

**UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"**  
FACULDADE DE CIÊNCIAS - CAMPUS BAURU  
DEPARTAMENTO DE COMPUTAÇÃO  
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

VINICIUS DIAS DE VASCONCELOS

**PROCESSAMENTO DE IMAGENS PARA CONTROLE DE  
CONTEÚDO EM SISTEMAS WEB**

BAURU  
Dezembro/2020

VINICIUS DIAS DE VASCONCELOS

**PROCESSAMENTO DE IMAGENS PARA CONTROLE DE  
CONTEÚDO EM SISTEMAS WEB**

Trabalho de Conclusão de Curso do Curso  
de Bacharelado em Ciência da Computação  
da Universidade Estadual Paulista “Júlio  
de Mesquita Filho”, Faculdade de Ciências,  
Campus Bauru.

Orientador: Prof. Associado Aparecido Nilceu  
Marana

BAURU  
Dezembro/2020

Vinicio Dias de Vasconcelos    PROCESSAMENTO DE IMAGENS PARA  
CONTROLE DE CONTEÚDO EM SISTEMAS WEB/ Vinicius Dias de  
Vasconcelos. – Bauru, Dezembro/2020-    37 p. : il. (algumas color.) ; 30 cm.  
Orientador: Prof. Associado Aparecido Nilceu Marana  
Trabalho de Conclusão de Curso – Universidade Estadual Paulista “Júlio de  
Mesquita Filho”  
Faculdade de Ciências  
Ciência da Computação, Dezembro/2020.

Vinicius Dias de Vasconcelos

## **PROCESSAMENTO DE IMAGENS PARA CONTROLE DE CONTEÚDO EM SISTEMAS WEB**

Trabalho de Conclusão de Curso do Curso de Bacharelado em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

**Prof. Associado Aparecido Nilceu Marana**

Orientador

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Computação

**Profa. Associada Roberta Spolon**

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Computação

**Profa. Dra. Simone das Graças  
Domingues Prado**

Universidade Estadual Paulista "Júlio de Mesquita Filho"  
Faculdade de Ciências  
Departamento de Computação

Bauru, 14 de Dezembro de 2020.

# Agradecimentos

Agradeço a Unesp e a todo o corpo docente pelo aprendizado em todos esses anos.

# Resumo

Com o avanço dos paradigmas de reconhecimento de imagens e novas técnicas de classificação e segmentação, suas possíveis utilizações crescem no mundo. Este trabalho visa apresentar meios modernos de se utilizar dessas técnicas, embasando-se no uso de *frameworks* e bibliotecas como Angular e TensorFlow e da capacidade do *data-set* COCO, para controle de conteúdos e vídeos em sistemas *Web*. Como exemplo da utilização dessas técnicas e ferramentas, apresenta-se uma aplicação na área de segurança pública e privada em que objetos em vídeos capturados por câmeras ao vivo são classificados em tempo real.

**Palavras-chave:** Angular, TensorFlow, COCO-SSD, Google Cloud, Reconhecimento de imagens, Análise de vídeos

# Abstract

With the advancement of image recognition paradigms and new classification and segmentation techniques, their possible uses are growing in the world. This work aims to describe modern ways of using these techniques, based on the use of frameworks and libraries such as Angular and TensorFlow and the ability of COCO data-set, for content control of videos in websites. As an example of the use of these techniques and tools, we present an example of application in the area of public and private security in which objects in videos captured by live cameras are classifier in real time.

**Palavras-chave:** Angular, TensorFlow, COCO-SSD, Google Cloud, Image recognition, Video Analysis

# Listas de figuras

Figura 1 – Segmentação nos 3 âmbitos . . . . .	11
Figura 2 – Exemplos de imagens com segmentação canônica. . . . .	12
Figura 3 – Exemplos de imagens com segmentação não canônica (L. MAIRE M., 2014). . . . .	12
Figura 4 – Classificação da imagem, localização do objeto e segmentação semântica do objeto. . . . .	14
Figura 5 – Estruturação do projeto Web em componentes. . . . .	26
Figura 6 – Estruturação do visual projeto Web. . . . .	26
Figura 7 – Classificação dentro do projeto Web . . . . .	27
Figura 8 – Arquivos no <i>Bucket</i> . . . . .	29
Figura 9 – Comando gsutil em uso. . . . .	29
Figura 10 – Grupo de Pessoas. . . . .	30
Figura 11 – Grupo de Pessoas com carro. . . . .	31
Figura 12 – Pessoas e carro em outro ângulo. . . . .	31
Figura 13 – Grupo de pessoas em outro ângulo. . . . .	32
Figura 14 – Pessoas e ciclista. . . . .	33
Figura 15 – Ciclista. . . . .	33
Figura 16 – Guarda sol. . . . .	34
Figura 17 – Carros em estacionamento. . . . .	34

# **Lista de abreviaturas e siglas**

AWS	Amazon Web Services
CNN	Convolutional Neural Networks
COCO	Common Objects in Context
DOM	Document Object Model
GCP	Google Cloud Platform
SSD	Single Shot Detector
YOLO	You only look once

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>10</b>
<b>2</b>	<b>INTRODUÇÃO AO COCO</b>	<b>14</b>
<b>2.1</b>	<b>Funcionamento</b>	<b>14</b>
<b>2.2</b>	<b>Retornos do COCO</b>	<b>15</b>
<b>3</b>	<b>FERRAMENTAS</b>	<b>17</b>
<b>3.1</b>	<b>Angular</b>	<b>17</b>
3.1.1	Componentes	17
3.1.2	Templates	19
3.1.2.1	Passagem de valores	20
3.1.3	Diretivas	21
3.1.3.1	Diretivas de atribuição	21
3.1.3.2	Diretivas Estruturais	21
3.1.4	Injeção de dependências	22
<b>3.2</b>	<b>TensorFlow</b>	<b>23</b>
3.2.1	Importância e Integração	23
3.2.2	Tensores e operadores	23
3.2.3	Plataforma e ambiente	24
3.2.4	Modelos	24
<b>3.3</b>	<b>Google Cloud Platform</b>	<b>24</b>
3.3.1	<i>App Engine</i>	24
3.3.2	<i>Cloud Storage</i>	25
<b>4</b>	<b>DESENVOLVIMENTO</b>	<b>26</b>
<b>4.1</b>	<b>Projeto Web</b>	<b>26</b>
<b>4.2</b>	<b>Hospedagem da aplicação</b>	<b>28</b>
<b>5</b>	<b>APLICAÇÃO</b>	<b>30</b>
<b>6</b>	<b>CONCLUSÃO</b>	<b>35</b>
6.0.1	Trabalho futuro	35
	<b>REFERÊNCIAS</b>	<b>36</b>

# 1 Introdução

O Reconhecimento de Imagens vem se tornando cada vez mais popular e com isso novas utilizações surgem nas mais diversas áreas, tais como: OCR (Reconhecimento tico de Caracteres) ([MEMON et al., 2020](#)), Biometria ([NAMBIAR; BERNARDINO ALEXANDRE ND NASCIMENTO, 2019](#)), Reconhecimento de Atividades e Ações Humanas ([SINGH; ISHWAKARMA, 2019; ZHANG et al., 2019](#)), Monitoramento de Trâfego ([KUMARAN; DOGRA; ROY, 2019](#)), Medicina ([KALKREUTH; KAUFMANN, 2020](#)) , etc.

Várias técnicas e processamento de imagens digitais foram desenvolvidas em meados de 1960 com a necessidade de reconhecimento interpretado de imagens de satélite, principalmente no *Jet Propulsion Laboratory* (Laboratório de propulsão a Jato dos EUA). A partir daí, as técnicas evoluíram e com seu rápido avanço, novas formas de interpretar imagens, tais como as técnicas baseadas em aprendizado profundo (*deep learning*), consideradas o estado da arte atualmente.

No reconhecimento de imagens, recebe-se uma imagem ou *frames* de um vídeo, como *input*, e traduz-se essa informação para uma fácil leitura humana apoiando-se em um *Dataset*. A detecção de objetos é um dos principais objetivos do reconhecimento de imagens e pode ser usado junto a modelos de redes neurais.

