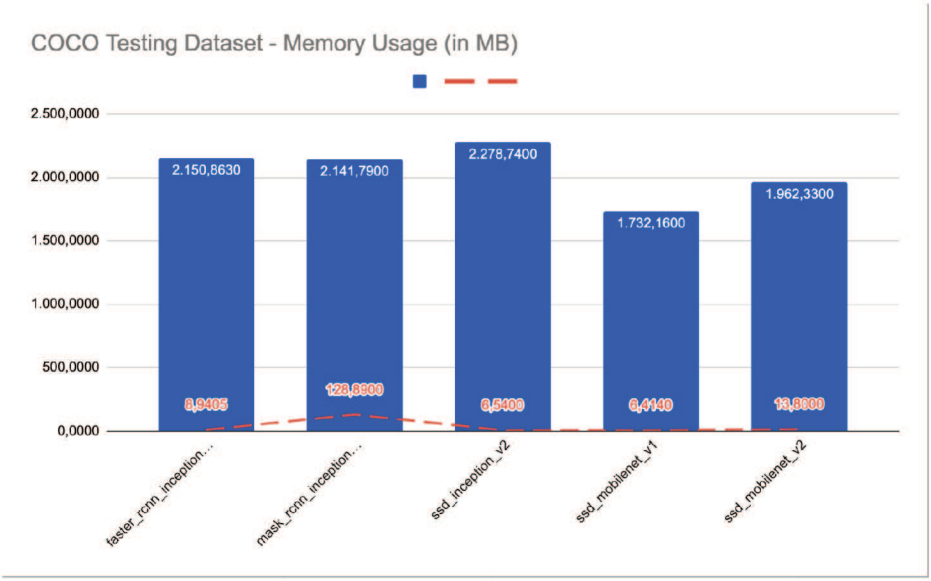


Figure 7.18: Benchmark resulting Average and Standard Deviation of GPU Memory use over COCO dataset

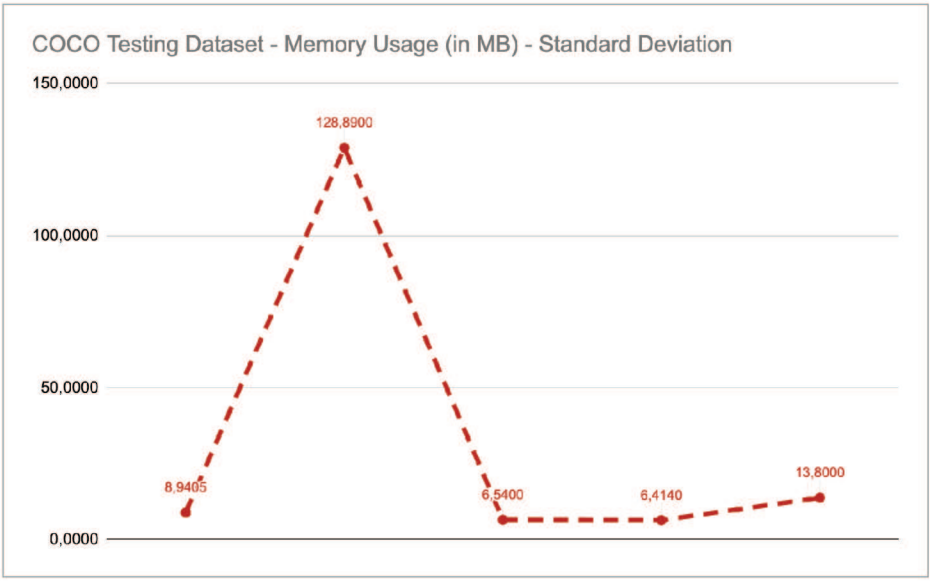


Figure 7.19: Benchmark resulting Standard Deviation of GPU Memory use over COCO dataset

7.7 KITTI Stereo Vision

The following results are collected over the KITTI Stereo Vision dataset for testing, which consists of 400 images of pedestrian and road scenarios under common circumstances. The execution of the benchmark was over all the testing images for each individual model, results are presented bellow.

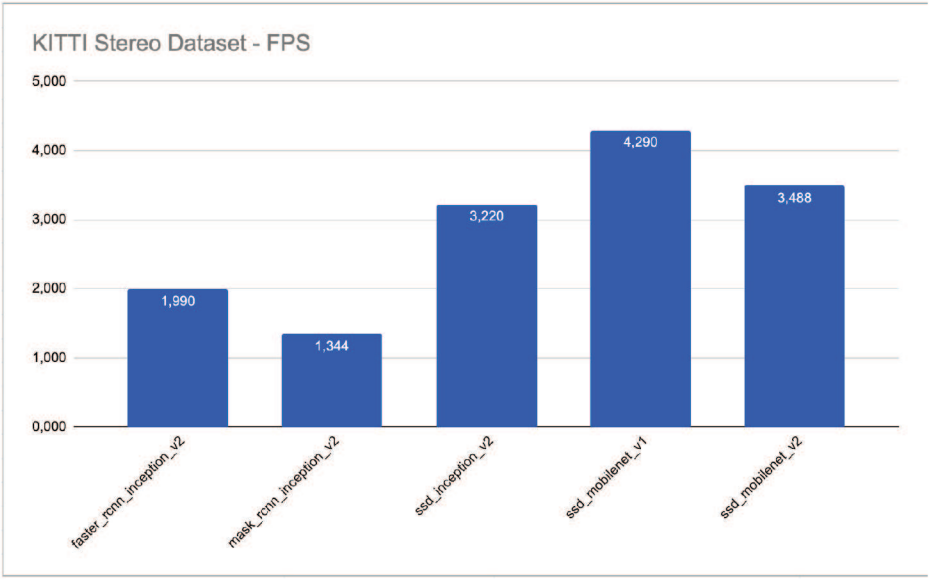


Figure 7.20: Benchmark resulting Frames per Second over KITTI Stereo Vision dataset

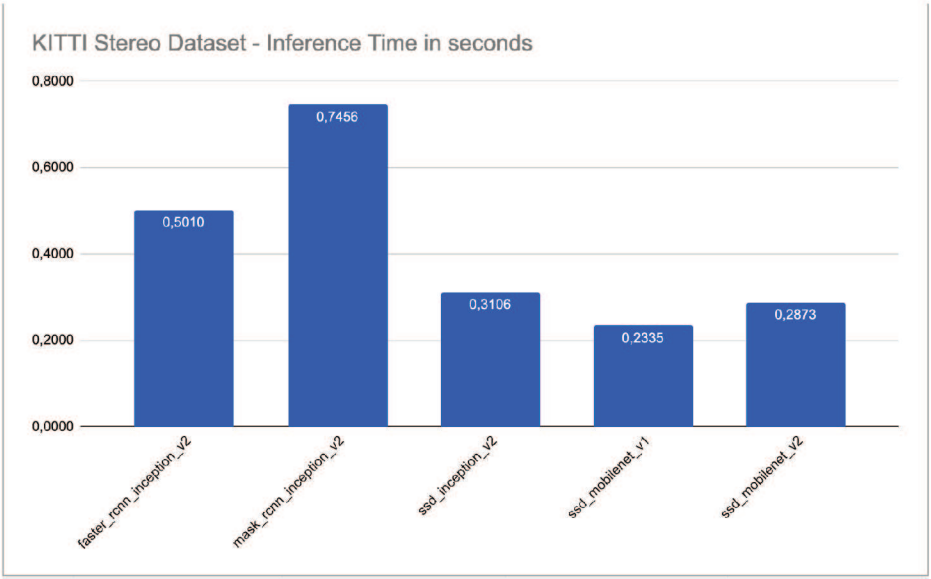


Figure 7.21: Benchmark resulting Average Inference Time over KITTI Stereo Vision dataset

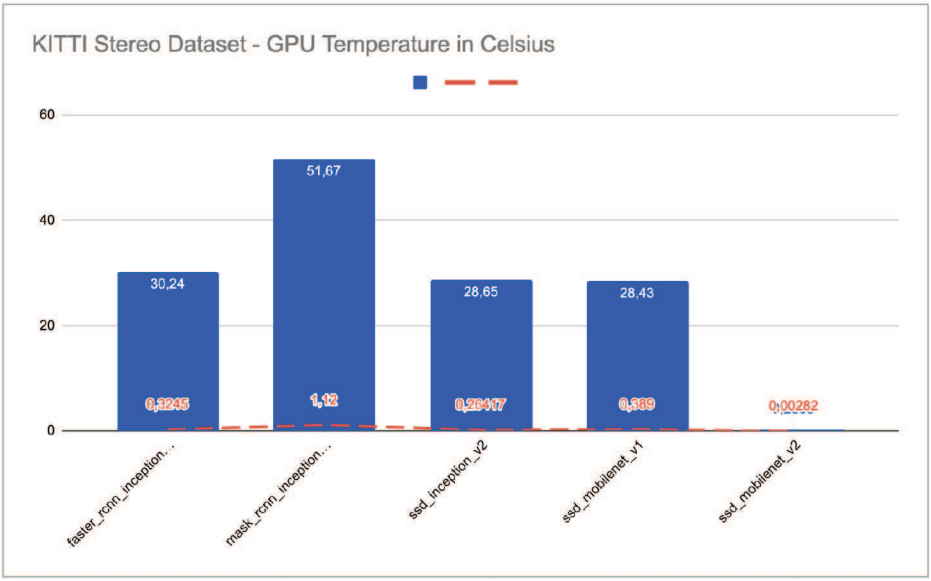


Figure 7.22: Benchmark resulting Average and Standard Deviation of GPU Temperature over KITTI Stereo Vision dataset

