UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Departamento de Sistemas de Computação

Estudo de abordagens de DL-CNN (Deep Learning Convolutional Neural Nets) visando a detecção de veículos em imagens de vídeo, a fim de aumentar a segurança e evitar acidentes em vias públicas

Nilo Conrado Messias Alves Cangerana

São Carlos - SP

Estudos de abordagens de DL-CNN (Deep Learning Convolutional Neural Nets) visando a detecção de veículos em imagens de vídeo, a fim de aumentar a segurança e evitar acidentes em vias públicas

Nilo Conrado Messias Alves Cangerana

Orientador: Fernando Santos Osório

Monografia referente ao projeto de conclusão de curso dentro do escopo da disciplina SSC0670 – Projeto de Formatura I do Departamento de Sistemas de Computação do Instituto de Ciências Matemáticas e de Computação – ICMC-USP para obtenção do título de Engenheiro de Computação.

Área de Concentração: Inteligência Computacional

USP – São Carlos
Junho, 2021

Resumo

Durante as últimas décadas, o número de veículos automotivos no Brasil tem crescido gradativamente. Com o aumento do número de veículos que circulam pelas vias, o índice de acidentes de trânsito também aumenta. Devido ao avanço do poder computacional nos últimos anos, a área de *Deep Learning* tem ganhado destaque pela sua capacidade de resolver problemas de reconhecimento de padrões e localizar objetos. Afim de se diminuir a quantidade de acidentes de trânsitos, pode-se utilizar abordagens de *Deep Learning*. Ao utilizar modelos de *Deep Learning*, é possível desenvolver um sistema capaz de reconhecer padrões de veículos, identificando e localizando os mesmos em imagens. O projeto consiste no desenvolvimento de um sistema que utiliza Redes Neurais Convolucionais para localização de veículos em vídeos de câmeras que monitoram cruzamentos, afim de se sinalizar motoristas de que outros veículos percorrem as vias, aumentando a segurança no trânsito. A arquitetura de Rede Neural Convolucional utilizada para o desenvolvimento é a U-Net, que é uma rede que localiza objetos através da segmentação semântica dos mesmos. Neste trabalho, é realizada a modelagem do sistema proposto, que consiste no processamento de um vídeo de entrada, na qual a Rede Neural Convolucional realiza a segmentação semântica dos possíveis veículos presentes no vídeo, obtendo suas localizações. Para o desenvolvimento do projeto, é necessário realizar o treinamento do modelo. É escolhido um conjunto de dados para compor os conjuntos de treino, validação e de teste. O desempenho do modelo é medido com a métrica *Intersect over Union*, que possibilita identificar se o modelo apresenta boa precisão. Além do modelo da U-Net, são implementados outros modelos de Redes Neurais Convolucionais para comparar o resultado em diferentes arquiteturas e determinar se a rede escolhida apresenta bons resultados para essa funcionalidade. Os resultados do sistema são apresentados no final do trabalho, indicando se o sistema apresenta potencial para aumentar a segurança no trânsito.

Palavras-chave: Visão Computacional; Redes Neurais; *Deep Learning*; Segmentação Semântica; Processamento de Imagens.

Índice

Lista de Abreviaturas/Siglas

CE – *Cross-Entropy*

CNN – *Convolutional Neural Network*

DL – *Deep Learning*

FCN – *Fully Convolutional Network*

FN – *False Negative*

FP – *False Positive*

fps – *Frames* por segundo

GT – *Ground Truth*

IA – Inteligência Artificial

IoU – *Intersection over Union*

ReLU – *Rectified Linear Unit*

TP – *True Positive*

Lista de Tabelas

Lista de Figuras

CAPÍTULO 1: INTRODUÇÃO

1.1. Contextualização e Motivação

Durante as últimas décadas, o número de veículos automotivos no Brasil tem crescido gradativamente. De acordo com os dados do Instituto Brasileiro de Geografia e Estatística (2006, 2020), a frota de veículos no Brasil em 2006 era composta por aproximadamente 45 milhões unidades, enquanto que, em 2020, a frota de veículos no Brasil cresceu cerca de 140% em relação a 2006, atingindo uma marca de 108 milhões de unidades de veículos de diversos tipos. Essa popularização de veículos automotivos decorre principalmente do crescimento econômico do Brasil durante as últimas décadas, que possibilitou um aumento de renda para diferentes classes sociais e, consequentemente, facilitou a aquisição de veículos pela população. Outro fator importante que explica a expansão da frota de veículos no Brasil é a redução do custo de fabricação e de venda destes produtos, desde que o Fordismo revolucionou a indústria de produção de automóveis, fazendo com que veículos deixassem de ser produtos caros, através da produção em massa, se tornando mais acessíveis para a população.

No entanto, com o crescimento elevado de veículos automotivos, problemas envolvendo o tráfego de veículos nas vias também crescem. Dentre estes problemas, pode-se observar um aumento de congestionamento em grandes centros urbanos, um aumento da emissão de gases poluentes na atmosfera e também um aumento nos índices de acidentes de trânsitos. Em relação aos acidentes, o Brasil ainda possui altos índices. Segundo os dados divulgados pelo DATASUS (2020), foram registrados cerca de 30 a 40 mil mortes por acidentes de trânsito nos últimos 5 anos. Muitos desses acidentes ocorrem principalmente em vias que possuem cruzamentos com problemas de oclusão, prejudicando a visibilidade de motoristas que trafegam por elas.

Com o avanço do poder computacional durante os últimos anos, e a facilidade na aquisição de grandes quantidades de dados devido a digitalização da sociedade, um subtópico da Inteligência Artificial (IA), conhecido como *Deep Learning* (DL), tem ganhado destaque atualmente. *Deep Learning* (em português, Aprendizado Profundo) é uma subárea de *Machine*

Learning (em português, Aprendizado de Máquina) e envolve a criação de modelos com diversas camadas, nas quais as saídas de uma camada servem de entrada para uma camada posterior. Cada camada é capaz de realizar transformações lineares e não-lineares aos dados que recebem em sua entrada, produzindo um resultado na camada de saída do modelo. O principal diferencial de algoritmos baseados em *Machine Learning* são suas capacidades de aprendizado a partir de um conjunto de dados de treinamento. Esses algoritmos são capazes de extrair padrões e adquirir conhecimento dos dados de entrada durante seu treinamento e, posteriormente, são utilizados para inferir sobre dados novos afim de se realizar tarefas de classificação, predição ou reconhecimento de padrões (GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A., 2016). O grande destaque que proporcionou a difusão de aplicações baseadas em *Deep Learning* na atualidade é a forma como o modelo aprende a partir dos dados. Modelos de DL aprendem *features*, ou características, dos dados automaticamente através do seu processo de treinamento (ALOM, M. Z. et al., 2018). Por exemplo, um modelo capaz de identificar veículos em uma imagem aprende a identificar as *features* que compõe um veículo (rodas, capô, formato) para então, combiná-las e determinar se o objeto observado é classificado como veículo ou não.

Diversas aplicações se beneficiam do uso de *Deep Learning*. Uma delas consiste na identificação e localização de determinados objetos em imagens. Para aplicações como essa, Redes Neurais Convolucionais (em inglês, *Convolutional Neural Networks* – CNN) geram ótimos resultados (LECUN, Y.; BENGIO, Y.; HINTON, G., 2015). Redes Neurais Convolucionais são redes profundas com diversas camadas de convolução e *pooling* conectadas e são capazes de extrair *features* presentes nas imagens, combinar essas *features* semanticamente para formar objetos conhecidos e então, classificar determinado objeto de acordo com o objetivo final da aplicação. As arquiteturas mais comuns de CNN para classificação de imagens são compostas por várias camadas de convolução, *max pooling* e, ao final da rede, camadas totalmente conectadas. Algumas dessas arquiteturas famosas são: AlexNet (KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E., 2012), VGG (SIMONYAN, K.; ZISSERMAN, A., 2014) e GoogLeNet (SZEGEDY, C. et al., 2014). Para aplicações que envolvem a segmentação semântica e localização de objetos em imagens, as arquiteturas mais comuns de CNN consistem na substituição das camadas totalmente conectadas por camadas de *upsampling* e uma camada convolucional na saída das redes,

sendo possível gerar as imagens segmentadas ao final da rede. Essas arquiteturas também são chamadas de *fully convolutional networks* (FCN). Algumas dessas arquiteturas são: U-Net (RONNEBERGER, O.; FISCHER, P.; BROX, T., 2015) e FCN (LONG, J.; SHELHAMER, E.; DARRELL, T., 2014).

Devido à grande flexibilidade proporcionada pelo uso de CNN em diversas aplicações, é possível que modelos de CNN possam contribuir para redução dos índices de acidentes nas vias públicas. Diversos cruzamentos possuem câmeras de monitoramento que captam veículos que trafegam pelas ruas. Um modelo capaz de localizar veículos em imagens provenientes dessas câmeras poderia sinalizar motoristas que atravessam cruzamentos com problema de oclusão de que outros veículos também estão passando pelo mesmo cruzamento, aumentando a atenção de motoristas através da sinalização e, por consequência, reduzindo o risco de acidentes.

1.2. Objetivos

Este trabalho tem como objetivo o desenvolvimento de um sistema capaz de processar vídeos de câmeras de monitoramento de cruzamentos, utilizando uma Rede Neural Convolucional para realizar a segmentação semântica de veículos, sendo possível identificar e localizar possíveis veículos trafegando pelo cruzamento através dos vídeos e assim aumentar a segurança nas vias públicas.

Para essa finalidade, a arquitetura de CNN escolhida para o estudo e desenvolvimento da aplicação é a U-Net, que é uma rede que possibilita a localização e segmentação de objetos em imagens (RONNEBERGER, O.; FISCHER, P.; BROX, T., 2015). É realizado o treinamento da rede com um conjunto de imagens de treino, afim de se obter uma rede capaz de exercer essa funcionalidade.

Para o processamento de vídeos, um sistema é desenvolvido para extração de *frames* (em português, quadros) dos vídeos. Um pré-processamento é realizado em cada *frame* antes de serem passados para a CNN, que realiza a segmentação semântica e retorna o resultado processado pelo sistema.

Para análise dos resultados, são realizadas comparações da configuração da U-net implementada em relação a outras arquiteturas de CNN que realizam a função de segmentação semântica como a FCN-8s e a FCN-16s (LONG, J.; SHELHAMER, E.; DARRELL, T., 2014). Os resultados são avaliados sobre um conjunto de imagens de teste.

1.3. Organização do Trabalho

No Capítulo 2 é apresentada uma descrição teórica dos conceitos e terminologias envolvidos no desenvolvimento do trabalho e também trabalhos relacionados ao presente projeto. A seguir, no Capítulo 3, são descritas todas as atividades realizadas durante o desenvolvimento do projeto, também são apresentados os resultados obtidos e dificuldades encontradas durante o desenvolvimento. Finalmente, no Capítulo 4, é apresentada a conclusão do trabalho e contribuições do projeto realizado.

CAPÍTULO 2: REVISÃO BIBLIOGRÁFICA

2.1. Considerações Iniciais

Neste capítulo são apresentados diversos conceitos teóricos que são abordados ao longo do projeto e também são descritas algumas terminologias bastante utilizadas em DL. Alguns conceitos abordados são: Segmentação Semântica, Redes Neurais Convolucionais para segmentação semântica em imagens e algumas arquiteturas, *Datasets* (em português, conjunto de dados), Regularização, Métrica *Intersection over Union* (IoU) para avaliação de desempenho da segmentação semântica, dentre outros. Ao final do capítulo, são apresentados alguns trabalhos relacionados ao projeto.

2.2. Conceitos e Terminologias

2.2.1. Segmentação Semântica

Segmentação Semântica consiste no processo atribuir uma determinada classe a cada pixel presente na imagem. Um exemplo de Segmentação Semântica pode ser visto na figura 1.