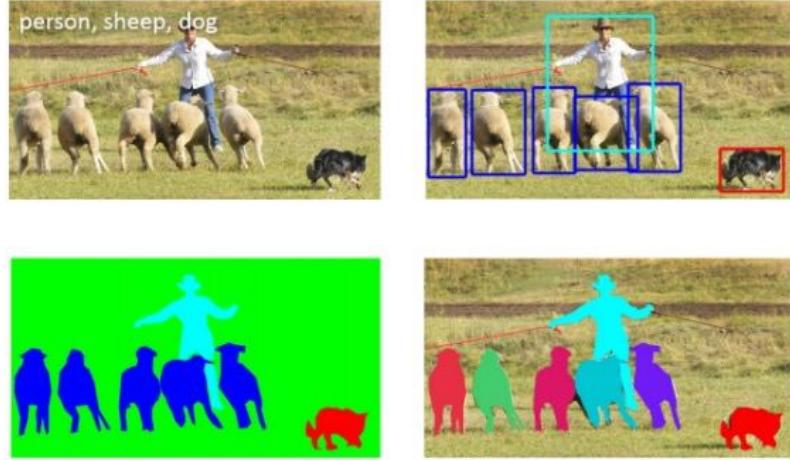
A aprendizagem profunda, do inglês *Deep-learning* é um ramo do aprendizado de máquina, que se baseia em uma série de algoritmos e abstrações, em várias camadas de processamento ([WHAT..., 2019](#)).

Vários métodos de *deep learning* para reconhecimento de imagens foram propostos, como por exemplo o método *region-based* ([Girshick et al., 2016](#)), em que combina uma região com CNN (*Convolutional Neural Networks*) para que, usando uma imagem como entrada, retira as regiões segmentadas propostas, classificado como *two-stages* ou dois-estágios, dado que, extrai informações de uma imagem no primeiro estágio e faz as previsões finais no segundo estágio. Ou também, como por exemplo, *Single-shot detector* (SSD) ([J. DONG W., 2009](#)), sendo um *single-stage* (um-estágio) que pretende ter um ganho de performance em relação ao *two-stage*, realizando previsões utilizando diferentes abordagens em uma única resolução.

Além dos métodos há também diferentes conjuntos de dados disponíveis para a resolução do problema de identificação e classificação de imagens. Sendo eles responsáveis pela acuracidade da classificação, a heterogeneidade dos *data-sets* é preponderante na capacidade classificativa dos mesmos. De fato, pode-se entender o *data-set* como o principal pilar do *deep-learning*.

Um classificador de imagens permite a classificação em três âmbitos diferentes. São eles, a classificação do objeto, a localização do objeto e a segmentação semântica do objeto,

Figura 1 – Segmentação nos 3 âmbitos



Fonte ([L. MAIRE M., 2014](#)).

como ilustra a Figura 1. Neste trabalho, a classificação será em sua maioria sobre, *single-shot detector* com múltiplas categorias. Isto significa uma abordagem mais efetiva que a proposta anteriormente, no caso, o *YOLO*, que mapeia a detecção de objetos como uma estrutura regressiva, ([Jeong; Park; Ha, 2018](#)), e com isso uma significante melhora em sua precisão.

Na Figura 1 pode-se observar que a classificação ocorre em primeira instância como uma apresentação da conjunto que aquele objeto pertence, no caso em específico *person*, *sheep* e *dog*. Após essa categorização, há então a localização do objeto e sua segmentação semântica.

A tradução de imagens para uma fácil interpretação humana é o objeto de estudo do *deep learning* e do *computer vision*. Há então grande importância na classificação de imagens e vídeos adicionadas por usuários comuns a páginas *web*. Dessa forma pode-se monitorar e classificar vídeos e neste trabalho, é apresentado o exemplo de câmeras de segurança, apresentando e classificando seus respectivos itens.

Além disso, outros conceitos importantes são o de imagens canônicas, que possuem um único objeto centralizado, e o de imagens não canônicas que possuem vários tipos de objetos diferentes dispostos de forma aleatória na imagem ([LIU et al., 2020](#)). A partir da segmentação, haverá uma classificação feita utilizando o *data-set* considerando então os objetos, bem como suas posições dentro da imagem e vídeo, apresentando as soluções mais prováveis.

Consegue-se visualizar a partir da Figura 2 um exemplo de imagens canônicas. Já na Figura 3 pode-se observar uma segmentação não canônica de várias categorias diferentes.

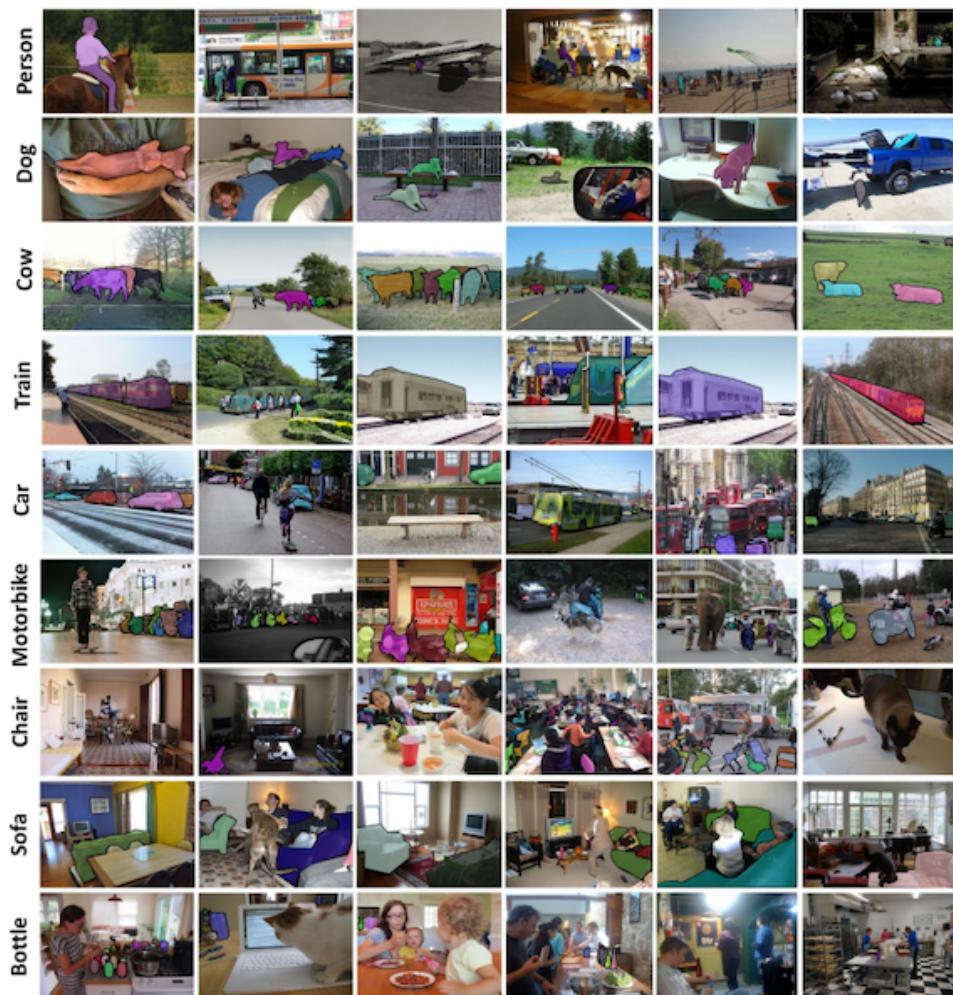
Com estes conceitos, neste trabalho aplica-se a segmentação semântica em *tags HTML* de imagens e vídeos em um *framework single page application*, ou seja, em uma página única que renderiza seus objetos dinamicamente, apresentando como exemplo a classificação de

Figura 2 – Exemplos de imagens com segmentação canônica.



Fonte ([M. VAN GOOL L., 2010](#)).

Figura 3 – Exemplos de imagens com segmentação não canônica ([L. MAIRE M., 2014](#)).



Fonte([L. MAIRE M., 2014](#)).

pessoas e objetos presentes em um vídeo de segurança.

No desenvolvimento de *software* um *framework* é, uma abstração que provê uma série de funcionalidades que podem ser utilizadas e alteradas pela escrita de código ([FRAMEWORK..., 2016](#)).

Os principais frameworks utilizados foram *Angular* ([ANGULAR..., 2020](#)) e *TensorFlow*. O primeiro com a responsabilidade de apresentar os dados de teste, bem como suas respectivas

classificações. O segundo responsável pela classificação dos objetos utilizando-se do *data-set COCO* ([COCO..., 2017](#)).

Com papel importante no desenvolvimento deste trabalho os *frameworks* modernos agregam possibilidades de controle da virtualização e renderização de objetos no navegador. Além disso, os *frameworks* são capazes de agregar uma série de conceitos que empoderam o desenvolvedor de *software*, como os apresentados no Capítulo [3](#).