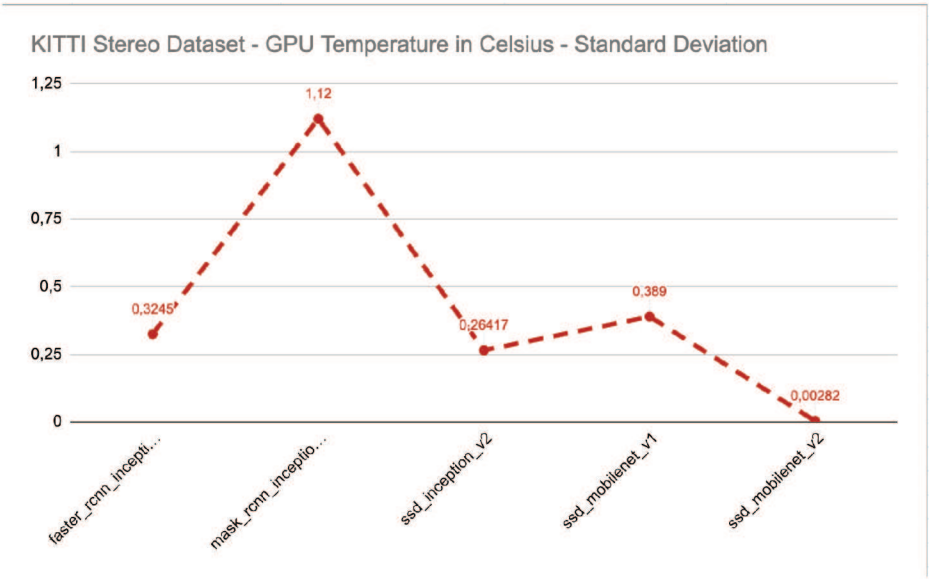


Figure 7.23: Benchmark resulting Standard Deviation of GPU Temperature over KITTI Stereo Vision dataset

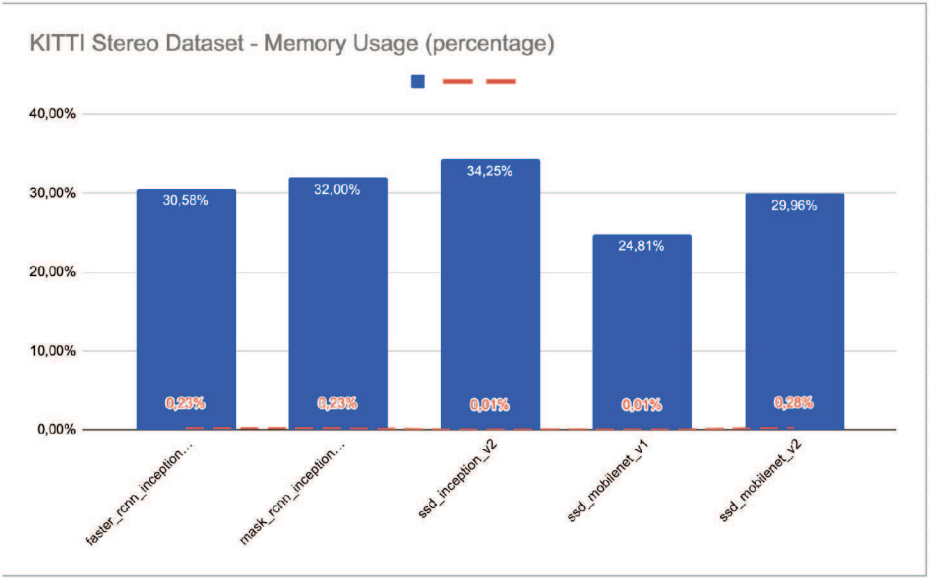


Figure 7.24: Benchmark resulting Average and Standard Deviation of GPU Memory use over KITTI Stereo Vision dataset

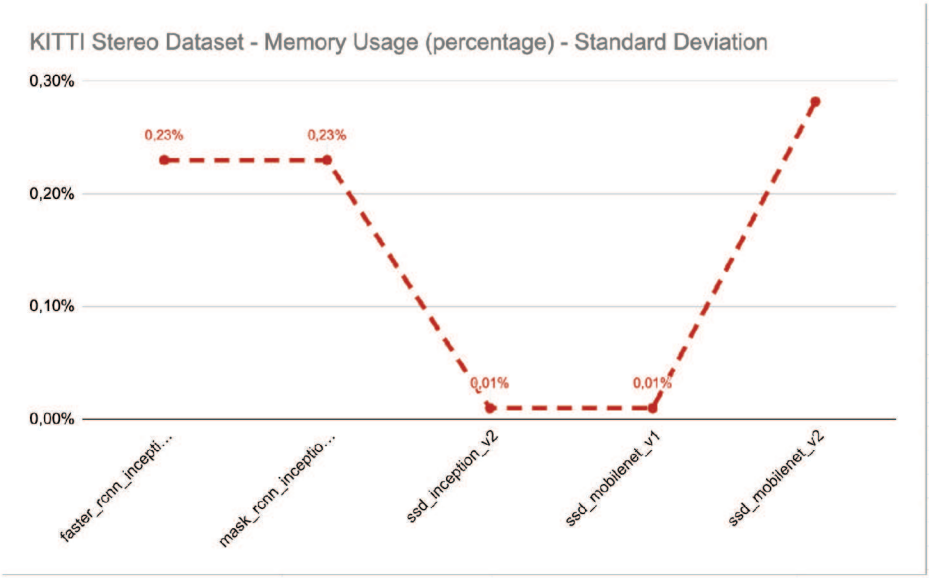


Figure 7.25: Benchmark resulting Standard Deviation of GPU Memory use over KITTI Stereo Vision dataset

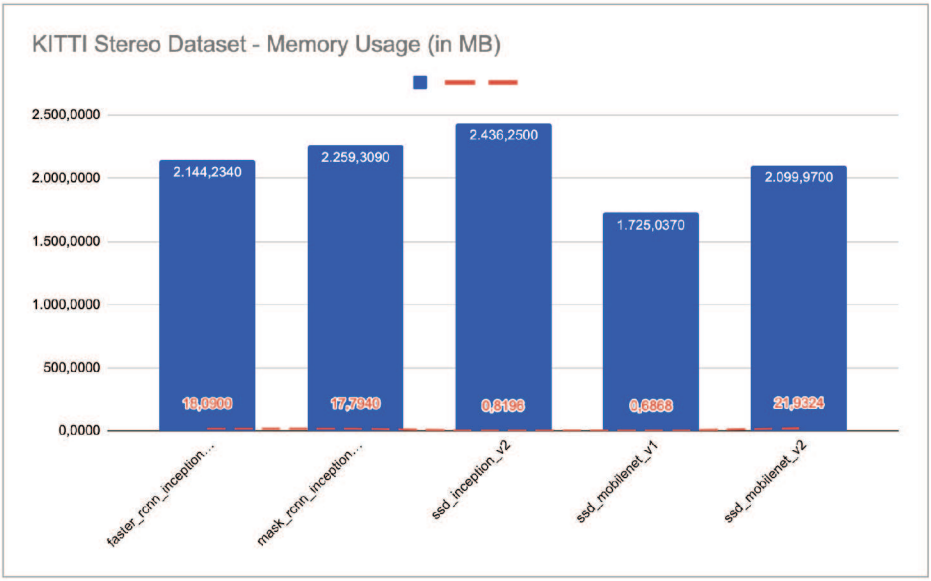


Figure 7.26: Benchmark resulting Average and Standard Deviation of GPU Memory use over KITTI Stereo Vision dataset

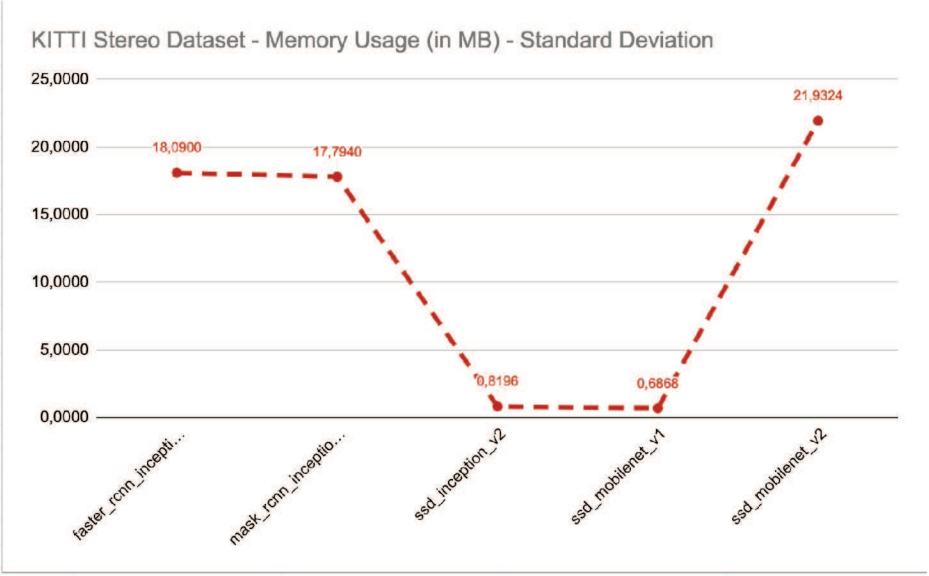


Figure 7.27: Benchmark resulting Standard Deviation of GPU Memory use over KITTI Stereo Vision dataset

7.8 Sample Results

The following images explore examples from the images extracted during the experiments. The expose detection boxes predicted from the models presented above for both datasets.