Figura 1 - Exemplo de Segmentação Semântica

Fonte: https://medium.com/intro-to-artificial-intelligence/semantic-segmentation-udaitys-self-driving-car-engineer-nanodegree-c01eb6eaf9d

Como pode ser observado na figura 1, os pixels que compõe carros são classificados com uma cor. Os pixels que compõe pessoas são classificados com outra cor, indicando uma classe diferente de carros. O mesmo pode ser observado para diferentes objetos na imagem, indicando diferentes classes. Esse processo permite que a aplicação seja capaz de identificar objetos e localizar a posição desses objetos nas imagens.

2.2.2. *Datasets*

Datasets são conjuntos de dados utilizados para treinar e testar modelos de *Machine Learning*. Existem *Datasets* com dados em diferentes formatos e cada aplicação requer um formato específico dependendo da funcionalidade que se deseja atingir. Além disso, como o aprendizado de CNN é supervisionado, cada dado do *Dataset* deve ser composto de um par de elementos: o dado em questão e um *label* (em português, rótulo) que identifica a classe do dado para que o modelo seja capaz de calcular o erro entre a predição realizada e a classificação real do elemento.

Existem três tipos de *Datasets*:

- *Training Set*: Consiste do conjunto de dados que é utilizado para o treinamento do modelo. Os parâmetros do modelo são alterados somente pela avaliação no conjunto de treinamento

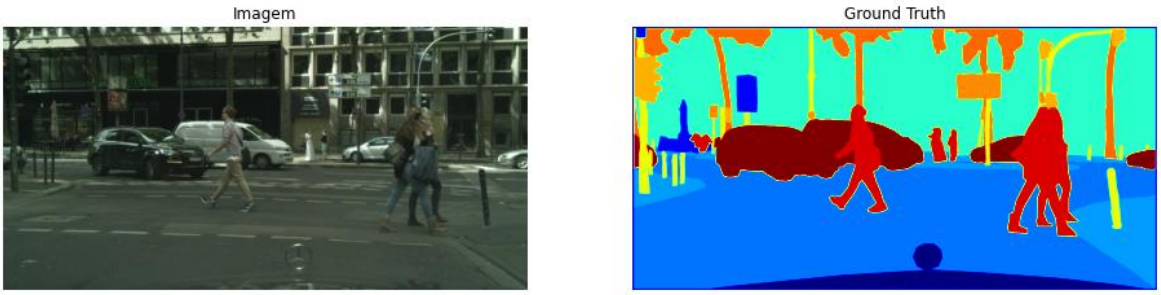
- *Validation Set*: Consiste de um conjunto de dados utilizado para avaliar o modelo durante o treinamento e pode ser utilizado para ajustar hiperparâmetros e evitar *Overfitting*. Os parâmetros do modelo não são alterados pelo conjunto de validação.

- *Test Set*: Consiste do conjunto utilizado para avaliar o modelo após ser treinado e ajustado. Também representa uma maneira de medir o erro de generalização do modelo, ou seja, o quanto o modelo aprendeu a prever dados que nunca viu (GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A., 2016).

Para a funcionalidade de segmentação semântica, os *Datasets* utilizados são compostos de um par de imagens: a imagem original e uma imagem, de mesma dimensão que

a original, onde cada pixel está indicado com uma classe ou *label*. Essa imagem de *labels* é comumente chamada de *ground truth* (GT). Um exemplo do par de imagens necessárias para treinar uma CNN para realizar a segmentação semântica está mostrada na Figura 2.

Figura 2 - Imagem e sua respectiva Ground Truth



Fonte: Adaptado de CORDTS, M. et al. (2016).

Conforme é possível observar na Figura 2, cada objeto diferente na imagem é representado por uma classe diferente no GT.

2.2.3. Regularização

Regularização é um conjunto de técnicas utilizadas para aumentar a capacidade de generalização dos modelos e controlar *Overfitting*, ou seja, quando o modelo possui baixo valor de erro na previsão de dados do conjunto de treino e alto valor de erro na previsão de dados dos conjuntos de validação e teste. No caso de *Overfitting*, o modelo é incapaz de generalizar para dados novos que nunca viu. Existem diversas técnicas de regularização para DL e algumas utilizadas neste trabalho são:

- *Early Stopping*: É utilizado juntamente com o conjunto de validação para monitorar o erro do mesmo durante o treinamento. Durante o treinamento, o erro no conjunto de treino e validação tende a diminuir enquanto o modelo aprende. A partir de certa iteração, o erro no conjunto de validação começa a subir enquanto que o erro no conjunto de treino continua descendo. Neste momento, o modelo está começando a apresentar *Overfitting*. O *Early Stopping* faz o treinamento parar, evitando o *Overfitting*.

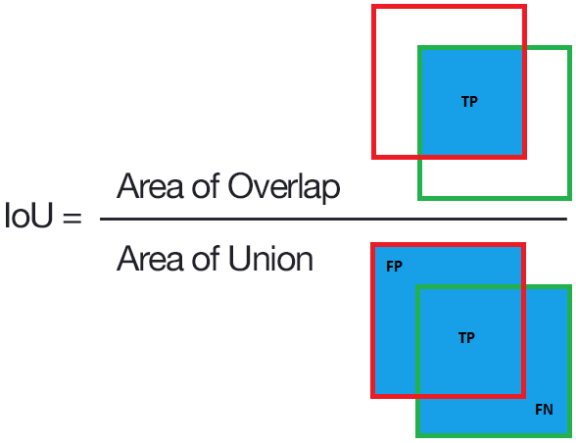
- *Data Augmentation*: Consiste na aplicação de transformações ao conjunto de treinamento, afim de se obter mais amostras para treinar os modelos. Quanto maior o número de amostras para treinamento, mais o modelo ganha capacidade de generalização, possibilitando reduzir o *Overfitting*. Para CNN que utilizam imagens como dado de treinamento, pode-se aplicar transformações como: translação, rotação, *flips*, adição de ruído, dentre outras.

- *Dropout*: Consiste de uma técnica na qual alguns neurônios selecionados aleatoriamente em uma camada são ignorados durante o treinamento, reduzindo a complexidade da rede e forçando que a rede treine com conexões diferentes entre cada camada. Com a redução da complexidade, ocorre também a redução de *Overfitting*. (GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A., 2016; SRIVASTAVA, N. et al., 2014).

2.2.4. Métrica *Intersection over Union* Média

A métrica de *Intersection over Union* avalia a performance do modelo na sua capacidade de detectar objetos. Essa métrica trabalha com a comparação entre o *ground truth* e a previsão gerada pelo modelo. A comparação é feita através das operações de união e interseção. A IoU é definida como a razão entre a área de sobreposição do objeto previsto com o *ground truth* (intersecção) e a área somada do objeto previsto e do *ground truth* (união). A visualização do cálculo dessa métrica pode ser vista na Figura 3.

Figura 3 - Visualização da IoU



Fonte: Adaptado de https://towardsdatascience.com/iou-a-better-detection-evaluation-metric-45a511185be1

Na Figura 3, o quadrado em verde representa o GT e o quadrado em vermelho representa a previsão do modelo. Quando o modelo prevê corretamente certa área de um objeto, temos a área de intersecção ou *True Positive* (TP). Quando o modelo prevê uma área que não pertence ao GT, temos a área de *False Positive* (FP). Quando o modelo não prevê uma área que pertence ao GT, temos a área de *False Negative* (FN). A área de união corresponde a soma de TP, FP e FN. Matematicamente, a IoU é calculada como:

$$IoU = \frac{TP}{FP+TP+FN} \qquad (1)$$

A IoU atinge valor máximo quando o objeto previsto é exatamente igual ao GT e atinge valor mínimo quando o objeto previsto é completamente diferente do GT.

A IoU média é calculada como a média entre a IoU de todas as classes do problema.

2.3. Redes Neurais Convolucionais e Arquiteturas

2.3.1. Definições

Redes Neurais Convolucionais para segmentação semântica são redes *feedforward*, ou seja, são redes nos quais as imagens são propagadas da entrada da rede até a saída. Nas camadas intermediárias, as imagens passam por camadas de convolução, *max pooling*, *upsampling* e transformações não-lineares por meio de funções de ativação. Na saída, o resultado previsto pela rede é comparado com o *ground truth* para se obter o erro entre a previsão e o resultado real. Com o erro calculado, a rede utiliza algoritmos de *backpropagation* para calcular o gradiente da função de erro em relação aos parâmetros treináveis e utiliza um otimizador para reduzir o erro através da atualização desses parâmetros (GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A., 2016). Os parâmetros são atualizados a cada passagem de um *Mini-Batch* (em português, Mini-Lote) pela rede, que é composto por partes do conjunto total de treino.

Neste trabalho, o otimizador utilizado é o Adam, que apresenta resultados eficientes na otimização de parâmetros de CNN (KINGMA, D. P.; BA, J. L., 2014). Também é utilizado a função de erro *Cross-Entropy* (CE), que permite o cálculo do erro para múltiplas classes. A *Cross-Entropy* calcula o erro para cada pixel da imagem de acordo com a seguinte fórmula:

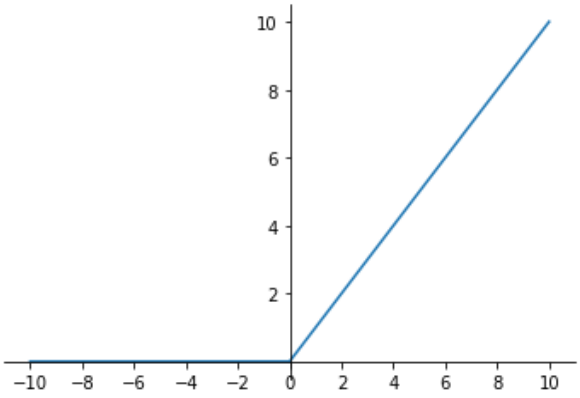
$$CE = -\sum_{i}^{N} y_i log(\hat{y}_i) \qquad (2)$$

Onde N representa o número de classes do problema, y_i representa o *ground truth* para a classe i e $\hat{y}_i$ representa a predição do modelo para determinada classe i. A média do erro de cada pixel na imagem é utilizada como o erro global do problema.

São aplicadas funções de ativação nas camadas de convolução afim de se garantir a não-linearidade dos resultados produzidos pelas CNN. Neste trabalho, a função de ativação utilizada nas camadas de convolução é a *Rectified Linear Unit* (ReLU), que reduz o custo computacional ao treinar modelos, aumentando a rapidez e eficiência do treinamento (KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E., 2012). A Figura 4 mostra o gráfico

da ReLU. Para valores negativos, a ReLU retorna zero e para valores positivos, retorna o próprio valor.

Figura 4 - Gráfico da ReLU



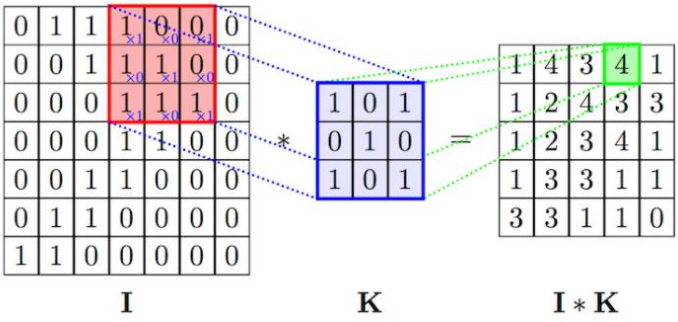
Fonte: Autor desta monografia.

Na camada de saída de CNN que resolvem problemas de múltiplas classes, é utilizada a função de ativação *Softmax*, que transforma a saída da rede em uma distribuição de probabilidades entre todas as classes do problema, possibilitando atribuir a classe com maior probabilidade ao elemento avaliado.