O *Cloud Computing* é a capacidade de viabilidade de recursos computacionais por demanda. Ou seja, o paradigma permite a escalabilidade de recursos dada a demanda do uso. A grande mudança é que o armazenamento de dados é feito de forma decentralizada, não necessitando de instalações de programas ou infraestruturas próprias.

O *Google Cloud Platform* ([GOOGLE..., 2020](#)), por exemplo, é usado neste trabalho para que a aplicação possa ser servida e apresentada. Em suma, o *GCP* é uma plataforma oferecida pela *Google* que provê infraestrutura como serviço diretamente ao usuário final.

Outro ponto importante neste trabalho é o *Docker* ([DOCKER..., 2020](#)). Ele permite a utilização de uma virtualização a nível de sistema operacional. Com isso pode-se atribuir definições de sistema a um arquivo do tipo *.yaml* que cria um sistema operacional virtualizado no servidor ou máquina do desenvolvedor. Esse paradigma no nosso contexto, permite que façamos a integração da aplicação com o *Google Cloud* facilmente, já que tudo que é necessário para o funcionamento da aplicação fica descrito nesse arquivo.

## 2 Introdução ao COCO

Um dos objetivos primários da visão computacional é entender cenas de forma virtual. Isto envolve um grande número de tarefas, incluindo o reconhecimento de objetos em ambientes 2D e 3D. Localização, descrição e caracterização da relação entre os objetos provendo descrições semânticas da cena também são seus objetos.

Embasando-se ainda nos conceitos de imagens canônicas ou não, ou seja, centralizadas e únicas em uma imagem, ou dispersas em uma imagem não centralizadas e com isso, analisando os *data-sets open-source* disponíveis, pode-se depositar grande confiança na aplicabilidade do COCO-SSD ([L. MAIRE M., 2014](#)) por diversos motivos.

O *COCO*, *Common Objects in Context* ([COCO..., 2017](#)), conjunto de dados de acesso público, foi proposto com o objetivo de elevar o estado da arte da classificação de imagens. Seu objetivo principal é segmentar e classificar diferentes tipos de imagens não canônicas.

O desafio na criação deste *data-set* veio, principalmente, da dificuldade que é encontrar imagens que atendem esses requisitos. Após uma seleção vasta, o *COCO* passou a contar com 91 categorias disponíveis para uso, 82 delas com mais de 5 mil instâncias classificadas. No total o *data-set* possui cerca de 2,5 milhões de instâncias classificadas em 328 mil imagens.

### 2.1 Funcionamento

Dentro de sua funcionalidade, o *COCO-SSD* separa-se em três alicerces, sendo eles:

- Rotulação de todas as categorias presentes na imagem
- Localização e marcação de todas as instâncias das categorias rotuladas
- Segmentação de cada instância dos objetos

Na Figura 4 podemos observar que há, na imagem da esquerda, uma categorização de objetos. Portanto, dentro do *data-set* o sistema classifica os objetos por categorias que

Figura 4 – Classificação da imagem, localização do objeto e segmentação semântica do objeto.



Fonte ([L. MAIRE M., 2014](#))

possam estar englobadas na imagem. Na imagem central, busca-se a localização dos objetos na imagem, de forma a indicar a posição dos mesmos em *pixels*, ponto importante para a realização deste trabalho, já na imagem da direita pode-se observar os objetos semanticamente segmentados.

Especificamente para a classificação em relação a uma categoria funcionar, aplica-se então 11 super categorias, que são categorias que englobam outras categorias mais segmentadas, como expresso em ([COCO..., 2017](#)). Esse passo é chave na performance da aplicação, pois com ele consegue-se reduzir drasticamente o tempo de execução.

Nesse passo, simplesmente a região da imagem que possui uma categoria recebe essa classificação. No próximo passo, passa-se a preocupar-se com a localização dos objetos na imagem.

Após a localização do objeto em questão, temos o último estágio que é a segmentação semântica dos itens, como apresentado na Figura 4.

O classificador retorna uma série de valores como exemplificado no Código 2.1, onde tem-se um *bbox* com as coordenadas do objeto presente na imagem ou vídeo, além da categoria e probabilidade de acerto, expostos em *class* e *score* respectivamente.

#### Código 2.1 – Retorno do COCO

```
bbox :  
0: 517.4567699432373  
1: 73.41245064139366  
2: 80.84642887115479  
3: 139.12531355023384  
class : "person"  
score : 0.7112269997596741
```

## 2.2 Retornos do COCO

Como expresso em ([COCO..., 2017](#)) pode-se descrever as métricas que envolvem a classificação e segmentação da seguinte forma:

- a) *AP Average Precision* é a precisão média.
- b) *AP Across Scales* é a precisão em relação a escalas, sendo elas pequena, menor que 32 *pixels* quadrados, média, maior que 32 *pixels* quadrados e menor que 96 *pixels* quadrados, e grande, maior que 96 *pixels* quadrados.
- c) *AR Average Recall* faz referência a quantidade de detecções em uma única imagem.
- d) *AR Across Scales* faz referência a quantidade de objetos por tamanho da imagem.

Também pode-se definir os parâmetros de saída de uma classificação.

- a) "imgIds": [all] N identificações de imagens
- b) "catIds": [all] K identificações da classe
- c) "iouThrs": [.05:.95] T=10 limiares para avaliação
- d) "recThrs": [0:.01:1] R=101 limiares de objetos para avaliação
- e) "areaRng": [all,small,medium,large] A=4 Áreas para avaliação
- f) "maxDets": [1 10 100] M=3 Limitares com número máximo de detecções por imagem
- g) "useSegm": [1] Se for verdadeito o segmento é classificado como verdade fundamental
- h) "useCats": [1] Se for verdadeiro essa categoria está contida na imagem

A partir da avaliação destes resultados pode-se dizer quais objetos estão contidos nas imagens.

É importante ressaltar que nos itens 2.2 como no Código 2.1 apresentam-se exemplos dos parâmetros recebidos após uma classificação, podendo eles serem variados de acordo com as características contidas nas imagens.

Portanto, a ideia principal na detecção de objetos utilizando o COCO é nomear o *AP* e o *AR* e suas variações.

# 3 Ferramentas

Como ferramentas principais para a realização desse projeto, utiliza-se *Angular* ([ANGULAR...](#), 2020) como *framework front-end*, *TensorFlow* ([TENSOR...](#), 2020) como apoio para o *machine learning* e reconhecimento de imagens, *Google Cloud Platform* ([GOOGLE...](#), 2020) como servidor para disponibilização do *software* e *Git* ([GIT...](#), 2020) para o versionamento do código.

## 3.1 Angular

O *Angular* ([ANGULAR...](#), 2020) é um *framework open source* desenvolvido pela *Google* para a criação de aplicações *single-page*, ou seja, de página única. Dessa forma a principal capacidade do *Angular* é renderizar dinamicamente *html* e *typescript* não necessariamente precisando trafegar essas informações via *http* ou via outro protocolo de rede.

A principal vantagem do *Angular* em relação aos outros *frameworks web* disponíveis no mercado, como *React*, desenvolvido pelo *Facebook* e também *open source*, ou *Vue*, desenvolvido por Evan You, é sua robustez além de permitir alta escalabilidade em sistemas complexos.

O *framework* se baseia, majoritariamente em 4 conceitos chave. São eles:

- a) Componentes
- b) *Templates*
- c) Diretivas
- d) Injeção de dependências

### 3.1.1 Componentes

Componente é um bloco de código, conforme ilustrado em Código 3.1, sob o qual pode-se atribuir uma responsabilidade. Cada componente consiste em um código *html* para seu *template* e, portanto, o que será renderizado na página. Uma classe em *typescript*, que define o comportamento esperado daquele bloco. Um bloco *CSS* que definirá como o *component* é usado dentro de seu *template*, ou seja, seu estilo, sendo este último opcional para o funcionamento completo do componente.

Código 3.1 – Exemplo de decorator em uso.

```
@Component({  
    selector: 'app-file-classifier',  
    templateUrl: './file-classifier.component.html',  
})
```

```
styleUrls: [ './file-classifier.component.scss' ]  
})
```

Essa estrutura também é composta por um ciclo de vida, chamado de *LifeCycle*. Sendo esse ciclo de vida e demais características atribuídos pelo uso de um *decorator*.

Esta estrutura de dados, chamada *decorator*, se faz importante durante todo o uso do *typescript* no *Angular*. O *decorator*, nada mais é que um tipo especial de declaração, no qual pode-se anexá-lo a declaração de uma classe ou método. No caso do Componente, faz-se uso dessa estrutura para direcionar o *framework* apontando onde estão seus respectivos arquivos de *template* e estilo.

