Universidade de São Paulo
Escola Politécnica

Marcelo Teixeira Rocha
Daniel Willian Braz Pires Domingues

Aplicação de detecção de sonolência em motoristas utilizando Deep Learning

São Paulo
2022

MARCELO TEIXEIRA ROCHA
DANIEL WILLIAN BRAZ PIRES DOMINGUES

Aplicação de detecção de sonolência em motoristas utilizando Deep Learning

— Versão Original —

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Orientador: Prof. Dr. Jun Okamoto Junior

São Paulo
2022

Catalogação-na-publicação

Rocha , Marcelo Teixeira ; Domingues , Daniel Willian Braz Pires
Aplicação de detecção de sonolência em motoristas utilizando Deep Learning/ M.T. Rocha , D.W.B.P. Domingues – São Paulo, 2022.
150p.

Trabalho de formatura – Escola Politécnica da Universidade de São Paulo.

1. Sonolência 2. Detecção 3. Motorista 4. Redes Neurais.
I. Universidade de São Paulo. Escola Politécnica. II.t.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Acidentes causados por sonolência representam cerca de 42% dos incidentes de trânsito, segundo pesquisa realizada em 2017 pela Associação Brasileira de Medicina de Tráfego (ABRAMET) em parceria com a Academia Brasileira de Neurologia e com o Conselho Regional de Medicina. Ainda pela ABRAMET. A campanha "Não dê carona ao sono!" expõe as principais causas para esse tipo de acidente, que se resumem à privação ou insuficiência de sono (dormir poucas horas por noite), a longos intervalos sem descanso (ficar acordado por várias horas seguidas) e transtornos relacionados ao sono.

Diante desse cenário, das adversidades no trânsito e dos avanços tecnológicos, Sistemas Avançados de Assistência ao Motorista (ADAS) ganharam ênfase na última década, sendo gradualmente aprimorados e implementados em linhas de montagem automotivas. Além de funcionalidades que afetam diretamente a condução, tais como frenagem automática e ajuste de velocidade, os ADAS também emitem sinais de alertas para o motorista, como é o caso na detecção de sonolência. Uma das possíveis abordagens para a detecção fundamenta-se no processamento dos *frames* provenientes de uma câmera que monitora a face do condutor, sendo este um procedimento também estudado na literatura (Chirra, Uyyala e Kolli (2019), Hashemi, Mirrashid e Shirazi (2020)).

Atualmente, se um usuário deseja adquirir um sistema de detecção de sonolência, a aquisição é realizada, principalmente, através de dois canais: ou pela compra de veículos que possuem os ADAS incorporados de fábrica ou por *kits* de sensores e câmeras que realizam a detecção. Nota-se, portanto, que a acessibilidade desse sistema é restrita quando se trata de veículos que não contém os ADAS, sobretudo pelo alto valor para obtenção dos *kits*.

De forma complementar, vale ressaltar o constante aprimoramento e o crescente número de ferramentas de visão computacional presentes nos dispositivos móveis. Nesse sentido, o uso de dispositivos móveis para solução do problema é um ponto-chave para

que se possa alcançar os resultados pretendidos, tendo em vista sua ampla acessibilidade atualmente. Porém, existem desafios a serem superados:

Por um lado, há complicações para o desenvolvimento nesse tipo de plataforma, tais como a dificuldade em garantir o funcionamento em tempo real e o processamento em ambientes em que existe pouco controle de parâmetros importantes, como a iluminação e o ângulo da câmera, além da necessidade de adequação às variações de hardware e software, já que o emprego da solução deve ser adequado aos diversos dispositivos com diferentes características. Entretanto, por outro lado, a implementação em aparelhos celulares acrescenta acessibilidade à aplicação, considerando que grande parte da população já os possui, possibilitando que mais pessoas se beneficiem de uma tecnologia restrita a poucas marcas automotivas, sobretudo no mercado de veículos populares.

Diante da necessidade de tornar acessível a tecnologia de detecção de sonolência para o amplo público de motoristas, surge a motivação para o presente trabalho. Além disso, estudado o problema e selecionada como melhor alternativa de solução o desenvolvimento de uma aplicação baseada em *Deep Learning* em Redes Neurais Convolucionais, verifica-se sua viabilidade de implementação. Nesse sentido, podemos atribuir a praticabilidade do projeto aos recentes e notórios avanços dos softwares de visão computacional e redes neurais e também à capacidade de armazenamento e disponibilidade de grandes volumes de dados para *Machine Learning* atualmente.

Nesse contexto, este trabalho possui como finalidade o desenvolvimento de um algoritmo de classificação binária que, a partir dos dados de entrada, que são os *frames* coletados pela câmera de um *smartphone* que monitora a região da face de uma pessoa (neste caso, desempenhando a função de motorista), seja capaz de retornar como saída um valor indicando seu estado de atenção entre ”Acordado”ou ”Sonolento”.

Em caso de detecção de sonolência, o objetivo é que o *smartphone* seja utilizado como um dispositivo de segurança; para isso, a aplicação deve, além de incorporar o algoritmo de visão computacional, também ser capaz de disparar um sinal de alerta, sonoro e/ou visual, por meio da tela e do auto-falante do *smartphone*, com o intuito de despertar a(o) motorista e evitar um potencial acidente.

Para isso, treinou-se, a partir de uma base de dados de imagens extraídas de vídeos, uma rede neural convolucional para detecção das características indicadoras de sonolência utilizando toda a face da(o) motorista, implementando-a em uma aplicação para *smartphones*. A seleção da base de dados, o treinamento do modelo e o desenvolvimento da aplicação foram realizados de forma que os requisitos do projeto fossem cumpridos, o que foi verificado a partir de testes com o produto final para validação do sistema proposto.

Capítulo 2

Detecção de sonolência por visão computacional e aprendizado de máquina

Para a detecção de sonolência, podem ser comumente utilizadas abordagens fisiológicas ou comportamentais. No primeiro caso, sinais fisiológicos, tais como os provenientes da atividade cerebral, são analisados através de técnicas, como a Eletroencefalografia (EEG), para verificar variações que indiquem sonolência, assim como é explorado pelo trabalho de Stancin, Cifrek e Jovic (2021). Estudos sobre a modelagem da sonolência por sinais resultantes do EEG, com redes neurais convolucionais (CNN), também são amplamente abordados e acurados, como se observa em Zhu et al. (2021). Para uma aplicação para motoristas, no entanto, esse tipo de abordagem é limitada, uma vez que ela requer captores, em alguns casos invasivos, que podem prejudicar a capacidade de condução do veículo. Neste caso, uma abordagem comportamental pode ser mais adequada, uma vez que usa um procedimento que pouco interfere na condução, sendo esta uma estratégia comumente utilizada em veículos com ADAS.

Observa-se, na literatura, que existe uma ênfase na análise facial do motorista para identificação de comportamento sonolento. Nesse sentido, nota-se uma metodologia comum, baseada nas seguintes etapas: detecção da face (obrigatória), refinamento da região de interesse (ROI) (opcional), treinamento de um modelo para classificação de sonolência (opcional), e, por fim, uma etapa de definição dos critérios para a categorização da sonolência (obrigatória). Neste documento, as etapas referenciadas como ”obrigatórias”são aquelas executadas em todos os casos analisados, enquanto que as ”opcionais”são procedidas somente por parte dos autores.

2.1 Detecção da face

O algoritmo de Viola-Jones é bastante utilizado para a detecção da face, especialmente pela facilidade de implementação e pela possibilidade de detecção de regiões de interesse, a exemplo da área facial, assim como implementado nos trabalhos de Chirra, Uyyala e Kolli (2019), Suresh et al. (2021), Indrabayu, Mufti e IntanSariAreni (2019) e Hashemi, Mirrashid e Shirazi (2020). Com a mesma finalidade, Sinha, Aneesh e Gopal (2021) exploraram o uso do Viola-Jones em uma análise comparativa com a biblioteca *Dlib* e o uso da arquitetura *CNN Yolo v3*. Ressalta-se que, conforme observado pelos autores, uma das limitações do Viola-Jones é não ser eficaz na identificação quando a face está em uma posição lateral. O Viola-Jones, com modificações e recursos extras auxiliados pela biblioteca *openCV*, é uma das implementações realizadas pelo *framework open-source Dlib*, que é comumente utilizado em trabalhos envolvendo aplicações *Android*, como explorado por Jabbar et al. (2020) na detecção da face. Outro *framework*, o *Google Face API*, foi utilizado pelos autores Rajkumarsingh e Totah (2021) para o rastreamento do rosto. Outra abordagem recorrente para identificação da face é por *CNN*, utilizando-se arquiteturas como *SSD (Single Shot MultiBox Detector)* com *ResNet-10* (Phan et al. (2021)) e *MobileNetSDDv1* (Dipu et al. (2021)). Outra possibilidade é o uso de recursos semelhantes a *Haar* (Dasgupta, Rahman e Routray (2019)).

2.1.1 Refinamento da região de interesse

Após detectada a face, alguns autores optam por delimitar ainda mais o ROI, realizando procedimentos para isolar regiões do rosto. Os algoritmos e *frameworks* utilizados para a identificação da face, em grande parte dos casos, também são utilizados para o recorte da região de interesse, como efetuado por Chirra, Uyyala e Kolli (2019), que limitaram como área de análise os elementos oculares. Para isolar tais pontos, os autores utilizaram Viola-Jones para detecção ocular, mesmo procedimento adotado por Suresh et al. (2021) e por Dasgupta, Rahman e Routray (2019). Viola-Jones também foi utilizado por Indrabayu, Mufti e IntanSariAreni (2019), mas com o ROI na região da boca.

Hashemi, Mirrashid e Shirazi (2020) fizeram uso do algoritmo de Kazemi e Sullivan para obtenção dos *landmarks* da face na identificação dos olhos. O mesmo algoritmo, de Kazemi e Sullivan, também foi utilizado por Phan et al. (2021) para extração dos *landmarks*, no entanto, o ROI foi delimitado tanto para os olhos quanto para a boca. Ainda para extração de *landmarks* da face, Rajkumarsingh e Totah (2021) utilizaram a biblioteca *Google Face API*, e Jabbar et al. (2020), o *framework Dlib*. Sinha, Aneesh e Gopal (2021) exploraram o uso desse *framework* em uma de suas metodologias. Nessa etapa, existem,

ainda, procedimentos elaborados pelos próprios autores para obtenção do ROI, tal como realizado por Indrabayu, Mufti e IntanSariAreni (2019) para isolar a região da boca.

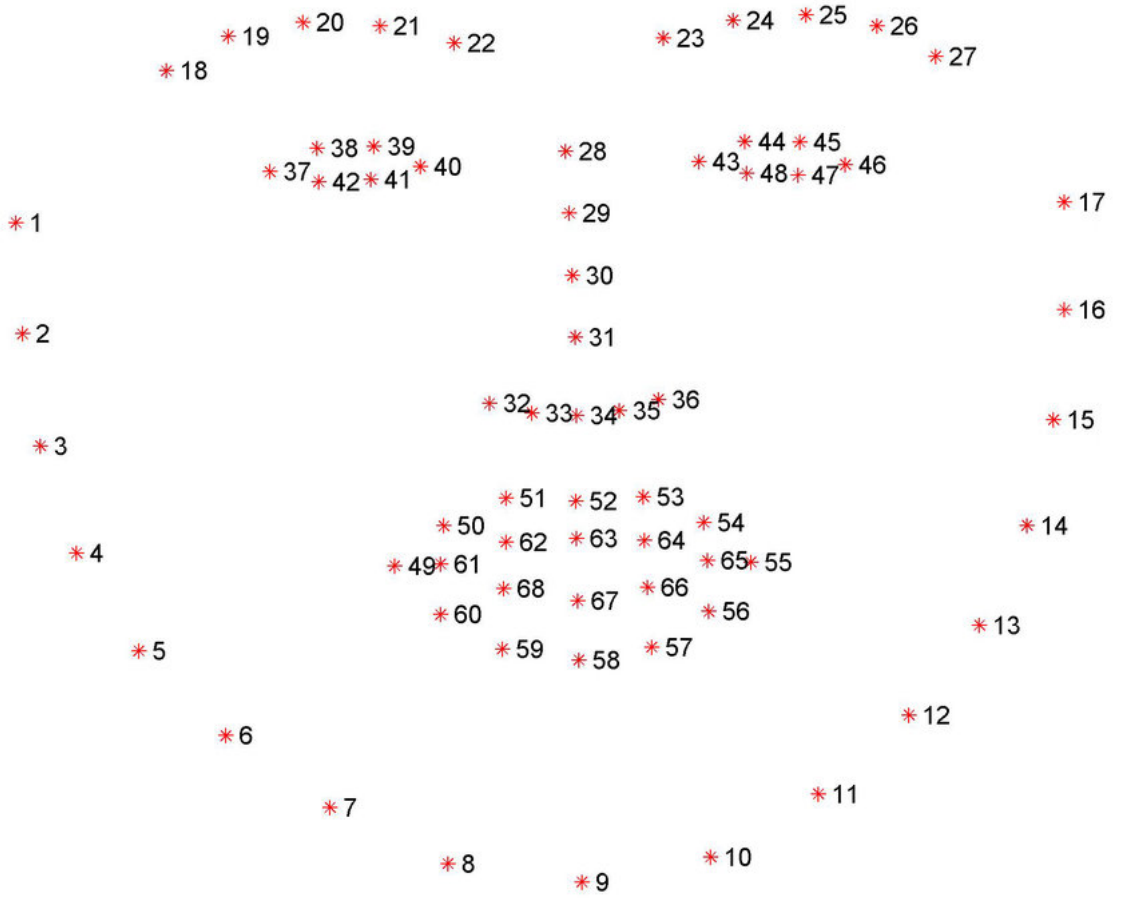


Figura 2.1: Modelo *landmark* 68 da face - Extraído de: Real-time 3D morphable shape model fitting to monocular in the wild videos - Scientific Figure on ResearchGate [accessed 15 May, 2022]

2.1.2 Classificação de sonolência

Após a obtenção do ROI, são comuns a elaboração, treinamento e teste de um modelo CNN para extrair, a partir da região de interesse, as características de sonolência. Diferentes tipos de CNN podem ser utilizadas com essa finalidade, desde redes simples com 3 camadas (Suresh et al. (2021)), 4 camadas (Chirra, Uyyala e Kolli (2019)) ou 5 camadas (Jabbar et al. (2020)), até redes mais complexas como *FD-NN*, *VGG16*, *VGG19* (Hashemi, Mirrashid e Shirazi (2020)), *MobileNet-SSD* (Dipu et al. (2021)), *LeNet* modificada (Sinha, Aneesh e Gopal (2021)), *ResNet-50V2* e *MobileNet-v2* (Phan et al. (2021)). Outra abordagem possível, nesse caso, é o uso de *Support Vector Machine* (SVM) (Dasgupta, Rahman e Routray (2019)).

Esses modelos são treinados em bases de dados selecionadas, visando identificar diferentes características propostas pelos autores. Uma possibilidade é a de identificação do estado dos olhos, entre aberto e fechado (Hashemi, Mirrashid e Shirazi (2020), Suresh et al. (2021) e Dasgupta, Rahman e Routray (2019)). Outra possível metodologia realiza a detecção de atributos gerais de sonolência em toda a face (Sinha, Aneesh e Gopal (2021), Phan et al. (2021) e Jabbar et al. (2020)). Outra possibilidade é a de detecção de características gerais de sonolência em regiões específicas, como os olhos (Chirra, Uyyala e Kolli (2019)) e olhos e boca (Dipu et al. (2021)).

