UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Lucas Locatelli Helena

Método baseado em imagem para detecção de eixos e classificação de caminhões.

São Carlos

2020

Lucas Locatelli Helena

Método baseado em imagem para detecção de eixos e classificação de caminhões.

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Adilson Gonzaga

São Carlos

2020

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Helena, Lucas Locatelli
H474m Método baseado em imagem para detecção de eixos e classificação de caminhões. / Lucas Locatelli Helena; orientador Adilson Gonzaga. São Carlos, 2020.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2020.

1. Deep Learning. 2. YoloV3. 3. Faster R-CNN. 4. Single Shot Multibox Detector. 5. Python. 6. Keras. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Lucas Locatelli Helena

Título: "Método baseado em imagem para detecção de eixos e classificação de caminhões."

Trabalho de Conclusão de Curso defendido e aprovado em 30/11/2020,

com NOTA 7,5 (Sete,Cinco), pela Comissão Julgadora:

Prof. Associado Adilson Gonzaga - Orientador/Professor Sênior - SEL/EESC/USP

Prof. Dr. André Luiz Barbosa Nunes da Cunha - STT/EESC/USP

Dra. Carolina Toledo Ferraz - Pós-doutorado UNIFESP São Paulo

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

RESUMO

Helena, L. L. **Método baseado em imagem para detecção de eixos e classificação de caminhões.** . 2020. 84p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

As redes neurais convolucionais tem sido empregadas com frequência cada vez maior para detecção e classificação de imagens. Esse trabalho tem como objetivo empregar três das mais notáveis arquiteturas de detecção baseadas em CNNs e aplicá-las na detecção de eixos de caminhões, comparando-as. Os resultados obtidos demonstram que as arquiteturas se comportam de maneira mais que adequada, sendo possível notar as diferentes filosofias de uso e o *trade-off* entre FPS e acurácia para cada uma delas.

Palavras-chave: Deep Learning. YoloV3. Faster R-CNN. Single Shot Multibox Detector. Python. Keras.

ABSTRACT

Helena, L. L. **Image based method for axis detection and truck classification**. 2020. 84p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

Convolutional Neural Networks are being employed with increased frequency in the detection and classification of images. This work objective is to assess the usability of the three state-of-the-art detection architectures based on CNN's and apply them o the detection of trucks axis, comparing the results. The obtainded outcomes show that those architectures behave more than appropriately, it is even possible to notice the different usage philosophy of each network in teh trade-off between FPS and accuracy for each of them.

Keywords: Deep Learning. YoloV3. Faster R-CNN. Single Shot Multibox Detector. Python. Keras.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AP	*Average Precision*
CNN	*Convolutional Neural Network*
CSV	*Comma Separated Values*
FN	*False Negative*
FP	*False Positive*
FPS	*Frames Per Second*
GT	*Ground Truth*
IoU	*Intersection Over Union*
NMS	*Non-Maximum Suppresion*
RoI	*Region of Interest*
RPN	*Region Proposal Network*
R-CNN	*Region Based Convolutional Neural Network*
SSD	*Single Shot MultiBox Detector*
TP	*True Positive*
VoTT	*Visual Object Tagging Tool*
YOLO	*You Only Look Once*

SUMÁRIO

1 INTRODUÇÃO

O transporte rodoviário representa no Brasil 65% da movimentação de cargas, ou aproximadamente 1.548 bilhões de toneladas quilômetros úteis (TKU) (S.A., 2015), isso representa $4,4\%$ do PIB nacional(TRANSPORTE, 2001).

Apesar disso, quando comparado ao sistema de transporte dos EUA, a nossa produtividade é de apenas 22% (TRANSPORTE, 2001), então qualquer avanço técnológico que melhore a produtividade dessa área é muito bem vindo, por isso este trabalho busca entender e aplicar três arquiteturas de redes convolucionais de modo a agilizar processos e minimizar a burocracia envolvida, diminuindo a necessidade de interação humana.

As três arquiteturas de detecção utilizadas foram a *Faster* R-CNN, YOLOv3 e SSD, que são as arquiteturas mais difundidas atualmente e consideradas como estado da arte na detecção de objetos.

1.1 Objetivos do trabalho

Este trabalho visa empregar as arquiteturas de detecção de objetos na identificação de eixos de caminhões, para que futuramente o mesmo possa ser implementado de maneira a agilizar processos que atualmente são demorados e burocráticos, como por exemplo na pesagem de cargas e cobrança de tarifas em praças pedágio.

1.2 Organização do texto

O trabalho está dividido em 5 capítulos principais:

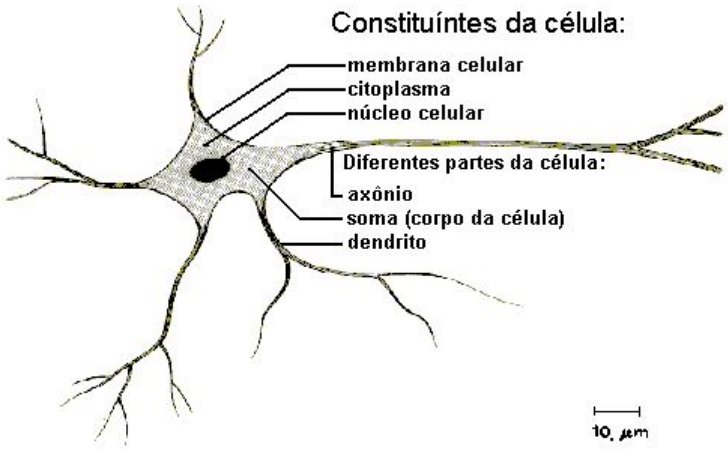
- No capítulo 2, denominado "Revisão Bibliográfica", é apresentada uma breve contextualização teórica dos assuntos abordados neste trabalho;
- O capítulo 3, "Materiais e Métodos", descreve os materiais utilizados e a metodologia adotada para treinar as redes neurais;
- O capítulo 4, "Resultados e Discussões", apresenta todos os resultados obtidos no capítulo anterior, compara-os e discute suas implicações;
- E por fim no capítulo 5, "Conclusões", são dispostas as considerações finais e estudadas possibilidades de trabalhos futuros relacionados a este.

2 REVISÃO BIBLIOGRÁFICA

2.1 Redes Neurais Artificiais

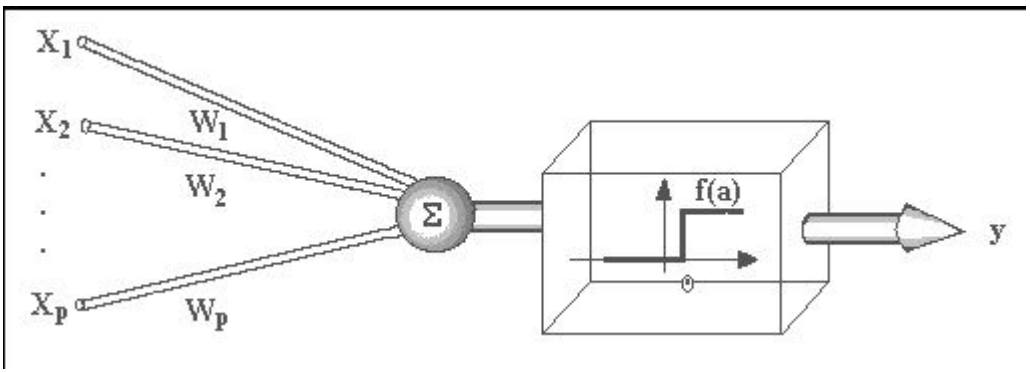
Redes Neurais Artificiais são modelos matemáticos que buscam simular o funcionamento de um cerébro para resolver problemas que outrora pareciam insolucionáveis. A rede neural artificial é composta, assim como um cerébro, de várias unidades básicas conectadas entre si, os perceptrons (equivalente matemático ao neurônio). As figuras 1 e 2 mostram a semelhança entre ambos.

Figura 1 – Neurônio Biológico



Fonte: (CARVALHO, 2009)

Figura 2 – Neurônio Artificial (Perceptron)



Fonte: Carvalho (2009)

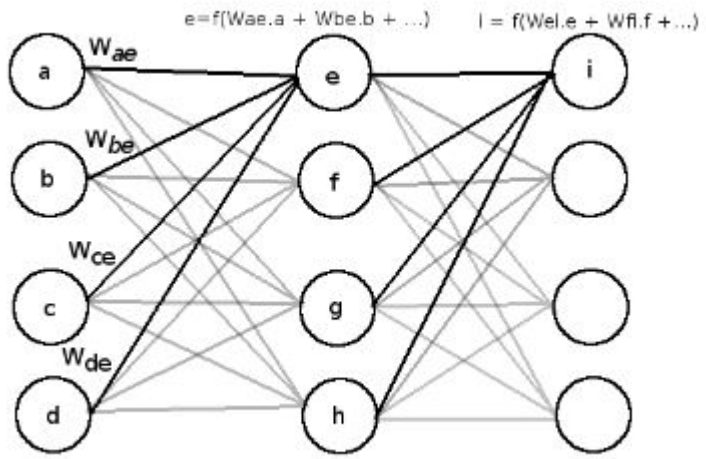
O perceptron é separado em três partes distintas, os dendritos, a soma e o axônio. Nos dendritos o perceptron recebe a entrada e a multiplica por um peso W_i, na soma

todas as entradas são somadas e no axônio o resultado da soma passará por uma função de ativação, gerando uma resposta às entradas originais.

2.1.1 Rede de Perceptron Multicamadas

Um único perceptron só consegue resolver problemas lineares, para problemas mais complexos é necessário organizar vários perceptrons em camadas, onde a entrada de dados de uma camada é a saída dos perceptrons da camada anterior, vide figura 3.

Figura 3 – Neurônio Artificial (Perceptron)



Fonte: Rocha (2015)

O treinamento dessa rede é baseado no algoritmo de *back propagation*, durante o treino os valores de entrada são inseridos na rede e a saída é comparada com a saída esperada, então é calculado o gradiente do erro e esse gradiente é utilizado para ajustar os parâmetros da rede, propagando da última camada até a primeira.

2.2 Redes Neurais Convolucionais (CNN)

Nas redes multicamadas todos os perceptrons da camada anterior estão conectados em todos os perceptrons da camada seguinte. Para problemas variantes no tempo e principalmente para problemas envolvendo imagens, onde as entradas do problema são as matrizes dos canais de cor da imagem, a posição de cada *pixel* de entrada com relação aos outros importa. Os pixels na vizinhança do *pixel* sendo observado tem muito mais peso para o perceptron do que um *pixel* do outro lado da imagem. Como o custo computacional para calcular os pesos dos perceptrons totalmente conectados é gigantesco, as redes convolucionais são utilizadas para contabilizar apenas os *pixels* mais próximos.

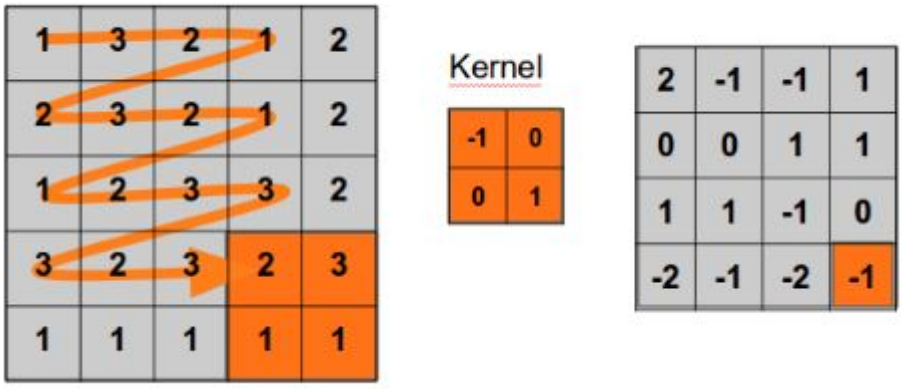
Cada camada de uma CNN tem diferentes funções. Cada uma delas será explicada com mais detalhes.

2.2.1 Camada de convolução

Cada camada de convolução é composta por um *kernel*(núcleo) que varre a imagem, como pode ser visto na figura 4, e é convoluído com a vizinhança do pixel atual, dando na saída um *feature map*(mapa de características).

O *kernel* é uma matriz *nxn* ímpar composta pelos pesos de entrada, a saída será uma outra matriz que contém o resultado da convolução entre o *kernel* e a matriz *n x n* derivada da imagem, centrada no pixel atual.

Figura 4 – Operação de convolução



Fonte: Rocha (2015)

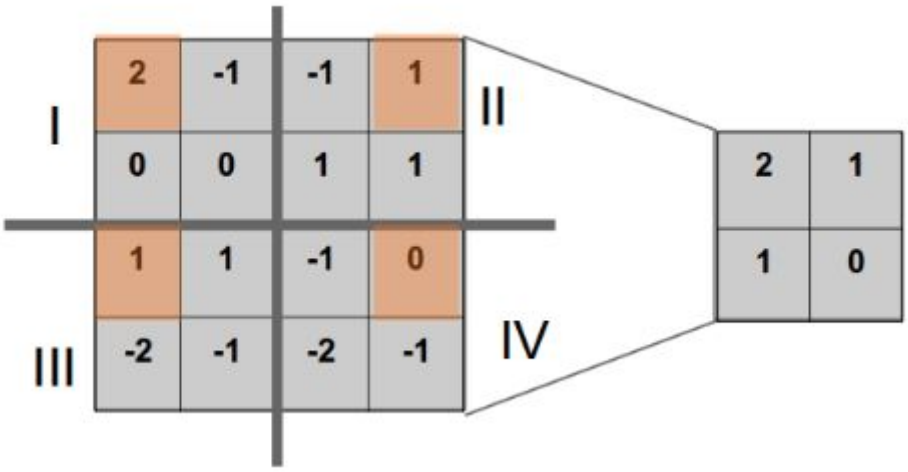
2.2.2 Camada de *Pooling*

A camada de *Pooling* reduz as dimensões da matriz de entrada por um fator constante, o que não apenas diminui o peso computacional da rede neural,como também reduz a sensibilidade da rede a pequenas variações na imagem(ROCHA, 2015), isso acontece pois as diversas *features* de uma região próxima são combinadas em uma única *feature*, o que diminui a redundância da rede.

Atualmente os *poolings* mais utilizados são o *max-pooling* e o *avg-pooling*, o *max-pooling* pega os valores da região do *pooling* e mantém o maior valor apenas (ver figura 5), já o *avg-pooling* mantém a média dos valores de entrada.

Neste trabalho será utilizado apenas o *max-pooling*.

Figura 5 – Operação de *max-pooling*



Fonte: Rocha (2015)

2.2.3 Camada *Softmax*

Após todas as operações de convolução e *pooling* necessárias a rede passa por uma camada *softmax*, essa camada transforma a matriz em uma única camada totalmente conectada, a soma dos resultados da saída vão ser sempre igual a 1 e todas a saídas são positivas, por isso pode-se interpretar as saídas como uma distribuição de probabilidade discreta da entrada pertencer a cada uma das classes de interesse.

