Construindo aplicativos de visão computacional voltados a dispositivos na borda da rede com Gstreamer e NVidia DeepStream

Alison do Rosário de Morais
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, Barbalho, Salvador, Bahia
armorais01@gmail.com

Antônio Carlos Santos Souza
Instituto Federal de Educação, Ciência e Tecnologia da Bahia
Rua Emídio dos Santos, Barbalho, Salvador, Bahia
antoniocarlos@ifba.edu.br

RESUMO

O crescimento no poder de processamento de computadores de menor custo, aperfeiçoamento das técnicas de aprendizado de máquina, explosão da Internet das Coisas, o crescimento da computação na borda da rede e crescente investimento de grandes empresas tem beneficiado e impulsionado a evolução da visão computacional. Este trabalho tem como foco a utilização de modelos de inteligência artificial e frameworks de streaming multimídia para a criação de um aplicativo modular de análise de vídeo em tempo real.

Palavras-chave

visão computacional; computação na borda; redes neurais artificiais.

1. INTRODUÇÃO

Visão Computacional, ou Computer Vision em inglês, é a forma com que conseguimos fazer os computadores enxergarem o mundo. Apesar do grande número de solução que utilizam a visão computacional para resolver seus problemas, ainda é uma área considerada em estágio de desenvolvimento inicial. Dentre as muitas áreas que se beneficiam do uso de técnicas de visão computacional, podemos citar algumas que as utilizam desde seu estágio inicial até outras que somente nos últimos anos estão se beneficiando com os avanços da utilização de redes neurais profundas:

- Robótica - a robótica, a visão computacional e a inteligência artificial andam juntos, como disse Brooks ainda em 1991[1].
- Indústria - comumente utiliza-se a visão computacional, inclusive junto com a robótica, para avaliar a qualidade do produto final ou detectar situações de perigo. Também é utilizada para controle de processos.
- Medicina - a análise de imagens para detecção e diagnósticos de doenças é uma realidade e tem tido um grande aumento nos estudos nos últimos anos. [2]
- Segurança - na segurança são usadas desde técnicas clássicas de reconhecimento facial, consideradas por alguns quase primitivas, até outras complexas de reconhecimento de comportamento suspeito e reidentificação de pessoas por Soft Biometrics [3].
- Automobilística - existe um grande crescimento atualmente no interesse em automóveis autônomos. Os carros autônomos são feitos para ser inteligentes e precisam detectar uma série de objetos e possíveis caminhos e tomar decisões rapidamente.
- Jogos e entretenimento - o Kinect da Microsoft provê um sensor de movimento de corpo inteiro e reconhecimento de gestos para a plataforma Xbox, ainda muito utilizado em pesquisas, principalmente pelas cameras estéreo que facilitam o cálculo de profundidade dos objetos detectados [4].

Até poucos anos atrás, realizar análise de imagens utilizando computação era algo custoso e demorado. Algumas das áreas citadas foram extremamente beneficiadas pelo queda no preço nas duas últimas décadas, tanto das câmeras utilizadas para captura das imagens, quando do hardware necessário para realizar o processamento e a análise dos frames capturados, além da avanço considerável no poder de processamento, principalmente nas placas gráficas, memória RAM e CPU dos computadores. Atualmente é possível aplicar filtros relativamente pesados em imagens em smartphones de baixo custo, como em aplicativos de simulação de envelhecimento. Naturalmente, como até os smartphones, que são aparelhos móveis pessoais, já possuem um poder de processamento considerável, hardwares especializados em processamento próximos à borda da rede começaram a surgir. Entre eles, temos as placas de computação embarcadas criadas especialmente pela NVIDIA para aceleração de aplicações que utilizam aprendizado de máquina, a série NVIDIA Jetson, que tem como foco a utilização em IoT e cidades inteligentes [5] .

O surgimento e o crescimento de aparelhos que fazem o processamento na borda da rede de imagens capturadas de diversas formas é justificado pelo fato da quantidade gigantesca de dados que são no máximo visualizados em forma de vídeo em monitores, sem haver qualquer tipo de processamento. Em alguns casos, as imagens são enviadas em tempo real para um servidor, que pode ser na nuvem, onde é realizado o processamento da imagem, o que cria um custo extra em transferência e armazenamento de dados já que nem todos os frames apresentam algo de valor para a aplicação. O processamento desses dados na borda garante que o que seja enviado para o servidor sejam os dados já processados, como visto na figura 1, o que diminui também o uso de banda e a

latência, principalmente quando há a necessidade de inserir novas câmeras. Além disso, em alguns casos, pode ser mais seguro o processamento dessas imagens no mesmo local onde são geradas quando elas têm conteúdo sensível [6].

Se a queda de preço e aumento do poder de processamento do hardware, principalmente das GPUs, fazem uma máquina relativamente simples ter a capacidade de realizar análise de imagens e vídeos em muitos casos de forma satisfatória, os estudos que já foram publicados, os estudos que estão sendo realizados atualmente, e as ferramentas que hoje temos disponíveis para processamento de imagens em tempo real, streaming de dados, aprendizado de máquina e redes neurais nos proporcionam um ganho imenso de produtividade ao criar nossas aplicações. Ferramentas como a biblioteca de processamento de imagens OpenCV, os frameworks de streaming multimídia GStreamer e Nvidia DeepStream, bibliotecas de redes neurais como Keras e Torch, e detectores de objetos como DetectNet e Faster R-CNN são algumas dessas ferramentas que estão sempre em crescimento e serão utilizadas ou discutidas neste trabalho.

Este trabalho visa a criação de uma aplicação utilizando pipelines e plugins criados para inferência de vídeos utilizando modelos de inteligencia artificial, otimizado para processamento de fluxo de vídeo, facilitando trabalhar com tipos de entradas diferentes, como câmeras USB, câmeras ip e arquivos de vídeo. Utilizando o SDK DeepStream da NVIDIA, baseado no framework GStreamer, é esperado alto throughput mesmo quando o número de fontes de dados aumente, por eliminar problemas relacionados a trafego de dados entre processos. Para alcançar esse objetivo, a intenção é que a aplicação tenha o mínimo de processamento feito em CPU, aproveitando-se dos benefícios da paralelização oferecidos por plugins especificamente feitos para aceleração por hardware, e somente envie o resultado das inferências.