Observa-se que, nesta etapa, as abordagens que realizaram o treinamento do modelo de forma a identificar se o olho está aberto ou fechado, a partir de bases de dados de pessoas com os olhos alternando entre esses estados, são limitadas para a identificação de estados intermediários, tais como olhos entreabertos, mas que também podem representar sonolência. De maneira similar, os procedimentos que realizam o treinamento de forma a classificar a abertura da boca também estão sujeitos a falso-positivos, uma vez que existem expressões naturais do motorista que apresentam movimento de abertura da boca, como, por exemplo, no caso em que ele estiver conversando.

2.2 Critérios para a consideração de sonolência

Na ultima etapa, é realizado um pós-processamento dos dados para a categorização final do motorista entre os estados ”acordado”ou ”sonolento”. Com esse propósito, uma possibilidade é considerar o condutor sonolento se, em qualquer *frame* processado, forem identificados atributos de fadiga (Chirra, Uyyala e Kolli (2019), Phan et al. (2021) e Sinha, Aneesh e Gopal (2021)). Outra abordagem, que pode ser utilizada quando é possível identificar os estados dos olhos, é a verificação da porcentagem de tempo em que os olhos ficam abertos em um período determinado (PERCLOS) (Dasgupta, Rahman e Routray (2019)). Caso o PERCLOS ultrapasse um *threshold*, o motorista é caracterizado como sonolento. Para distinção entre as condições de abertura de um olho, uma possibilidade é o cálculo da *EAR* (Phan et al. (2021)), que quantifica o nível de abertura a partir da proximidade entre pontos distribuídos nas bordas das pálpebras, conforme indicado nas Figuras 2.2 e 2.3.

Outra possibilidade, seguida caso não seja realizada a modelagem de um classificador, é, a partir dos *landmarks* extraídos, calcular taxas de abertura, ocular e bucal, para que se considere o estado de sonolência (Phan et al. (2021)). Também é possível realizar a contagem de *frames* em que a boca está aberta (Indrabayu, Mufti e IntanSariAreni (2019)) ou em que os olhos estão fechados (Hashemi, Mirrashid e Shirazi (2020), Dipu et al. (2021)) para que, caso ela ultrapasse um valor pré-definido, seja considerada fadiga.

Vale ressaltar que abordagens que exigem uma duração mínima para tomada de decisão ou para que se obtenha dados suficientes para estimação de uma taxa, tal como o PERCLOS ou mesmo a duração do bocejo, impedem que o processo seja realizado em tempo real. Nota-se, ainda, que a metodologia para a categorização de sonolência, aliada ao ROI escolhido, pode resultar em uma abordagem demasiadamente simplificada. Como exemplo, se, em uma abordagem, deseja-se determinar se o motorista está sonolento a partir da análise do bocejo isoladamente, são desconsiderados outros fatores que podem ser relevantes para a identificação de fadiga, tais como o estado dos olhos. Da mesma maneira, ao escolher a face toda como região de análise, podem ser levados em consideração elementos da face que não influenciam na sonolência, o que pode reduzir a eficiência da análise. Assim, fica clara a existência de uma solução de compromisso entre as regiões de interesse escolhidas somadas ao processamento para categorização do estado de fadiga e a eficiência da identificação.

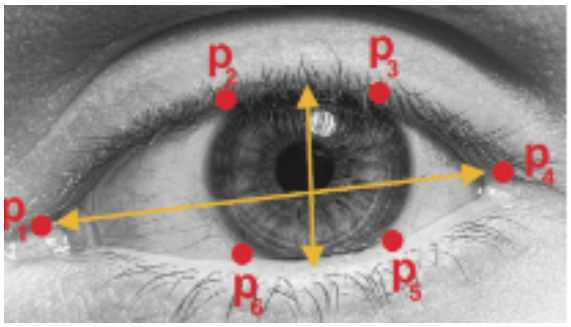


Figura 2.2: *landmarks* pra cálculo do EAR - extraído de Phan et al. (2021)

$$EAR = \frac{(|p_2 - p_6| + |p_3 - p_5|)}{(2|p_1 - p_4|)}.$$

Figura 2.3: Cálculo do EAR. Tende a 0 quando olho está fechado - extraído de Phan et al. (2021)

2.3 Considerações finais

Na tabela 2.1 é apresentado um resumo dos métodos de detecção de sonolência de acordo com a região de interesse (ROI) e o método de classificação empregado. Vale ressaltar que os resultados obtidos pelos autores estão condicionados às bases de dados utilizadas em cada trabalho e são produto da maneira que os testes foram conduzidos.

Tabela 2.1: *Comparação entre os métodos de detecção de sonolência*

ROI	Classificação	Categorização	Resultado	Autores
Olhos	CNN com 4 camadas de convolução para determinação de sono	Resultado da classificação a cada frame	96.42%	Chirra, Uyyala e Kolli (2019)
	Em olhos abertos ou fechados, com SVM	Quando o PERCLOS ultrapassa o threshold	93.33%	Dasgupta, Rahman e Routray (2019)
	Em olhos abertos ou fechados com CNN de 3 camadas	Contagem de frames em que foi classificado com olhos fechados	86.05	Suresh et al. (2021)%
	Em olhos abertos ou fechados com as arquiteturas FD-NN, VGG16 e VGG19	Olhos estiverem fechados por mais de 2 segundos	95% com TL-VGG19, 95.45% com TL-VGG16 e 98.15% com FD-NN	Hashemi, Mirrashid e Shirazi (2020)
Toda face	MobileNet-SSD em bases de dados que expõem a fadiga em função dos olhos e boca	Resultado da classificação em cada frame	Cerca de 97%	Dipu et al. (2021)
	Em características sonolentas, com CNN LeNet modificada	Utilização direta do resultado da classificação	97%	Sinha, Aneesh e Gopal (2021)
	Em características sonolentas, com CNN com 5 camadas	Resultado da classificação em cada frame	83.33%	Jabbar et al. (2020)
	ResNet-50V2 e MobileNet-v2 detectar sono	Resultado direto da classificação do frame	96% com MobileNet-V2 e 97% com ResNet-50V2	Phan et al. (2021)
Olhos e boca	Não utiliza um classificador	Contagem dos frames em que o EAR e a abertura da boca ultrapassam o threshold para categorizar como sono	83%	Phan et al. (2021)
	Google Vision API para obtenção do estado dos olhos e boca	Principalmente, através da frequência de bocejo e duração da piscagem	93.3%	Rajkumarsingh e Totah (2021)
Boca	Não utilizou um modelo.	Boca aberta durante 7 segundos	88.7%	Indrabayu, Mufti e IntanSariAreni (2019)

Capítulo 3

Metodologia

Métodos que dispensam o treinamento de um modelo de aprendizado para determinação de sonolência utilizam *thresholds* fixos, tais como a razão entre a largura e a altura da região ocular, para determinação do estado do olho (aberto ou fechado), ou de um procedimento análogo para determinação do bocejo. Tais abordagens tornam-se pouco robustas quando são levadas em consideração variações entre as pessoas, já que expressões e características faciais são diferentes para cada indivíduo. Além disso, tendo-se em vista que a base de dados a ser tratada é composta por uma grande quantidade de dados, todos não estruturados (vídeos e imagens), e que Phan et al. (2021) obteve bons resultados com a arquitetura elaborada (conforme destacado na tabela 2.1), propõe-se o uso dessa arquitetura, a *MobileNet-V2 modificada*, para extração de diferentes características que indiquem sonolência.

A metodologia do projeto pode ser separada em duas etapas: uma relativa ao treinamento do modelo e outra relativa à aplicação embarcada, conforme observado na figura 3.1.

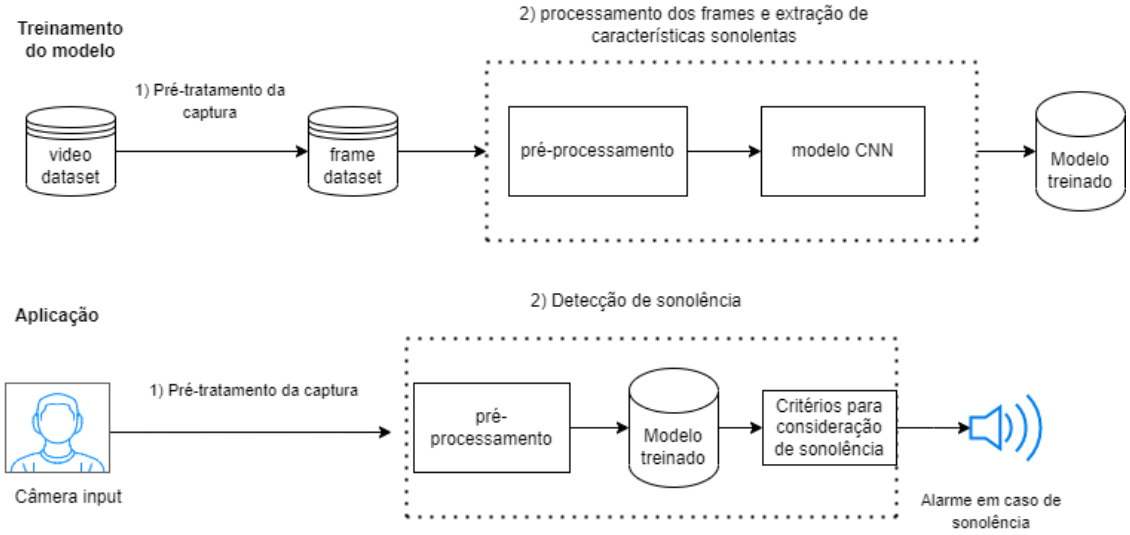


Figura 3.1: Modelo da metodologia utilizada para detecção de sonolência

3.1 Requisitos de projeto

Para que a seleção da metodologia fosse apropriada, de maneira que o projeto cumprisse adequadamente seus propósitos, a solução desenvolvida foi avaliada mediante as seguintes exigências (agrupadas a seguir em Requisitos Funcionais e Requisitos Não Funcionais):

3.1.1 Requisitos Funcionais:

- **Detectar a condição da(o) motorista**: o algoritmo implementado na aplicação deve ser capaz de interpretar as entradas, que são imagens (*frames*) coletados por uma câmera que monitora a região da face da(o) motorista, e retornar como saídas valores binários indicando seu estado: ”Acordado”ou ”Sonolento”;

- **Emitir aviso**: o aplicativo deve ser capaz de emitir alguma sinalização, seja ela sonora, através dos auto-falantes, ou visual, através da tela do *smartphone* que executa a aplicação, para alertar e despertar a(o) motorista em caso de sonolência;

- **Ser robusto**: o aplicativo deve ser capaz de operar satisfatoriamente em diferentes condições de iluminação (tanto na alta claridade do dia quanto na escuridão parcial da noite), a depender da qualidade e resolução da câmera do *smartphone* utilizado. A operação em condições não ideais de orientação e posicionamento da câmera, inclinação e movimentação da cabeça da(o) motorista e vibração, trepidação ou balanço do automóvel não será abordada para o desenvolvimento inicial do projeto e prova de conceito, sendo deixada para trabalhos futuros.

- **Ser inclusivo**: o algoritmo de detecção deve ser capaz de desempenhar sua função independentemente do gênero, etnia ou características fenotípicas naturais da(o) motorista e do fato desta(e) utilizar ou não acessórios na região do rosto (como óculos, por exemplo), contanto que a câmera possa capturar a região dos olhos e da boca sem obstruções visuais.

3.1.2 Requisitos Não Funcionais:

- **Compatibilidade para *Smartphones***: a aplicação deve ser compatível para instalação e execução em *smartphones* que possuam câmera, tela e auto-falante (para soar o sinal de alarme); desse modo, ela poderá ser empregada em quaisquer automóveis, caminhonetes e caminhões.

- **Velocidade de Execução / Tempo de Resposta**: a aplicação deve ser executada em tempo real. Sundfør, Sagberg e Høye (2019) citam estudos que mostram que dois segundos de falta de atenção são o tempo limite antes que o risco de eventos críticos aumente substancialmente. Determina-se, portanto, um limite máximo de 1 segundo para a resposta da aplicação desenvolvida, de forma que ela cumpra sua funcionalidade como dispositivo de segurança;

- **Desenvolvimento Legal**: o projeto e desenvolvimento da aplicação deve respeitar a Política de Proteção de Dados cabível (no caso do Brasil, a LGPD - Lei Geral de Proteção de Dados Pessoais), de modo que os dados utilizados devem ser oriundos somente de fontes em conformidade com ela e mediante autorização prévia.

3.2 Organização do *dataset*

Para o treinamento da CNN, é requisito fundamental um conjunto de dados ("*dataset*") adequado, de modo que a qualidade do *dataset* empregado e sua semelhança em relação às condições de uso reais da aplicação determinam a assertividade final obtida.

Construir um *dataset* que contenha boa variedade de voluntários, retrate diferentes comportamentos , diferentes condições ambientais e inclua variações compatíveis com níveis de sonolência de motoristas, por si só, constitui uma grande e complexa tarefa. Dado isso, optou-se pela adoção de *datasets* já prontos e disponíveis online para fins acadêmicos e científicos, selecionando-se aqueles compatíveis com a aplicação pretendida.

Pesquisou-se tanto por vídeos quanto por fotos (quadros/*frames* unitários) representativos de pessoas em diferentes estágios de sonolência. A seguir, são apresentadas as principais bases encontradas.

3.2.1 The ULg Multimodality Drowsiness Database (DROZY)

A base DROZY (Electrical Engineering e Computer Science of the University of Liège (ULg) (2016)) consiste em um conjunto de dados de diferentes naturezas levantados pelo Laboratório de Exploração de Sinais e Imagens (INTELSIG), que é parte do Departamento de Engenharia Elétrica e Ciência da Computação da Universidade de Liège (ULg). Os dados foram coletados de 14 sujeitos (3 homens e 11 mulheres) saudáveis em condições de crescente privação de sono. Cada sujeito realizou 3 testes de vigilância psicomotora (PVC) seguidos, cada um com duração de 10 minutos. Tempo entre estímulo e resposta dos participantes, sinais de Polissonografia (PSG), *landmarks* da face e vídeos do *Kinect v2* são exemplos de informações presentes na base. No início de cada teste, o participante atribuí uma nota, de acordo com a Escala de Sonolência *Karolinska* (KSS), que varia de 1 (situação de extremo alerta) a 9 (estado de extrema sonolência, em que é necessário alto esforço para permanecer acordado). O acesso à base é livre, mas limita-se a pesquisas, não sendo permitido seu uso para fins comerciais.

Figura 3.2: Exemplos de *frames* coletados de vídeos da base DROZY - Extraído de: The ULg Multimodality Drowsiness Database [accessed 25 May, 2022]

Tabela 3.1: Escala de Sonolência *Karolinska (KSS)*

Pontuação	Significado
1	Extramente alerta
2	Muito alerta
3	Bastante alerta
4	Alerta
5	Nem alerta nem sonolento
6	Alguns sinais de sonolência
7	Sonolento, mas sem dificuldades para ficar acordado
8	Sonolento, com dificuldades para ficar alerta
9	Extremamente sonolento, lutando contra o sono

3.2.2 University of Texas at Arlington Real-Life Drowsiness Dataset (UTA-RLDD)

O Conjunto de dados de sonolência da vida real da Universidade do Texas em Arlington (UTA-RLDD) (Ghoddoosian, Galib e Athitsos (2019)) consiste em um banco de dados que contém diferentes estágios de sonolência, desde estados extremos e visíveis até casos em que micro-expressões são fatores discriminantes. A base contém 30 horas de vídeos de 60 participantes saudáveis, com 51 homens e 9 mulheres, de diferentes etnias (sendo 10 caucasianos, 5 hispânicos não brancos, 30 indo-arianos e dravidianos, 8 do oriente médio e 7 do leste asiático), diferentes idades (20 a 59 anos com média de 25 e desvio padrão de 6), com alguns vídeos em que os participantes utilizavam óculos. Cada participante fez 3 vídeos em 3 estados: um de alerta, um em baixa vigilância e outro em sonolência, com 10 minutos cada estado. Os vídeos foram gravados pelos próprios sujeitos (com câmeras do *smartphone* ou do *notebook*) quando encontravam-se nessas circunstâncias. As distâncias de filmagem foram estabelecidas para serem semelhantes às obtidas ao se fixar o *smartphone* no painel do carro. Na descrição do *dataset*, os autores determinam os estados de atenção dos participantes pela seguinte correspondência: estado de ”alerta”para pontuações de 1 a 3 na escala KSS, estado de ”baixa vigilância”para pontuações 6 e 7, e estado de ”sonolência”para pontuações 8 e 9.