2.3 *Faster R-CNN*

O sistema *Faster Region Based CNN*(R-CNN) é composto basicamente por três módulos(REN et al., 2015),uma rede de *backbone* que extrai os mapas de características, uma *CNN* que propõe as regiões nas quais os objetos poderão estar (RPN) e um módulo de detecção(GIRSHICK, 2015).

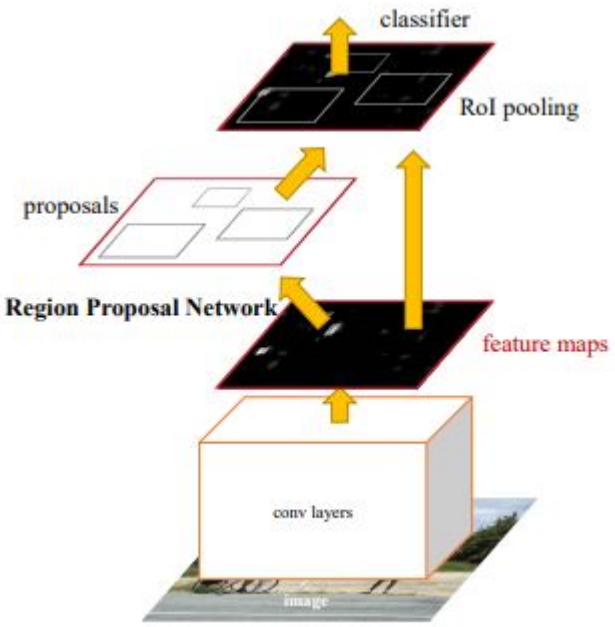
As figuras 6 e 7 mostram o esquema básico de funcionamento da *Faster R-CNN*.

2.3.1 *Region Proposal Network* (RPN)

A função da RPN é receber o mapa de características como entrada e mostrar na saída um conjunto de áreas retangulares, cada uma com sua probabilidade de conter o objeto.

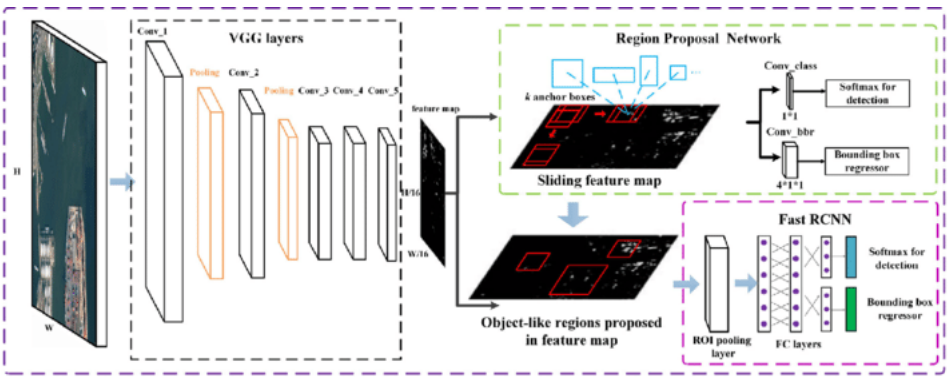
Para cada janela no mapa de características de entrada a RPN assume $k = 9$ *anchor boxes* com 3 escalas e 3 *aspect ratio* diferentes, como mostra a figura 8, assim cada

Figura 6 – Estrutura básica do *Faster* R-CNN



Fonte: Ren et al. (2015)

Figura 7 – Arquitetura *Faster* R-CNN

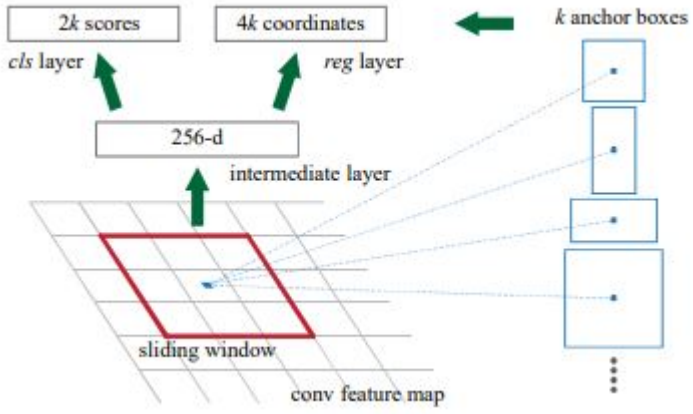


Fonte: Deng et al. (2018)

imagem gera um total de $W.H.k$ regiões de interesse(RoI), onde W e H são as dimensões do mapa de características.

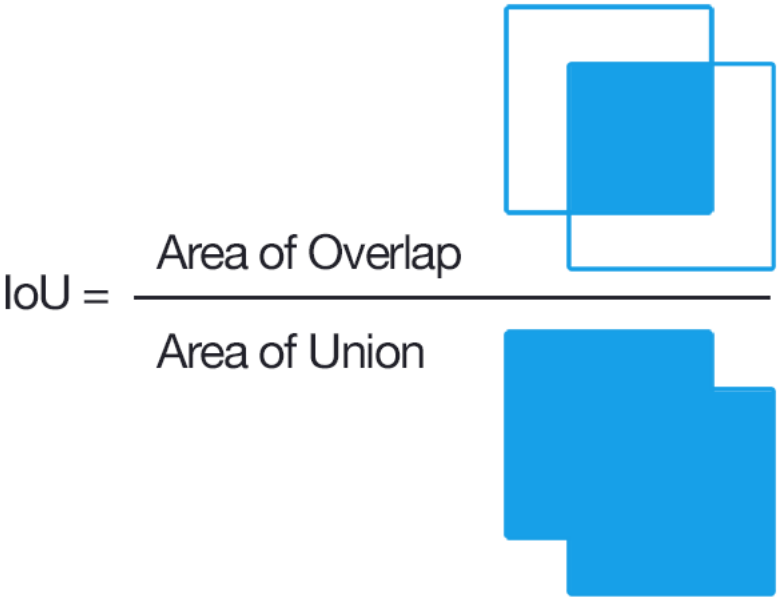
Para remover as redundâncias é aplicado o método de *Non-Maximum Suppression*(NMS) que consiste em pegar a *RoI* com maior probabilidade e computar a *Intersection over Union(IoU)*(figura 9) com todas as outras regiões, excluindo-as quando o IoU for superior ao limiar pré-definido, repetindo o processo até não existirem regiões redundantes(KIM, 2018).

Figura 8 – *Anchor Boxes*



Adaptado de: Ren et al. (2015)

Figura 9 – Equação de IoU



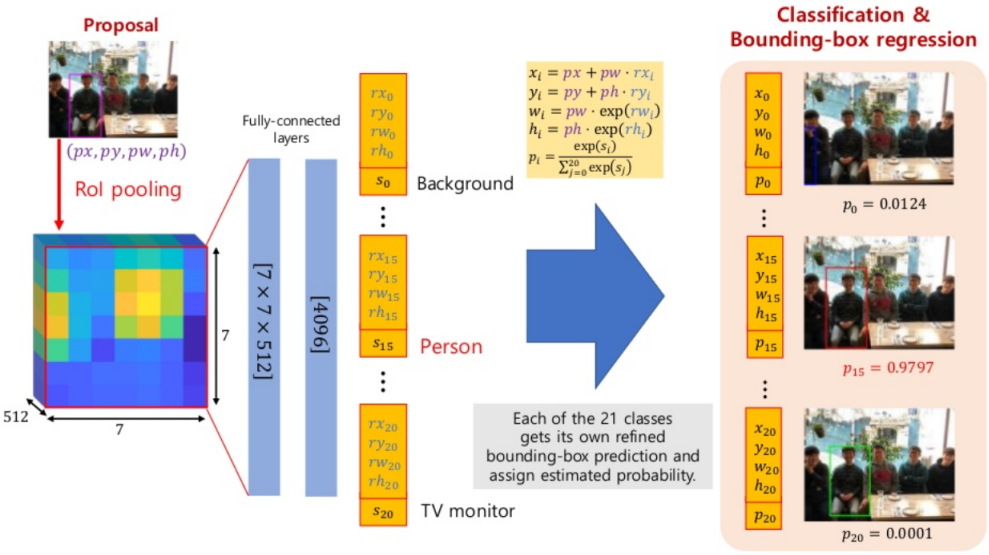
Fonte: Rosebrock (2016)

2.3.2 Classificação e Regressão

Para classificação e regressão a *Faster R-CNN* utiliza a mesma rede de sua antecessora(GIRSHICK, 2015), primeiro as *RoI's* e o mapa de características passam por uma camada de *pooling* e interpolação, em seguida elas passam por uma cada de *fully-connected layers* que nos dá a probabilidade de cada classe em cada RoI, depois as *RoI's* com probabilidades abaixo do limiar ou classificadas como plano de fundo são excluídas, e por último é aplicado mais uma vez o método *NMS* para excluir as redundâncias, este

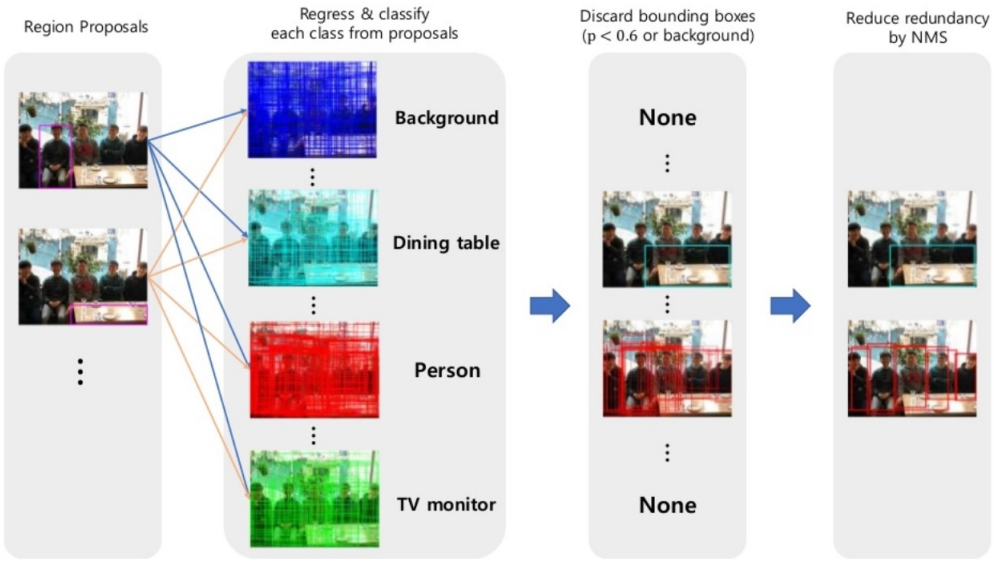
procedimento pode ser visualizado nas figuras 10 e 11

Figura 10 – Classificação e Regressão



Fonte: Kim (2018)

Figura 11 – Remoção de redundâncias utilizando NMS



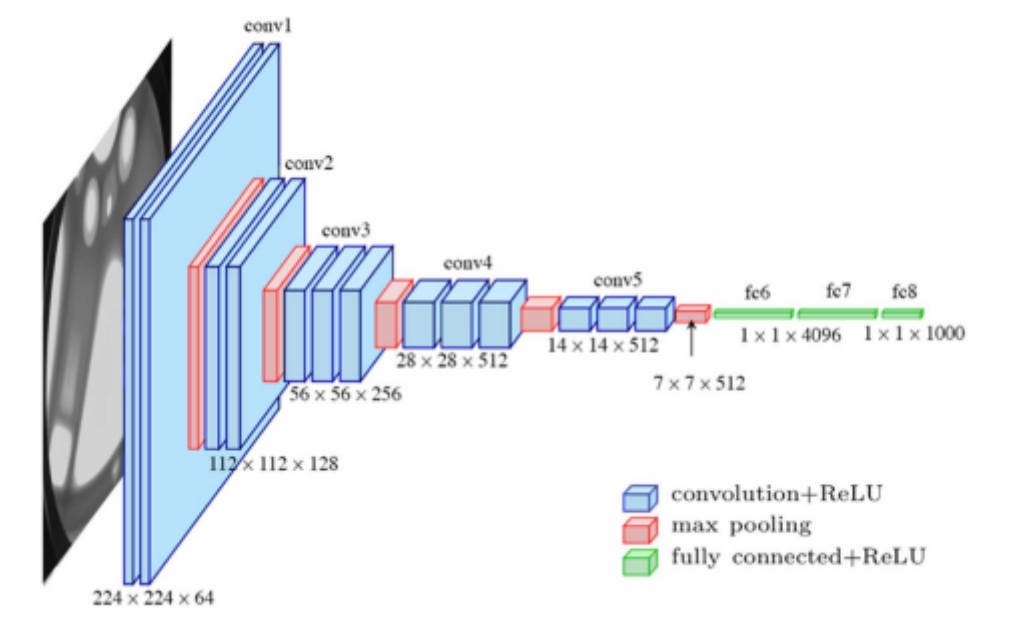
Fonte: Kim (2018)

2.4 *Single Shot MultiBox Detector(SSD)*

O método SSD(LIU et al., 2016) é baseado em uma rede convolucional de *feed-forward* que produz uma coleção de *bounding boxes* e probabilidades de presença das

classes de objetos nessas caixas, em sequência é aplicado um *NMS*, já discutida na seção do *Faster R-CNN*, para gerar a saída. O *backbone* da rede é baseado na arquitetura *VGG16* contendo todas as camadas até a última camada de convolução para extrair o mapa de características, conforme a figura 12.

Figura 12 – Arquitetura VGG16



Fonte: Ferguson et al. (2017)

2.4.1 Diferenciais

2.4.1.1 Mapas de características multi-escala

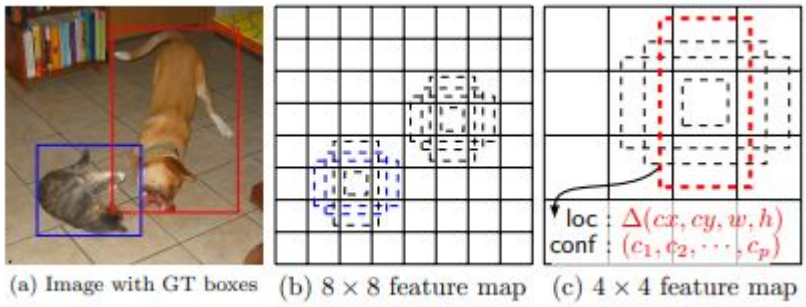
São adicionadas camadas de características ao final da rede de *backbone*. Essas camadas reduzem de tamanho progressivamente e permitem a detecção em diversas escalas, conforme mostra a figura 13.

2.4.1.2 Preditores convolucionais

Para cada camada de características é produzido um conjunto de predições de detecção usando uma série de filtros convolucionais, isso pode ser visto na figura 14.

