Uso de Inteligência Artificial Para Identificar Motoristas Com Sinais De Sono A Fim De Evitar Futuros Acidentes

Richard Rangel

ICT

Unifesp

Sao Jose dos Campos, Brasil

richard.junior@unifesp.br

***Resumo*—Esse artigo irá tratar do uso da inteligência artificial para identificar motoristas que apresentam sinais de sono para que assim possam ser evitados futuros acidentes, nele vai ser falado sobre todo o processo de criação desse algoritmo que irá contar com o Deep Learning a resolução desse problema.**

Palavras-chave—inteligência artificial, motoristas, sono, acidentes, Deep Learning

I. Introdução

No Brasil, cerca de 42% dos acidentes nas rodovias federais estão relacionados a sonolência, sendo a terceira maior causa de acidentes de trânsito no país, segundo levantamento realizado em 2019 pela Associação Brasileira de Medicina de Tráfego (ABRAMET) [1]. Além disso, em estudo divulgado pela National Sleep Foudantion's em 2023 mostra que o impacto de dirigir com sono é semelhante ao impacto de dirigir bêbado [2]. Sendo assim, é perceptível o impacto negativo de motoristas com sono no mundo inteiro e como uma possível solução para esse problema que enfrentamos que veio a ideia desse artigo, que é a criação de uma inteligência artificial que identifica quando o motorista está com sono e já ciente dessa situação o sistema do carro poderia tomar certas providencias para evitar que ocorra um acidente, como por exemplo aumentar o volume do som do carro ou emitir um sinal sonoro de alerta para tentar acordar o motorista em questão, colaborando para que um possível futuro acidente não ocorra.

II. Conceitos Fundamentais

A. Linguagem de programação usada

Para a construção desse algoritmo será utilizado a linguagem de programação python, que é uma das linguagens mais utilizadas na área de inteligência artificial devido a facilidade do uso e capacidade de procedimentos complexos, fora suas inúmeras bibliotecas que colaboram para o melhor desenvolvimento do algoritmo.

B. Deep Learning

Além disso, dentro da inteligência artificial usaremos o Deep Learning, que segundo definição encontrada no site da IBM, é um subconjunto de aprendizado de máquina, que é essencialmente uma rede neural com três ou mais camadas. Essas redes neurais tentam simular o comportamento do cérebro humano, embora longe de corresponder a sua capacidade, permitindo que ele “aprenda” com grandes quantidades de dados [3]. Seguindo essa linha que trabalharemos com o algoritmo para resolver o problema em questão.

C. CNN

A Convolution Neural Network (CNN), ou em português Redes neurais convolucionistas, é um tipo de algoritmo de deep learning especializado principalmente em tarefas que necessitam do reconhecimento do objeto, incluindo

classificação da imagem, detecção e segmentação. As CNNs se distinguem dos modelos clássicos de machine learning (aprendizado de máquina) por sua habilidade de fazendo de maneira autônoma extrair as características e em uma larga escala, ultrapassando a maneira manual de ajustá-las e consequentemente aumentando a eficiência.

D. Domain Adaptation

O Domain Adaptation(ou Adaptação de Domínio em português) é a tarefa de adaptar modelos entre domínios. Isso é motivado pelo desafio em que os conjuntos de dados de teste e de treinamento pertencem a diferentes distribuições de dados devido a algum fator. O domain adaptation visa construir modelos de aprendizado de máquina que possam ser generalizados para um domínio-alvo, lidando com a discrepância entre as distribuições dos domínios.

III. Trabalhos Relacionados

Um trabalho bem semelhante a esse já foi feito e publicado pelo International Journal of Research Publication and Reviews em 2022, nesse trabalho feito na Índia eles partiram do mesmo ponto em comum, vendo a grande causa de acidentes no trânsito pautada no erro do motorista, como dirigir com sono. Então criaram um sistema de identificação baseado na fadiga da visão usando a técnica de Convolution Neural Network (CNN), que é um algoritmo do Deep Learning [4].