Figura 3.3: Exemplos de *frames* coletados de vídeos da base UTA-RLDD - Extraído de: UTA Real-Life Drowsiness Dataset [accessed 01 June, 2022]

3.2.3 NHTU Driver Drowsiness Detection Dataset

Este conjunto de dados foi coletado pelo Laboratório de Visão Computacional da NTHU (Ching-Hua Weng (2016)) e consiste em vídeos de 36 sujeitos de diferentes etnias, gravados em um ambiente de simulação de condução que inclui assento, pedais e volante, enquanto estes praticavam em um simulador de direção veicular. A base contém vídeos de participantes com e sem o uso de óculos sob uma variedade de cenários simulados de direção, incluindo condução normal, bocejos, piscadas lentas, sonolência, risadas etc., sob condições de iluminação diurna e noturna.

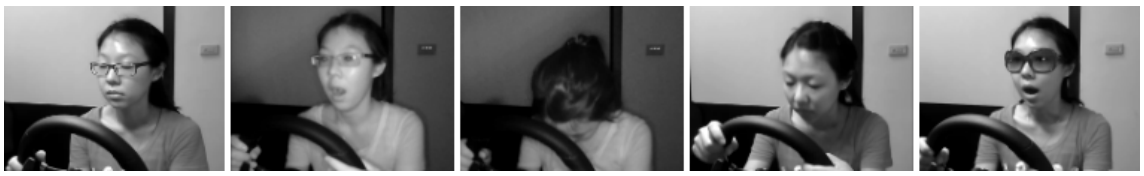


Figura 3.4: Exemplos de *frames* coletados de vídeos da base NTHU - Extraído de: Driver Drowsiness Detection Dataset [accessed 05 June, 2022]

3.3 Pré-tratamento da captura

O processo de pré-tratamento do vídeo ou imagem capturados pela câmera consiste em rastrear e recortar, em cada *frame* da captura, a região correspondente ao rosto da(o) motorista. Essa etapa é fundamental para eliminar elementos externos não desejados, que não compõem as expressões faciais e nem mesmo aspectos comportamentais da(o) motorista, e que podem gerar ruídos na análise do algoritmo de classificação e, consequentemente, na tomada de decisão resultante.

Para viabilizar e otimizar a etapa de captura da face, buscou-se ferramentas previamente treinadas capazes de realizar a detecção do rosto humano em tempo real. Uma dessas ferramentas é a biblioteca *Dlib*, em que uma das soluções disponíveis é a detecção de face. Sua implementação pode ser realizada de forma nativa em dispositivos *Android* utilizando C++. No entanto, através dos testes realizados, foi possível perceber que as alternativas apresentadas por essa solução ou consomem tempo de processamento em troca de maior acurácia ou são mais rápidas mas com menores acurácias, descumprindo os requisitos estabelecidos. Outra ferramenta computacional análisada foi o *MediaPipe*, *framework* de código aberto multiplataforma desenvolvido pela *Google* e que oferece diversas soluções para tarefas de visão computacional. Ao implementar algumas das soluções oferecidas, verificou-se que o *framework* oferece alta capacidade de resolução, com alta acurácia e execução em tempo real, seja em um ambiente *Desktop* ou *Android*. Testou-se diversas variações ambientais (mudança da intensidade de iluminação, alteração da distância da câmera e da posição da cabeça, uso de óculos, etc.) para validação do

framework nos propósitos da aplicação deste projeto. Em todos os cenários, o *MediaPipe* apresentou superioridade em relação ao *Dlib* em termos de custo computacional e de acertabilidade, e, portanto, foi a ferramenta escolhida para atender aos requisitos. Vale ressaltar que os testes referentes ao *Dlib* foram realizados somente em ambiente *Desktop*, enquanto que o *MediaPipe* foi testado tanto em *Desktop* quanto em *Android* e apresentou elevada acertabilidade com baixo custo computacional, em ambos os casos superiores ao apresentado pelo *Dlib*.

Dentre as várias soluções oferecidas pelo *MediaPipe*, a que se mostrou mais adequada e vantajosa para os propósitos do projeto é a "*Face Detection*"; as vantagens oferecidas contemplam:

- (i) **velocidade ultrarrápida**, garantida pela base *BlazeFace*, que usa uma rede de extração de recursos leve e inspirada na *MobileNetV1/V2*, um esquema de âncora compatível com GPU modificada do *Single Shot MultiBox Detector (SSD)* e uma estratégia alternativa de resolução de empate aprimorada para supressão não máxima (Google (2019)), sendo computacionalmente econômica, com elevado desempenho e adaptada para GPU's de dispositivos móveis (Bazarevsky et al. (2019));

- (ii) **simplicidade**, já que o retorno gerado pelo "*Face Detection*" representa cada face detectada como um simples conjunto de 6 pontos-chave ("*key-points*") [1: olho direito, 2: olho esquerdo, 3: ponta do nariz, 4: centro da boca, 5: tragus da orelha direita e 6: tragus da orelha esquerda] dentro de uma caixa limitadora ("*bounding box*"); e

- (iii) **flexibilidade**, pois a biblioteca do pacote *MediaPipe* como um todo pode ser instalada e utilizada para aplicativos *Android*, bem como para outras plataformas, como *iOS* e *Desktop*, além de possuir, neste último ambiente, suporte em *python*, o que permite a implementação por meio do *Google Colab*, que, por sua vez, facilita o trabalho colaborativo, compartilhamento e processamento em nuvem.

A partir das coordenadas das extremidades da *bounding box* obtidas pelo algoritmo "*Face Detection*" do *MediaPipe*, é possível recortar a região que engloba, com boa precisão, a face da pessoa filmada. Entretanto, a depender do método e das variáveis de classificação adotados, pode ser necessário selecionar uma área menor dentro da *bounding box*: uma região que contemple somente a faixa dos olhos, ou apenas um único olho, ou ambos os olhos e a boca, ou apenas a boca, e assim por diante.

No caso deste projeto em particular, decidiu-se utilizar toda a face do motorista como ROI (Região de Interesse). Dessa forma, características de sonolência podem ser extraídas dos olhos, dos movimentos e inclinação da cabeça, das sobrancelhas, da boca e de

expressões faciais, não limitando a solução a atender somente um desses critérios. Além disso, dessa forma se reduz a necessidade de acrescentar processamento para realizar recortes extras, diminuindo o uso de recursos computacionais, o que se torna especialmente importante na aplicação embarcada.

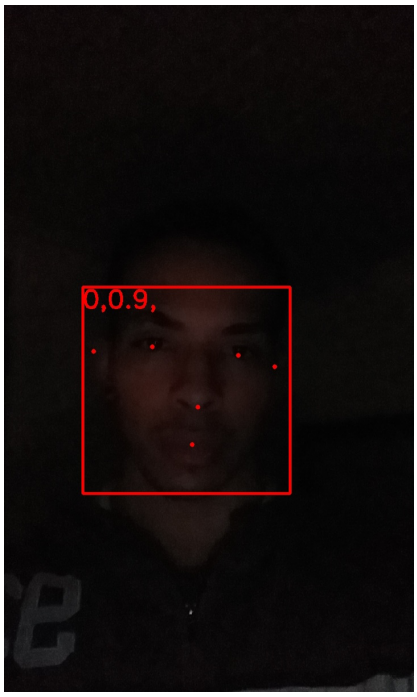


Figura 3.5: Exemplo de uso do *MediaPipe* em baixas condições de iluminação

3.4 Modelo

Para realizar a classificação de sonolência, escolheu-se a arquitetura *MobileNet-v2* com adição de algumas camadas, como pode-se visualizar na figura 3.6. Segue-se o mesmo modelo utilizado por Phan et al. (2021), com somente algumas alterações, como utilização de somente um neurônio para classificação final e mudança da dimensão de entrada dos dados. A escolha da arquitetura base *MobileNet-v2* foi realizada tendo em vista que o número de parâmetros utilizados é significativamente reduzido quando comparado com convoluções regulares com a mesma profundidade de rede, o que resulta em CNNs mais leves, diminuindo o custo computacional, fenômeno que se deve à *depthwise separable convolution* (Sandler et al. (2018)). Ademais, utiliza-se o modelo com pesos pré-treinandos na *ImageNet* (*ImageNet* (2022)) a fim de melhorar a eficácia da classificação e reduzir o tempo de treino, além de que, por conter pesos treinados com uma grande quantidade de dados, os pesos das camadas iniciais, responsáveis por detecções de características mais básicas, tais como arestas e vértices, podem ser totalmente reaproveitados, o que aumenta a *performance* do modelo.

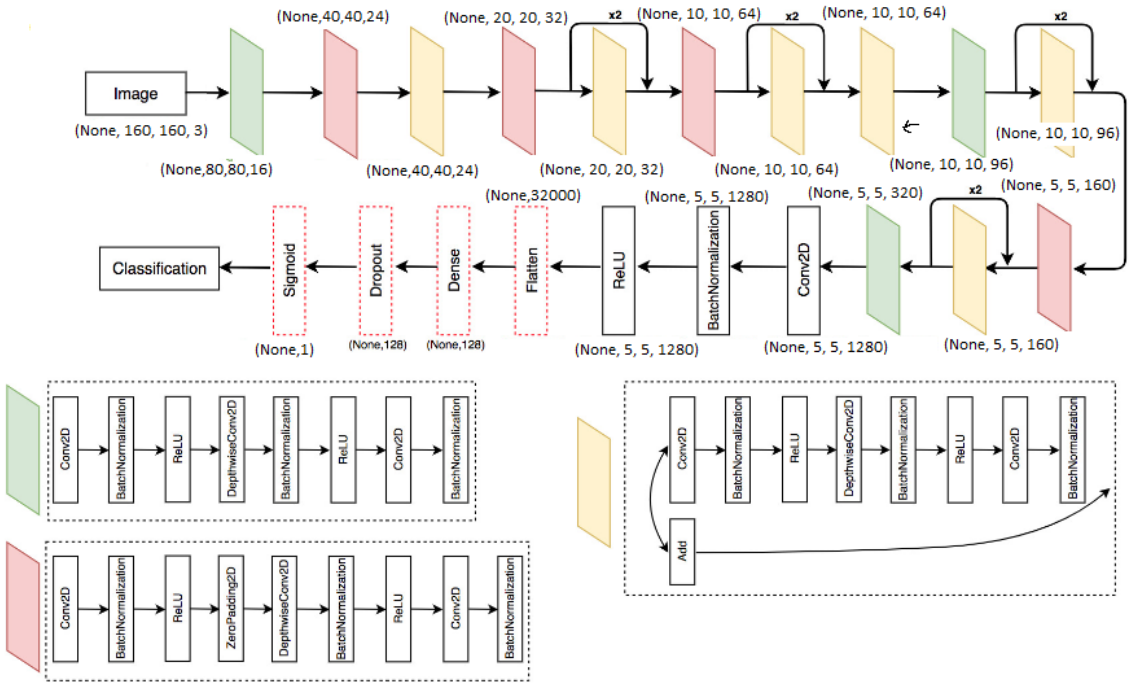


Figura 3.6: Arquitetura utilizada para a classificação de sonolência - Adaptado de Phan et al. (2021)

3.5 Instrumentos

Realiza-se o treinamento do modelo no ambiente do *Google Colab*, utilizando-se GPU como *Hardware accelerator*, em um *Notebook Acer Nitro 5*, com um processador *Intel Core i5-7300HQ*, placa de vídeo *NVIDIA GeForce GTX 1650* com com $4GB$ de memória *gDDR5* e $8GB$ de memória *RAM*. A aplicação é desenvolvida no *Android Studio Chipmunk Patch 2* (2021.2.1). Já os dispositivos *Android* em que a aplicação é testada, neste projeto, são: *Galaxy J8* e *Xiaomi Mi A3*. Os testes são realizados em situações correspondentes às reais, em que o motorista está na posição de condução do veículo, com o celular fixado em um suporte na posição adequada do painel do veículo, em distintos cenários de iluminação. Ressalva-se que, por questões de segurança e dado o escopo do projeto, o veículo permanece parado durante os testes.

Capítulo 4

Treinamento do modelo

Para o treinamento do modelo escolhido, é necessário transformar a base de dados, que é composta por um conjunto de vídeos, em *frames* organizados. Posteriormente, os *frames* são pré-processados e utilizados como entrada para o classificador.

4.1 Pré-processamento da base de dados

Foi escolhida para a realização do treino a base de dados *NHTU Driver Drowsiness Detection Dataset* (Ching-Hua Weng (2016)), pois ela contém filmagens que representam situações de condução simulada, muito próximas da aplicação real. Além disso, outras características também se destacam e justificam a escolha. A primeira é a maneira como a câmera é posicionada relativamente ao condutor, de forma equivalente a um cenário real, com distância e orientação fixadas devido ao uso de um suporte em todas as filmagens. A segunda é a diversidade de participantes, de aparências variadas e em diferentes situações de sonolência, representadas em trechos significativos de vídeo. Assim, haja vista que as demais bases não possuem esse controle preciso da posição da câmera - algumas, inclusive, contém obstruções, tais como eletrodos - e que os trechos de sonolência não são tão bem retratados quando comparados com os apresentados na NHTU, a NHTU se mostrou a melhor opção.

A estrutura em que os vídeos estão dispostos na base de dados NHTU está representada na figura 4.1, em que os arquivos de texto indicados contém a classificação de sonolência a cada *frame* (1 para sonolência e 0 o caso contrário).

```
.
└── NTHU/
    ├── Testing_Dataset/
    │   ├── {subject}_{scenario}_mix.mp4 (70 videos of subjects in different scenarios)
    │   └── test_label_txt/
    │       ├── nightglasses/
    │       │   └── {subject}_nightglasses_mixing_drowsiness.txt (14 text files)
    │       ├── glasses/
    │       │   └── {subject}_glasses_mixing_drowsiness.txt (14 text files)
    │       ├── sunglasses/
    │       │   └── {subject}_sunglasses_mixing_drowsiness.txt (14 text files)
    │       ├── noglasses/
    │       │   └── {subject}_noglasses_mixing_drowsiness.txt (14 text files)
    │       └── night_noglasses/
    │           └── {subject}_night_noglasses_mixing_drowsiness.txt (14 text files)
    └── Training_Evaluation_Dataset/
        ├── Training Dataset/
        │   └── {subject} ( 18 folders of subjects, and, for each: )/
        │       └── {scenario} (4 scenarios folder: nightglasses, glasses, sunglasses, noglasses and night_noglasses, for each:)/
        │           ├── yawning.avi
        │           ├── slowBlinkWithNodding.avi
        │           ├── sleepyCombination.avi
        │           ├── nonsleepyCombination.avi
        │           ├── {subject}_yawning_drowsiness.txt
        │           ├── {subject}_slowBlinkWithNodding_drowsiness.txt
        │           ├── {subject}_sleepyCombination_drowsiness.txt
        │           └── {subject}_nonsleepyCombination_drowsiness.txt
        └── Evaluation Dataset/
            └── {subjects} (4 subjects, and, for each:)/
                ├── {subject}_{scenario}_mix.mp4
                └── {subject}_{scenario}_mixing_drowsiness.txt
```

Figura 4.1: Estrutura dos dados da base NTHU (Ching-Hua Weng (2016))

No entanto, para os propósitos desse projeto, deseja-se que os dados estejam em uma estrutura semelhante à representada na figura 4.2, em que cada uma das pastas (de treino, de teste e de avaliação) contenha duas subpastas que dividam os conjuntos de imagens entre os possíveis estados (acordado ou sonolento) e, dentro de cada uma, haja os *frames* representativos da condição, já recortados com somente a região da face dos sujeitos.

