INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA

IAN SCHMIEGELOW DANNAPEL

DESENVOLVIMENTO DE UMA SOLUÇÃO PARA CONTROLE AUTOMÁTICO DA ILUMINAÇÃO URBANA

FLORIANÓPOLIS, 2019

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SANTA CATARINA – CAMPUS FLORIANÓPOLIS
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE ENGENHARIA ELETRÔNICA

IAN SCHMIEGELOW DANNAPEL

DESENVOLVIMENTO DE UMA SOLUÇÃO PARA CONTROLE AUTOMÁTICO DA ILUMINAÇÃO URBANA

Trabalho de conclusão de curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina como parte dos requisitos para obtenção do título de Engenheiro Eletrônico.

Orientador:
Professor Dr. Renan Augusto Starke

FLORIANÓPOLIS, 2019

Ficha de identificação da obra elaborada pelo autor.

Dannapel, Ian Schmiegelow
DESENVOLVIMENTO DE UMA SOLUÇÃO PARA CONTROLE AUTOMÁTICO DA ILUMINAÇÃO URBANA / Ian Schmiegelow Dannapel ; orientação de Renan Augusto Starke. - Florianópolis, SC, 2019.
64 p.

Trabalho de Conclusão de Curso (TCC) - Instituto Federal de Santa Catarina, Câmpus Florianópolis. Bacharelado em Engenharia Eletrônica. Departamento Acadêmico de Eletrônica.
Inclui Referências.

1. Iluminação urbana inteligente. 2. Visão computacional. 3. Detecção de objetos. 4. Sistemas embarcados. I. Starke, Renan Augusto. II. Instituto Federal de Santa Catarina. Departamento Acadêmico de Eletrônica. III. Título.

DESENVOLVIMENTO DE UMA SOLUÇÃO PARA CONTROLE AUTOMÁTICO DA ILUMINAÇÃO URBANA

IAN SCHMIEGELOW DANNAPEL

Este trabalho foi julgado adequado para obtenção do título de Engenheiro Eletrônico e aprovado na sua forma final pela banca examinadora do Curso de Engenharia Eletrônica do Instituto Federal de Educação, Ciência e Tecnologia de Santa Catarina

Florianópolis, 12 / 12 / 2019

Banca Examinadora:

Renan Augusto Starke, Dr.

Reginaldo Steinbach, Me.

Samir Bonho, Me.

AGRADECIMENTOS

Ao meu orientador professor Renan, pela paciência, atenção, apoio e dedicação ao meu projeto de pesquisa.

A banca examinadora pela disponibilidade e atenção ao meu trabalho de conclusão de curso.

À minha mãe, Maria Ligia, que me apoia diaramente para manter-me firme em minha jornada.

À minha namorada Sara, pelo apoio e animação durante toda a minha graduação, você foi a motivação que eu precisei.

Aos meus colegas colaboradores do projeto de inovação do Desafio IFSC Ideias Inovadoras, principalmente aos meus amigos e parceiros de pesquisa Cleissom e Jhonatan, que estiveram comigo durante a graduação e fizeram parte da minha formação e da minha vida durante estes cinco anos.

Ao Instituto Federal de Santa Catarina e ao departamento de Engenharia Eletrônica que proporcionou uma graduação de muito aprendizado.

RESUMO

Com este trabalho desenvolve-se uma solução que identifica a presença humana para o controle automático da iluminação urbana. O controle visa otimizar o consumo de energia elétrica baseado na percepção da presença próxima da iluminação, que é reconhecida com um sensor que utiliza visão computacional para detecção objetos. A solução proposta consiste em um sistema embarcado *Jetson Nano*, especializado para aplicações com inteligência artificial, em que um fluxo de vídeo é processado em tempo real pela rede neural convolucional *SSD_Inception_V2*. O modelo treinado é convertido para a execução com o programa *Deepstream*, que descreve o fluxo de processamento da aplicação. Para testar o conceito da solução, é desenvolvido um protótipo que realiza a detecção de carros em miniatura e aciona lâmpadas baseado na posição relativa do objeto. O desempenho da execução da aplicação também é avaliada, considerando o tempo de processamento total de cada imagem. Por fim, a detecção é testada com condições próximas da realidade, em um estudo de caso com gravações de uma via local. A rede neural tem um tempo de execução de aproximadamente 85 ms na arquitetura do *Jetson Nano*, que utilizou praticamente 100% de sua capacidade de computação. No estudo de caso, o modelo obteve um *recall* de 83,8%, em que na maioria dos casos os objetos de interesse foram detectados corretamente.

Palavras chave: Iluminação urbana inteligente. Visão computacional. Detecção de objetos. Sistemas embarcados.

ABSTRACT

This work presents a solution that identifies the human presence for the automatic control of urban lighting . The control aims to optimize the power consumption based on the perception of human presence near of the illumination, which is recognized with a computer vision based sensor for object detection. The proposed solution consists of a embedded system Jetson Nano, specialized for artificial intelligence applications, in which a video stream is processed in real time by the *SSD_Inception_V2* convolutional neural network. The trained model is converted to execute with the Deepstream program, which describes the application processing pipeline. To test the concept of the solution, it is developed a prototype that detects miniature cars and activates lamps based on the relative position of the object. Application execution performance is also evaluated considering the total processing time of each image. Finally, detection is tested under near-reality conditions in a case study of a local with video recordings from a local road. The neural network has a runtime of approximately 85 ms on the Jetson Nano architecture, which has used almost 100% of its computing power. In the case study, the model obtained a recall of 83.8%, in which most cases the objects of interest were detected correctly.

Keywords: Smart urban lighting. Computer vision. Object detection. Embedded systems

SUMÁRIO

1 INTRODUÇÃO

A energia gasta anualmente com setor público de iluminação é de 15.443 GWh e representa 3,3% do consumo energético brasileiro (EMPRESA DE PESQUISA ENERGÉTICA, 2018), e por isso, mesmo uma pequena otimização, que reduza percentualmente este consumo pode causar um grande impacto. A solução proposta neste trabalho visa reduzir o consumo energético e intensidade da iluminação em momentos nos quais não há presença de pessoas próximas, otimizando o uso desse recurso.

Para diminuir o consumo de energia da iluminação, a intensidade de luz emitida de luminárias pode ser diminuida conforme a presença humana, através de um sensor baseado em inteligência artificial e um rede de comunicação das luminárias. Como a iluminação pública é de suma importância para os usuários da via pois ela proporciona visibilidade e segurança, portanto, o controle que não retire esses benefícios é um desafio.

Por isto, este trabalho desenvolve e analisa uma solução para detecção de presença de pedestres, veículos e que possa enviar informações do estado da via para um conjunto de luminárias. A detecção de presença assertiva exige interpretar situações diversas que os sensores de presença tradicionais não têm capacidade de reconhecimento. Por exemplo, o sensor de presença de alarme detecta somente movimento transversal em uma curta distância, ou seja, sua aplicação é voltada para ambientes internos.

Este sensor será baseado em uma rede neural para detecção e classificação de objetos em imagens de câmeras, e embora este método possa exigir muito processamento de dados, o reconhecimento de pessoas e veículos é bastante assertivo podendo ser amplamente replicado.

Para desenvolver esta solução, foi selecionado a plataforma embarcada *Jetson Nano*, que conta com um hardware dedicado para inteligência artificial e operação de até 8 redes neurais em paralelo em imagens de alta resolução. Por isso, o objetivo deste trabalho é utilizar as ferramentas disponíveis dessa plataforma para programar e testar a assertividade da inferência. E ainda, deseja-se validar a viabilidade deste tipo de sensor de presença.

A solução fará proveito de detecções em alto nível com rede neurais já treinadas e que identificam as classes necessárias para esta aplicação e será centrada na solução para a detecção da presença humana em rede neural e monitoramento de vídeo.

O conceito será testado em protótipo, em que a detecção de carros em miniatura são parâmetro para acionamento de lampadas através de comunicação sem fio. A posição do carro na imagem corresponde a qual das duas lâmpadas deverá estar acesa, a esquerda ou a direita.

Com a validação do conceito, será desenvolvido uma aplicação escalável com a rede convolucional *Single Shot Multibox Detector* (LIU, ANGUELOV, *et al.*, 2016). O desempenho do programa é testado tanto em tempo de execução quanto utilização dos recursos de hardware. Por fim, o sistema será utilizado em estudo de caso em uma rua local que avaliará quantos dos parâmetros de interesse não são detectados.

1.1 Justificativa

Atualmente, a iluminação urbana é acionada durante toda a noite, sem verificar em quais momentos essa iluminação é realmente necessária. Se houvesse essa verificação, a eficiência energética de um sistema de iluminação seria otimizado, diminuindo custos de operação e estendendo a vida útil, no caso de luminárias com tecnologia LED.

Com o sensor de presença inteligente, essa verificação teria um baixo índice de falsos negativos, ao contrário dos sensores de presença tradicionais. E embora este tipo de sensor possa exigir muito processamento devido a rede neural, existem diversas plataformas embarcadas que disponibilizam *hardware* apto para esse tipo de aplicação.

Além disso, a iluminação inteligente é uma tendência de mercado mundial devido ao surgimento de *Smart Cities*, que tem previsão de crescimento de 42% ao ano entre 2019 e 2023, gerando uma economia de energia equivalente a U$ 15 bilhões (SORRELL, 2019).

1.2 Definição do problema

Este trabalho limita-se a estudar e aplicar métodos de detecção para a presença humana no sistema embarcado *Jetson Nano*, utilizando uma rede neural convolucional com a arquitetura SSD (*Single Shot Multibox Detector*) já treinada.

1.3 Objetivo geral

O objetivo deste trabalho é realizar uma prova de conceito da solução em forma de maquete para validação inicial e desenvolver uma programa a fim de avaliar a solução em um possível cenário de aplicação.

1.3.1 Objetivos específicos

- Realizar a prova de conceito em forma de maquete.
- Aplicar a estrutura da rede convolucional para detecção de objetos no sistema embarcado.
- Testar a capacidade de execução de uma aplicação baseada no modelo *SSD_Inception_V2*.
- Utilizar gravações de uma via local para simulação da solução de um caso próximo da realidade.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentado a fundamentação teórica e ferramentas utilizadas para o desenvolver a metodologia de testes. Na Seção 2.1 é abordado o conceito da solução proposta e as soluções já desenvolvidas para a iluminação urbana inteligente. Na Seção 2.2, é revisada a fundamentação teórica de redes neurais utilizada no desenvolvimento deste trabalho e o motivo da seleção da estrutura *SSD_Inception_V2*.

O Item 2.3 aborda o conceito da computação de ponta, exemplificando a necessidade a seleção do sistema embarcado *Jetson Nano*, descrito na Seção 2.4. A Seção 2.5 faz o fechamento da fundamentação teórica, resumindo as ferramentas utilizadas na metodologia.

2.1 Iluminação urbana inteligente

Ainda não há uma definição universal da iluminação inteligente. Muitas vezes está relacionada apenas com a telegestão das luminárias, em que um operador realiza o monitoramento e controle remotamente.

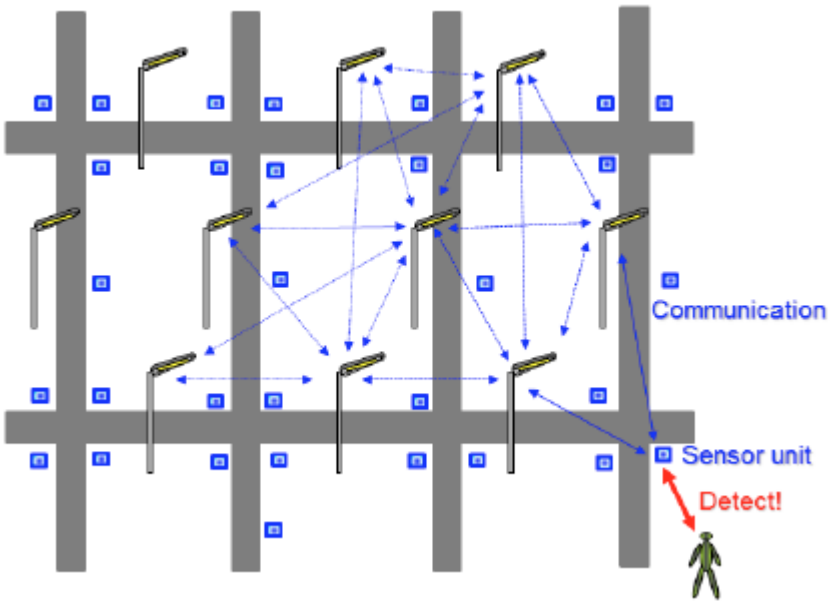
> A iluminação urbana inteligente é definida por um sistema inteligente que incorpora grupos de lâmpadas de iluminação pública que podem comunicar-se entre si e fornecer dados de iluminação a um concentrador local. O concentrador gerencia e transmite os dados relevantes, geralmente por meio de um modem celular digital, para um servidor seguro que captura os dados e os apresenta em uma interface de navegador da web. (BRUNO, FRANCO, *et al.*, 2012).

Neste trabalho, a iluminação inteligente visa realizar o controle automático da iluminação urbana, sem a necessidade da gestão remota. A comunicação com as luminárias deve se dar também no sentido concentrador luminária, dessa forma elas estarão aptas a receber comandos para aumento e diminuição da potência da luminária. O controle automático baseia-se na presença humana: se não há presença nas proximidades, as luminárias podem emitir apenas um percentual da luz ou até serem desativadas para economia energia. E quando houver a presença de ao menos uma pessoa, a intensidade deve voltar ao máximo.

Existem tentativas para reduzir o desperdício de energia que combinam sensor de claridade com sensor de movimento como realizado por Velaga e Kumar (2012). Nesse trabalho, a luz é ativada durante um tempo quando é detectado movimentação e está escuro. No entanto, geralmente o acionamento é tardio pois é necessário acionar a luminária antes que um carro ou uma pessoa se aproxime dela.

O trabalho de Fujii, Yoshiura, *et al.* (2013) propõe melhorar o acionamento com a comunicação das luminárias, de modo que um sensor de movimento acione um conjunto de luminárias. A luminária modular é composta por componentes para comunicação sem fio ZigBee (2020), para sensoriamento da luminosidade e para detecção de movimento, de forma que a detecção de movimento no período da noite aciona o conjunto de luminárias conforme a Figura 1:

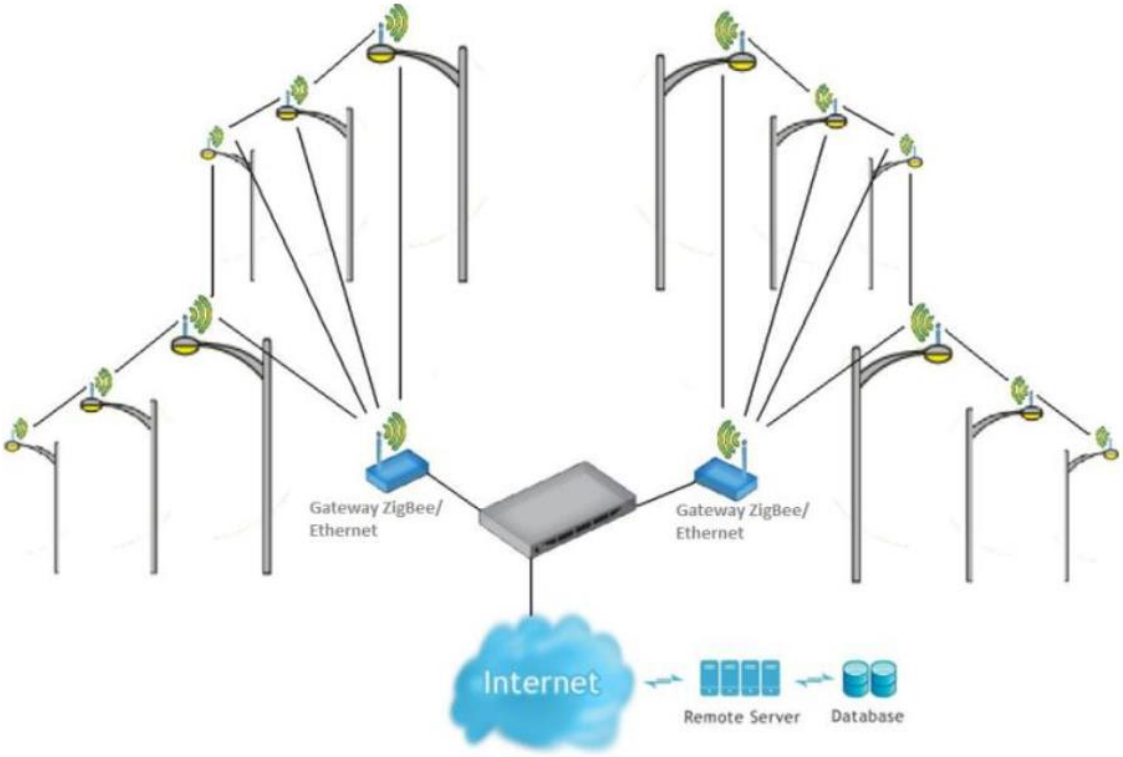
Figura 1 - Componentes para a iluminação pública inteligente



Fonte: Fujii, Yoshiura, *et al.* (2013)

O projeto de iluminação inteligente de Langner *et al.* (2015) também utilizou a tecnologia ZigBee para a intercomunicação e gestão da iluminação de um estacionamento em Curitiba – PR. A estrutura de comunicação do projeto está esquematizado na Figura 2, em que as luminárias se comunicam entre si e com o *gateway*, que está conectado a internet.