Outro artigo também já havia sido publicado no mesmo jornal um ano antes pelo Bundelkhand Institute of Engineering and Technology (Instituto de Engenharia e Tecnologia Bundelkhand) da Índia, nesse estudo eles usam o python para a detecção de sonolência do motorista, nesse processo envolve a detecção facial, detecção da posição dos olhos e o padrão que o olho pisca. O algoritmo analisa e se os olhos estiverem fechados por alguma razão então o alarme será acionado, alertando o motorista [5].

No Department of Software Engineering (Departamento de engenharia de software) da Daffodil International University de Bangladesh publicaram o artigo Real-time Driver Drowsiness Detection using Deep Learning (Detecção em tempo real de motorista sonolento usando Deep Learning) em que tentam em sua pesquisa propor um procedimento de bom custo-benefício para identificar os motoristas sonolentos, para isso, uma das técnicas que é utilizada é as redes neurais convolucionistas(CNN) para a detecção no sistema e mais algumas outras técnicas [6].

Agora um estudo do lado ocidental, a universidade de Waterloo, no Canadá, criou um sistema de detecção de sonolência que inclui algo interessante que é o PERCLOS SystemsPERCLOS, definido como a medida de porcentagem de tempo que as pupilas dos olhos estão 80% ou mais fechadas em um período específico, a proposta do trabalho deles está em um algoritmo de detecção de olhos além de outras abordagens também [7].

IV. Objetivo

O objetivo nesse trabalho é desenvolver um algoritmo em python usando uma rede neural convolucional (CNN) que tem como objetivo identificar se um motorista apresenta ou não sinais de sonolência baseado na aprendizagem adquirida no treinamento com a base de dados recebida

V. Metodologia Experimental

Nesse trabalho usaremos o python, devido a maior facilidade e suas bibliotecas disponíveis para a execução dessa tarefa que vamos fazer.

Iremos usar duas bases de dados já prontas também, ambas relacionadas ao assunto de identificar se um motorista está ou não com sinais de sono.

Iremos trabalhar com o Google Colab como área de trabalho. Será utilizado algumas bibliotecas para esse algoritmo, como opendatasets, para instalar a base de dados diretamente do kaggle, Tensorflow para a parte do aprendizado de máquina, mais especificamente o Keras. Para o tratamento dos dados utilizaremos várias bibliotecas, como o Numpy, Panda, Splitfolders e train_test_split do SKleran (para separar a base de dados em treino e teste), já para poder trabalhar com os gráficos e maior visibilidade do processo a biblioteca Matplotlib e também a biblioteca Sklearn para avaliar a acurácia do algoritmo.

Nesse trabalho temos alguns passos até chegarmos ao resultado final. Primeiramente treinamos e testamos a primeira base de dados com o primeiro modelo, após isso, treinamos e testamos a segunda base de dados usando outro tipo de modelo, então usamos o treino já feito com a primeira base de dados e testamos com o teste da segunda, após isso fazemos o contrário e testamos a primeira base com o treino já feito com a segunda. Então juntamos as duas bases de dados e treinamos e testamos com um modelo normal e após isso testamos agora com um modelo de domain adaptation. E sempre analisando os resultados que obtivemos.

Para começar, instalamos a base de dados na nossa área de trabalho e começamos a trabalhar com ela, primeiro separando-a em porção de treino (80%) e teste (20%), após isso se inicia o processo de normalização das imagens e depois conferimos se tudo ocorreu bem.

Então iniciamos a modelagem de nosso algoritmo, foi usado o modelo já pré-treinado MobileNetV2, carregando a imagenet, além disso adicionamos mais algumas camadas para completar o modelo.

Assim começamos a treinar com essa base para que o algoritmo possa aprender o máximo possível, testamos ela no modelo e então geramos os gráficos e analisamos os resultados obtidos.

Após isso, fazemos de forma semelhante com a segundo base de dados, primeiro preparamos a base de dados, usamos uma função para processarmos os rostos das imagens e processa-los de maneira mais eficiente, Nesse caso foi necessário fazer data augmentation para aumentar os dados e para termos uma base de dados melhor para trabalharmos, os dados são normalizados e então testamos com outro modelo.