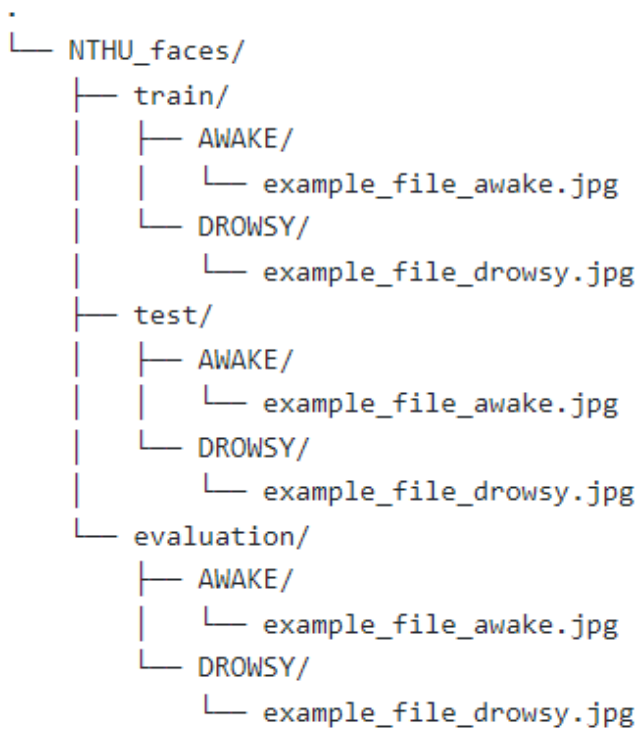


Figura 4.2: Estrutura de dados desejada

Dessa maneira, os procedimentos utilizados para realizar o recorte das regiões das faces dos sujeitos, nos *frames* da base NTHU e os armazenar na estrutura de dados desejada (4.2) são ilustrados nos algoritmos 1, 2 e 3 a seguir. Observa-se que foi coletado um a cada 18 *frames*, sendo esta a quantidade determinada, de forma empírica, para que quadros consecutivos não sejam praticamente idênticos, o que reduz a quantidade de imagens processadas e, consequentemente, o custo computacional para realizar o treinamento.

Após a coleta dos recortes das regiões das faces dos motoristas em cada vídeo, optou-se pela seleção aleatória de imagens: para o uso da maior quantidade de quadros de diferentes participantes e para melhorar a *performance* do modelo, de forma que maior variedade étnica fosse coberta, adotou-se um fluxo no qual todos os quadros disponíveis são carregados e é realizada uma seleção randômica de quadros para treino, teste e validação. Escolheu-se essa estratégia considerando-se que a base de treino contém uma diversidade limitada de pessoas. Optou-se, ainda, por não realizar transformações - tais como rotações - nas imagens para o treino, para que a inclinação da cabeça seja intencionalmente levada da em consideração na caracterização de sonolência.

4.2 Construção do modelo

O modelo apresentado na figura 3.6 foi construído em *python*, no ambiente do *Google Colab*, com o *framework TensorFlow*. O procedimento realizado para sua construção é ilustrado no algoritmo 4.

4.2.1 Conversão do modelo

Uma vez treinado, o modelo pode ser convertido para o formato otimizado *FlatBuffer* com o auxilio do *TensorFlow* (*FlatBuffers* (2022)), que é suportado pela API do *TensorFlow* no *Android*. O procedimento de conversão pode ser observado na figura 4.3.

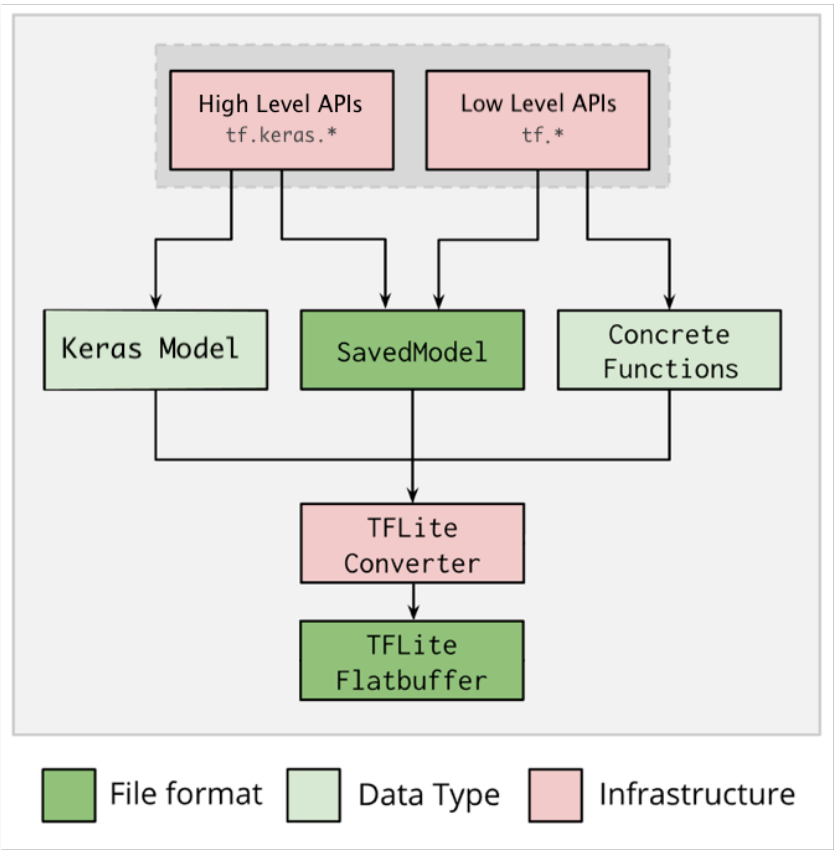


Figura 4.3: Conversão do modelo - Extraído de: Converter modelos do TensorFlow [accessed 12 July, 2022]

Algorithm 1 Algoritmo de recorte da região da face e armazenamento na estrutura desejada para os dados de treino

```
Store a list of all possible scenarios (glasses, night noglasses, night glasses, no glasses, sunglasses)
Store a list of all possible states (nonsleepyCombination, sleepyCombination,slowBlinkWithNodding, yawning )
Store a list of all subjects for training        ▷ Each folder in training directory has a number of the subject
for each subject do
    for each scenario do
        for each state do
            Get the video path using the current state, subject and scenario
            Load the label file
            Start mediapipe
            Get the current frame
            if Frame_number mod 18 then        ▷ Collecting a frame every 18 frames
                Cutout of the driver's face from the current frame (ROI)
                Resize face cut in (160, 160)
                if Label for frame is drowsy then
                    Save the ROI in /train/drowsy folder
                else
                    Save the ROI in /train/awake folder
            end if
        end if
    end for
end for
end for
```

Algorithm 2 Algoritmo de recorte da região da face e armazenamento na estrutura desejada para os dados de validação

```
Store a list of all possible scenarios (glasses, night noglasses, night glasses, no glasses, sunglasses)
Store a list of all subjects for training    ▷ Each folder in evaluation directory has a number of the subject
for each subject do
    for each scenario do
        Get the video path using the current subject and scenario
        Load the label file
        Start mediapipe
        Get the current frame
        if Frame_number mod 18 then
            Cutout of the driver's face from the current frame (ROI)
            Resize face cut in (160, 160)
            if Label for frame is drowsy then
                Save the ROI in /evaluation/drowsy folder
            else
                Save the ROI in /evaluation/awake folder
        end if
    end if
end for
end for
```

Algorithm 3 Algoritmo de recorte da região da face e armazenamento na estrutura desejada para os dados de teste

```
Store a list of all possible scenarios (glasses, night noglasses, night glasses, no glasses, sunglasses)
Store a list of all subjects for training  ▷ Each folder in test directory has a number of the subject
for each subject do
    for each scenario do
        Get the video path using the current subject and scenario
        Load the label file
        Start mediapipe
        Get the current frame
        if Frame_number mod 18 then
            Cutout of the driver's face from the current frame (ROI)
            Resize face cut in (160, 160)
            if Label for frame is drowsy then
                Save the ROI in /test/drowsy folder
            else
                Save the ROI in /test/awake folder
        end if
    end if
end for
end for
```

Algorithm 4 Algoritmo para construção do modelo de classificação

```
Load the basel model MobileNetV2_1.00_160 from keras with pre-trained weights from ImageNet without including the top
Create the input layer
Data preprocessing of the input using the same weights that the base model was trained on
Add data preprocess as the input layer from base model
for Layer in base model layers until layer 81 do  ▷ Freeze all layers from MobileNet until layer 81
    Freeze the layer weights
end for
Add a Flatten layer
Add a dense layer with 128 units and ReLU as activation function
Add a dropout of 0.5
Add a dense layer with 1 unit and Sigmoid as activation function to be the output
```

Capítulo 5

Aplicação para *smartphone*

Uma vez realizado o treinamento do modelo e sua respectiva conversão, a próxima etapa é embarcá-lo no dispositivo *Android*.

5.1 Estrutura do Android

A arquitetura Android pode ser dividida da seguinte maneira, de acordo com *The Android software stack* (2022) :

- (i) **Aplicativos do sistema**: Aplicativos principais contidos nos dispositivos Android, tais como serviços de e-mail, SMS, calendários, browsers, etc. Tais aplicativos podem ser invocados pelo desenvolvedor para realizar uma função específica, dispensando a necessidade de programar uma tarefa que já é realizada por um aplicativo do sistema;
- (ii) **Java API Framework**: API escrita em Java, que possibilita o uso de algumas bibliotecas nativas, e é utilizada para a criação de aplicativos. Contém diversos blocos que facilitam a reutilização de componentes e de serviços. Estão inclusos: Um sistema de visualização, em que é possível definir diversos tipos de Layouts em uma estrutura XML, bem como componentes, tais como botões, lista e outras funções relacionadas a UI; Um gerenciador de recursos, a partir do qual é possível gerir arquivos adicionais inseridos pelo desenvolvedor, tais como layouts, imagens e animações; Um gerenciador de notificações, que permite a exibição de notificações, por parte dos aplicativos, na barra de status, fora da UI do aplicativo; Um gerenciador de atividade, que permite controlar o ciclo de vida das *activities*, que são, em resumo, as telas da aplicação, sendo possível definir, através do gerenciador, como a *activitie* deve responder quando é criada, iniciada, pausada, continuada após a pausa, reiniciada e destruída. Além disso, o gerenciador de atividade

fornece uma pilha de navegação reversa, sendo possível retornar para telas anteriores em uma estrutura de pilha; Um provedor de conteúdos, que permite que os usuários acessem ou compartilhem dados de outros aplicativos.

- (iii) **Bibliotecas C/C++ Nativas**: Muitos componentes e serviços principais do Andoroid são implementados em código nativo C/C++, tais como acesso a áudios e vídeos, banco de dados e manipulação de dados 2D ou 3D. A camada Java API possibilita o acesso a funcionalidades de algumas dessas bibliotecas. Também é possível, por parte do desenvolvedor, escrever de forma nativa utilizando o Android NDK;

- (iii) **Android Runtime**: Em dspositivos com versões de API superiores a 21, cada aplicativo é executado em uma instância própria do Android Runtime (ART). O ART foi desenvolvido para que seja possível a execução de diversas quantidades de máquina virtuais em dispositivos de baixa memória que executam um formato bytecode especial para android, que é otimizado para consumir a menor quantidade de memória. O ART possui diversos recurso que melhoram o desempenho do aplicativo, tais como compilação antecipada (AOT) e Coleta de lixo (GC) otimizada;

- (iv) **Camada de abstração de hardware (HAL)**: Fornece interfaces padrões para acesso, através do Java API, dos hardwares do dispositivo. O HAL é composto por diversas biblitecas específicadas para cada hardware (como módulo de Câmera ou Bluetooth), tornando possível acessar tais componentes através de uma chamada em alto nível; e

- (v) **Linux Kernel**: É a fundamentação do sistema Android, realiza tarefas como encadeamento e gerenciamento de memória de baixo nível; fornece recursos de segurança, impedindo que informações particulares de um determinado aplicativo seja obtida ou alterada por outros; possui drivers específicos; faz o gerenciamento de processos, dentre outras funcionalidades.

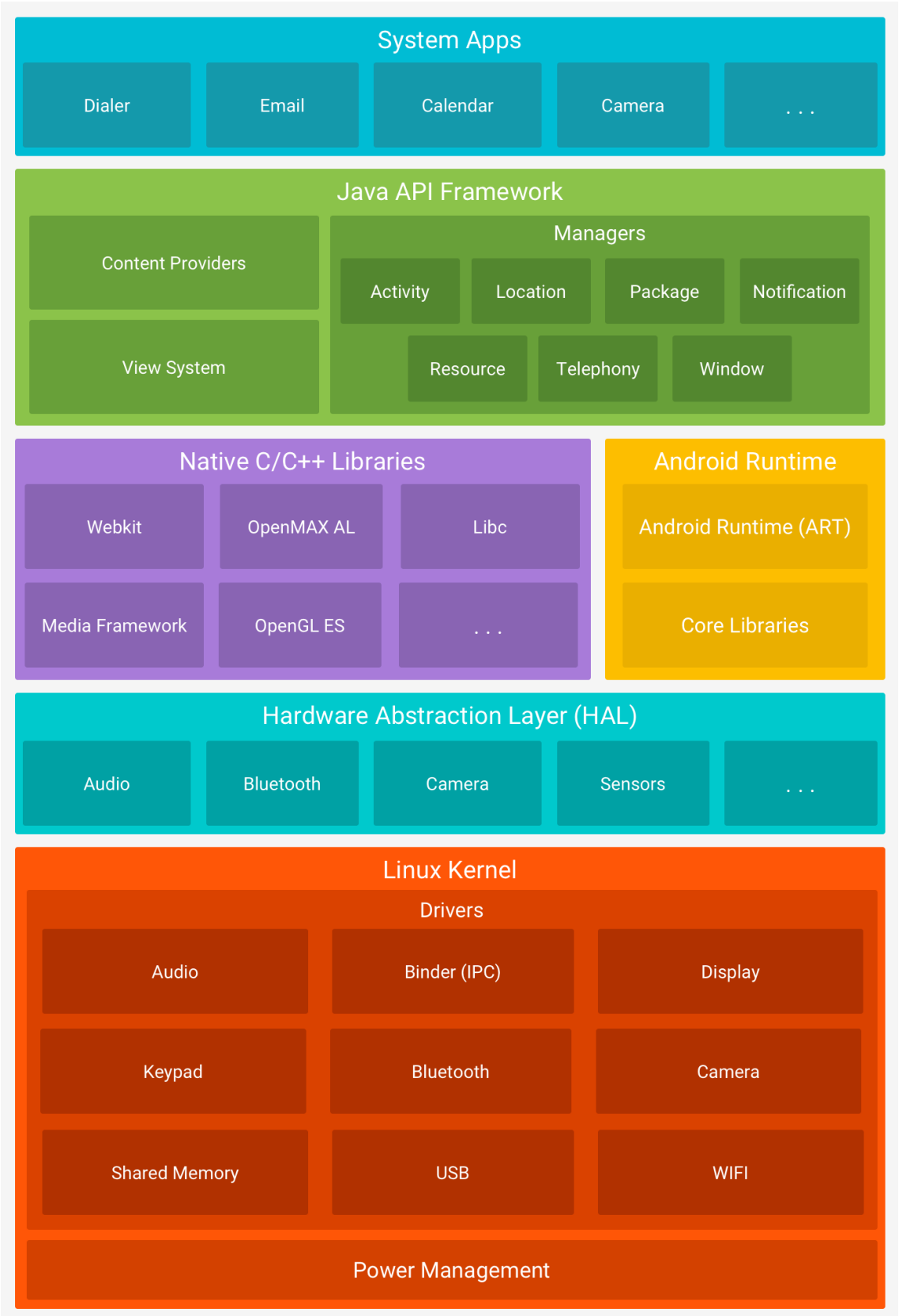


Figura 5.1: A arquitetura do sistema Android - Extraído de: Arquitetura Android [accessed 25 September, 2022]

5.2 Arquitetura do Mediapipe

Desenvolvido pela Google, o Mediapipe é uma das ferramentas centrais utilizadas para reconhecimento de imagem, de grande relevância nesse trabalho no que se refere à detecção de face. Os recursos disponíveis em sua biblioteca implementam soluções de Machine Learning prontas para várias tarefas de visão computacional. Portanto, ele se mostrou facilitador na construção e otimização de um canal de percepção de imagem, contando com opções de configuração e ferramentas de *benchmarking*.