Figura 2 – Arquitetura Mesh para Iluminação Urbana

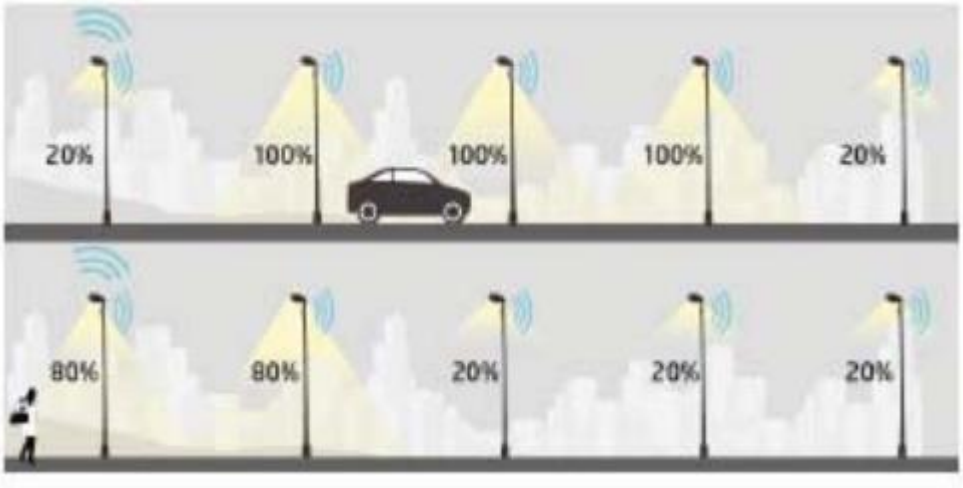


Fonte: Langner *et al.* (2015)

Nessa solução, o hardware proposto possibilita o controle remoto da intensidade de luz emitida da luminária com um sinal de 0V a 10V, além do acionamento automático ao anoitecer com um sensor fotoelétrico tradicional.

Outros trabalhos propõem a utilização de sensores para detecção de movimento, além da intercomunicação das luminárias. Por exemplo, Abdullah *et al.* (2018) utiliza a detecção de sensores de infravermelho para controlar a iluminação. O sistema consiste em um LDR (*Light Dependent Resistor*) para iniciar o sistema de iluminação, e utiliza o tempo de detecção entre dois sensores infravermelhos para medir a velocidade de um objeto. Quanto maior a velocidade do objeto, mais intensidade de luz é emitida. O conceito está ilustrado na Figura 3, em que a iluminação para um carro é 100% e para um pedestre é 80%.

Figura 3 – Diferentes configução de iluminação por objeto



Fonte: Abdullah *et al.* (2018)

O protótipo em maquete obteve uma redução de energia estimada em 42,45% numa situação teórica (Abdullah *et al.* 2018).

A implantação de uma iluminação inteligente de Ouerhani *et al.* (2016) também visa a economia de energia elétrica. Neste projeto, a comunicação das luminárias é sem fio, utilizando módulos ZigBee e o protocolo DALI (2020) para o alternar a intensidade de iluminação de cada ponto. A solução utiliza uma câmera que detecta atividade baseada em detecção de movimento, que configura a luminária para a intensidade máxima com movimentos e 30% quando não há. Esta solução foi testada ininterruptamente durante dois meses e obteve uma economia de energia de aproximadamente 56%. Ouerhani *et al.* (2016) ainda faz uma avaliação qualitativa do sistema de detecção para verificar sua confiabilidade e concluí que, a segurança oferecida é comparável a iluminação contínua máxima.

Geralmente, as soluções que procuram realizar o controle automático dependem de sensores de movimento, um sensor de baixa confiabilidade para ambientes externos. Por isso, para realizar este controle é de complexa execução é requerida a detecção assertiva de presença humana. Se essa detecção falhar, a segurança do usuário da via estará afetada. A solução proposta baseia-se em inteligência artificial, uma rede neural profunda para detecção e classificação de objetos em imagens para cumprir os requisitos de identificação de pedestres e carros.

As próximas seções fazem uma revisão das ferramentas e aplicações utilizadas na solução proposta e o motivo da seleção delas.

2.2 Inteligência artificial para detecção de objetos

O principal desafio da execução assertiva é a detecção da presença humana. Em uma rodovia existem diversas classes de objetos que precisam de iluminação, como por exemplo: pedestres, ciclistas, motociclistas, carros, ônibus, caminhões, etc. Evidentemente, existem ainda muitas subdivisões em cada um desses exemplos e, mesmo assim, nenhum desses poderia ficar no escuro devido a um erro de detecção.

Utilizando-se de sensores de presença tradicionais, não seria viável cobrir todos os pontos de uma rodovia pois seria necessária uma combinação de muitos dispositivos de detecção de movimento. Além de um curto alcance, usualmente em torno de 10 metros, sua precisão pode ser limitada pela interferência com distintas fontes de luz a sua volta.

Por causa da complexidade da detecção, propõe-se uma rede neural profunda para a detecção e classificação de objetos. A entrada da rede neural são imagens de vídeo de câmeras posicionadas estrategicamente para cobrir toda a via, utilizando-se um método com visão computacional.

2.2.1 Visão computacional

A visão computacional é um campo de estudo que tem como objetivo identificar imagens com a mesma percepção da visão humana. Atualmente, o trabalho da visão computacional é entender o conteúdo de dados em formato de imagem ou vídeo digitais em alto nível, buscando automatizar as tarefas que somente a visão humana poderia realizar.

Existem diversas áreas em que a visão computacional é aplicada, como por exemplo: inspeção automática placas de circuito impresso, detecção de massas em mamografias, carros autônomos, etc. Muitas vezes ela supera a capacidade humana em uma determinada tarefa e permite a interação dos computadores no mundo real.

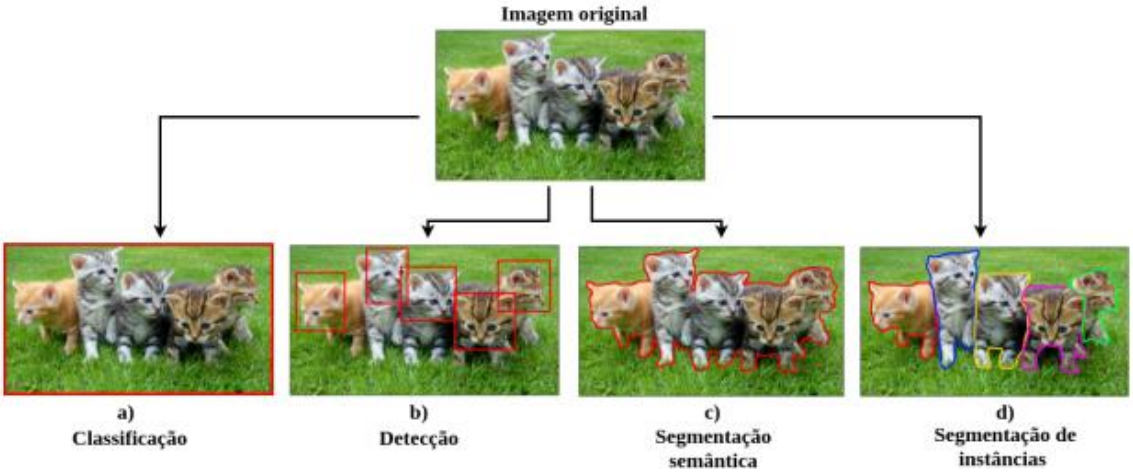
Martins (2019) divide as principais tarefas realizadas no campo da visão computacional em:

- Classificação: relaciona imagem dentro de um conjunto de possíveis classes conhecidas.

- Detecção de objetos: encontrar instâncias de objetos em uma imagem. Além da classificação dos objetos, é necessário também localizá-los dentro da imagem.
- Segmentação semântica: classificar individualmente os pixels de uma imagem dentre o conjunto de possíveis classes conhecidas.
- Segmentação semântica de instâncias: segmentação semântica que separa cada ocorrência de objetos dentro de uma classe. Cada instância de um classe é segmentada independentemente.

A Figura 4 ilustra os tipos de tarefa. Em a) é realizada a classificação da imagem como um todo. Em b) é realizada a detecção de cinco objetos, em c) é realizada a separação dos objetos do fundo da imagem e em d) é realizada a separação semantica de cada gatinho.

Figura 4 - Tarefas da Visão Computacional



Fonte: Martins (2019)

Cada uma dessas tarefas consiste em interpretar a informação contida na imagem original de maneira útil. Isso requer que a variável original (a imagem) seja condensada em uma variável de mais alto nível. Essa nova variável será um vetor com dimensão muito menor que a imagem, mas que é representativo das suas principais características (MARTINS, 2019).

Neste trabalho, é necessário detectar os objetos que são parâmetros para acionamento das luminárias urbanas, não é suficiente apenas realizar a classificação e a segmentação é desnecessária. Na visão computacional, existem técnicas descritoras que extraem informações estatísticas da imagem tais como cor, textura, histograma e a partir disso realizam a tomada de decisões. Porém estes descritores

são selecionados e adaptados para tarefas específicas, algo inviável para aplicação em larga escala. Para resolver este problema, surgem técnicas de redes neurais e redes convolucionais, capazes de aprender a solucionar diversos campos da visão computacional.

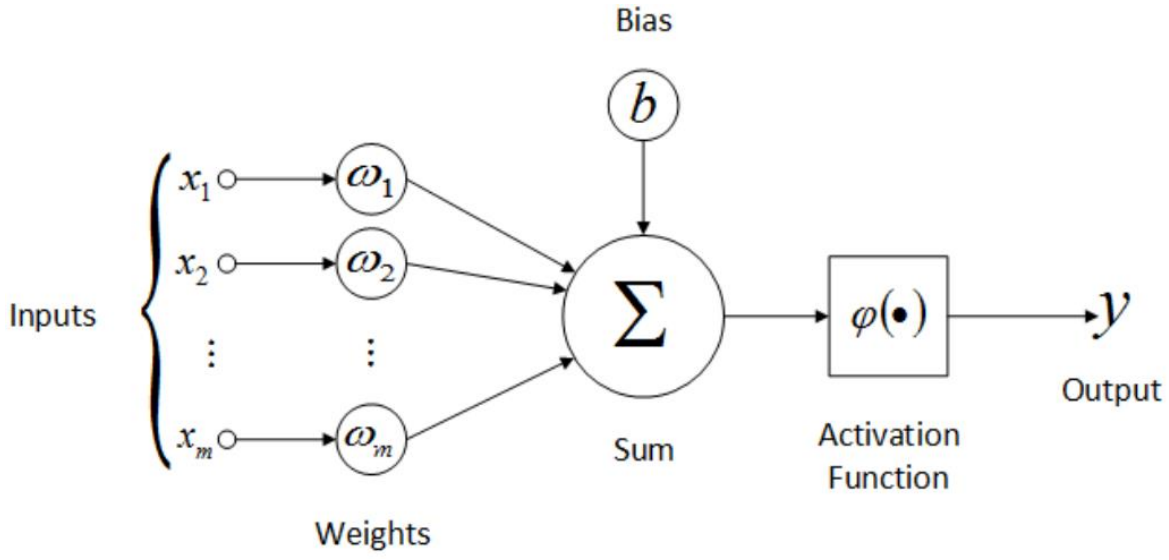
2.2.2 Redes neurais

Redes neurais artificiais são modelos computacionais com uma inspiração biológica baseada no cérebro humano, capazes de realizar funções complexas a partir da combinação de uma grande quantidade de elementos básicos (MARTINS, 2019).

Um modelo neural é extremamente parametrizável. Ele é capaz de aprender e realizar tarefas específicas, pois é montado a partir de da combinação de múltiplos neurônios artificiais, em que cada neurônio realiza apenas uma função matemática simples. Essa função é uma junção de uma combinação linear das entradas com uma função de ativação não linear. A simplicidade de um nó permite-lhe apenas realizar funções de baixa complexidade, mas um conjunto suficientemente grande de nós é capaz de aproximar qualquer função matemática (NIELSEN, 2015).

A Figura 5 ilustra a estrutura da função matemática de um neurônio artificial. O vetor de entrada **x** é multiplicada por um peso **W**, essa combinação linear produz o escalar que combinado com a *bias* **b** gera a saída **z**, que por sua vez gera a saída **y** através de uma função de ativação específica.

Figura 5 - Representação neurônio artificial

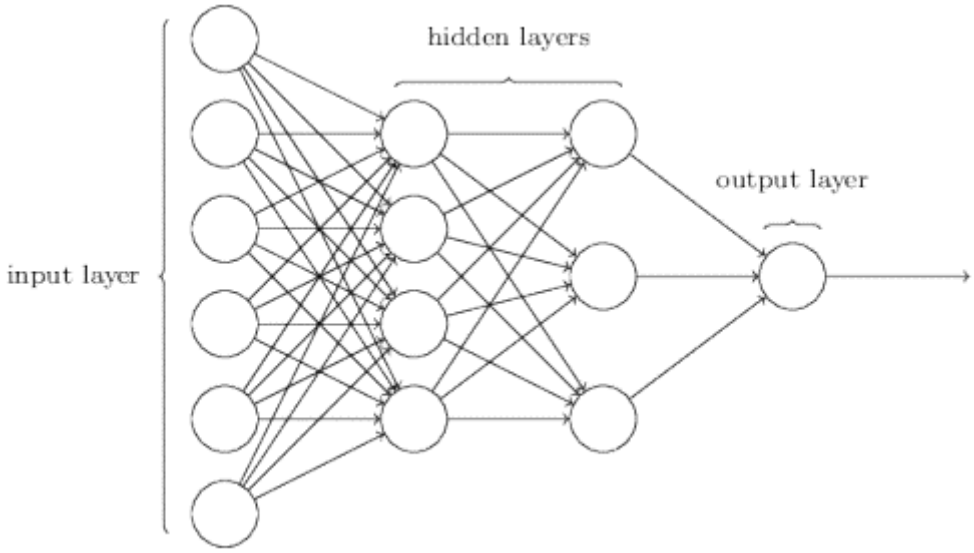


Fonte: Oliveira *et al.* (2017)

A função de ativação **σ()** é responsável por adicionar um componente não linear ao comportamento do nó. Sem essa função de ativação, todas os nós realizariam apenas funções lineares que não são capazes de produzir aproximações mais complexas. Ela também é muito importante na etapa de treinamento para gerar resultados similares com uma baixa variação dos pesos **W** e assim aproximar a tomada de decisão da rede a realidade.

Em uma rede neural, os nós ou neurônios são estruturados em camadas conforme mostrado na Figura 6. As características estruturais da rede, como número de nós, número de camadas, funções de ativação, etc., são conhecidas como hiper-parâmetros da rede.

Figura 6 – Rede neural totalmente conectada



Fonte: Nielsen (2015)

O objetivo da estrura da rede neural é aproximar uma função *f(x)* qualquer com o menor erro possível entre **y** real e **y** predito, Isso é realizado com ajustes dos parâmetros de cada nó através de um processo iterativo, em que os pesos são modificados para atingir um resultado satisfatório.

O processo iterativo não é tentativa e erro, ele consiste em algoritmos que podem diminuir o erro entre o valor real e o esperado de forma gradual através de uma retropropagação (*backpropagation*). Atualmente, a retropropagação é a base do sistema de aprendizado de redes neurais e resumidamente é uma expressão da derivada $\partial C/\partial w$ da função de custo C (erro entre o valor real e o esperado) a qualquer peso **W**. Dessa maneira, pode-se encontrar um ponto mínimo através de um gradiente

descendente na curva de aprendizado e calcular os pesos otimizados de todos os neurônios.

Muitas vezes, os pesos iniciais **W** são inseridos de maneira aleatória, isto implica que a mesma estrutura pode encontrar pontos de otimização diferentes dependendo dos pesos iniciais. Duas estruturas idênticas podem implicar em resultados reais diferentes devido a esse fato.

2.2.3 Redes neurais convolucionais

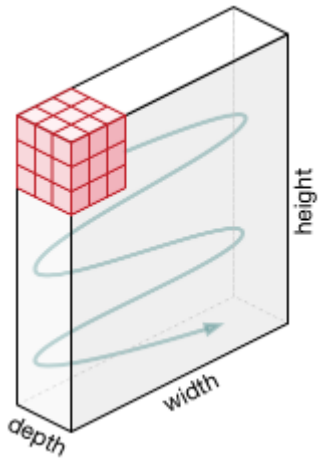
As redes neurais convolucionais surgiram para melhorar as redes neurais completamente conectadas, pois a propriedade de coêrencia de sinais como áudio (dependência temporal) e imagem (depedência espacial) é perdida na estrutura básica. A rede neural convolucional (*CNN – Convolutional Neural Network*) é amplamente utilizada em estruturas do estado da arte e permite processar imagens e ainda manter a coerência espacial do sinal com filtros convolucionais em uma ou mais etapas, uma operação mais adequada para o tratamento de imagens do que os neurônios artificiais (SAHA, 2018).