No caso do componente, também há a possibilidade de usá-lo com um *template inline*. Todavia, é mais agradável o uso de vários arquivos diferentes, obtendo uma melhor separação de conceitos, ponto de grande importância na engenharia de software.

Dentro da linha do ciclo de vida desse componente, na declaração de sua classe podemos definir uma série de implementações de interfaces que permitirão o uso desse ciclo de vida da melhor forma. Com isso temos métodos como:

- a) *ngOnChanges*, chamado sempre que houver mudanças nas entradas de dados desse componente
- b) *ngOnInit*, chamado sempre no início do carregamento dessa classe
- c) *ngAfterViewInit*, chamado sempre que o *template* termina de ser inicializado
- d) *ngOnDestroy*, chamado ao final do ciclo de vida do componente, quando ele é retirado da árvore

Dentro da construção deste trabalho cada um desses métodos ou interfaces tem sua importância, especialmente o *ngAfterViewInit* que é usado para aplicar o modelo de resolução da classificação de imagem à *tag html* exposta, ou seja, somente após o carregamento do *template* por completo se faz necessário o carregamento do modelo de resolução.

Pela definição de *DOM*: "The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.", ([DOM...](#), 2020) e destas observações pode-se notar a capacidade técnica do controle do *DOM* que o Angular provê, permitindo sua total racionalização em segmentos definidos.

Outro ponto importante dentro desse *decorator*, é como os dados podem caminhar por uma árvore de componentes, passando de seus pais para filhos.

### Código 3.2 – Decorator e classe FileButtonComponent

```
@Component({  
  selector: 'app-file-button',  
  templateUrl: './file-button.component.html',  
  styleUrls: [ './file-classifier.component.scss' ]  
})
```

```

        styleUrls: [ './file-button.component.scss' ]
    })
export class FileButtonComponent {
    @Input() fileAccept = 'image/jpg,video/mp4';
    @Input() buttonName = 'Adicionar';
    @Output() onFileChange = new EventEmitter<Blob>();
}

```

No Código 3.2 pode-se observar que temos dois *Inputs* e um *Output*, sendo o primeiro para o tipo de arquivo aceito e o segundo para o texto do botão, ambos valores recebidos pelo componente pai e usados pelo filho. O *Output* tem a função de emitir o *Blob* do arquivo adicionado pelo usuário para o componente pai.

Com isso, observa-se que a estrutura de dados *decorator* também é importante para a definição de propriedades de um componente, como no caso do *Input* e *Output*.

Código 3.3 – Decorator *viewChild* para definição do canvas

```

@ViewChild( 'canvas' , { static: true } ) canvas:
    ElementRef<HTMLCanvasElement>;
@ViewChild( 'img' , { static: true } ) imagem:
    ElementRef<HTMLImageElement>;

```

Ambos *viewChilds* recebem elementos do *template*, ou seja, da *view*, para sua interpretação e uso. No Código 3.3 apresenta-se o canvas, sendo ele estático e recebendo um valor do tipo *ElementRef*, ou seja, que faz uma referência a um elemento, do tipo *HTMLCanvasElement*. Em outras palavras, o *canvas* é um elemento de referência do tipo *html canvas*. A segunda variável definida é uma imagem, também elemento de referência *html*, só que dessa vez do tipo *imagem*.

Estes elementos foram usados na visualização do *COCO-SSD* dentro do *TensorFlow* como apresenta o Capítulo 4.

### 3.1.2 Templates

No *Angular* os *templates* são nada mais que um bloco de *html*. Porém, com capacidade do uso de uma sintaxe especial que capacita-nos de usar muitos elementos diferentes.

A interpolação é o principal conceito de utilização do *template*. Esse conceito permite a atribuição de elementos do *typescript* no *html*. Com isso consegue-se apresentar um dado, ou mesmo atribuir um valor a uma *tag html*, conforme ilustra o Código 3.4 com total controle da mudança de valor.

Código 3.4 – Exemplos de passagem de valor por interpolação  
<h4>{{ recommended }}</h4>

```
<img [src]="item.imageUrl2">
```

Com esse conceito, consegue-se aproveitar de eventos de elementos *html* para disparar funções expostas no *typescript*. Temos diferentes exemplos do uso desse conceito no Código 3.5, o principal deles é quando a partir de uma mudança no valor do *input*, emite-se um evento chamado *onFileChange*, como apresentado no Código 3.2, por sua vez, emite o objeto que faz referência à esse evento de mudança. Embasado nessa técnica pode-se enviar o arquivo selecionado pelo usuário em seu navegador através da árvore de componentes.

#### Código 3.5 – Exemplos

```
<input
#fileUpload
type="file"
class="no-display"
(click)="fileUpload.value=null"
(change)="onFileChange.emit($event)"
[accept]="fileAccept"
/>
<button type="button" mat-flat-button
(click)="fileUpload.click()" color="primary">
<i class="fa fa-plus"></i>
{{ buttonName }}
</button>
```

##### 3.1.2.1 Passagem de valores

A partir dos conceitos ilustrados nos Códigos 3.4, 3.5 e 3.2 pode-se utilizar o conceito chave da passagem de valores através da árvore de componentes. A importância desse conceito se faz pelo uso da detecção de mudanças em um componente, é isso que de fato realiza uma nova renderização da tela. Deste modo, esse conceito está totalmente ligado a performance do *framework* como um todo.

#### Código 3.6 – Uma via do typescript para a view

```
{{ expression }}
[target]="expression"
bind-target="expression"
```

#### Código 3.7 – Uma via da view para o typescript

```
(target)="statement"
on-target="statement"
```

### Código 3.8 – Duas vias

```
[( target )]= "expression"
bindon-target= "expression"
```

Pode-se passar parâmetros de uma via, sendo ela a partir de um dado expresso no *typescript*. Nesse caso o Angular tem a capacidade de apresentar variáveis na tela, ou mesmo atribuindo-as a um elemento *html*. No Código 3.5 pode-se ver um elemento *html accept*, que dita quais os tipos de arquivos permitidos para leitura. Então tem-se a possibilidade de a partir de dados do *typescript* realizar ações no *html*.

Outro exemplo da conversa de uma via é o uso de elementos ou eventos *html* na atribuição de valores no *typescript*. No Código 3.7 pode-se analisar o caso em que o evento *target* é atribuído a variável *statement*. No Código 3.5 tem-se um exemplo é o clique do botão, em que disparamos o evento do *input* de clique e assim pode-se abrir a tela de arquivos no computador do usuário. Como expresso no Código 4 essa atribuição é importante para levantar o dado a ser testado.

No caso do exemplo, ilustrado no Código 3.8 permitimos que o *typescript* e o *html* tenham uma conversa bilateral, desta forma a mudança de valor pode sugerir de ambas as partes. Um exemplo prático desse ponto é o uso de formulários, em que o valor interno ao *input* pode mudar e também a partir do *typescript* esse valor pode ser alterado.

## 3.1.3 Diretivas

As diretivas mudam a aparência ou comportamento de um elemento *DOM* no caso das de atribuição. Ou podem alterar completamente a estrutura do *DOM*, redesenhando-a de forma completa que são as estruturais.

### 3.1.3.1 Diretivas de atribuição

Uma diretiva de atribuição propõe uma característica interna a *tag html* da qual ela pertence. No caso expresso no Código 3.9 usa-se uma diretiva de atribuição que permite o destaque em uma cor do texto.

### Código 3.9 – Duas vias

```
<p [appHighlight]= "color">Destaque!</p>
```

### 3.1.3.2 Diretivas Estruturais

Uma diretiva estrutural permite que manejemos toda a estrutura do *template* baseando-se em conceitos básicos de programação. O caso mais simples pra isso é quando usamos um condicional no *html*. No Código 3.10 tem-se uma diretiva *ngIf* que só renderiza o que está

dentro dela se o condicional for verdadeiro. No Código 3.11 tem-se um caso semelhante, porém com a estrutura de dados do laço.

Código 3.10 – Duas vias

```
<div *ngIf="heroi" class="name">{{ heroi.name }}</div>
```

Código 3.11 – Duas vias

```
<li *ngFor="let heroiof herois">{{ heroi.name }}</li>
```

### 3.1.4 Injeção de dependências

Toda a organização de arquivos do *Angular* é baseada no conceito de injeção de dependências. Isto significa que organizam-se as estruturas, reproveitáveis ou não, em módulos e desta forma, importam-se outros módulos, declaram-se componentes, provêm-se serviços e exportam-se componentes. Com isso, um componente pode ser usado em diversos lugares de uma aplicação, a regra de negócio pode estar isolada em um serviço e o princípio de separação de conceitos é atingido a nível de organização de arquivos e dependências.