Agora vem a parte de usarmos o modelo já treinado com os dados de uma base para testar na outra base, primeiro usamos a porção de teste da base 2 no modelo já treinado com a base 1 e após isso fazemos o inverso, usando a porção de teste da base 1 no modelo já treinado com a base 2.

Agora nessa parte juntamos as duas bases da dados para formar apenas uma só, contudo há uma grande diferença no número de imagens que cada uma tem, então para tirar tal contraste a base foi tratada, primeiramente fizemos o data augmentation da segunda base que contava com poucas imagens e na primeira base removemos aleatoriamente cerca de 89% das imagens. Dessa forma obtivemos a proporção de uma base para outra baixou bruscamente, de tal forma que não interferisse mais na hora de testar o modelo e assim obtivemos uma base completa.

Então com essa nova base criamos um novo modelo bem similar ao modelo 2 (pois obteve melhor resultado que o 1 em nossos experimentos) mudando apenas o formato da entrada e treinamos e testamos essa nova base.

E na última parte, como estávamos agora com uma base que era composta na realidade de duas bases de dados diferentes, usamos um algoritmo de domain adaptation que é próprio para tal tarefa. Para tal tarefa foi escolhido o algoritmo DANN, modelamos ele totalmente do início e após isso treinamos e testamos com a base completa.

Sempre após cada experimento foram gerados os gráficos de acurácia e perda para que pudéssemos comparar e avaliar os modelos da melhor forma.

VI. Resultados/ Discussões

Na primeira parte do trabalho, trainamos e testamos a primeira base com o modelo q era baseado em grande parte também no modelo já pré-treinado MobileNetV2, foram rodadas somente 5 épocas, tanto pelo tempo para fazer isso tanto pelos resultados, nessas poucas épocas já foi obtido uma acurácia bem próxima de 1(como mostra na figura 1) e a função de perda já estava bem próxima de 0 também (como mostra na figura 2).

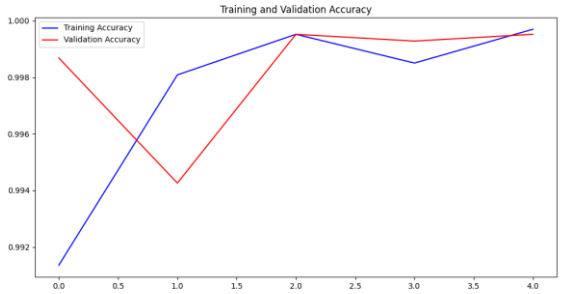

Fig.1. Gráfico de acurácia do modelo 1 testado na base de dados 1

Fig.2. Gráfico de perda do modelo 1 testado na base de dados 1

Assim teria duas hipóteses para esses resultados tão extremos, ou seria um modelo ótimo, ou uma base de dados não muito boa, que é a resposta certa.

Na segunda parte, por ser uma base de dados menor fizemos o data augmentation, usamos ele devido ao baixo número de imagens e pudemos rodar mais épocas, foram rodadas 30 épocas e a acurácia foi evoluindo bem e chegou na casa dos 93% (como mostra a figura 3) com a função de perda abaixo de 0.3 (como mostra a figura 4).