2.3.2. Camadas

Redes Neurais Convolucionais são redes profundas que possuem diversas camadas e a principal camada responsável pelo seu funcionamento é a Camada de Convolução. A Camada de Convolução consiste de uma camada que realiza a operação de convolução entre uma matriz de pixels e um filtro 2D, também chamado de *kernel*, cuja dimensão é um hiperparâmetro que pode ser definido. A visualização da operação de convolução está mostrada na Figura 5, na qual é aplicado um filtro de 3x3 a imagem.

Figura 5 - Visualização da operação de Convolução



Fonte: https://anhvnn.wordpress.com/2018/02/01/deep-learning-computer-vision-and-convolutional-neural-networks/

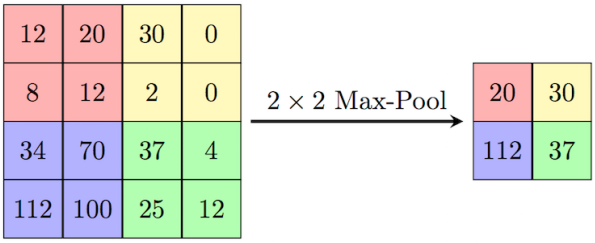
Cada valor do filtro é um parâmetro treinável da rede e esses filtros são ajustados durante o treinamento afim de serem capazes de detectar *features* que compõem a imagem como arestas, círculos e bordas. A quantidade de filtros em cada camada convolucional também é um hiperparâmetro que pode ser definido. A saída produzida por cada convolução entre a imagem e um filtro é uma matriz com dimensão reduzida, chamada de *feature map* (em português, mapa de ativação). A saída da camada de convolução produz uma quantidade de *feature maps* igual a quantidade de filtros que a camada possui. A quantidade de parâmetros treináveis em uma camada de convolução é dada pela equação (3).

$$quantidade\ de\ parâmetros = (h_f w_f C_{entrada} + 1) C_{saída} \qquad (3)$$

Onde h_f e w_f são as dimensões dos filtros da camada, $C_{entrada}$ é o número de canais da imagem na entrada da camada e $C_{saída}$ é a quantidade de canais da imagem na saída da camada.

A Camada de *Max Pooling* é responsável por reduzir o tamanho da imagem conforme ela é propagada pela rede, reduzindo o custo computacional necessário para processar grandes quantidades de dados de múltiplas dimensões. A operação de *Max Pooling* é realizada para todos mapas de ativação produzidos pela camada de convolução anterior e pode ser visualizada na Figura 6.

Figura 6 - Visualização da operação de *Max Pooling*

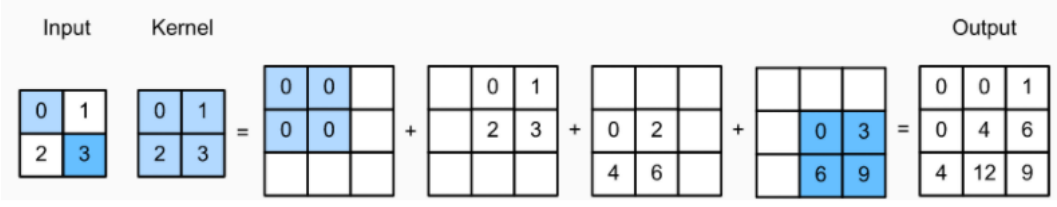


Fonte: https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

Esta camada realiza uma varredura nos mapas de ativação, buscando sempre o maior valor dentro de uma janela definida como hiperparâmetro (no caso da Figura 6, a janela possui tamanho 2x2). O *stride* (em português, passo) em que a janela varre a figura também é definido por um hiperparâmetro (no caso da Figura 6, o *stride* é 2). Essa operação permite que as informações mais importantes da imagem, que são as de maiores valores, sejam mantidas na figura de tamanho reduzido. As camadas de *Max Pooling* não possuem parâmetros treináveis.

Para realizar a reconstrução das imagens de tamanho reduzido pelas camadas de convolução e *max pooling*, são utilizadas Camadas de *Upsampling*. Ao utilizar a operação de convolução transposta como forma de *upsampling*, é possível aumentar o tamanho das imagens através de filtros com parâmetros treináveis. A convolução transposta também possui hiperparâmetros como tamanho do filtro, *stride* e quantidade de filtros. A visualização da convolução transposta está mostrada na Figura 7.

Figura 7 - Visualização da operação de Convolução Transposta



Fonte: https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba

A Figura 7 mostra a operação de convolução transposta com *kernel* 2x2 e *stride* unitário. Cada elemento da entrada é multiplicado por todos valores do filtro. O resultado

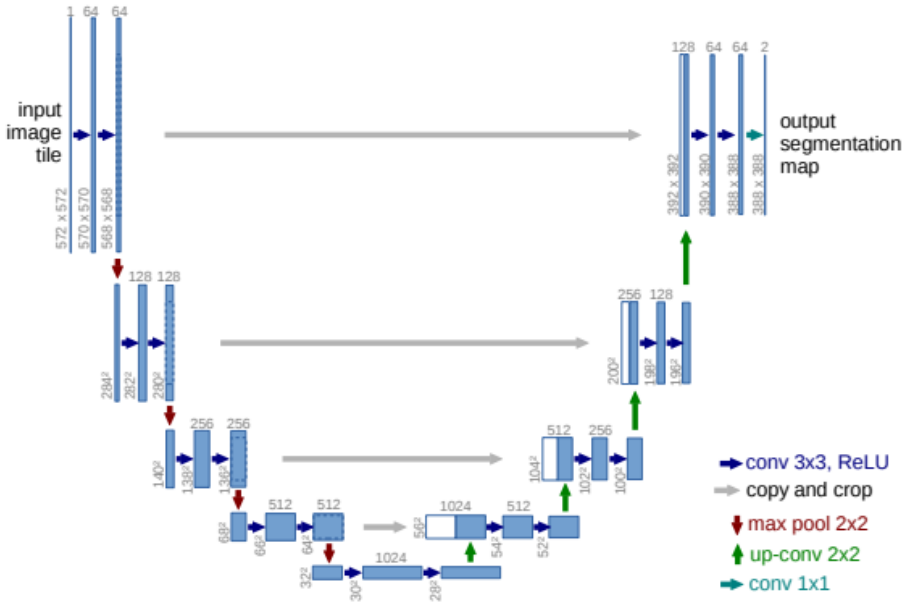
final corresponde a soma dos valores produzidos em cada posição da matriz de saída. A quantidade de parâmetros treináveis em camadas de *upsampling* que utilizam a convolução transposta é dada pela equação (3).

2.3.3. Arquiteturas

As arquiteturas mais comuns para CNN que realizam segmentação semântica consistem de camadas de convolução, seguidas por uma camada de *max pooling*, seguida por mais camadas de convolução, seguida por mais uma camada de *max pooling* e assim por diante. Esse bloco de camadas é utilizado para extrair *features* e informações semânticas da imagem. Em camadas iniciais, são obtidas informações de arestas e bordas que compõem os objetos, bem como suas localizações nas imagens. Em camadas mais profundas, são obtidas informações de objetos completos através da combinação de *features*, porém com baixa resolução. Esse bloco é chamado de *Encoder*. Após isso, são utilizadas camadas de *upsampling* para produção da segmentação semântica da imagem com mesmo tamanho da entrada. Esse processo consiste em aumentar a resolução e é chamado de *Decoder* (XING, Y.; ZHONG, L.; ZHONG, X., 2020). As arquiteturas discutidas a seguir possuem a estrutura *Encoder-Decoder*.

A U-Net, proposta por Ronneberger et al. (2015), é uma arquitetura proposta para segmentação semântica de imagens biomédicas e pode ser vista na Figura 8.

Figura 8 - Arquitetura da U-Net

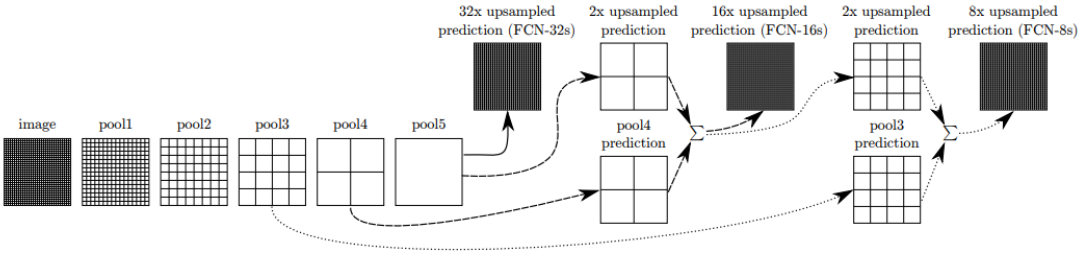


Fonte: RONNEBERGER, O.; FISCHER, P.; BROX, T., 2015.

A U-Net apresenta dois caminhos simétricos nos quais as imagens são propagadas, o *Contracting Path* à esquerda e o *Expansive Path* à direita, que correspondem à estrutura *Encoder-Decoder*. Além disso, são realizadas diversas operações de concatenação de *feature maps* do *Contracting Path* com o *Expansive Path*, afim de se obter maior precisão na localização de objetos devido à alta resolução proporcionada pelas camadas iniciais. A saída da U-Net consiste de uma camada de convolução, que produz uma quantidade de *feature maps* igual ao número de classes do problema, onde cada *feature map* apresenta a segmentação semântica de sua respectiva classe.

As arquiteturas FCN-32s, FCN-16s e FCN-8s, propostas por Long et al. (2014), podem ser vistas na Figura 9.

Figura 9 - Arquiteturas FCN-32s, FCN-16s e FCN-8s



Fonte: LONG, J.; SHELHAMER, E.; DARRELL, T., 2014.

Essas arquiteturas se diferem pela operação de *upsampling* final com diferentes *strides* (32, 16 ou 8) e também pelas operações de soma de informação de camadas anteriores, afim de se obter localização de objetos, presentes em camadas iniciais, com maior precisão. As operações de soma combinam previsões feitas sobre camadas de *pooling* intermediárias e a camada de *pooling* final.

2.4. Trabalhos Relacionados

Para realização deste trabalho, são utilizadas as arquiteturas e conceitos propostas por Long et al. (2014) e Ronneberger et al. (2015) para implementações de sistemas capazes de reconhecer e localizar veículos.

Além disso, outros trabalhos envolvendo a segmentação semântica foram propostos como a SegNet, proposta por Badrinarayanan et al. (2015) que é uma arquitetura de CNN do tipo *Encoder-Decoder*.

A segmentação semântica também é bastante utilizada na visão computacional de veículos autônomos. Os conceitos de detecção de objetos são importantes para o desenvolvimento de veículos autônomos seguros. A predição feita para cada pixel na imagem garante que o veículo autônomo consiga identificar diversos obstáculos e assim, tomar decisões baseadas nas informações obtidas, aumentando a segurança. No entanto, o custo computacional para realizar a segmentação semântica em tempo real é alto e muitos sistemas embarcados presentes em veículos autônomos não possuem essa capacidade. Treml et al.

(2016) propõe técnicas para aumentar a velocidade da segmentação semântica, para que sistemas computacionais possam reagir rapidamente, através de uma arquitetura do tipo *Encoder-Decoder* e assim, aumentar a segurança no trânsito.

2.5. Considerações Finais

Neste capítulo foram apresentados diversos conceitos teóricos relacionados ao projeto e também foram descritas algumas terminologias utilizadas em DL. Também foram apresentados algumas das principais arquiteturas de CNN que realizam segmentação semântica e suas principais camadas. Posteriormente, foram apresentados trabalhos relacionados ao desenvolvimento do projeto. O capítulo seguinte consiste na descrição do desenvolvimento do trabalho proposto e na discussão dos resultados obtidos.