Código 3.12 – Declaração da classe FileClassifierModule

```
@NgModule({
  imports: [
    CommonModule,
    MatButtonModule,
    FileButtonModule,
    MatCardModule,
    MatGridListModule,
    MatSelectModule,
    ReactiveFormsModule,
  ],
  declarations: [FileClassifierComponent],
  exports: [FileClassifierComponent]
})
export class FileClassifierModule {}
```

No Código 3.12 pode-se observar novamente o uso de um *decorator*, dessa vez na declaração de uma classe módulo, que tem a função de importar diversos itens, declarar e exportar seu componente.

- CommonModule* é o módulo comum para que as diretivas estruturais possam ser utilizadas.
- MatButtonModule* é o módulo do botão

- c) *FileButtonModule* é o módulo do botão que adiciona arquivos ao projeto
- d) *MatCardModule* é o módulo do card no qual os inputs e botão de arquivos estão adicionados
- e) *MatGridListModule* é o módulo que cria um grid de itens
- f) *MatSelectModule* é o módulo do select que usamos para definir se é imagem ou video
- g) *ReactiveFormsModuleModule* é o módulo do formulário que utilizamos no projeto

Com isso, se desejarmos utilizar esse componente de classificação de imagens em outro lugar do projeto somente precisamos importá-lo e todas as suas dependências são importadas também.

## 3.2 *TensorFlow*

O TensorFlow ([TENSOR..., 2020](#)) é um software *open-source* que permite o uso de aprendizado de máquina. Ele pode ser usado com modelos treinados que possuem inferência sobre *deep learning*. Ele é uma biblioteca matemática baseada em um fluxo de dados e programação.

### 3.2.1 Importância e Integração

A principal tarefa do *TensorFlow* neste trabalho é a possibilidade de integração com o *COCO-SSD* ([COCO..., 2017](#)), permitindo o uso do *data-set COCO* apoiado no conceito de *single-shot detection*.

Para seu uso, apoia-se no administrador de pacotes *NPM*, que permite a utilização de pacotes no ecossistema *Node*. Com este conceito podemos instalar um novo pacote [3.13](#) e fazer sua utilização a partir de sua importação [3.14](#).

Código 3.13 – Instalação do TensorFlow com COCO-SSD

```
npm instal —save @tensorflow—models/coco—ssd
```

Código 3.14 – Importação do COCO-SSD

```
import * as cocoSsd from '@tensorflow—models/coco—ssd';
```

### 3.2.2 Tensores e operadores

Tensores são uma generalização de vetores e matrizes para dimensões maiores, enquanto operadores são as funções que nos permitem manipular dados. Um tensor contém as seguintes propriedades:

- a) *rank*: Define quantas dimensões tem o tensor.
- b) *shape*: Define o tamanho de cada dimensão do dado.
- c) *dtype*: Define o tipo de dado do tensor.

Estes operadores são importantes para que consiga-se manipular as matrizes de forma a atingir a resolução necessária. Estas funções são embasadas no *WebGL* como *backend*, ou seja, o *TensorFlow* ([TENSOR... , 2020](#)) faz uso de um *backend* mais poderoso que o CPU comum. Nesse caso é possível armazenar texturas e operações matemáticas com muito mais memória disponível. No caso do ecossistema Node um *backend* diferente é utilizado, sendo mais comum o uso do próprio CPU

### 3.2.3 Plataforma e ambiente

O *TensorFlow javascript* está inserido no contexto do navegador ou no ecossistema Node. Ambas as plataformas possuem diferentes configurações que permitem customizar a aplicação como um todo. No navegador utilizamos como *WebGL*, que é uma biblioteca *javascript* especializada em renderização de objetos 2D e 3D. Esta biblioteca tem sua importância no processamento dos modelos dentro do *TensorFlow*, pela sua performance computacional, dado que faz uso da GPU em sua maioria para o processamento. Do outro lado podemos utilizar o ecossistema Node e realizar o processamento dentro de um CPU comum de forma mais lenta.

No caso deste trabalho operamos todas as classificações embasando-se no *backend WebGL* que é cerca de 100 vezes mais poderoso que a CPU comum.

### 3.2.4 Modelos

Em *machine learning* o modelo é uma função com parâmetros configuráveis e com capacidade de aprendizado. Isto significa que a partir do *TensorFlow* pode-se treinar um modelo, ou mesmo utilizar de um modelo treinado para realizar classificações

## 3.3 Google Cloud Platform

O *Google Cloud* ([GOOGLE... , 2020](#)) tem como objetivo modernizar cargas de trabalho provendo uma infraestrutura como serviço. Dentre essas soluções existem diferentes produtos que funcionam com um pagamento por uso, criando uma relação com o usuário de valor por demanda. O serviço de *Cloud* da *Google* ainda dispõem de suporte técnico para seus clientes.

### 3.3.1 App Engine

O *App Engine* é um dos produtos disponíveis dentro do *Google Cloud*. Ele apresenta uma plataforma sem servidor, totalmente gerenciada para hospedar aplicativos web em escala.

Seu papel é fundamental pois permite aplicações de diversos formatos, apoiadas ou não em *frameworks*, em diversas escalas serem hospedadas. O ponto crucial no seu funcionamento é a auto escalabilidade por demanda, ou seja, o *Google* aloca mais recursos em suas máquinas virtuais dado o uso das aplicações.

### 3.3.2 *Cloud Storage*

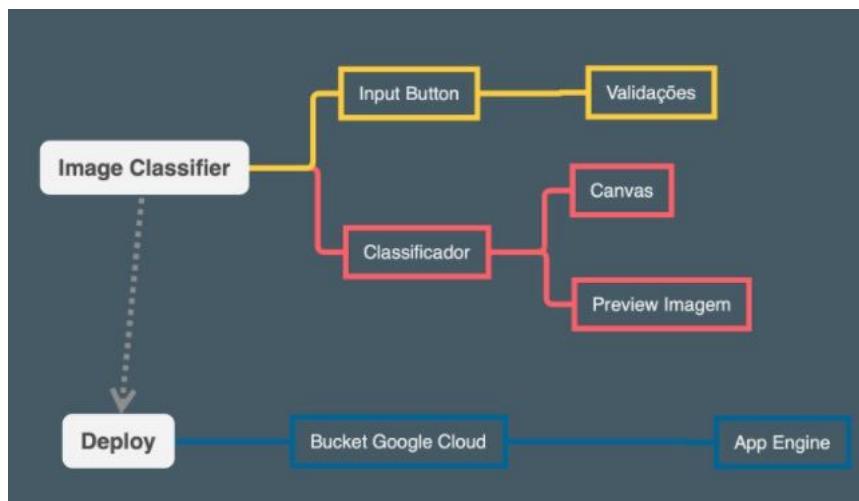
O *Cloud Storage*, ou *bucket*, é simplesmente um armazenador de objetos na nuvem. Ele permite que armazenam-se os arquivos gerados após a compilação do projeto *Angular* na nuvem, o que torna a hospedagem do site a partir do *App Engine* com uma maior performance computacional.

# 4 Desenvolvimento

## 4.1 Projeto Web

Dentro do projeto web, propõe-se um estruturação, como ilustra a Figura 5, para os componentes que realizam a classificação.

Figura 5 – Estruturação do projeto Web em componentes.



Fonte: Elaborada pelo autor.

Dentro do componente principal *ImageClassifier* temos as classificações, bem como o botão de *input* que recebe a imagem ou vídeo. A partir do recebimento deste arquivo o apresentamos em tela e permitimos o usuário escolher entre vídeo e imagem, conforme Figura 6.

Figura 6 – Estruturação do visual projeto Web.