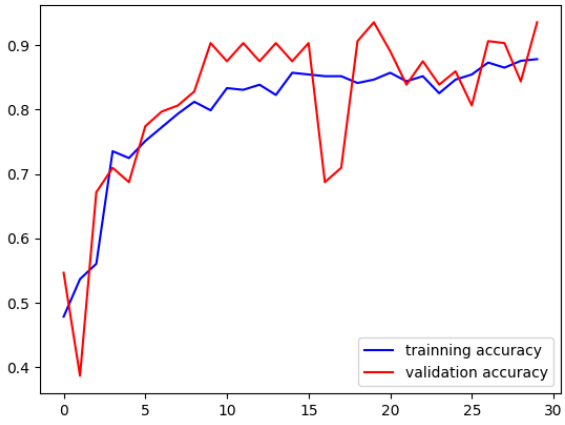

Fig.3. Gráfico de acurácia do modelo 2 testado na base de dados 2

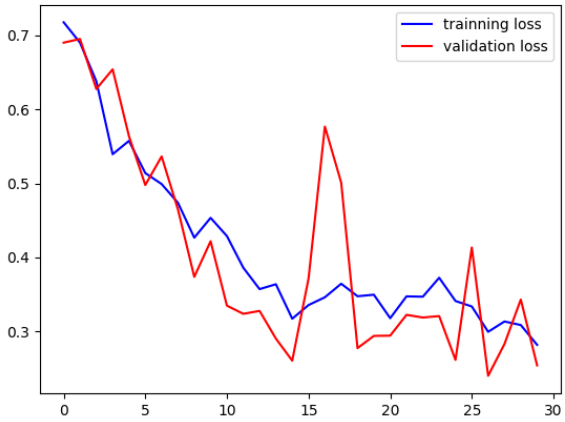

Fig.4. Gráfico de perda do modelo 2 testado na base de dados 2

Esses resultados apesar de serem menores e olhando somente os números piores que do modelo anterior foram mais

satisfatórios pois nesse caso não se trata de uma base ruim então esse resultado é mais expressivo.

Agora chegando na parte que testamos os modelos com a outra base, na terceira parte, selecionamos a segunda base e tratamos da mesma forma que foi tratada a primeira base e testamos usando o modelo já treinado com a primeira base. O resultado foi catastrófico, mas já era algo esperado também. Rodando o modelo novamente resultou em uma acurácia de cerca da metade adquirida com a primeira base, sendo a máxima de 50,34% (como mostra a figura 5) e a função de perda com resultado mínimo de 9.49(como mostra a figura 6).

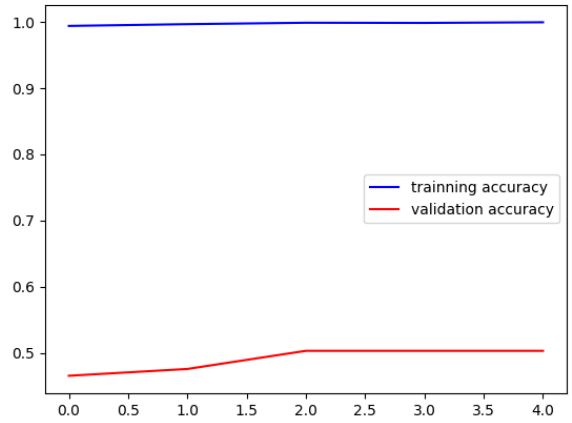

FIG.5. GRÁFICO DE ACURÁCIA DO MODELO 1 TESTADO NA BASE DE DADOS 2

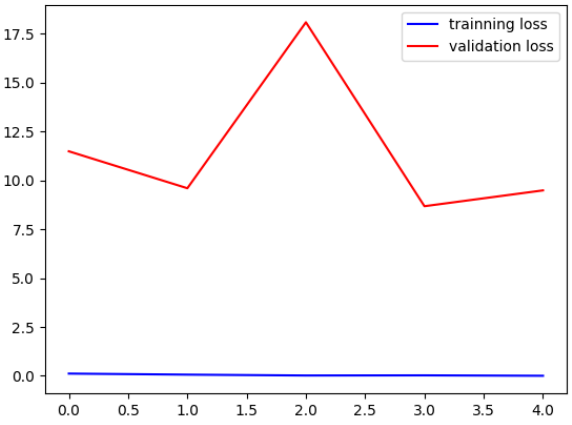

FIG.6. GRÁFICO DE PERDA DO MODELO 1 TESTADO NA BASE DE DADOS 2

Além disso, avaliando o modelo usando a função ‘evaluate’ do Keras foi obtido uma acurácia de 45,65% e uma perda de mais de 8.5(como mostra a figura 7). Mostrando o resultado desse primeiro modelo quando posto em prova.

```
loss, accuracy = model1.evaluate(test_batches2)
print("Loss:", loss)
print("Accuracy:", accuracy)
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 10s 241ms/step - accuracy: 0.4565 - loss: 8.5625
Loss: 8.873912811279297
Accuracy: 0.4383561611175537
```

FIG.7. AVALIANDO O PRIMEIRO MODELO COM A SEGUNDA BASE

Agora na quarta parte chegou a vez de colocar o segundo modelo a prova, então a primeira base foi tratada de maneira completamente similar a primeira base e foi testada no modelo 2 que já havia sido treinado com a segunda base. Foram executadas 20 épocas e como resultado desse experimento tivemos uma acurácia bem inconstante, mas com resultados parecidos com o que já havia sido treinado (como mostra a figura 8), de forma semelhante aconteceu com a função de perda (como mostra a figura 9).