Do ponto de vista de sua arquitetura, o processamento realizado pelo Mediapipe pode ser representado graficamente por componentes modulares, incluindo modelos de inferência, funções de processamento de mídia e demais subprocessos. A Figura 5.2 ilustra esses componentes:

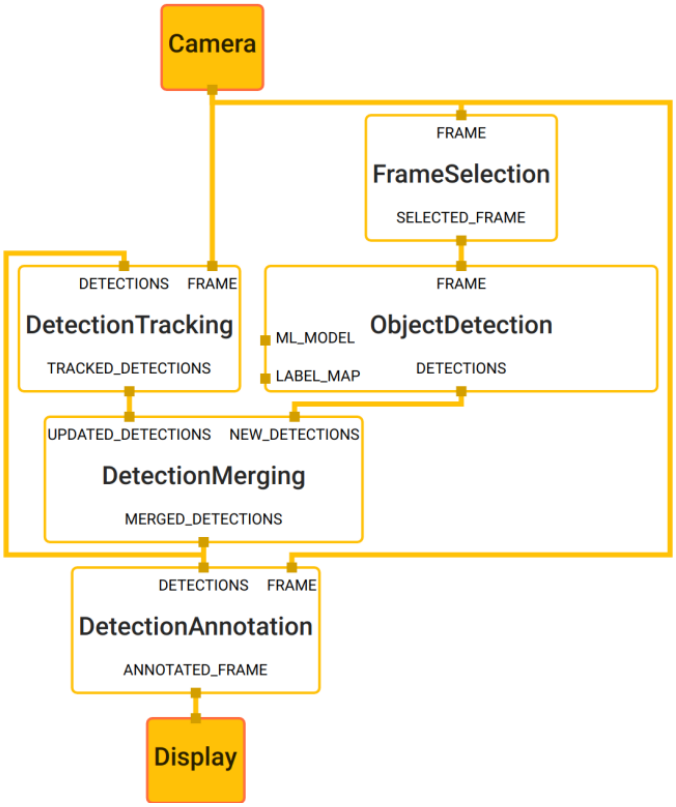


Figura 5.2: Representação esquemática das etapas de processamento realizadas pelo MediaPipe: MediaPipe: A Framework for Perceiving and Processing Reality [accessed 30 September, 2022]

O vídeo capturado pela câmera é processado em etapas sequenciais ou em cadeias paralelas, conforme apresentado na imagem. Cada etapa pode ser resumida da seguinte maneira:

- **FrameSelection**: *Frames* do vídeo são coletados de forma amostral, a uma taxa que depende do poder computacional gráfico disponível e da evolução temporal do estado da imagem (identificando se existem diferenças relevantes entre os *Frames* subsequentes);
- **ObjectDetection**: Sobre cada um dos *Frames* selecionados é executado o algoritmo de reconhecimento que identifica o objeto desejado (por exemplo, uma face) e cria a malha dos pontos de interesse sobre ele;
- **DetectionTracking**: Os *Frames* sucessivos do vídeo, rastreados, são comparados e, pelo deslocamento dos *pixels* registrados pela câmera, é calculado o movimento dos elementos presentes na região de captura — tanto deslocamento quanto deformação (por questões de perspectiva);
- **DetectionMerging**: A malha de pontos criada sobre o *Frame* selecionado é adaptada para todos demais *Frames* rastreados ajustando-se sua posição e forma obedencendo-se a relação identificada na etapa de *DetectionTracking*;
- **DetectionAnnotation**: O resultado obtido é *renderizado* e apresentado de forma sincronizada ao próprio vídeo capturado, de forma que a malha de pontos de interesse é exibida sobreposta ao vídeo.

5.3 Interface da aplicação

As telas responsáveis pelo funcionamento do sistema podem ser divididas em três principais: Uma de configuração, outra de seleção do tipo de alarme e, por fim, a tela em que ocorre o fluxo de monitoramento de sonolência.

5.3.1 Tela de configuração

A interface de configuração é responsável por estimar o número de quadros por segundo que são coletados da face do motorista pelo mediapipe. Outra função que é realizada nessa tela é a estimativa do tempo de duração do pré-processamento e classificação. A partir das amostras coletadas, as estimativas para os tempos são realizadas considerando o pior caso, em que o FPS e o tempo de classificação são máximos, de forma a considerar o atraso que é oriundo de mudanças bruscas de textura, já que esse fenômeno resulta em um aumento da taxa de coleta de frames, o que é abordado na subseção 5.3.4.

5.3.2 Fluxo básico

1. O motorista inicia o procedimento de configuração
2. A aplicação:
    a) Inicializa a câmera e renderiza os frames na tela
    b) Inicializa a detecção da face para estimar o FPS do MediaPipe
    c) Indica para o motorista quando o processo for concluído
3. O motorista avança para a próxima etapa
4. A aplicação:
    a) Retira a renderização dos frames
    b) Inicializa o classificador de sonolência na GPU
    c) Realiza a estimação do tempo de pré-processamento e classificação
    d) Indica para o motorista quando o processo for concluído
5. O motorista pode avançar para a próxima etapa
6. A aplicação enviará os dados coletados para a tela de seleção do tipo de alarme

5.3.3 Tela de seleção do tipo de alarme

Essa tela é responsável por oferecer ao motorista opções quanto ao tipo de alarme que deseja, podendo ser:

- Alarme visual: Caso seja detectada sonolência, ocorrerá, na tela do aplicativo, uma alternância de cores, previamente definidas pelo motorista nesta etapa. Sua implementação é realizada por meio de um timer que fica ativo por dois segundos após a detecção de sono. Durante o seu acionamento, a interface alterna de cores de acordo com os valores selecionados pelo usuário.
- Alarma sonoro: Caso seja detectada sonolência, será emitido um som previamente configurado. Sua implementação ocorre através do uso do *MediaPlayer Java API*, que realizará o acionamento da saída da áudio do dispositivo.
- Alarmes sonoro e visual: Caso seja detectada sonolência, ocorrerá tanto indicação visual quanto sonora

Fluxo básico

1. A aplicação exibirá para o motorista as opções de alarme sonoro e/ou visual
2. O motorista poderá prosseguir sem escolher uma opção
3. A aplicação:
    a) Selecionará a opção de alarme sonoro
    b) Realizará o envio dos dados selecionados pelo motorista e dos oriundos da tela de configuração para a tela de monitoramento

Fluxos alternativos

1. O motorista poderá selecionar a opção de alarme visual com ou sem alarme sonoro
2. A aplicação:
    a) Disponibilizará para o motorista a possibilidade de escolher duas cores quaisquer, a partir de uma palheta de cores, que alternarão entre si em caso de sonolência
3. O motorista poderá seguir para a próxima etapa
4. A aplicação:
    a) Salvará os dados selecionados pelo motorista
    b) Realizará o envio dos dados selecionados pelo motorista e dos oriundos da tela de configuração para a tela de monitoramento

5.3.4 Tela de monitoramento

Conforme ilustrado na figura 5.3, após a coleta do frame e a posterior detecção da face, é inciada uma *thread* para renderização, processamento dos resultados e a realização da predição. No entanto, essa *thread* sempre é inicializada após a detecção da face e, como sua execução é realizada em uma pilha de *threads*, o funcionamento em tempo real é comprometido. Soma-se a isso a necessidade de inclusão da variação de tempo de execução conforme a variação da textura, uma vez que é utilizado, para coleta de frames, a solução do MediaPipe, que utiliza o *Frame Selection*, conforme apresentado na seção 5.2. Dessa maneira, sabe-se que caso ocorra variação suficiente de textura para que a decisão de coleta de frame, pelo MediaPipe, seja efetuada, é aumentada a taxa de coleta de frames, o que resultará em uma renderização mais lenta na *thread* de processamento,

por consequência, o pré-processamento e classificação dos resultados é atrasado, fenômeno que ocasiona em uma propagação de atrasos, já que a *thread* seguinte somente renderizará o frame atual na tela caso não tenha nada sendo renderizado no momento, já que o objeto de renderização é um recurso a ser compartilhado .Assim, para que a renderização seja contínua, em tempo real, e não seja um limitador para a utilização do classificador, é realizada uma decisão de coleta de frames, de tal forma que todos frames utilizados pelo MediaPipe são renderizados, mas é feito um gerenciamento de quais serão utilizados para a classificação de sonolência. Esse gerenciamento é realizado através da estimativa de duração do pré-processamento e classificação e a sua posterior conversão para um número de quadros. Dessa forma, frames múltiplos dessa taxa de quadros são utilizados para a decisão de sonolência, enquanto que os demais são ignorados em relação a essa finalidade. Realizando o gerenciamento dessa maneira, o tempo corrente do usuário é levado em consideração, uma vez que os quadros que atrasarão a etapa de classificação, por conta do atraso na renderização e da crescente concorrência de processos, são, em sua maioria, ignorados, de tal forma que o pré-processamento e classificação do frame corrente ocorra quando passada a estimativa da duração do processamento do frame anterior, o que resulta em uma pequena defasagem temporal entre a última classificação no dispositivo e o tempo atual. Além disso, como a decisão se o frame será coletado ou não é feita no instante em que ele é capturado pela câmera, antes da renderização, envio pro FaceDetection e pré-processamento e classificação, garante-se que a decisão seja tomada levando em consideração o tempo corrente, impedindo que ocorram gargalos por conta de variação de textura. Para que o processo ocorra de forma satisfatória e não haja muito consumo de memória, são utilizadas as informações coletadas na etapa de configuração (5.3.1). Dessa maneira, um frame deverá ser coletado e enviado para o pré-processamento e classificação a cada $(Tempo_{pre-processamento} + Tempo_{classificacao}) * FPS_{mediapipe}$ quadros, ressaltando-se que são utilizados os valores máximos do $FPS_{mediapipe}$ (representando situação de alta taxa de coleta de frames por variação de textura) e do $(Tempo_{pre-processamento} + Tempo_{classificacao})$, de forma a prevenir a pior situação, o que se mostrou uma taxa aceitável para o processamento em tempo real. Assim, o frame coletado pela câmera e que é utilizado pelo MediaPipe sempre é renderizado, mas só é pré-processado e utilizado para classificação de sonolência de acordo com a estimativa de duração desse processo, de maneira que a sua execução é realizada somente quando terminada a anterior.

Além da estimativa da duração que a *thread* anterior possuirá, é utilizado um objeto responsável por gerenciar a classificação. Dessa maneira, caso tenha passado o tempo estimado para o processamento mas ainda existe uma classificação em progresso, a *thread* que é responsável pela renderização, pré-processamento e classificação somente reali-

zará a renderização, de forma que sempre haja somente uma etapa de pré-processamento e classificação por vez, prevenindo, além de aumentos progressivos de *delay* entre os tempos do motorista e o da aplicação, gargalos de memória e concorrência de processos.

A fase de decisão final de atribuição de sonolência para posterior disparo do alarme, por sua vez, leva em consideração minimizar a quantidade de falsos positivos sem comprometer o requisito de tempo. A partir de testes do classificador, observou-se que, por serem realizadas classificações sucessivas, ocorre uma tendência à predição errônea de sonolência durante a piscagem dos olhos. O estudo de Hashemi, Mirrashid e Shirazi (2020) indica que o tempo limite para piscagem dos olhos, em uma pessoa saudável, é de *400ms*, que foi o adotado para o projeto. Assim, de posse do tempo necessário para o pré-processamento e classificação (obtidos na etapa de configuração), sabe-se o número de quadros sucessivos necessários para considerar situação de sonolência. Dessa forma, durante a piscagem dos olhos, que possui duração máxima de *400ms*, os resultados da predição são inseridos em uma estrutura semelhante a uma fila circular. O tamanho da fila é definido pelo número de classificações que são necessárias para preencher um período de *400ms*, ou seja, $Tamanho_{fila} = \frac{Tempo_{piscagem}}{Tempo_{classificacao}}$, sendo o resultado aproximado para o inteiro mais próximo. Assim, são necessárias tantas classificações de sonolência seguidas quanto o tamanho da fila para disparar o alarme. Dessa maneira, a asserção final do estado de sonolência ao motorista leva em conta a eliminação de possíveis expressões que possam vir a serem confundidas com sonolência, como, em especial, a expressão da face humana durante a piscagem dos olhos, evitando alarmes desnecessários. Na figura 5.4 pode-se visualizar o fluxograma do processo.