2. REFERENCIAL BIBLIOGRÁFICO

2.1 GStreamer

O GStreamer é um framework multiplataforma escrito na linguagem C que tem como meta a criação de aplicações de streaming multimídia utilizando vários filtros. Entre o uso mais comuns do GStreamer estão aplicativos de reprodução, edição e criação de audio e vídeo, além de serviço de streaming. O framework tem como característica muito importante o uso intenso de plugins para cada etapa do processamento dos dados, inclusive com a possibilidade de criação

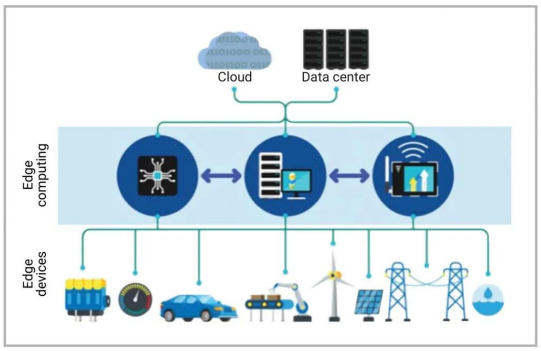

Figura 1: Infraestrutura da computação de borda. Fonte: [7]

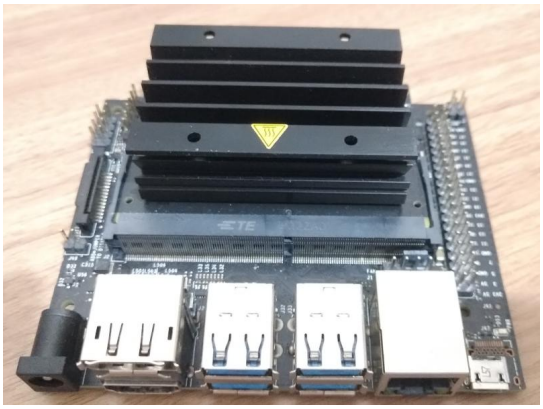

Figura 2: NVIDIA Jetson Nano Development Kit.

e utilização de plugins de terceiros, como é visto na figura 3.

O Gstreamer usa o conceito de fluxo de dados utilizando pipelines apresentado na Oregon Graduate Institute em 2001 [8], com a idéia de inserir elementos como filtros para manipulação dos dados e buffers para trânsito de dados e seus metadados.

A Figura 4 mostra o fluxo de um pipeline básico do GStreamer para rodar um arquivo ogg.

2.1.1 *Elementos*

Cada item no pipeline é chamado de Elemento, e cada elemento é um plugin que é utilizado para uma função específica. Usando como exemplo o pipeline da figura 3 temos os seguintes elementos:

- file-source - é o primeiro elemento no pipeline de exemplo, é responsável por carregar um arquivo localizado em disco. Os elementos do tipo source são usados para produzir dados para o pipeline.
- ogg-demuxer - é um demultiplexador que age sobre o arquivo ogg carregado pelo file-source para seus componentes de áudio e vídeo codificados.
- vorbis-decoder e theora-decoder - decodificam respectivamente os componente de áudio e vídeo entregues pelo elemento ogg-demuxer.
- audio-sink e video-sink - reproduzem, respectivamente, os componentes de audio e vídeo. Os elementos do tipo sink são usados para consumir dados da pipeline.

Os elementos são utilizados em cadeia e são linkados para que a os dados possam fluir entre eles. O framework GStreamer já disponibiliza uma grande quantidade de componentes, incluindo diferentes tipos de sources, sinks, filtros, multiplexadores e demultiplexadores entre outros.

2.1.2 *Pads*

Os elementos do pipeline precisam de algo que conecte uns aos outros em cadeia, e esse é exatamente o trabalho dos pads. Como é possível ver na figura 3, os elementos são conectados por portas que comunicam-se entre si. Existem dois tipo de pads: source e sink.

- source - é por onde os dados saem de um elemento. Naturalmente, um elemento do tipo source, responsável por alimentar o pipeline com dados externos, só tem o pad do tipo source, já que ele não consome dados de nenhum elemento do pipeline. É possível um elemento ter mais que um source. Na figura 3 temos o elemento ogg-demuxer que tem dois pads do tipo source, um para transmitir os dados de áudio e outro para transmitir os dados de vídeos.
- sink - é por onde os dados entram um elemento. Naturalmente, um elemento do tipo sink, responsável por consumir os dados do pipeline e gerar uma saída, só tem o pad do tipo sink, já que ele somente consome dados da pipeline, sem alimentar nenhum outro elemento. É possível um elemento ter mais que um sink em um elemento, como por exemplo, em elementos multiplexadores.

Os dados que percorrem o pipeline sempre vão transitar dos sources para os sinks, e cada pad tem um conjunto de propriedades chamados de capabilities, ou caps, que são utilizados para validar o tipo de dados que trafegam pelo pipeline entre os elementos. Os caps existem para evitar que, por exemplo, um sink de um elemento que consome áudio não consuma dados de um source de um elemento que transmita vídeo, ou até para garantir que os dados de entrada seja de um tipo exato de formato, como é o caso do sink do elemento ogg-demuxer.

2.1.3 *Módulos do GStreamer*

O framework GStreamer está dividido em diferentes módulos. Além do núcleo, que tem as funcionalidades e elementos padrão.

- Base - são plugins e elementos muito bem implementados e bem mantidos pelos colaboradores. Também fornece bibliotecas e classes base para ajudar no desenvolvimento de plugins para o GStreamer. Contém uma variedade de codecs, conversores e filtros, e é um bom ponto de partidas para desenvolvedores que precisam de referência para o desenvolvimento de bons plugins.
- Good - são plugins considerados bem implementados e bem mantidos, porém geralmente não são tão comumente utilizados quantos os plugins do módulo Base. Utilizam a licença LGPL.
- Bad - plugins que não tem qualidade comparável ao restante dos módulos. Geralmente não são realmente desastrosos, mas necessitam de mantenedores, documentação ou revisão dos seus códigos. Não são recomendados para uso em aplicações críticas ou ampla utilização. É encorajado pela comunidade que colaboradores que tenham interesse realizem melhoras nesses plugins.

  Ugly - são plugins que funcionam bem e com boa qualidade, mas com licença nos plugins ou nas bibliotecas que eles usam que não é livre. A utilização desses plugins em aplicações comerciais podem resultar em quebras de patente.