Uma *CNN* é um algoritmo de aprendizado profundo que pode captar uma imagem de entrada, atribuir importância (pesos e vieses aprendíveis) a vários aspectos/objetos da imagem e ser capaz de diferenciar um do outro. O pré-processamento necessário em uma rede convolucional é muito menor em comparação com outros algoritmos de classificação. Enquanto nos métodos primitivos os filtros são projetados manualmente, com treinamento suficiente, as *CNN* têm a capacidade de aprender essas características. A arquitetura é análoga à do padrão de conectividade dos neurônios no cérebro humano e foi inspirada pela organização do córtex visual. Neurônios individuais respondem a estímulos apenas em uma região restrita do campo visual, conhecida como campo receptivo. Uma coleção desses campos se sobrepõe para cobrir toda a área visual. (SAHA, 2018).

Em uma imagem há um grande número de informações relevantes que devem ser aproveitadas para a tomada de decisões, o objetivo é reduzir as imagens para um formato mais fácil de processar, sem perder recursos essenciais para obter uma boa previsão. Isso é importante quando é necessário projetar uma arquitetura que não seja apenas boa em aprender algumas características, mas também seja escalável para conjuntos de dados em massa.

Na camada convolucional, a imagem é processada através de um filtro (*Kernel*) que se descloca em toda imagem (Figura 7), produzindo uma nova informação reduzida a partir de informações vizinhas.

Figura 7 – Movimentação do *Kernel*



Fonte: Saha (2018)

A profunidade do *Kernel* é igual ao número de informações da imagem, por exemplo, tem profunidade três para uma imagem no domínio RGB. O objetivo da operação de convolução é extrair os recursos de alto nível, como bordas, da imagem de entrada. Muitos modelos de detecção possuem mais de uma camada convolucional e geralmente, o primeiro é responsável por capturar os recursos de baixo nível, como bordas, cores, orientação de gradiente, etc. Com camadas adicionadas, a arquitetura também se adapta aos recursos de alto nível, fornecendo uma rede com o entendimento saudável de imagens no conjunto de dados, semelhante ao método humano (SAHA, 2018).

2.2.4 Métricas de avaliação da detecção

Em detectores de objeto, é usual medir a precisão média (*Average Precision* - AP) das caixas de predição (*Bounding Boxes* - BB) em um conjunto de dados catalogado para realizar comparações de assertividade entre modelos. Cada *dataset* propõe um método de cálculo com algumas variações (ZOU, SHI, *et al.*, 2019), porém o conceito permanece inalterado na base.

A métrica computa um valor de zero a um de uma precisão para um *recall*. A precisão mede quão exatas são as predições e e*recall* mede quantas predições estão

corretas. As equações descritas em (1) e (2) descrevem a precisão (*precision*) e revocação (*recall)*, respectivamente:

$$Precision=TP/(TP+FP) \quad (1)$$

$$Recall=TP/(TP+FN) \quad (2)$$

Onde:

TP = positivos verdadeiros

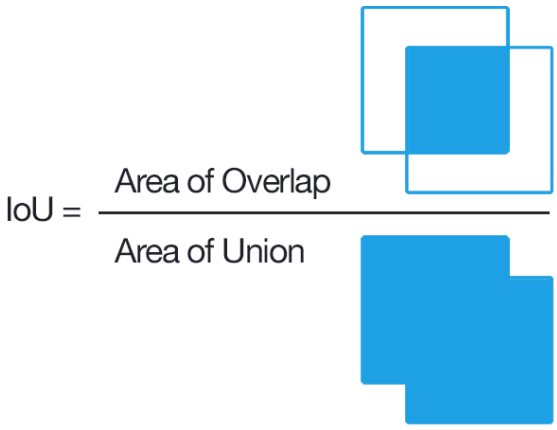
FP = Falsos positivos

FN = Falsos negativos

De acordo com as equações, para um determinado modelo de classificação, existe uma troca entre sua precisão e desempenho de *recall*. Para que a precisão seja alta, é necessário diminuir número de FP, e ao fazê-lo, o *recall* diminuirá. Da mesma forma, diminuir o número de FN aumentaria o *recall* e diminuiria a precisão. Muitas vezes, para casos de recuperação de informações e detecção de objetos, é necessário que a precisão seja alta (TAN, 2019).

Porém para determinar se um modelo tem bom desempenho esses índices não são suficientes e é necessário utilizar a métrica AP (*Average Precision*). Para estipular se uma detecção é uma TP, FP ou FN é utilizada o IoU (Intersecção sobre união), em que são comparadas a caixa de predição e a caixa real, ilustrado na Figura 8:

Figura 8 – Cálculo do IoU



Fonte: Tan (2019)

Resumidamente, um IoU acima de 0,5 significa um TP, um IoU menor que 0,5 significa um FP e um FN é caso aconteça um IoU maior que 0,5, porém com uma

predição incorreta em que classe detectada não corresponde a classe real. Com essa formalização, é possível calcular o AP de uma classe, que corresponde a equação (3):

$$AP = \int_0^1 p(r)\, dr \quad (3)$$

Onde:

p(r) é a curva do gráfico Precisão por *Recall*

Usualmente, detectores realizam a localização de diversos objetos, por isso a métrica geral é a média de todos AP's, abreviada por mAP.

2.2.5 Âncoras

Âncoras, também conhecidas como caixas delimitadoras padrão, são técnicas usadas por alguns detectores de objetos. Um conjunto de âncoras são inicializados em diversos tamanhos pré-definidos e são alocadas posições pré-definidas. A determinação de objetos é realizada a partir da classificação e ajuste fino de cada âncora.

O processo de detecção é feito gerando-se uma relação entre cada âncora na imagem original e a saída da rede. Cada âncora definida sobre a imagem original é considerada uma região de referência, e o objetivo da rede é classificar (definir se a âncora corresponde à um objeto) e ajustar (corrigir a posição da âncora em relação ao objeto) (MARTINS, 2019).

Na prática, o processo de detecção com âncora segue os seguintes passos (CHRISTIANSEN, 2018):

1. Milhares de "caixas âncora" ou "caixas delimitadoras " são criadas para cada preditor que representa uma localização, forma e tamanho do objeto especializado na previsão.

2. Para cada âncora, é calculada a Intersecção sobre união ou IoU.

3. Se a IoU mais alta for maior que 0,5, é feita a detecção na caixa âncora do objeto que resultou no IoU mais alto.

4. Caso contrário, se o IOU for maior que 0,4, a detecção verdadeira é ambigua e rede neural não deve aprender com esse exemplo.

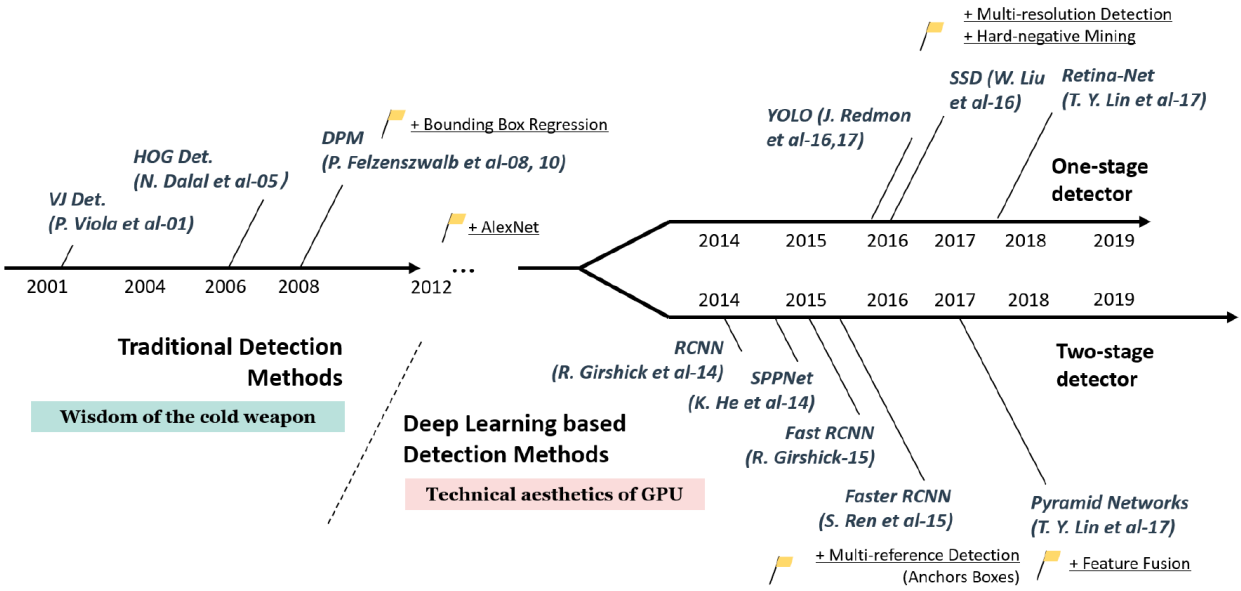
5. Se o IOU mais alto for inferior a 0,4, a caixa de predição deve prever que não há objeto.

2.2.6 Redes neurais para a detecção de objetos

A detecção de objetos é uma importante tarefa de visão computacional que lida com a detecção de instâncias de objetos visuais de uma determinada classe (como seres humanos, animais ou carros) em imagens digitais. O objetivo da detecção de objetos é desenvolver modelos e técnicas computacionais que fornecem uma das as informações mais básicas necessárias ao computador: quais objetos estão onde?

Segundo Zou, Shi, *et al.* (2019)., os detectores de objetos podem ser divididos em duas linhas. A primeira linha se refere a detectores de um estágio ou estágio único, e a segunda classificação se refere a detectores de dois estágios. A Figura 9 ilustra a evolução dos métodos para detecção de objeto citando os principais modelos publicados.

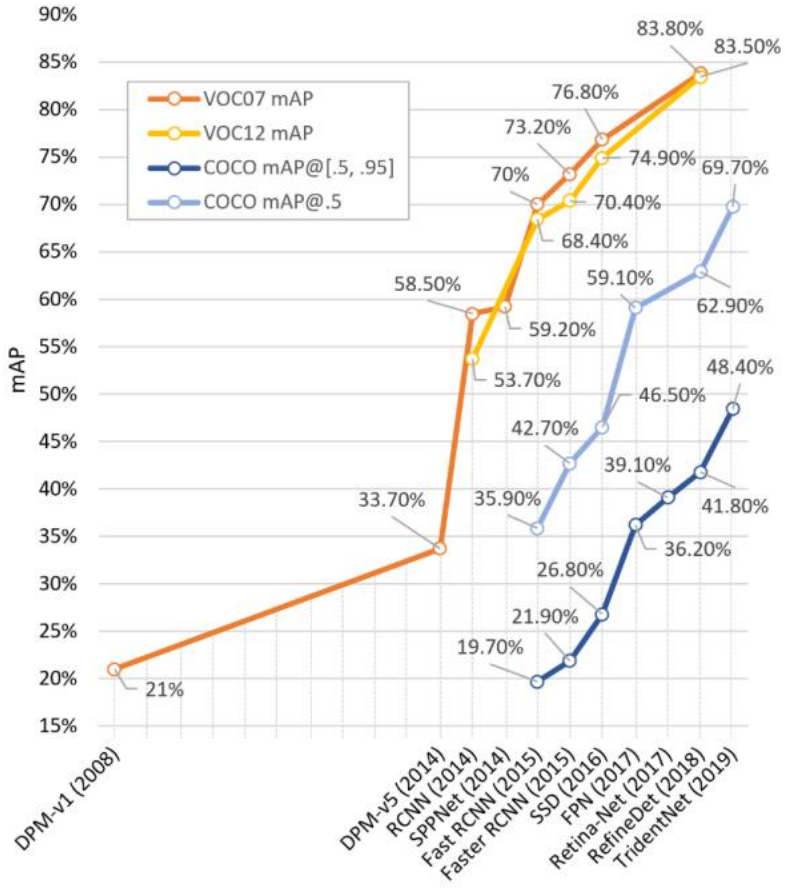
Figura 9 - Histórico da Detecção de Objeto



Fonte: Zou, Shi, *et al.* (2019)

A precisão da detecção evoluiu drasticamente na última decada. Na Figura 10 pode-se verificar que atualmente ela ultrapassa um mAP de 80%, propiciando a ampla difusão da técnica em muitas aplicações do mundo real como direção autônoma, visão robótica, vigilância por vídeo, etc.

Figura 10 – Precisão dos detectores de objeto

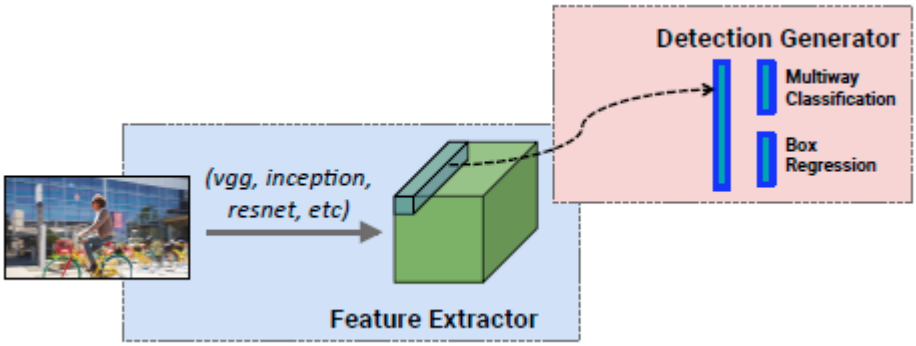


Fonte: Zou, Shi, *et al.* (2019)

Porém uma das dificuldades é realizar a comparação direta das diversas redes de detecção de objetos pois as abordagens variam, os testes são realizados em conjunto de dados diferentes e até computadors com especificações diferentes. Além disso, apenas a precisão não é parametro suficiente em aplicações que requerem tempo real ou sistemas com recursos limitados, é preciso verificar o tempo de execução e quantidade de memória utilizada, por exemplo, celulares *smartphone* requerem baixo uso de memória, enquanto que carros autônomos necessitam de uma desempenho em tempo real.

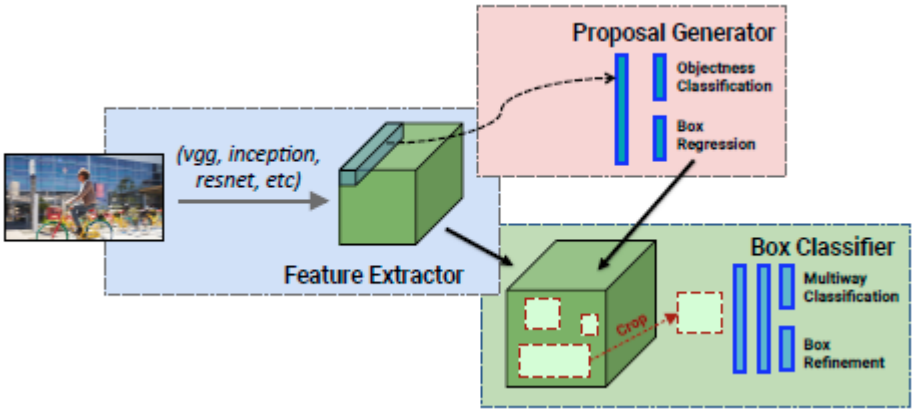
As comparações de redes neurais para detecção de objeto realizadas por Huang *et al.* (2017) mostram que os principais modelo de detecção convertem em uma metodologia, e que combina uma extrator de características (VGG, MobileNet, Inception V2) e uma meta arquitetura (Faster R-CNN, R-FCN, SSD) ilustrada na Figura 11 e Figura 12:

Figura 11 – Extrator de característica e meta-arquitetura SSD



Fonte: Huang *et al*. (2017)

Figura 12 - Extrator de característica e meta-arquitetura RCNN

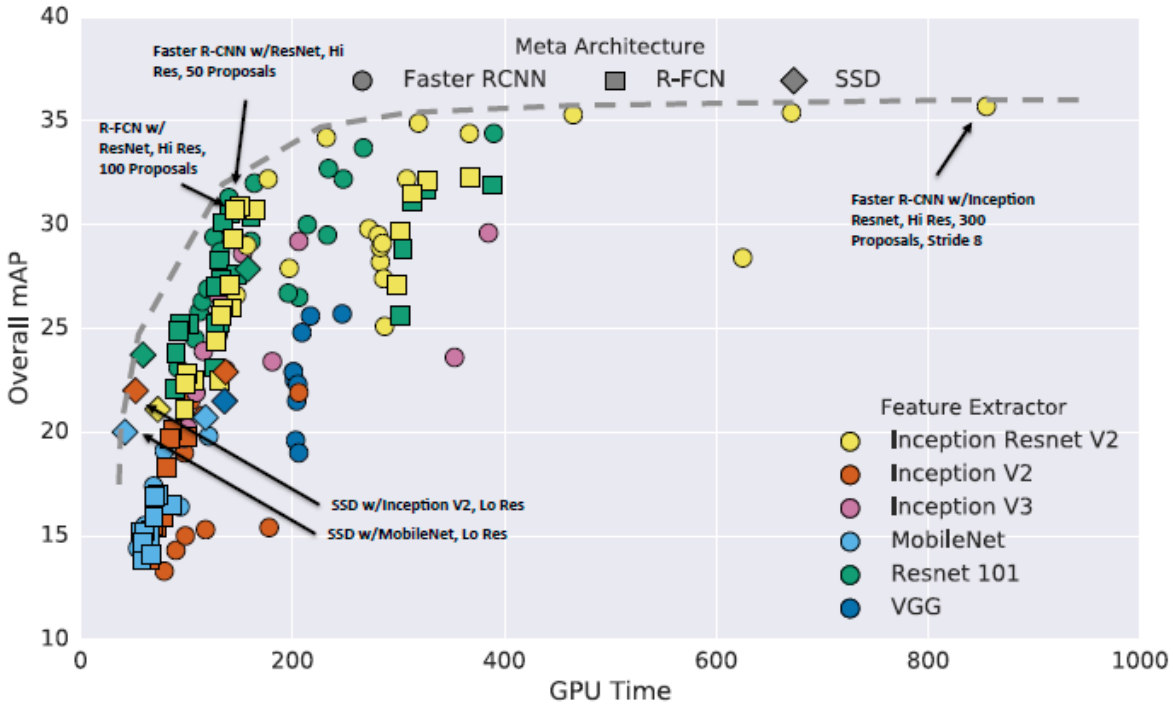


Fonte: Huang *et al*. (2017)

Na primeira etapa, o extrator tem como objetivo tranformar a imagem para realçar as características dela, a imagem transformada então é processada pela meta-arquitetura, que utiliza diversos métodos para detectar os objetos.