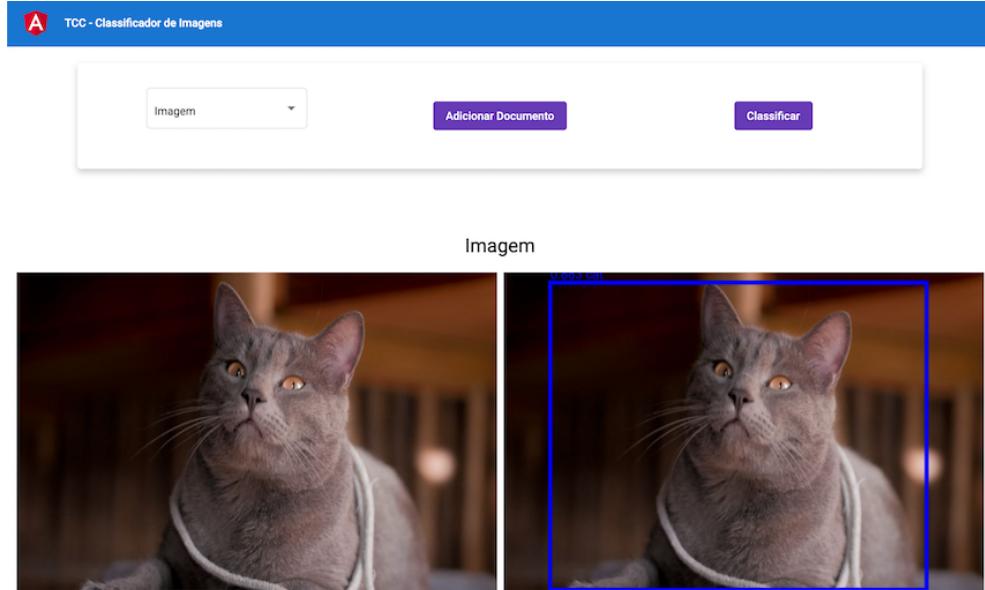


Fonte: Elaborada pelo autor.

A partir do estabelecimento do que será classificado, faz-se uso do *canvas*, permitindo que desenhos sejam construídos na tela de forma a segmentar semanticamente o item encontrado.

Na Figura 7 pode-se observar o sistema em funcionamento, classificando uma imagem, desta forma, junto ao resultado expressado pelo classificador, consegue-se localizar o item na imagem e desenhá-lo.

Figura 7 – Classificação dentro do projeto Web



Fonte: Elaborada pelo autor.

Com isso o *COCO-SSD* pode ser carregado após a *view* ser renderizada, como ilustrado no Código 4.1.

#### Código 4.1 – Função que executa o COCO-SSD

```
async ngAfterViewInit() {
    this.model = await cocoSsd.load();
}
```

A seleção da imagem se dá pelo estabelecimento da referência da *tag HTML* que a expõe. Esta referência é passada para o modelo e com isto o desenho pode ser feito a partir da matriz de resultados. Nos trechos apresentados no Código 4.2 e Código 4.3, pode-se analisar como o código é feito a partir dos resultados obtidos pelo *TensorFlow*. No Código 4.4 é apresentado como o resultado do COCO é desenhado no *canvas*

#### Código 4.2 – Função que desenha

```
result = await this.model.detect(this.imagem.nativeElement);
context = this.canvas.nativeElement.getContext('2d');
this.drawImage = true;
context.drawImage(this.imagem.nativeElement, 0, 0, 600, 399);
```

#### Código 4.3 – HTML do canvas da imagem

```
<h1>Imagen</h1>
<mat-grid-list cols="2" rowHeight="400px">
<mat-grid-tile [colspan]="1" [rowspan]="1">
```

```

<img #img width="600" height="399" [src]="imageUrl"/>
</mat-grid-tile>
<br/>
<mat-grid-tile [colspan]="1" [rowspan]="1">
<canvas #canvas width="600" height="399"></canvas>
</mat-grid-tile>
</mat-grid-list>

```

Código 4.4 – Desenho dos resultados

```

result.forEach((element, index) => {
    context.beginPath();
    context.rect(
        element.bbox[0],
        element.bbox[1],
        element.bbox[2],
        element.bbox[3]);
    context.lineWidth = 5;
    context.strokeStyle = 'blue';
    context.fillStyle = 'blue';
    context.stroke();
    context.fillText(
        element.score.toFixed(3) + ' ' +
        element.class,
        element.bbox[0],
        element.bbox[1] > 10 ? element.bbox[1] - 5 : 10);
});

```

## 4.2 Hospedagem da aplicação

Como já mencionado no texto, o *Google Cloud* é utilizado na hospedagem da aplicação, dentro dele cria-se um *bucket*, com os arquivos correspondentes a compilação da aplicação e ao objeto de configuração do *deploy* como pode-se observar no Código 4.5, o objeto define o caminho dos arquivos e pastas de compilação e qual o hospedeiro. Na Figura 8 pode-se observar estes arquivos hospedados.

Código 4.5 – Objeto de configuração da hospedagem

```

runtime: python27
api_version: 1
threadsafe: true

```

```

handlers:
- url: /
  static_files: dist/tcc-image-classifier/index.html
  upload: dist/tcc-image-classifier/index.html
- url: /
  static_dir: dist/tcc-image-classifier

```

Figura 8 – Arquivos no *Bucket*.

	Nome	Tamanho	Tipo
<input type="checkbox"/>	app.yaml	228 B	application/x-yaml
<input type="checkbox"/>	dist/	–	Pasta

Fonte: Elaborada pelo autor.

Na sequência realizam-se a hospedagem, efetivamente, do site no *app-engine*, atribuindo uma pasta dentro da máquina virtual apontando para o *bucket* conforme Figura 9. E por fim realizamos o *deploy* da aplicação.

Figura 9 – Comando gsutil em uso.

```

viniciusdvasc@cloudshell:~ (tcc-proj-295122)$ gsutil rsync -r gs://tcc-bucket-1123 .
Building synchronization state...
Starting synchronization...
Copying gs://tcc-bucket-1123/app.yaml...
Copying gs://tcc-bucket-1123/dist/tcc-image-classifier/vendor-es2015.js.map...
- [2 files] [ 6.0 MiB/ 6.0 MiB]
Operation completed over 2 objects/6.0 MiB.

```

Fonte: Elaborada pelo autor.

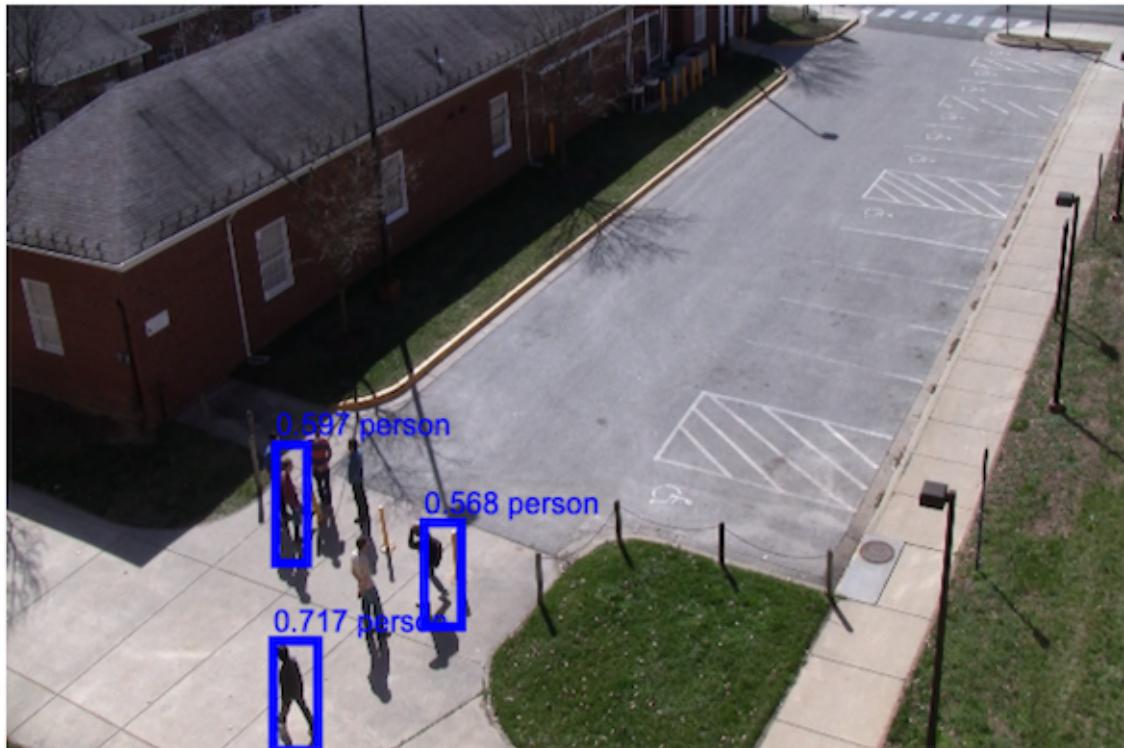
## 5 Aplicação

Foram usados diferentes vídeos de câmeras de segurança para a realização das classificações, como na Figura 10, onde apresenta-se um grupo de pessoas detectadas pelo sistema. O intuito desta análise é detectar objetos em vídeos de monitoramento a partir da sincronização em sistemas *Web*, averiguando não só a aproximação de pessoas, mas também de outros objetos. Os vídeos utilizados neste trabalho foram retirados de um data-set aberto de câmeras de vigilância espalhadas pelo mundo ([OH ANTHONY HOOGS; DESAI, 2011](#)).