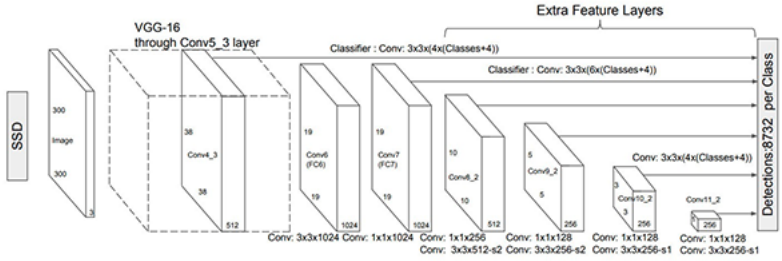
Para uma *layer* de características de tamanho $m \times n$ com p canais, o elemento básico para predizer os parâmetros de uma potencial detecção é um pequeno *kernel* que produz ou uma probabilidade para uma classe, ou um *offset* na forma, relativo as coordenadas padrões da *bounding box*. Em cada uma das mxn posições onde o *kernel* é aplicado é

Figura 13 – Múltiplas escalas de detecção



Fonte: Liu et al. (2016)

Figura 14 – Arquitetura SSD



Fonte: Liu et al. (2016)

gerado um valor de saída. Os valores de saída para *offset* das *bounding boxes* são medidos em relação à posição padrão relativa a cada posição do mapa de características.

2.4.1.3 *Boxes* e *Aspect Ratios* Padrões

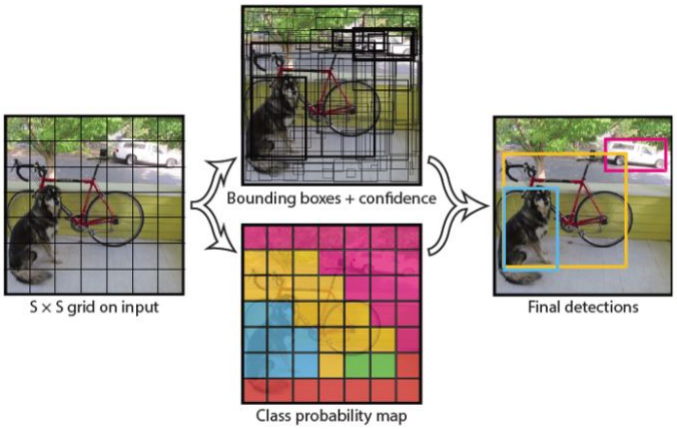
Para cada célula do mapa de características é associado um conjunto de *bounding boxes* padrão. As *boxes* padrões cobrem o mapa de características de uma maneira convolucional, de modo que a posição de cada *box* em relação à célula correspondente é fixa. Para cada *box* entre as k *boxes* são computadas c probabilidades de classe e 4 *offsets* relativas a *box* padrão, com isso tem-se um total de $(c + 4)kmn$ filtros que são aplicados a um mapa de características mxn. Aplicar as *boxes* a diversos mapas de características de diferentes resoluções permite um discretização eficiente do espaço.

2.5 *You Only Look Once*(YOLO)

YOLO é um método de detecção de objetos que alcança processamento em tempo real transformando a detecção de objetos em um simples problema de regressão (REDMON et al., 2015). Para isso a imagem de entrada em uma malha de $S \times S$ quadrados, se o

centro de um objeto cai em uma célula da malha, essa célula unitária é responsável por detectar aquele objeto. Cada célula prediz B *bounding boxes* e uma probabilidade da *boundig box* conter um objeto, conforme mostra a figura 15.

Figura 15 – Funcionamento básico da arquitetura YOLO



Fonte: Redmon et al. (2015)

2.5.1 Predição das *bounding boxes*

A rede prediz 4 coordenadas para cada *bounding box*, tx, ty, tw, th (REDMON; FARHADI, 2018) se a célula estiver deslocada do canto superior esquerdo da imagem por (C_x, C_y) e a *box* anterior tiver largura e altura p_w e p_h, respectivamente, então as predições são:

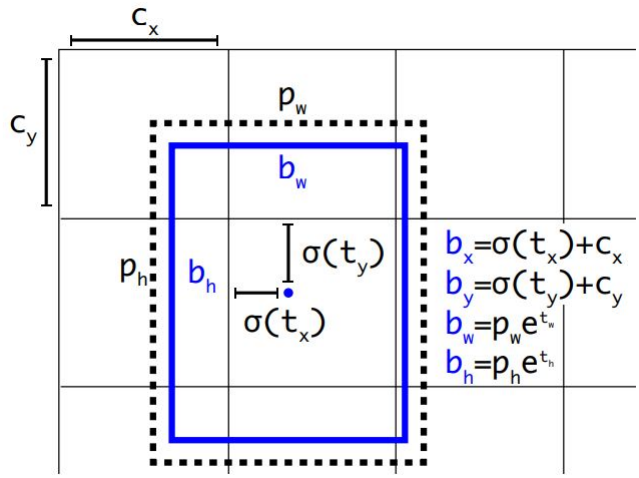
$$
\begin{aligned}
b_x &= \sigma(t_x) + C_x, \\
b_y &= \sigma(t_y) + C_y, \\
b_w &= p_w e^{t_w}, \\
b_h &= p_h e^{t_h}.
\end{aligned}
$$

A figura 16 exemplifica a predição das *bounding boxes*

2.5.2 Extração de características

Para extração de características é utilizada a rede $Darknet - 53$, figura 17, uma rede neural residual que com 53 camadas convolucionais que é composta de sucessivas camadas de tamanho 1×1 e 3×3.

Figura 16 – Predição das bounding boxes



Fonte: Redmon e Farhadi (2018)

Figura 17 – *Darknet-53*

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

Fonte: Redmon e Farhadi (2018)

2.6 *Average Precision* (AP)

Para classificar as diferentes arquiteturas é preciso encontrar uma boa métrica para comparação. Atualmente uma das métricas mais utilizadas para classificar detectores de objeto é a *Average Precision* (ou AP).

A métrica se baseia em 3 parâmetros:

- *True Positive* (TP) - Quando o objeto detectado está presente no *ground truth*;
- *False Positive* (FP) - Quando o objeto detectado não está presente no *ground truth*;
- *False Negative* (FN) - Quando o objeto está presente no *groun truth* e não foi detectado.

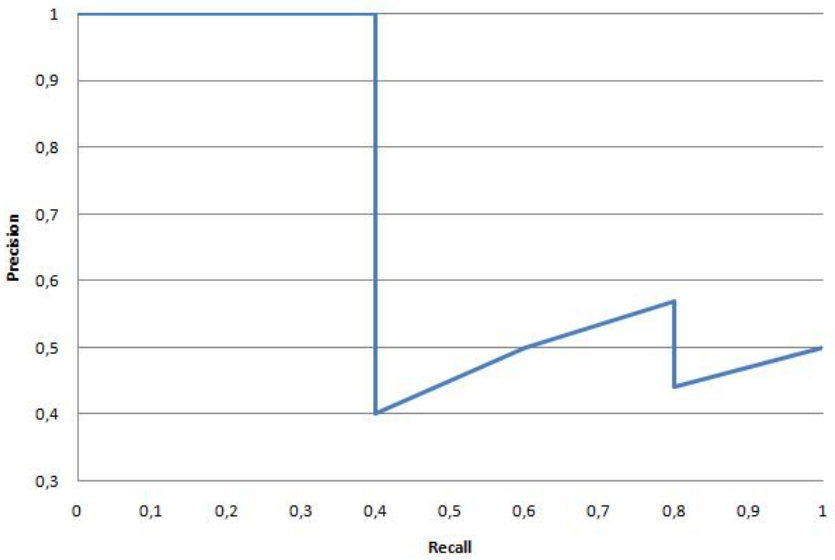
Com estes parâmetros podemos calcula-se a *precision*, que mede a precisão das predições da rede e o *recall*, que mede a capacidade da rede de encontrar todos os objetos relevantes Amorim (2019), as equações de *precision* e *recall* são mostradas em 2.1 e 2.2

$$Precision = \frac{TP}{TP + FN} \tag{2.1}$$

$$Recall = \frac{TP}{TP + FP} \tag{2.2}$$

Para calcular o AP, basta calcular a área abaixo da curva de *Precision x Recall*, porém como mostra a figura 18 conforme diminuimos o *threshold* de confiança da rede, o valor do *recall* aumenta, pois há uma diminuição no número de FN, porém a precisão diminui, já que o numero de FP aumenta. Entretanto essa relação não é linear, pois a diminuição do *threshhold* pode tambem gerar um aumento nos casos de TP, formando zigue-zagues na curva.

Figura 18 – Curva AP



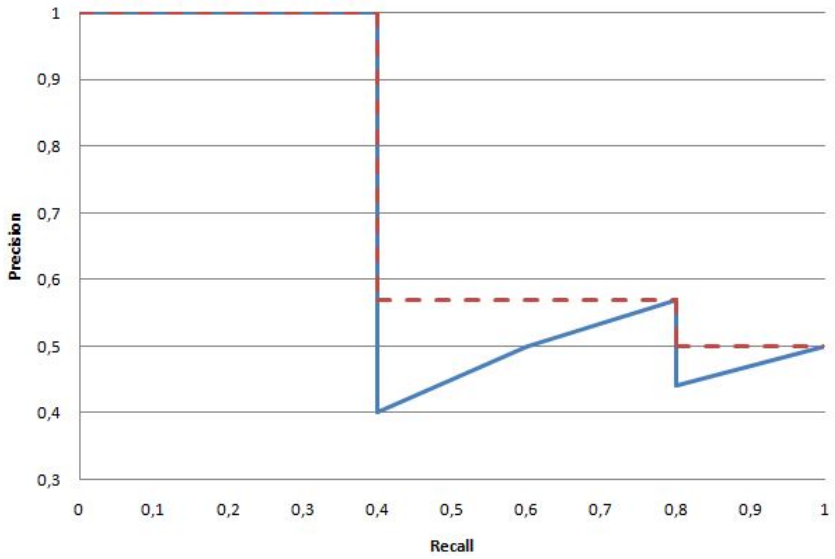
Fonte: Hui (2018)

Para evitar o erro de comparação entre *datasets* devido a esse padrão é necessário suavizar a curva, para isso deve-se interpolar pontos na curva tais que,

$$P_{interp}(r) = max\, p(\tilde{r})\ \forall\, \tilde{r} \geq r$$

Assim obtem-se a curva mostrada na figura 19, agora basta calcular a integral abaixo da curva para se obter o AP da rede.

Figura 19 – Curva AP interpolada



Fonte: Hui (2018)

3 MATERIAIS E MÉTODOS

3.1 Banco de Imagens

Para treinar as redes foram utilizadas 399 imagens laterais de caminhão, figura 20, obtidas no banco de dados do LabITS do departamento de engenharia de transportes da EESC. Dessas imagens foram separadas 346 para treino, que foram divididas em 90% para o treino e 10% para a validação, e 53 para testar a rede após os treinamentos.

Figura 20 – Exemplo de imagem do banco de dados

Fonte: Cunha e Marcomini (2018)

Como o aprendizado das redes será supervisionado se faz necessário anotar nas imagens todos os *ground truths*, para isso foi utilizada a ferramenta da Microsoft, VoTT (*Visual Object Tagging Tool*, (VOTT, 2020). Ela é uma ferramenta visual *open source* que permite marcar os *ground truths* nas imagens, figura 21, e os converte para os mais diversos tipos de dados utilizados pelas principais arquiteturas de detecção, como .CSV, .XML e .JSON. Para este projeto sera utilizado o formato .CSV (*Comma Separated Values* ou Valores separados por vírgula), que dará uma listagem de todos os *ground truths* com as seguintes informações:

$$Imagem, xmin, xmax, ymin, ymax, Classe.$$

A figura 22 mostra o significado prático de $xmin$, $xmax$, $ymin$, $ymax$ e a figura 23 mostra um exemplo do arquivo arquivo .CSV gerado pelo VoTT.

3.2 YOLOv3

Para treinar a rede na arquitetura YOLOv3 foi utilizada uma versão adaptada da biblioteca obtida em (KERAS-YOLO3, 2018), todos os códigos usados neste projeto

Figura 21 – Imagem marcada no VoTT

Fonte: VoTT (2020)

podem ser encontrados no anexo A.

Para agilizar o processo de aprendizado da rede foram usados pesos pré-treinados, obtidos em (YOLOV3. . . , 2018).

A tabela 1 mostra os parâmetros de treino utilizados neste projeto.

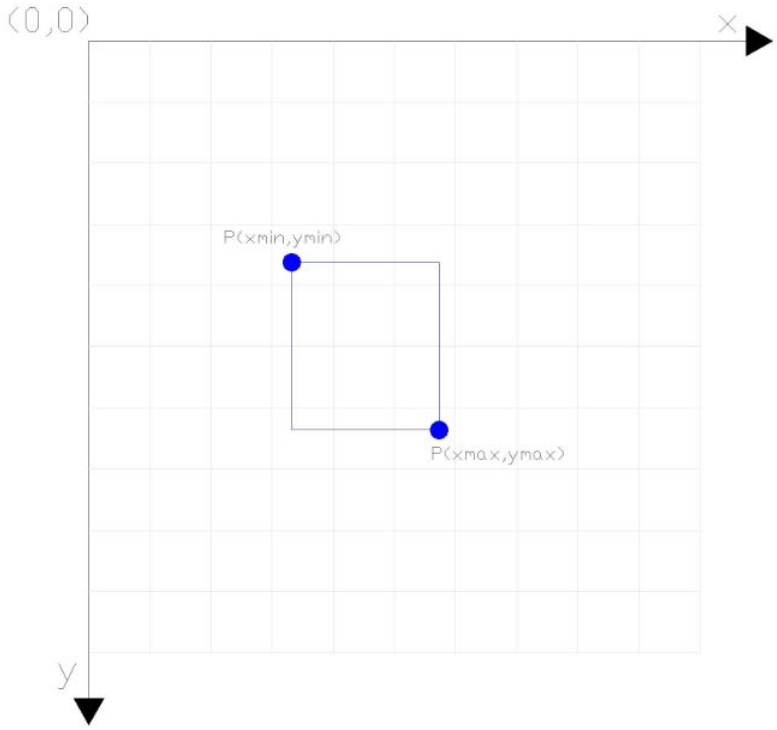
Tabela 1 – Parâmetros de treinamento para YOLOv3

Parâmetro de treinamento	Valor utilizado
Épocas	102
Imagens	346
Validação	10%
Otimizador	Adam
Learning Rate	1e-3
Batch Size	100

Fonte: Elaborada pelo autor.

Para as primeiras 51 épocas de treinamento todas as camadas da rede menos as quatro últimas foram congeladas. Para agilizar o treinamento e para estabilizar as perdas da rede, para as últimas 51 épocas todas as camadas foram descongeladas, o *batch size* e o *Learning Rate* para 32 e $1e-4$ respectivamente, permitindo um ajuste fino final para a rede.