As comparações levam em consideração diversas combinações entre extrator, arquitetura e tamanho da imagem. Os resultados estão mostrado na Figura 13, um gráfico de dispersão que visualiza o mAP de cada uma das configurações de modelo, com cores representando extratores de características, e formas de marcador representando a meta-arquitetura.

Figura 13 - Precisão por tempo de execução



Fonte: Huang *et al.* (2017)

As arquiteturas R-FCN e SSD são mais rápidas, em média, que a Faster R-CNN, porém ela é mais precisa.Os resultados estão sintetizados na Tabela 1.

Tabela 1 – Sumário dos Modelos

Modelo	Conclusão	mAP (Precisão)
SSD + Mobilnet	Mais rápida	19,3
SSD + Inception V2	Mais rápida	22,0
R-CNN + Resnet 101	Equilibrada	32,0
R-FCN + Resnet 101	Equilibrada	30,4
Faster RCNN + Inception Resnet V2	Mais precisa	35,7

Fonte: Huang *et al.* (2017)

Baseado nessa comparação, o modelo SSD (*Single Shot Detector*) é uma das melhores alternativas para aplicação da visão computacional em sistemas embarcados, pois apesar de não possuir uma alta precisão, o tempo de execução é menor, permitindo o processamento contínuo de vídeo.

2.2.7 *Single Shot Multibox Detector* – SSD

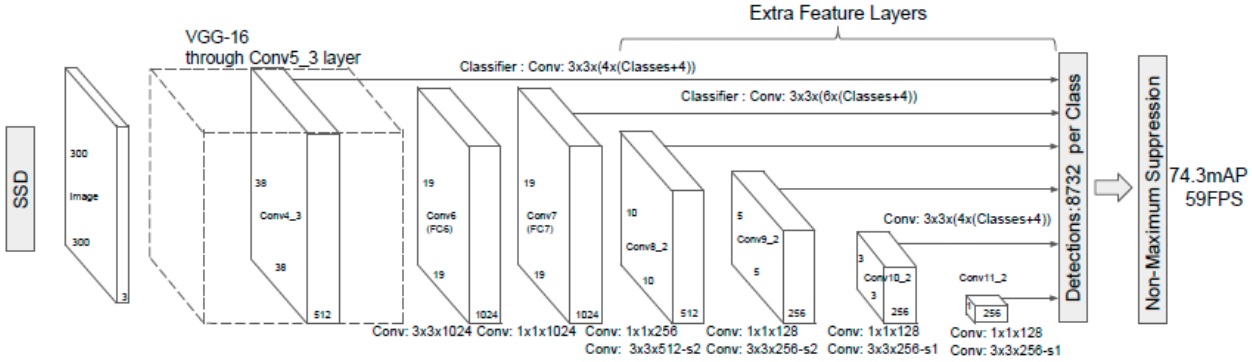
Os sistemas atuais de detecção de objetos do estado da arte são variações da seguinte abordagem: hipótese de caixas delimitadoras, reamostragem píxeis ou

recursos para cada caixa, e por fim aplicar um classificador alta qualidade. Esse fluxo prevaleceu como referência na detecção por bastante tempo, porém ele é computacionalmente custoso para sistemas embarcados e até mesmo para computadores de ponta, inviabilizando aplicações que requisitam tempo real. Uma métrica que avalia o tempo de execução é o QPS (quadros por segundo) e o modelo proposto por Ren *et al.* (2015), *Faster R-CNN*, alcança um mAP de 73,2% mas opera a apenas 5 QPS em uma *GPU NVIDIA Tesla K40*.

A arquitetura *SSD* proposta por Liu *et al.* (2016), é a primeira a não reamostrar pixeis ou características para hipótese de caixas delimitadoras que é tão precisa quanto as que fazem uso dessa abordagem, alcançando um mAP de 74,3% a 59 QPS no mesmo conjunto de dados e hardware de testes do modelo Faster R-CNN.

O modelo realiza a inferência de objeto em uma única tiragem, proporcionando um tempo de execução muito menor que as abordagens em que primeiramente são realizadas propostas de região e depois a classificação de cada região. A Figura 14 esquematiza a estrutura da rede neural *SSD*, em que a entrada é uma imagem de tamanho 300 por 300 píxeis de três níveis, em que cada nível corresponde a uma cor do domínio RGB. Na primeira etapa, a imagem é transformada pelo extrator de características VGG-16, e é realizada a detecção com classificadores convolucionais de diversos tamanhos, a fim de identificar objetos de variadas escalas. Por fim, é realizado o algorítmo de NMS (*Non-Maximum Suppression*), que define a melhor caixa delimitador de cada objeto identificado, com base nas diversas caixas resultantes dos classificadores convolucionais.

Figura 14 – Estrutura SSD



Fonte: Liu *et al.* (2016)

A abordagem SSD é baseada em uma rede convolucional de *feed-forward* que produz uma coleção de tamanho fixo de caixas delimitadoras e pontuações para a

presença de classe de objeto instâncias nessas caixas, seguida por uma etapa de supressão não máxima (NMS) para produzir as detecções finais. As primeiras camadas de rede são baseadas em uma arquitetura padrão usada para classificação de imagem de alta qualidade denominada rede base.

Em seguida, a estrutura auxiliar à rede para produzir detecções tem os seguintes recursos principais (LIU, ANGUELOV, *et al.*, 2016):

- **Detecção em múltiplas escalas**: As camadas convolucionais no fim da estrutura decrescem progressivamente e permitem detecção em multiplas escalas.
- **Preditores convolucionais para detecção**: Cada camada de característica (*feature layer*) existente da rede base pode produzir um conjunto fixo de previsões de detecção usando um conjunto de filtros convolucionais. Elas são indicadas na parte superior da arquitetura de rede SSD da Figura 14. Para uma camada de característica de tamanho $m \times n$ com número de canais p, o elemento básico para predição é um pequeno filtro convolucional (*kernel*) de tamanho $3 \times 3 \times p$ que produz ou uma pontuação para uma categoria ou um deslocamento relativo a uma caixa de predição padrão.
- **Caixas padrão e proporções**: Um conjunto de caixas delimitadoras padrão são associadas a cada célula do mapa de caracteristicas, para vários mapas de características na parte superior da rede. As caixas de predição padrão agrupam o mapa de características de maneira convolucional, para que a posição de cada caixa em relação à célula correspondente seja fixa. Em cada célula do mapa de características, é previsto o deslocamento relativo à forma da caixa padrão na célula, bem como as pontuações por classe que indicam a presença de uma instância de classe em cada uma dessas caixas. Isto permite discretizar com eficiência o espaço de diversas possíveis formas na saída.

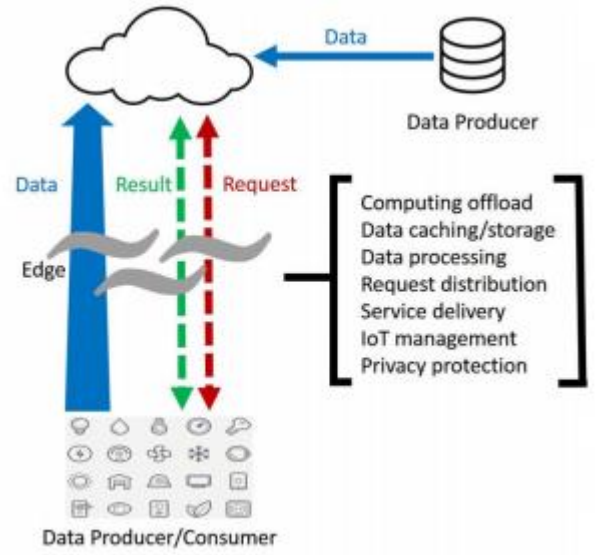
2.3 Computação de ponta

Segundo Shi, Cao, *et al.* (2016) colocar todas as tarefas de computação na nuvem provou ser uma maneira eficiente de processamento de dados, uma vez que o poder de computação na nuvem ultrapassa a capacidade de sistemas que estão na

ponta. No entanto, com a crescente quantidade de dados gerados na ponta, a velocidade dos dados transporte está se tornando o gargalo do paradigma da computação baseada em nuvem.

A computação em ponta refere-se às tecnologias que permitem que a computação seja executada na ponta da rede, o mais próximo possível da geração dos dados. A Figura 15 esquematiza a computação de ponta, em que sistemas são produtores de dados, porém também necessitam de informação da nuvem e por isso o fluxo de computacional é bidirecional.

Figura 15 - Paradigma de Computação de Ponta



Fonte: Shi, Cao, *et al.* (2016)

No paradigma da computação de ponta, os sistemas não são apenas consumidores de dados, mas também são produtores e não estão limitados apenas solicitar serviço e conteúdo da nuvem. Dessa maneira, um sistema na ponta também executa as tarefas de computação da nuvem. A ponte pode executar descarregamentos de computação, armazenamento de dados, armazenamento em cache e processamento, além de distribuir serviços de solicitação e entrega da nuvem para o usuário. Com esses trabalhos na rede, a própria ponta precisa ser bem projetada para atender aos requisitos de maneira eficiente nos serviços, como confiabilidade, segurança, e proteção de privacidade. (SHI, CAO, *et al.*, 2016)

A solução proposta necessita realizar a detecção em tempo real, caso contrário as luminárias não serão ativadas a tempo, afetando a segurança dos usuários da via. O requisito de tempo real dificilmente será atendido com processamento em nuvem pois necessita de constante comunicação com alta taxa de transmissão de imagens

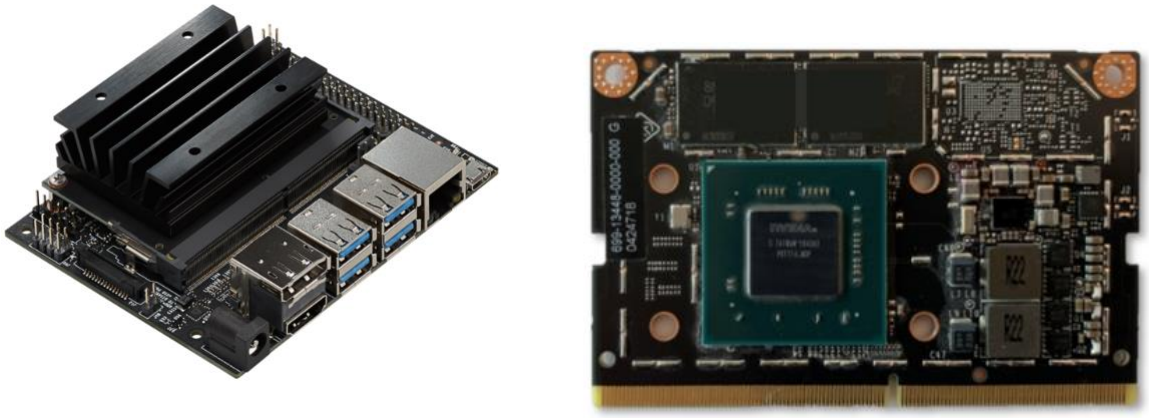
de vídeo em alta qualidade. Além disso, qualquer falha de comunicação irá causar a interrupção do funcionamento de todo o sistema de controle e iluminação.

Por isso, é necessário realizar processamento e inferência na ponta, mais próxima do sensoriamento, que nesse caso seriam câmeras de monitoramento. Existem módulos para sistemas embarcados que podem viabilizar este tipo de estrutura, como por exemplo, a plataforma embarcada *Jetson Nano*, que conta com hardware para aceleração de decodificação de fluxos de vídeo e para a execução de redes neurais.

2.4 NVIDIA Jetson Nano

Anunciado no início de 2019, o módulo NVIDIA Jetson Nano, exposto na Figura 16, é um computador para aplicações de Inteligência Artificial. A capacidade de processamento e facilidade para desenvolvimento de aplicações foram os fatores para a seleção deste módulo para prova de conceito da solução proposta.

Figura 16 – Kit de desenvolvimento (esquerda) e módulo Jetson Nano



Fonte: Franklin (2019)

Com preço oficial fixado em U$ 99, aproximadamente R$ 400 em conversão direta (Out/2019), o módulo conta com um ambiente Linux completo baseado no Ubuntu e tem total compatibilidade com estruturas de IA da NVIDIA, o que simplifica a implantação de aplicações de inferência baseadas em AI.

Ele foi projetado para reduzir o tempo geral de desenvolvimento e levar os produtos ao mercado mais rapidamente, reduzindo o tempo gasto no projeto de hardware, teste e verificação de um sistema de IA complexo. O Jetson Nano oferece visão computacional em tempo real e inferências em uma ampla variedade de

modelos complexos de redes neurais profundas. Esses recursos permitem robôs autônomos com vários sensores, dispositivos de IoT com análise de ponta inteligente e sistemas avançados de IA.

A tabela a seguir mostra as especificações técnicas do módulo:

Tabela 2 - Especificações Técnicas Jetson Nano

CPU	64-bit Quad-core ARM A57 @ 1.43GHz
GPU	128-core NVIDIA Maxwell @ 921MHz
Memória	4GB 64-bit LPDDR4 @ 1600MHz | 25.6 GB/s
Video Encoder	4Kp30 | (4x) 1080p30 | (2x) 1080p60
Video Decoder	4Kp60 | (2x) 4Kp30 | (8x) 1080p30 | (4x) 1080p60

Fonte:Franklin (2019)

O subsistema de memória unificada da Jetson é compartilhado entre CPU, GPU e mecanismos de multimídia, fornecendo entrada simplificada de sensores, cópia nula entre memórias e *pipelines* de processamento eficientes. Este subsistema será abordado em um capítulo específico pois é essencial para o funcionamento otimizado da aplicação.

Outro ponto importante para a seleção deste módulo é a sua comparação de execução de rede neural com outros computadores populares. Por exemplo, o modelo para detecção de objetos *SSD Mobilenet-V2* executa a 39 quadros por segundo enquanto o *Raspberry Pi 3* executa o mesmo modelo a um quadro por segundo (FRANKLIN, 2019)**.**

Com esta unidade de processamento é possível realizar as inferências na ponta da aplicação, pois testes realizados pelo fabricante informam que a detecção de objetos otimizada pode ser realizada simultaneamente em 8 canais de vídeo independentes, cada um executando a 30 quadros por segundo com resolução de 1080p. Assim, pode ser implantado como uma plataforma de análise de vídeo inteligente de ponta para gravadores de vídeo em rede (NVR), câmeras inteligentes e *gateways* IoT (FRANKLIN, 2019).

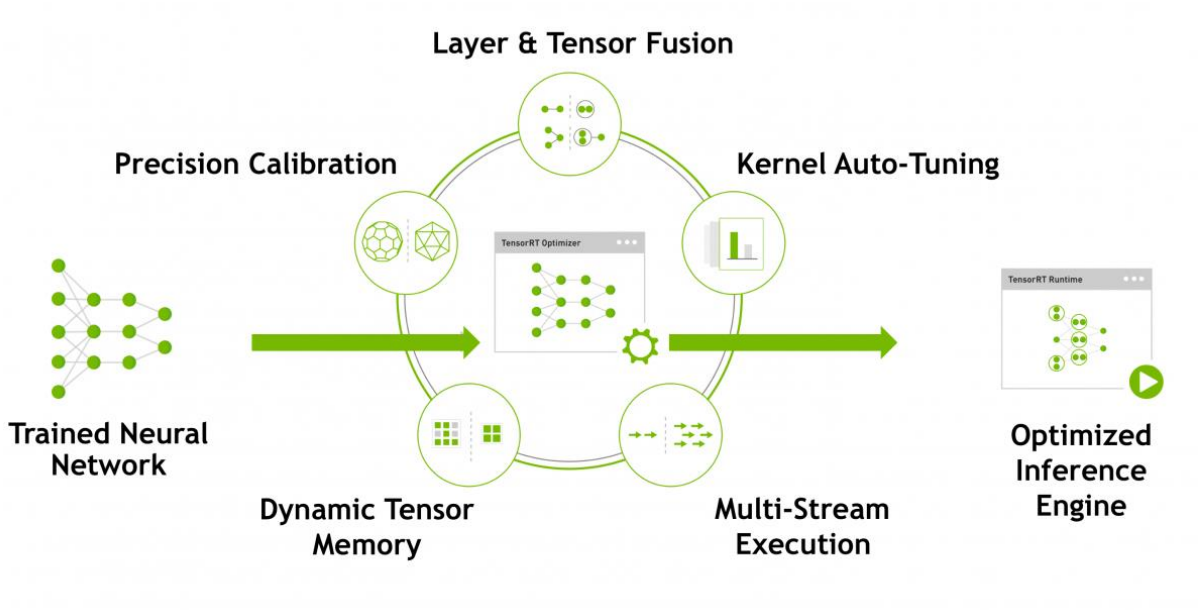
Porém, são necessárias algumas otimizações para atingir a performance citada acima, por exemplo, a rede neural treinada deve passar por um processo de transformação para a execução mais rápida, denominada *NVIDIA TensorRT*™. E a aplicação para a detecção de objetos é desenvolvida através de uma SDK (*software development kit*) específica denominada *NVIDIA Deepstream*. Essas plataformas são brevemente descritas a seguir.