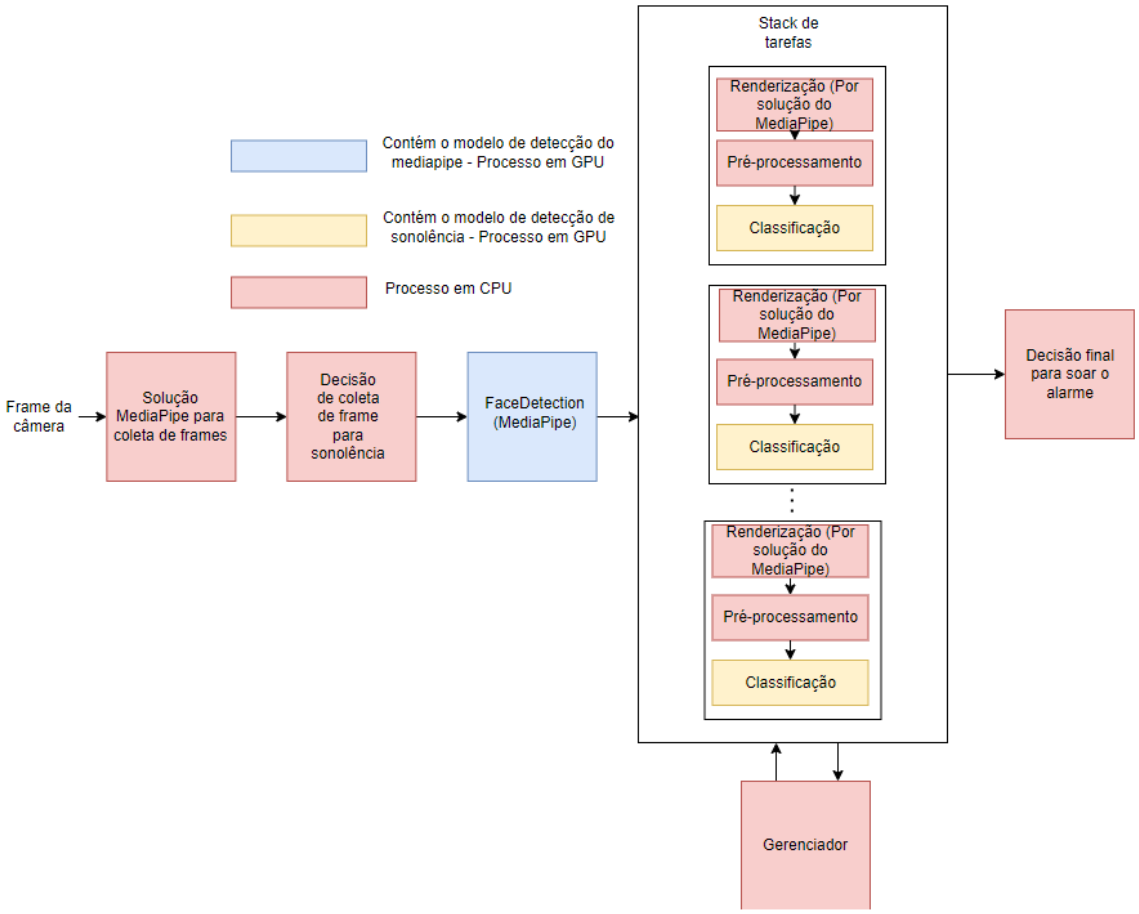


Figura 5.3: Representação da estrutura principal do monitoramento

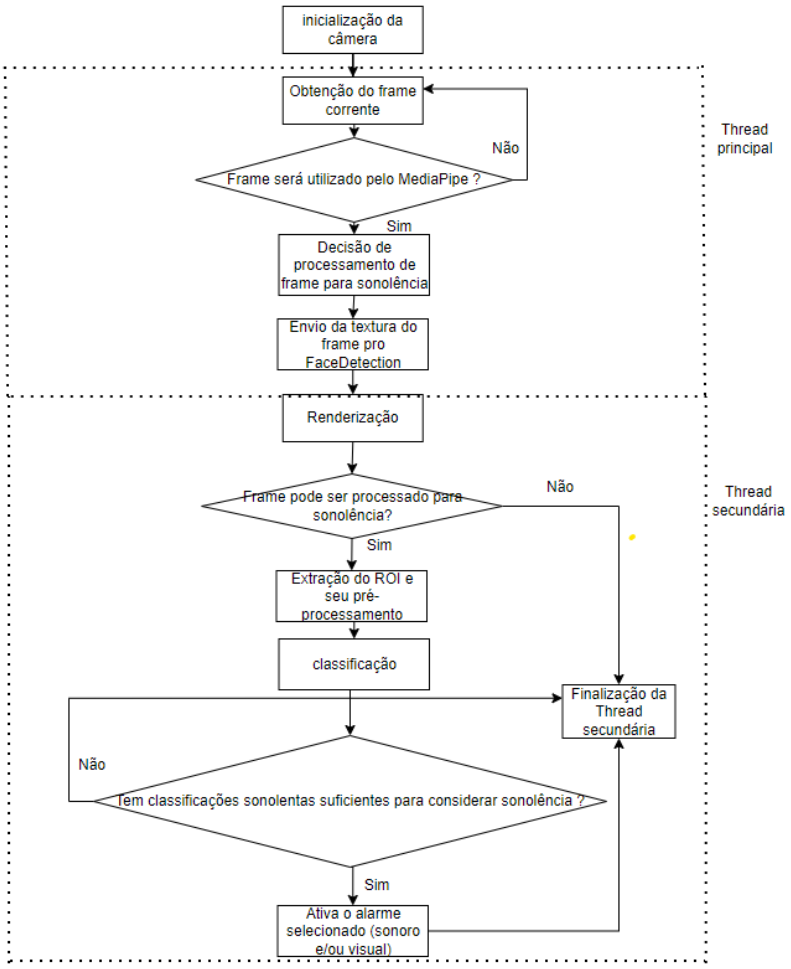


Figura 5.4: Fluxograma da tela de monitoramento

Fluxo básico

1. A aplicação:
    a) Inicializará a câmera e a renderização dos frames
    b) Inicializará o Face Detection
    c) Inicializará o classificador de sonolência
    d) Em caso de sonolência:
        i. Abortará o monitoramento
        ii. Ativará o alarme selecionado
        iii. Após a finalização do alarme, retomará o monitoramento

Fluxos alternativos

1. O motorista poderá voltar para a tela de configuração

a) A aplicação redirecionará a interface para a configuração

2. O motorista poderá voltar para a tela de seleção de alarme

a) A aplicação manterá a salvo as informações de configuração

b) A aplicação redirecionará a interface para a seleção do tipo de alarme

5.4 Pré-processamento da captura

O pré-processamento do frame decorrente da captura da câmera inicia-se através da utilização do modelo do MediaPipe para detecção da face. Após a sua detecção, as coordenadas x_{min}, y_{min} de referência são fornecidas pelo MediaPipe, bem como a altura e largura da bounding box, conforme ilustrado na figura 5.5.

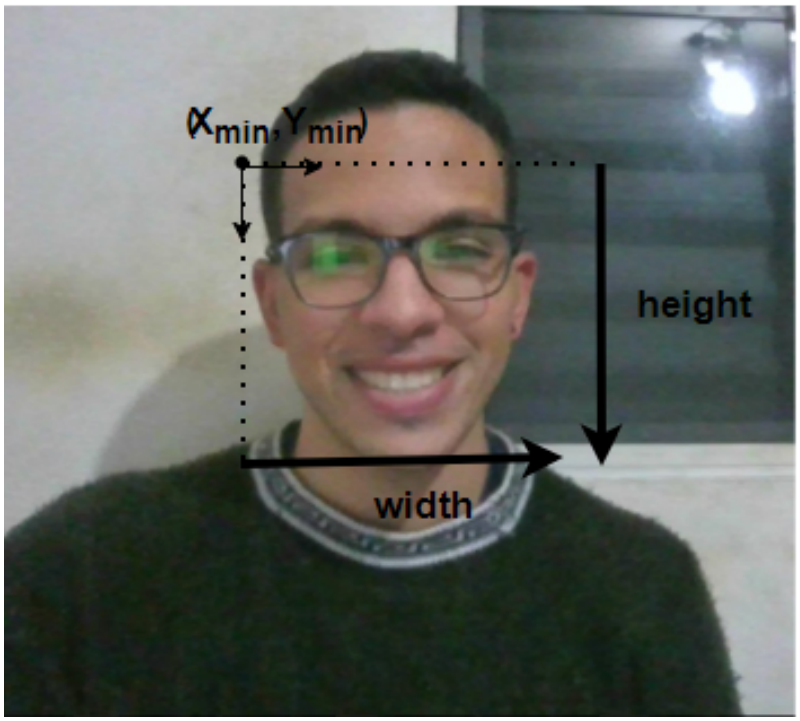


Figura 5.5: Bounding box da face (fonte: Autor)

De posse do recorte, é feita sua reflexão em relação ao eixo y, uma vez que o framework retorna a imagem espelhada. A seguir, o recorte é transformado em escala de cinza, através da extração dos canais R, G e B da imagem e com a relação

$$Grayscale = 0.2989R + 0.5870G + 0.1140B$$

Por fim, a imagem é escalada para o tamanho (160, 160) e é realizada a adição de cada pixel em um *byteBuffer* *FlatBuffers* (2022), que é inserido em um *tensorBuffer* *Ten-*

sorBuffer (2022) de dimensão (1, 160, 160, 3). O processo descrito encontra-se no site oficial do*tensorflow* *Process input and output data with the TensorFlow Lite Support Library* (2022).

De posse do objeto resultante, a *thread* de classificação pode ser inicializada para a detecção.

Figura 5.6: ROI pré-processado (fonte: Autor)

Algorithm 5 Algoritmo de monitoramento

Get the maximum time for pre-processing + classification and maximum FPS for mediapipe fps which were collected in the config screen
Get the number X of frames from which multiples will be used to process the frame
▷ ratio between classification time and mediapipe fps
Get the number N of consecutive frames that will be used to account for drowsiness
▷ ratio between flashing duration (400ms) and classification time
Initialize the queue K with the size equal to N and filled up with awake values
Initialize the classifier
Initialize the manager object
Start camera and MediaPipe

for `each frame that the mediapipe decided to use from the camera` **do**
    Frame can be processed if $Frame_{number}$ mod X and has no process and classification ocurring and alarm not triggered
    Send texture frame to facedetection
    Render the frame
    if Frame can be processed **then**
        Set the manager to process ocurring
        Get the mediapipe facedetection result
        Extract roi coordinates
        Extract input bitmap from result
        Flip the input bitmap in x direction
        Create a new bitmap from cropped inputbitmap (for ROI)
        Scale ROI to (160, 160)
        Transform the roi into input buffer
        Convert from tensor input with shape (1, 160, 160, 3)
        Use the tensor as input to the model
        Set the classifier in manager as busy
        Classify the frame
        Dequeue K and enqueue the result from classifier into K
        Unset the classifier in manager as busy
        if last N ratings contained in K is drowsy **then**
            Set the manager to alarm triggered
            Trigger the alarm
            Unset the manager to alarm not triggered
        end if
        Unset the process ocurring state in manager
    end if
end for

Capítulo 6

Resultados e Discussão

Os resultados da execução da metodologia proposta nos processos para treinamento do modelo (capítulo 4) e desenvolvimento da aplicação (capítulo 5), bem como testes realizados, tanto de tempos de processamento quanto de funcionamento em relação à aplicabilidade do projeto como dispositivo de segurança em situações de uso dentro do veículo, são apresentados a seguir.

6.1 Treinamento

Na tabela a seguir, encontram-se as quantidades de imagens utilizadas para o treinamento, tal como descrito no capítulo 4.

Tabela 6.1: *Separação em teste, treino e validação*

Dados	Quantidade de imagens
Treinamento	68.128
Teste	3.744
Validação	3.808
Total	75.680

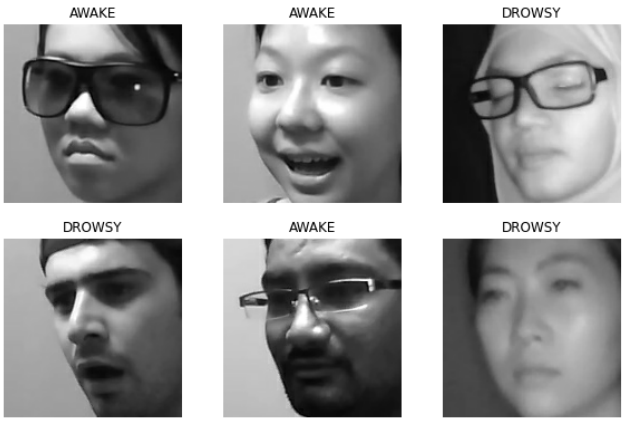


Figura 6.1: Amostras do dataset (fonte: Ching-Hua Weng (2016))

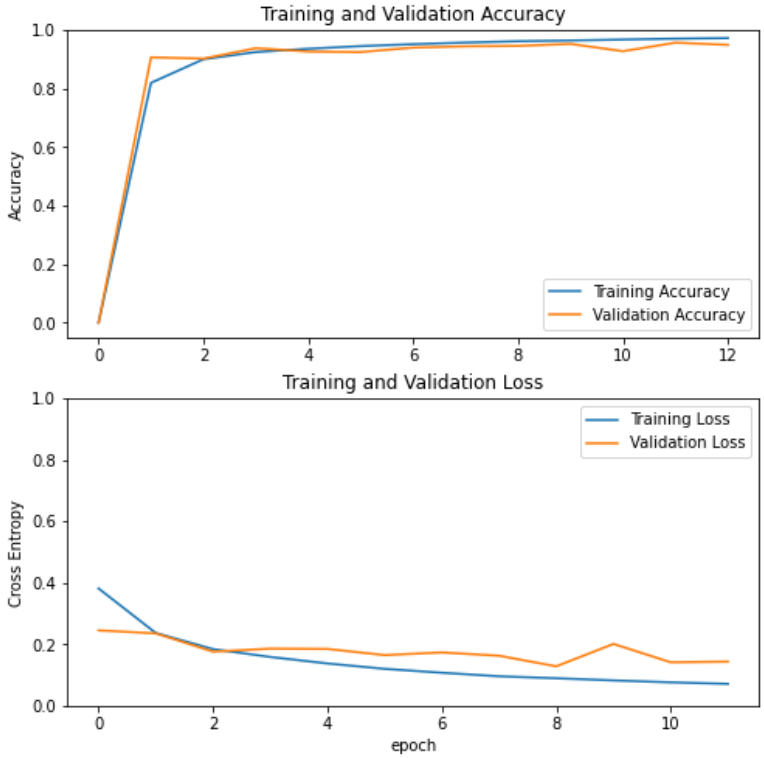


Figura 6.2: Resultado do treinamento do modelo

Os pesos que foram salvos para utilizar no modelo foram os que resultaram em uma menor função de custo nos dados de validação, de forma a prevenir *overfitting*. Os resultados são apresentados a seguir:

Tabela 6.2: *Matriz de confusão*

	Detecção em sonolência	Não detecção
Estava sonolento	1834	70
Não estava sonolento	128	1712

Tabela 6.3: *Resultados do treinamento do modelo*

Dados	Quantidade de batches (de tamanho 32)	Acurácia	Função de custo
Treinamento	2129	96.36%	0.0891
Teste	117	94.71%	0.1327
Validação	119	95.19 %	0.1280

6.2 Aplicação para smartphone

Os resultados da implementação, de acordo com os procedimentos realizados (capitulo 5), tais como interface, tempos de processamento e de testes no veículo, são apresentados a seguir.

6.2.1 Interface

A interface de comunicação do usuário consiste em um conjunto de telas que visam informá-lo e orientá-lo acerca do procedimento que será realizado. Nesse sentido, conforme descritas no capítulo 5, as telas de:

- Configuração: Apresentadas nas figuras 6.4 e 6.5 . Visam orientar o usuário sobre o posicionamento da câmera, bem como sobre evitar obstruções entre a câmera e a face, e quanto ao processo que será realizado;
- Seleção de tipo de alarme: Apresentadas nas figuras 6.6, 6.7 e 6.8. Têm como finalidade instruir o usuário sobre qual tipo de alarme deseja que seja disparado em caso de sonolência
- Monitoramento: Apresentadas nas figuras 6.8, 6.10 e 6.9. Retorna para o usuário a textura que está sendo renderizada, bem como apresentação de um botão a partir do qual é aberto o menu, no qual é possível refazer o processo de configuração ou selecionar outro tipo de alarme.