Figura 22 – Coordenadas do *ground truth*



Fonte: Elaborado pelo autor.

Figura 23 – Arquivo CSV

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	image,"xmin","ymin","xmax","ymax","label"												
2	20160927110318-550_color-%5BROI-1%5D-12.jpg,990.7949790794978,627.8980719237435,1109.958158995816,741.6242417677643,"Eixo"												
3	20160927110318-550_color-%5BROI-1%5D-12.jpg,1600,622.5462521663778,1720.5020920502093,737.61037694974,"Eixo"												
4	20160927110318-550_color-%5BROI-1%5D-12.jpg,1296.0669456066944,490.0887131715771,1404.5188284518827,585.0835138648181,"Eixo"												
5	20160927110318-550_color-%5BROI-1%5D-12.jpg,954.6443514644351,488.75075823223574,1055.062761506276,581.0696490467938,"Eixo"												

Fonte: Elaborado pelo autor.

3.3 *Faster* R-CNN

O segundo método utilizado foi o *Faster R-CNN*, assim como na arquitetura anterior foram utilizados pesos pré treinados, no *database ImageNet*, para a rede *ResNet50*, obtidos na biblioteca *Keras*(RESNET50. . . , 2020), para agilizar o processo de treino, a biblioteca básica para treinar a rede pode ser encontrada em (KERAS-FRCNN, 2020).

A tabela 2 mostra os parâmetros de treino utilizados.

Tabela 2 – Parâmetros de treinamento para *Faster* R-CNN

Parâmetro de treinamento	Valor utilizado
Épocas	100
Imagens	346
Validação	10%
Regiões de interesse	32
Otimizador	Adam
Learning Rate	1e-5
Batch Size	100

Fonte: Elaborada pelo autor.

3.4 SSD

Para treinar a rede na arquitetura SSD foi usada uma versão modificada da biblioteca obtida em (SSD-KERAS, 2018) . Os pesos pré-treinados no *database ImageNet* também podem ser encontrados na biblioteca.

A tabela 3 contém os parâmetros de treinamento utilizados.

Tabela 3 – Parâmetros de treinamento para SSD

Parâmetro de treinamento	Valor utilizado
Épocas	100
Imagens	346
Validação	10%
Otimizador	Adam
Learning Rate	1e-3
Batch Size	100

Fonte: Elaborada pelo autor.

Após 80 épocas a *Learning Rate* foi diminuida para 1e-4 para ajustes mais finos na rede.

4 RESULTADOS E DISCUSSÕES

Foram obtidos para as três redes, resultados acima da média esperada, tabela 4 com detecções precisas e com graus de confiança altos, figuras 24, 25, 26 e 27, apesar disso nenhuma das redes obteve cem por cento de acerto, seja por oclusões dos objetos (figura 28) ou eixos no *background* (figura 29).

Tabela 4 – Desempenho das arquiteturas em outros bancos de dados

Arquitetura	*dataset*	mAP
YOLOv3	COCO	33%
Faster R-CNN	COCO+07+12	78,8%
SSD	COCO+07+12	79,6%

Fonte: Adaptada de (REDMON; FARHADI, 2018), (REN et al., 2015) e (LIU et al., 2016)

Os metódos selecionados para verificar a eficácia da rede foram: o de AP (*Average Precision*) para avaliar a precisão das detecções e o FPS (*Frames Per Second*) para avaliar a velocidade. Para encontrar o AP, foi utilizado o código encontrado em (MAP, 2020) e o FPS foi medido nas imagens de teste pela equação 4.1.

$$FPS = \frac{Number\ of\ Images}{Processing\ Time}; \tag{4.1}$$

A tabela 5 mostra o resumo das detecções de teste e a tabela 6 apresentam o AP e FPS de cada rede.

Tabela 5 – Resumo de Treinamento

Arquitetura	GT	TP	FP	FN
YOLOv3	173	154	2	19
Faster R-CNN	173	169	3	4
SSD	173	167	0	6

Fonte: Elaborada pelo autor.

Com os valores de AP e FPS traça-se o gráfico da figura 30. Nele percebe-se que apesar de ser a arquitetura mais precisa, a *faster* R-CNN é a mais lenta, sendo 11 vezes mais lenta que a SSD e 5 vezes mais lenta que a YOLO, com uma pequena perda na

Tabela 6 – Resultados de treinamento

Arquitetura	AP	FPS
YOLOv3	88,57%	0,5
Faster R-CNN	97,68%	0,1
SSD	96,53%	1,1

Fonte: Elaborada pelo autor.

acurácia percebe-se que a melhor arquitetura, combinando acurácia e velocidade foi a SSD, enquanto para esta aplicação a arquitetura YOLOv3 foi mais lenta e menos precisa que a SSD.

Figura 24 – Exemplos de detecções YOLOv3

Fonte: Elaborado pelo autor.

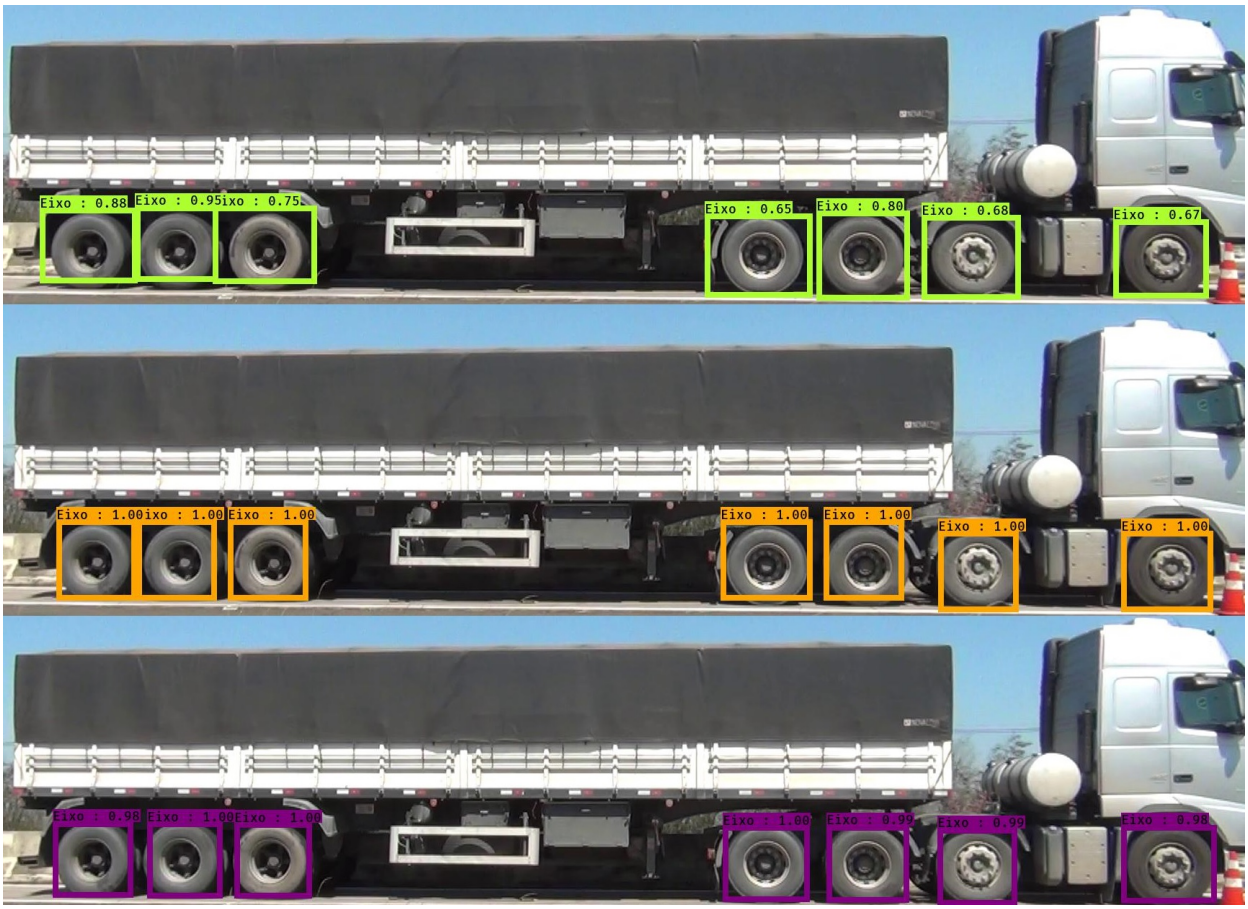
Figura 25 – Exemplos de detecções *Faster* R-CNN

Fonte: Elaborado pelo autor.

Figura 26 – Exemplos de detecções SSD

Fonte: Elaborado pelo autor.

Figura 27 – Comparação dos graus de confiança das detecções; YOLO, *Faster* R-CNN e SSD respectivamente.



Fonte: Elaborado pelo autor.

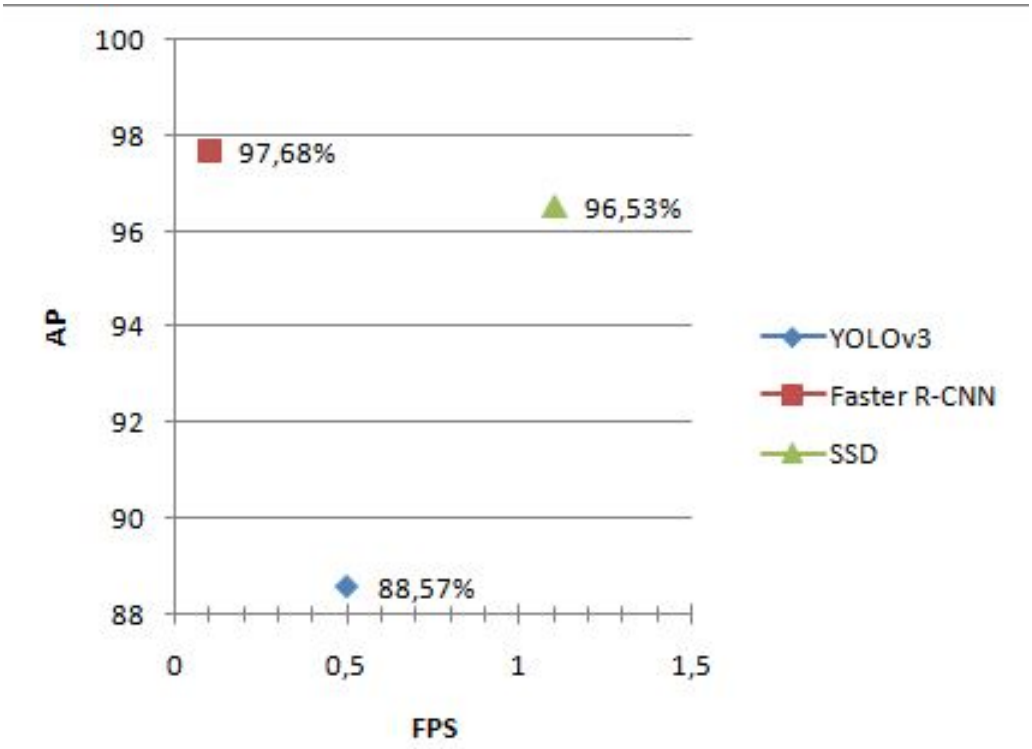
Figura 28 – Exemplos de oclusão atrapalhando a detecção

Fonte: Elaborado pelo autor.

Figura 29 – Exemplos de eixo no plano de fundo não localizado

Fonte: Elaborado pelo autor.

Figura 30 – Comparação das arquiteturas



Fonte: Elaborado pelo autor.

5 CONCLUSÕES

O objetivo deste trabalho é utilizar as três arquiteturas de CNNs mais populares de detecção de objetos, YOLOv3, Faster R-CNN e SSD, na detecção de eixos de rodagem de caminhões.

De forma geral os resultados demonstraram bom desempenho, mostrando uma eficácia nas detecções acima das porcentagens obtidas para as mesmas redes em outros bancos de dados. As arquiteturas se mostraram robustas mesmo com as dificuldades técnicas, como poucas épocas de treinamento e pouca quantidade de imagens para treinamento(346 imagens). As CNNs em geral necessitam de um número bem maior de imagens para que a rede não se especialize em apenas um grupo de imagens (overffiting).

O tempo de processamento para cada imagem foi adequado até para máquinas menos potentes e treinando apenas na CPU, sem usar a GPU, mostrando a portabilidade e replicabilidade do trabalho.

5.1 Trabalhos Futuros

Para trabalhos futuros com base neste projeto, são propostos:

- Aplicação das redes treinadas em vídeos ou *feeds* ao vivo para detecção em tempo real;
- Modificação dos parâmetros de treinamento para classificar os caminhões detectados baseado na quantidade e tipo de eixo detectado (eixo simples, duplo, duplo em tandem, triplo ou triplo em tandem);
- Implementação de um sistema de *feedback* para re-treinar a rede a partir de novas imagens detectadas.

REFERÊNCIAS

AMORIM, J. G. A. **Mean Average Precision**. 2019. Disponível em: <http://www.lapix.ufsc.br/ensino/visao/visao-computacionaldeep-learning/visao-computacionalmetricasmean-average-precision/>.

CARVALHO, A. P. de Leon F. de. **Redes Neurais Artificiais**. 2009. Disponível em: <https://sites.icmc.usp.br/andre/research/neural/>.

CUNHA, A. L. B. N. da; MARCOMINI, L. A. **A Comparison between Background Modelling Methods for Vehicle Segmentation in Highway Traffic Videos**. 2018. Disponível em: <https://doi.org/10.5281/zenodo.3612131>.

DENG, Z. et al. Multi-scale object detection in remote sensing imagery with convolutional neural networks. **ISPRS Journal of Photogrammetry and Remote Sensing**, 2018.

FERGUSON, M. et al. Automatic localization of casting defects with convolutional neural networks. In: . [S.l.: s.n.], 2017. p. 1726–1735.

GIRSHICK, R. **Fast R-CNN**. 2015.

HUI, J. **mAP (mean Average Precision) for Object Detection**. 2018. Disponível em: <https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173#:~:text=mAP\%20(mean\%20average\%20precision)\%20is,difference\%20between\%20AP\%20and\%20mAP>.

KERAS-FRCNN. 2020. Disponível em: <https://github.com/kbardool/keras-frcnn>.

KERAS-YOLO3. 2018. Disponível em: <https://github.com/qqwweee/keras-yolo3>.

KIM, H. P. **Faster R-CNN Object Detection: Localization & Classification**. 2018. Disponível em: <https://www.slideshare.net/hpkim0512/tutorial-of-faster-rcnn>.

LIU, W. et al. Ssd: Single shot multibox detector. **Lecture Notes in Computer Science**, Springer International Publishing, p. 21–37, 2016. ISSN 1611-3349. Disponível em: <http://dx.doi.org/10.1007/978-3-319-46448-0_2>.

MAP. 2020. Disponível em: <https://github.com/Cartucho/mAP>.