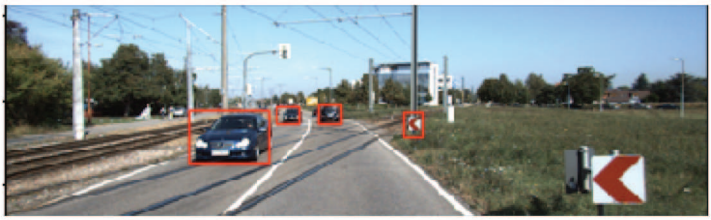


Figure 7.28: Detection results - Mask R-CNN over KITTI Stereo Vision

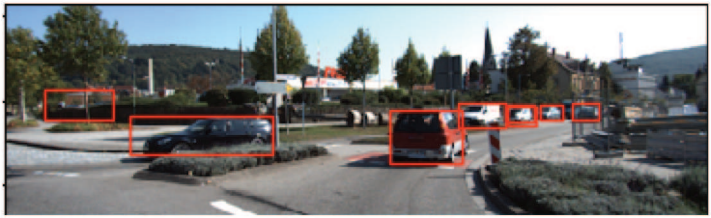


Figure 7.29: More detection results - Mask R-CNN over KITTI Stereo Vision

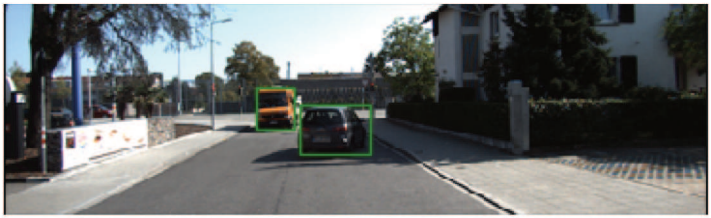


Figure 7.30: Detection results - Faster R-CNN over KITTI Stereo Vision

Figure 7.31: Detection results - SSD Inception V2 over KITTI Stereo Vision

Figure 7.32: More detection results - SSD Inception V2 over KITTI Stereo Vision

Figure 7.33: More detection results - Faster R-CNN over KITTI Stereo Vision

Figure 7.34: Detection results - SSD MobileNet V1 over Microsoft COCO

Figure 7.35: More detection results - SSD MobileNet V1 over Microsoft COCO

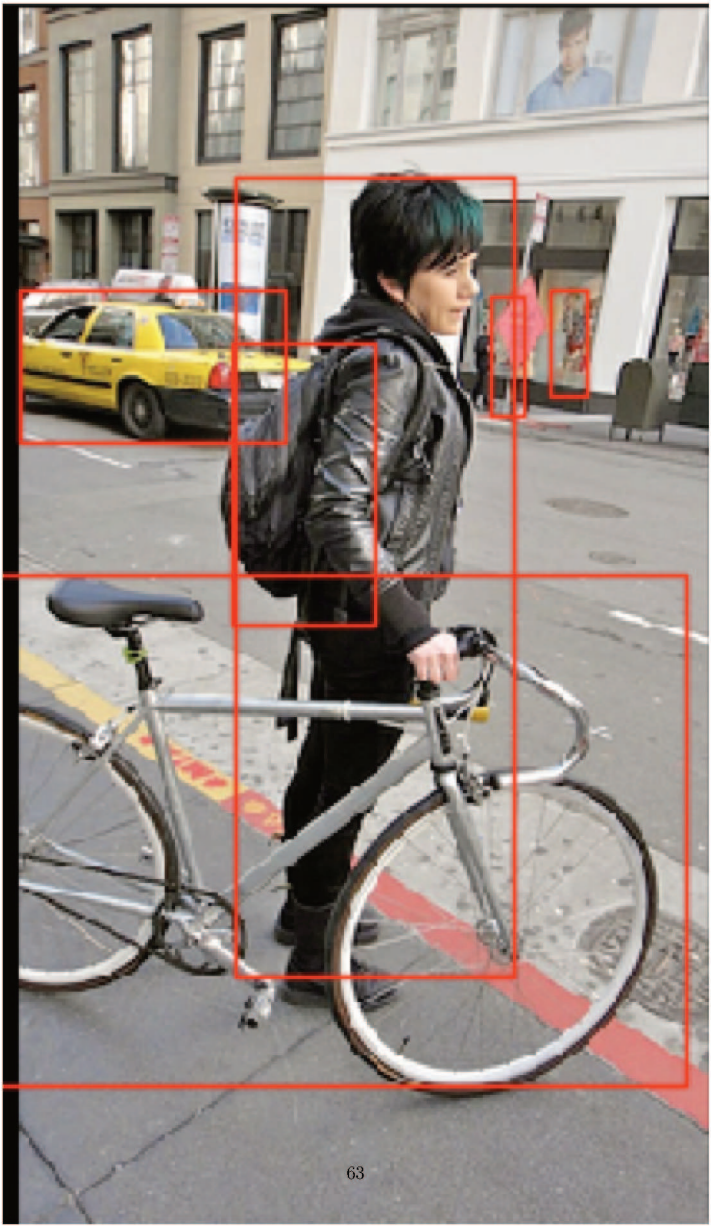


Figure 7.36: Detection results - SSD MobileNet V2 over Microsoft COCO

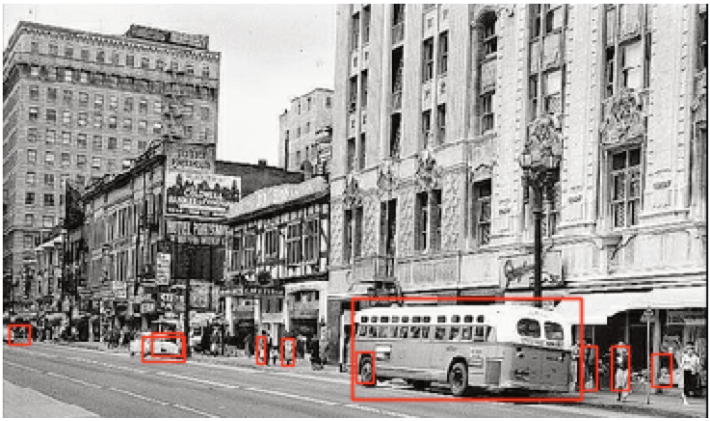


Figure 7.37: More detection results - SSD MobileNet V2 over Microsoft COCO

7.9 Results

The charts present a few insights over the collected data. It is possible to see that the SSD V1 implementation, over the MobileNets base network is the model that best-performs under normal circumstances on the Jetson TX2, at both datasets.

It is interesting to notice that, as presented by (31), the memory usage, with the exception of MobileNet implementations in which the focus is on lower resources usage, the necessity of memory to execute a given model is not much different from one model to the other.

Finally, the worst average inference time and, consequently, amount of frames per second (FPS) metrics goes to Mask R-CNN. Given the necessity of executing both network branches for object detection and mask segmentation makes the network not the best option to be used under such environment, in matter of inference performance.

Chapter 8

Conclusion and Future Work

In conclusion, the present work brings an overview of artificial intelligence and, mainly, deep learning fields with a focus on image recognition and the history behind the models and techniques present nowadays.

Beyond that, we explored how embedded hardware work with the new scenarios that AI brings to the table and how companies are developing task-specific boards to tackle new problems. After that, we worked with one of such hardware, called NVIDIA Jetson TX2, explaining how to flash, setup and use it with a hands-on tutorial.

Finally, we developed a benchmark to measure how well the Jetson platform behaves under real-world circumstances and explore it results, bringing more information to the decision-making process when deciding which architecture better fits the present hardware.

As future work, the application of the TensorRT framework on the benchmark, developed by the manufacturer, is expected and the execution of experiments under different datasets and with new models to enrich the data presented by the current work.

Bibliography

[1] Amazon jetpack. https://developer.nvidia.com/embedded/jetpack. Accessed: 2019-10-20.

[2] Cs231n convolutional neural networks for visual recognition. http://cs231n.github.io/. Accessed: 2019-10-20.

[3] Indoor positioning system amazon. https://indoo.rs/amazon. Accessed: 2019-10-16.