2.4.1 Plataforma NVIDIA Tensor RT

Para realizar a detecção de objetos de maneira otimizada, utilizando a aceleração de computação paralela da GPU, é necessário realizar uma transformação na rede neural já treinada com a plataforma *NVIDIA TensorRT*.

TensorRT foi desenvolvido com base no CUDA, o modelo de programação paralela da NVIDIA, e permite otimizar a inferência para estruturas de aprendizado profundo utilizando bibliotecas e ferramentas de desenvolvimento. A Figura 17 ilustra os processos que podem ser utilizados para transormar uma estrutura treinada, entre eles achatamento de camadas ou execução paralela. O resultado da obtido com a ferramenta é uma estrutura para execução otimizada na plataforma de hardware.

Figura 17 – Processo TensorRT para otimização da rede neural



Fonte: Nvidia Corporation (2019)

A ferramenta também fornece otimizações para implantações de produção de aplicativos de inferência de aprendizado profundo, como fluxos de vídeo, reconhecimento de fala, recomendação e processamento de linguagem natural. A inferência de precisão reduzida reduz significativamente a latência de aplicações, que é um requisito para muitos serviços em tempo real, aplicativos automáticos e incorporados.

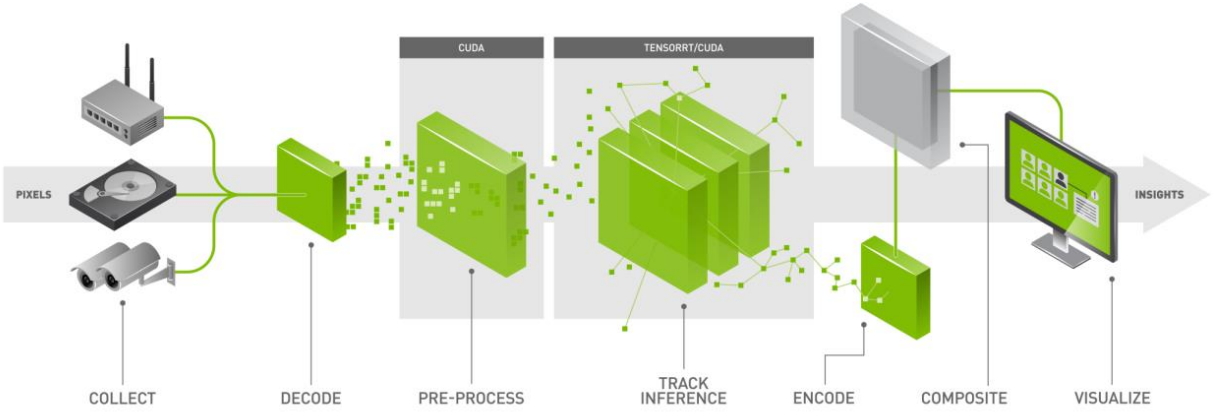
O núcleo do TensorRT é uma biblioteca C que facilita a inferência de alto desempenho nas unidades de processamento gráfico (GPUs) da NVIDIA. Ele foi projetado para funcionar de maneira complementar com estruturas de treinamento

como TensorFlow (HUANG, RATHOD, *et al.*, 2019), Caffe (JIA, SHELHAMER, *et al.*, 2014), PyTorch (PASZKE, GROSS, *et al.*, 2019), MXNet (CHEN, LI, *et al.*, 2015), etc. Ele se concentra especificamente na execução de uma rede já treinada de forma rápida e eficiente em uma GPU (NVIDIA CORPORATION, 2019).

2.4.2 SDK *Deepstream*

Este SDK (*Software Development Kit*) fornece ferramentas para análise de fluxos de vídeo. A estrutura do aplicativo *DeepStream* apresenta blocos de construção acelerados por *hardware*, chamados *plugins*, que são ligados para gerar um fluxo de processamento que utiliza os recursos da plataforma embarcada de maneira otimizada. A Figura 18 ilustra uma possível aplicação utilizando o conjunto de ferramentas, em que dados em forma de imagens são inseridos no fluxo de processamento, que realiza a decodificação do vídeo, pré-processamento, inferência e detecção, e por fim codifica para extração de informações.

Figura 18 – Fluxo Típico com *Deepstream* com Visão Computacional

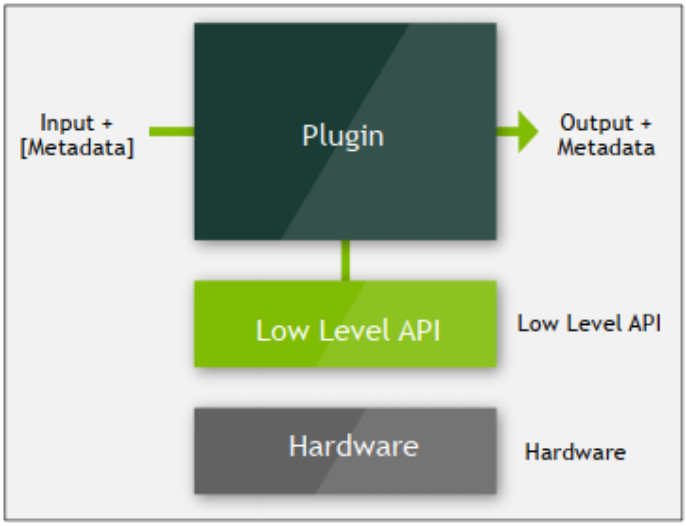


Fonte: Jeun (2018)

Esse *SDK* utiliza plugins baseados no *framework GStreamer*, que é um estrutura de código aberto para desenvolvimento de aplicações que lidam *streams* multimídia, como áudio e vídeo. Cada *plugin* realiza uma tarefa especifica e tem compatbilidade de conexão com outros *plugins*, que são montados em forma de fluxo único, com uma entrada e uma saída, ilustrado na Figura 19. O fluxo de dados da *pipeline* são parametrizações das informações da imagem, também chamados de meta-dados, evitando a cópia múltiplas imagens na memória. Na realidade, esse SDK evita ao máximo realizar cópias na memória para aumentar a performance. Cada

plugin da aplicação adiciona ou altera informações presentes na estrutura dos meta-dados.

Figura 19 – Estrutura *plugin GStreamer*



Fonte: Jeun (2018)

A comunicação do plugin com o *hardware* é realizada por interface de programação (API) de baixo nível, ou seja, facilita o acesso ao hardware para montar aplicações, como por exemplo conectar e utilizar uma câmera. Os principais plugins com aceleração de hardware utilizados são os seguintes:

Tabela 3 - Exemplos de *plugins* Deepstream

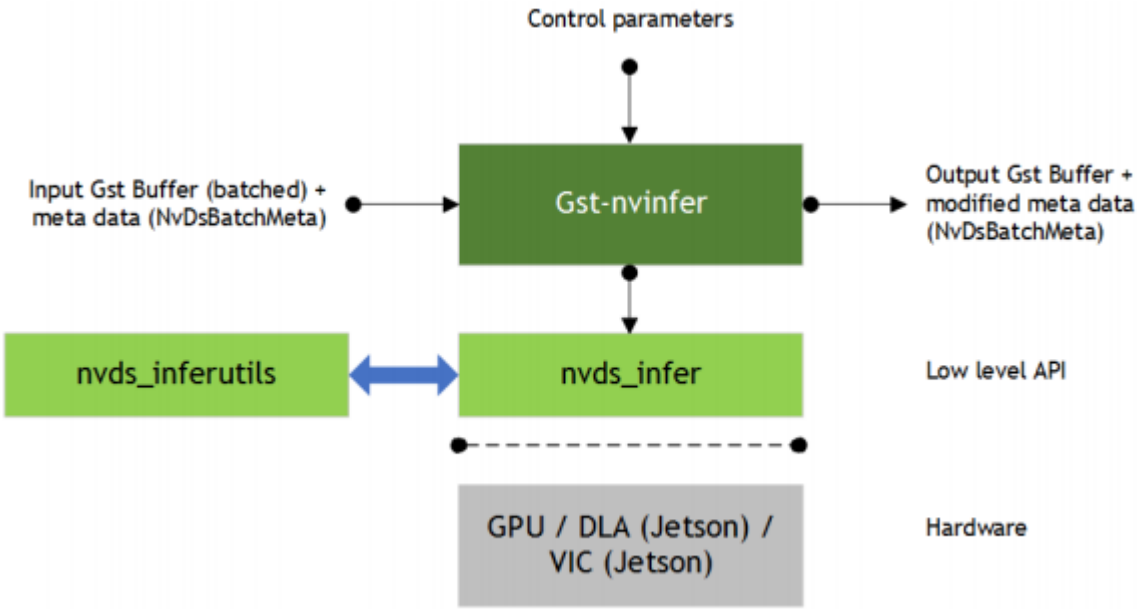
Nome	**Funcionalidade**
gst-nvvideocodecs	Decodifica vídeos H.265 e H.264
gst-nvstreammux	Agrega serializa fluxos de vídeo paralelos
gst-nvinfer	Usa o modelos TensorRT para detecção e classificação
gst-nvosd	Renderiza na tela a imagem com as caixas de identificação

Fonte: Jeun (2018)

O *plugin gst-nvinfer,* pode ser considerado o coração da aplicação que utiliza a computação paralela da GPU para realizar o pré-processamento da imagem, detectar classes e adicionar informações aos meta-dados do fluxo de processamento. A Figura 20 esquematiza a estrutura o plugin de inferência, que em alto nível possui somente entrada e saída de metadados, além de parâmetros de controle. E realiza a

comunicação com com o *hardware* através de API´s (*Application Programming Interface*) de nível baixo.

Figura 20 – *Plugin GST*-NVInfer



Fonte: Jeun (2018)

A sua configuração é realizada através de um arquivo texto, que define suas propriedades como caminho local para a o modelo *TensorRT* a ser utilizado, número de classes, limiares de detecção, rótulos de classes, etc. O *plugin* executa uma normalização do píxeis de acordo com a equação (4) e produz dados planares RGB / BGR / GREY que são transmitidos ao mecanismo TensorRT para inferir. O tipo de saída gerado pela biblioteca de baixo nível depende do tipo de rede.

$$y = fator\_norm*(x-media) \quad (4)$$

Onde:

y = saída do respectivo píxel do tipo *float*

fator_norm = fator de normalização do tipo *float*

x = píxel de entrada do tipo int8 com dimensão 0 a 255

media = valor médio correspondente to tipo *float*

É importante ressaltar que apesar da base *GStreamer* ser de código aberto com base em C, as APIs de baixo nível são proprietárias do fabricante. Existem diversas formas para montar o fluxo de aplicação, é possível usar linguagem C que permite uma maior flexibildade porém exige um entendimento profundo da forma que

os *plugins* são estruturados. Também é possível conectar os *plugins* com um arquivo de configuração, em que basicamente os *plugins* com suas respectivas propriedades são detalhados de forma sequencial e as informações omitidas de cada *plugin* são empregadas com configurações padrão. Ou ainda para testes rápidos, a *pipeline* pode ser montado em linha de comando direto no terminal da plataforma.

2.5 Resumo da solução proposta

A solução proposta para a detecção da presença humana utiliza o sistema *Jetson Nano* para execução do modelo neural *SSD_Inception_V2*. O modelo treinado diponibilizado em formato *TensorFlow* na página do *github* de Huang *et al.* (2019) é convertido utilizando o *TensorRT* para a integração do modelo na aplicação *Deepstream*, que executa uma *pipeline* de processamento de dados em que a fonte é um vídeo em formato MP4 e a saída com os objetos detectados é visualizada no monitor. A Tabela 4 exibe o resumo da solução proposta, demonstrando o propósito de cada item e a motivação respectiva para a sua seleção ou utilização.

Tabela 4 - Resumo da solução proposta

Item	Propósito	Motivação
Jetson Nano	Execução da aplicação para detecção e envio de comandos	Ferramentas de otimização e plataforma com *hardware* dedicado para modelos IA.
TensorFlow	Modelo *SSD_Inception_V2* voltado para a aplicações com recursos limitados	Modelo do estado da arte treinado com as classes de interesse para a aplicação.
TensorRT	Conversão do modelo *TensorFlow* para o modelo executável no *Jetson Nano*.	Ferramenta para otimização da execução da estrutura neural.
Deepstream	Aplicação que executa a o fuxo de processamento do vídeo até a visualização da imagem com objetos detectados	Ferramenta para desenvolvimento de aplicações escalaláveis e execução otimizada da arquitetura *Jetson Nano*.

Fonte: Elaborado pelo autor (2019)

3 METODOLOGIA

Esse capítulo apresenta os principais métodos utilizados no projeto, bem como as soluções utilizadas para os problemas mais importantes. Na Seção 3.1 é demonstrado o funcionamento da rede no *Jetson Nano* e a conversão do modelo *TensorFlow* para o modelo *TensorRT*. A Seção 3.2 aborda o desenvolvimento do protótipo que detecta um carro em miniatura e aciona uma lâmpada respectiva a posição.

A Seção 3.3 desenvolve a aplicação *Deepstream* que descreve o fluxo de processamento desde a fonte de imagens até a visualização do vídeo com os objetos detectados. Na seção 3.4 é abordado a metodologia para verificar os testes de desempenho da aplicação e a seção 3.5 exemplifica como foi realizado o estudo de caso para a verificação da assertividade do modelo neural..

3.1 Execução da rede neural

A rede neural é baseada na implementação de Huang *et al.* (2017), que possui uma variação da rede originalmente proposta por Liu *et al.* (2016), que possui a estrutura *VGG16* (SIMONYAN e ZISSERMAN, 2015) como classificador principal e o modelo aplicado utiliza o classificador *Inception_V2*.

A rede neural SSD_Inception_V2 inicialmente está no modelo para execução em *TensorFlow* e precisa ser convertida para o modelo do TensorRT. O modelo já está treinado no dataset *MSCOCO*, que conta com 91 classes, incluindo carros, pessoas, bicicletas, motos e caminhões, todos são parâmetros para a detecção da presença humana. No total, esse conjunto de dados possui 2,5 milhões de objetos rotulados em 328 mil imagens (LIN, PATTERSON, *et al.*, 2015).

O modelo é disponiblizado pelos pesquisadores do TensorFlow na página do *github* de Huang *et al.* (2019). Trata-se de um modelo no estado da arte e embora tenha muito mais classes de detecção do que o necessário para a aplicação, elimina a necessidade do treinamento da rede neural, que pode demorar centenas de horas em computadores de ponta Zhu *et al.* (2018), além da necessidade de preparar e catalogar um conjunto de dados extenso. O modelo selecionado de *SSD_Inception_V2* possui um mAP de 24 pontos no conjunto de dados *MSCOCO*.

Após carregar os arquivos, é necessário utilizar a API do *TensorRT* para converter o arquivo do grafo da rede *TensorFlow* para um modelo executável na arquitetura Jetson. Porém, o grafo SSD do TensorFlow possui algumas operações que atualmente não são suportadas no TensorRT. Por isso é necessário usar um pré-processador no modelo que combina várias operações do grafo em uma única operação personalizada, implementada como uma camada de plug-in no TensorRT.

Os passos para converter o modelo podem ser verificados na página do Github de Nvidia TensorRT (2019) e as principais etapas da execução da rede neural são:

- **Pré-processamento da imagem**: A imagem é redimensionada de acordo com a entrada da primeira camada da rede neural que possui um tamanho de 300x300x3. Também realiza a normalização dos valores dos píxeis para que seus valores estejam entre [-1, 1].
- **Extrator de características**: Esta etapa executa a rede *InceptionV2* na imagem pré-processada. Os mapas de recursos gerados são usados pela etapa de geração da âncora para gerar caixas delimitadoras padrão. Nesta rede, o tamanho dos mapas de características usados para a geração de âncoras são de [(19x19), (10x10), (5x5), (3x3), (2x2), (1x1)].
- **Predição das caixas delimitadoras**: Esta etapa consiste em utilizar os mapa de recursos em alto nível como entrada para produzir uma lista de coordenadas (x,y) atrelada a uma lista de pontuação de cada classe, essa lista é enviada para o pós processamento.
- **Gerador de grade e âncoras**: Neste passo é gerado um conjunto de caixas delimitadoras padrão para cada célula do mapa de características através do *plugin* gridAnchorGenerator do TensorRT.
- **Pós processamento**: O resultado da predição das caixas delimitadoras com seus respectivas coordenadas e pontuação de confiança de cada mapa de características são processadas junto das caixas delimitadoras padrão produzidas pelo gerador de âncoras através de um algorítmo *non-maximum suppression* (NMS) que remove a maioria das caixas delimitadoras baseado na pontuação de confidência e IoU (Interseção sobre união).