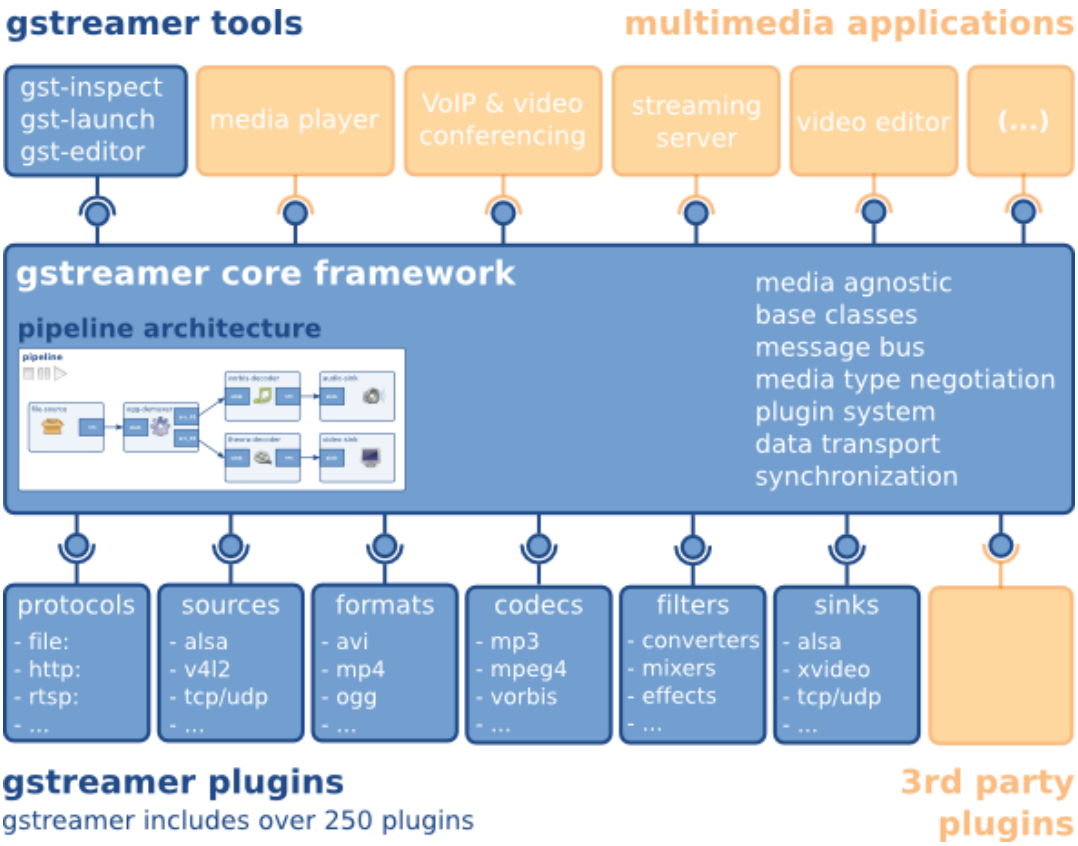

Figura 3: Overview do GStreamer. Fonte: [9]

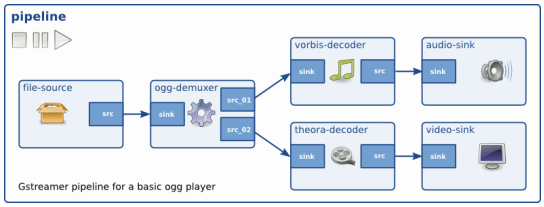

Figura 4: Pipeline simples do GStreamer para rodar arquivos oggz. Fonte: [9]

RTSP server - Servidor de Real Time Streaming Protocol, que garante a transferência de dados em tempo real

Python Bindings - Garante aos desenvolvedores o desenvolvimento de aplicações e plugins para o GStreamer utilizando a linguagem Python. Os templates de plugins no momento da realização deste trabalho estão desatualizados comparando com os plugins desenvolvidos utilizando C/C++, mas para funcionamento básico funcionam satisfatoriamente.

2.2 GStreamer e visão computacional

Na visão computacional, principalmente nas formas mais tradicionais, onde muitas vezes precisamos descobrir regiões de interesse (ROI, do inglês Regions of Interest), é muito comum a utilização do processamento dos frames dos vídeos. Em algumas técnicas menos sofistificadas de detecção de faces, como os propostos por Michael Jones e Paul Viola em 2003 no artigo Fast multi-view face detection [10], é necessário que os frames estejam em escala de cinza. Em alguns casos de detecção de dígitos, inclusive de placas, também pode haver a necessidade de aplicar alguns filtros para remover ruídos da imagem e facilitar a detecção.

Por o GStreamer ter uma grande ênfase na modularização das aplicações utilizando plugins e pela simplicidade de implementação de um novo plugin para propósito específico, como aplicar uma série de filtros nos frames de uma câmera e a análise do vídeo por algorítmos específicos para cada caso, a utilização do framework em conjunto com outros frameworks em biblioteca ajuda a acelerar e a customizar processos [11]. Alguns plugins disponíveis no GStreamer já oferencem a capacidade de processamento dos frames, mas em casos específicos há a possibilidade da utilização de ferramentas externas, como o OpenCV.

2.3 OpenCV

A biblioteca OpenCV, Open Source Computer Vision, é uma das ferramentas mais importantes para o processamento de imagens em tempo real. Foi lançada oficialmente em 1999, sendo desenvolvida inicialmente pela Intel. A biblioteca é escrita em C++, mas existem bindings em Python, MATLAB e JAVA. O lançamento da versão 2.0 do OpenCV em 2009 trouxe grandes melhorias de performance e importantes modificações da interface em C++, mas foi o suporte à biblioteca NumPy, que suporta arrays e matrizes multidimensionais, que agiliza bastante o desenvolvimento. A versão 4 trouxe suporte ao C++ 11, melhoria de performance e o início do suporte ao Android, além de ter eliminar grande parte da API disponível em C.

2.4 Detecção de Objetos

Durante a detecção de objetos, basicamente é detectado um objeto em um frame, e é colocado um "bounding box"em torno dele e então ele é classificado. Em cada frame podem ser detectados vários objetos. A forma em que enxergamos as imagens e a forma que o computador as enxerga é diferente. O que para nós é claramente um cão em uma imagem, para o computador é uma matriz grande de números. Para nós, diferenciar uma imagem contendo um cão de outra imagem contendo um urso é uma tarefa fácil, mas para o computador, continua sendo uma comparação entre duas matrizes com números. A tarefa parece ser ainda mais difícil quando se trata de reconhecer e diferenciar faces.

Quando criamos uma aplicação que utiliza visão computacional devemos levar em consideração vários fatores que podem prejudicar a detecção de objetos [12].