[4] Jetpack 2.3 with tensorrt doubles jetson tx1 deep learning inference. https://devblogs.nvidia.com/jetpack-doubles-jetson-tx1-deep-learning-inference/. Accessed: 2019-10-20.

[5] Jetpack doubles jetson inference performance. https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/. Accessed: 2019-10-20.

[6] Jetson nano ai computing. https://devblogs.nvidia.com/jetson-nano-ai-computing/. Accessed: 2019-10-20.

[7] Safety of laser products. part 1: Equipment classification, requirements and user's guide, 2001.

[8] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Abdi, L., and Meddeb, A. Driver information system: a combination of augmented reality, deep learning and vehicular ad-hoc networks. *Multimedia Tools and Applications 77*, 12 (Jun 2018), 14673–14703.

[10] Benenson, R., Omran, M., Hosang, J., and Schiele, B. Ten years of pedestrian detection, what have we learned? 613–627.

[11] Bobrow, D. Natural language input for a computer problem solving system.

[12] Bokovoy, A., Muravyev, K., and Yakovlev, K. Real-time vision-based depth reconstruction with nvidia jetson.

[13] Bradski, G. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[14] Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cudnn: Efficient primitives for deep learning. *CoRR abs/1410.0759* (2014).

[15] Cui, N. Applying gradient descent in convolutional neural networks. *Journal of Physics: Conference Series 1004* (04 2018), 012027.

[16] Daily, M., Medasani, S., Behringer, R., and Trivedi, M. Self-driving cars. *Computer 50*, 12 (December 2017), 18–23.

[17] Edwin, J., Greeshma, M., Haridas, M. T. P., and Supriya, M. H. Face recognition based surveillance system using facenet and mtcnn on jetson tx2. In *2019 5th International Conference on Advanced Computing Communication Systems (ICACCS)* (March 2019), pp. 608–613.

[18] Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics 36*, 4 (Apr 1980), 193–202.

[19] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* (Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010), Y. W. Teh and M. Titterington, Eds., vol. 9 of *Proceedings of Machine Learning Research*, PMLR, pp. 249–256.

[20] Han, Y., and Oruklu, E. Traffic sign recognition based on the nvidia jetson tx1 embedded system using convolutional neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (Aug 2017), pp. 184–187.

[21] He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask r-cnn. 2980–2988.

[22] Heredia, A., and Barros-Gavilanes, G. Video processing inside embedded devices using ssd-mobilenet to count mobility actors. In *2019 IEEE Colombian Conference on Applications in Computational Intelligence (ColCACI)* (June 2019), pp. 1–6.

[23] Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications.

[24] Ivakhnenko, A. G., and Lapa, V. G. *Cybernetic Predicting Devices*. 1965.

[25] Jackel, L., Boser, B., Graf, H., Denker, J., Lecun, Y., Henderson, D., Matan, O., Howard, R., and Baird, K. Vlsi implementations of electronic neural networks: An example in character recognition. pp. 320 – 322.

[26] Janocha, K., and Czarnecki, W. On loss functions for deep neural networks in classification. *Schedae Informaticae 25* (02 2017).

[27] Karagöz, M. F., Aktas, M., and Akgün, T. Cuda implementation of vibe background subtraction algorithm on jetson tx1/tx2 modules. In *2018 26th Signal Processing and Communications Applications Conference (SIU)* (May 2018), pp. 1–4.

[28] La, Q. D., Ngo, M. V., Dinh, T. Q., Quek, T. Q., and Shin, H. Enabling intelligence in fog computing to achieve energy and latency reduction. *Digital Communications and Networks 5*, 1 (2019), 3 – 9. Artificial Intelligence for Future Wireless Communications and Networking.

[29] Lai, Y., Ho, C., Huang, Y., Huang, C., Kuo, Y., and Chung, Y. Intelligent vehicle collision-avoidance system with deep learning. In *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)* (Oct 2018), pp. 123–126.

[30] Lee, D., Lee, J., Lee, S., and Kee, S. The real-time implementation for the parking line departure warning system. In *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)* (Sep. 2018), pp. 236–240.

[31] Li, J., Zhang, C., Cao, Q., Qi, C., Huang, J., and Xie, C. An experimental study on deep learning based on different hardware configurations. In *2017 International Conference on Networking, Architecture, and Storage (NAS)* (Aug 2017), pp. 1–6.

[32] Li, Q., Xiao, Q., and Liang, Y. Enabling high performance deep learning networks on embedded systems. In *IECON 2017 - 43rd Annual Conference of the IEEE Industrial Electronics Society* (Oct 2017), pp. 8405–8410.

[33] Li, Q., Zhu, J., Liu, T., Garibaldi, J., Li, Q., and Qiu, G. Visual landmark sequence-based indoor localization. In *Proceedings of the 1st Workshop on Artificial Intelligence and Deep Learning for Geographic Knowledge Discovery* (New York, NY, USA, 2017), GeoAI '17, ACM, pp. 14–23.

[34] Liu, S., Lv, S., Zhang, H., and Gong, J. Pedestrian detection algorithm based on the improved ssd. In *2019 Chinese Control And Decision Conference (CCDC)* (June 2019), pp. 3559–3563.

[35] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. Ssd: Single shot multibox detector. In *Computer Vision – ECCV 2016* (Cham, 2016), B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Springer International Publishing, pp. 21–37.

[36] M., E., L., V. G., I., W. C. K., J., W., and A., Z. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[37] Mao, H., Hu, Y., Kar, B., Gao, S., and McKenzie, G. Geoai 2017 workshop report: The 1st acm sigspatial international workshop on geoai: @ai and deep learning for geographic knowledge discovery: Redondo beach, ca, usa - november 7, 2016. *SIGSPATIAL Special 9*, 3 (Jan. 2018), 25–25.

[38] Menze, M., and Geiger, A. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2015).

[39] Newell, A., and Simon, H. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory 2*, 3 (Sep. 1956), 61–79.

[40] Nonavinakere, S., Aldana, J., Sisson, S., Cruz, E., and George, K. Memory aid device to improve face-name memory in individuals with alzheimer's disease. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)* (June 2018), pp. 353–354.

[41] Ramacher, U., Raab, W., Anlauf, J., Beichter, J., Hachmann, U., Brüls, N., Wesseling, M., Sicheneder, E., Männer, R., Gläss, J., and Wurz, A. Multiprocessor and memory architecture of the neurocomputer synapse-1. In *ICANN '93* (London, 1993), S. Gielen and B. Kappen, Eds., Springer London, pp. 1034–1039.

[42] Ren, S., He, K., Girshick, R., and Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell. 39*, 6 (June 2017), 1137–1149.

[43] Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* (1958), 65–386.

[44] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (June 2018), pp. 4510–4520.

[45] Sawada, J., Akopyan, F., Cassidy, A. S., Taba, B., Debole, M. V., Datta, P., Alvarez-Icaza, R., Amir, A., Arthur, J. V., Andreopoulos, A., Appuswamy, R., Baier, H., Barch, D., Berg, D. J., d. Nolfo, C., Esser, S. K., Flickner, M., Horvath, T. A., Jackson, B. L., Kusnitz, J., Lekuch, S., Mastro, M., Melano, T., Merolla, P. A., Millman, S. E., Nayak, T. K., Pass, N., Penner, H. E., Risk, W. P., Schleupen, K., Shaw, B., Wu, H., Giera, B., Moody, A. T., Mundhenk, N., Essen, B. C. V., Wang, E. X., Widemann, D. P., Wu, Q., Murphy, W. E., Infantolino, J. K., Ross, J. A., Shires, D. R., Vindiola, M. M., Namburu, R., and Modha, D. S. Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov 2016), pp. 130–141.

[46] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks 61* (2015), 85 – 117.

[47] Shakirov, V. V., Solovyeva, K. P., and Dunin-Barkowski, W. L. Review of state-of-the-art in deep learning artificial intelligence. *Optical Memory and Neural Networks 27*, 2 (Apr 2018), 65–80.

[48] Shihadeh, J., Ansari, A., and Ozunfunmi, T. Deep learning based image classification for remote medical diagnosis. In *2018 IEEE Global Humanitarian Technology Conference (GHTC)* (Oct 2018), pp. 1–8.

[49] STANDARD, A. N. *American National Standard for Safe use of Lasers Outdoors.* Orlando, FL, 2005.

[50] Sze, V. Designing hardware for machine learning: The important role played by circuit designers. *IEEE Solid-State Circuits Magazine 9*, 4 (Fall 2017), 46–54.

[51] Sze, V., Chen, Y.-H., Einer, J., Suleiman, A., and Zhang, Z. Hardware for machine learning: Challenges and opportunities. pp. 1–8.

[52] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision.

[53] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015), pp. 1–9.

[54] Tsung-Yi, L., Michael, M., Serge, B., James, H., Pietro, P., Deva, R., Piotr, D., and C, L. Z. Microsoft coco: Common objects in context.

[55] Turing, A. M. Computing machinery and intelligence. *Mind 59*, 236 (1950), 433–460.

[56] Widrow, B. *Adaptive "adaline" neuron using chemical "memistors.".* 1960.