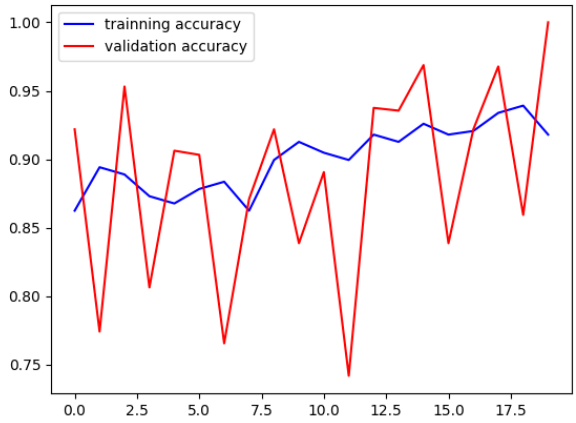

FIG.8. GRÁFICO DE ACURÁCIA DO MODELO 2 TESTADO NA BASE DE DADOS 1

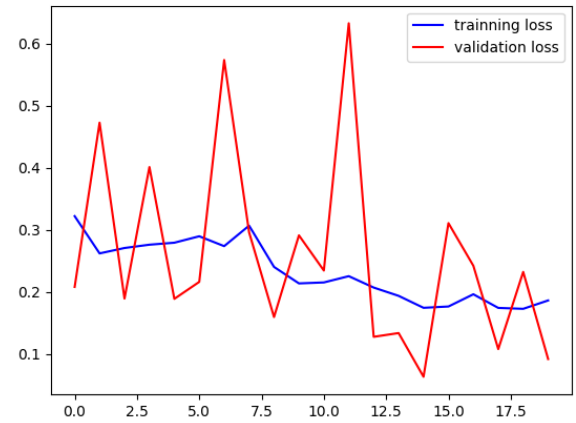

FIG.9. GRÁFICO DE PERDA DO MODELO 2 TESTADO NA BASE DE DADOS 1

Em comparação com o teste feito na terceira parte os resultados foram melhores. E ao usar a função ‘evaluate’ tivemos uma acurácia bem parecida com a já obtida na parte anterior, contudo, a perda foi cerca de 8 vezes menor (como mostra a figura 10), já mostrando então um avanço em relação ao modelo anterior.

```
loss, accuracy = model2.evaluate(test_batches_1)
print("Loss on dataset1:", loss)
print("Accuracy on dataset1:", accuracy)
```

```
523/523 ━━━━━━━━━━━━━━━━━━━━ 34s 61ms/step - accuracy: 0.4295 - loss: 1.2002
Loss on dataset1: 1.1815904378890991
Accuracy on dataset1: 0.43175020813941956
```

FIG.10. AVALIANDO O SEGUNDO MODELO COM A PRIMEIRA BASE

Agora nessa quinta parte houve uma mudança, nessa parte unimos as duas partes, claro que, antes adaptamos cada uma das bases (removendo boa parte da primeira que era bem maior e realizando o data augmentation na segunda que era bem menor) para que a diferença entre elas não influenciasse tanto no resultado, criando assim uma base completa.

Além disso, usamos um modelo quase igual ao modelo 2 (que obteve melhores resultados), mudando apenas o formato da entrada. Neste teste, rodamos 7 épocas e os resultados foram muito bons, a acurácia foi sempre superior aos 90% e ficando acima dos 95% após a segunda época (como mostra a figura 11) e a função de perda se manteve baixa ao decorrer do treinamento, estando abaixo de 0.1 a partir da segunda época (como mostra a figura 12).