A conversão do modelo Tensorflow para o modelo TensorRT é feito pelo *script* em linguagem *python* convert_to_uff.py que utiliza a basicamente uma API do TensorRT uff.from_tensorflow_frozen_model() em combinação com um arquivo de

configuração config.py devido a adaptação das camadas não suportadas. Essa função tem os seguintes principais argumentos de entrada:

- Modelo TensorFlow a ser convertido.
- Lista de nós de saída, se este argumento não for fornecido a saída é estimada automáticamente.
- Nome do modelo UFF de saída.
- Pré-processador: Um caminho para um script de pré-processamento para execução antes da conversão.

Neste caso, o arquivo config.py tem a função do pré-processador definida e faz a adição de plugins customizados do TensorRT ao modelo UFF para funcionamento deste modelo na plataforma. O *script* adiciona um *plugin* para entrada da imagem de tamanho 300x300x1, também insere o gerador de âncoras e caixas delimitadoras para com 6 camadas, com escalas variáveis de 3 a 0,33 e camadas de ativação (*feature maps*) de tamanhos 19x19, 10x10, 5x5, 3x3, 2x2, e 1x1. Outro plugin inserido é o NMS (*non-maximum suppression*) que realiza o algoritmo de filtro das caixas com sobrepostas.

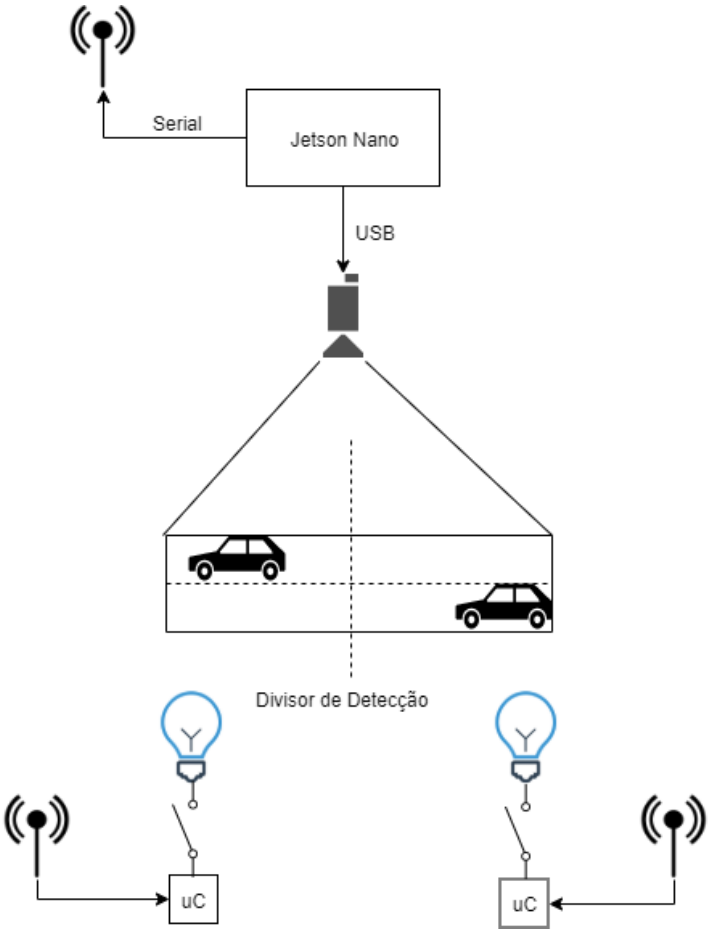
Outros plugins auxiliares são inseridos para concatenar os plugins customizados ao modelo e achatar camadas para acelerar o tempo de execução. Os detalhes da aplicação desta conversão podem ser encontrados na documentação do TensorRT.

3.2 Prova de conceito

A prova de conceito tem como objetivo realizar testes preliminares da ideia proposta para o controle da iluminação. Nesta maquete, o objetivo é realizar a detecção de carros e acionar uma lâmpada convencional baseada na posição do carro, se o carro estiver no lado direito, a lâmpada da direita deve estar acesa e caso contrário, a lâmpada da esquerda deverá estar acesa. Em caso de nenhuma detecção, nenhuma das lâmpadas deve estar acionada.

A Figura 21 esquematiza a ligação dos componentes básicos da prova de conceito. A comunicação do *Jetson Nano* com o controle das lâmpadas é sem fio utilizando o módulo de rádio HC-12.

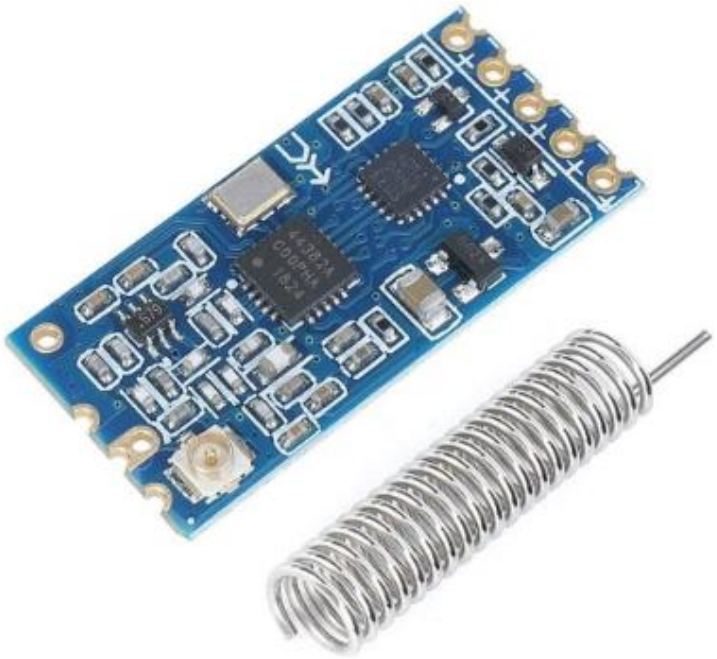
Figura 21 - Esquema da Prova de Conceito



Fonte: Elaborado pelo autor (2019)

A comunicação realizada pelo módulo de rádio HC-12 possui interface serial para o envio de dados e sua configuração é realizada com comandos AT. A Figura 22 ilustra o módulo utilizado que opera em frêquencia de 433 MHz e tem alcance especificado em até 1000 metros.

Figura 22 – Módulo RF HC-12



Fonte: Filipeflop (2019)

As lâmpadas convencionais são acionadas através de um relé de estado sólido para tensão alternada. Este relé é comandado pelo microcontrolador STM32F1 (2020), também conhecido como *BluePill*, que recebe uma mensagem via serial com codificação simplificada do módulo de rádio.

A detecção de objetos pela rede neural é realizada no *Jetson Nano*, com o programa *DetectNet*, que realiza a toda a execução da rede neural em alto nível em linguagem Python com *bindings* para a linguagem C e a *pipeline* dos plugins *GStreamer*. As instruções para instalar este programa podem ser encontrados na página do Github de Franklin (2019) e pode ser executado com diversos modelos treinados no *dataset* MSCOCO.

O programa retorna uma lista dos objetos detectados junto de suas respectivas pontuações de confiança e suas coordenadas. O número da classe do objeto detectado depende do conjunto de dados em que a rede foi treinada. Neste caso, o carro é o número três do *dataset* MSCOCO e é o parâmetro para acionamento das lâmpadas. O divisor do lado da detecção é baseado nas coordenadas da imagem, que neste caso é uma câmera do tipo *WebCam* com tamanho 640x480 pixeis. Logo, se o objeto três (carro) for detectado na coordenada **x** entre [0,240) ele está no lado direito, devido ao espelhamento da estrutura montada, e uma mensagem codificada deve ser

enviada a porta serial. Caso contrário, (**x** entre [240, 480]) o objeto está do lado esquerdo.

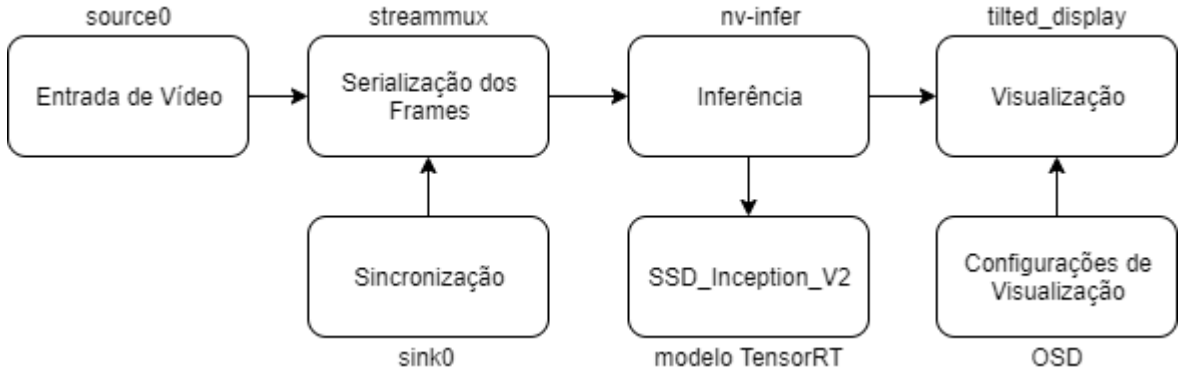
Além disso, para não sobrecarregar o módulo de rádio ou permitir que a lâmpada ligue e desligue muito rapidamente, foi montado um filtro baseado na média das detecções. A cada dez detecções é enviado o comando para a luminária, que depende no número de contagem de detecções do carro do lado esquero e direito. No filtro implementado, é necessário que ocorram ao menos duas detecções em um lado a cada dez tentativas para enviar um comando de acionamento do respectivo lado, caso não sejam realizadas duas detecções, o comando enviado é para desligamento do respectivo lado. Ainda, optou-se por usar um limiar de confiança de detecção de 0,4 para contar como objeto detectado.

A implementação em Python que realiza a detecção em alto nível está disponível no **Apêndice** A. O código em linguagem C implementado no microcontrolador que faz o recebimento da mensagem via serial, decodificação e acionamento do relé de estado sólido está disponível em *https://github.com/Ailumi/STMLightControl*.

3.3 Aplicação Deepstream

A utilização desta SDK requer o sistema operacional *JetPack* e as bibliotecas *GStreamer*. Para montar a *pipeline*, foi utilizado o método com arquivo de configuração que descreve a aplicação. A fonte da pipeline é o arquivo de vídeo em formato tipo mp4 que é processado através da rede neural. Por fim a imagem com a tentativa de detecção de objeto é renderizada na tela. A Figura 23 demonstra a visão geral desta *pipeline* de processamento com seus respectivos *plugins*.

Figura 23 - Pipeline Deepstream



Fonte: Elaborado pelo autor (2019)

O arquivo de configuração é descrito no bloco de *source0* o tipo de arquivo e uma *URI* do caminho para pasta em que está armazenado o vídeo. O bloco *tilted-display* descreve o formato da saída para visualização no monitor de tamanho 640x480. Com uma coluna e uma linha, pois trata-se de apenas um fluxo de vídeo por vez. Em caso de multiplas fontes de vídeo, o número de linhas e colunas poderia ser alterado.

No bloco *sink0* é descrito como a aplicação irá sincronizar o fluxo, pois o tempo de execução da inferência da rede neural e a taxa de atualização de *frames* do vídeo são divergentes. A sincronização foi definida de acordo com a capacidade de execução da rede neural em quadros por segundo. O bloco *osd* transforma os meta-dados do fluxo da pipeline em imagem, em que as classes detectadas e suas respectivas coordenadas são inseridos. Ainda é possível realizar alterações na fonte de escrita, largura da caixa delimitadora, etc, para facilitar a visualização.

O bloco de *streammux* realiza a serialização das entradas, em que diversas entradas são afuniladas em uma, e realiza um algoritmo de escalonamento *round-robin* para selecionar o *batch* ou lote de frames a serem processados. Mesmo que fluxo da aplicação contenha apenas uma fonte de dados, este bloco é sempre obrigatório pois ele inicia o fluxo de meta-dados. Por fim, é defindo o mecanismo de inferência em *primary-gie*, que descreve o *plugin nv-infer* e aponta para o arquivo de rótulos das classes e o arquivo de configuração da rede neural. As propriedades deste bloco também definem que cada *frame* deve ser processado pela rede neural, sem intercalação ou saltos. O **Apêndice** B contém a descrição do arquivo de configuração para utilização com o aplicativo *Deepstream*.

O arquivo de descrição do inferidor indica o caminho de pastas para o local do arquivo do modelo neural convertido do tipo UFF. Além disso define as dimensões da entrada em 300x300x1x0 para diálogo entra o programa *Deepstream* e o modelo UFF. O fator de normalização foi inserido de acordo com a sugestão do fabricante em 0.0078431372 e o limiar de pontuação para detecção é de 0.5 para todas as classes do modelo, conforme pode ser averiguado no **Apêndice** C, que contém o arquivo de configuração completo.

3.4 *Benchmarks* de execução

Para testar o desempenho da execução da *pipeline*, foi selecionado um vídeo de amostra em formato MP4 com compressão no padrão H.264 e tamanho de 1920x1080. A duração da gravação tem 48 segundos de duração e possui um *framerate* de 24 quadros por segundo.

Para medir o desempenho da aplicação na arquitetura *Jetson Nano* foram utilizados duas ferramentas. A própria aplicação pode realizar a medição dos quadros por segundo processados habilitando um *flag* no arquivo de configuração. O cálculo do QPS é flexível e nessa aplicação optou-se por usar a média de quadros a cada cinco segundos. Assim, no terminal da plataforma, são exibidos as informações do tempo de execução a cada cinco segundos.

Além disso, foi utilizado um monitor de recursos *Tegrasts* (NVIDIA CORPORATION, 2019), que mede o uso dos componentes do sistema embarcado, informando:

- Memória RAM
- Memória SWAP
- IRAM
- Uso percentual e frequência de cada um dos 4 núcleos do CPU
- EMC (External Memory Controller)
- Uso percentual e frequência da GPU
- Temperaturas e níveis de tensão

Este programa salva as informações em um arquivo de texto de registro a cada intervalo de tempo, quando especificado. Para permitir a visualização destas informações em forma de gráfico, foi criado um *script* em Python (descrito no **Apêndice** D) que converte o arquivo de texto puro em arquivo de planilha. O tempo de intervalo de medição foi selecionado em 100 milisegundos e as informações mais relevantes para o estudo são o uso da memória ram, cpu e gpu. Os outros parâmetros como frequência e temperatura não variaram durante os testes.

Para a execução da pipeline, foi selecionado o modo de potência e frequência máxima. A potência máxima do sistema Jetson Nano é de 10 W e pode ser ativado com o comando *$ sudo nvpmodel -m 0*. A frêquencia de trabalho do hardware foi fixado na capacidade máxima executando o *script jetson_clock.sh*. A frequencia de trabalho durante os teste foi de 1428 Mhz para a CPU e 921 Mhz para a GPU.

3.5 Assertividade no estudo de caso

Para o estudo de caso, foi utilizado um período de duas horas de gravação em uma via local, no período da noite para testar a rede neural em uma situação próxima da realidade. Para capturar as imagens, foi utilizado uma câmera IP e armazendo em um computador através do software VLC, usando o protocolo RTSP. O tamanho da imagem gravada é de 1920x1080, a uma frequência de 24 quadros por segundo. Para facilitar os testes, o vídeo foi recortado nos eventos e enviado via rede/ssh para o *Jetson Nano*, que por fim executa a pipeline de processamento para identificar objetos. A Figura 24 mostra uma captura de imagem da visão da câmera utilizada nos testes.

Figura 24 – Visão da camera do estudo de caso

Fonte: Acervo do autor (2019)

A via avaliada tem baixo fluxo, e a maioria dos eventos consistem em passagens com um único objeto por vez, seja ele um carro, moto ou pessoa. Além disso, a baixa velocidade do tráfego aumenta a duração do evento para a detecção da rede neural. Os carros estacionados não são considerados como evento e não foram testadas suas detecções.

O teste realizado utiliza imagens de um caso em que a solução poderia ser aplicada, porém não estão rotuladas para utilizar uma métrica de avaliação mais completa como o mAP. Neste caso, o teste considera qualquer detecção correta em

qualquer instante como um evento assertivo pois apenas uma detecção seria suficiente para acionar um conjunto de luminárias. Assim é possível determinar o *recall* do modelo utilizando a equação descrita em (2), em que os positivos verdadeiros (TP) são os eventos em que predição é correta e o falsos negativos (FN) são eventos em que não houve detecção ou houve uma detecção que não correspondia ao objeto na realidade.