[57] Yan, P., and Feng, Y. A hybrid gomoku deep learning artificial intelligence. In *Proceedings of the 2018 Artificial Intelligence and Cloud Computing Conference* (New York, NY, USA, 2018), AICCC '18, ACM, pp. 48–52.

[58] Yunchao, G., and Jiayao, Y. Application of computer vision and deep learning in breast cancer assisted diagnosis. In *Proceedings of the 3rd International Conference on Machine Learning and Soft Computing* (New York, NY, USA, 2019), ICMLSC 2019, ACM, pp. 186–191.

[59] Zhan, H., Dai, L., and Huang, Z. Deep learning in the field of art. In *Proceedings of the 2019 International Conference on Artificial Intelligence and Computer Science* (New York, NY, USA, 2019), AICS 2019, ACM, pp. 717–719.

[60] Zhang, Y., Bi, S., Dong, M., and Liu, Y. The implementation of cnn-based object detector on arm embedded platforms. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)* (Aug 2018), pp. 379–382.

[61] Zhaojin Zhang, Cunlu Xu, and Wei Feng. Road vehicle detection and classification based on deep neural network. In *2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS)* (Aug 2016), pp. 675–678.

Chapter 9

Attachments

9.1 Donation Letter

March 6, 2018

Aldo von Wangenheim
Federal University of Santa Catarina - UFSC
Rua Ana Maria Nunes, 275
Res. Araquãs, Casa 20
Córrego Grande
Florianópolis, SC 88.037-020 Brazil
TaxID 660.566.679-87

Re: Equipment Donations for Research Purposes

Dear Aldo,

NVIDIA's GPU Grant Program recently donated (1) Jetson TX2 Developer Kit for your research efforts. This donation is an unrestricted gift to support your research efforts and comes at no charge to you. NVIDIA has paid all shipping.

Best Regards,

Chandra Cheij
Academic Programs Manager

2701 San Tomas Expressway | Santa Clara, CA 95050 | **T** 408.486.2000 | **F** 408.486.2200 | www.nvidia.com

APÊNDICE B – Artigo

Desenvolvendo um benchmark para deep learning sobre a plataforma Jetson TX2

Lucas P. Bordignon[1], **Lucas de Almeida Fernandes** [1], **Aldo Von Wangenheim**[1]

[1]Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC)
88.040-900 – Florianópolis – SC – Brazil

lucas.bordignon@grad.ufsc.br, lucas_alfernandes@hotmail.com, awangenh@inf.ufsc.br

Abstract. *With the recent advances in the areas of visual navigation and deep learning, many areas are undergoing changes in the way they tackle and solve problems within them, such as visual navigation and autonomous vehicles. However, not only theoretical techniques have advanced, but also embedded systems with a focus on accelerating prototyping of new solutions are keeping pace with new changes. The present work focuses on developing a benchmark on embedded systems specialized in artificial intelligence algorithms, conducting a comparative study of state-of-the-art models on the Jetson TX2 platform, aiding decision making on the most common tools. suitable for implementing real applications in the field of deep learning, such as autonomous cars.*

Resumo. *Com os recentes avanços nas áreas de navegação visual e inteligência artificial (deep learning), diversas áreas sofrem mudanças quanto ao modo de atacar e solucionar os problemas nelas existentes, como é o caso da área de navegação visual e veículos autônomos. Porém, não só técnicas teóricas tem avançado como também sistemas embarcados com foco em acelerar a prototipação de novas soluções vem acompanhando as novas mudanças. O presente trabalho tem como foco desenvolver um benchmark sobre sistemas embarcados especializados em algoritmos de inteligência artificial, realizando um estudo comparativo de modelos alinhados ao estado da arte, sobre a plataforma Jetson TX2, auxiliando a tomada de decisão quanto as ferramentas mais apropriadas para implementação de aplicações reais sobre a área de deep learning, como é o caso de carros autônomos.*

1. Introdução

A industria automobilística, direta ou indiretamente, faz parte da rotina de diversos cidadãos do mundo. Atualmente, trilhões de veículos são conduzidos durante o ano ao redor do planeta, passando por diversas situações de alta complexidade e de risco aos condutores e passageiros [Daily et al. 2017].

Para contornar os possíveis riscos presentes na condução de automóveis em rodovias urbanas e rurais, diversos esforços vem sendo dedicados a projetos de pesquisa que utilizem visão computacional e inteligência artificial para auxiliar a reduzir o risco existente, bem como trazer ainda mais comodidade aos passageiros ou condutores dos mesmos.

Focando, como exemplo, na área de veículos autônomos, modelos que utilizam navegação visual passiva vem sendo preferidos se comparado com modelos de navegação ativa utilizando sensores como LIDAR. Tal configuração permite conjuntos de hardware embarcado mais baratos e menores fisicamente, além de utilizarem menor quantidade de energia para operação e serem considerados menos invasivos se comparados com técnicas de navegação ativa [Bokovoy et al. 2019], [STANDARD 2005], [slp 2001].

Para trabalhar com tais aplicações, diversos aceleradores em *hardware* vem sendo desenvolvidos pela indústria para que as necessidades de sistemas de tempo real e privacidade sejam atendidos. Porém, quando trabalhando com tais aceleradores, os mesmos devem possuir características específicas para que atendam a limitações de espaço, energia, largura de banda e outros [Sze 2017].

O objetivo principal do trabalho consiste em apresentar um estudo comparativo com foco em métricas de desempenho entre modelos de reconhecimento de imagens baseados em redes neurais convolucionais, alinhados ao estado da arte (*SSD* [Liu et al. 2016], *Mask R-CNN* [He et al. 2017] e *Faster R-CNN* [Ren et al. 2017], bem como suas variações de implementação), através da implementação de um *benchmark* sobre a plataforma embarcada Jetson TX2, da fabricante NVIDIA Corporation.

2. Fundamentação Teórica

2.1.

Redes Neurais Artificiais

Um dos precursores da técnica de redes neurais artificiais (ANNs) utilizada nos dias de hoje foi Rosenblatt, com base em seu trabalho apresentado no ano de 1958, o perceptron [Rosenblatt 1958].

A ideia principal do perceptron consiste em uma representação do sistema neurológico biológico que existe em uma grande parte de animais, como seres humanos. Após receberem impulsos de entrada, os neurônios desse sistema em formato de rede, são disparados e ativados caso sejam responsáveis pelo reconhecimento do devido dado de entrada, seja ele visual ou tátil. Assim, o modelo biológico permite que animais possuam tal sistema neurológico bem desenvolvido consigam reconhecer físicos e visuais.

A Figura 1 mostra uma abstração do modelo proposto. Sua arquitetura segue os seguintes passos:

- Recebe uma entrada binaria, como um estímulo externo ou decorrente de neurônios anteriores na rede;
- Cada valor de entrada é multiplicado por um valor, chamado de peso, que indica a intensidade de tal neurônio sendo ativado com relação a entrada passada;
- O modelo limita os resultados para ou 0 ou 1, permitindo que o valor de saída seja então binário. O valor representa, como na biologia, se o neurônio em questão foi ativado pelo estímulo ou não

Para cada neurônio da rede e cada valor de entrada, seja ele a entrada inicial da rede ou a saída de uma camada intermediária, existe um peso representando a intensidade do nodo dado o conjunto de entrada.

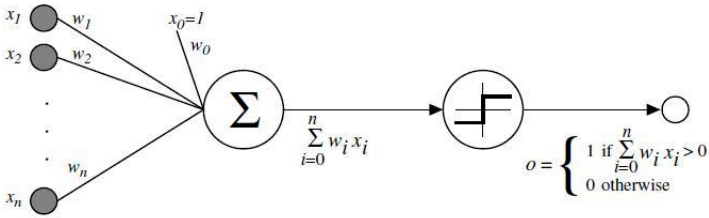


Figura 1. Representação do conceito do Perceptron, apresentado por Rosenblatt

Abstraindo para modelos mais recentes de redes neurais artificiais obtemos a seguinte formulação:

$$output = \sum_{i=0}^{n} w_i x_i + \theta$$

- n é o número de entradas da camada;
- x é o valor atual de entrada, dado o índice i;
- w é o valor do peso do neurônio, dado o índice i;
- θ é um valor constante;

2.2. Redes Neurais Convolucionais

Redes Neurais Convolucionais são similares a Redes Neurais Artificiais. Elas são constituídas de neurônios com pesos e constantes. Cada neurônio recebe dados de entrada, na forma de imagens e realiza uma multiplicação matricial, seguido opcionalmente por uma função não-linear de ativação. CNNs ainda trabalham sobre funções de custo em sua última camada [Fei-Fei et al.].

Uma das maiores mudanças com relação a redes tradicionais é a utilização de dados de entrada serem multidimensionais, permitindo que propriedades sejam aplicadas diretamente na arquitetura.

A base para modelos de CNNs parte de operações de convolução sobre os dados de entrada. Nessa operação, cria-se o que é chamado de filtro, uma matriz bidimensional na qual os parâmetros a serem treinados estão localizados.