REDMON, J. et al. **You Only Look Once: Unified, Real-Time Object Detection**. 2015.

REDMON, J.; FARHADI, A. **YOLOv3: An Incremental Improvement**. 2018.

REN, S. et al. **Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks**. 2015.

RESNET50 Weights. 2020. Disponível em: <https://keras.rstudio.com/reference/application_resnet50.html>.

ROCHA, R. H. S. **Reconhecimento de Objetos por Redes Neurais Convolutivas**. 2015.

ROSEBROCK, A. **Intersection over Union (IoU) for object detection**. 2016. Disponível em: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>.

S.A., E. de Planejamento e L. **Plano Nacional de Logística**. 2015.

SSD-KERAS. 2018. Disponível em: <https://github.com/pierluigiferrari/ssd_keras>.

TRANSPORTE, C. N. do. **Transporte de cargas no Brasil: Ameaças e Oportunidades para o Desenvolvimento do País**. 2001.

VOTT. 2020. Disponível em: <https://github.com/microsoft/VoTT>.

YOLOV3 Weights. 2018. Disponível em: <https://pjreddie.com/media/files/yolov3-tiny.weights>.

ANEXO A – CÓDIGO DE TREINAMENTO YOLOV3

```
import os
import numpy as np
import keras.backend as K
from keras.layers import Input, Lambda
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import TensorBoard, ModelCheckpoint, ReduceLROnPlateau, EarlyStopping

from yolo3.model import preprocess_true_boxes, yolo_body, tiny_yolo_body, yolo_loss
from yolo3.utils import get_random_data


from time import time


def get_parent_dir(n=1):
    """ returns the n-th parent dicrectory of the current
    working directory """
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


def _main():

#Caminhos para os arquivos

    Data_Path = os.path.join(get_parent_dir(0), "Data")
    Image_Folder = os.path.join(Data_Path, "Source_Images", "Training_Images")
    annotation_path = os.path.join(Image_Folder, "data_train.txt")

    Model_Folder = os.path.join(Data_Path, "Model_Weights")
    log_dir = Model_Folder

    classes_path = os.path.join(Model_Folder, "data_classes.txt")
    anchors_path = os.path.join("utils","model_data", "yolo_anchors.txt")
    weights_path = os.path.join("utils", "yolo.h5")
    class_names = get_classes(classes_path)
    num_classes = len(class_names)
    anchors = get_anchors(anchors_path)

    input_shape = (416,416) # multiplo de 32, altura x largura

    #quantidade de epocas por periodo de treinamento
    epoch1, epoch2 = 51, 51


    model = create_model(input_shape, anchors, num_classes,
            freeze_body=2, weights_path=weights_path)

```

```
52
53      log_dir_time = os.path.join(log_dir, "{}".format(int(time())))
54      logging = TensorBoard(log_dir=log_dir)
55
56
57      #checkpoint
58      checkpoint = ModelCheckpoint(
59          os.path.join(log_dir, "checkpoint.h5"),
60          monitor="val_loss",
61          save_weights_only=True,
62          save_best_only=True,
63          period=5,
64      )
65
66
67      reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1)
68      early_stopping = EarlyStopping(monitor='val_loss', min_delta=0, patience=10, verbose=1)
69
70      val_split = 0.1
71      with open(annotation_path) as f:
72          lines = f.readlines()
73
74
75
76
77
78      #separa as imagens de treino e validação
79      np.random.seed(10101)
80      np.random.shuffle(lines)
81      np.random.seed(None)
82      num_val = int(len(lines)*val_split)
83      num_train = len(lines) - num_val
84
85      # Train with frozen layers first, to get a stable loss.
86      # Adjust num epochs to your dataset. This step is enough to obtain a not bad model.
87      if True:
88          model.compile(optimizer=Adam(lr=1e-3), loss={
89              # use custom yolo_loss Lambda layer.
90              'yolo_loss': lambda y_true, y_pred: y_pred})
91
92          batch_size = 100
93          print('Train on {} samples, val on {} samples, with batch size
            ↪ {}.'.format(num_train, num_val, batch_size))
94
95
96
97
98          history = model.fit_generator(
99              data_generator_wrapper(
100                 lines[:num_train], batch_size, input_shape, anchors, num_classes
101             ),
102             steps_per_epoch=max(1, num_train // batch_size),
103             validation_data=data_generator_wrapper(
104                 lines[num_train:], batch_size, input_shape, anchors, num_classes
105             ),
106             validation_steps=max(1, num_val // batch_size),
```

```
107             epochs=epoch1,
108             initial_epoch=0,
109             callbacks=[logging, checkpoint],
110         )
111         model.save_weights(os.path.join(log_dir, "trained_weights_stage_1.h5"))
112 
113         step1_train_loss = history.history["loss"]
114 
115         file = open(os.path.join(log_dir_time, "step1_loss.npy"), "w")
116         with open(os.path.join(log_dir_time, "step1_loss.npy"), "w") as f:
117             for item in step1_train_loss:
118                 f.write("%s\n" % item)
119         file.close()
120 
121         step1_val_loss = np.array(history.history["val_loss"])
122 
123         file = open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w")
124         with open(os.path.join(log_dir_time, "step1_val_loss.npy"), "w") as f:
125             for item in step1_val_loss:
126                 f.write("%s\n" % item)
127         file.close()
128 
129     # Unfreeze and continue training, to fine-tune.
130     # Train longer if the result is not good.
131     if True:
132         for i in range(len(model.layers)):
133             model.layers[i].trainable = True
134         model.compile(optimizer=Adam(lr=1e-4), loss={'yolo_loss': lambda y_true, y_pred:
        ↪  y_pred}) # recompile to apply the change
135         print('Unfreeze all of the layers.')
136 
137         batch_size = 2 # note that more GPU memory is required after unfreezing the body
138         print('Train on {} samples, val on {} samples, with batch size
        ↪  {}.'.format(num_train, num_val, batch_size))
139 
140 
141         history = model.fit_generator(
142             data_generator_wrapper(
143                 lines[:num_train], batch_size, input_shape, anchors, num_classes
144             ),
145             steps_per_epoch=max(1, num_train // batch_size),
146             validation_data=data_generator_wrapper(
147                 lines[num_train:], batch_size, input_shape, anchors, num_classes
148             ),
149             validation_steps=max(1, num_val // batch_size),
150             epochs=epoch1 + epoch2,
151             initial_epoch=epoch1,
152             callbacks=[logging, checkpoint, reduce_lr, early_stopping],
153         )
154         model.save_weights(os.path.join(log_dir, "trained_weights_final.h5"))
155         step2_train_loss = history.history["loss"]
156 
157         file = open(os.path.join(log_dir_time, "step2_loss.npy"), "w")
158         with open(os.path.join(log_dir_time, "step2_loss.npy"), "w") as f:
159             for item in step2_train_loss:
160                 f.write("%s\n" % item)
```

```
        file.close()

        step2_val_loss = np.array(history.history["val_loss"])

        file = open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w")
        with open(os.path.join(log_dir_time, "step2_val_loss.npy"), "w") as f:
            for item in step2_val_loss:
                f.write("%s\n" % item)
        file.close()



def get_classes(classes_path):
    '''loads the classes'''
    with open(classes_path) as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]
    return class_names

def get_anchors(anchors_path):
    '''loads the anchors from a file'''
    with open(anchors_path) as f:
        anchors = f.readline()
    anchors = [float(x) for x in anchors.split(',')]
    return np.array(anchors).reshape(-1, 2)


def create_model(input_shape, anchors, num_classes, load_pretrained=True, freeze_body=2,
            weights_path='model_data/yolo_weights.h5'):
    '''create the training model'''
    K.clear_session() # get a new session
    image_input = Input(shape=(None, None, 3))
    h, w = input_shape
    num_anchors = len(anchors)

    y_true = [Input(shape=(h//{0:32, 1:16, 2:8}[l], w//{0:32, 1:16, 2:8}[l], \
        num_anchors//3, num_classes+5)) for l in range(3)]

    model_body = yolo_body(image_input, num_anchors//3, num_classes)
    print('Create YOLOv3 model with {} anchors and {} classes.'.format(num_anchors,
    ↪  num_classes))

    if load_pretrained:
        model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
        print('Load weights {}.'.format(weights_path))
        if freeze_body in [1, 2]:
            # Freeze darknet53 body or freeze all but 3 output layers.
            num = (185, len(model_body.layers)-3)[freeze_body-1]
            for i in range(num): model_body.layers[i].trainable = False
            print('Freeze the first {} layers of total {} layers.'.format(num,
            ↪  len(model_body.layers)))

    model_loss = Lambda(yolo_loss, output_shape=(1,), name='yolo_loss',
        arguments={'anchors': anchors, 'num_classes': num_classes, 'ignore_thresh': 0.5})(
        [*model_body.output, *y_true])
    model = Model([model_body.input, *y_true], model_loss)
```

```
215
216     return model
217
218 def create_tiny_model(input_shape, anchors, num_classes, load_pretrained=True,
    ↪  freeze_body=2,
219             weights_path='model_data/tiny_yolo_weights.h5'):
220     '''create the training model, for Tiny YOLOv3'''
221     K.clear_session() # get a new session
222     image_input = Input(shape=(None, None, 3))
223     h, w = input_shape
224     num_anchors = len(anchors)
225
226     y_true = [Input(shape=(h//{0:32, 1:16}[l], w//{0:32, 1:16}[l], \
227         num_anchors//2, num_classes+5)) for l in range(2)]
228
229     model_body = tiny_yolo_body(image_input, num_anchors//2, num_classes)
230     print('Create Tiny YOLOv3 model with {} anchors and {} classes.'.format(num_anchors,
    ↪  num_classes))
231
232     if load_pretrained:
233         model_body.load_weights(weights_path, by_name=True, skip_mismatch=True)
234         print('Load weights {}.'.format(weights_path))
235         if freeze_body in [1, 2]:
236             # Freeze the darknet body or freeze all but 2 output layers.
237             num = (20, len(model_body.layers)-2)[freeze_body-1]
238             for i in range(num): model_body.layers[i].trainable = False
239             print('Freeze the first {} layers of total {} layers.'.format(num,
                ↪  len(model_body.layers)))
240
241     model_loss = Lambda(yolo_loss, output_shape=(1,), name='yolo_loss',
242         arguments={'anchors': anchors, 'num_classes': num_classes, 'ignore_thresh': 0.7})(
243         [*model_body.output, *y_true])
244     model = Model([model_body.input, *y_true], model_loss)
245
246     return model
247
248 def data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes):
249     '''data generator for fit_generator'''
250     n = len(annotation_lines)
251     i = 0
252     while True:
253         image_data = []
254         box_data = []
255         for b in range(batch_size):
256             if i==0:
257                 np.random.shuffle(annotation_lines)
258             image, box = get_random_data(annotation_lines[i], input_shape, random=True)
259             image_data.append(image)
260             box_data.append(box)
261             i = (i+1) % n
262         image_data = np.array(image_data)
263         box_data = np.array(box_data)
264         y_true = preprocess_true_boxes(box_data, input_shape, anchors, num_classes)
265         yield [image_data, *y_true], np.zeros(batch_size)
266
267 def data_generator_wrapper(annotation_lines, batch_size, input_shape, anchors, num_classes):
```

```
268     n = len(annotation_lines)
269     if n==0 or batch_size<=0: return None
270     return data_generator(annotation_lines, batch_size, input_shape, anchors, num_classes)
271
272 if __name__ == '__main__':
273     _main()
```

ANEXO B – CÓDIGO DE TREINAMENTO *FASTER* R-CNN

```
from __future__ import division
import random
import pprint
import sys
import time
import numpy as np

import pickle

from keras import backend as K
from keras.optimizers import Adam
from keras.layers import Input
from keras.models import Model
from keras_frcnn import config, data_generators
from keras_frcnn import losses as losses
import keras_frcnn.roi_helpers as roi_helpers
from keras.utils import generic_utils

sys.setrecursionlimit(40000)



from keras_frcnn.simple_parser import get_data

# pass the settings from the command line, and persist them in the config object
C = config.Config()

C.use_horizontal_flips = False
C.use_vertical_flips = False
C.rot_90 = False

C.model_path = './model_frcnn.hdf5'
C.num_rois = 32



from keras_frcnn import resnet as nn
C.network = 'resnet50'


# check if weight path was passed via command line

C.base_net_weights = './model_frcnn.hdf5'


all_imgs, classes_count, class_mapping = get_data("annotate.txt")

if 'bg' not in classes_count:
        classes_count['bg'] = 0
        class_mapping['bg'] = len(class_mapping)