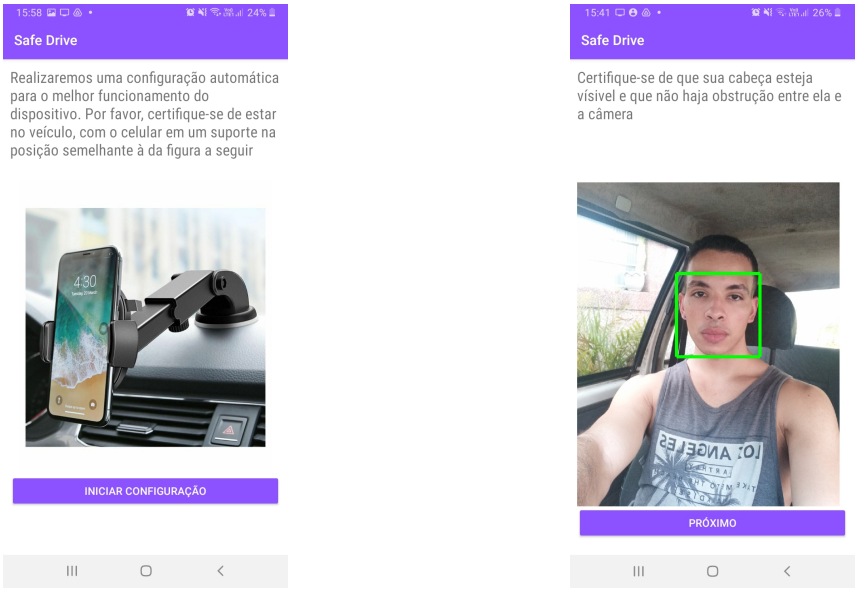


Figura 6.4: Telas de configuração - Inicial e de estimação do FPS

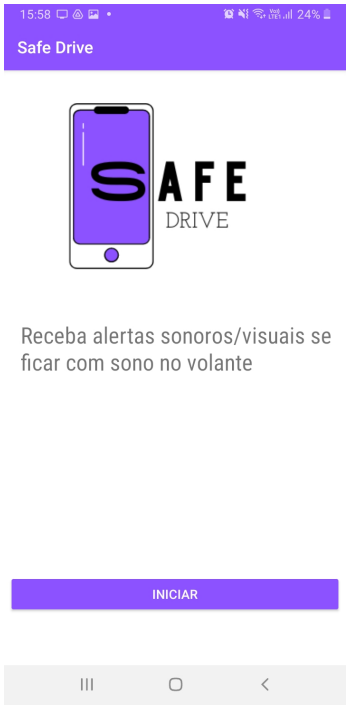


Figura 6.3: Tela de apresentação

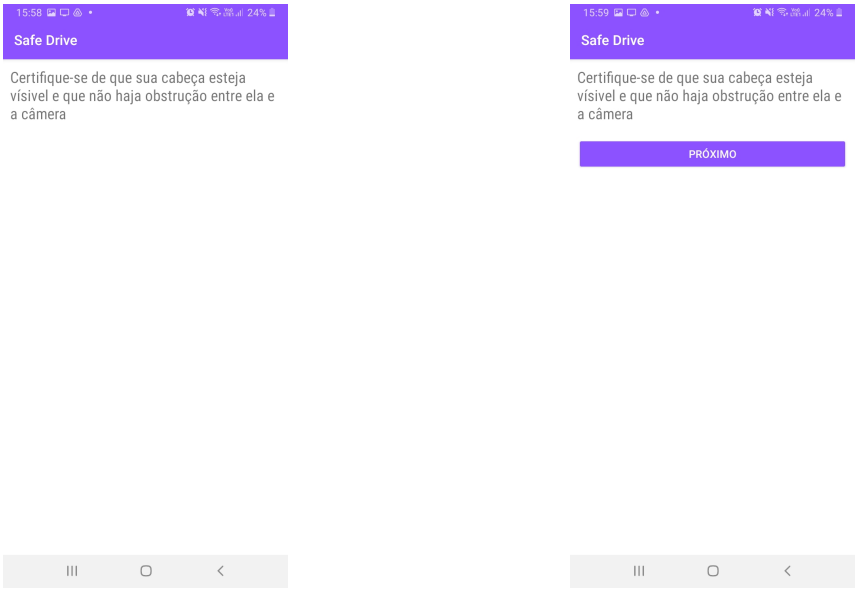


Figura 6.5: Telas de configuração - Estimação do tempo de classificação e pré-processamento

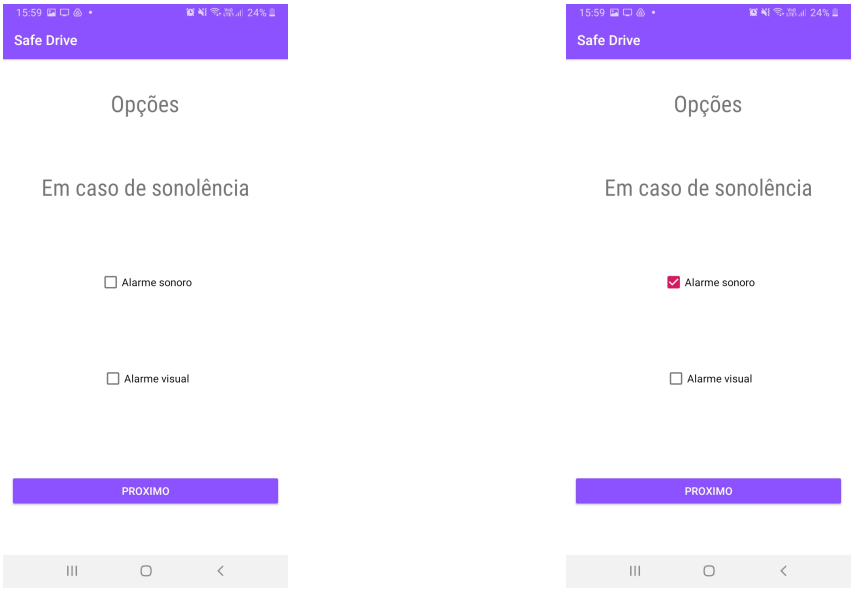


Figura 6.6: Tela de seleção de alarme - Inicial e seleção de sonoro

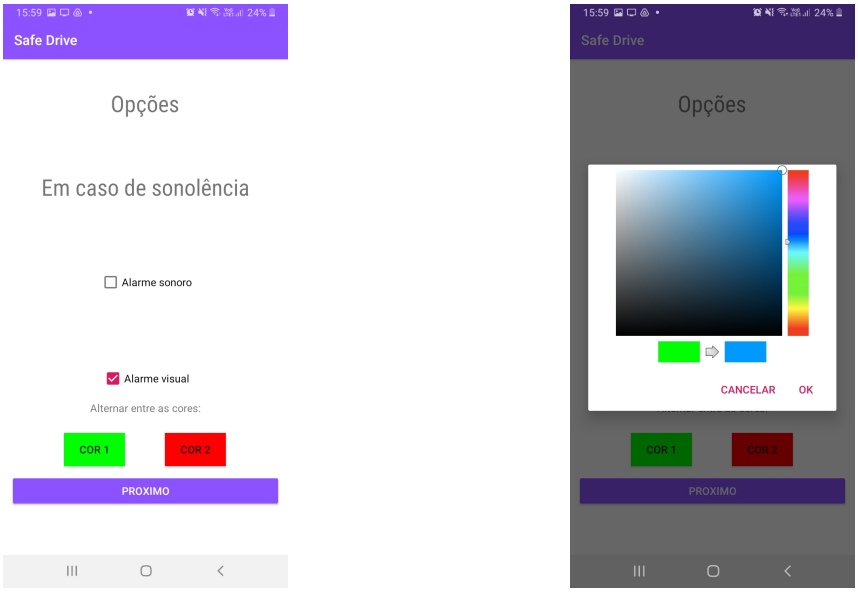


Figura 6.7: Telas de seleção de alarme - Visual, seleção das cores

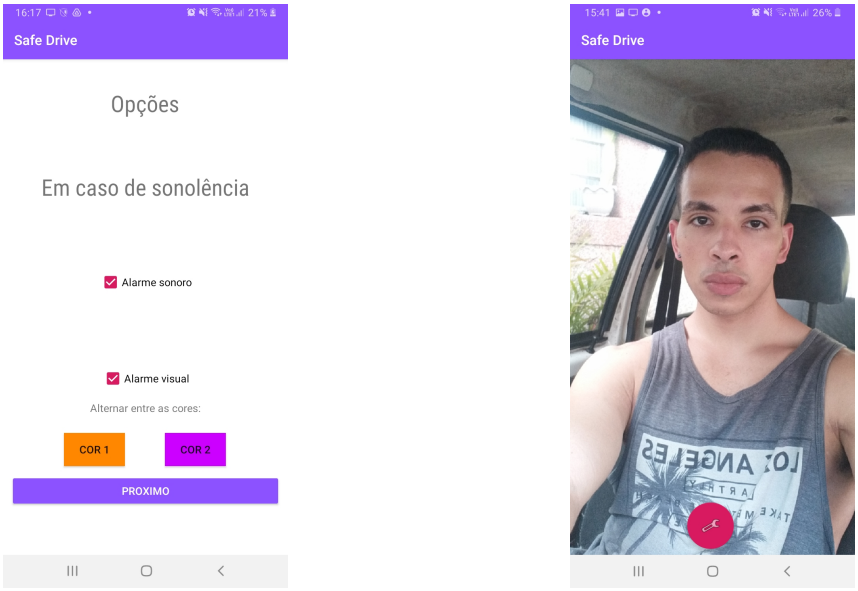
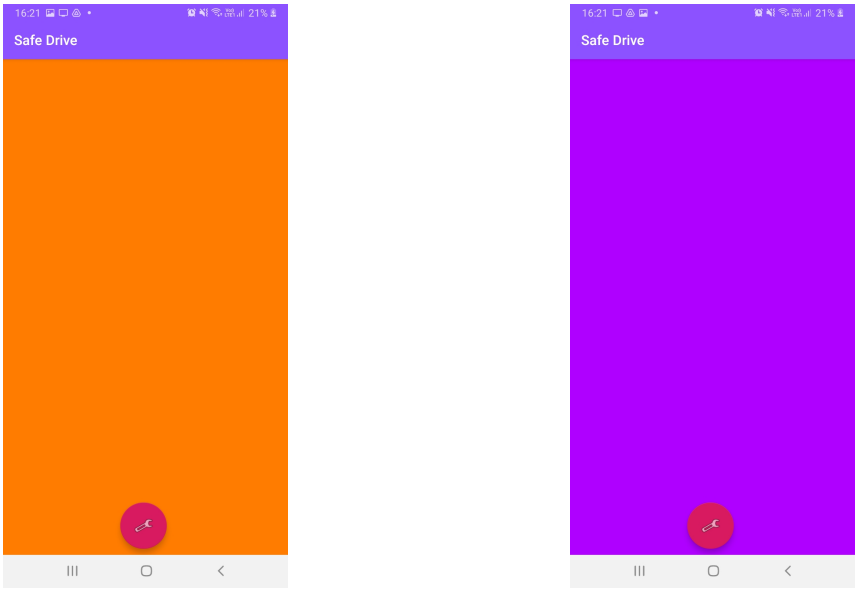


Figura 6.8: Telas de seleção de alarme (Sonoro e visual) e Monitoramento



(a) Interface 1 (b) Interface 2

Figura 6.9: Tela de monitoramento - Alternância das interfaces no alarme visual

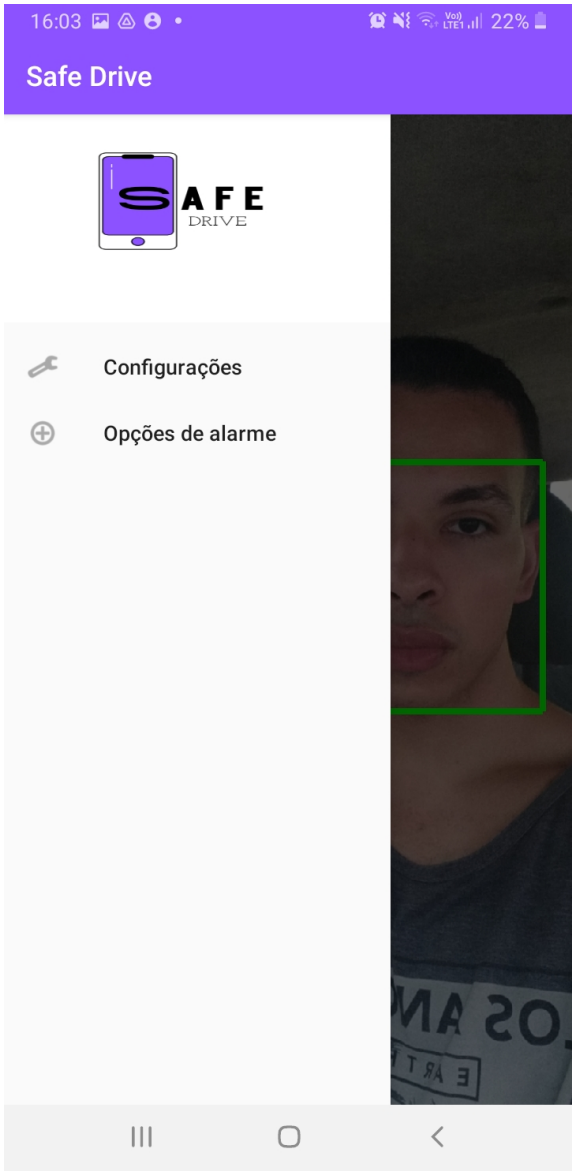


Figura 6.10: Tela de monitoramento - Menu

6.2.2 Tempo de processamento

A duração dos processos ilustrados em 5.3 são de grande importância para a medição do tempo de resposta da aplicação. Dessa forma, foram medidos os tempos que cada processo demora para ser realizado no Galaxy j8, que são apresentados na tabela 6.4. O disparo do alarme, de acordo com a metodologia proposta, de forma a reforçar a redução de disparo por falso positivo durante, principalmente, a piscagem dos olhos, é realizada a cada $Math.ceil(\frac{400ms}{167ms}) = 3$ quadros consecutivos de sonolência, o que representa que, caso o motorista expresse sonolência por mais de, aproximadamente, 501 *ms*, o alarme será disparado. As estimativas são feitas supondo o pior caso, em que o tempo de pré-processamento e classificação é máximo, de 167 *ms*, e o FPS do MediaPipe é máximo, de 28.57 quadros por segundo.

Tabela 6.4: *Tempo de processamento*

Processo	Tempo mínimo	Tempo médio	Tempo máximo	Desvio padrão	Amostras
MediaPipe	28.57 quadros/segundo	13.35 quadros/segundo	10.42 quadros/segundo	2.56	100
Classificação com GPU	38.0 *ms*	41.35 *ms*	55.0 *ms*	3.70	100
Classificação sem GPU	61.0*ms*	78.16 *ms*	171.0*ms*	19.72	100
Pré-processamento e classificação(GPU)	102.0 *ms*	125.89 *ms*	167.0 *ms*	15.32	100

6.2.3 Testes

Foram realizadas diversos testes (*Testes de sonolência* (2022)) com o aplicativo, que podem ser separados em duas categorias: Com o ambiente em luminosidade aproximadamente constante e com variações de luminosidade, ambas em situações de direção. Os resultados são apresentados a seguir. Observa-se que o uso do termo ”adormecimentos”engloba situações de prolongamento do fechamento dos olhos com ou sem inclinação da cabeça.