É importante ressaltar que esta é uma das aplicações possíveis para o sistema apresentado, sendo ele funcional a partir de todas as categorias existentes no COCO ([COCO..., 2017](#)).

Na Figura 11 pode-se observar um grupo de pessoas com carro, referenciados pelo classificador. Também pode-se observar falsos negativos de pessoas próximas ao carro.

Figura 10 – Grupo de Pessoas.



Fonte: Elaborada pelo autor.

Nas Figuras 12 e 13 pode-se observar um enquadramento diferente dos objetos na imagem. Ainda sim a Figura 12 apresenta um false negativo.

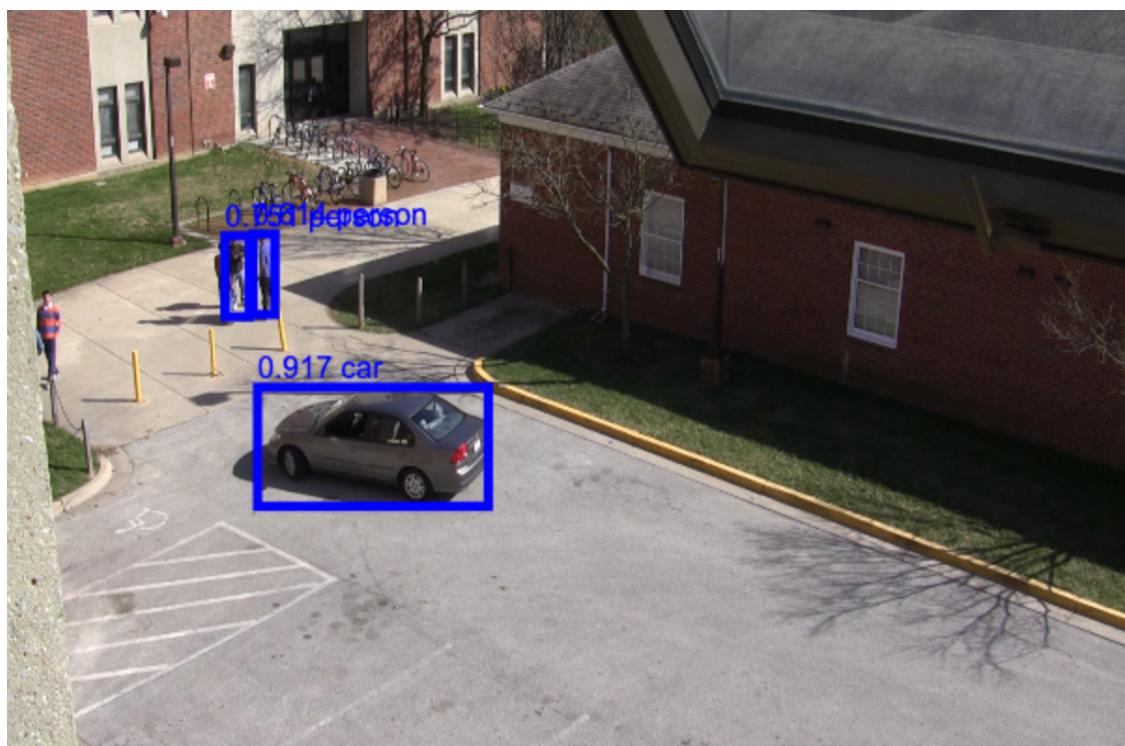
Nas Figuras 14 e 15 pode-se observar que obteve-se a segmentação correta de todas as

Figura 11 – Grupo de Pessoas com carro.



Fonte: Elaborada pelo autor.

Figura 12 – Pessoas e carro em outro ângulo.



Fonte: Elaborada pelo autor.

Figura 13 – Grupo de pessoas em outro ângulo.



Fonte: Elaborada pelo autor.

pessoas presentes na imagem.

Em um contexto com ausência de pessoas o classificador ainda se mostra útil, como nas Figuras 16 e 17 onde tem-se duas situações diferentes. Na primeira guarda-sois são classificados e na segunda um grupo de carros é classificado em um estacionamento. Em ambas imagens pertencentes a câmeras de segurança o classificador tem a capacidade de identificar itens presentes na imagem, porém ainda sim com grande número de falsos negativos.

Em todas as análises efetuadas para das Figuras 11, 10, 16 e 17 e em outros momentos do vídeo, temos uma imprecisão em relação ao número de acertos do classificador.

Figura 14 – Pessoas e ciclista.



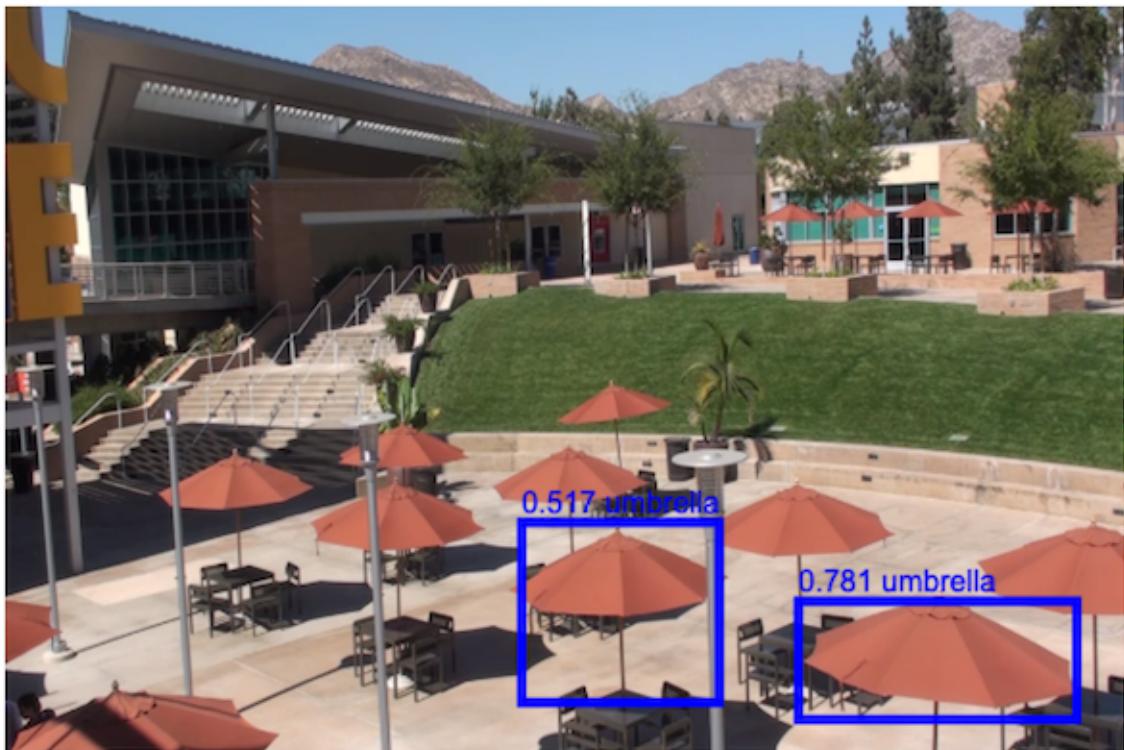
Fonte: Elaborada pelo autor.

Figura 15 – Ciclista.