```

```
52  C.class_mapping = class_mapping
53
54  inv_map = {v: k for k, v in class_mapping.items()}
55
56  print('Training images per class:')
57  pprint.pprint(classes_count)
58  print('Num classes (including bg) = {}'.format(len(classes_count)))
59
60  config_output_filename = "config.pickle"
61
62  with open(config_output_filename, 'wb') as config_f:
63          pickle.dump(C,config_f)
64          print('Config has been written to {}, and can be loaded when testing to ensure
            ↪  correct results'.format(config_output_filename))
65
66  random.shuffle(all_imgs)
67
68  num_imgs = len(all_imgs)
69
70  train_imgs = [s for s in all_imgs if s['imageset'] == 'trainval']
71  val_imgs = [s for s in all_imgs if s['imageset'] == 'test']
72
73  print('Num train samples {}'.format(len(train_imgs)))
74  print('Num val samples {}'.format(len(val_imgs)))
75
76
77  data_gen_train = data_generators.get_anchor_gt(train_imgs, classes_count, C,
    ↪  nn.get_img_output_length, K.image_dim_ordering(), mode='train')
78  data_gen_val = data_generators.get_anchor_gt(val_imgs, classes_count, C,
    ↪  nn.get_img_output_length,K.image_dim_ordering(), mode='val')
79
80  if K.image_dim_ordering() == 'th':
81          input_shape_img = (3, None, None)
82  else:
83          input_shape_img = (None, None, 3)
84
85  img_input = Input(shape=input_shape_img)
86  roi_input = Input(shape=(None, 4))
87
88  # define the base network (resnet here, can be VGG, Inception, etc)
89  shared_layers = nn.nn_base(img_input, trainable=True)
90
91  # define the RPN, built on the base layers
92  num_anchors = len(C.anchor_box_scales) * len(C.anchor_box_ratios)
93  rpn = nn.rpn(shared_layers, num_anchors)
94
95  classifier = nn.classifier(shared_layers, roi_input, C.num_rois,
    ↪  nb_classes=len(classes_count), trainable=True)
96
97  model_rpn = Model(img_input, rpn[:2])
98  model_classifier = Model([img_input, roi_input], classifier)
99
100 # this is a model that holds both the RPN and the classifier, used to load/save weights for
    ↪  the models
101 model_all = Model([img_input, roi_input], rpn[:2] + classifier)
102
```

```
103 try:
104         print('loading weights from {}'.format(C.base_net_weights))
105         model_rpn.load_weights(C.base_net_weights, by_name=True)
106         model_classifier.load_weights(C.base_net_weights, by_name=True)
107 except:
108         print('Could not load pretrained model weights. Weights can be found in the keras
        ↪  application folder \
109                 https://github.com/fchollet/keras/tree/master/keras/applications')
110
111 optimizer = Adam(lr=1e-5)
112 optimizer_classifier = Adam(lr=1e-5)
113 model_rpn.compile(optimizer=optimizer, loss=[losses.rpn_loss_cls(num_anchors),
    ↪  losses.rpn_loss_regr(num_anchors)])
114 model_classifier.compile(optimizer=optimizer_classifier, loss=[losses.class_loss_cls,
    ↪  losses.class_loss_regr(len(classes_count)-1)],
    ↪  metrics={'dense_class_{}'.format(len(classes_count)): 'accuracy'})
115 model_all.compile(optimizer='sgd', loss='mae')
116
117 epoch_length = 100
118 num_epochs = 50
119 iter_num = 0
120
121 losses = np.zeros((epoch_length, 5))
122 rpn_accuracy_rpn_monitor = []
123 rpn_accuracy_for_epoch = []
124 start_time = time.time()
125
126 best_loss = np.Inf
127
128 class_mapping_inv = {v: k for k, v in class_mapping.items()}
129 print('Starting training')
130
131 vis = True
132
133 for epoch_num in range(num_epochs):
134
135         progbar = generic_utils.Progbar(epoch_length)
136         print('Epoch {}/{}'.format(epoch_num + 1, num_epochs))
137
138         while True:
139                 try:
140
141                         if len(rpn_accuracy_rpn_monitor) == epoch_length and C.verbose:
142                                 mean_overlapping_bboxes =
                                ↪  float(sum(rpn_accuracy_rpn_monitor))/len(rpn_accuracy_rpn_monitor)
143                                 rpn_accuracy_rpn_monitor = []
144                                 print('Average number of overlapping bounding boxes from RPN
                                ↪  = {} for {} previous
                                ↪  iterations'.format(mean_overlapping_bboxes,
                                ↪  epoch_length))
145                                 if mean_overlapping_bboxes == 0:
146                                         print('RPN is not producing bounding boxes that
                                        ↪  overlap the ground truth boxes. Check RPN
                                        ↪  settings or keep training.')
147
148                         X, Y, img_data = next(data_gen_train)
```

```
149
150                     loss_rpn = model_rpn.train_on_batch(X, Y)
151
152                     P_rpn = model_rpn.predict_on_batch(X)
153
154                     R = roi_helpers.rpn_to_roi(P_rpn[0], P_rpn[1], C,
                        ↪  K.image_dim_ordering(), use_regr=True, overlap_thresh=0.7,
                        ↪  max_boxes=300)
155                     # note: calc_iou converts from (x1,y1,x2,y2) to (x,y,w,h) format
156                     X2, Y1, Y2, IouS = roi_helpers.calc_iou(R, img_data, C,
                        ↪  class_mapping)
157
158                     if X2 is None:
159                             rpn_accuracy_rpn_monitor.append(0)
160                             rpn_accuracy_for_epoch.append(0)
161                             continue
162
163                     neg_samples = np.where(Y1[0, :, -1] == 1)
164                     pos_samples = np.where(Y1[0, :, -1] == 0)
165
166                     if len(neg_samples) > 0:
167                             neg_samples = neg_samples[0]
168                     else:
169                             neg_samples = []
170
171                     if len(pos_samples) > 0:
172                             pos_samples = pos_samples[0]
173                     else:
174                             pos_samples = []
175
176                     rpn_accuracy_rpn_monitor.append(len(pos_samples))
177                     rpn_accuracy_for_epoch.append((len(pos_samples)))
178
179                     if C.num_rois > 1:
180                             if len(pos_samples) < C.num_rois//2:
181                                     selected_pos_samples = pos_samples.tolist()
182                             else:
183                                     selected_pos_samples = np.random.choice(pos_samples,
                                        ↪  C.num_rois//2, replace=False).tolist()
184                             try:
185                                     selected_neg_samples = np.random.choice(neg_samples,
                                        ↪  C.num_rois - len(selected_pos_samples),
                                        ↪  replace=False).tolist()
186                             except:
187                                     selected_neg_samples = np.random.choice(neg_samples,
                                        ↪  C.num_rois - len(selected_pos_samples),
                                        ↪  replace=True).tolist()
188
189                             sel_samples = selected_pos_samples + selected_neg_samples
190                     else:
191                             # in the extreme case where num_rois = 1, we pick a random
                                ↪  pos or neg sample
192                             selected_pos_samples = pos_samples.tolist()
193                             selected_neg_samples = neg_samples.tolist()
194                             if np.random.randint(0, 2):
195                                     sel_samples = random.choice(neg_samples)
```

```
196                             else:
197                                     sel_samples = random.choice(pos_samples)
198
199                     loss_class = model_classifier.train_on_batch([X, X2[:, sel_samples,
                        :]], [Y1[:, sel_samples, :], Y2[:, sel_samples, :]])
200
201                     losses[iter_num, 0] = loss_rpn[1]
202                     losses[iter_num, 1] = loss_rpn[2]
203
204                     losses[iter_num, 2] = loss_class[1]
205                     losses[iter_num, 3] = loss_class[2]
206                     losses[iter_num, 4] = loss_class[3]
207
208                     progbar.update(iter_num+1, [('rpn_cls', losses[iter_num, 0]),
                        ('rpn_regr', losses[iter_num, 1]),
209                                                 ('detector_cls',
                                                    losses[iter_num,
                                                    2]),
                                                 ('detector_regr',
                                                    losses[iter_num,
                                                    3])])
210
211                     iter_num += 1
212
213                     if iter_num == epoch_length:
214                             loss_rpn_cls = np.mean(losses[:, 0])
215                             loss_rpn_regr = np.mean(losses[:, 1])
216                             loss_class_cls = np.mean(losses[:, 2])
217                             loss_class_regr = np.mean(losses[:, 3])
218                             class_acc = np.mean(losses[:, 4])
219
220                             mean_overlapping_bboxes = float(sum(rpn_accuracy_for_epoch))
                                / len(rpn_accuracy_for_epoch)
221                             rpn_accuracy_for_epoch = []
222
223                             if C.verbose:
224                                     print('Mean number of bounding boxes from RPN
                                        overlapping ground truth boxes:
                                        {}'.format(mean_overlapping_bboxes))
225                                     print('Classifier accuracy for bounding boxes from
                                        RPN: {}'.format(class_acc))
226                                     print('Loss RPN classifier:
                                        {}'.format(loss_rpn_cls))
227                                     print('Loss RPN regression:
                                        {}'.format(loss_rpn_regr))
228                                     print('Loss Detector classifier:
                                        {}'.format(loss_class_cls))
229                                     print('Loss Detector regression:
                                        {}'.format(loss_class_regr))
230                                     print('Elapsed time: {}'.format(time.time() -
                                        start_time))
231
232                             curr_loss = loss_rpn_cls + loss_rpn_regr + loss_class_cls +
                                loss_class_regr
233                             iter_num = 0
234                             start_time = time.time()
```

```
235
236                                 if curr_loss < best_loss:
237                                         if C.verbose:
238                                                 print('Total loss decreased from {} to {},
                                                    ↪  saving
                                                    ↪  weights'.format(best_loss,curr_loss))
239                                         best_loss = curr_loss
240                                         model_all.save_weights(C.model_path)
241
242                                 break
243
244                 except Exception as e:
245                         print('Exception: {}'.format(e))
246                         continue
247
248 print('Training complete, exiting.')
```

ANEXO C – CÓDIGO DE TREINAMENTO SSD

```
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TerminateOnNaN,
↪  CSVLogger
from keras import backend as K

from math import ceil


from models.keras_ssd300 import ssd_300
from keras_loss_function.keras_ssd_loss import SSDLoss



from ssd_encoder_decoder.ssd_input_encoder import SSDInputEncoder


from data_generator.object_detection_2d_data_generator import DataGenerator
from data_generator.object_detection_2d_geometric_ops import Resize
from data_generator.object_detection_2d_photometric_ops import ConvertTo3Channels
from data_generator.data_augmentation_chain_original_ssd import SSDDataAugmentation




img_height = 300 # Height of the model input images
img_width = 300 # Width of the model input images
img_channels = 3 # Number of color channels of the model input images
mean_color = [123, 117, 104] # The per-channel mean of the images in the dataset. Do not
↪  change this value if you're using any of the pre-trained weights.
swap_channels = [2, 1, 0] # The color channel order in the original SSD is BGR, so we'll
↪  have the model reverse the color channel order of the input images.
n_classes = 1 # Number of positive classes, e.g. 20 for Pascal VOC, 80 for MS COCO
scales_pascal = [0.1, 0.2, 0.37, 0.54, 0.71, 0.88, 1.05] # The anchor box scaling factors
↪  used in the original SSD300 for the Pascal VOC datasets
scales_coco = [0.07, 0.15, 0.33, 0.51, 0.69, 0.87, 1.05] # The anchor box scaling factors
↪  used in the original SSD300 for the MS COCO datasets
scales = scales_pascal
aspect_ratios = [[1.0, 2.0, 0.5],
                 [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
                 [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
                 [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
                 [1.0, 2.0, 0.5],
                 [1.0, 2.0, 0.5]] # The anchor box aspect ratios used in the original
                 ↪  SSD300; the order matters
two_boxes_for_ar1 = True
steps = [8, 16, 32, 64, 100, 300] # The space between two adjacent anchor box center points
↪  for each predictor layer.
offsets = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5] # The offsets of the first anchor box center points
↪  from the top and left borders of the image as a fraction of the step size for each
↪  predictor layer.
```

```
clip_boxes = False # Whether or not to clip the anchor boxes to lie entirely within the image boundaries
variances = [0.1, 0.1, 0.2, 0.2] # The variances by which the encoded target coordinates are divided as in the original implementation
normalize_coords = True









K.clear_session()

model = ssd_300(image_size=(img_height, img_width, img_channels),
                n_classes=n_classes,
                mode='training',
                l2_regularization=0.0005,
                scales=scales,
                aspect_ratios_per_layer=aspect_ratios,
                two_boxes_for_ar1=two_boxes_for_ar1,
                steps=steps,
                offsets=offsets,
                clip_boxes=clip_boxes,
                variances=variances,
                normalize_coords=normalize_coords,
                subtract_mean=mean_color,
                swap_channels=swap_channels)

weights_path = 'ssd300_pascal_07+12_epoch-16_loss-2.0669_val_loss-2.5514.h5'

model.load_weights(weights_path, by_name=True)

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)

ssd_loss = SSDLoss(neg_pos_ratio=3, alpha=1.0)

model.compile(optimizer=adam, loss=ssd_loss.compute_loss)




train_dataset = DataGenerator(load_images_into_memory=False, hdf5_dataset_path=None)
val_dataset = DataGenerator(load_images_into_memory=False, hdf5_dataset_path=None)




images_dir = 'Data\JPEGImages'

train_labels_filename = 'Train_csv.csv'
val_labels_filename   = 'val_csv.csv'

train_dataset.parse_csv(images_dir=images_dir,
                        labels_filename=train_labels_filename,
```

```
                          input_format=['image_name', 'xmin', 'ymin', 'xmax', 'ymax',
                          ↪  'class_id'], # This is the order of the first six columns in the
                          ↪  CSV file that contains the labels for your dataset. If your
                          ↪  labels are in XML format, maybe the XML parser will be helpful,
                          ↪  check the documentation.
                          include_classes='all')

val_dataset.parse_csv(images_dir=images_dir,
                      labels_filename=val_labels_filename,
                      input_format=['image_name', 'xmin', 'ymin', 'xmax', 'ymax',
                      ↪  'class_id'],
                      include_classes='all')

train_dataset.create_hdf5_dataset(file_path='dataset_pascal_voc_07+12_trainval_1.h5',
                                  resize=False,
                                  variable_image_size=True,
                                  verbose=True)

val_dataset.create_hdf5_dataset(file_path='dataset_pascal_voc_07_test_1.h5',
                                resize=False,
                                variable_image_size=True,
                                verbose=True)









batch_size = 16









ssd_data_augmentation = SSDDataAugmentation(img_height=img_height,
                                            img_width=img_width,
                                            background=mean_color)


convert_to_3_channels = ConvertTo3Channels()
resize = Resize(height=img_height, width=img_width)


predictor_sizes = [model.get_layer('conv4_3_norm_mbox_conf').output_shape[1:3],
                   model.get_layer('fc7_mbox_conf').output_shape[1:3],
                   model.get_layer('conv6_2_mbox_conf').output_shape[1:3],
                   model.get_layer('conv7_2_mbox_conf').output_shape[1:3],
                   model.get_layer('conv8_2_mbox_conf').output_shape[1:3],
                   model.get_layer('conv9_2_mbox_conf').output_shape[1:3]]

ssd_input_encoder = SSDInputEncoder(img_height=img_height,
                                    img_width=img_width,
                                    n_classes=n_classes,
                                    predictor_sizes=predictor_sizes,
                                    scales=scales,
                                    aspect_ratios_per_layer=aspect_ratios,
                                    two_boxes_for_ar1=two_boxes_for_ar1,
```

```
                                    steps=steps,
                                    offsets=offsets,
                                    clip_boxes=clip_boxes,
                                    variances=variances,
                                    matching_type='multi',
                                    pos_iou_threshold=0.5,
                                    neg_iou_limit=0.5,
                                    normalize_coords=normalize_coords)


train_generator = train_dataset.generate(batch_size=batch_size,
                                         shuffle=True,
                                         transformations=[ssd_data_augmentation],
                                         label_encoder=ssd_input_encoder,
                                         returns={'processed_images',
                                                  'encoded_labels'},
                                         keep_images_without_gt=False)

val_generator = val_dataset.generate(batch_size=batch_size,
                                     shuffle=False,
                                     transformations=[convert_to_3_channels,
                                                      resize],
                                     label_encoder=ssd_input_encoder,
                                     returns={'processed_images',
                                              'encoded_labels'},
                                     keep_images_without_gt=False)


train_dataset_size = train_dataset.get_dataset_size()
val_dataset_size   = val_dataset.get_dataset_size()

print("Number of images in the training dataset:\t{:>6}".format(train_dataset_size))
print("Number of images in the validation dataset:\t{:>6}".format(val_dataset_size))



def lr_schedule(epoch):
    if epoch < 80:
        return 0.001
    elif epoch < 100:
        return 0.0001
    else:
        return 0.00001



model_checkpoint =
↪  ModelCheckpoint(filepath='ssd300_pascal_07+12_epoch-{epoch:02d}_loss-{loss:.4f}_val_loss-{val_loss:.4f}.h5',
                                   monitor='val_loss',
                                   verbose=1,
                                   save_best_only=True,
                                   save_weights_only=False,
                                   mode='auto',
                                   period=1)