- Oclusão - é quando um objeto sobrepõe outro no frame. Na figura 5 temos a imagem de óculos parcialmente visível, ocluído de nossa visão. Um bom algoritmo de deteção de objetos deve ser capaz de detectá-los mesmo que os mesmo estejam cem porcento visíveis.
- Variação de escala e modelos - continuando como óculos como exemplo, um bom algoritmo de detecção deve ser capaz de aceitar obejetos de tamanhos variados e modelos diferentes. Um par de óculos infantil continua sendo um par de óculos, e o algoritmo deve detectá-lo como tal.
- Variação de ponto de vista - dado como exemplo a figura 6a, onde os oculos são perfeitamente visíveis e fáceis de detectar, o algoritmo deve considerar também mudança de posição do objeto e mudança do ponto de vista, como modificação do posicionamento de uma câmera, como mostrado na figura 6b, que tem o mesmo par de óculos da figura 6a visto de uma vista de cima para baixo.
- Ruído - ruído é o excesso de informação do fundo de um objeto na imagem que pode levar a dificuldades da identificação.

Figura 5: Exemplo de oclusão de um objeto.

2.4.1 *Rastreamento de objetos detectados*

Para resolver alguns problemas usando visão computacional, somente detectar o objeto não é o suficiente, é necessário identificá-lo por diversos frames. Cada objeto rastreado precisa ter um ID atribuído no momento da detecção inicial, e esse ID deve ser mantido enquanto o objeto se move nos frames de um vídeo. Para que o ID seja mantido por vários frames consecutivos, é importante que a detecção seja consistente em cada um deles, já que muitas da formas de

(a) Óculos visto de frente.

(b) Os mesmos óculos vistos de cima.

Figura 6: Computadores de processamento na borda.

se rastrear um objeto envolve calcular o quanto a área de seu bounding box se assemelha com a de um objeto de um frame anterior. Por demonstrar resultado interessantes, a popularização das Redes Neurais Artificiais são um ponto importante na história da visão computacional.

2.5 Redes Neurais e Aprendizado de Máquina

O cérebro humano é formado por bilhões de neurônios, que têm o dever de transmitir impulsos nervosos. Os neurônios são basicamente divididos em 3 partes:

- Dentritos - Extensões que captam sinais de outros neurônios
- Corpo celular - É o núcleo do neurônio, e basicamente processa os sinais recebidos
- Axônio - Transmitem os sinais processados para outras células ou neurônios

As redes neurais foram criadas para simular a capacidade dos neurônios de comunicarem-se entre si e processarem sinais, utilizando nós interconectados de modo a formar uma rede. Ainda que a tecnologia esteja em rápido crescimento nos últimos anos, a analogia entre atividades de células nervosas e circuitos elétricos já era abordada em estudos na década de 1940 nos EUA[13].

Redes neurais básicas são formadas por três tipos de camadas: entrada, oculta e saída. Os dados entram pela camada de entrada, são processados nas camadas ocultas, e o resultado final é fornecido na camada de saída. o número de nós em cada camada depende do tipo de dados da camada de entrada, e qual tipo de problema a rede foi feita para resolver.

2.5.1 Redes Neurais Convolucionais

Tempos atrás era custoso resolver problemas de visão computacional utilizando redes neurais artificiais, mas em 1998 já eram estudadas técnicas de aprendizado de máquina para reconhecimento de caracteres escritos a mão[14]. O mesmo autor do estudo citado anteriormente, Yann Lecun, já tinha como proposta o uso de redes neurais convolucionais para extração de características de objetos em imagens[15].

Redes neurais convolucionais são tipos de redes profundas, ou seja, com muitas camadas. Nas camadas iniciais, próximas à camada de entrada, geralmente realizam filtros na imagem como um todo, como detecção de bordas, e extração de características, enquanto as camadas ocultas mais profundas são responsáveis por classificar os objetos na imagem. Desde que a rede AlexNet [16] venceu a ILSVRC (ImageNet Large Scale Visual Recognition Competition) em 2012, o uso de Redes Neurais Convolucionais se tornou extremamente comum para classificação de objetos.

2.5.2 Treinamento de uma rede neural artificial

Para ensinar uma rede neural, o primeiro e mais importante passo é ter os dados baseados no que a rede deverá apresentar como saida. Existem tipos diferentes de aprendizado utilizados em aprendizado de máquina:

- Aprendizado supervisionado: o algoritmo de aprendizado de máquina recebe tanto os dados de entrada quanto as saídas esperadas. O algoritmo então tenta encontrar padrões que o faça chegar ao output desejado. Baseado no que o algoritmo aprende e nos resultados anteriores, a cada iteração do treinamento ele se ajusta para ter resultados melhores.
- Aprendizado não-supervisionado: Os algoritmos recebem os dados de input, sem nenhum informação extra qye os descreva, e tenta descobrir características dos inputs. É mais complexo que o aprendizado supervisionado, pricipalmente quando utilizado para classificar imagens.

Atualmente existem estudos importantes sendo feitos em aprendizado não-supervisionado e semi-supervisionado para classificação e detecção de objetos em imagens, mas os mais importantes frameworks e toolkits de aprendizado de máquina têm o aprendizado de máquina supervisionado como base, ou seja, esperam que o usuário ofereça um conjunto de dados que contenha as imagens como dados de entrada e o resultado esperado. No caso da classificação de imagens, é esperado a classe a qual o objeto que está em uma imagem pertence, por exemplo, um ”carro”. No caso de detecção de objetos, é esperado além da classe do objeto, como um ”carro”, os bounding boxes, que são retângulos que marcam a região onde aquele objeto se encontrem na imagem.

Os passos básicos para treinar uma rede neural são os seguintes:

- Criar o dataset: Tendo como exemplo este trabalho, para treinar uma rede neural que seja capaz de detectar objetos, é necessário um conjunto de imagens, assim como os dados das categorias dos objetos e a localização nas imagens. Quando a intenção é treinar com múltiplas categorias, é importante se preocupar em ter um número parecido de imagens para cada uma delas, para evitar problemas de viés para categorias que são mais representadas. Também é importante ter uma grande quantidade de dados de cada categoria, e que cada categoria tenha exemplos diferentes dos objetos representados. Por exemplo, quando é possível, ter o

mesmo tipo de objeto com cores e plano de fundo diferentes, vistos de ângulos diferentes, assim como iluminação variada. Também podem ser utilizadas técnicas de "data augmentation", que aplicam transformações nas imagens para aumentar a diversidade dos dados.