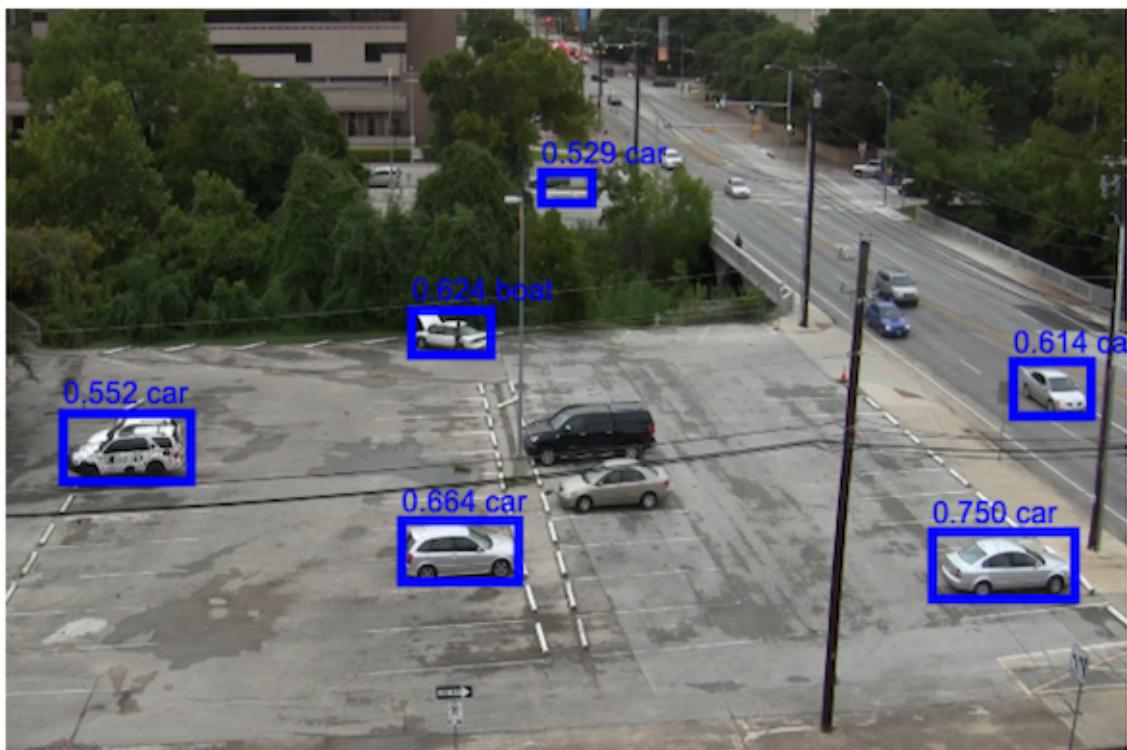
Fonte: Elaborada pelo autor.

Figura 16 – Guarda sol.



Fonte: Elaborada pelo autor.

Figura 17 – Carros em estacionamento.



Fonte: Elaborada pelo autor.

# 6 Conclusão

A partir da importância do *machine learning* e do reconhecimento de imagens as suas aplicabilidades só crescem no mundo. Com a definição de *Single Shot Detector* e utilizando o *dataset COCO* ([COCO... , 2017](#)) este trabalho desenvolve uma integração entre o *TensorFlow* e o *COCO*, apresentando um exemplo de aplicabilidade de suas funções em um sistema *web* para classificação objetos em vídeos capturados por câmeras de segurança. Este sistema foi construído utilizando um dos *frameworks* modernos mais robustos, o *Angular*. Com este *framework* torna-se possível a atualização da tela de forma dinâmica, sem a necessidade do tráfego de informações ou mesmo *HTML* pela rede. Além disso, embasado no *TensorFlow* o sistema é capaz de utilizar o processamento da máquina do usuário para realizar tarefas complexas de *machine learning* e classificação de imagens. Além disso foi projetado uma árvore de componentes que permite o usuário inserir suas próprias imagens e vídeos para classificação.

Para hospedagem da aplicação utiliza-se o conceito de *Cloud Computing* junto do *Google Cloud Platform* e seus serviços para que possa-se ter alta disponibilidade.

Dada a análise final das imagens e vídeos atribuídos à este *software* pode-se concluir que o *data-set COCO* é de grande capacidade de classificação, porém apresenta inúmeros falsos negativos.

## 6.0.1 Trabalho futuro

Nos próximos passos deste trabalho pretende-se aplicar outros data-sets e averiguar o ganho de performance utilizando o *WebGL* como *backend* do *TensorFlow* em comparação com o *CPU*. Pretende-se também avaliar mais profundamente a causa de tantos falsos negativos na utilização do *TensorFlow* com o *data-set COCO*.

# Referências

- ANGULAR Documentations. 2020. Disponível em: <<https://angular.io/>>. Acesso em 20 de Outubro de 2020.
- COCO - Docs. 2017. Disponível em: <<https://cocodataset.org/#explore>>. Acesso em 23 de Outubro de 2020.
- DOCKER Documentations. 2020. Disponível em: <<https://docs.docker.com/>>. Acesso em 12 de Outubro de 2020.
- DOM Documentations. 2020. Disponível em: <[https://developer.mozilla.org/en-US/docs/Web/API/Document\\_Object\\_Model/Introduction#:~:text=The%20Document%20Object%20Model%20\(DOM\)%20is%20a%20programming%20interface%20for,structure%2C%20style%2C%20and%20content.&text=The%20DOM%20is%20an%20object,scripting%20language%20such%20as%20JavaScript](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction#:~:text=The%20Document%20Object%20Model%20(DOM)%20is%20a%20programming%20interface%20for,structure%2C%20style%2C%20and%20content.&text=The%20DOM%20is%20an%20object,scripting%20language%20such%20as%20JavaScript)>. Acesso em 20 de Outubro de 2020.
- FRAMEWORK definition Quora. 2016. Disponível em: <<https://www.quora.com/What-is-framework-in-software-engineering>>. Acesso em 12 de Outubro de 2020.
- Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Region-based convolutional networks for accurate object detection and segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 38, n. 1, p. 142–158, 2016.
- GIT documentations. 2020. Disponível em: <<https://git-scm.com/doc>>. Acesso em 12 de Outubro de 2020.
- GOOGLE Cloud Documentations. 2020. Disponível em: <<https://cloud.google.com/>>. Acesso em 13 de Outubro de 2020.
- J. DONG W., e. a. D. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Proceedings / CVPR, 2009.
- Jeong, H.; Park, K.; Ha, Y. Image preprocessing for efficient training of yolo deep learning networks. In: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. [S.I.: s.n.], 2018. p. 635–637.
- KALKREUTH, R.; KAUFMANN, P. Covld-19: a survey on public medical imaging data resources. *arXiv preprint arXiv:2004.04569*, 2020.
- KUMARAN, S. K.; DOGRA, D. P.; ROY, P. P. Anomaly detection in road traffic using visual surveillance: A survey. *arXiv preprint arXiv:1901.08292*, 2019.
- L. MAIRE M., B. S. T.-Y. *Microsoft COCO: Common Objects in Context*. [S.I.]: arXiv:1405.0312, 2014. 261–318 p.
- LIU, L.; OUYANG, W.; WANG, i.; FIEGUTH, P.; CHEN, J.; LIU, X.; PIETIKÄINEN, M. Deep learning for generic object detection: A survey. *International journal of computerision*, Springer, v. 128, n. 2, p. 261–318, 2020.

- M. VAN GOOL L., W. C. e. a. E. The pascal visual object classes (voc) challenge. *IJCV*, Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition., v. 88, n. 1, p. 303–338, 2010.
- MEMON, J.; SAMI, M.; KHAN, R. A.; UDDIN, M. Handwritten optical character recognition (ocr): A comprehensive systematic literature review (slr). *IEEE Access*, IEEE, v. 8, p. 142642–142668, 2020.
- NAMBIAR, A.; BERNARDINO ALEXANDRE ND NASCIMENTO, J. C. Gait-based person re-identification: A survey. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 52, n. 2, p. 1—34, 2019.
- OH ANTHONY HOOGS, A. P. N. C. C.-C. C. J. T. L. S. M. J. A. H. L. L. D. E. S. X. W. Q. J. K. R. M. S. C. V. H. P. D. R. J. Y. A. T. B. S. A. F. A. R.-C. S.; DESAI, M. A large-scale benchmark dataset for event recognition in surveillance video. *IJCV*, Proceedings of IEEE Comptuer Vision and Pattern Recognition (CVPR)., v. 1, n. 1, p. 1–10, 2011.
- SINGH, T.; ISHWAKARMA, D. K. Human activity recognition in video benchmarks: A survey. In: . [S.I.]: Springer, 2019. p. 247–259.
- TENSOR Flow Documentations. 2020. Disponível em: <<https://www.tensorflow.org/js/>>. Acesso em 13 de Outubro de 2020.
- WHAT is deep learning? 2019. Disponível em: <<https://machinelearningmastery.com/what-is-deep-learning/>>. Acesso em 16 de Outubro de 2020.
- ZHANG, H.-B.; ZHANG, Y.-X.; ZHONG, B.; LEI, Q.; YANG, L.; DU, J.-X.; CHEN, D.-S. A comprehensive survey of vision-based human action recognition methods. Multidisciplinary Digital Publishing Institute, v. 19, n. 5, p. 1005, 2019.