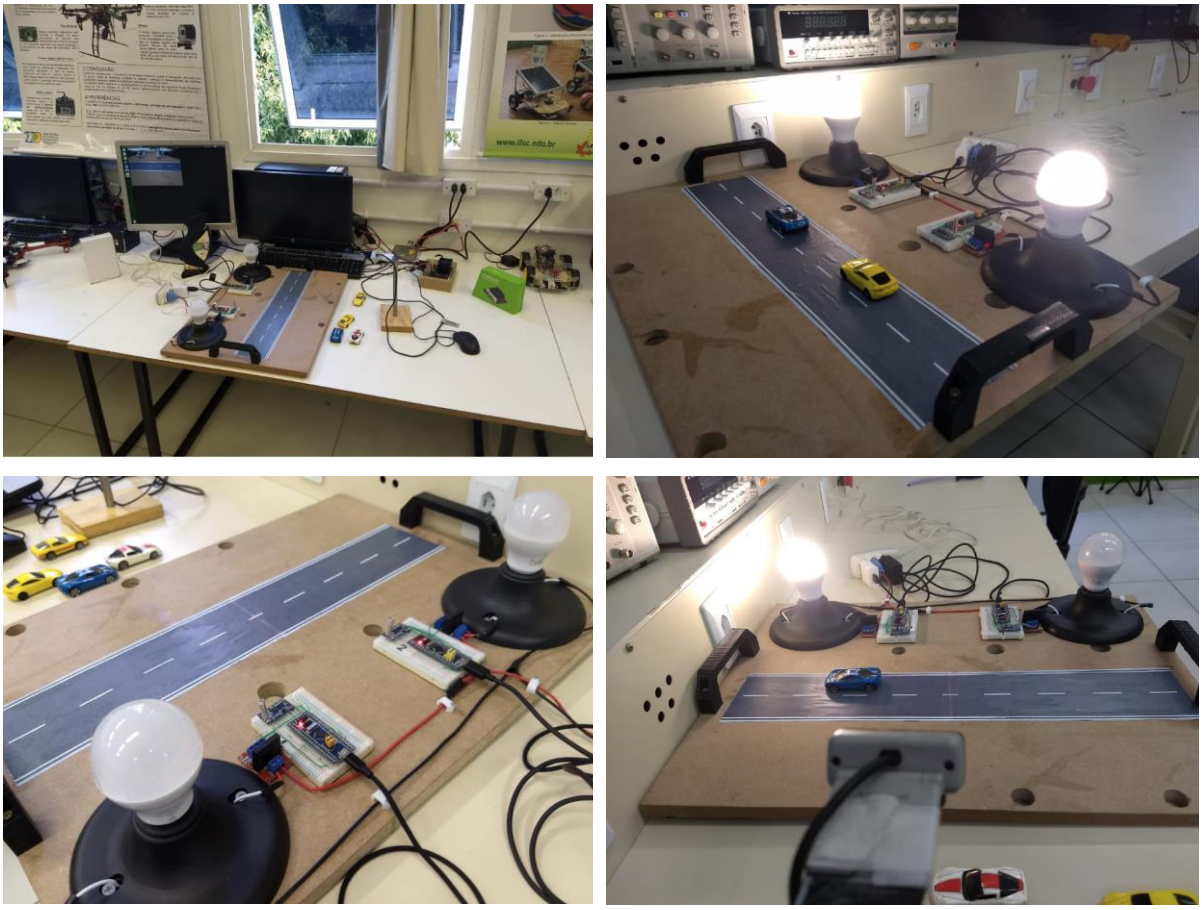
4 APRESENTAÇÃO DOS RESULTADOS

Neste capítulo são apresentados os resultados dos testes realizados. Na Seção 4.1 são demonstrados os principais resultados atingidos com o protótipo desenvolvido. A Seção 4.2 aborda o desempenho atingido com a execução da aplicação *Deepstream* e a Seção 4.3 apresenta o resultado das detecções no estudo de caso.

4.1 Testes do protótipo

A protótipo em maquete foi exposto na 16ª Semana Nacional de Ciência e Tecnologia do IFSC para exemplificar o funcionamento da solução. A Figura 25 mostra o resultado da montagem, em que a rede neural detecta com sucesso a maioria dos objetos de interesse, e envia comando para acionamento ou desligamento para as respectivas lâmpadas.

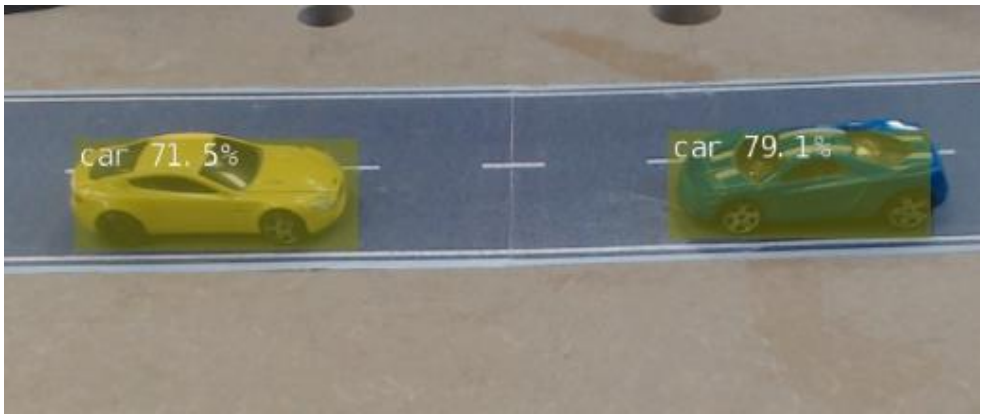
Figura 25 - Exposição da prova de conceito



Fonte: Acervo do autor (2019)

De maneira geral, os carros coloridos são detectados com facilidade. E observou-se que a pontuação da detecção aumenta quando estes carros estão sendo vistos lateralmente pela câmera. Na Figura 26 pode se observar a captura de tela da detecção durante os testes.

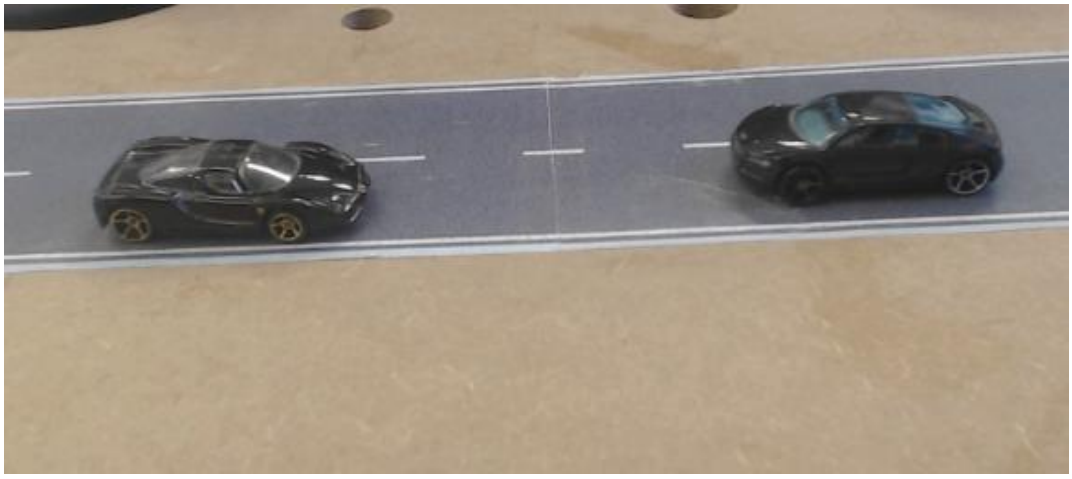
Figura 26 - Objetos detectados



Fonte: Acervo do Autor (2019)

Entranto, os carros totalmente pretos da Figura 27 raramente são detectados pela rede neural, possivelmento pelo baixo contraste em relação ao fundo da imagem. Variações de posição, na orientação ou na distância também não facilitaram a detecção. E este caso resultou em uma falha no comando automático da iluminação.

Figura 27 – Objetos não detectados



Fonte: Acervo do autor (2019)

Neste cenário de aplicação, a distância entre o comunicador e os receptores não ultrapassa dois metros e não houveram problemas em relação a comunicação sem fio. Os comandos para a chave de acionamento da luminária também foram

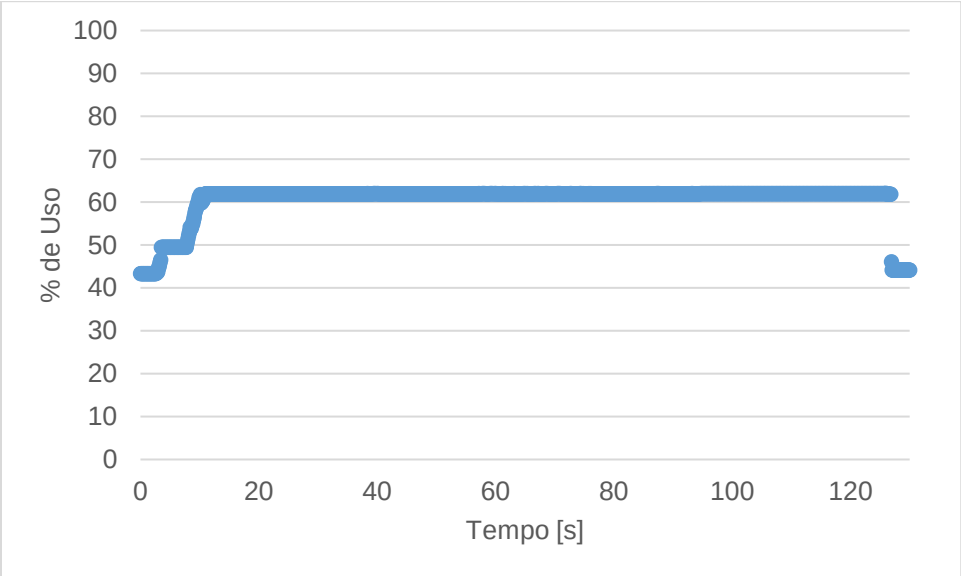
realizados com sucesso, comprovando o funcionamento da solução com a maioria dos objetos de interesse deste caso.

4.2 Desempenho da aplicação

O desempenho de processamento da aplicação *Deepstream* descrevida pelo arquivo de configuração e modelo *TensorRT* resultou em uma execução média de 11,73 quadros por segundo. A entrada de imagens é atualizada a uma taxa de 24 quadros por segundo, fazendo com a duração total da visualização do vídeo estivesse próxima de 120 segundos devido ao sincronismo com a capacidade de tempo de execução da rede neural, que neste caso é de aproximadamente 85,25 mili segundos para cada imagem.

A taxa de atualização dos quadros permaneceu constante durantes os testes e mudança de parâmetros secundários da *pipeline* não alteraram o tempo de execução total. A quantidade de memória utilizada durante a aplicação também não alterou significativamente, sempre próxima de 60% de uso de um total de 4GB, conforme pode ser observado na Figura 28.

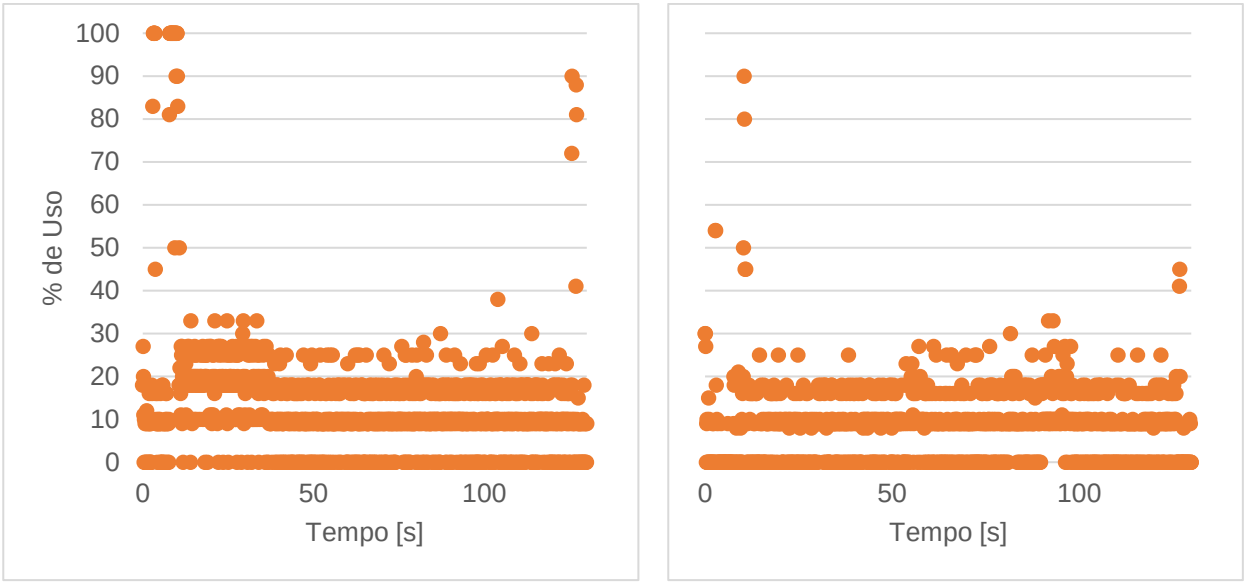
Figura 28 – Uso percentual da Memória RAM



Fonte: Elaborado pelo autor (2019)

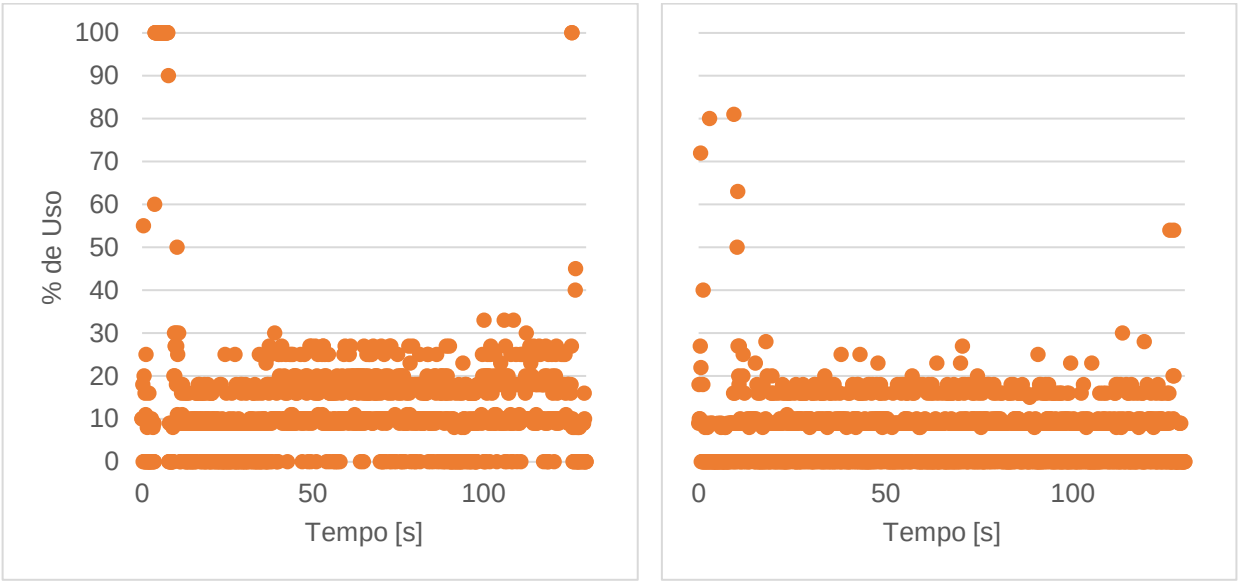
Além disso, durante a execução do teste nenhum núcleo da CPU estava sobrecarregado, conforme pode ser observar na Figura 29 e Figura 30, que mostram o percentual de uso individual dos quatro núcleos obtidos do registro *Tegrastats*.

Figura 29 -Percentual de uso da CPU núcleos 1 (esquerda) e 2



Fonte: Elaborado pelo autor (2019)

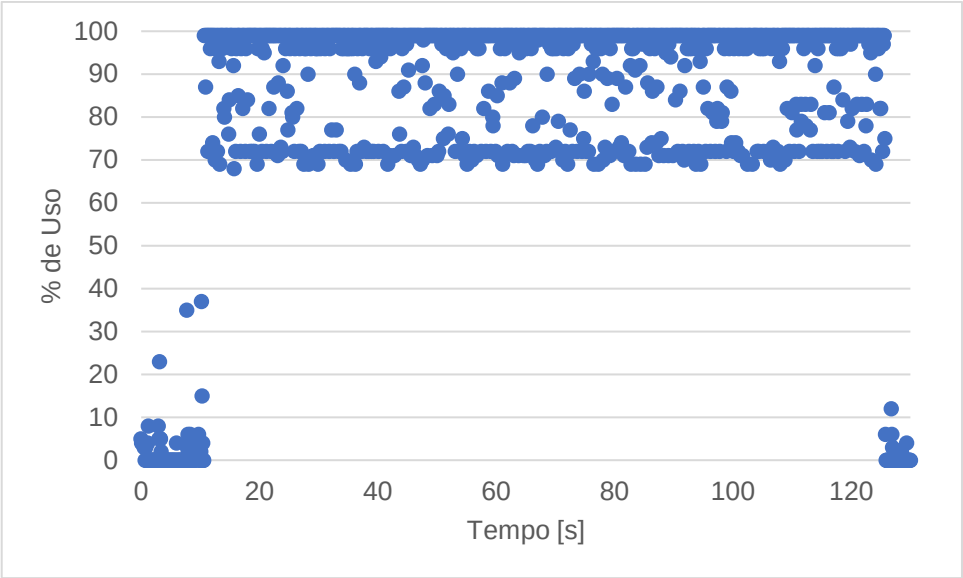
Figura 30 - Percentual de uso da CPU núcleos 3 (esquerda) e 4



Fonte: Elaborado pelo autor (2019)

Ao contrário da CPU e RAM, que estão distantes do 100% de utilização na maioria do tempo, o uso da GPU está sempre próximo da sua capacidade máxima. A Figura 31 mostra o momento em que a aplicação é iniciada e parada. E o processador de computação paralela tem alta utilização para execução da rede neural, indicando que o componente é empregado para essa tarefa e que está muito próximo da sua limitação física.