- Dividir o dataset: Separar os dados que serão utilizados para treinamento da rede e para teste após o treinamento. Os dados de treinamento servem para que o modelo de machine learning faça predições a partir dos dados de entrada, e se corrija caso o resultado esteja errado. Após o treinamento, o conjunto de treinamento é utilizado para avaliar o desempenho.

- Treinar a rede neural: Com os dados de treinamento, a rede aprende a reconhecer cada uma das categorias dos dados anotados. Durante o treinamento, os dados são quebrados em lotes, ou batches, e o tamanho de cada batch depende do tamanho da memória utilizada. Os batches são enviados um a um para treinamento, e quando todos os batches são usados, é finalizado o que é chamado de época, ou epoch. Um treinamento deve ter diversas épocas, onde um algoritimo de regressão linear com gradiente descendente realiza o ajuste de parâmetros para que os resultados se ajustem aos dados.

- Avaliar os resultados: As imagens do conjunto de dados de teste são apresentados à rede, e os resultados das predições são comparados como os dados das categorias e localização nas imagens.

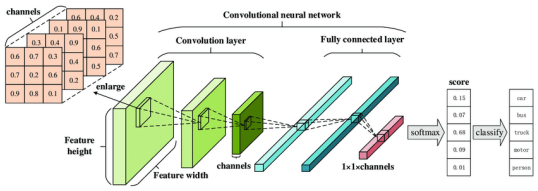

Figura 7: Rede Neural Convolucional. Fonte [17]

O modelo que resulta de um treinamento é exportado pelo algoritmo, geralmente sendo chamados de pesos (weights) ou checkpoints. Esses pesos são carregados pelos aplicativos que realizarão as inferências, mas podem também ser utilizados para treinar novamente o modelo.

2.5.3 *Transfer Learning*

Apesar dos avanços na tecnologia, criar uma rede neural e treinar um modelo do zero é algo que pode ser extramamente custoso. Conseguir um conjunto de dados grande e bom o suficiente para ter um bom resultado muitas vezes custa muito dinheiro e tempo, e treinar um modelo de IA com milhões de imagens pode levar semanas, mesmo utilizando placas de vídeos específicas, como a V100 da NVidia.

Transfer learning é um método que utiliza um modelo que já foi treinado como ponto inicial do treinamento, quando as característica da rede do modelo pré-treinado se adequam à tarefa a ser realizada pelo modelo alvo.

Atualmente é possível facilmente encontrar pesos de modelos usando frameworks específicos, como Keras ou Torch, tendo como base redes específicas, como Resnet50 ou VGG, pré-treinados usando conjuntos de dados com milhões de imagens como do Coco da Microsoft[18], ou ImageNet [16]. Re-treinar um modelo de classificação de carros com novos dados, a partir de um checkpoint similar aos citados anteriormente, que foi treinado como milhões de imagens de diferentes modelos, pode diminuir uma tarefa que poderia demorar semanas para algumas horas e economizar em quantidade de imagens.

2.6 NVIDIA DeepStream

O DeepStream SDK da NVIDIA disponibiliza um toolkit livre para uso acadêmico e comercial que facilita desenvolver e colocar em produção aplicativos voltados para análise de vídeos. O DeepStream faz parte do NVIDIA Metropolis, uma plataforma de ferramentas da NVIDIA voltada para desenvolvimento de aplicativos voltados à inteligência artificial e Internet das Coisas. O SDK DeepStream utiliza o GStreamer como base para alcançar alto throughput e baixa latência em aplicações que seguem o mesmo modelo de desenvolvimento em pipelines, como mostrado na figura 8, porém com plugins implementados pela NVidia para aceleração por hardware. A aceleração por hardware dos plugins do Deepstream pode não ser extremamente importante em aplicações simples de streaming, mas sim quando são utilizados para a criação de aplicações de análise de vídeo complexas utilizando inteligencia artificial, pois o foco principal do SDK é utilizar redes neurais profundas em um pipeline de processamento de dados em fluxo.

2.6.1 *Plugins do Deepstream*

A Nvidia disponibiliza no SDK uma série de plugins para GStreamer otimizados para uso em suas GPUs, como decodificadores, multiplexadores e renderizadores acelerados. Segue abaixo outros plugins que geralmente exerce funções importantes em pipelines de aplicativos criados utilizando Deepstream.

- Gst-nvinfer - realiza inferência nos dados de entrada utilizando o SDK de inferência TensorRT que usa deep learning e otimiza as inferências com a promessa de entregar baixa latência e alto throughput para aplicações.

- Gst-nvtracker - rastreia objetos detectados. Cada novo objeto recebe um ID único.

- Gst-nvosd - desenha bounding boxes e textos. O plugin recebe do elemento upstream um buffer com metadados. O formato, cor e preenchimento dos bounding boxes dependem das configurações. Os parâmetros dos textos são configuráveis por metadados.

- Gst-tiler - Renderiza multiplos frames de diferentes fontes lado a lado.

- Gst-nvmsgconv - plugin responsável por gerar metadados. Recebe um buffer como entrada e fornece como saída o mesmo buffer com metadados adicionais gerados pelo plugin. Entre os dados fornecidos estão informações sobre cada objeto detectado, status de módulos e eventos.

3. TRABALHOS CORRELATOS

Li, Qi e Banerjee [20] da Universidade de Wisconsin criaram o framework EdgeEye, com a intenção de habilitar

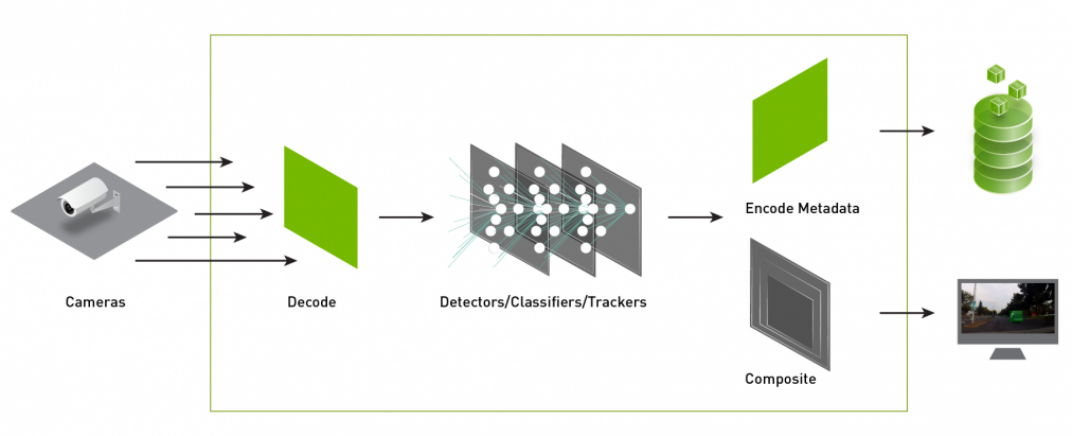

Figura 8: Características de um aplicativo básico com DeepStream. Fonte [19]

desenvolvedores a construir aplicações com modelos de redes neurais profundas treinados em frameworks populares. Essas aplicações seriam utilizadas para analise de vídeos em tempo real. O EdgeEye é um framework que, assim como o DeepStream, tem a intenção de ser modular, baseado no GStreamer e tem a mesma missão de simplificar e acelerar o processo de desenvolvimento.