```

```
202
203 csv_logger = CSVLogger(filename='ssd300_pascal_07+12_training_log.csv',
204                         separator=',',
205                         append=True)
206
207 learning_rate_scheduler = LearningRateScheduler(schedule=lr_schedule,
208                                                 verbose=1)
209
210 terminate_on_nan = TerminateOnNaN()
211
212 callbacks = [model_checkpoint,
213              csv_logger,
214              learning_rate_scheduler,
215              terminate_on_nan]
216
217
218
219 initial_epoch   = 25
220 final_epoch     = 100
221 steps_per_epoch = 200
222
223 history = model.fit_generator(generator=train_generator,
224                               steps_per_epoch=steps_per_epoch,
225                               epochs=final_epoch,
226                               callbacks=callbacks,
227                               validation_data=val_generator,
228                               validation_steps=ceil(val_dataset_size/batch_size),
229                               initial_epoch=initial_epoch)
```

ANEXO D – CÓDIGO DE DETECÇÃO YOLOV3

```
import os



def detect_object(yolo, img, save_img=True, save_img_path="output"):
    try:
        image = Image.open(img)
        if image.mode != "RGB":
            image = image.convert("RGB")
        image_array = np.array(image)
    except:
        print("File Open Error! Try again!")
        return None, None

    prediction, r_image = yolo.detect_image(image)

    if save_img:
        r_image.save(os.path.join(save_img_path, os.path.basename(img)))

    return prediction, image_array

def get_parent_dir(n=1):
    #retorna o caminho para o diretório de trabalho
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


def GetFileList(dirName, endings=[".jpg", ".jpeg", ".png", ".mp4"]):
    # cria uma lista de todos os arquivos no diretório
    listOfFile = os.listdir(dirName)
    allFiles = list()


    # garante que todos os finais começam com .

    for i, ending in enumerate(endings):
        if ending[0] != ".":
            endings[i] = "." + ending


    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + GetFileList(fullPath, endings)
        else:
            for ending in endings:
                if entry.endswith(ending):
```

```
                    allFiles.append(fullPath)
    return allFiles

from utils.yolo import YOLO, detect_video
from PIL import Image
from timeit import default_timer as timer

import pandas as pd
import numpy as np


os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

# Diretórios de trabalho
data_folder = os.path.join(get_parent_dir(0), "Data")

image_folder = os.path.join(data_folder, "Source_Images")

input_path = os.path.join(image_folder, "Test_Images")

output_path = os.path.join(image_folder, "Test_Image_Detection_Results")
detection_results_file = os.path.join(output_path, "Detection_Results.csv")

model_folder = os.path.join(data_folder, "Model_Weights")

model_weights = os.path.join(model_folder, "trained_weights_final.h5")
model_classes = os.path.join(model_folder, "data_classes.txt")

anchors_path = os.path.join( "utils", "model_data", "yolo_anchors.txt")



save_img = True

#grau de confiança
score = 0.5

input_paths = GetFileList(input_path)



# separa imagens de videos
img_endings = (".jpg", ".jpeg", ".png", ".JPG", ".JPEG")
vid_endings = (".mp4", ".mpeg", ".mpg", ".avi")

input_image_paths = []
input_video_paths = []
for item in input_paths:
        if item.endswith(img_endings):
                input_image_paths.append(item)
        elif item.endswith(vid_endings):
                input_video_paths.append(item)

if not os.path.exists(output_path):
        os.makedirs(output_path)

```

```
108
109 yolo = YOLO(
110         **{
111                 "model_path": model_weights,
112                 "anchors_path": anchors_path,
113                 "classes_path": model_classes,
114                 "score": score,
115                 "gpu_num": 1,
116                 "model_image_size": (416, 416),
117         }
118 )
119
120 # Cria um dataframe para os resultados
121 out_df = pd.DataFrame(
122         columns=[
123                 "image",
124                 "image_path",
125                 "xmin",
126                 "ymin",
127                 "xmax",
128                 "ymax",
129                 "label",
130                 "confidence",
131                 "x_size",
132                 "y_size",
133         ]
134 )
135
136
137 class_file = open(model_classes, "r")
138 input_labels = [line.rstrip("\n") for line in class_file.readlines()]
139 print("Found {} input labels: {} ...".format(len(input_labels), input_labels))
140
141 if input_image_paths:
142         print(
143                 "Found {} input images: {} ...".format(
144                         len(input_image_paths),
145                         [os.path.basename(f) for f in input_image_paths[:5]],
146                 )
147         )
148         start = timer()
149         text_out = ""
150
151
152         for i, img_path in enumerate(input_image_paths):
153                 print(img_path)
154                 prediction, image = detect_object(
155                         yolo,
156                         img_path,
157                         save_img=save_img,
158                         save_img_path=output_path,
159                 )
160                 y_size, x_size, _ = np.array(image).shape
161                 for single_prediction in prediction:
162                         out_df = out_df.append(
163                                 pd.DataFrame(
```

```
                                        [
                                                [
                                                        os.path.basename(img_path.rstrip("\n")),
                                                        img_path.rstrip("\n"),
                                                ]
                                                + single_prediction
                                                + [x_size, y_size]
                                        ],
                                        columns=[
                                                "image",
                                                "image_path",
                                                "xmin",
                                                "ymin",
                                                "xmax",
                                                "ymax",
                                                "label",
                                                "confidence",
                                                "x_size",
                                                "y_size",
                                        ],
                                )
                        )
        end = timer()
        print(
                "Processed {} images in {:.1f}sec - {:.1f}FPS".format(
                        len(input_image_paths),
                        end - start,
                        len(input_image_paths) / (end - start),
                )
        )
        out_df.to_csv(detection_results_file, index=False)


if input_video_paths:
        print(
                "Found {} input videos: {} ...".format(
                        len(input_video_paths),
                        [os.path.basename(f) for f in input_video_paths[:5]],
                )
        )
        start = timer()
        for i, vid_path in enumerate(input_video_paths):
                detect_video(yolo, vid_path, output_path=output_path)
        end = timer()
        print(
                "Processed {} videos in {:.1f}sec".format(
                        len(input_video_paths), end - start
                )
        )

yolo.close_session()
```

ANEXO E – CÓDIGO DE DETECÇÃO *FASTER* R-CNN

```
from __future__ import division
import os
import cv2
import numpy as np
import sys
import pickle
from optparse import OptionParser
import time
from keras_frcnn import config
from keras import backend as K
from keras.layers import Input
from keras.models import Model
from keras_frcnn import roi_helpers


from PIL import Image, ImageFont, ImageDraw
from timeit import default_timer as timer



sys.setrecursionlimit(40000)

parser = OptionParser()

parser.add_option("-p", "--path", dest="test_path", help="Path to test data.",
↪  default="Test_Images")
parser.add_option("-n", "--num_rois", type="int", dest="num_rois",
                                help="Number of ROIs per iteration. Higher means more memory
                                ↪  use.", default=32)
parser.add_option("--config_filename", dest="config_filename", help=
                                "Location to read the metadata related to the training
                                ↪  (generated when training).",
                                default="config.pickle")
parser.add_option("--network", dest="network", help="Base network to use. Supports vgg or
↪  resnet50.", default='resnet50')

(options, args) = parser.parse_args()

if not options.test_path:   # if filename is not given
        parser.error('Error: path to test data must be specified. Pass --path to command
        ↪  line')


config_output_filename = options.config_filename

with open(config_output_filename, 'rb') as f_in:
        C = pickle.load(f_in)

if C.network == 'resnet50':
        import keras_frcnn.resnet as nn
elif C.network == 'vgg':
```

```
47          import keras_frcnn.vgg as nn
48
49  # turn off any data augmentation at test time
50  C.use_horizontal_flips = False
51  C.use_vertical_flips = False
52  C.rot_90 = False
53
54  img_path = options.test_path
55
56  def format_img_size(img, C):
57          """ formats the image size based on config """
58          img_min_side = float(C.im_size)
59          (height,width,_) = img.shape
60
61          if width <= height:
62                  ratio = img_min_side/width
63                  new_height = int(ratio * height)
64                  new_width = int(img_min_side)
65          else:
66                  ratio = img_min_side/height
67                  new_width = int(ratio * width)
68                  new_height = int(img_min_side)
69          img = cv2.resize(img, (new_width, new_height), interpolation=cv2.INTER_CUBIC)
70          return img, ratio
71
72  def format_img_channels(img, C):
73          """ formats the image channels based on config """
74          img = img[:, :, (2, 1, 0)]
75          img = img.astype(np.float32)
76          img[:, :, 0] -= C.img_channel_mean[0]
77          img[:, :, 1] -= C.img_channel_mean[1]
78          img[:, :, 2] -= C.img_channel_mean[2]
79          img /= C.img_scaling_factor
80          img = np.transpose(img, (2, 0, 1))
81          img = np.expand_dims(img, axis=0)
82          return img
83
84  def format_img(img, C):
85          """ formats an image for model prediction based on config """
86          img, ratio = format_img_size(img, C)
87          img = format_img_channels(img, C)
88          return img, ratio
89
90  # Method to transform the coordinates of the bounding box to its original size
91  def get_real_coordinates(ratio, x1, y1, x2, y2):
92
93          real_x1 = int(round(x1 // ratio))
94          real_y1 = int(round(y1 // ratio))
95          real_x2 = int(round(x2 // ratio))
96          real_y2 = int(round(y2 // ratio))
97
98          return (real_x1, real_y1, real_x2 ,real_y2)
99
100 class_mapping = C.class_mapping
101
102 if 'bg' not in class_mapping:
```

```
103         class_mapping['bg'] = len(class_mapping)
104
105 class_mapping = {v: k for k, v in class_mapping.items()}
106 print(class_mapping)
107 class_to_color = {class_mapping[v]: np.random.randint(0, 255, 3) for v in class_mapping}
108 C.num_rois = int(options.num_rois)
109
110 if C.network == 'resnet50':
111         num_features = 1024
112 elif C.network == 'vgg':
113         num_features = 512
114
115 if K.image_dim_ordering() == 'th':
116         input_shape_img = (3, None, None)
117         input_shape_features = (num_features, None, None)
118 else:
119         input_shape_img = (None, None, 3)
120         input_shape_features = (None, None, num_features)
121
122
123 img_input = Input(shape=input_shape_img)
124 roi_input = Input(shape=(C.num_rois, 4))
125 feature_map_input = Input(shape=input_shape_features)
126
127 # define the base network (resnet here, can be VGG, Inception, etc)
128 shared_layers = nn.nn_base(img_input, trainable=True)
129
130 # define the RPN, built on the base layers
131 num_anchors = len(C.anchor_box_scales) * len(C.anchor_box_ratios)
132 rpn_layers = nn.rpn(shared_layers, num_anchors)
133
134 classifier = nn.classifier(feature_map_input, roi_input, C.num_rois,
    ↪  nb_classes=len(class_mapping), trainable=True)
135
136 model_rpn = Model(img_input, rpn_layers)
137 model_classifier_only = Model([feature_map_input, roi_input], classifier)
138
139 model_classifier = Model([feature_map_input, roi_input], classifier)
140
141 print('Loading weights from {}'.format(C.model_path))
142 model_rpn.load_weights(C.model_path, by_name=True)
143 model_classifier.load_weights(C.model_path, by_name=True)
144
145 model_rpn.compile(optimizer='sgd', loss='mse')
146 model_classifier.compile(optimizer='sgd', loss='mse')
147
148 all_imgs = []
149
150 classes = {}
151
152 bbox_threshold = 0.8
153
154 visualise = True
155
156 save_path = "Result_Images"
157
```

```
start_t = timer()

for idx, img_name in enumerate(sorted(os.listdir(img_path))):
    start = timer()
    if not img_name.lower().endswith(('.bmp', '.jpeg', '.jpg', '.png', '.tif', '.tiff')):
        continue
    filepath = os.path.join(img_path,img_name)
    img = cv2.imread(filepath)
    img2 = Image.open(filepath)

    X, ratio = format_img(img, C)

    if K.image_dim_ordering() == 'tf':
        X = np.transpose(X, (0, 2, 3, 1))

        # get the feature maps and output from the RPN
    [Y1, Y2, F] = model_rpn.predict(X)


    R = roi_helpers.rpn_to_roi(Y1, Y2, C, K.image_dim_ordering(), overlap_thresh=0.7)

        # convert from (x1,y1,x2,y2) to (x,y,w,h)
    R[:, 2] -= R[:, 0]
    R[:, 3] -= R[:, 1]

        # apply the spatial pyramid pooling to the proposed regions
    bboxes = {}
    probs = {}

    for jk in range(R.shape[0]//C.num_rois + 1):
        ROIs = np.expand_dims(R[C.num_rois*jk:C.num_rois*(jk+1), :], axis=0)
        if ROIs.shape[1] == 0:
            break

        if jk == R.shape[0]//C.num_rois:
                        #pad R
            curr_shape = ROIs.shape
            target_shape = (curr_shape[0],C.num_rois,curr_shape[2])
            ROIs_padded = np.zeros(target_shape).astype(ROIs.dtype)
            ROIs_padded[:, :curr_shape[1], :] = ROIs
            ROIs_padded[0, curr_shape[1]:, :] = ROIs[0, 0, :]
            ROIs = ROIs_padded

        [P_cls, P_regr] = model_classifier_only.predict([F, ROIs])

        for ii in range(P_cls.shape[1]):

            if np.max(P_cls[0, ii, :]) < bbox_threshold or np.argmax(P_cls[0, ii, :]) ==
            ↪  (P_cls.shape[2] - 1):
                continue

            cls_name = class_mapping[np.argmax(P_cls[0, ii, :])]

            if cls_name not in bboxes:
                bboxes[cls_name] = []
                probs[cls_name] = []
```

```
213
214             (x, y, w, h) = ROIs[0, ii, :]
215
216             cls_num = np.argmax(P_cls[0, ii, :])
217             try:
218                 (tx, ty, tw, th) = P_regr[0, ii, 4*cls_num:4*(cls_num+1)]
219                 tx /= C.classifier_regr_std[0]
220                 ty /= C.classifier_regr_std[1]
221                 tw /= C.classifier_regr_std[2]
222                 th /= C.classifier_regr_std[3]
223                 x, y, w, h = roi_helpers.apply_regr(x, y, w, h, tx, ty, tw, th)
224             except:
225                 pass
226             bboxes[cls_name].append([C.rpn_stride*x, C.rpn_stride*y, C.rpn_stride*(x+w),
                ↪  C.rpn_stride*(y+h)])
227             probs[cls_name].append(np.max(P_cls[0, ii, :]))
228
229     all_dets = []
230
231
232
233     for key in bboxes:
234         bbox = np.array(bboxes[key])
235
236         new_boxes, new_probs = roi_helpers.non_max_suppression_fast(bbox,
            ↪  np.array(probs[key]), overlap_thresh=0.5)
237
238
239         total = "Caminhão com {} eixo(s)".format(len(range(new_boxes.shape[0])))
240         font_path = os.path.join(os.path.dirname(__file__), "font/FiraMono-Medium.otf")
241         font = ImageFont.truetype(
242             font=font_path, size=np.floor(3e-2 * img2.size[1] + 0.5).astype("int32")
243         )
244         thickness = (img2.size[0] + img2.size[1]) // 300
245
246         for jk in range(new_boxes.shape[0]):
247             (x1, y1, x2, y2) = new_boxes[jk,:]
248
249             (real_x1, real_y1, real_x2, real_y2) = get_real_coordinates(ratio, x1, y1, x2,
                ↪  y2)
250
251             draw = ImageDraw.Draw(img2)
252             total_size = draw.textsize(total, font)
253             label = "Eixo : {:.2f}".format(new_probs[jk])
254             label_size = draw.textsize(label, font)
255
256
257             if real_y1 - label_size[1] >= 0:
258                 text_origin = np.array([real_x1, real_y1 - label_size[1]])
259             else:
260                 text_origin = np.array([real_x1, real_y2])
261
262             for i in range(thickness):
263                 draw.rectangle(
264                     [real_x1 + i, real_y1 + i, real_x2 - i,real_y2 - i], outline="#FFA500"
265                 )
```

```
266             draw.rectangle(
267                 [tuple(text_origin), tuple(text_origin + label_size)],
268                 fill="#FFA500",
269             )
270 
271             all_dets.append((key,100*new_probs[jk]))
272 
273 
274 
275             draw.rectangle(
276                 [(0,0), tuple(total_size)],
277                 fill="#FFA500",
278             )
279             draw.text(text_origin, label, fill=(0, 0, 0), font=font)
280             draw.text((0,0), total, fill=(0, 0, 0), font=font)
281             del draw
282 
283 
284 
285             txt = os.path.join("txt", os.path.splitext(img_name)[0])
286             file_name = "{}.txt".format(txt)
287 
288             line = "Eixo " + " " + str(new_probs[jk]) + " " + str(int(round(real_x1))) + " "
            ↪  + str(int(round(real_y1))) + " " + str(int(round(real_x2))) + " " +
            ↪  str(int(round(real_y2))) + '\n'
289             with open(file_name, 'a') as output:
290                 output.write(line)
291 
292 
293 
294     end = timer()
295     print("Time spent: {:.3f}sec".format(end - start))
296     img2.save(os.path.join(save_path, os.path.basename(img_name)))
297     print(all_dets)
298 
299 
300 end_t = timer()
301 
302 
303 print(
304                 "Processed {} images in {:.1f}sec - {:.1f}FPS".format(
305                         len(os.listdir(img_path)),
306                         end_t - start_t,
307                         len(os.listdir(img_path)) / (end_t - start_t),
308 )
309 )
```