CAPÍTULO 3: DESENVOLVIMENTO DO TRABALHO

3.1. Considerações Iniciais

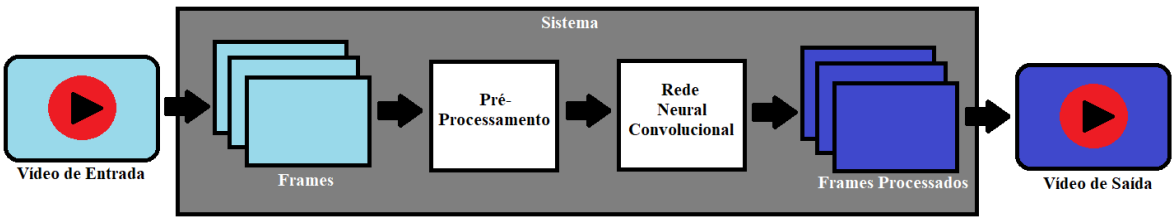
Neste capítulo são apresentados todos os passos envolvidos no desenvolvimento do projeto. É apresentada a modelagem do sistema que recebe um vídeo como entrada e realiza a segmentação semântica para localizar veículos nas imagens e as implementações necessárias para seu desenvolvimento. Também é mostrado o *dataset* escolhido para treino da CNN e o benefício de desempenho em relação ao *data augmentation* aplicado. Posteriormente, é realizada a implementação e comparação da U-Net com outras configurações de CNN (FCN-8s e a FCN-16s) para avaliar o desempenho e os resultados obtidos. Por fim, são discutidas as dificuldades e limitações encontradas durante o desenvolvimento do projeto.

3.2. Descrição do Projeto

3.2.1. Modelagem do Sistema

O trabalho consiste no desenvolvimento de um sistema capaz de receber vídeos provenientes de câmeras de monitoramento em cruzamentos, afim de se localizar veículos que trafegam pelas vias através da segmentação semântica. Para isso, foi proposto o sistema apresentado na Figura 10.

Figura 10 - Modelagem do Sistema



Fonte: Autor desta monografia.

A primeira parte do sistema consiste da entrada do vídeo ao sistema. Dentro do sistema, o vídeo é quebrado em *frames* e esses *frames* são armazenados em uma estrutura de dados do tipo lista, na ordem em que aparecem no vídeo. Também são armazenadas informações como tempo do vídeo, *frames* por segundo (fps) do vídeo e número de *frames* presentes no vídeo.

A segunda parte do sistema consiste de um pré-processamento realizado nos *frames*. Esse pré-processamento consiste em redimensionar a largura e altura dos *frames* para o tamanho que a CNN suporta. O pré-processamento também transforma o sistema de cores dos *frames* de RGB para *Grayscale* (em português, níveis de cinza), para que o processamento realizado pela CNN possua menos custo computacional. Por fim, é aplicada uma normalização aos *frames* para reescalar os valores dos pixels entre zero e um e assim, poderem ser processadas pela CNN.

A terceira parte do sistema consiste da Rede Neural Convolucional, que recebe cada *frame* pré-processado para poder realizar a predição sobre os pixels. Cada *frame* que passa pela rede tem o valor de seus pixels preditos. A saída da CNN retorna, para cada *frame*, um vetor de três dimensões. As duas primeiras representam a largura e altura do *frame* e a terceira representa a quantidade de classes do problema, ou seja, cada classe produz uma matriz de dimensões iguais do *frame* contendo a segmentação semântica correspondente da classe em questão.

A quarta parte do sistema corresponde a geração de uma lista com os *frames* processados pela rede. As classes separadas em cada matriz são juntadas em uma única matriz de mesmo tamanho. É atribuída uma cor diferente a cada classe diferente para facilitar a visualização e distinção entre cada classe predita. Ao final do processo, é obtido uma lista de todos *frames* processados, onde a cor de cada pixel indica a classe que foi atribuída aquele pixel. Com os *frames* processados, um vídeo é reconstruído com mesmo tempo do vídeo original, mesmo fps e mesma quantidade de *frames*. O vídeo com a localização dos veículos preditas pela rede é retornado como saída do sistema. Essa ferramenta é um protótipo, que permite validar a aplicação do sistema de detecção veículos, e assim, futuramente, poderia sinalizar e evitar uma possível colisão.

3.2.2. Linguagem de Programação

A linguagem de programação escolhida para o desenvolvimento do projeto é a linguagem Python. Python possui diversas bibliotecas para manipulação e processamento de imagens, facilitando a manipulação das estruturas de dados que compõem esse tipo de dado.

Além disso, Python possui uma *Application Programming Interface* (API) para implementação e desenvolvimento de redes neurais, chamado Keras. Keras é uma API de alto nível que roda em cima de TensorFlow, que é uma plataforma de *Machine Learning*. A utilização do Keras em conjunto com o TensorFlow permite a criação de modelos altamente configuráveis e de fácil implementação. Além disso, o treinamento dos modelos pode ser acelerado por unidades de processamento gráfico (GPU) para aumentar a velocidade do treinamento (CHOLLET, F. et al., 2015; ABADI, M. et al., 2015).

3.2.3. Plataforma de Desenvolvimento e Recursos Computacionais

O sistema foi implementado e testado no sistema operacional de 64 bits Windows 10 Pro Versão 10.0.19041. Os recursos de hardware são:

- Processador: Intel® Core™ i7-6820HK CPU @ 2.70GHz
- Memória RAM: 16.0 GB
- GPU: NVIDIA GeForce GTX 1070 – 8.0 GB

Para desenvolvimento do software, foi utilizado o ambiente de desenvolvimento integrado (IDE) Spyder, que é uma plataforma *open source* para desenvolvimento de códigos em Python. As versões utilizadas estão listadas a seguir:

- Versão Spyder IDE: 4.2.5
- Versão Python: 3.7.9
- Versão Keras: 2.4.3

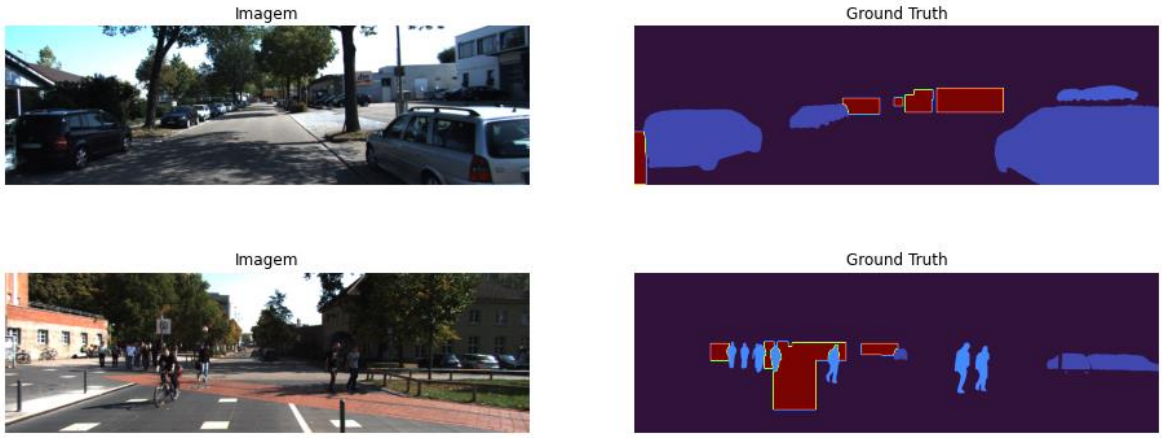
3.3. Descrição das Atividades Realizadas

3.3.1. Escolha e Processamento do Dataset

Para realizar o treinamento de CNN é necessário um *dataset* de imagens de veículos e suas respectivas GT. Foram analisados dois *datasets* para o desenvolvimento do trabalho e apenas um foi escolhido.

O primeiro consiste do *dataset* de Voigtlaender et al. (2019) chamado de KITTI *Multi-Object and Segmentation* (KITTI MOTS). Esse *dataset* consiste de 8008 imagens RGB para treino, onde cada imagem possui tamanho aproximado de 1242x375 e um respectivo GT de mesma dimensão. As imagens de GT possuem rótulos que identificam *background*, carros, pessoas e objetos diferentes (caminhões, bicicletas, dentre outros). Um exemplo das imagens do KITTI MOTS *Dataset* é mostrado na Figura 11.

Figura 11 - Imagens do KITTI MOTS *Dataset*

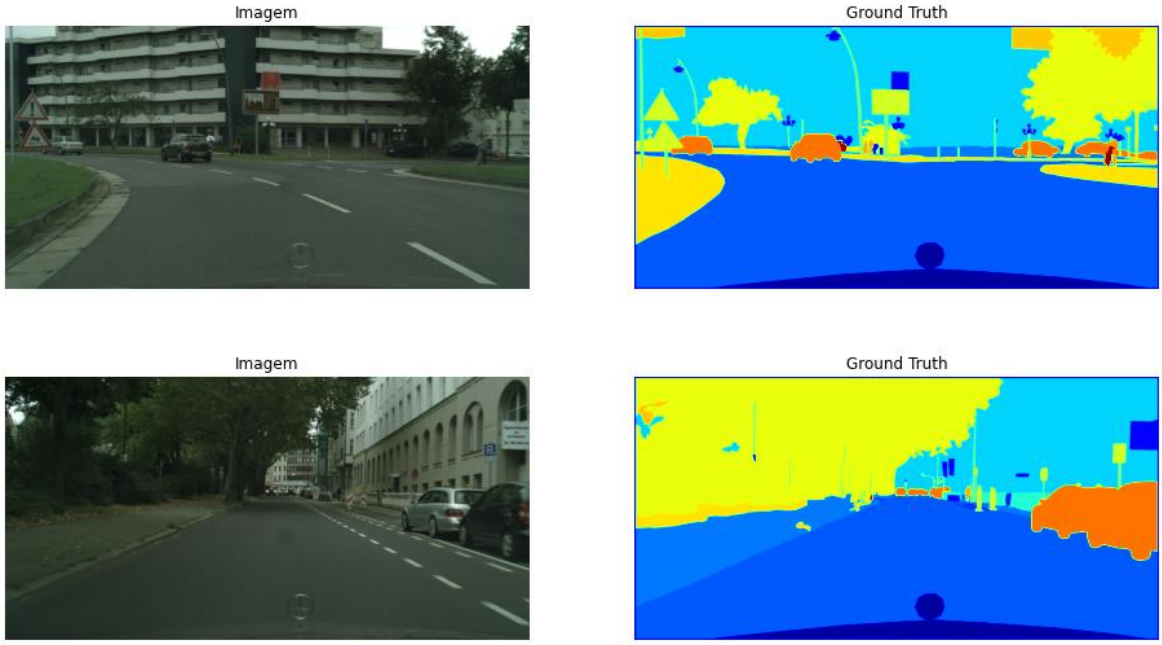


Fonte: Adaptado de Voigtlaender et al. (2019).

O segundo consiste do *dataset* de Cordts et al. (2016) chamado de *Cityscapes Dataset*. Esse *dataset* consiste de 2975 imagens RGB para treino, onde cada imagem possui tamanho 2048x1024 e um respectivo GT de mesma dimensão. Além disso, o *dataset* possui um conjunto de validação com 500 imagens diferentes das imagens de treino e com mesmo tamanho e rótulos no GT. As imagens de GT possuem rótulos que indicam 30 classes

diferentes (carros, caminhões, motos, pessoas, ruas, dentre outras). Um exemplo das imagens do *Cityscapes Dataset* é mostrado na Figura 12.

Figura 12 - Exemplo do *Cityscapes Dataset*



Fonte: Adaptado de Cordts et al. (2016).