É utilizada a técnica de janelas deslizantes dos filtros, de dimensões inferiores aos dados de entrada, realizando o produto matricial entre a o conjunto de dados e o filtro em questão. Dadas as dimensões, os filtros realizam a multiplicação em partes da imagem de entrada, resultando em mapas intermediários, como mostra a Figura 2.

Com o resultado da multiplicação de cada filtro, aplica-se um somatório e uma função de ativação, como em redes neurais artificiais, resultando na intensidade de ativação do filtro para a parte específica da imagem de entrada. Assim, é possível perceber que os filtros tendem a ser responsáveis pela ativação de partes de objetos ou cenas, aprendidos durante a fase de treinamento [Sermanet et al. 2013].

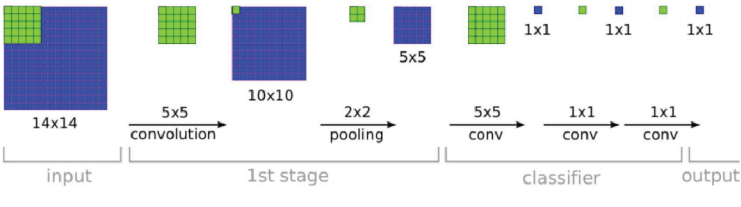


Figura 2. Funcionamento da etapa de convolução de uma CNN

3. Desenvolvimento

Baseando-se em modelos de CNN alinhados ao estado da arte, foi determinado a plataforma Jetson TX2, produzida pela empresa NVIDIA Corporation, como base para o presente trabalho. A Figura 3 apresenta o kit de desenvolvimento sobre a plataforma, utilizado para os experimentos.

A plataforma utilizada executa o sistema operacional Ubuntu 16.04 LTS. Com uma central de processamento (CPU) utilizando a arquitetura ARM64, os seguintes pacotes e dependências foram utilizados:

- JetPack 3.3 for Jetson TX2

  Conjunto de software fornecido pelo fabricante como kit de desenvolvimento, contendo *drivers* e auxiliares de compilação.

- CUDA 9.0 [Nickolls et al. 2008]

  Conjunto de ferramentas fornecido pela fabricante. Disponibiliza uma interface comum e otimização de operações para o chip gráfico presente na plataforma.

- OpenCV 3.4.6 [Bradski 2000]

  Biblioteca de código aberto para processamento de imagens. Utilizada como alternativa para execução e treinamento dos modelos e pré e pós processamento das imagens de validação.

- Tensorflow 1.14.0 [Abadi et al. 2015]

  Biblioteca de código aberto para treinamento e execução de fases de inferência das redes avaliadas, bem como serialização dos modelos para uso futuro.

Figura 3. NVIDIA Jetson TX2

Foi desenvolvido um *benchmark* de propósito geral sobre determinados componentes da plataforma acima descrita. A Tabela 1 apresenta as condições e os componentes metrificados.

A decisão sobre tais componentes baseia-se na alta busca por tais elementos em *benchmarks* existentes na literatura. Além disso, sabe-se que o desempenho de GPUs é um dos fatores de maior influência sobre um modelo baseado em redes neurais [Li et al. 2017].

Para trabalhar com o tópico, temperatura média de GPU e tempo de inferência, por exemplo, foram coletadas no processo, com o intuito fornecer uma visão mais clara das capacidades do sistema.

Métrica	**Mensurado por**
Frames por Segundo (FPS)	Agrupado por interação
Tempo de Inferência	Tempo médio de todas as interações
Temperatura da GPU	Tempo médio e desvio padrão
Uso de memória	Tempo médio e desvio padrão

Tabela 1. Elementos mensurados durante o *benchmark*

Para ser possível coletar dados que forneçam informação suficiente para uma tomada de decisão, decidiu-se por executar um conjunto de modelos baseados em redes neurais convolucionais já existentes na literatura. A Tabela 2 apresenta as arquiteturas utilizadas no projeto. Vale notar que os modelos utilizados foram coletados dos autores originais, treinados sobre os conjuntos de dados apresentados na tabela.

A decisão de utilizar modelos pré-treinados para a coleta de dados foi tomada, dado que métricas de performance de precisão das etapas de detecção, como intersecção sobre união (IoU), não são foco do presente trabalho e os autores dos modelos utilizados, em maioria, apresentam estudos comparativos com tais métricas.

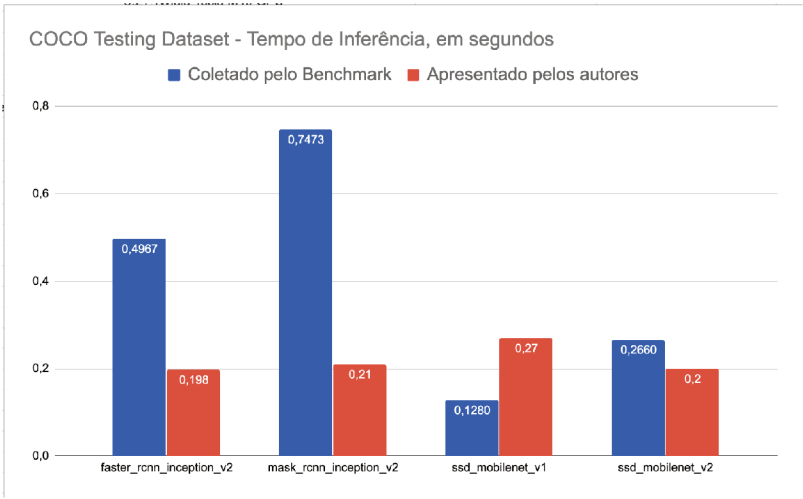


Figura 4. Comparativo de resultados apresentados por autores das arquiteturas

Arquitetura	Treinado em	Referência
Faster R-CNN	Microsoft COCO	[Ren et al. 2017]
Mask R-CNN	Microsoft COCO	[He et al. 2017]
SSD Inception V2	Microsoft COCO	[Szegedy et al. 2016], [Liu et al. 2016]
SSD Mobilenet V1	Microsoft COCO	[Howard et al. 2017], [Liu et al. 2016]
SSD Mobilenet V2	Microsoft COCO	[Sandler et al. 2018], [Liu et al. 2016]

Tabela 2. Modelos de *deep learning* utilizados para experimentação

4. Resultados

Após o desenvolvimento do algoritmo, o mesmo foi executado sobre dois conjuntos de dados popularmente utilizados no contexto de carros autônomos, chamados de Microsoft COCO [Lin et al. 2014] e KITTI Stereo Vision [Menze and Geiger 2015].

É interessante notar que, como apresentado na literatura, o uso de memória, com exceção de modelos baseados em *MobileNet* os quais possuem foco em redução de uso de recursos, a necessidade de uso de memória para execução dos modelos não difere muito entre si.

Como referência, A Figura 4 apresenta um comparativo entre os tempos de inferência coletados sobre o conjunto Microsoft COCO utilizando o *benchmark* e resultados coletados pelos autores das respectivas arquiteturas.

GPU	Consumo Energético
NVIDIA Jetson TX2 [Franklin 2017]	7.5/15W
NVIDIA Tesla K40 [CNET a]	235W
NVIDIA Tesla M40 [CNET b]	250W
Google Pixel 1	Fonte de carregamento de 18W

Tabela 3. Sistemas utilizados pelos autores para metrificação

Os autores do modelo *Faster R-CNN* utilizam a plataforma NVIDIA Tesla K40 para inferência. Os autores do trabalho *Mask R-CNN* realizam os experimentos sobre a GPU NVIDIA Tesla M40. Por último, ambos os modelos sobre a rede MobileNets são testados sobre o dispositivo Google Pixel 1.

A Tabela 3 apresenta a relação entre consumo energético utilizado pela plataforma Jetson TX2 se comparado aos sistemas testados pelos autores das arquiteturas.

Além disso, a Tabela 4 mostra um comparativo das métricas entre os modelos para ambos os conjuntos de dados. É possível traduzir as legendas do seguinte modo:

I Quadros por Segundo (FPS);
II Tempo de inferência, en segundos;
III Temperatura da GPU, em graus célsius;
IV Uso de memória total;

Modelo	I	II	III	IV
Faster R-CNN - COCO	2,013	0,4967	31,67	30,65%
Mask R-CNN - COCO	1,338	0,7473	36,86	30,54%
SSD Inception V2 - COCO	3,370	0,2960	28,20	32,15%
SSD Mobilenet V1 - COCO	7,806	0,1280	31,81	24,89%
SSD Mobilenet V2 - COCO	3,750	0,2660	29,54	28,24%
Faster R-CNN - KITTI	1,990	0,5010	30,24	30,58%
Mask R-CNN - KITTI	1,344	0,7456	51,67	32,00%
SSD Inception V2 - KITTI	3,220	0,3106	28,65	34,25%
SSD Mobilenet V1 - KITTI	4,290	0,2335	28,43	24,81%
SSD Mobilenet V2 - KITTI	3,488	0,2873	28,43	29,96%

Tabela 4. Comparativo dos resultados obtidos

5. Conclusão

Através do presente trabalho foi realizada a implementação de um *benchmark* comparativo utilizando o *framework* TensorFlow e as otimização da biblioteca cuDNN, com foco em desempenho de modelos alinhados ao estado da arte, sobre a plataforma NVIDIA Jetson TX2.