ANEXO F – CÓDIGO DE DETECÇÃO SSD

```
import os
from PIL import Image, ImageFont, ImageDraw
from timeit import default_timer as timer



def get_parent_dir(n=1):
    #retorna o caminho para o diretório de trabalho
    current_path = os.path.dirname(os.path.abspath(__file__))
    for k in range(n):
        current_path = os.path.dirname(current_path)
    return current_path


def GetFileList(dirName, endings=[".jpg", ".jpeg", ".png", ".mp4"]):
    # cria uma lista de todos os arquivos no diretório
    listOfFile = os.listdir(dirName)
    allFiles = list()


    # garante que todos os finais começam com .

    for i, ending in enumerate(endings):
        if ending[0] != ".":
            endings[i] = "." + ending


    for entry in listOfFile:
        # Create full path
        fullPath = os.path.join(dirName, entry)
        # If entry is a directory then get the list of files in this directory
        if os.path.isdir(fullPath):
            allFiles = allFiles + GetFileList(fullPath, endings)
        else:
            for ending in endings:
                if entry.endswith(ending):
                    allFiles.append(fullPath)
    return allFiles


from keras import backend as K

from keras.preprocessing import image
from keras.optimizers import Adam
from imageio import imread
import numpy as np


from models.keras_ssd300 import ssd_300
from keras_loss_function.keras_ssd_loss import SSDLoss

```

```
52
53
54
55
56  img_height = 300
57  img_width = 300
58
59
60
61  K.clear_session()
62
63  model = ssd_300(image_size=(img_height, img_width, 3),
64                  n_classes=1,
65                  mode='inference',
66                  l2_regularization=0.0005,
67                  scales=[0.1, 0.2, 0.37, 0.54, 0.71, 0.88, 1.05],
68                  aspect_ratios_per_layer=[[1.0, 2.0, 0.5],
69                                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
70                                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
71                                           [1.0, 2.0, 0.5, 3.0, 1.0/3.0],
72                                           [1.0, 2.0, 0.5],
73                                           [1.0, 2.0, 0.5]],
74                  two_boxes_for_ar1=True,
75                  steps=[8, 16, 32, 64, 100, 300],
76                  offsets=[0.5, 0.5, 0.5, 0.5, 0.5, 0.5],
77                  clip_boxes=False,
78                  variances=[0.1, 0.1, 0.2, 0.2],
79                  normalize_coords=True,
80                  subtract_mean=[123, 117, 104],
81                  swap_channels=[2, 1, 0],
82                  confidence_thresh=0.5,
83                  iou_threshold=0.45,
84                  top_k=200,
85                  nms_max_output_size=400)
86
87
88  weights_path = 'ssd300_pascal_07+12_epoch-29_loss-1.7706_val_loss-2.0828.h5'
89
90  model.load_weights(weights_path, by_name=False)
91
92
93  adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)
94
95  ssd_loss = SSDLoss(neg_pos_ratio=3, alpha=1.0)
96
97  model.compile(optimizer=adam, loss=ssd_loss.compute_loss)
98
99
100
101
102
103
104 input_images = []
105 orig_images = []
106
107
```

```
data_folder = os.path.join(get_parent_dir(0), "Data")

input_path = os.path.join(data_folder, "Test_Images")
save_path = os.path.join(data_folder, "Result_Images")
txt_path = os.path.join(data_folder, "TXT")


input_paths = GetFileList(input_path)


start_t = timer()

for item in input_paths:
    start = timer()

    input_images = []
    orig_images = []

    orig_images.append(imread(item))
    img = image.load_img(item, target_size=(img_height, img_width))
    img = image.img_to_array(img)
    input_images.append(img)
    input_images = np.array(input_images)









    y_pred = model.predict(input_images)

    confidence_threshold = 0.5

    y_pred_thresh = [y_pred[k][y_pred[k,:,1] > confidence_threshold] for k in
    ↪  range(y_pred.shape[0])]

    y_pred_txt = y_pred_thresh[0]


    image2 = Image.open(item)
    width, height = image2.size

    ratio_y = height/img_height
    ratio_x = width/img_width
    a=0

    txt = os.path.join(txt_path, os.path.splitext(os.path.basename(item))[0])
    file_name = "{}.txt".format(txt)
    for row in y_pred_txt:
        a = a+1
        line = "Eixo " + " " + str(row[1]) + " " + str(int(round(row[2]*ratio_x))) + " " +
        ↪  str(int(round(row[3]*ratio_y))) + " " + str(int(round(row[4]*ratio_x))) + " " +
        ↪  str(int(round(row[5]*ratio_y))) + '\n'
        with open(file_name, 'a') as output:
            output.write(line)

    font_path = os.path.join(os.path.dirname(__file__), "font/FiraMono-Medium.otf")
```

```
161     font = ImageFont.truetype(
162             font=font_path, size=np.floor(3e-2 * image2.size[1] + 0.5).astype("int32")
163         )
164     thickness = (image2.size[0] + image2.size[1]) // 300
165 
166     np.set_printoptions(precision=2, suppress=True, linewidth=90)
167     print("Predicted boxes:\n")
168     print('   class   conf xmin   ymin   xmax   ymax')
169     print(y_pred_thresh[0])
170 
171     total = "Caminhão com {} eixo(s)".format(a)
172 
173     for box in y_pred_thresh[0]:
174             draw = ImageDraw.Draw(image2)
175             total_size = draw.textsize(total, font)
176             label = "Eixo : {:.2f}".format(box[1])
177             label_size = draw.textsize(label, font)
178 
179             if box[3]*ratio_y - label_size[1] >= 0:
180                 text_origin = np.array([box[2]*ratio_x, box[3]*ratio_y - label_size[1]])
181             else:
182                 text_origin = np.array([box[2]*ratio_x, box[5]*ratio_y])
183 
184 
185             for i in range(thickness):
186                 draw.rectangle(
187                     [box[2]*ratio_x + i, box[3]*ratio_y + i, box[4]*ratio_x - i,
                        ↪  box[5]*ratio_y - i], outline="#800080"
188                 )
189             draw.rectangle(
190                 [tuple(text_origin), tuple(text_origin + label_size)],
191                 fill="#800080",
192             )
193 
194             draw.rectangle(
195                 [(0,0), tuple(total_size)],
196                 fill="#800080",
197             )
198             draw.text(text_origin, label, fill=(0, 0, 0), font=font)
199             draw.text((0,0), total, fill=(0, 0, 0), font=font)
200             del draw
201 
202 
203     end = timer()
204     print("Time spent: {:.3f}sec".format(end - start))
205 
206 
207     image2.save(os.path.join(save_path, os.path.basename(item)))
208 
209 end_t = timer()
210 
211 
212 print(
213                 "Processed {} images in {:.1f}sec - {:.1f}FPS".format(
214                         len(input_paths),
215                         end_t - start_t,
```

```
                        len(input_paths) / (end_t - start_t),
    )
    )
```

ANEXO G – DETECÇÕES

Todas as detecções podem ser encontradas em:

-YOLOv3

Detecções

-Faster R-CNN

Detecções

-SSD

Detecções

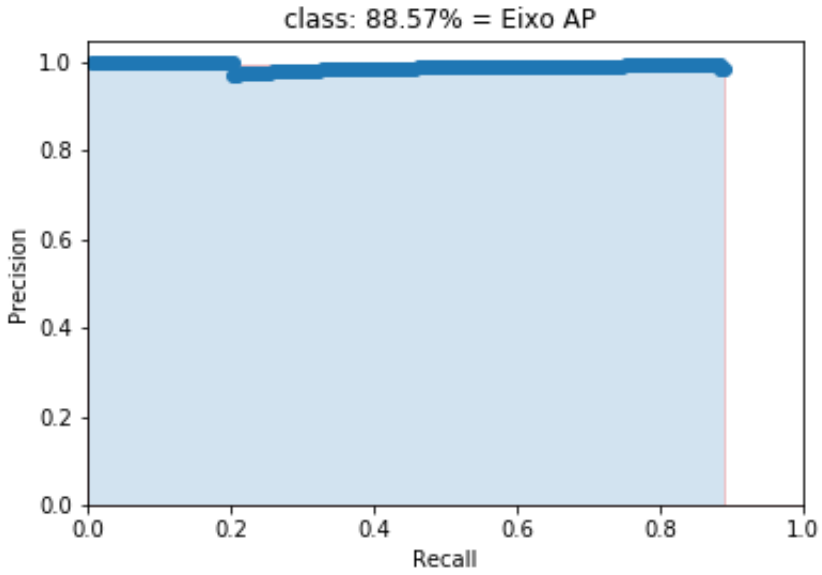
ANEXO H – RESULTADOS AP

Os resultados dos testes de AP podem ser encontrados em:

Resultados

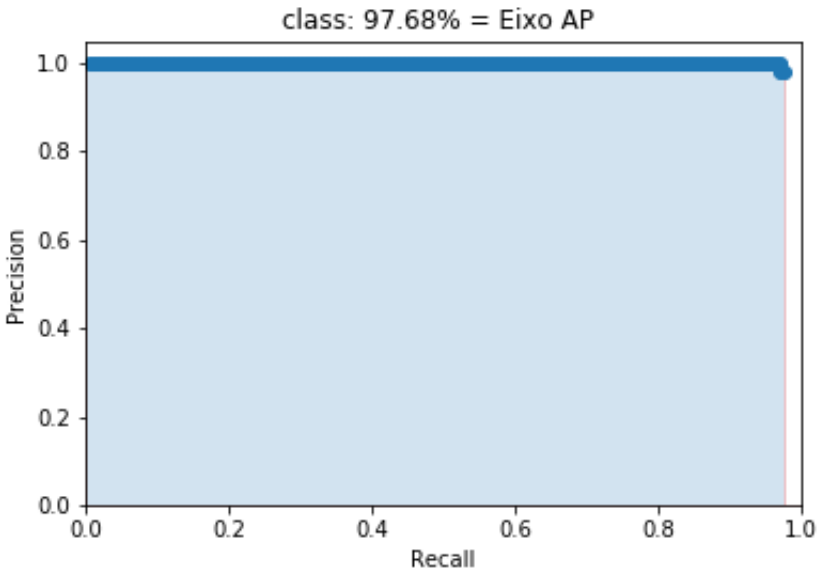
As figuras 31, 32 e 33 mostram as curvas de *Precision x Recall* de cada arquitetura.

Figura 31 – Gráfico *Precision x Recall para YOLOv3*



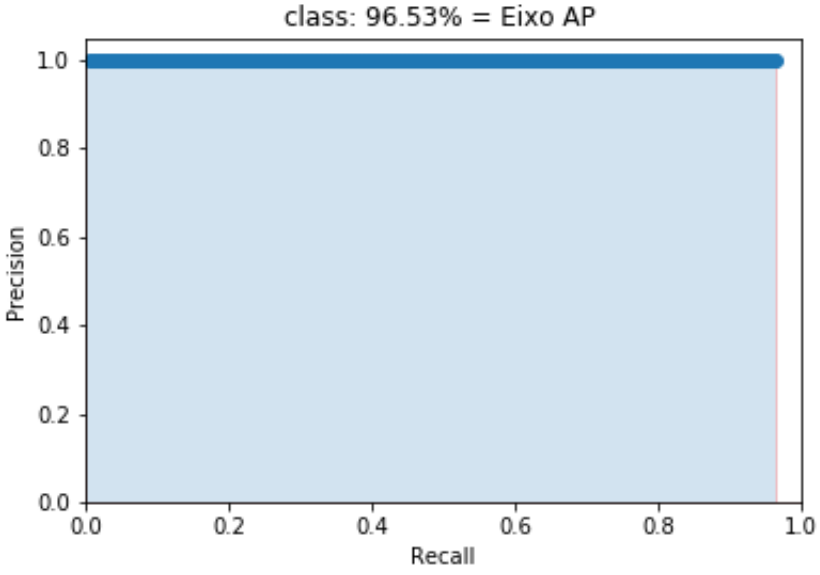
Fonte: Elaborado pelo autor.

Figura 32 – Gráfico *Precision x Recall* para *Faster* R-CNN



Fonte: Elaborado pelo autor.

Figura 33 – Gráfico *Precision x Recall* para SSD



Fonte: Elaborado pelo autor.