Para este trabalho, foi escolhido a utilização do *Cityscapes Dataset* pois ele apresenta uma maior variabilidade nos tipos de veículos rotulados, devido a quantidade maior de classes rotuladas. No entanto, ele apresenta uma desvantagem em relação ao KITTI MOTS devido a menor quantidade de imagens.

Com o *dataset* escolhido, foi realizado um processamento em todas as imagens afim de se padronizar o *dataset*. O processamento consiste em remover as bordas das imagens, nas quais aparecem a frente do carro que fotografou as ruas, para que isso não influencie no treinamento da CNN. Também foi realizada uma filtragem nas 30 classes para manter apenas 11 classes que representam todos os tipos de veículos e pedestres. Isso foi feito para remover classes que não fariam sentido para o desenvolvimento do sistema (prédios, placas, árvores, dentre outras) e para reduzir o custo computacional ao processar múltiplas classes. As imagens também tiveram a dimensão reduzida para reduzir o custo computacional. A lista com todas as 11 classes restantes e seus respectivos *labels* está mostrada na Tabela 1.

Tabela 1 – Lista de classes e seus valores de *labels*

Valor do pixel no GT (*label*)	Tipo
0	*Background*
1	Pedestres
2	Ciclistas/Motoqueiros
3	Carros
4	Caminhões
5	Ônibus
6	Trailers
7	Carretas
8	Trem
9	Motos
10	Bicicletas

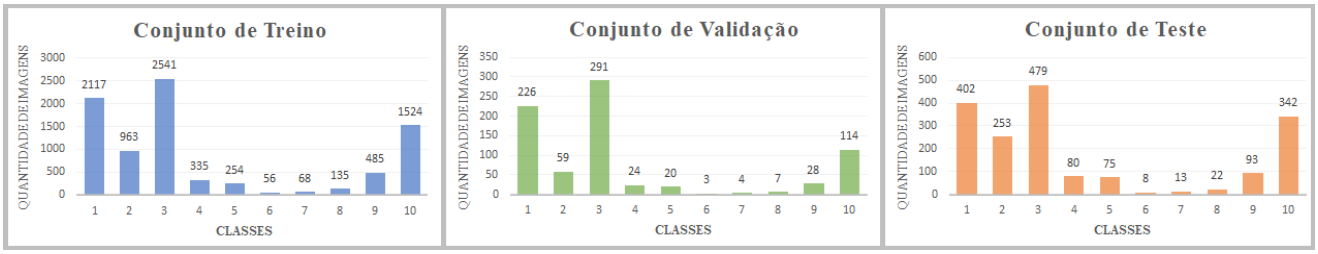
A Figura 13 mostra uma imagem e seu GT após o processamento realizado.

Figura 13 - Imagem e GT após o processamento

Fonte: Adaptado de Cordts et al. (2016).

Por fim, foram definidos os *datasets* de treino, validação e teste. As 500 imagens de validação do *Cityscapes Dataset* foram separadas para serem utilizadas como o conjunto de teste. Foram separadas 300 imagens do conjunto de treino para serem utilizadas como o conjunto de validação, restando 2675 imagens para o conjunto de treino. A distribuição da quantidade de imagens por classes nos diferentes conjuntos pode ser vista na Figura 14.

Figura 14 - Distribuição da quantidade de imagens por classes nos *datasets*



Fonte: Autor desta monografia.

3.3.2. *Data Augmentation*

Afim de se aumentar a quantidade de imagens de treino e reduzir o *overfitting*, foram aplicados três métodos de *data augmentation*: *flip* horizontal, translação e adição de ruído. As transformações realizadas foram aplicadas somente no conjunto de treino e principalmente em imagens que contém classes com baixa ocorrência, para aumentar a frequência dessas classes.

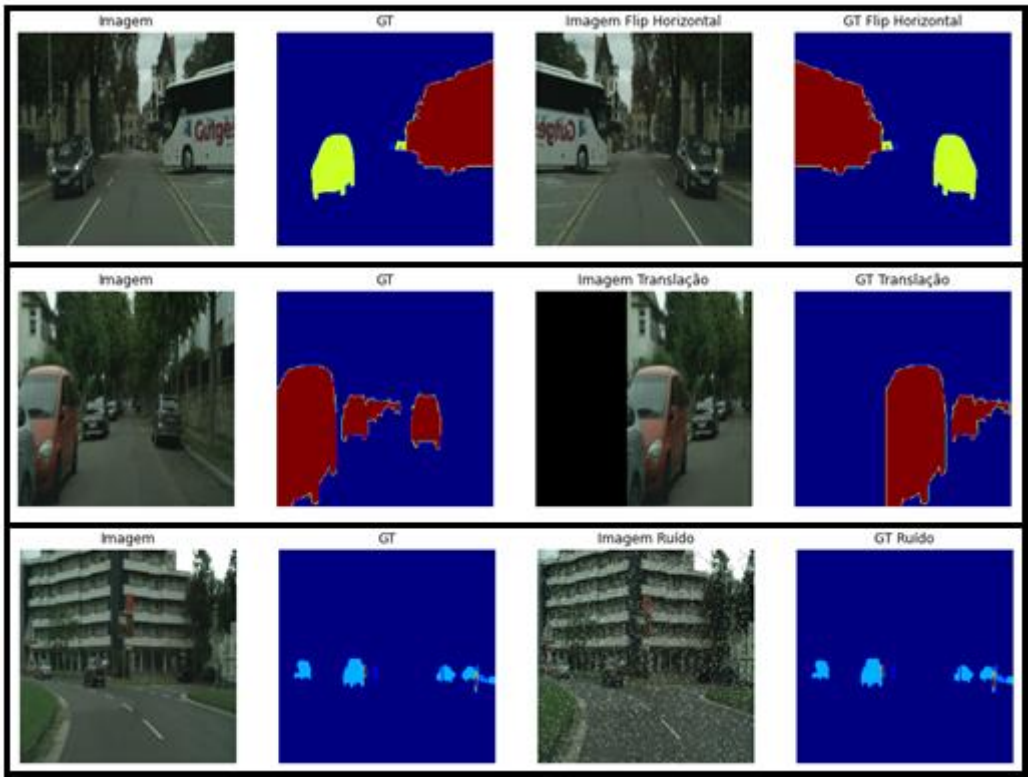
O *flip* horizontal consiste no espelhamento horizontal da imagem e sua GT. As imagens ficam invertidas em relação ao eixo vertical.

A translação consiste em um deslocamento aplicado a todos os pixels da imagem. Foi implementado a translação para direita, esquerda e para cima. Um valor aleatório entre 40 e 128 pixels é utilizado para definir o tamanho do deslocamento aplicado a imagem e a direção é escolhida aleatoriamente entre cima, direita e esquerda. O limite inferior 40 foi definido para não gerar imagens muito próximas à original e o limite superior 128 foi definido para não gerar imagens com deslocamento muito grande que retiram objetos da cena. A translação é aplicada tanto na imagem quanto no GT.

O ruído aplicado consiste na alteração aleatória de valores dos pixels da imagem para 255 (branco) ou 0 (preto), afim de se produzir imagens com deformações. Cada pixel da imagem possui uma probabilidade de 5% de ser alterado para 255 ou 0.

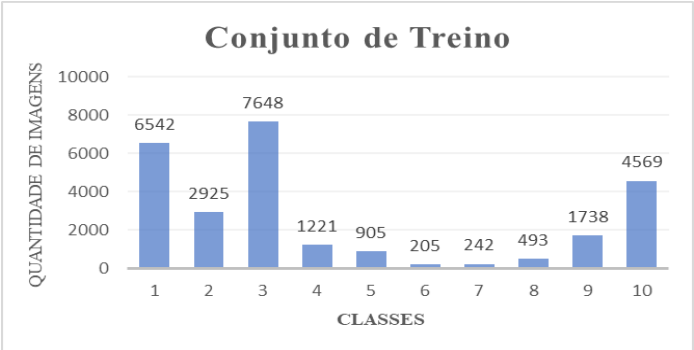
Na Figura 15 estão mostrados exemplos do *flip* horizontal, translação para direita e ruído aplicado nas imagens. Após as transformações, o conjunto de treino ficou composto por 8097 imagens e GT. A distribuição da quantidade de imagens de treino por classe após a aplicação do *data augmentation* está mostrada na Figura 16.

Figura 15 - Exemplos da aplicação do *data augmentation*



Fonte: Adaptado de Cordts et al. (2016).

Figura 16 – Quantidade de imagens por classe após o *data augmentation*



Fonte: Autor desta monografia.

3.3.3. Implementação do *Custom Data Generator*

Para realizar o treinamento da CNN com conjuntos de dados muito grandes que não cabem inteiramente na memória RAM, foi necessário a implementação de um *Data Generator*, que gera *batchs* de imagens e suas GT durante o treinamento, sem que seja necessário carregar todos os dados ao mesmo tempo na memória RAM. Para isso, foi utilizado a classe *Sequence* presente no pacote de utilidades do Keras (*keras.utils*).

Foi implementado uma classe chamada de *CustomDataGenerator*, que é herdada da classe *Sequence* do Keras. O tamanho do lote que será gerado durante o treinamento é passado como parâmetro (chamado de *batch_size*) ao se instanciar o objeto da classe. Com isso, cada lote gerado possui uma quantidade de *batch_size* imagens e GT, que são lidas do disco durante o treinamento da rede.

Cada imagem é lida em *grayscale*, para reduzir o custo computacional do treinamento. Também é aplicado normalização às imagens de treinamento através do método *normalize* do pacote de utilidades do Keras. Cada GT também é lido em *grayscale*, onde cada pixel possui um valor de *label* entre zero e dez. Para poder ser utilizado na rede, é necessário transformar cada GT no formato *one-hot-encoding*. Isso é feito utilizando o método *to_categorical* do pacote de utilidade do Keras. Com isso, cada GT possui o formato HxWx11, onde H e W são as dimensões da imagem e o 11 representa as onze classes do problema, no formato *one-hot-enconding*. Todos esses passos são aplicados a cada lote gerado. O objeto desta classe é passado como parâmetro para o método que realiza o treinamento no Keras.

3.3.4. Implementação dos Modelos de CNN

Para realizar as implementações dos modelos, foi necessário definir um tamanho fixo das imagens de entrada que serão processadas pela rede. Para isso, foram analisados os tamanhos 128x128, 256x256 e 512x512. Como o *dataset* possui muitos objetos de diferentes classes, a redução da resolução para 128x128 poderia prejudicar a classificação de objetos que estão muito distantes nas imagens, pela pouca quantidade de pixels que os compõe. Para imagens com resolução de 256x256, os objetos ficam bem definidos e a visualização e distinção entre eles é facilitada em relação à 128x128. Para imagens de 512x512, foi

verificado que o tempo de treinamento aumenta muito em relação à 256x256 (na ordem de horas) e não apresenta ganho considerável de performance. Além disso, o tamanho de lote máximo suportado pela memória RAM é reduzido em relação a 256x256. Assim, foi escolhido utilizar o tamanho 256x256.

Os modelos foram implementados utilizando os pacotes *keras.models* e *keras.layers*. A Tabela 2 mostra a implementação da U-Net, indicando as camadas utilizadas e alguns de seus parâmetros, a entrada que cada camada recebe, o formato da saída de cada camada e a quantidade de parâmetros treináveis de cada camada. Foi utilizado o parâmetro que define a função de ativação *activation='relu'* para todas camadas de convolução, exceto para a saída, que utiliza *activation='softmax'*. Também foi utilizado o parâmetro *padding='same'* para manter o tamanho da imagem na entrada da convolução igual o tamanho na saída.