Tabela 6.5: *Testes em direção simulada com luminosidade aproximadamente constante*

Video	Duração(s)	Cenário	Falsos alarmes	Detecções/Situações sonolentas
1	57	Direção simulada, com óculos, olhando retrovisores (externo e interno), ambiente externo, e com bocejos e adormecimentos	2	2/2
2	67	Direção simulada, sem óculos, olhando retrovisores (externo e interno), com bocejos e adormecimentos	4	4/4
3	94	Direção simulada, sem óculos, olhando retrovisores (externo e interno), ambiente externo, simulando conversas, com bocejos e adormecimentos	0	5/6
4	49	Direção simulada, com óculos,olhando retrovisores (externo e interno), ambiente externo, simulando conversas, com bocejos e adormecimentos	1	3/4
5	75	Direção simulada, com óculos, olhando retrovisores (externo e interno), ambiente externo, simulando conversas, com bocejos	0	3/4
6	95	Direção simulada, olhando retrovisores (externo e interno), ambiente externo, simulando conversas, com bocejos	0	2/2
7	19	Direção simulada, olhando retrovisores (externo e interno), sem sinal de sonolência	0	0/0
8	19	Direção simulada, olhando retrovisores (externo e interno), com adormecimento	0	0/1

Tabela 6.6: *Testes em direção simulada com variações de luminosidade*

Video	Duração(s)	Cenário	Falsos alarmes	Detecções/Situações sonolentas
9	90	Direção simulada, com óculos, olhando retrovisores (externo e interno), com adormecimento e bocejo	1	9/10
10	45	Direção simulada, sem óculos, olhando retrovisores (externo e interno), com adormecimento e bocejo	2	4/4
11	75	Direção simulada, olhando retrovisores (externo e interno), com adormecimento ,bocejo e variação da frequência de piscagem	1	9/9
12	47	Direção simulada, sem óculos, olhando retrovisores (externo e interno), com adormecimento ,bocejo e variação de piscagem	0	6/6

6.3 Discussão

Tendo-se em vista que a sonolência representa grande parte dos acidentes de trânsito e que a solução atual para esse tipo de problema engloba uma minoria da população capaz de adquirir carros que possuem de fábrica sistemas de assistência ao motorista e/ou produtos com elevados custos que podem ser implementados no veículo, neste trabalho foi desenvolvida uma aplicação funcional, em um dispositivo celular, que realiza a detecção de sono emitindo alarmes para o motorista.

Dentro dos requisitos listados, destaca-se que a aplicação abrange uma elevada diversidade étnica e independe de gênero, sendo um ponto fundamental para tornar-se um produto útil para a população. Neste sentido, a convergência do modelo foi dificultada por conta das diferentes pessoas e culturas, uma vez que elementos estéticos, tais como túnicas, não podem influenciar na decisão do classificador, bem como variações de fisionomias têm que ser interpretadas para que sinais de sonolência, como bocejos e fechamentos dos olhos, sejam universalizados e não enviesados para uma determinada etnia e possam ser detectados independente do uso de óculos de grau. Como resultado, foram necessárias diversas amostras para o treinamento de modelo, para que não ocor-

resse sobre-ajuste nos dados, o que tornaria a aplicação enviesada. Embora tenha sido utilizada transferência de aprendizado da *ImageNet*, a partir da qual foram aproveitados os pesos das camadas inicias que reconhecem padrões mais básicos, grande parte do trabalho envolvido na aplicação pode ser atribuído ao tempo gasto durante o treinamento modelo, haja vista que foram necessárias 75.680 amostras para a convergência, além do *tuning* da CNN.

Outro requisito fundamental para a aplicação trata-se do tempo decorrente entre a demonstração de sono pelo motorista e a detecção pela aplicação ser inferior à 1 segundo, metade do tempo de falta de atenção necessário para que o risco de acidentes aumente de forma significativa (Sundfør, Sagberg e Høye (2019)). Nesse sentido, constatou-se que a aplicação da *MobileNet-V2*, embora modificada de acordo com 3.6, que foi a estrutura proposta pelos autores Phan et al. (2021), apresenta uma resposta de *41.35ms*, que somado com o pré-processamento resulta em uma duração média de 125.89ms, o que é relativamente rápida frente aos requisitos exigidos. No entanto, como o número de classificações, dada a velocidade da rede e que a aplicação funcionará continuamente, cresce significativamente ao longo do tempo, tornou necessária a adoção de classificações sucessivas para que seja soado o alarme, haja vista que expressões sonolentas podem ser obtidas de frames individuais, como os em que o motorista esteja piscando os olhos. Soma-se a isso a necessidade de sincronismo da aplicação, uma vez que a classificação anterior não pode estar muito defasada do quadro atual do motorista, o que resultou nas medidas citadas de descarte de frames e controle de uso do classificador. Dessa maneira, embora o grupo tinha como pressuposto que a principal dificuldade de utilizar a aplicação embarcada estaria na classificação do frame, o que de fato tornou-se mais desafiador foi a sincronização do tempo corrente do motorista com a classificação que está em andamento e a eliminação de gargalos de memória.

Outro ponto levantado como requisito pelo grupo foi a robustez da aplicação. Nesse sentido, optou-se pelo uso do MediaPipe para o reconhecimento da face, uma vez que, conforme apresentado, é capaz de realizar a detecção mesmo em ambientes de baixa luminosidade. A base de dados utilizada, Ching-Hua Weng (2016), contém filmagens de situações que incluem o cenário noturno. Nesse sentido, o único obstáculo para a detecção durante a noite encontra-se no dispositivo em que a aplicação estará rodando, que precisa, obrigatoriamente, possuir uma câmera capaz de detectar com resolução suficiente a face do motorista em ambientes sem iluminação, tais como câmeras infravermelhas presentes em alguns modelos de celular. Assim, não foram realizados testes quanto ao ambiente com baixa iluminação, dado que o grupo não possui um celular com tais requisitos.

Embora a aplicação, por conta de limitação do grupo, tenha sido realizada em um ambiente Android, as ferramentas utilizadas são possíveis de serem transferíveis para iOS, dado que os principais frameworks utilizados (MediaPipe e Tensorflow lite) são suportados em ambos ambientes. Neste sentido, a limitação de acessibilidade encontra-se nos requisitos dessas próprias bibliotecas, o que, no caso do Android, por exemplo, necessita de um smartphone com, no mínimo, API 30, o que corresponde a, aproximadamente, 40.5% dos dispositivos Android, de acordo com o Android Studio. Além disso, a aplicação foi desenvolvida de forma que utilizasse dados de processamento específicos do aparelho em que será utilizada, evitando o uso de constantes empíricas especificas para um determinado modelo, processo que se inicia na tela de configuração. Fora isso, ressalta-se que a GPU não é necessária para o funcionamento do aplicativo, mas a sua ausência implica em uma detecção mais lenta. Dessa maneira, considera-se que a aplicação é compatível com smarthphones.

Ressalta-se, também, as limitações do aplicativo. Nesse sentido, fora a incapacidade de atuar em ambientes com pouca iluminação caso o celular não possua câmera infravermelho, através das simulações realizadas, apresentadas nas tabelas 6.5 e 6.6, pode-se concluir que a maior probabilidade de falso alarme ocorre em situações em que o motorista toma algumas ações não previstas na base de dados utilizada, tais como olhar para trás, colocar a mão na face, olhar para baixo, em especial para o chão da pista à sua esquerda, já que a cabeça fica inclinada para baixo, que é um sinal de sonolência, o que se agrava quando o condutor utiliza óculos, posição em que os olhos são obstruídos pelo óculos, o que impossibilita a câmera de detectá-los. Ressalta-se, também, que caso o celular não esteja fixo na posição correta, existe uma grande possibilidade de falso alarme, uma vez que os dados utilizados para treinar o modelo são em ambientes de direção em que a posição da câmera frente ao motorista é pré-determinada, assim, caso o motorista utilize a aplicação com uma posição relativa da câmera não adequada, a situação pode ser interpretada de maneira errônea pelo modelo, já que o mesmo leva em consideração a posição da cabeça para que inclinações sejam consideradas na asserção de sonolência. Além disso, ressalta-se que cenários em que o motorista utiliza óculos de sol e máscara não foram considerados, entrando, também, como limitações do projeto. Vale retomar, conforme apontado nos requisitos de projeto, que trepidações, vibrações e balanços do veículo não foram considerados no contexto deste projeto.

Capítulo 7

Conclusão

Com finalidades de detecção de sono em tempo real em um smartphone, neste trabalho foi realizado o treinamento de uma rede neural MobileNetV2 modificada para extração de recursos indicadores de sonolência da face do motorista. A rede tem por objetivo realizar uma classificação binária, nos estados ”Sonolento”e ”Acordado”, e apresenta acurácia de 94.71%. A detecção do rosto do condutor no frame da câmera para posterior envio para a classificação é realizada pelo framework do Google *MediaPipe*, que viabilizou a implementação do projeto com baixo tempo de resposta. O pré-processamento e classificação de cada frame, no dispositivo utilizado pelo autor (Galaxy J8), é de, no pior caso, 167 *ms*, e a duração do estado de sonolência necessário para o disparo do alarme é de, aproximadamente, 501 *ms*. Conforme discutido na seção 6.3, a aplicação possui limitações que abrem espaço para melhorias. Nesse sentido, propõe-se, para trabalhos futuros, a inclusão de mais situações/posturas que o motorista possa vir a expressar durante a condução (como colocar a mão na face, olhar para trás, utilização de máscaras/óculos de sol,etc.), bem como abranger maiores possibilidades de posicionamento da câmera; implementação do pré-processamento com GPU; Adaptações para inclusão de casos de vibração, trepidação ou balanço do automóvel e estudar a implementação da aplicação em background, de forma que outros aplicativos possam utilizá-lo em plano de fundo durante seu uso.

Referências

Bazarevsky, Valentin et al. (2019). *BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs*. DOI: 10.48550/ARXIV.1907.05047. URL: https://arxiv.org/abs/1907.05047.

Ching-Hua Weng Ying-Hsiu Lai, Shang-Hong Lai (2016). *Driver Drowsiness Detection via a Hierarchical Temporal Deep Belief Network, In Asian Conference on Computer Vision Workshop on Driver Drowsiness Detection from Video, Taipei, Taiwan*. URL: http://cv.cs.nthu.edu.tw/php/callforpaper/datasets/DDD/ (acesso em 05/06/2022).

Chirra, Venkata Rami Reddy, Srinivasulu Reddy Uyyala e Venkata Krishna Kishore Kolli (2019). "Deep CNN: A machine learning approach for driver drowsiness detection based on eye state". Em: *Revue d'Intelligence Artificielle* 33 (6). v
. ISSN: 19585748. DOI: 10.18280/ria.330609.

Dasgupta, Anirban, Daleef Rahman e Aurobinda Routray (nov. de 2019). "A Smartphone-Based Drowsiness Detection and Warning System for Automotive Drivers". Em: *IEEE Transactions on Intelligent Transportation Systems* 20 (11), pp. 4045–4054. ISSN: 15580016. DOI: 10.1109/TITS.2018.2879609.

Dipu, Md Tanvir Ahammed et al. (2021). "Real-time Driver Drowsiness Detection using Deep Learning". Em: *International Journal of Advanced Computer Science and Applications* 12 (7). ISSN: 21565570. DOI: 10.14569/IJACSA.2021.0120794.

Electrical Engineering, Department of e Liège Computer Science of the University of Liège (ULg) (2016). *ULg Multimodality Drowsiness Database*. URL: http://www.drozy.ulg.ac.be/ (acesso em 25/05/2022).

FlatBuffers (2022). URL: https://google.github.io/flatbuffers/ (acesso em 12/07/2022).

FlatBuffers (2022). URL: https://docs.oracle.com/javase/7/docs/api/java/nio/ByteBuffer.html (acesso em 12/08/2022).

Ghoddoosian, Reza, Marnim Galib e Vassilis Athitsos (2019). "A Realistic Dataset and Baseline Temporal Model for Early Drowsiness Detection". Em: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0.

Google (2019). *MediaPipe BlazeFace*. URL: https://google.github.io/mediapipe/solutions/face_detection (acesso em 20/05/2022).

Hashemi, Maryam, Alireza Mirrashid e Aliasghar Beheshti Shirazi (2020). "Deep learning based Driver Distraction and Drowsiness Detection". Em: *arXiv* (September). v
. ISSN: 2661-8907. URL: https://doi.org/10.1007/s42979-020-00306-9.

ImageNet (2022). URL: https://www.image-net.org/index.php (acesso em 05/05/2022).

Indrabayu, Siti Khumaerah Mufti e IntanSariAreni (2019). "Car Driver Drowsiness Recognition Android-Based System". Em: DOI: 10.1088/1757-899X/619/1/012021.

Jabbar, Rateb et al. (fev. de 2020). "Driver Drowsiness Detection Model Using Convolutional Neural Networks Techniques for Android Application". Em: Institute of Electrical e Electronics Engineers Inc., pp. 237–242. ISBN: 9781728148212. DOI: 10.1109/ICIoT48696.2020.9089484.

Phan, Anh Cang et al. (set. de 2021). "An efficient approach for detecting driver drowsiness based on deep learning". Em: *Applied Sciences (Switzerland)* 11 (18). ISSN: 20763417. DOI: 10.3390/app11188441.

Process input and output data with the TensorFlow Lite Support Library (2022). URL: https://www.tensorflow.org/lite/inference_with_metadata/lite_support (acesso em 12/08/2022).

Rajkumarsingh, B e D Totah (2021). "Drowsiness Detection using Android Application and Mobile Vision Face API". Em: *Journal of the South African Institution of Mechanical Engineering* 37, pp. 26–34. DOI: 10.17159/2309-8988/2021/v37a4. URL: http://dx.doi.org/10.17159/2309-8988/2021/v37a4http://www.saimeche.org.za.

Sandler, Mark et al. (2018). "MobileNetV2: Inverted Residuals and Linear Bottlenecks". Em: DOI: 10.48550/ARXIV.1801.04381. URL: https://arxiv.org/abs/1801.04381.

Sinha, Avigyan, R. P. Aneesh e Sarada K. Gopal (mar. de 2021). "Drowsiness Detection System Using Deep Learning". Em: Institute of Electrical e Electronics Engineers Inc. ISBN: 9781665441261. DOI: 10.1109/ICBSII51839.2021.9445132.

Stancin, Igor, Mario Cifrek e Alan Jovic (2021). "A review of eeg signal features and their application in driver drowsiness detection systems". Em: *Sensors* 21 (11). ISSN: 14248220. DOI: 10.3390/s21113786.

Sundfør, Hanne Beate, Fridulv Sagberg e Alena Høye (abr. de 2019). "Inattention and distraction in fatal road crashes – Results from in-depth crash investigations in Norway".

Em: *Accident Analysis Prevention* 125, pp. 152–157. ISSN: 0001-4575. DOI: 10.1016/J.AAP.2019.02.004.

Suresh, Yeresime et al. (2021). "Driver Drowsiness Detection using Deep Learning". Em: Institute of Electrical e Electronics Engineers Inc., pp. 1526–1531. ISBN: 9781665433686. DOI: 10.1109/ICOSEC51865.2021.9591957.

TensorBuffer (2022). URL: https://www.tensorflow.org/lite/api_docs/java/org/tensorflow/lite/support/tensorbuffer/TensorBuffer (acesso em 12/08/2022).

Testes de sonolência (2022). URL: https://drive.google.com/drive/folders/1KnCVJEbxVwPOzNid9NrZ6piG5K4RE5vN?usp=sharing (acesso em 12/11/2022).

The Android software stack (2022). URL: https://developer.android.com/guide/platform (acesso em 12/07/2022).

Zhu, Miankuan et al. (2021). "Vehicle driver drowsiness detection method using wearable EEG based on convolution neural network". Em: *Neural Computing and Applications* 33 (20). ISSN: 14333058. DOI: 10.1007/s00521-021-06038-y.