Em um trabalho recente, lançado no início de 2020, Prodaiko [21] propôs um sistema que rastreia pessoas utilizando multiplas câmeras que filmam de cima para baixo. Prodaiko utilizou o DeepStream SDK para realizar inferência em múltiplas câmeras, detectando pessoas e aplicando técnicas avançadas de re-identificação para que a mesma pessoa vista em diferentes câmeras não seja identificada como uma pessoa diferente. Prodaiko também utilizou o fluxo no pipeline para, com um plugin, realizar a calibração da câmera calculando a matriz de transformação. Prodaiko concluiu que a utilização do Deepstream SDK tornou o sistema modular, conseguindo substituir qualquer parte facilmente, pois cada módulo é representado por plugins.

4. CARACTERÍSTICAS DO TRABALHO

Como toda aplicação baseada no framework GStreamer, este trabalho tem como foco a criação de uma solução baseada em plugins, com elementos que podem ser modificados e substituídos sem a necessidade de afetar outros elementos. Como o Deepstream SDK, que também é baseado no GStreamer, tem como uma das dependências a presença de uma GPU Nvidia, a aplicação rodará somente em máquinas com GPUs que suportam CUDA. Além disso, o máquina precisa ter instalados os drivers da Nvidia, GStreamer, CUDA, TensorRT, e de preferência o sistema operacional Ubuntu, homologado pela Nvidia.

A aplicação de inferência aceita como entrada no mínimo 2 câmeras, configuradas em um arquivo de configuração. Cada câmera precisa ter um nome, um endereço, e suas regiões de interesse configuradas neste arquivo, para que essas informações sejam carregadas pelos plugins do pipeline.

4.1 Aplicativo de inferência e processamento

4.1.1 Source

O primeiro plugin da pipeline sempre deve ser um plugin do tipo Source. A intenção principal é que esse plugin tenha como entrada frames de uma câmera. Nessa aplicação, as entradas são carregadas a partir de um arquivo yaml, e precisa ter no mínimo 2 entradas. As entradas poder ser arquivos de vídeo ou stream RTSP. No início da execução da aplicação, para cada fonte de dado é criado um elemento deste tipo.

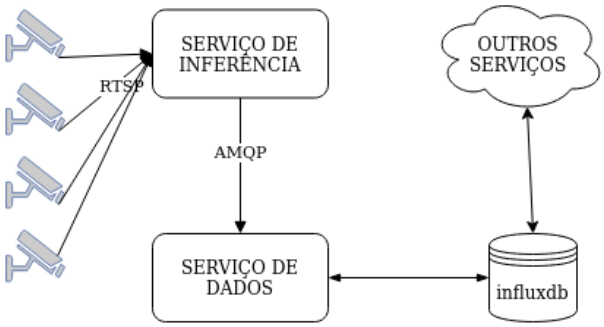

Figura 9: Visão básica da solução

4.1.2 Streamux

É um plugin importante no fluxo do pipeline, pois funciona como um demultiplexador, dividindo batches de frames que vieram do source em buffers individuais. Ele tem uma cópia do endereço de memória (de GPU) de cada frame do lote. Os metadados deste plugin, NvDsBatchMeta, muitas vezes são necessários para realizar algumas operações.

4.1.3 Inferência primária - Detecção de Veículos

É um plugin de inferência, gst-nvinfer, padrão no DeepStream e criado pela NVidia. É responsável por carregar os pesos do modelo treinado e realizar a inferência. Pode-se dizer que este plugin é o coração da aplicação, já que sem a detecção de objetos, nenhum dado de output seria gerado.

Para este trabalho, foi utilizado como base o modelo pré-treinado DetectNetv2 TrafficCamNet da NVIDIA, baseado na ResNet18. O dados utilizados pra treinamento foram retirados de 500 frames de cada um dos 6 vídeos do dataset disponibilizado junto ao artigo Automatic Camera Calibration for Traffic Understanding apresentado na British Ma-

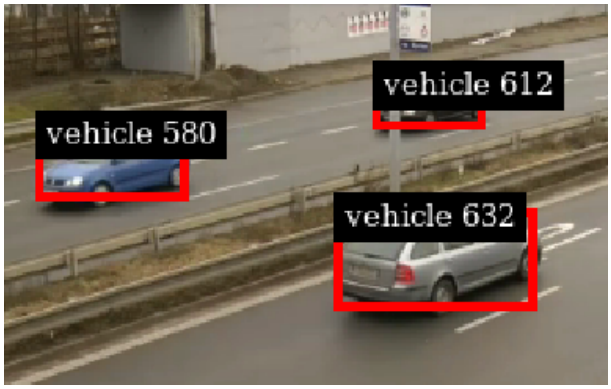

Figura 10: Veículos com bounding boxes da detecção e IDs do tracking

chine Vision Conference 2014[22]. O treinamento durou 200 épocas, utilizando a ferramenta Transfer learning Toolkit da NVIDIA.

4.1.4 *Inferência secundária - Classificação de Tipo de Veículos*

Também utiliza o plugin gst-nvinfer, para fazer classificação dos veículos detectados. Carrega os pesos do modelo de classificação de tipo de veículos VehicleTypeNet, disponibilizado pela NVIDIA, sem modificações. No momento deste trabalho, as classes disponíveis para esse modelo são: ”coupe”, ”sedan”, ”SUV”, ”van”, ”large vehicle”e ”truck”.

4.1.5 *Tracker*

Para rastreamento dos objetos detectados, é utilizado o plugin gst-nvtracker, padrão no DeepStream e criado pela NVidia. Foram utilizadas as configurações padrão de tracker (KLT Tracker).

4.1.6 *Contador de veículos e envio de mensagens*

É um plugin customizado, escrito em Python 3. Tem a tarefa de contar quantos veículos foram detectados em cada fonte, além de contar quantos veículos cruzaram certas regiões. As regiões de interesse de detecção, assim como as de contagem de veículos, devem ser previamente configuradas em um arquivo yaml.