Tabela 2 – Implementação da U-Net

Camadas	Entrada	Formato da Saída	Número de Parâmetros Treináveis
1)Entrada	-	256x256x1	0
2)Conv2D – *filters*=32, *kernel_size*=(3,3)	1	256x256x32	320
3)Conv2D – *filters*=32, *kernel_size*=(3,3)	2	256x256x32	9248
4)Dropout – *rate*=0.1	3	256x256x32	0
5)MaxPooling2D – *pool_size*=(2,2)	4	128x128x32	0
6)Conv2D – *filters*=64, *kernel_size*=(3,3)	5	128x128x64	18496
7)Conv2D – *filters*=64, *kernel_size*=(3,3)	6	128x128x64	36928
8)Dropout – *rate*=0.1	7	128x128x64	0
9)MaxPooling2D – *pool_size*=(2,2)	8	64x64x64	0
10)Conv2D – *filters*=128, *kernel_size*=(3,3)	9	64x64x128	73856
11)Conv2D – *filters*=128, *kernel_size*=(3,3)	10	64x64x128	147584
12)Dropout – *rate*=0.2	11	64x64x128	0
13)MaxPooling2D – *pool_size*=(2,2)	12	32x32x128	0
14)Conv2D – *filters*=256, *kernel_size*=(3,3)	13	32x32x256	295168
15)Conv2D – *filters*=256, *kernel_size*=(3,3)	14	32x32x256	590080
16)Dropout – *rate*=0.3	15	32x32x256	0
17)MaxPooling2D – *pool_size*=(2,2)	16	16x16x256	0
18)Conv2D – *filters*=512, *kernel_size*=(3,3)	17	16x16x512	1180160
19)Conv2D – *filters*=512, *kernel_size*=(3,3)	18	16x16x512	2359808
20)Dropout – *rate*=0.5	19	16x16x512	0
21)Conv2DTranspose – *filters*=256, *kernel_size*=(2,2), *strides*=(2,2)	20	32x32x256	524544
22)concatenate	21, 16	32x32x512	0
23)Conv2D – *filters*=256, *kernel_size*=(3,3)	22	32x32x256	1179904

24)Conv2D – *filters*=256, *kernel_size*=(3,3)	23	32x32x256	590080
25)Dropout – *rate*=0.4	24	32x32x256	0
26)Conv2DTranspose – *filters*=128, *kernel_size*=(2,2), *strides*=(2,2)	25	64x64x128	131200
27)concatenate	26, 12	64x64x256	0
28)Conv2D – *filters*=128, *kernel_size*=(3,3)	27	64x64x128	295040
29)Conv2D – *filters*=128, *kernel_size*=(3,3)	28	64x64x128	147584
30)Dropout – *rate*=0.3	29	64x64x128	0
31)Conv2DTranspose – *filters*=64, *kernel_size*=(2,2), *strides*=(2,2)	30	128x128x64	32832
32)concatenate	31, 8	128x128x128	0
33)Conv2D – *filters*=64, *kernel_size*=(3,3)	32	128x128x64	73792
34)Conv2D – *filters*=64, *kernel_size*=(3,3)	33	128x128x64	36928
35)Dropout – *rate*=0.2	34	128x128x64	0
36)Conv2DTranspose – *filters*=32, *kernel_size*=(2,2), *strides*=(2,2)	35	256x256x32	8224
37)concatenate	36, 4	256x256x64	0
38)Conv2D – *filters*=32, *kernel_size*=(3,3)	37	256x256x32	18464
39)Conv2D – *filters*=32, *kernel_size*=(3,3)	38	256x256x32	9248
40)Dropout – *rate*=0.2	39	256x256x32	0
41)Conv2D – *filters*=11, *kernel_size*=(1,1)	40	256x256x11	363
42)Saída	40	256x256x11	0

A U-Net implementada apresenta 7759851 parâmetros treináveis. A Tabela 3 e a Tabela 4 mostram as implementações das redes FCN-16s e FCN-8s respectivamente. As mesmas configurações para função de ativação e *padding* da U-Net foram utilizadas.

Tabela 3 - Implementação da FCN-16s

Camadas	Entrada	Formato da Saída	Número de Parâmetros Treináveis
1)Entrada	-	256x256x1	0
2)Conv2D – *filters*=16, *kernel_size*=(3,3)	1	256x256x16	160
3)Conv2D – *filters*=16, *kernel_size*=(3,3)	2	256x256x16	2320
4)Dropout – *rate*=0.1	3	256x256x16	0
5)MaxPooling2D – *pool_size*=(2,2)	4	128x128x16	0
6)Conv2D – *filters*=32, *kernel_size*=(3,3)	5	128x128x32	4640
7)Conv2D – *filters*=32, *kernel_size*=(3,3)	6	128x128x32	9248
8)Dropout – *rate*=0.1	7	128x128x32	0
9)MaxPooling2D – *pool_size*=(2,2)	8	64x64x32	0
10)Conv2D – *filters*=64, *kernel_size*=(3,3)	9	64x64x64	18496
11)Conv2D – *filters*=64, *kernel_size*=(3,3)	10	64x64x64	36928

12)Conv2D – *filters*=64, *kernel_size*=(3,3)	11	64x64x64	36928
13)Dropout – *rate*=0.2	12	64x64x64	0
14)MaxPooling2D – *pool_size*=(2,2)	13	32x32x64	0
15)Conv2D – *filters*=128, *kernel_size*=(3,3)	14	32x32x128	73856
16)Conv2D – *filters*=128, *kernel_size*=(3,3)	15	32x32x128	147584
17)Conv2D – *filters*=128, *kernel_size*=(3,3)	16	32x32x128	147584
18)Dropout – *rate*=0.3	17	32x32x128	0
19)MaxPooling2D – *pool_size*=(2,2)	18	16x16x128	0
20)Conv2D – *filters*=128, *kernel_size*=(3,3)	19	16x16x128	147584
21)Conv2D – *filters*=128, *kernel_size*=(3,3)	20	16x16x128	147584
22)Conv2D – *filters*=128, *kernel_size*=(3,3)	21	16x16x128	147584
23)Dropout – *rate*=0.4	22	16x16x128	0
24)MaxPooling2D – *pool_size*=(2,2)	23	8x8x128	0
25)Conv2D – *filters*=1024, *kernel_size*=(7,7)	24	8x8x1024	6423552
26)Dropout – *rate*=0.5	25	8x8x1024	0
27)Conv2D – *filters*=1024, *kernel_size*=(1,1)	26	8x8x1024	1049600
28)Dropout – *rate*=0.5	27	8x8x1024	0
29)Conv2DTranspose – *filters*=11, *kernel_size*=(2,2), *strides*=(2,2)	28	16x16x11	45067
30) Conv2D – *filters*=11, *kernel_size*=(1,1)	19	16x16x11	1419
31)Add	30,29	16x16x11	0
32)Conv2DTranspose – *filters*=11, *kernel_size*=(16,16), *strides*=(16,16)	31	256x256x11	30987
33)Saída	31	256x256x11	0

Tabela 4 - Implementação da FCN-8s

Camadas	Entrada	Formato da Saída	Número de Parâmetros Treináveis
1)Entrada	-	256x256x1	0
2)Conv2D – *filters*=16, *kernel_size*=(3,3)	1	256x256x16	160
3)Conv2D – *filters*=16, *kernel_size*=(3,3)	2	256x256x16	2320
4)Dropout – *rate*=0.1	3	256x256x16	0
5)MaxPooling2D – *pool_size*=(2,2)	4	128x128x16	0
6)Conv2D – *filters*=32, *kernel_size*=(3,3)	5	128x128x32	4640
7)Conv2D – *filters*=32, *kernel_size*=(3,3)	6	128x128x32	9248
8)Dropout – *rate*=0.1	7	128x128x32	0
9)MaxPooling2D – *pool_size*=(2,2)	8	64x64x32	0
10)Conv2D – *filters*=64, *kernel_size*=(3,3)	9	64x64x64	18496
11)Conv2D – *filters*=64, *kernel_size*=(3,3)	10	64x64x64	36928
12)Conv2D – *filters*=64, *kernel_size*=(3,3)	11	64x64x64	36928
13)Dropout – *rate*=0.2	12	64x64x64	0
14)MaxPooling2D – *pool_size*=(2,2)	13	32x32x64	0
15)Conv2D – *filters*=128, *kernel_size*=(3,3)	14	32x32x128	73856

16)Conv2D – *filters*=128, *kernel_size*=(3,3)	15	32x32x128	147584
17)Conv2D – *filters*=128, *kernel_size*=(3,3)	16	32x32x128	147584
18)Dropout – *rate*=0.3	17	32x32x128	0
19)MaxPooling2D – *pool_size*=(2,2)	18	16x16x128	0
20)Conv2D – *filters*=128, *kernel_size*=(3,3)	19	16x16x128	147584
21)Conv2D – *filters*=128, *kernel_size*=(3,3)	20	16x16x128	147584
22)Conv2D – *filters*=128, *kernel_size*=(3,3)	21	16x16x128	147584
23)Dropout – *rate*=0.4	22	16x16x128	0
24)MaxPooling2D – *pool_size*=(2,2)	23	8x8x128	0
25)Conv2D – *filters*=1024, *kernel_size*=(7,7)	24	8x8x1024	6423552
26)Dropout – *rate*=0.5	25	8x8x1024	0
27)Conv2D – *filters*=1024, *kernel_size*=(1,1)	26	8x8x1024	1049600
28)Dropout – *rate*=0.5	27	8x8x1024	0
29)Conv2DTranspose – *filters*=11, *kernel_size*=(2,2), *strides*=(2,2)	28	16x16x11	45067
30) Conv2D – *filters*=11, *kernel_size*=(1,1)	19	16x16x11	1419
31)Add	30,29	16x16x11	0
32)Conv2DTranspose – *filters*=11, *kernel_size*=(2,2), *strides*=(2,2)	31	32x32x11	495
33) Conv2D – *filters*=11, *kernel_size*=(1,1)	14	32x32x11	715
34)Add	33,32	32x32x11	0
35)Conv2DTranspose – *filters*=11, *kernel_size*=(8,8), *strides*=(8,8)	34	256x256x11	7755
36)Saída	34	256x256x11	0

A FCN-16s apresenta 8471121 parâmetros treináveis e a FCN-8s apresenta 8449099 parâmetros treináveis.