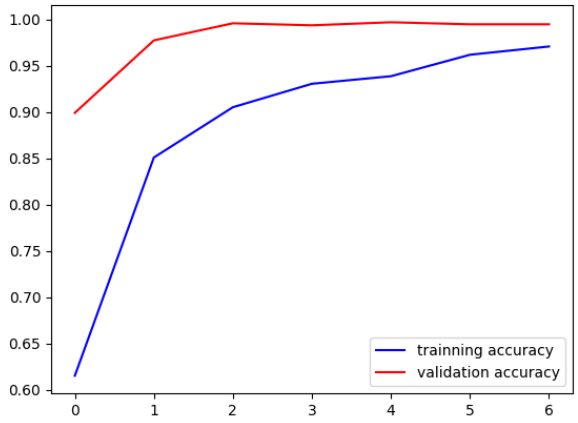

FIG.11. GRÁFICO DE ACURÁCIA DO MODELO 3 TESTADO NA BASE COMPLETA

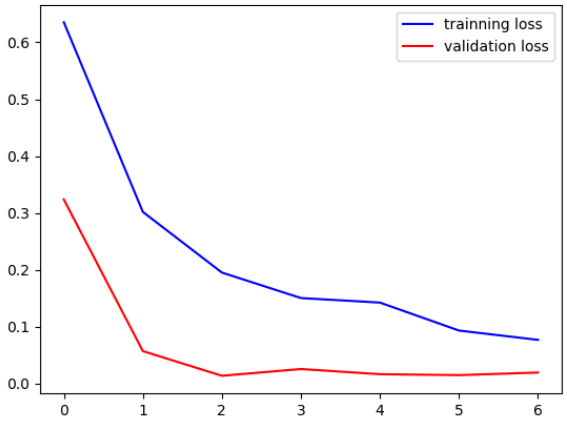

FIG.12. GRÁFICO DE PERDA DO MODELO 3 TESTADO NA BASE COMPLETA

Esses resultados já evidenciam uma melhora diante de tudo que já havia sido feito, obtivemos resultados muito bons com uma base de dados mais vasta e variada, possibilitando um treinamento mais genérico também, fora a acurácia que foi bem satisfatória e de igual forma a função de perda.

E por último, usando a base completa a treinamos e rodamos, mas agora com o algoritmo de domain adaptation DANN, os resultados foram completamente satisfatórios, foram rodadas apenas 7 épocas devido ao custo computacional e o tempo levado para isso e nesse teste foi obtido uma acurácia de cerca de 99% e a curva de desenvolvimento continuava a crescer (como mostra a figura 13), já na parte da perda o resultado foi muito motivador também com valores bem baixos (como mostra a figura 14), sendo a média da perda no teste de cerca de 0.15 (como mostra a figura 15).