A contagem de detecção serve para quantificar a intensidade do tráfego naquele momento e utiliza como base os metadados do detector de objetos do plugin gst-nvinfer. A contagem de quantos veículos passaram por uma certa região serve para saber, naquele período de tempo, qual o sentido em que os veículos estão seguindo com maior intensidade, e utiliza como base os metadados de rastreamento do plugin gst-nvtracker, assim como os metadados do detector de objetos do plugin gst-nvinfer. Para fazer a contagem de veículos que passam em uma região, é traçada uma linha, utilizando 2 pontos configurados no arquivo de configuração yaml, e é levado em consideração quantas vezes o bounding box de objetos com IDs diferentes passaram por esta linha.

A abordagem para contagem de veículos é um cálculo simples baseado na distância entre o centróide do objeto detectado e rastreado, e a linha que demarca a região de contagem:

```
linha = (p1, p2)
p3 = numpy.asarray(centroide_do_objeto)
distancia = numpy.linalg.norm(numpy.cross(p2-p1, p1-
p3))/numpy.linalg.norm(p2-p1)
```

Já que a contagem depende do objeto de mesmo ID ser detectado 2 vezes próximo da linha, o sentido em que o objeto se move é encontrado por distância[1]-distância[0].

Por limitações na API python do Deepstream no momento deste trabalho, o plugin também têm a tarefa de enviar mensagens de dados processados para uma fila utilizando o AMQP (Advanced Message Queuing Protocol). O software utilizado neste trabalho é RabbitMQ, que foi escolhido pela simplicidade de uso e por não utilizar muito recurso do sistema, em comparação com outras opções como o Kafka, inclusive em alvos onde é necessário se preocupar bastante com economia de recursos, como a plataforma Jetson Nano. O RabbitMQ usa, por padrão a porta 5672/TCP, para entregar e receber mensagens, e a porta 15672/TCP, para acessar a interface web de gerenciamento, quando este está disponível.

Segue abaixo um exemplo de modelo de mensagem:

```
per_frame_count_msg = {
    "measurement": "frameEvents",
    "tags": {
        "appId": "deepstream_roads_app",
        "sourceId": frame_meta.source_id,
    },
    "time": local_time,
    "fields": {
        "numObjects": frame_meta.num_obj_meta
    }
}

up_down_count_msg = {
    "measurement": "upDownEvents",
    "tags": {
        "appId": "deepstream_roads_app",
        "sourceId": frame_meta.source_id,
    },
    "time": local_time,
    "fields": {
        "up": up_count,
        "down": down_count
    }
}
```

Os dados foram modelados com a intenção de alimentar series temporais no influxdb.

4.1.7 *OSD*

O plugin de OSD é responsável por desenhar nas imagens do buffer resultados de plugins anteriores no pipeline. Por exemplo, é ele que desenha os bounding boxes, o nome das classes dos objetos detectados, assim como o ID, quando é utilizado o tracker. Como desenhar objetos nos frames é uma operação custosa e pelo ponto de vista geral desta aplicação, nesta aplicação é basicamente um plugin de debug, pois as imagens não são utilizadas após o fim do pipeline.

4.1.8 *Tiler e Sink*

O plugin de Sink é responsável por gerar um output no pipeline. Quando a aplicação está em modo de depuração, mostra o output em vídeo. Para exibir o output em tela,

se estiver sendo utilizado o container com docker é necessários ter as permissões necessárias na máquina host. Foi descartada a opção de gravar o output em arquivo, pois o objetivo da aplicação é enviar os dados colhidos em formas de mensagens. Em casos de produção, é mais interessante utilizar este plugin como "fakesink", que descarta todos os dados que chegam pelo upstream. 12

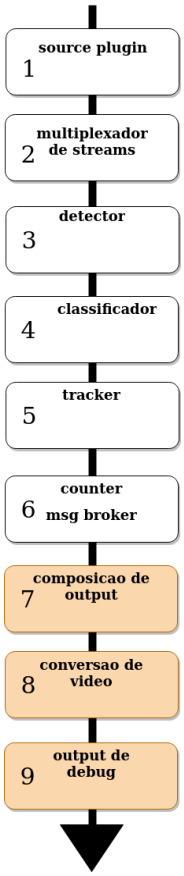

Figura 11: Fluxo básico da solução utilizando o SDK Nvidia Deepstream

4.2 Aplicação cliente

É um cliente básico que consome as mensagens da fila do rabbitmq e as escreve no influxdb. Como o foco da aplicação é a inferência na borda e envio das mensagens processadas para fora do pipeline, o propósito dele é somente consumir o resultado como uma série temporal para rápida visualização dos resultados.

```
def on_message(channel, method_frame,
                header_frame, body):
    json_body = json.loads(body)
    client.write_points(json_body)
    channel.basic_ack()
```

5. RESULTADOS OBTIDOS

Para realizar inferência em dispositivos na borda na rede, é importante que não exista o tráfegos de frames, que são grandes matrizes tridimensionais, de uma ponta a outra da aplicação. Esse objetivo foi alcançado, pois o output da aplicação são mensagens com o resultado da inferência e processamento desses dados.

O uso do DeepStream se mostrou favorável pelo fato de operações pesadas em frames serem feitas na GPU. Operações como redimensionamento dos frames seguido de detecção de objetos costumam ter grande impacto na performance dos aplicativos. Realizar o rastreamento de objetos com técnicas convencionais, como IoU (Intersection Over Union) utilizando cálculos em cpu também acabam afetando negativamente a performance.

A intenção inicial era que a aplicação, assim como os plugins fossem desenvolvidos em python, pelo benefício da utilização de bibliotecas otimizadas como o numpy, que oferece um ótimo suporte a arrays e matrizes multidimensionais, porém, no momento deste trabalho a API para python do Deepstream não oferece o suporte a criação de metadados customizados, sendo necessário modificação do código base original de bibliotecas, principalmente da biblioteca NvDsMeta. Por este motivo, o plugin de contagem e messege broker se fundiram em um só, por havia a necessidade de criar metadados específicos tanto para objetos quanto para frames. Se por um lado causa um problema de separação de preocupações, somente é necessário iterar sobre os metadados de cada frame, e cada metadado de cada objeto, em um só plugin.