3.4. Resultados Obtidos

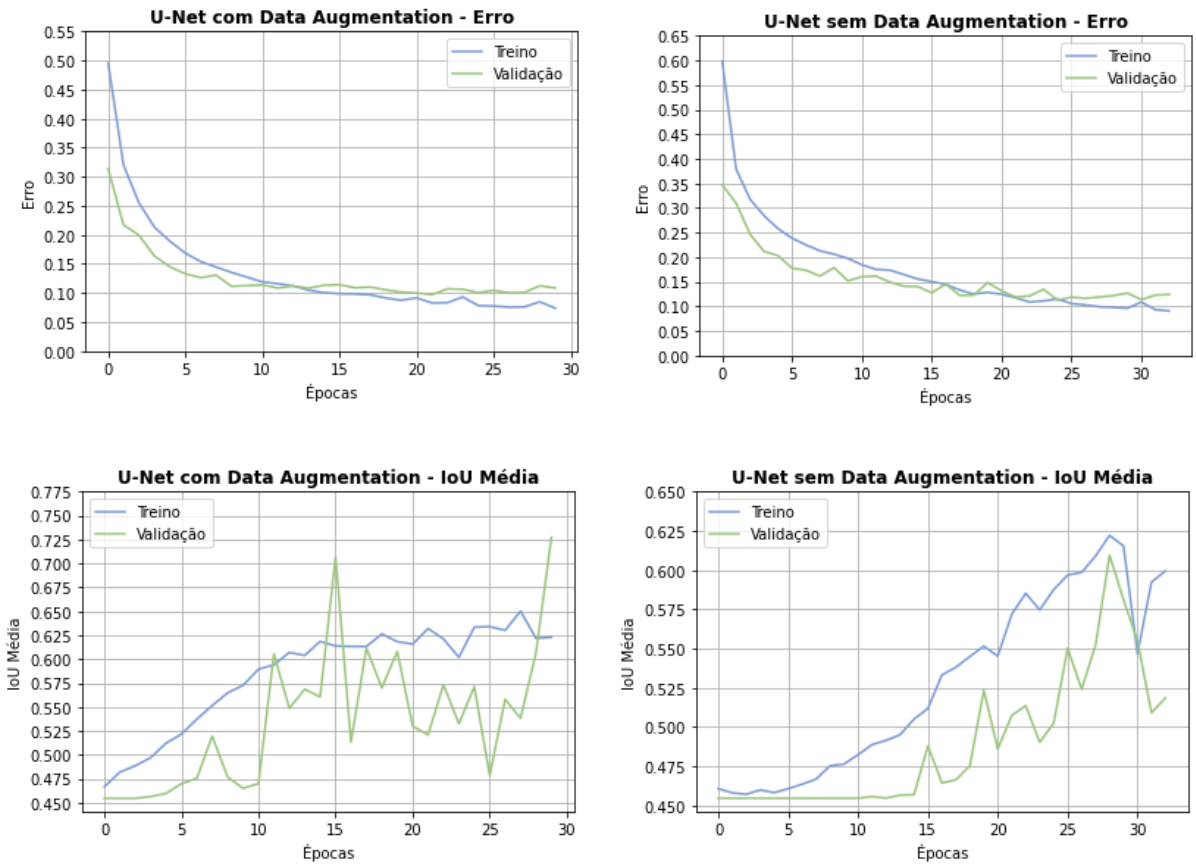
3.4.1. Treinamento dos Modelos e Resultados

Os modelos foram treinados com configurações iguais para que seu desempenho fosse analisado nas mesmas condições. O parâmetro *batch_size* do *Custom Data Generator* foi definido para 16. Os parâmetros dos modelos: *loss='categorical_crossentropy'* e *optimizer='adam'* foram definidos para utilizar a função de erro *Cross-Entropy* e o otimizador Adam. A métrica para avaliação de desempenho é a média IoU e ela foi definida através do parâmetro *metrics=[tf.keras.metrics.MeanIoU(num_classes=11)]*. Além dos

parâmetros, foi utilizado um *callback* de *Early Stopping* que termina o treinamento caso não ocorra melhora no erro de validação do modelo durante 8 épocas. O número total de épocas de treinamento foi definido para 50.

O modelo da U-Net foi avaliado em relação à aplicação de *data augmentation* ao conjunto de treinamento. Para isso, o modelo foi treinado com o conjunto de treino padrão e com o conjunto de treino com *data augmentation*. Os gráficos do erro e da IoU média por épocas estão mostrados na Figura 17. Além disso, a Tabela 5 apresenta os resultados do erro e da IoU média avaliados no conjunto de treino, validação e teste na melhor configuração obtida com o treino.

Figura 17 - Gráficos do Erro e da IoU Média para U-Net



Fonte: Autor desta monografia.

Tabela 5 - Erro e IoU Média nos diferentes conjuntos para a U-Net

	Erro			**IoU Média**		
	Treino	Validação	Teste	Treino	Validação	Teste
U-Net sem *data augmentation*	0.0906	0.1243	0.1570	0.5994	0.5185	0.5241
U-Net com *data augmentation*	0.0739	0.1085	0.1461	0.6228	0.7271	0.7222

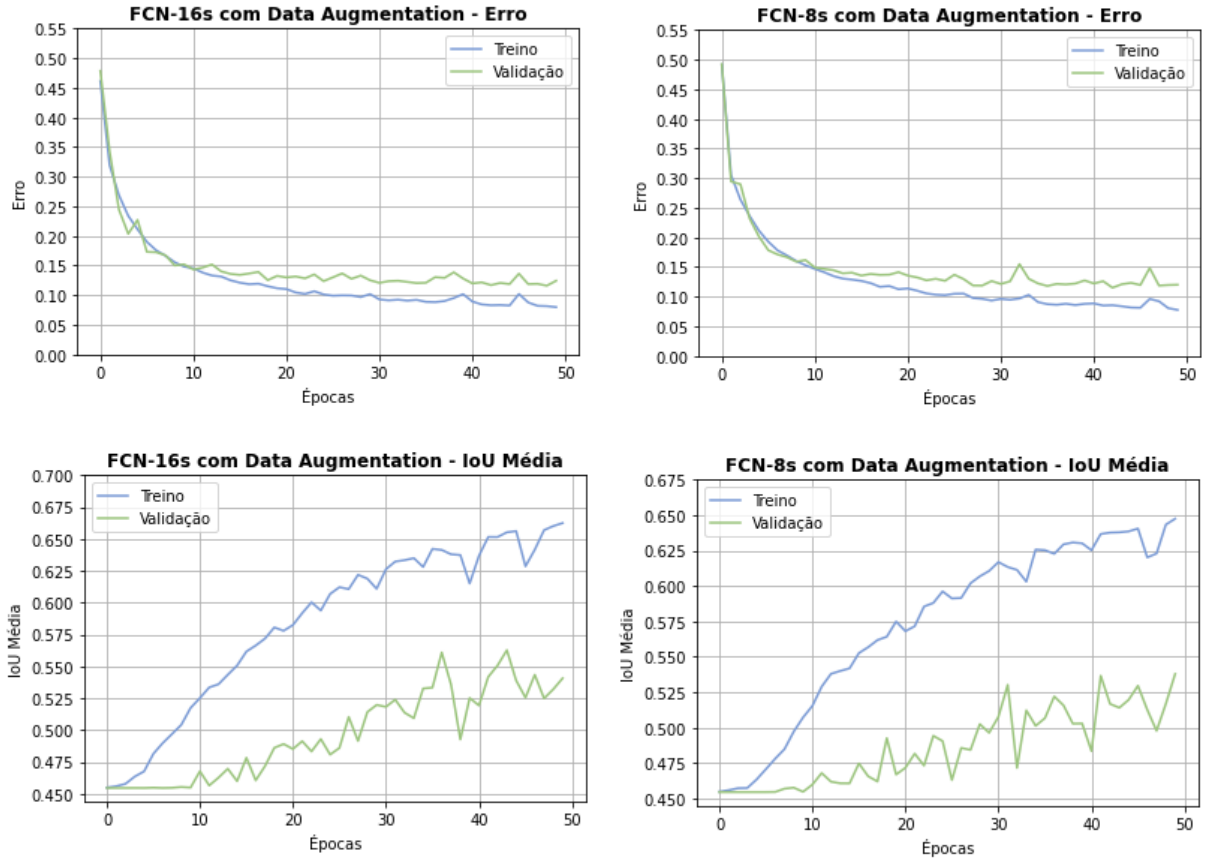
De acordo com os resultados obtidos, a aplicação de *data augmentation* ao conjunto de treino beneficia o desempenho do modelo para localizar os veículos. Isso é proveniente da maior quantidade de dados que o modelo possui para treinar, possibilitando maior generalização, como é observado nos altos valores de IoU média obtidos para os conjuntos de validação e teste. Além disso, o modelo treinado com *data augmentation* converge em menos épocas que o modelo treinado com o conjunto de treino padrão.

Para comparar o desempenho da U-Net com outras CNN, foram treinadas as redes FCN-16s e FCN-8s com *data augmentation* afim de se observar os resultados para diferentes arquiteturas de CNN. A Figura 18 mostra os gráficos do erro da IoU Média para as redes FCN-8s e FCN-16s. A Tabela 6 apresenta os resultados do erro e da IoU média avaliados no conjunto de treino, validação e teste na melhor configuração obtida com o treino para essas redes.

Tabela 6 - Erro e IoU Média nos diferentes conjuntos para as FCN

	Erro			**IoU Média**		
	Treino	Validação	Teste	Treino	Validação	Teste
FCN-16s	0.0799	0.1243	0.1556	0.6624	0.5407	0.5395
FCN-8s	0.0778	0.1201	0.1481	0.6474	0.5379	0.5356

Figura 18 - Gráficos do Erro e da IoU Média para FCN-16s e FCN-8s



Fonte: Autor desta monografia.

De acordo com os resultados mostrados nas Tabelas 5 e 6, o desempenho da U-Net é superior aos modelos FCN-16s e FCN-8s para a segmentação semântica. Os modelos de FCN-16s e FCN-8s apresentam menor IoU média e menor capacidade de generalização em imagens diferentes do conjunto de treino. Além disso, o modelo da U-Net apresenta menor quantidade de parâmetros treináveis, proporcionando redução no custo computacional ao treinar os modelos.

3.4.2. Resultados do Sistema

O modelo da U-Net com melhor desempenho foi integrado ao sistema descrito na seção 3.2.1 para avaliar os resultados em imagens de vídeos reais. Para isso, foram utilizados vídeos provenientes de câmeras de cruzamentos para que o sistema possa processar e

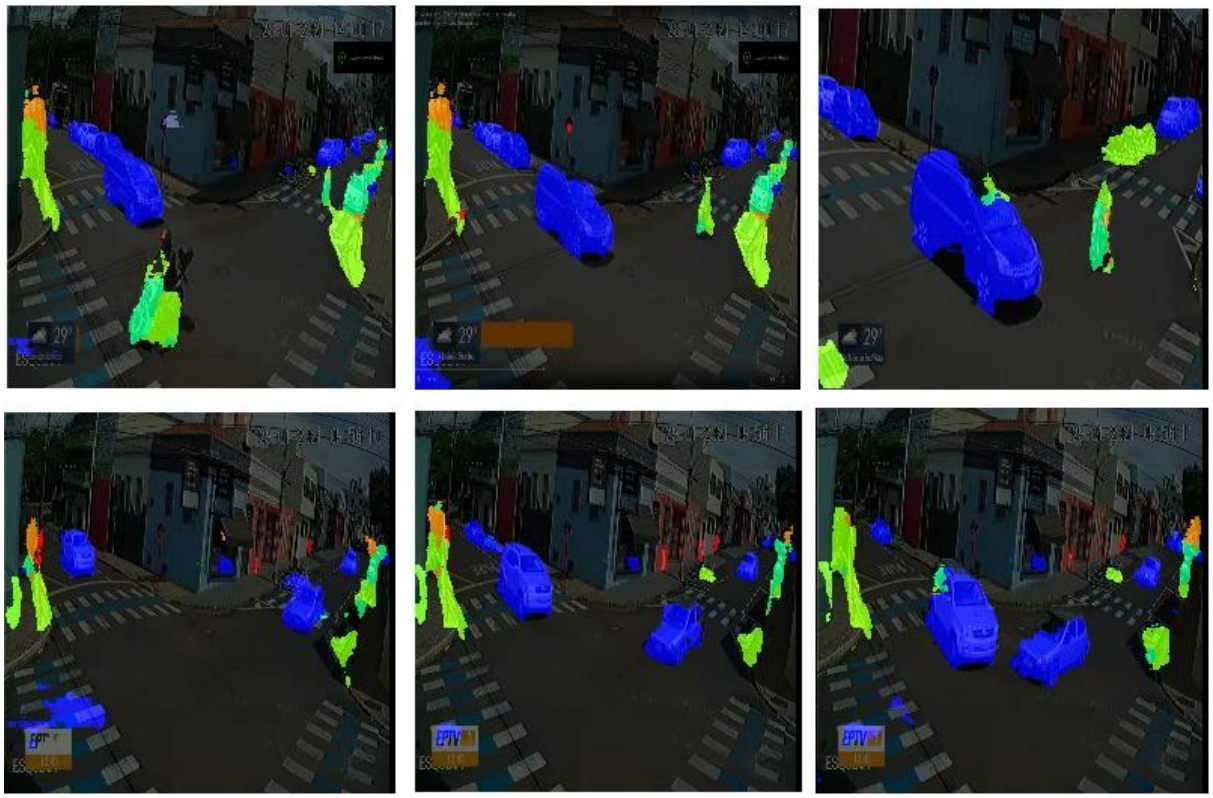
identificar veículos. Cada classe prevista pelo modelo é definida por uma cor e a legenda está mostrada na Figura 19.