Figura 31 – Uso percentual da GPU



Fonte: Elaborado pelo autor (2019)

4.3 Resultados do estudo de caso

Durante a gravação para testes do estudo de caso ocorreram 37 eventos. Entre eles há passagem de carros, motos e pedestres.

O teste resultou em 31 positivos verdadeiros, em que a rede detectou corretamente a presença de um objeto na imagem. A Figura 32 mostra um exemplo de positivo verdadeiro, em que um carro é detectado corretamente pelo modelo utilizado. Outros exemplos de TP capturados podem ser avaliados no **Apêndice** E.

Nos 6 eventos restantes, os objetos que seriam utilizados para acionamento do sistema de iluminação não foram detectados resultando em 6 falsos negativos. Desta forma, o *recall* obtido neste estudo de caso foi de 83,8%.

Figura 32 – Evento com detecção

Fonte: Acervo do autor (2019)

A precisão do modelo não foi avaliado neste estudo pois foram testados apenas recortes da gravação com eventos devido ao considerável tempo que seria necessário para processar toda a filmagem. Porém, em um dos 37 eventos um falso positivo foi detectado conforme mostrado na Figura 33.

Figura 33 –Detecção com falso positivo e falso negativo

Fonte: Acervo do autor (2019)

5 CONCLUSÃO

Neste trabalho utilizou-se um modelo do estado da arte das redes neurais para detecção de objeto treinada em um conjunto de dados com 91 classes, incluindo os objetos que são parâmetros para controle da iluminação. O modelo utilizado suprimiu a necessidade do treinamento, que é uma tarefa de longa duração. Porém, ele é genérico e não é ideal para aplicação devido ao fato de ter mais classes que o necessário.

O modelo utilizado é a estrutura *SSD_Inception_V2* do *Tensorflow*, que foi convertida para o formato *TensorRT*. Dessa forma, a execução é otimizada na plataforma embarcada *Jetson Nano*. A integração do modelo neural no sistema embarcado exige um entendimento profundo da arquitetura do *Jetson Nano* e da estrutura do detector de objetos. A utilização das ferramentas da maneira correta é essencial devido a limitação dos recursos de processamento disponíveis, e por isso uma série de otimizações devem ser feitas para viabilizar aplicações que requerem tempo real.

Na prova de conceito elaborada, o controle remoto do micro-sistema de iluminação foi controlado com base da detecção da posição relativa de miniaturas de carros e mostrou que há espaço para melhoria no modelo de detecções pois alguns objetos de interesse raramente foram identificados.

O fluxo de processamento do vídeo descrito com o método *Deepstream* realizou a detecção de objetos com um tempo de execução de aproximadamente 85 ms na plataforma embarcada, utilizando a maioria da capacidade da unidade de processamento dedicada para inferência com visão computacional de rede neurais. Este tempo de resposta para uma detecção poderia ser suficiente para controlar um sistema de iluminação, considerando que uma detecção acionaria um conjunto de luminária setorizadamente.

Na simulação da aplicação em um cenário prático, o estudo de caso obteve uma alta de taxa de detecção de verdadeiros positivos em que apenas 16,2% dos eventos testados não corresponderam a uma detecção correta.

De forma geral, a solução realiza a detecção com uma alta taxa de assertividade com um curto período para processamento, mostrando que há um grande potencial para ser aplicada em conjunto com luminárias aptas a receberem comandos de acionamento ou controle.

REFERÊNCIAS

AILUMI. Github. **STMLightControl**, 2019. Disponivel em: https://github.com/Ailumi/STMLightControl. Acesso em: 30 out. 2019.

AZIERA ABDULLAH, S. H. Y.; SYASYA AZRA ZAINI, N. S. M.; MOHAMAD, S. Y. Smart Street Light Using Intensity Controller. **IEEE**, 2018.

BRUNO, A. *et al.* **Stmicroelectronics Smart Street Lighting Solutions: Remote Control Protocol Over Power Line Communication**. [*S.l.*]. 2012.

CHEN, T. *et al.* MXNet: A Flexible and Efficient Machine Learning. **In Neural Information Processing Systems, Workshop on Machine Learning Systems**, 2015.

CHRISTIANSEN, A. Anchor Boxes — The key to quality object detection. **Medium**, 2018. Disponivel em: https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9. Acesso em: 27 nov. 2019.

DIGITAL ILLUMINATION INTERFACE ALLIANCE. Introducing DALI. **DALI**, 2020. Disponivel em: https://www.digitalilluminationinterface.org/dali/. Acesso em: 5 Fevereiro 2020.

EMPRESA DE PESQUISA ENERGÉTICA. **Anuário Estatístico de Energia Elétrica**. [*S.l.*]. 2018.

FILIPEFLOP. Módulo RF Wireless HC-12 com Antena. **FilipeFlop**, 2019. Disponivel em: https://www.filipeflop.com/produto/modulo-rf-wireless-hc-12-com-antena/. Acesso em: 15 set. 2019.

FRANKLIN, D. Github. **Deploying Deep Learning**, 2019. Disponivel em: https://github.com/dusty-nv/jetson-inference. Acesso em: 26 nov. 2019.

FRANKLIN, D. NVIDIA Developer. **NVIDIA Developer Blog**, 18 mar. 2019. Disponivel em: https://devblogs.nvidia.com/jetson-nano-ai-computing/. Acesso em: 07 nov. 2019.

FUJII, Y. *et al.* Smart street light system with energy saving function. **ResearchGate**, jan. 2013.

HUANG, J. *et al.* **Speed/accuracy trade-offs for modern convolutional object detectors**. Google Research. [*S.l.*], p. 21. 2017.

HUANG, J. *et al.* Tensorflow Object Detection API. **Github**, 2019. Disponivel em: https://github.com/tensorflow/models/tree/master/research/object_detection. Acesso em: 12 out. 2019.

HUI, J. mAP (mean Average Precision) for Object Detection. **medium**, 2018. Disponivel em: https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173. Acesso em: 28 out. 2019.

JEUN, J. **DEEPSTREAM SDK 2.0 WEBINAR**. NVIDIA. [*S.l.*]. 2018.

JIA, Y. *et al.* Caffe: Convolutional Architecture for Fast Feature Embedding. **arXiv**, jun. 2014. ISSN 1408.5093.

KONINKLIJKE PHILIPS ELECTRONICS N.V. **Xitanium Dim 150W 0.7A 1-10V 230V I175**. [*S.l.*]. 2018.

LANGNER, A. L. *et al.* Pilot Project of Street Lighting Telemanagement in a Smart Grid Environment. **IEEE**, 2015.

LIN, T.-Y. *et al.* Microsoft COCO: Common Objects in Context. **arXiv**, 2015. Disponivel em: http://cocodataset.org/#home. Acesso em: 05 nov. 2019.

LIU, W. *et al.* SSD: Single Shot MultiBox Detector. **European conference on computer vision**, Cham, 2016.

MARTINS, R. A. P. **Aplicação de Redes Convolucionais Profundas para Detecção de Massas em Mamografias**. UFSC. Florianópolis, p. 100. 2019.

NIELSEN, M. **Neural Networks and Deep Learning**. San Francisco: Determination press, 2015.

NVIDIA CORPORATION. **L4T Documentation**. [*S.l.*]. 2019.

NVIDIA CORPORATION. TensorRT SDK Documentation. **Nvidia Docs**, 2019. Disponivel em: https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/. Acesso em: 16 nov. 2019.

NVIDIA DEEPSTREAM. Quick Start Guide. **NVIDIA DeepStream SDK**, 2019. Disponivel em: https://docs.nvidia.com/metropolis/deepstream/dev-guide/index.html. Acesso em: 25 out. 2019.

NVIDIA-TENSORRT. Object Detection With A TensorFlow SSD Network. **Github**, 2019. Disponivel em: https://github.com/NVIDIA/TensorRT/tree/release/6.0/samples/opensource/sampleUffSSD. Acesso em: 2019 nov. 10.

OLIVEIRA, R. M. S. D. *et al.* A System Based on Artificial Neural Networks for Automatic Classification of Hydro-generator Stator Windings Partial Discharges. **Journal of Microwaves, Optoelectronics and Electromagnetic Applications**, v. 16, Setembro 2017. ISSN 629.

OUERHANI, N. *et al.* IoT-Based Dynamic Street Light Control for Smart Cities Use Cases. **IEEE**, 2016.

PASZKE, A. *et al.* PyTorch. **Github**, 2019. Disponivel em: https://github.com/pytorch/pytorch. Acesso em: 01 dez. 2019.

REN, S. *et al.* Faster r-cnn: Towards real-time object detection with region proposal networks. **Advances in neural information processing systems**, 2015.

SAHA, S. Towards Data Science. **A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way**, 2018. Disponivel em: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Acesso em: 20 nov. 2019.

SHI, W. *et al.* Edge Computing: Vision and Challenges. **IEEE INTERNET OF THINGS JOURNAL**, 3, 2016.

SIGNIFY HOLDING. Iluminação exterior ligada - City Touch. **Lighting Phillips**, 2020. Disponivel em: https://www.lighting.philips.com.br/sistemas/iluminacao-ligada/citytouch. Acesso em: 04 Fevereiro 2020.

SIMONYAN, K.; ZISSERMAN, A. VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. **arXiv**, Oxford, 2015.

SORRELL, S. **THE FUTURE OF LIGHTING & URBAN MOBILITY IN SMART CITIES**. Hampshire. 2019.

STMICROELECTRONICS. STM32F103C8 - Overview. **ST**, 2020. Disponivel em: https://www.st.com/en/microcontrollers-microprocessors/stm32f103c8.html. Acesso em: 10 Fevereiro 2020.

TAN, R. J. Towards Data Science. **Breaking Down Mean Average Precision (mAP)**, 2019. Disponivel em: https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52. Acesso em: 03 nov. 2019.

TENSFORFLOW. Tensorflow Object Detection API. **GitHub**. Disponivel em: https://github.com/tensorflow/models/tree/master/research/object_detection. Acesso em: 15 out. 2019.

VELAGA, N. R.; KUMAR, A. Techno-economic Evaluation of the Feasibility of a Smart Street Light System: A case study of Rural India. **Procedia - Social and Behavioral Sciences**, 62, 24 out. 2012.

ZHU, H. *et al.* Benchmarking and Analyzing Deep Neural Network. **2018 IEEE International Symposium on Workload Characterization (IISWC)**, p. 88-100, 2018.

ZIGBEE ALLIANCE. What is Zigbee? **Zigbee Alliance**, 2020. Disponivel em: https://zigbeealliance.org/solution/zigbee/. Acesso em: 05 Fevereiro 2020.

ZOU, Z. *et al.* Object Detection in 20 Years: A Survey. **arXiv**, 2019. ISSN 1905.05055.

APÊNDICES

Apêndice A – Detecção *python* do protótipo

```
import jetson.inference
import jetson.utils
import serial

CAR = 3
MIDDLE = 240

car_left = 0
minimun_left = 2

car_right = 0
minimun_right = 2

trys = 0
max_trys=10

port = serial.Serial('/dev/ttyTHS1', baudrate=9600)

net = jetson.inference.detectNet("ssd-inception-v2", threshold=0.4)
camera = jetson.utils.gstCamera(640, 480, "/dev/video0")
display = jetson.utils.glDisplay()

while display.IsOpen():
	trys+=1
	img, width, height = camera.CaptureRGBA()
	detections = net.Detect(img, width, height)

	for object in detections:
		if object.ClassID==CAR and object.Confidence>0.4:

			if object.Left>MIDDLE:    # carro na esquerda
				car_left+=1
			else:                           # carro na direita
				car_right+=1

	display.RenderOnce(img, width, height)
```

```
	display.SetTitle("Object Detection | Network {:.0f}
FPS".format(net.GetNetworkFPS()))

	# mandar via serial a cada X tentativas
	if trys>=max_trys:

		if car_left>=minimun_left: # liga esquerda
			left='0010'
			print('L detected')
		else: # desliga esquerda
			left='0011'
		port.write(str.encode(left))

		if car_right>=minimun_right: # liga direita
			right='0020'
			print('R detected')
		else:
			right='0021'
		port.write(str.encode(right))

		trys=0
		car_left=0
		car_right=0
```

Apêndice B – Configuração *pipeline Deepstream*

```
[application]
enable-perf-measurement=1
perf-measurement-interval-sec=5
gie-kitti-output-dir=streamscl

[tiled-display]
enable=1
rows=1
columns=1
width=640
height=480
gpu-id=0
nvbuf-memory-type=0
```

```
[source0]
enable=1
#Type - 1=CameraV4L2 2=URI 3=MultiURI
type=2
num-sources=1
uri=file:/home/ian/Downloads/estudo_de_caso/001_2.mp4
gpu-id=0
cudadec-memtype=0

[sink0]
enable=1
#Type - 1=FakeSink 2=EglSink 3=File 4=RTSPStreaming 5=Overlay
type=2
sync=0
display-id=0
offset-x=0
offset-y=0
width=0
height=0
overlay-id=1
source-id=0

[osd]
enable=1
gpu-id=0
border-width=3
text-size=15
text-color=1;1;1;1;
text-bg-color=0.3;0.3;0.3;1
font=Serif
show-clock=1
clock-x-offset=800
clock-y-offset=820
clock-text-size=12
clock-color=1;0;0;0
nvbuf-memory-type=0

[streammux]
##Boolean property to inform muxer that sources are live
live-source=1
```

```
batch-size=1
##time out in usec, to wait after the first buffer is available
##to push the batch even if the complete batch is not formed
batched-push-timeout=40000
## Set muxer output width and height
width=1280
height=720

[primary-gie]
enable=1
gpu-id=0
batch-size=1
gie-unique-id=1
interval=0
labelfile-path=ssd_coco_labels.txt
model-engine-file=sample_ssd_relu6.uff_b1_fp32.engine
config-file=config_infer_primary_ssd.txt
nvbuf-memory-type=0
```

Apêndice C – Configuração inferidor primário *NVDSInfer*

```
[property]
gpu-id=0
net-scale-factor=0.0078431372
offsets=127.5;127.5;127.5
model-color-format=0
model-engine-file=sample_ssd_relu6.uff_b1_fp32.engine
labelfile-path=ssd_coco_labels.txt
uff-file=sample_ssd_relu6.uff
uff-input-dims=3;300;300;0
uff-input-blob-name=Input
batch-size=1
## 0=FP32, 1=INT8, 2=FP16 mode
network-mode=2
num-detected-classes=91
interval=1
gie-unique-id=1
is-classifier=0
output-blob-names=MarkOutput_0
parse-bbox-func-name=NvDsInferParseCustomSSD
```

```
custom-lib-path=nvdsinfer_custom_impl_ssd/libnvdsinfer_custom_impl_ssd.so

[class-attrs-all]
threshold=0.5
roi-top-offset=0
roi-bottom-offset=0
detected-min-w=0
detected-min-h=0
detected-max-w=0
detected-max-h=0
```

Apêndice D – Conversor *python tegrastats excel*

```
import pandas as pd
import re

df=pd.DataFrame(columns=['RAM','CPU_1','CPU_2','CPU_3','CPU_4','GPU])
idx = 0

with open('tregralog.txt','r') as file:
    for line in file.readlines():
        ram_idx = line.find('RAM')
        df.at[idx,'RAM']= line[ram_idx+4:ram_idx+8]

        cpu_idx = 0
        for i in range(1,5):
            cpu_idx = line.find('%', cpu_idx)
            cpu=line[cpu_idx-3:cpu_idx]
            virg = cpu.find(',')
            if virg!=-1:
                cpu = cpu[virg+1:]
            cpu = int(re.sub('\D',"",cpu))
            cpu_idx+=1
            col = 'CPU_'+str(i)
            df.at[idx,col]= cpu

        gpu_idx = line.find('GR3D_FREQ')
        try:
            gpu=int(line[gpu_idx+9:gpu_idx+12])
        except:
```

```
            gpu=int(line[gpu_idx+9:gpu_idx+11])

        df.at[idx,'GPU']= gpu
        idx+=1
    df.to_excel('tegra.xlsx')
```

Apêndice E – Resultados do estudo de caso

Figura 34 - Evento com detecção de pessoas

Captura de tela da aplicação mostrando a detecção correta da rede neural SSD_Inception_V2 em que as pessoas são detectadas. Embora a imagem contenha dois objetos, o evento foi contado apenas como um positivo verdadeiro.

Fonte: Acervo do autor (2019)

Figura 35 – Evento com detecção de múltiplos carros

Captura de tela da aplicação mostrando a detecção correta da rede neural SSD_Inception_V2 em que há um carro estacionado e outro em movimento. A detecção foi considerada um positivo verdadeiro.

Fonte: Acervo do autor (2019)

Figura 36 – Evento com detecção de carro e pessoa

Captura de tela da aplicação mostrando a detecção correta da rede neural SSD_Inception_V2 em que há um pedestre em movimento, contando como positivo verdadeiro.

Fonte: Acervo do autor (2019)

Figura 37 - Evento com detecção de pessoa

Captura de tela da aplicação mostrando a detecção correta do modelo de um pedestre. O evento foi contablizado como positivo verdadeiro.

Fonte: Acervo do autor (2019)