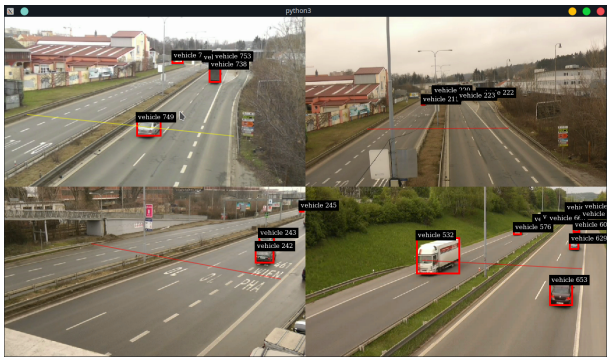

Figura 12: Output de debug com 4 cameras

A utilização do rabbitmq como gerenciador de fila se mostrou satizfatória, pois em nenhum momento durante a execução com o aplicativo consumir de testes ocorreu o acúmulo de mensagens na fila. É um resultado que se alinha com o objetivo de enviar uma carga de dados que não interfira muito com o tráfego da rede local, e nem tenha um aumento de custos muito algo, caso a aplicação de visualização dos dados obtidos esteja na núvem.

Durante testes, foi possível comparar a diferença entre o tamanho de cada mensagem e de um frame. Um frame de um vídeo do dataset utilizado, com dimensões de 854x480, ocupa aproximamente 500kB, espaço ocupado por 844 mensagens com dados processados. Foi utilizado o Grafana para consumir os dados do Influxdb, para avaliar se a idéia de persistir o resultado das detecções como séries temporais facilitaria a visualização posterior.

6. CONCLUSÃO

Todos os dias, aparelhos geram uma grande quantidade de dados de imagens que são descartadas após visualização, sem nenhum tipo de análise mais profunda. Quando surge

a necessidade de uma análise mais profunda dessas imagens para tomada de decisão, surge a necessidade cada vez maior de aplicações que sejam capazes de extrair o máximo de informações de cada frame com um mínimo de utilização de recursos computacionais.

Com a aplicação desenvolvida neste trabalho, foi possível utilizar modelos de inteligência artificial em cascata para extrair informações sobre o tráfego de veículos em rodovias de imagens vindo de fontes distintas, e plugins desenvolvidos especificamente para analisar, tratar e exportar os dados usando serviço de message broker. Utilizando a mesma base de código, é possível fazer o deploy da aplicação tanto em um PC comum com placa de vídeo dedicada quando em um mini-computador NVIDIA Jetson Nano baseado em arm64.

A utilização de um framework baseado em plugins como o GStreamer garantiu que a aplicação tenha como característica a reutilização dos mesmos plugins já desenvolvidos para outros casos de uso. Ter cada parte da resolução do problema de visão computacional em elementos do pipeline também tem como benefício a possibilidade de que alguns elementos não-específicos a um determinado problema chave fossem alterados sem necessidade de alterar o código dos outros plugins. Por exemplo, para melhorar o desempenho da aplicação no Jetson Nano, remover o elemento de inferência secundária, que classifica o tipo do veículo, não afeta o funcionamento dos outros plugins.

Para trabalhos futuros, estão sendo desenvolvidos plugins que, com base no conhecimento prévio do mapa da rodovia, seja possível calcular a velocidade aproximada, distância entre veículos e re-identificação de veículos em múltiplas câmeras baseada tanto nas características do veículo quanto na posição global calculada a partir da posicão do veículo na imagem da câmera.

7. REFERÊNCIAS

[1] R. A. Brooks, "New approaches to robotics," *Science*, vol. 253, no. 5025, pp. 1227–1232, 1991.

[2] M. De Bruijne, "Machine learning approaches in medical image analysis: From detection to diagnosis," 2016.

[3] G. Wang, J. Lai, P. Huang, and X. Xie, "Spatial-temporal person re-identification," pp. 8933–8940, 2019.

[4] Z. Zhang, "Microsoft kinect sensor and its effect," *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[5] M. Naphade, D. C. Anastasiu, A. Sharma, V. Jagrlamudi, H. Jeon, K. Liu, M.-C. Chang, S. Lyu, and Z. Gao, "The nvidia ai city challenge," in *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCAL-COM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 2017, pp. 1–6.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[7] "How ai in edge computing drives 5g and the iot," Feb 2020. [Online]. Available: https://semiengineering.com/how-ai-in-edge-computing-drives-5g-and-the-iot/

[8] R. Koster, A. P. Black, J. Huang, J. Walpole, and C. Pu, "Infopipes for composing distributed information flows," in *Proceedings of the 2001 international workshop on Multimedia middleware*, 2001, pp. 44–47.

[9] [Online]. Available: https://gstreamer.freedesktop.org/

[10] M. Jones and P. Viola, "Fast multi-view face detection," *Mitsubishi Electric Research Lab TR-20003-96*, vol. 3, no. 14, p. 2, 2003.

[11] S. D. Burks and J. M. Doe, "Gstreamer as a framework for image processing applications in image fusion," in *Multisensor, Multisource Information Fusion: Architectures, Algorithms, and Applications 2011*, vol. 8064. International Society for Optics and Photonics, 2011, p. 80640M.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] Y. LeCun, Y. Bengio *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[17] X. Kang, B. Song, and F. Sun, "A deep similarity metric method based on incomplete data for traffic anomaly detection in iot," *Applied Sciences*, vol. 9, no. 1, p. 135, 2019.

[18] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[19] "Nvidia deepstream sdk," Sep 2020. [Online]. Available: https://developer.nvidia.com/deepstream-sdk

[20] P. Liu, B. Qi, and S. Banerjee, "Edgeeye: An edge service framework for real-time intelligent video analytics," in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking*, 2018, pp. 1–6.

[21] I. Prodaiko, "Person re-identification in a top-view multi-camera environment," 2020.

[22] M. Dubská, A. Herout, and J. Sochor, "Automatic camera calibration for traffic understanding." in *BMVC*, vol. 4, no. 6, 2014, p. 8.

8. GLOSSÁRIO

Stream - fluxo de dados.

Upstream - termo usado para descrever a direção da pipeline. Informações upstream significam fluxo de dados que estão sendo consumidas pelo elemento.

Downstream - termo usado para descrever a direção da pipeline. Informações downstream significam fluxo de dados será transmitido por um elemento.

Inferência - busca do algoritmo por objetos a partir de um modelo que foi treinado.

SDK - Software Development Kit.

GPU - Unidade de Processamento Gráfico

Message Broker - Módulo intermediário de mensagens

Bounding box - retângulo desenhado em volta de um objeto ou pessoa encontrado por um detector de objetos.