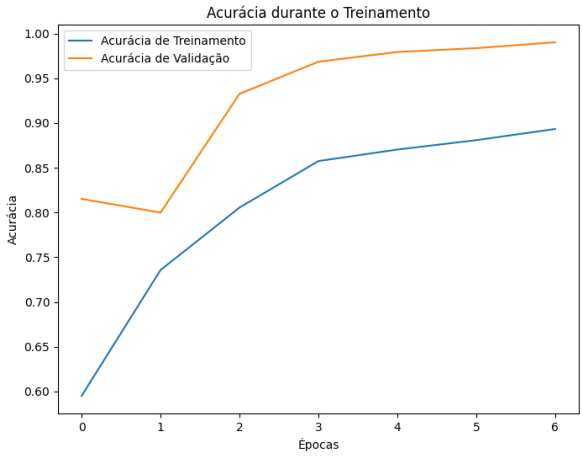

FIG.13. GRÁFICO DE ACURÁCIA DO MODELO DANN TESTADO NA BASE COMPLETA

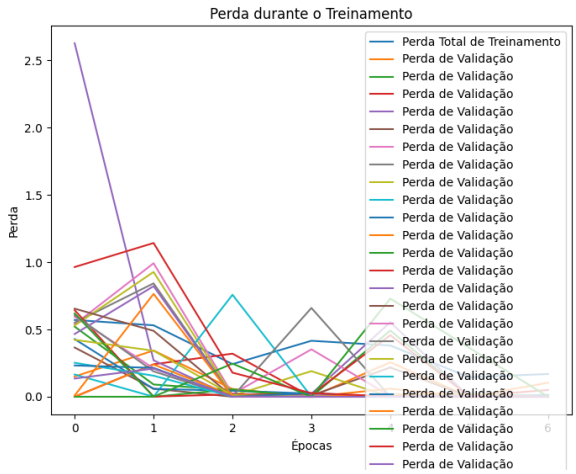

FIG.14. GRÁFICO DE PERDA DO MODELO DANN TESTADO NA BASE COMPLETA

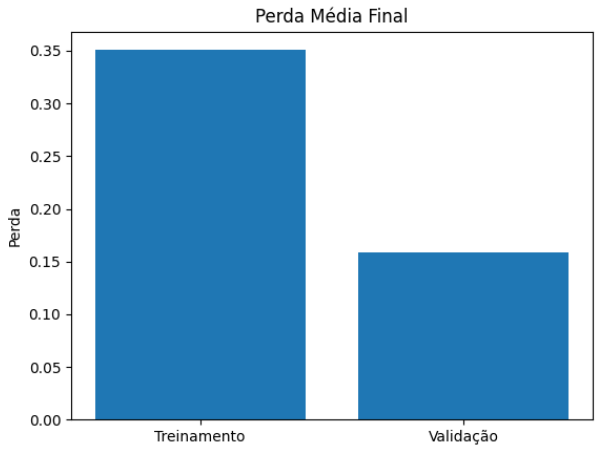

FIG.14. GRÁFICO DE PERDA MÉDIA DO MODELO DANN TESTADO NA BASE COMPLETA

Assim, encerrando com esse experimento foi o que apresentou resultados mais animadores

VII. CONCLUSÕES E TRABALHOS FUTUROS

Após rodar todos esses experimentos das formas mais variadas, é possível concluir que o melhor modelo foi o último de domain adaptation, principalmente por ser uma das características de modelos desse tipo lidar com bases de dados variadas. Só foram rodadas 7 épocas nele, mas as curvas do gráfico de acurácia indicavam ainda um movimento de constante subida, diferente do modelo 3 com a base completa em que a curva já estava em um platô. Além disso, a função de perda teve melhor desempenho também quando comparado ao modelo 3.

As partes 3 e 4 acabaram com os piores resultados, mas já era algo esperado principalmente levando em consideração a primeira base de dados e suas características.

Mas em geral, a junção das duas bases de dados aliada ao algoritmo de domain adaptation gerou um resultado muito satisfatório. Tanto com uma base de dados nova mais completa e consequentemente mais genérica, como com os resultados obtidos após rodar o modelo.

E para trabalhos futuros talvez seja interessante juntar mais bases, dando assim uma maior variedade de possíveis casos, como ocorre no mundo real e gerando assim um modelo mais genérico e treinado para tal situação de identificar se o motorista está ou não com sono, podendo assim prevenir acidentes e , consequentemente, podendo até salvar vidas.

VIII. REFERÊNCIAS BIBLIOGRÁFICAS

[1] *https://imirante.com/noticias/brasil/2023/10/16/dormir-no-volante-esta-entre-as-principais-causas-de-acidentes-nas-rodovias-no-pais*.

[2] https://www.thensf.org/wp-content/uploads/2023/11/NSF-2023-Drowsy-Driving-Survey-Report.pdf

[3] https://www.ibm.com/br-pt/topics/deep-learning

[4] https://ijrpr.com/uploads/V3ISSUE2/ijrpr2617-driver-drowsiness-detection-using-python.pdf

[5] https://www.researchgate.net/publication/356594817_DRIVER_DROWSINESS_DETECTION_USING_PYTHON

[6] https://thesai.org/Downloads/Volume12No7/Paper_94-Real_time_Driver_Drowsiness_Detection_using_Deep_Learning.pdf

[7] https://uwaterloo.ca/systems-design-engineering/current-undergraduate-students/courses/workshop-projects/fourth2002/drowsiness-detection-system

[8] https://paperswithcode.com/task/domain-adaptation