Uma análise comparativa de modelos de CNN foi realizada, sobre dois conjuntos de dados amplamente utilizados na comunidade acadêmica. O estudo permite extrair métricas e auxiliar no processo de tomada de decisão sobre quais componentes são otimizados para a plataforma testada, dados os componentes de *hardware* nela presentes.

Por fim, através dos resultados apresentados, é possível realizar um comparativo relacionado ao consumo energético da plataforma apresentada e sistemas especializados não embarcados. Embora a performance durante a fase de inferência possa ser inferior a aceleradores tradicionais, é notável a proximidade da capacidade de processamento de sistemas embarcados, dadas suas limitações.

Como trabalhos futuros, a aplicação do *benchmark* sobre modelos distintos, conjuntos de dados alternativos e com características distintas as presentes nos dados explorados (como vídeos e cenários diversos) e plataformas alternativas são os focos principais.

Referências

(2001). Safety of laser products. part 1: Equipment classification, requirements and user's guide.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems.

Bokovoy, A., Muravyev, K., and Yakovlev, K. (2019). Real-time vision-based depth reconstruction with nvidia jetson.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

CNET. Nvidia tesla k40 gpu computing processor specs. `https://www.cnet.com/products/nvidia-tesla-k40-gpu-computing-processor-tesla-k40-12-gb/`. Acesso em: 26 nov. 2019.

CNET. Nvidia tesla m40 gpu computing processor specs. `https://www.cnet.com/products/nvidia-tesla-m40-gpu-computing-processor-tesla-m40-24-gb/`. Acesso em: 26 nov. 2019.

Daily, M., Medasani, S., Behringer, R., and Trivedi, M. (2017). Self-driving cars. *Computer*, 50(12):18–23.

Fei-Fei, L., Karpathy, A., and Johnson, J. Cs231n convolutional neural networks for visual recognition. `http://cs231n.github.io/`. Acesso em: 20 out. 2019.

Franklin, D. (2017). Jetpack doubles jetson inference performance. `https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/`. Acesso em: 20 out. 2019.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

Li, J., Zhang, C., Cao, Q., Qi, C., Huang, J., and Xie, C. (2017). An experimental study on deep learning based on different hardware configurations. In *2017 International Conference on Networking, Architecture, and Storage (NAS)*, pages 1–6.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. pages 740–755.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). Ssd: Single shot multibox detector. pages 21–37.

Menze, M. and Geiger, A. (2015). Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6(2):40–53.

Ren, S., He, K., Girshick, R., and Sun, J. (2017). Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(6):1137–1149.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and Lecun, Y. (2013). Overfeat: Integrated recognition, localization and detection using convolutional networks. *International Conference on Learning Representations (ICLR) (Banff)*.

STANDARD, A. N. (2005). *American National Standard for Safe use of Lasers Outdoors*. Orlando, FL.

Sze, V. (2017). Designing hardware for machine learning: The important role played by circuit designers. *IEEE Solid-State Circuits Magazine*, 9(4):46–54.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2016). Rethinking the inception architecture for computer vision.

APÊNDICE C – Código Fonte

```
import numpy as np
import tensorflow as tf
import os
import wget
import random

from zipfile import ZipFile

from model import Model
from networks import paths


class Benchmark():
    _data = []
    _data_base_path = '../data/'

    def __init__(self, dataset='kitti'):
        tf.enable_eager_execution()

        self.download_dataset(dataset)

    def run(self):
        ''' Execute each model in a given selection of models '''

        for name, url in paths.items():
            model = Model(name, url)

            random.shuffle(self._data)

            for image in self._data:
                image_path = self._data_base_path + image
                image_raw = tf.read_file(image_path)
                image_tensor = tf.image.decode_image(image_raw)

                model.run(np.array(image_tensor.numpy()))

            model.export()

    def download_dataset(self, name):
        ''' Downloads the given dataset for benchmark '''

        print('[INFO] Checking dataset existence...')

        if (name == 'coco'):
            if (not os.path.exists('../data/coco')):
                print('[INFO] Downloading Microsoft COCO dataset:')
```

```
                url = 'http://images.cocodataset.org/zips/test2017.zip'
                compressed_data = wget.download(url)

                with ZipFile(compressed_data, 'r') as file:
                    filename = 'test2017'

                    file.extractall('../data/')
                    os.rename('../data/{}'.format(filename),
                              '../data/{}'.format(name))
                    os.remove('{}.zip'.format(filename))

            self._data_base_path = '../data/coco/'
            self._data = os.listdir(self._data_base_path)
        if (name == 'kitti'):
            if (not os.path.exists('../data/kitti')):
                print('[INFO] Downloading Kitti Stereo Vision dataset:')

                url = 'https://s3.eu-central-1.amazonaws.com/avg-kitti/' \
                    'data_scene_flow.zip'
                compressed_data = wget.download(url)

                with ZipFile(compressed_data, 'r') as file:
                    filename = 'data_scene_flow'

                    file.extractall('../data/')
                    os.rename('../data/{}'.format(filename),
                              '../data/{}'.format(name))
                    os.remove('{}.zip'.format(filename))

            self._data_base_path = '../data/kitti/testing/image_2/'
            self._data = os.listdir(self._data_base_path)
        print('[INFO] Images dataset loaded')


if __name__ == '__main__':
    benchmark = Benchmark()
    benchmark.run()

import numpy as np
import tensorflow as tf
import os
import wget
import tarfile
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

```
from networks import graph_path, tensors
from statistics import Statistics
from utils import resize


class Model():
    '''
        Represents a deep learning model, loading it's weights, graphs and
        collecting every metric which refers to it.
    '''
    def __init__(self, name, url):
        self._name = name
        self._url = url
        self._session = None
        self._statistics = Statistics(name)

        self.setup()

    def __del__(self):
        if (self._session):
            self._session.close()

    def run(self, image):
        '''
            Pre-process images and execute the inference for given model,
            collecting statistics for future analytics
        '''
        print('[INFO] Executing inference for {}'.format(self._name))
        print('[INFO] Input Image shape:', image.shape)

        input_tensor_name = tensors[self._name]['input']

        expanded_image = self.preprocess(image)

        self._statistics.start()
        output = self._session.run(tensors[self._name]['output'],
                                   feed_dict={
                                       input_tensor_name: expanded_image
                                   })
        self._statistics.end(output=output)

        # Just to collect images
        self.show_predictions(expanded_image[0], output[0][0])

        predictions_count = output[1]
```

```
        if (self._name == 'yolo_v3_coco'):
            predictions_count = len(predictions_count)
        else:
            predictions_count = predictions_count[0]

        print('[DEBUG] Forward pass finished! Predictions count:' +
              ' {}, elapsed time: {}'.format(
                  predictions_count,
                  self._statistics.timer.checkpoint['elapsed']))

    def setup(self):
        '''
            Setup Tensorflow session to execute a given model. The steps
            for setup are:
                - Download and extract pretrained models;
                - Setup the Session used during inference;
        '''
        print('[INFO] Creating session with model {}'.format(self._name))

        if (not os.path.exists(graph_path(self._name))):
            print('[INFO] Downloading {} weights & graph:'.format(self._name))
            weights_file = wget.download(self._url,
                                         out="{}.tar.gz".format(self._name))

            tar = tarfile.open(weights_file)
            tar.extractall(path='../model/')
            folder = tar.getmembers()[0]
            folder_name = folder.name.partition('/')[0]
            os.rename('../model/{}'.format(folder_name),
                      '../model/{}'.format(self._name))
            os.remove('{}.tar.gz'.format(self._name))

        with tf.gfile.FastGFile(graph_path(self._name), 'rb') as file:
            graph_def = tf.GraphDef()
            graph_def.ParseFromString(file.read())
            final_graph = tf.import_graph_def(graph_def, name="")

        self._session = tf.Session(graph=final_graph)
        print('[INFO] Graph for {} loaded'.format(self._name))

    def preprocess(self, image):
        ''' Preprocess image before the inference step '''
        processed = image

        if (np.average(processed) <= 1):
```

```
            # Must convert [0-1] RGB images to [0-255]
            processed *= 255

        resized_image = resize(processed)
        return np.expand_dims(resized_image, 0)

    def export(self):
        ''' Export collected statistics to /statistics/name.csv file '''
        self._statistics.export(base_path='../statistics/')

    def show(self, image_data):
        plt.imshow(image_data)
        plt.show()

    def show_predictions(self, image_data, boxes):
        height, width, _channels = image_data.shape
        _, image_axis = plt.subplots(1)

        plt.imshow(image_data)

        for box in boxes:
            ymin, xmin, ymax, xmax = box
            left, right, top, bottom = (int(xmin * width), int(xmax * width),
                                        int(ymin * height), int(ymax * height))

            patch = patches.Rectangle((left, top), right - left, bottom - top,
                                      linewidth=1, edgecolor='y',
                                      facecolor='none')

            image_axis.add_patch(patch)

        plt.show(block=False)
        plt.pause(4)
        plt.close()