Figura 19 - Legenda das classes da segmentação semântica

Fonte: Autor desta monografia.

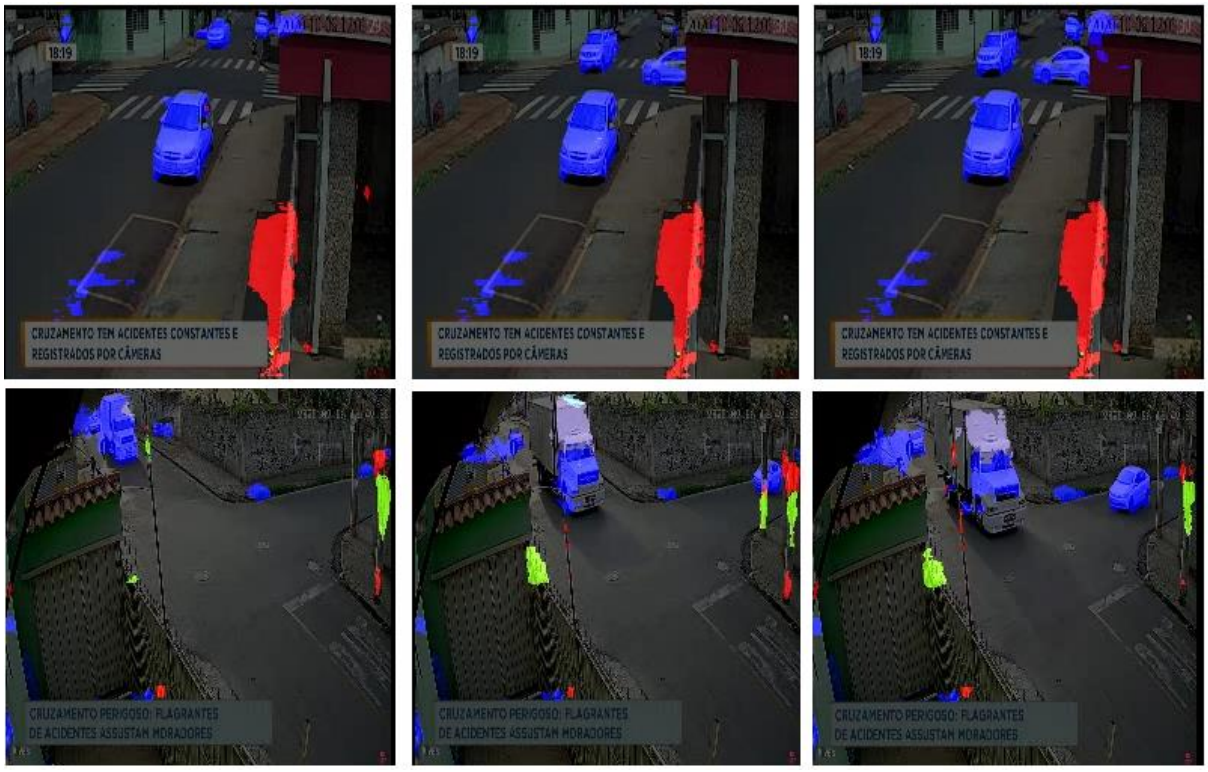
Os resultados da segmentação semântica nos vídeos estão mostrados nas Figuras 20 e 21.

Figura 20 - Resultados da segmentação semântica em vídeo 1



Fonte: Adaptado de https://g1.globo.com/sp/sao-carlos-regiao/noticia/2021/01/30/videos-cruzamento-no-centro-de-sao-carlos-tem-3-acidentes-nos-ultimos-3-dias.ghtml

Figura 21 - Resultados da segmentação semântica em vídeo 2



Fonte: Adaptado de https://www.youtube.com/watch?v=55XfxCPzD-s; https://www.youtube.com/watch?v=3B9d4Z4p18g

Com base nos resultados obtidos da segmentação semântica realizada, é possível ver que o sistema é capaz de localizar os veículos presentes nos vídeos durante sua locomoção. Além disso, o sistema apresenta maior facilidade na localização de carros, pertencentes a classe 3, devido a grande quantidade de imagens dessa classe no *dataset* de treino. Em razão da menor quantidade de imagens de outras classes, o sistema apresenta maior dificuldade em reconhecê-las. No geral, é possível concluir que o sistema apresenta capacidade para detectar veículos em movimento nos cruzamentos, sendo possível sinalizar possíveis riscos aos motoristas que trafegam por eles, aumentando a segurança no trânsito.

3.5. Dificuldades e Limitações

A principal dificuldade encontrada no desenvolvimento do projeto foi a determinação de um *dataset* adequado para o treinamento da CNN. Para aplicações que utilizam redes

neurais, é de extrema importância que o *dataset* utilizado para o treinamento corresponda aos dados reais nos quais a rede será utilizada. Para aplicações que detectam objetos via segmentação semântica, os *datasets* devem ser compostos pelas imagens e suas respectivas GT com todos pixels anotados por *labels* correspondentes às classes. O processo de anotação de imagens é um processo custoso que consome muito tempo devido a grande quantidade de dados necessários para treinamento e, por essa questão, foi escolhido um *dataset* completamente anotado para realização do projeto. Além disso, o *dataset* escolhido possui a limitação de não conter imagens provenientes de câmeras de monitoramento em cruzamentos e, portanto, não é completamente adequado para o desenvolvimento de projetos reais com essa finalidade.

Afim de se desenvolver um sistema com a finalidade de se detectar objetos por câmeras de monitoramento em cruzamentos, o ideal seria obter uma grande quantidade de imagens reais provenientes das câmeras e realizar o processo de anotação nessas imagens, com o propósito de gerar um *dataset* especifico para essa situação e assim aumentar o desempenho na detecção.

3.6. Considerações Finais

Neste capítulo foi abordado o desenvolvimento do trabalho e seus resultados. Foi realizada uma descrição da modelagem do sistema para detectar veículos em vídeos. Foram apresentados os recursos computacionais e a linguagem de programação utilizada para desenvolvimento do projeto. Também foram descritas todas as implementações realizadas para a construção do projeto. Posteriormente, foram apresentados os resultados obtidos com o treinamento das CNN e os resultados gerais do sistema. Por fim, foram discutidas as dificuldades encontradas durante o desenvolvimento do projeto. O capítulo seguinte consiste nas conclusões proporcionadas pelo desenvolvimento do projeto.

CAPÍTULO 4: CONCLUSÃO

4.1. Contribuições

A proposta do trabalho desenvolvido consiste no aumento da segurança no trânsito, através da utilização de sistemas inteligentes capazes de localizar veículos. Para isso, o projeto desenvolvido apresenta potencial para realização dessa tarefa. Com a utilização de conjuntos de dados mais específico e maior poder computacional para o treinamento de modelos de CNN, o sistema apresentado poderia ser utilizado para realizar o monitoramento das vias e assim contribuir com o aumento da segurança e a redução nos acidentes de trânsito.

O trabalho desenvolvido proporcionou ao autor um grande entendimento de como sistemas que utilizam *Deep Learning* funcionam, através de diversos experimentos realizados com os modelos de redes neurais convolucionais para realização do projeto. Além disso, o projeto proporcionou o aprendizado de uma nova linguagem de programação ao autor, que não possuía conhecimentos prévios em Python.

REFERÊNCIAS

ABADI, M. et al. **TensorFlow: Large-scale machine learning on heterogeneous systems**, 2015. Disponível em: <https://www.tensorflow.org/>. Acesso em: 20 jun. 2021.

ALOM, M. Z. et al. **The History began from AlexNet: A Comprehensive Survey on Deep Learning Approaches**, 2018. Disponível em <https://arxiv.org/abs/1803.01164>. Acesso em: 20 jun. 2021.

BADRINARAYANAN, V.; KENDALL, A.; CIPOLLA, R. **SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation**, 2015. Disponível em <https://arxiv.org/abs/1511.00561>. Acesso em: 20 jun. 2021.

CHOLLET, F. et al. **Keras**, 2015. Disponível em <https://keras.io/>. Acesso em: 20 jun. 2021.

CORDTS, M. et al. **The Cityscapes Dataset for Semantic Urban Scene Understanding**, in Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. Disponível em <https://www.cityscapes-dataset.com/>. Acesso em: 20 jun. 2021.

DATASUS; VIAS SEGURAS. **Estatísticas nacionais de acidentes de trânsito**, 2020. Disponível em <http://vias-seguras.com/os_acidentes/estatisticas/estatisticas_nacionais>. Acesso em: 20 jun. 2021.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**, MIT Press, 2016. Disponível em <https://www.deeplearningbook.org/>. Acesso em: 20 jun. 2021.

IBGE – INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. **Frota de Veículos**, 2006, 2020. Disponível em <https://cidades.ibge.gov.br/brasil/pesquisa/22/28120?ano=2020&indicador=28120&tipo=grafico>. Acesso em: 20 jun. 2021.

KINGMA, D. P.; BA, J. L. **Adam: A Method for Stochastic Optimization**, 2014. Disponível em <https://arxiv.org/abs/1412.6980>. Acesso em: 20 jun. 2021.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. **ImageNet Classification with Deep Convolutional Neural Networks**, 2012. Disponível em <https://papers.nips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf>. Acesso em: 20 jun. 2021.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep Learning. **Nature**, v. 521, p. 436-444, 2015. Disponível em <https://www.researchgate.net/publication/277411157_Deep_Learning>. Acesso em: 20 jun. 2021.

LONG, J.; SHELHAMER, E.; DARRELL, T. **Fully Convolutional Networks for Semantic Segmentation**, 2014. Disponível em <https://arxiv.org/abs/1411.4038v1>. Acesso em 20 jun. 2021.

RONNEBERGER, O.; FISCHER, P.; BROX, T. **U-Net: Convolutional Networks for Biomedical Image Segmentation**, 2015. Disponível em <https://arxiv.org/abs/1505.04597>. Acesso em: 20 jun. 2021

SIMONYAN, K.; ZISSERMAN, A. **Very Deep Convolutional Networks for Large-Scale Image Recognition**, 2014. Disponível em <https://arxiv.org/abs/1409.1556v1>. Acesso em: 20 jun. 2021.

SRIVASTAVA, N. et al. **Dropout: A Simple Way to Prevent Neural Networks from Overfitting**, 2014. Disponível em <https://jmlr.org/papers/v15/srivastava14a.html>. Acesso em: 20 jun. 2021.

SZEGEDY, C. et al. **Going Deeper with Convolutions**, 2014. Disponível em <https://arxiv.org/abs/1409.4842>. Acesso em: 20 jun. 2021.

TREML, M. et al. **Speeding up Semantic Segmentation for Autonomous Driving**, 2016. Disponível em <https://www.researchgate.net/publication/309935608_Speeding_up_Semantic_Segmentation_for_Autonomous_Driving>. Acesso em: 20 jun. 2021.

VOIGTLAENDER, P. et al. **MOTS: Multi-Object Tracking and Segmentation**, 2019. Disponível em <http://www.cvlibs.net/datasets/kitti/eval_mots.php>. Acesso em: 20 jun. 2021.

XING, Y.; ZHONG, L.; ZHONG, X. **An Encoder-Decoder Network Based FCN Architecture for Semantic Segmentation**, 2020. Disponível em <https://www.hindawi.com/journals/wcmc/2020/8861886/>. Acesso em: 20 jun. 2021.