    @property
    def graph(self):
        return self._session.graph

    @property
    def timer(self):
        return self._statistics.timer.checkpoints

paths = {
    'ssd_mobilenet_v1_coco': 'http://download.tensorflow.org/models' +
                             '/object_detection/ssd_mobilenet_v1_coco' +
```

```
                            '_2018_01_28.tar.gz',
    'ssd_mobilenet_v2_coco': 'http://download.tensorflow.org/models'
                            '/object_detection/ssd_mobilenet_v2_coco' +
                            '_2018_03_29.tar.gz',
    'ssd_inception_v2_coco': 'http://download.tensorflow.org/models' +
                            '/object_detection/ssd_inception_v2_coco'
                            '_2018_01_28.tar.gz',
    'faster_rcnn_inception_v2_coco': 'http://download.tensorflow.org/models' +
                                     '/object_detection/faster_rcnn' +
                                     '_inception_v2_coco_2018_01_28.tar.gz',
    'mask_rcnn_inception_v2_coco': 'http://download.tensorflow.org/models' +
                                   '/object_detection/mask_rcnn_inception' +
                                   '_v2_coco_2018_01_28.tar.gz',
    'yolo_v3_coco': ''
}

tensors = {
    'ssd_mobilenet_v1_coco': {
        'input': 'image_tensor:0',
        'output': [
            'detection_boxes:0',
            'num_detections:0',
            'detection_scores:0',
            'detection_classes:0',
        ],
    },
    'ssd_mobilenet_v2_coco': {
        'input': 'image_tensor:0',
        'output': [
            'detection_boxes:0',
            'num_detections:0',
            'detection_scores:0',
            'detection_classes:0',
        ],
    },
    'ssd_inception_v2_coco': {
        'input': 'image_tensor:0',
        'output': [
            'detection_boxes:0',
            'num_detections:0',
            'detection_scores:0',
            'detection_classes:0',
        ],
    },
    'faster_rcnn_inception_v2_coco': {
        'input': 'image_tensor:0',
```

```
        'output': [
            'detection_boxes:0',
            'num_detections:0',
            'detection_scores:0',
            'detection_classes:0',
        ],
    },
    'mask_rcnn_inception_v2_coco': {
        'input': 'image_tensor:0',
        'output': [
            'detection_boxes:0',
            'num_detections:0',
            'detection_scores:0',
            'detection_classes:0',
            'detection_masks:0',
        ],
    },
    'yolo_v3_coco': {
        'input': 'Placeholder:0',
        'output': [
            'yolo_v3_model/concat_10/axis:0',
            'yolo_v3_model/concat_10:0'
        ],
    },
}


def graph_path(path):
    return '../model/{}/frozen_inference_graph.pb'.format(path)

import csv
from timer import Timer
from collectors.cpu_collector import CPUCollector
from collectors.fps_collector import FPSCollector
from collectors.mem_collector import MemCollector


class Statistics():
    '''
        This class is responsible to collect data and checkpoints during
    fast-forward steps of networks. It collects the following piece of
    information:
        - Time data: As elapsed time, inference time, etc.
        - CPU/GPU data: As temperature for both CPU and GPU
        - Memory data: As process usage, swap usage, etc.
        - FPS data: As amount of images passed and predictions count
```

```
    '''
    def __init__(self, model_name):
        self._name = model_name
        self._collectors = {
            'timer': Timer(model_name),
            'cpu': CPUCollector(model_name),
            'memory': MemCollector(model_name),
            'fps': FPSCollector(model_name)
        }

    def start(self):
        self._collectors['timer'].start()

    def end(self, output=None):
        self._collectors['timer'].end()
        self._collectors['cpu'].collect()
        self._collectors['memory'].collect()

        if (output):
            self._collectors['fps'].collect(output)

    def export(self, base_path='./'):
        for metric in self._collectors:
            fields = self._collectors[metric].fields
            checkpoints = self._collectors[metric].checkpoints
            filename = base_path + self._name + '_' + metric + '.csv'

            with open(filename, 'w') as file:
                writer = csv.DictWriter(file, fieldnames=fields)

                writer.writeheader()
                list(map(lambda x: writer.writerow(x), checkpoints))

    @property
    def timer(self):
        return self._collectors['timer']

import time


class Timer():
    '''
        Class used to measure performance of the tested models. It stores
        a checkpoints list, containing all the information about the tracked
        times. Each item has the following format:
         - started_formatted: Human ready format to when the timer started;
```

```
            - started: Unix timestamp measured in nanoseconds;
            - ended: Unix timestamp measured in nanoseconds;
            - elapsed: Unix timestamp of the difference measured in nanoseconds;
        '''
        def __init__(self, name):
            self._name = name
            self._checkpoints = []
            self._current = 0

        def start(self):
            self.update_checkpoint({
                'started_formatted': time.ctime(),
                'started': time.time()
            }, override=True)

        def end(self):
            now = time.time()

            self._current += 1

            self.update_checkpoint({
                'ended': now,
                'elapsed': now - self.checkpoint['started']
            })

        def update_checkpoint(self, data, override=False):
            if (override):
                self._checkpoints.append(data)

            current_data = self.checkpoint

            self._checkpoints[self._current - 1] = {**current_data, **data}

        @property
        def checkpoint(self):
            return self._checkpoints[self._current - 1]

        @property
        def checkpoints(self):
            return self._checkpoints

        @property
        def fields(self):
            return ['started_formatted', 'started', 'ended', 'elapsed']

    from PIL import Image
```

```
def resize(image, output_size=(416, 416)):
    '''
        Resize a given image to expected output_size keeping its aspect
    ratio and adding zeros to fill to the desired format.
    '''
    raw_image = Image.fromarray(image)
    raw_image.thumbnail(output_size, Image.ANTIALIAS)
    resized_image = Image.new('RGB', output_size, (0, 0, 0))
    resized_image.paste(raw_image,
                        (int((output_size[0] - raw_image.size[0]) / 2),
                         int((output_size[1] - raw_image.size[1]) / 2)))

    return resized_image
```

```
import platform
import psutil


class CPUCollector():
    '''
        Class used to measure CPU performance of the tested models. It stores
        a checkpoints list, containing all the information about the tracked
        CPU usage.
    '''
    def __init__(self, name):
        self._name = name
        self._checkpoints = []
        self._current = 0

    def collect(self):
        if (platform.system() == 'Linux'):
            temperatures = psutil.sensors_temperatures()

            self.update_checkpoint({
                'gpu_temperature': temperatures['GPU-therm'][0].current,
                'cpu_temperature': temperatures['MCPU-therm'][0].current,
            }, override=True)

        self._current += 1

    def update_checkpoint(self, data, override=False):
        if (override):
            self._checkpoints.append(data)
```

```
            current_data = self.checkpoint

            self._checkpoints[self._current - 1] = {**current_data, **data}

        @property
        def checkpoint(self):
            return self._checkpoints[self._current - 1]

        @property
        def checkpoints(self):
            return self._checkpoints

        @property
        def fields(self):
            return ['gpu_temperature', 'cpu_temperature']

    class FPSCollector():
        '''
            Class used to measure FPS performance of the tested models. It stores
            a checkpoints list, containing the amount of predictions on the
            given input.
        '''
        def __init__(self, name):
            self._name = name
            self._checkpoints = []
            self._current = 0

        def collect(self, output):
            predictions = output[1]

            if (self._name == 'yolo_v3_coco'):
                predictions = len(predictions)
            else:
                predictions = predictions[0]

            self.update_checkpoint({'predictions': predictions}, override=True)

            self._current += 1

        def update_checkpoint(self, data, override=False):
            if (override):
                self._checkpoints.append(data)

            current_data = self.checkpoint

            self._checkpoints[self._current - 1] = {**current_data, **data}
```

```
    @property
    def checkpoint(self):
        return self._checkpoints[self._current - 1]

    @property
    def checkpoints(self):
        return self._checkpoints

    @property
    def fields(self):
        return ['predictions']

import platform
import psutil


class MemCollector():
    '''
        Class used to measure Memory usage of the tested models. It stores
        a checkjoints list, containing all the information about the tracked
        memory usage.

        Worth notice that all metrics stored all measured in bytes.
    '''
    def __init__(self, name):
        self._process = psutil.Process()
        self._name = name
        self._checkpoints = []
        self._current = 0

    def collect(self):
        full_info = self._process.memory_full_info()

        self.update_checkpoint({
            'percentage': self._process.memory_percent(),
            'process_total': full_info.uss,
            'ram_total': full_info.rss,
        }, override=True)

        if (platform.system() == 'Linux'):
            self.update_checkpoint({'swap_total': full_info.swap})

        self._current += 1

    def update_checkpoint(self, data, override=False):
```

```
        if (override):
            self._checkpoints.append(data)

        current_data = self.checkpoint

        self._checkpoints[self._current - 1] = {**current_data, **data}

    @property
    def checkpoint(self):
        return self._checkpoints[self._current - 1]

    @property
    def checkpoints(self):
        return self._checkpoints

    @property
    def fields(self):
        fields = ['percentage', 'process_total', 'ram_total']

        if (platform.system() == 'Linux'):
            fields.append('swap_total')

        return fields
```